Kernun UTM Handbook

Trusted Network Solutions, a.s.

March 30, 2023

The documentation of Kernun UTM consists of several parts; all of them are available in the electronic form. The complete documentation is installed with the software in the directories `/usr/local/kernun/doc` and `/usr/local/kernun/man`, so it is always available on any Kernun UTM system. The documentation is also contained in the `kernun-doc` directory on the installation medium and is therefore also accessible before the installation. The Kernun UTM documentation is available in the following formats:

Text files Only several short documents that should be read before the installation of Kernun UTM are available as plain text files:

`KERNUN-CHANGES.txt` List of changes between individual versions of Kernun UTM.

`KERNUN-INSTALL.txt` Short installation instructions. This file basically refers to Chapter 2 in the Kernun UTM Handbook.

`KERNUN-RELNOTES.txt` Release notes; various notices concerning the installation, configuration, and use of Kernun UTM.

PDF The Kernun UTM Handbook, that is, this document. The PDF version of the handbook contains also the reference pages except for section 6. This format is suitable for printing and reading as a book, basically from the beginning to the end.

HTML The Kernun UTM Handbook. The HTML version of the handbook contains also all the reference pages. It is available either as a single very long HTML file, or broken into many smaller HTML files. This format is suitable as a reference, with the possibility of hypertext navigation between its parts.

Manual pages The reference part of the documentation is available also in the form of the standard manual pages that can be viewed using the **man**(1) command. The manual pages are categorized into sections, similarly as the system manual pages. Kernun UTM uses the following manual page sections:

Section 1 User commands, mainly various tools for runtime monitoring and generation of statistics.

Section 5 Configuration. Individual sections of the `/usr/local/kernun/conf/kernun.cml` configuration file are documented in this section.

Section 6 For each log message, except for the debugging ones, there is a manual page that describes the conditions, under which the message is logged, and the possible consequences of its appearance in the Kernun UTM log. The manual pages' names are the IDs of the corresponding messages.

Section 7 The manual pages in this section explain general concepts. They cover features that are common to many parts of Kernun UTM, such as proxies.

Section 8 Administrative commands, including application proxies and configuration management tools.

If you are looking for the description of a Kernun UTM feature, you can find its explanation in Section 8 (if it is a separate program), or in Section 7 (if it is a part of a program). If the feature is configurable, its configuration is defined in detail in Section 5. The corresponding manual pages in Section 5 and Section 7 or 8 often have the same name; they are distinguished only by the section number.

This Handbook will help you learn how to administer Kernun UTM. An overview of individual products from the Kernun family is given in Chapter 1. The first steps and the installation instructions are provided in Chapter 2. For the first time, it suffices to read only the sections needed for the initial installation (Section 2.3, Section 2.5.1, and Section 2.5.2). Reading of the remaining parts of the chapter can be postponed until you need to know more about alternative installation methods, upgrades, backups, or disk layout. If you already have a preinstalled and licensed instance of Kernun UTM, you can skip Chapter 2 altogether. Chapter 3 contains an introduction to the graphical and command line administrative interface. Beginners will probably find the GUI (Section 3.1) to be the easiest way of controlling Kernun UTM. If, for any reason, you cannot (or do not want to) use the graphical interface, you find the information about the command line tools in Section 3.2 and Section 3.3. If you know how to connect to a running Kernun UTM system, monitor and control its operation, view logs, and edit the configuration, you may learn principles of the Kernun UTM configuration and find an explanation of the initial configuration generated during the installation in Chapter 4. Chapter 5 deals with configuration of advanced features . At any time, details about features, commands, configuration syntax and semantics, as well as the meaning of log messages can be found in the reference pages, which are contained in the Appendix of this Handbook and available also in the form of manual pages.

Chapter 1

The Kernun family consists of several products that are each useful for a specific set of network security tasks. We will provide a brief introduction to each of them now.

1.1 Kernun UTM

Kernun UTM is a new type of a UTM secure device that contains multiple features, such as firewall, antivirus, antispam, antispyware, content filtering, intrusion detection (IDS or IPS), routing, QoS or VPN, in a single package. It has been designed to protect private data networks and DMZ segments (demilitarized zones, including servers with public services, for example WWW, FTP, mail servers, secure remote VPN connection, etc.). It provides antivirus and antispam protection, as well as an ability to block unsuitable protocols (Skype, ICQ, etc.) and unsuitable Web pages.

Kernun UTM is highly flexible during the process of secure policy implementation. This includes simple rules of status inspection, as well as sophisticated management on the level of application protocols. Thanks to its ability to inspect the contents of each application protocol, this technology is the ideal solution for environments with high security demands.

A typical implementation of the Kernun UTM technology is located on the perimeter of the protected network as a gateway between the Internet and the internal network. All connections to and from the Internet are authorized or prohibited at a central location. Kernun UTM also serves as an antivirus and antispam gateway, and as a server, where VPN connections for clients who work from home or while travelling and of VPN tunnels between branches are terminated. Public service network servers (DMZ) are usually located on another network interface.

1.2 Kernun Clear Web

Kernun Clear Web is a web filter. It controls access of users to the WWW according to a configured policy. The main functions of Kernun Clear Web are: web server categorization, user authentication, definition of web access policy, antivirus protection, traffic monitoring, statistical reports, and web-based graphical user interface.

The interface for administration of Kernun Clear Web is completely different from that of Kernun UTM and it is not covered by this Handbook. For more information about use and

management of a Kernun Clear Web system, see its own documentation.

Chapter 2

In this chapter, we explain how to create and manage a Kernun UTM installation. The system management tasks include installation, upgrade, system backup and restore. An auditing tool can be used to receive notification of discovered bugs and available new software updates. We also provide information about the use of license files and installation of up to three independent Kernun UTM versions on a single computer.

Kernun UTM uses (slightly modified) FreeBSD as its underlying operating system. Although experience with FreeBSD or another operating system based on Unix would certainly be beneficial when performing advanced administrative tasks, it is not required. Kernun UTM provides its own set of powerful tools for installation, configuration, and monitoring of operation.

2.1 Installation Media, Releases, and Builds

Each Kernun UTM release is distributed using the following types of distribution media:

USB flash drive image A bootable disk image, which contains the installation tools and the full installation image.

Full image An installable image of the Kernun UTM system partition. It can be installed either using the installer booted from the installation medium, or from a running Kernun UTM system using the Kernun GUI or the **sysmgr**(8) command line tool. Each full image is uniquely identified by its build number.

Patch image A patch image contains only the differences between two versions of Kernun UTM, and is therefore much smaller than the full image. Patch images are usually created for maintenance updates. Their sole purpose is to optimize the amount of data that needs to be downloaded in order to update a Kernun UTM installation to the current version. The result of installation is the same, no matter whether the full image or a patch image is used; the only difference is in the size of the image. A patch image is identified by its build number and by the build number of its base image.

Kernun UTM releases are identified by version and build numbers. The *version number* denotes the source code version of the Kernun UTM software (the operating system, application proxies, administrative tools, preinstalled third-party software packages, etc.). The format of the version number is either *3.0* for releases (containing new features), or *3.0.1* for patch releases (containing bug corrections and minor improvements). Some bug fixes are implemented using the fast development cycle and are distributed as hotfix releases, numbered e.g. *3.0.1-h3*.

The *build number* identifies the particular build, i.e., a binary image that comprises the core Kernun UTM software, the operating system, and third-party software, such as antivirus scanners, system monitoring tools, or administrative utilities. A build number contains the version number (formatted without the dots and with a fixed number of digits), the date and time when the image was created, and the hardware architecture. Examples: *030000h00.200809241501.i386* or *030001h00.200810170823.amd64*.

2.2 Disk Space Layout

Kernun UTM is able to use one or two disk devices. Each disk device is either a physical disk, or a logical disk provided by a hardware RAID. The disk space is divided into three *system partitions*, one *data partition*, and swap space. In single disk configurations, all four partitions and the swap space are located on the single disk. In configuration with two disks, the system partitions are on one disk, whereas the data partition and the swap space on the other.

Each system partition may contain a complete Kernun UTM installation including the operating system, application proxies, administrative tools, and additional software. The data partition contains logs, statistics, installation images, and backups. The contents of the data partition are shared by all Kernun UTM installations in the system partitions.

The use of three system partitions minimizes downtimes during reinstallations and upgrades. While the system started from one system partition is fully operational, it is possible to install another version in the second partition. Then the new version can be started by a simple reboot. It is always possible to revert to the old version if anything goes wrong with the new one. The next upgrade will be installed in the first partition while running the system from the second one. In this way, two system partitions can be alternated for subsequent upgrades. The third system partition can be used in a similar fashion, so that two previous versions are always available, or for an alternative installation, e.g. when testing a completely new configuration.

When a system partition is booted, it becomes the root file system. The other system partitions can be mounted to the directories `/1`, `/2`, and `/3`. There are lines in `/etc/fstab` prepared for this, but the partitions are not mounted automatically. The data partition is always mounted as `/data` automatically. It contains the following directories:

`/data/backup` System backups are stored here. They can be used for restoration or copied to another medium.

`/data/dist` This is where Kernun UTM installation images are kept. During each installation, the installed image is stored here for future reuse.

`/data/log` This directory contains log files. The log directory `/var/log` from all system partitions is symlinked here.

/data/rrd This directory contains database files used to store system data for system performance monitoring, as well as graphs generated from this data.

/data/statistics Reports with detailed statistics of proxy operation are stored in this directory.

The standard disk space layout is created during the first installation of Kernun UTM on a new computer. It can be re-created or modified using the installer booted from the installation medium, but such action deletes all data on the system and data disks.

> **Warning**
>
> It is strongly recommended not to modify the standard disk layout, as many parts of Kernun UTM depend on it. You may add additional file systems and directories, but do not delete or move any file system or directory created by the Kernun UTM installer.

2.3 Licensing

Kernun UTM requires a valid license file to operate properly. Without a license file, the software can be installed, the operating system runs allowing both local and remote administrator access, but no licensed component may be started. The licensed components include all application-level network proxies and some additional modules (for example, antivirus, antispam, and Web filter).

The license file is a cryptographically signed text file. It contains the following information:

- The customer identification
- An optional identifier used to distinguish different licenses of the same customer
- A unique serial number
- The license size (the permitted number of protected network devices)
- A computer identifier, if the license is valid exclusively with particular hardware.
- The expiration date, if the license is valid for a limited time.
- *(Only Kernun 3.3 and newer)* The expiration date of upgrade subscription. Before this date, new features (components) added to Kernun will be automatically licensed if covered by the subscription. After this date, existing features will continue to work (until the optional license expiration date), but new features will not be licensed.
- *(Only Kernun 3.2 and older)* The release version number, if the license is valid for a single Kernun release (e.g., 3.1) only. The license can be used on all patch releases and hotfixes of the licensed release (e.g, 3.1.2 or 3.1.1-h5), but not on other releases (e.g., 3.2).
- The list of licensed components.

- *(Only Kernun 3.3 and newer)* The list of licensed groups of components. Licenses are usually issued for groups of components. For example, there are groups corresponding to various Kernun products, such as Kernun Net Access or Kernun Kernun Mail Access. The use of component groups makes it possible to add new licensed components to users with active subscription without the need for a new license file.
- *(Only Kernun 3.3 and newer)* Various parameters of the licensed components.
- A cryptographic signature used to verify the integrity of the license.

Note

- License files from Kernun 3.0 are not valid for 3.1 and newer releases.
- Licenses from Kernun 3.1 and 3.2 are recognized by Kernun 3.3 and newer.

The license file must be installed as `/usr/local/kernun/license.dat`. The license file is stored in the system partition and must therefore be reinstalled after each installation or upgrade. The license file can be copied to Kernun UTM either from the command line using SCP, or at the License tab of the GUI System Manager.

The set of configurable components changes depending on the type of the Kernun product and the set of licensed components. For example, if the HTTP proxy is not licensed, it should not be configured. A single configuration file may comprise configurations of many Kernun systems with different products. In each configuration section related to a single system (section `system`), the product can and should be specified using the `product` item. The product specification consists of the Kernun software type, the list of licensed components, the list of licensed component groups, and the upgrade subscription expiration value. The product specification should be filled according to the contents of the license file present in the configured system. When the configuration is verified, a check is made that only components usable in the selected products are configured. When the configuration is applied, it is checked that the product specified in the configuration complies with the product installed in the target Kernun system. At the time of writing of this text, there are two product types available:

- `kernun` — all Kernun products;
- `unspecified` — the product type is not specified and will not be checked when applying the configuration.

The recognized names of licensed components and component groups are the same as in the license files. Components:

- `product-kernun`, `product-kernun-net-access`, `product-kernun-mail-access`, `product-kernun-vpn-access`, `product-kernun-office-access`, `product-kernun-web-access`, `product-kernun-secure-box`, `product-kernun-secure-box-retail` — Kernun product names;

- `dns-proxy`, `ftp-proxy`, `gk-proxy`, `h323-proxy`, `http-proxy`, `imap4-proxy`, `pop3-proxy`, `sip-proxy`, `smtp-proxy`, `sqlnet-proxy`, `tcp-proxy`, `udp-proxy` — individual proxies;
- `icap-server` — server for the ICAP protocol;
- `mod-antivirus` — module for communication with an antivirus in proxies;
- `mod-antispam` — module for spam checking in mail proxies;
- `mod-pwf` — module for communication with an external Web filter in the HTTP proxy;
- `http-cookie` — support for special handling of security-related HTTP cookies, for example, various session ID cookies;
- `mod-match`, `mod-match-replace` — module for matching and replacement of HTML form data.

Component groups:

- `kernun-net-access`, `kernun-mail-access`, `kernun-vpn-access`, `kernun-office-access`, `kernun-web-access`, `kernun-secure-box`, `kernun-secure-box-retail` — individual Kernun products;
- `modules-data-scanning` — modules for security scanning of data, such as the antivirus module;
- `modules-secure-box` — special modules for the Kernun Secure Box products;
- `modules-web-filter` — modules providing URL-based categorization and filtration of WWW servers.

When the initial configuration file is created (see Section 2.5.2), the product type is detected, the currently installed license file is examined, and the `system.product` item is set appropriately. Therefore, it is recommended to install the license file during the installation of the system, before the initial configuration script is executed. The license file can be installed by the standalone installer, as described in Section 2.5.1. If the license file is not installed during the generation of the initial configuration or if a new system is being added to an already existing configuration, the `product` item must be set manually.

If you set the `product` item manually, select the correct product type and enter the list of licensed components, the list of licensed component groups, and the upgrade subscription expiration date according to your license file[1]. It is also possible to include the `samples/include/products.cml` file in the main configuration file. This file contains definitions of variables that can be used instead of the `system.product` item.

[1] Collect values from lines starting with `component:`, `group:`, and `upgrade:` in the license file.

Note

Some products may have optional components. Their respective variables in `samples/include/products.cml` have a parameter containing the list of licensed optional components. For example, Kernun Net Access with the optional antivirus and antispam modules will be specified as:

```
$PRODUCT-KERNUN-NET-ACCESS { mod-antivirus, mod-antispam };
```

Even if no optional components are licensed, the empty list must be written explicitly as the variable's parameter:

```
$PRODUCT-KERNUN-NET-ACCESS { };
```

Variables for products without optional components do not have a parameter and are therefore written without the braces:

```
$PRODUCT-KERNUN-MAIL-ACCESS;
```

2.4 Boot Manager

The Kernun UTM boot manager is located on the system disk. It is installed during the initialization of the system performed by the standalone installer. Depending on the partitioning scheme, there is either ZFS or LEGACY (UFS) boot manager.

The Kernun GUI or the command line **bootmgr**(8) utility can be used to change partition labels, enable and disable booting from individual partitions, and set whether the default boot partition is fixed, or is always changed to the last booted partition.

2.4.1 ZFS boot manager

There is a default partition selected for boot, which boots automatically. It is also possible to select another partition during the boot sequence. The boot manager displays the menu as shown in Figure 2.1. Press option **7** to select the boot partition. Another menu is displayed (see Figure 2.2). In this menu, press the number that corresponds to the system to be booted. For example, press option **5<Enter>** to boot the Kernun 3.10.6.h3.

2.4.2 LEGACY (UFS) boot manager

The boot manager displays labels of up to three system partitions and allows selection of the partition to boot from by pressing **F1**, **F2**, or **F3**.

```
F1   Kernun 3.0 2008/10/01 07:36 (030000h00.200809241501.i386)
F2   Kernun 3.0 2008/10/18 05:21 (030000h00.200810170852.i386)
F3   Kernun 3.0.1 2008/11/15 07:22 (030001h00.200811142135.i386)
```

Figure 2.1: ZFS boot manager screen 1

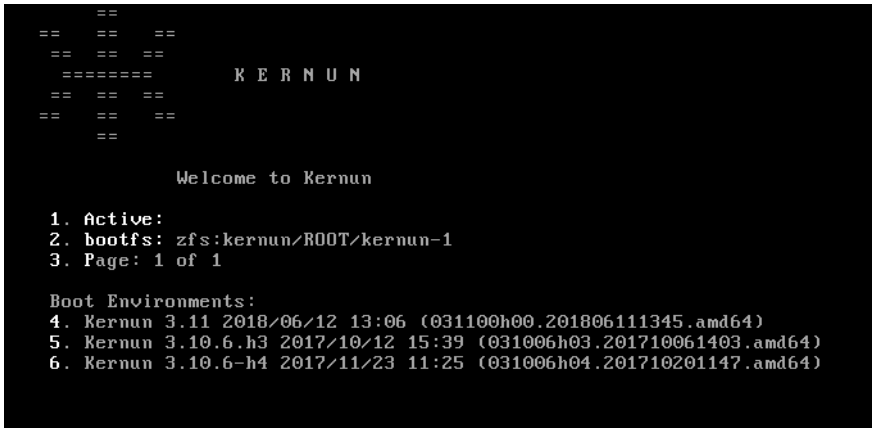

Figure 2.2: ZFS boot manager screen 1

```
Default: F2
```

If no option is selected, the default one is chosen automatically after a timeout.

2.4.3 Security notice

Anybody with physical access to the Kernun UTM console may select a system partition to boot from, boot a different kernel or kernel modules, or boot to the single user mode and access the system without a password. If the system console is not physically secure, the following actions can be done to protect the system against unauthorized access:

1. Disable boot device selection in the BIOS (for example, by setting a BIOS password).
2. Enable only the desired system partition in the boot manager (using **bootmgr**(8)).
3. Add line "`-n`" to `/boot.config`. This prevents interrupting the boot process in the stages one and two.

   ```
   # printf --
           '-n\n' >
           /boot.config
   ```

4. Protect the loader with a password by adding a password line to `/boot/loader.conf`. Make the file readable only by root.

   ```
   #
           echo
           'password="SECRET"' >>
           /boot/loader.conf # chmod
           go-rw
           /boot/loader.conf
   ```

5. Force verification of the root password as a condition for entering the single user mode. Locate the line beginning with "`console`" in `/etc/ttys` and change its last word to "`insecure`".

2.5 Installation

Kernun UTM can be installed using either the standalone installer booted from the installation medium, or command line or GUI system management tools. The first installation on a new computer must be done using the standalone installer, which does not require an already installed Kernun UTM with initialized system and data disks and is able to initialize the standard disk layout, as described in Section 2.2. Once there is at least one working Kernun UTM instance on the computer, further installations can by done from it using either the GUI, or the **sysmgr** command line tool. The standalone installer is able to install in any system partition. The GUI and command line installations cannot be performed in the system partition that contains the currently running Kernun UTM instance.

Note

Regardless of the installation method, the newly installed system partition is, by default, enabled in the boot manager and made the default selection for the next boot. The boot manager can be reconfigured using the GUI or the command line utility **bootmgr**(8).

2.5.1 Standalone Installer

The standalone installer is normally used only for the first installation on a new computer, after replacing a disk, or if disk repartitioning is needed. In other situations, installation using the GUI (Section 2.5.3) or the command line (Section 2.5.4) is more comfortable.

Note

Since version 3.11.7-h3, the installer boots using only UEFI (with disabled Secure Boot), while previous versions used only legacy BIOS. If your hardware does not support UEFI, consider installing an older version and upgrading it to the latest version.

To start the standalone installer, you need the Kernun UTM installation medium[2]. Boot from the USB flash drive and following the boot loader and kernel messages, you will see the installer menu.

```
*** KERNUN INSTALLATION ***

Build 030000h00.200809241501.i386

1. Install Kernun
2. Check for existing Kernun installations
3. Restore backup
4. Start rescue shell
5. Mount Kernun file systems
6. Resize installer's in-memory temporary file system (current size 32m)
7. Halt
8. Power down
9. Reboot
0. Install license

Select action:
```

Press **1<Enter>**. If the disk partitioning for Kernun UTM has already been done, the device names of the system and data disks are displayed and the installer asks whether you want repartitioning.

[2] You can use `dd` on Linux / BSD or `https://github.com/openSUSE/kiwi/downloads` on Windows to copy the USB flash drive image to the device.

```
Detected Kernun system disk ad0
Detected Kernun data disk ad0
Repartition disks (y/n)?
```

Reply **n** to skip disk partitioning. If you reply **y** or if the disk partitioning has not been done yet, the system suggests the default installation:

```
Default installation parameters:
System disk: ada0 (131072 MB)
SSD disk: ada1 (524288 MB)
Swap size: 8192 MB

Install with default parameters? (y/n)
```

Reply **y<Enter>** to finish the installation with default parameters. If you reply **n**, the installation parameters are asked. Select the system disk and partitioning schema:

```
Detected disk devices:

ada0 131072 MB
ada1 524288 MB
  media RPM              non-rotating
Kernun system disk(ada0 ada1) [ada0]: <Enter>
Use ZFS (y/n) y<Enter>
```

Always select a disk that the BIOS (or UEFI) will be able to boot from as the system disk[3]. If there is only one disk device, the selection of devices will be skipped and the single device will be used. The ZFS partitioning schema is prefered over the legacy (UFS) schema. Note that ZFS boots using UEFI while UFS boots using legacy BIOS.

> **Tip**
>
> When the installer asks a question, it offers a default value in brackets. Press **<Enter>** to select the default value.

The installer then asks for the swap size. Reasonable default value is provided. It can be changed if the default value does not meet the expectations.

```
System disk size is 131072 MB
Memory size is 4096 MB
Swap partition size in MB [8192]: <Enter>

Disk ada0 [131072 MB] will contain 3 GPT partitions:
  ada0p1 with freebsd-boot
  ada0p2 with freebsd-swap [8192 MB]
```

[3] It is usually the first disk: `da0` (SCSI), `ad0` (PATA), `ad10` (SATA).

```
  ada0p3 with freebsd-zfs
Use these values (y/n)? y<Enter>
Disk partitioning will delete contents of selected disks,
continue (y/n)? y
```

If you want to cancel the installation process, answer **n** to the last question. It will return to the main menu without changing the disk contents.

Warning

Answering **y** to the "continue" question will initialize the selected system and data disks with the standard disk layout for Kernun UTM. Any existing contents of the disks will be lost.

If there is a SSD disk, it is offered to be used. Select the device name to use the SSD disk, or select **NO** not to use it.

```
Detected disk devices:
ada1 524288 MB
  media RPM           non-rotating
Create SSD disk (NO ada1) [NO]: ada1<Enter>
Creating KBI disk on ada1
```

Messages concerning creation of disk partitions and file systems will then be displayed, followed by:

```
Current Kernun installations:
Boot manager on ZFS pool 'kernun'
F1: Unused
F2: Unused
F3: Unused
type=Kernun ZFS boot manager ver. 1.0
current_booted=NONE
bootable=
update=1
default_selection=NONE
Select partition for installation (1 2 3) [1]: <Enter>
Overwrite partition /dev/ad0s1 by new Kernun installation (y/n)? y
```

These lines show the configuration of the Kernun UTM boot manager, see **bootmgr**(8). The first installation will be usually performed in the first system partition, so just press **<Enter>**. Finally, you are asked to confirm whether you want to overwrite the selected system partition.

The installer creates any missing standard directories in the data partition, creates a new empty file system in the selected system partition, and displays a list of the installation images (identified by build numbers) available on the medium and in the `/data/dist` directory. If there is more than one image, one can be selected, with the newest image as the default. If the image from the

medium is selected, it is first copied to `/data/dist`. The selected image is then unpacked to the system partition. The `/etc/fstab` file in the newly installed partition is adjusted according to the system partition number. The build number of the installed Kernun UTM is stored in the `/kernun-version` file in the system partition. The content of the newly installed Kernun UTM instance is stored in `/kernun-installed.fsdb.bz2`. This file is used by the backup tools in order to decide which files have changed since the installation and therefore need to be backed up. After the installation is finished, the installer waits for **<Enter>** and then returns to the main menu.

```
...
Available installation images:
     1  030000h00.200809241501.i386
Copying installation image to /data/dist
Clearing system partition 1
...
Installing kernun-030000h00.200809241501.i386.txz to system partition 1
Unpacking image
Removing file system content databases for installed images
Creating /etc/fstab
Writing build number into /kernun-version
Creating file system content database
Installation successfully finished

Press Enter for return to menu...
```

Optionally, if you have a license file for your newly installed system available, you can install it now. This ensures that the initial configuration script will set the `system.product` configuration item correctly after reboot. It will also ask whether the licensed proxies should be enabled in the initial configuration. The license installation is done in several steps:

1. Prepare a USB disk with a UFS or FAT file system.

2. Copy the license file `license.dat` to the root directory of the USB disk. Alternatively, if you have some other license files (for example, for the antivirus engine), you can pack them all[4] in the `license.tar` file in the **tar** format with all paths relative to the Kernun system root directory.

3. Do not connect the USB disk yet and select **0** from the installer main menu.

4. When prompted, connect the USB disk. The license files present will be installed.

Select **9** from the main menu to have the newly installed Kernun UTM booted. You can then perform its initial configuration, as described in the following section.

[4] including `usr/local/kernun/license.dat`

Note

The /data/dist directory may contain full and patch installation images. A full image can be always installed. A patch image contains only the differences from a base image. Hence the base image must be available in order to install the patch image. The base image may itself be a patch image, and its base image is then required as well. Generally, each patch image requires a continuous sequence of base images starting with a full image followed by zero or more patch images.

2.5.2 Initial Configuration

When a newly installed Kernun UTM system is booted for the first time, an interactive initial configuration script (/etc/rc/kernun-config) is executed early in the boot process[5]. It prompts the administrator for various basic system parameters, creates and applies the Kernun UTM configuration file, and finishes the boot procedure with the new configuration. The initial configuration can be modified later using the standard Kernun GUI or command line configuration tools.

First, the time zone needs to be set. We recommend to use UTC for the CMOS clock—select **Yes** by pressing **<Tab><Enter>** in the first dialog. Even if the CMOS clock is currently set to the local time, it is better to select UTC here and adjust the time later using the **date**(1) command or by configuring NTP, see section ntp in system(5). After selecting the CMOS clock mode, the time zone menu is displayed. Choose the time zone suitable for your location. Then set the administrator password (user root).

After that, a new SSH host key is generated. It is used to authenticate the system to a remote access client[6] (GUI or command line SSH). You should write down the reported key fingerprint and compare it with the fingerprint reported by SSH or the GUI when making the first remote connection to the system. The SSH host keys should be the same for all Kernun UTM installations on the same computer. Therefore, if an SSH host key exists during the installation, it is copied to the newly installed system partition and the generation of a new key is skipped during the initial configuration. The GUI and command line installers look for an SSH host key in the current system partition. The standalone installer takes an SSH host key from the first system partition that contains one and is different from the partition, in which the installation is taking place.

Answer **n** to the following question (or just press **<Enter>**) if you want to input the basic configuration parameters and generate the initial Kernun UTM configuration file.

```
*****************************************************************
Fingerprint of the SSH host DSA key. Compare this value with the value
reported by SSH client or Kernun GUI when connecting in order to check
that you are connecting to this system.
```

[5] More precisely speaking, the initial configuration script is executed during any system boot if there is no Kernun UTM configuration file /usr/local/kernun/conf/kernun.cml and none of the files /etc/rc.conf and /etc/rc.conf.local contain the line kernun_config_enable=NO.

[6] The host key is used by the SSH client (or GUI) to ensure that it is communicating with the intended server. It is different from the client's key, which is used to authenticate the client to the server.

```
1024 71:0a:ec:8d:dd:9e:e7:2d:2b:91:79:0e:1a:ca:89:2b
        /etc/ssh/ssh_host_dsa_key.pub
**********************************************************************

*** KERNUN INITIAL SYSTEM CONFIGURATION ***

Skip Kernun configuration (y/n)? [n] <Enter>
```

Two network interfaces are configured in the default configuration: *internal*, intended to be connected to the protected network, and *external*, which is typically connected to the Internet. The configuration script asks for the names, IP addresses, and network masks of these interfaces. Then, the DNS server and default router addresses need to be specified. The initial configuration will allow the administrator SSH access from the internal network (using the GUI or a command line SSH client). If you want to allow some application protocols to pass from clients in the internal network to servers in the external network, you can enable the respective proxies. The configuration of the proxies will contain the default values of various parameters, which will be sufficient for the simplest use. More complicated configuration requirements can be implemented later by editing the generated initial configuration file using the GUI or command line configuration tools (modifying proxy configuration, adding new proxies, etc.). An example of the initial configuration setup is given and explained below.

Caution

In many environments, an initial configuration with enabled proxies may violate a security policy. Therefore, it is recommended not to enable any proxy in the initial configuration unless you are sure that you really need it.

```
Hostname without domain []: fw ❶
Domain []: example.com
Show only Ethernet interfaces (y/n)? [y] ❷

By repeating the following test with connected and
disconnected network cables, you can determine interface
names of physical network cards.

*** Media state of network interfaces ***
ed0:    media: Ethernet autoselect (100baseTX <full-duplex>)
ed1:    media: Ethernet autoselect (100baseTX <full-duplex>)
Show again (y/n)? [y]
*** Media state of network interfaces ***
ed0:    media: Ethernet autoselect (none) ❸
ed1:    media: Ethernet autoselect (100baseTX <full-duplex>)
```

```
Show again (y/n)? [y]
*** Media state of network interfaces ***
ed0:    media: Ethernet autoselect (100baseTX <full-duplex>)
ed1:    media: Ethernet autoselect (100baseTX <full-duplex>)
Show again (y/n)? [y] n
Internal interface name (ed0 ed1) []: ed0 ❹
Internal IP address []: 192.168.10.1
Internal interface netmask [24]:
External interface name (ed0 ed1) []: ed1 ❺
External IP address []: 192.168.11.2
External interface netmask [24]:
DNS server IP address []: 10.1.1.1 ❻
Default router IP address []: 192.168.1.1 ❼
Postmaster e-mail [postmaster@example.com]: ❽
Enable some proxies (y/n)? y ❾
Enable DNS proxy (y/n)? [n] y
Enable FTP proxy (y/n)? [n]
Enable HTTP proxy (y/n)? [n]
Enable HTTPS proxy (y/n)? [n]
Enable POP3 proxy (y/n)? [n]
Enable IMAP4 proxy (y/n)? [n]
Enable SMTP proxy (y/n)? [n]
Enable SSH proxy (y/n)? [n] y
Hostname:           fw ❿
Domain:             example.com
Internal interface: ed0
Internal IP:        192.168.10.1
Internal netmask:   24
External interface: ed1
External IP:        192.168.11.2
External netmask:   24
Name server:        10.1.1.1
Default router:     192.168.11.1
Postmaster e-mail:  postmaster@example.com
Enabled proxies:    DNS SSH

Use these values (y/n)? y ①
```

The configuration begins ❶ with setting the host name and the domain name. Then, the internal and external interfaces are selected. First, the available network interfaces are listed. You can choose ❷ whether you want to show all interfaces, or just Ethernet interfaces. The interfaces are repeatedly listed with their media states. This can be useful if you are not sure about the names of physical interfaces. You can unplug network cables one by one and observe, which interface

changes its state. In the example ❸, the cable was unplugged from the network interface `ed0`. The internal ❹ and external ❺ interface names, IP addresses, and network masks are defined. The DNS server IP address ❻ is used by Kernun UTM for domain name resolution. The default router ❼ is typically a router in the external network. The postmaster e-mail address ❽ is used by the SMTP proxy to forward mail sent to the postmaster.

You can also enable some proxies ❾ for access from the internal to the external network. Questions about individual proxies are asked only if you reply **y** to the initial "enable some proxies" query. Otherwise, all proxies are disabled without further questions. The generated initial configuration file will contain configuration of the disabled proxies as well, with their configuration sections marked as hidden. A proxy can be easily enabled later by unhiding its configuration using the GUI or the command line configuration interface. Only licensed proxies are offered for enabling.

Finally, all values defined during the configuration setup are listed ❿. If you are satisfied, reply **y** ① and the initial configuration file will be generated and applied. If you reply **n**, the whole configuration setup will be repeated with the previously specified values as defaults.

After defining values for the initial configuration, the SSH key for remote administrator access is generated. You must enter a passphrase used to encrypt the key. The same passphrase is also used for the initial download of the key from Kernun UTM.

```
The configuration script will now generate the root's SSH key.
The passphrase for the key will be also used as the password
for initial key download from Kernun GUI.
Enter SSH key passphrase:
Repeat SSH key passphrase:
Generating public/private dsa key pair.
Your identification has been saved in /home/keygen/id_dsa.
Your public key has been saved in /home/keygen/id_dsa.pub.
The key fingerprint is:
33:27:5a:63:53:b1:ba:47:bf:e8:58:4a:d0:f6:d4:d4 root@fw.example.com
```

The SSH key generation is the last step in the initial configuration process. After that, the normal operation of the newly installed Kernun UTM begins.

The SSH (private) key needs to be downloaded to the administrator's local computer and subsequently copied to any system used by the administrator to access Kernun UTM. The administrator's computer must be in a network routed via the Kernun UTM internal interface, e.g., `192.168.10.0/24` in our configuration example. There is a special user account `keygen` dedicated to SSH key download. The GUI is able to download the key automatically, you only need to select Initialize new firewall in the Connect to Server dialog. See also Section 3.1.1 for details. For command line SSH access, you can either use the key downloaded by the GUI, or download the key manually:

1. Use SCP to copy the private OpenSSH key (`id_dsa`), the public OpenSSH key (`id_dsa.pub`), and the Putty key (`key.ppk`).

   ```
   $ scp keygen@192.168.10.1:* .
   ```

```
keygen@192.168.10.1's password:
id_dsa                        100%  736     0.7KB/s   00:00
id_dsa.pub                    100%  609     0.6KB/s   00:00
key.ppk                       100%  807     0.8KB/s   00:00
$
```

2. Log in to Kernun UTM as user `root` using the newly obtained key.

```
$ ssh -i id_dsa root@192.168.10.1
Enter passphrase for key 'id_dsa':
...
[root@fw ~]#
```

3. Delete the key files in the home directory of user `keygen`.

```
[root@fw ~]# rm ~keygen/*
```

4. Disable the `keygen` account.

```
[root@fw ~]# pw lock keygen
```

5. Log out from Kernun UTM.

```
[root@fw ~]# logout
Connection to 192.168.10.1 closed.
$
```

The steps after the first one are not strictly necessary, but they are recommended for security reasons. Although the secret SSH keys are protected by a passphrase, they should be kept in a secure store that can be accessed only by authorized administrators. If the key is downloaded by the GUI, the key files on Kernun UTM as well as the `keygen` account are automatically removed when the GUI connects to Kernun UTM with the downloaded key for the first time.

2.5.3 Installation from the GUI

In this section, we assume that the reader has at least the basic knowledge of the Kernun GUI. An introduction to the Kernun GUI can be found in Section 3.1 of this manual. The installation and its related tasks are controlled by the Kernun GUI System Manager, which is accessible using the ❶ button in the main window toolbar, as shown in Figure 2.3.

Figure 2.3: The System Manager icon in the toolbar

The installation is done from the Installation images tab in the System Manager window, see Figure 2.4. It displays a list of available installation images (stored on Kernun UTM in

/kernun/dist). An image is marked as installable if it is either a full image, or a patch image with an available base image. The version number, build date, and build number are listed for each image. Installation images can be copied from the administrator's local machine, where the GUI runs, to Kernun UTM by clicking the Upload button. The Download button can be used to copy in the opposite direction. It is also possible to delete a selected image (Remove) or all images older than the selected one[7] (Remove older).

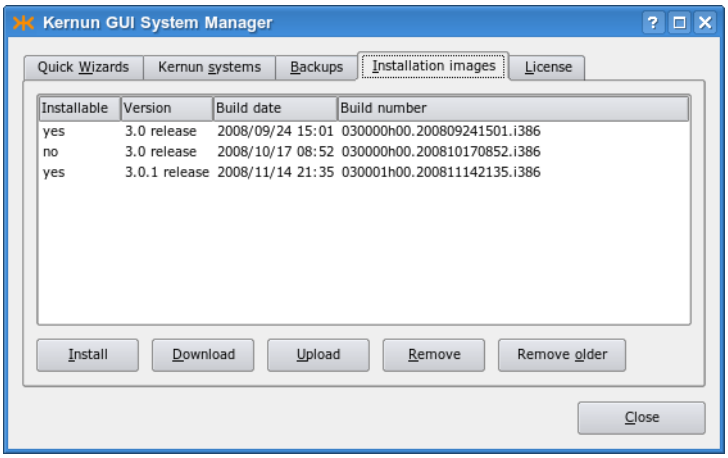

Figure 2.4: Installation images in the System Manager

Tip

Each installed image is is copied to /data/dist. As the images may consume a lot of disk space on a regularly updated Kernun UTM, it is recommended to delete old images regularly or when you need more space on the data disk. An easy way to do this is to select one of the newest images and click *Remove older*. It is usually sufficient to retain only the one or two most recent images.

To initiate the installation of the selected image, click the Install button. In the example, we will install the newest (last) installation image from the list. The installation of Kernun UTM can be alternatively initiated using the Install firewall button on the Quick Wizards page. A wizard window (see Figure 2.5) appears and prompts you to select the target system partition. It displays the number and label of the system partition that contains the currently running system. This partition cannot be overwritten by the installation. One of the other two system partitions, which are also listed with their labels, needs to be chosen. If you started the wizard from the Quick Wizards page, you are then supposed to choose the desired installation image. Finally, the recapitulation of the selected values is displayed. Click the Finish button to launch the installation process (it deletes all the existing content of the selected partition).

[7] An image is considered older if it has a lower version number or an earlier build date.

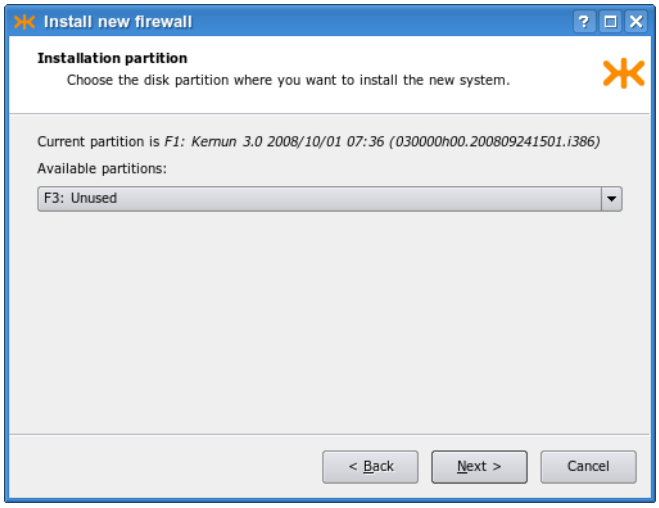

Figure 2.5: Selection of the installation target

Caution

When performing an installation, make sure that you have selected the correct system partition, in order to avoid inadvertently overwriting a system partition that you want to retain.

The installation process takes several minutes; it can be aborted using a button in the progress dialog displayed in the meanwhile. The newly installed system partition is made bootable, but the default boot partition is not changed. The reason is that the new Kernun UTM instance is not configured and until its initial configuration is performed from the console, it will be inaccessible via the network. The boot manager configuration after the finished installation can be viewed in the System Manager's Kernun systems tab, as shown in Figure 2.6. It is possible to change the partition label (using the Change Title button) or make the new system partition the default boot partition (the Set Default button).

If the installation process terminates because of an error, the output of the failed command is displayed. The example in Figure 2.7 shows an error message caused by a corrupted installation image file.

2.5.4 Installation from the Command Line

The command line installation functionality is provided by the **sysmgr**(8) and **bootmgr**(8) utilities. An installation image that is to be installed must be stored in the `/data/dist` directory, along with the corresponding base image(s), if it is a patch image. The existing images can be listed using the following command:

```
[root@fw ~]# sysmgr images
*  030000h00.200809241501.i386
   030000h00.200810170852.i386 ❶
*  030001h00.200811142135.i386
```

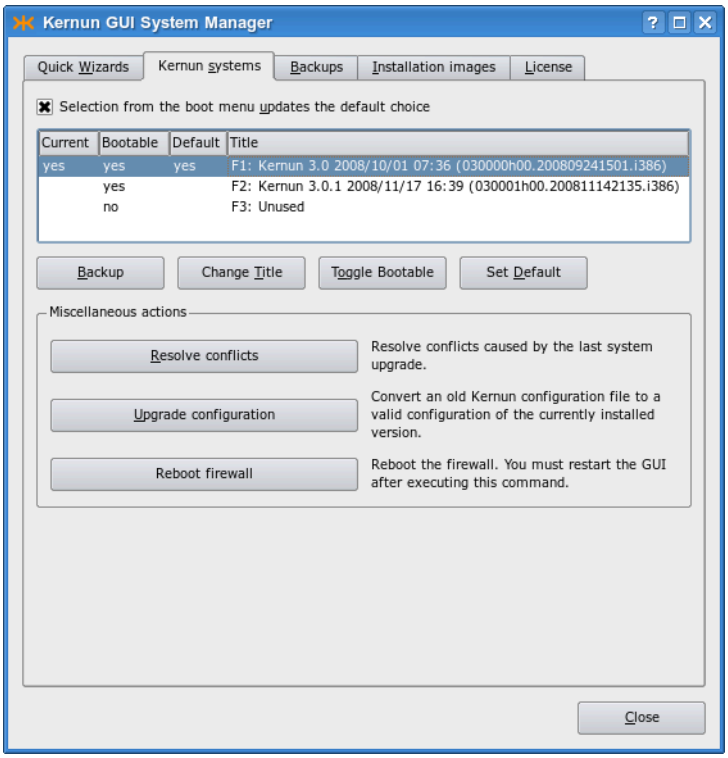

Figure 2.6: The system partitions after the installation

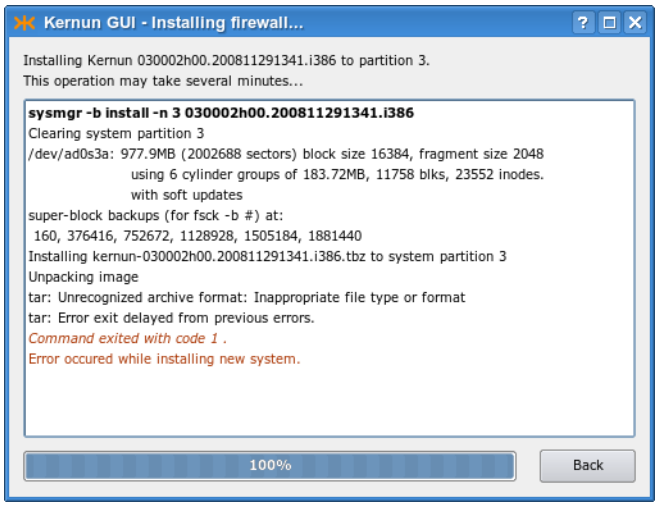

Figure 2.7: An error during the installation

The installable images are marked with an asterisk. The image ❶ is a patch image that cannot be installed, because its base image is missing. Information about the currently installed instances of Kernun UTM can be obtained using the **bootmgr** command or from the `/kernun-version` file. In order to get access to this file in other system partitions, the file systems in those partitions need to be mounted first.

```
[root@fw ~]# bootmgr
Boot manager on /dev/ad0
F1: Kernun 3.0 2008/10/01 07:36 (030000h00.200809241501.i386) ❶
F2: Unused
F3: Unused
type=Kernun 1024 B boot manager (74 character labels)
current_booted=1 ❷
bootable=1
update=yes
default_selection=F1
[root@fw ~]# cat /kernun-version
030000h00.200809241501.i386
[root@fw ~]# mount /2
[root@fw ~]# cat /2/kernun-version
030000h00.200810170852.i386 ❸
[root@fw ~]# mount /3
mount: /dev/ad0s3a on /3: incorrect super block ❹
```

The **bootmgr** command displays labels of the system partitions ❶ and the number of the system partition that contains the currently running system ❷. The second system partition in the example contains another Kernun UTM version ❸, even though it was manually relabeled as "Unused". The third system partition is really unused; it does not even contain a file system ❹.

We will install a new Kernun UTM version in the second system partition. We choose the newest version available according to the **sysmgr images** report. Unlike the standalone installer described in Section 2.5.1, the command line installer asks no questions. The image build number and the target system partition number are given on the command line and the installation starts immediately. The standard partition label, containing the Kernun UTM version, date of installation, and build number, is set for the newly installed partition. The initial configuration process (see Section 2.5.2) is started after booting from the newly installed system partition.

```
[root@fw ~]# sysmgr install 2 030001h00.200811142135.i386
Clearing system partition 2
...
Installing kernun-030001h00.200811142135.i386.txz to system partition 2
Unpacking image
Installing SSH host keys
Removing file system content databases for installed images
Creating /etc/fstab
```

```
Writing build number into /kernun-version
Creating file system content database
Installation successfully finished
[root@fw ~]# bootmgr
Boot manager on /dev/ad0
F1: Kernun 3.0 2008/10/01 07:36 (030000h00.200809241501.i386)
F2: Kernun 3.0.1 2008/11/17 16:39 (030001h00.200811142135.i386)
F3: Unused
type=Kernun 1024 B boot manager (74 character labels)
current_booted=1
bootable=1 2
update=yes
default_selection=F2
```

Caution

Be careful when running **sysmgr install**. Especially, make sure that you specify the correct system partition number. Otherwise, you might inadvertently overwrite a system partition that you would like to retain.

Caution

The newly installed system partition is made the default choice for the next boot. As it is not configured, it will be inaccessible via the network after the reboot and its initial configuration will need to be performed from the console. If you want to keep the current default boot partition, so that you retain a fully working system after the reboot, use the `-n` parameter of the **sysmgr** command:

```
[root@fw ~]# sysmgr install -n 2 030001h00.200811142135.i386
```

2.5.5 Enabling Serial Console Output

Typically, the interaction with Kernun UTM is performed using a VGA console. However, in some cases it is neccessary to switch the system output to a serial console, which is also available on most of Kernun devices. In order to redirect the output to a serial console, plug-in the console to a serial port, restart Kernun UTM, and follow these steps:

- Press <Space> repeatedly until the booting process prompts with `boot:`
- When the prompt `boot:` appears, type "**-h**" and confirm by pressing **<Enter>**. The output of booting proccess should now be redirected to the serial console and continue.

```
>> FreeBSD/i386 BOOT
Default: 0:ad(0,a)/boot/loader
boot: -h
```

2.6 Backup and Restoring

Kernun UTM provides both GUI and command line tools used to back up system partitions and restore data from backups created in this way. They can be used to back up not only the current system partition, but any of the three system partitions. A backup file does not contain the complete contents of a system partition, but only the changes made since its installation. The size of the backup file therefore depends on the amount of changes that have been made in the system partition since its last installation. After an installation, the content of a system partition is stored in the `/kernun-installed.fsdb.bz2` file. When doing backup, this file is compared with the current content of the system partition. Added and modified files are stored in the backup file, along with information about deleted files and files with changed metadata attributes[8].

The backup and restore operations process only a subset of files contained in a system partition, mainly Kernun configuration files. The list of files included in a backup can be viewed and modified in the `/etc/kernun-fsdb-include` file. During backup and restore operations, this file is passed to **diskdb**(1) using the `-I` parameter.

Backup files are stored in the `/data/backup` directory, from which they should be copied to a safe place. They should not be renamed, because their names contain important information for backup processing: the build number of the Kernun UTM instance, the number of the backed up system partitions, and the date and time when the backup was created.

A backup created on a particular Kernun UTM version (build number) should be restored to a system partition containing a newly installed image with the same build number. On the other hand, a system partition with any partition number can be used for restoring, not only the one where the backup was created. The restore program adjusts the contents of the file system table `/etc/fstab` accordingly.

Kernun UTM provides tools for manual backup and restoring of system partitions using local backup files in `/data/backup`. The administrator should create a backup at least after every major configuration change and copy it to a storage medium other than a local disk. Solutions for automated backup, remote backup, or backup of the data partition are not provided out of the box, because backup policies required for different deployments vary significantly. More sophisticated backup scenarios can be implemented using operating system tools (**tar**(1), **cron**(8), etc.) or various third-party backup software. The Kernun UTM tools support only complete restoring of a backup to a newly installed system partition. Nevertheless, a backup file is a **tar**(1) archive compressed by **bzip2**(1) and can therefore be freely manipulated using these tools.

In some situations, especially when a backup is restored to a different version of Kernun UTM or to a system partition that has been modified since the installation, *conflicts* may be reported during restoring. It is also possible that unresolved conflicts from an earlier restore operation interfere with the current one. In such a case, the old conflicts need to be resolved or discarded first. See Section 2.7 for explanation of conflicts and instructions on how to resolve them.

[8] for example, access rights, owner, or modification time

2.6.1 Backup and Restoring from the GUI

A backup can be created in the GUI in the Kernun systems tab of the System Manager (Figure 2.6). All you need to do is select a system partition and click on the Backup button. A backup file will be created and stored in `/data/backup`. The new backup will appear in the Backups tab, see Figure 2.8. Using buttons under the list of backup files, a file can be downloaded to the administrator's computer, uploaded back to Kernun UTM, or removed.

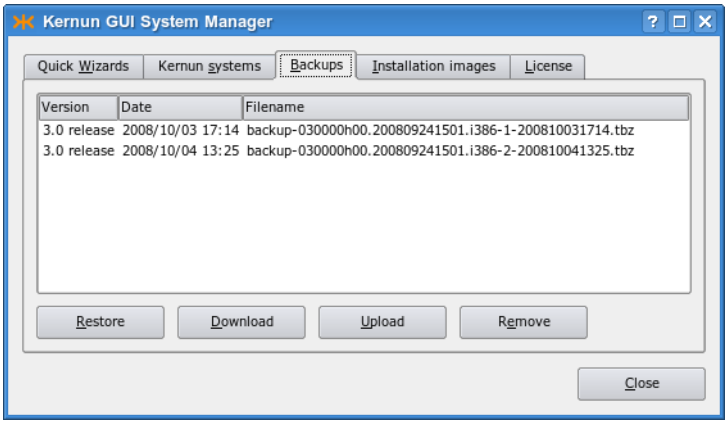

Figure 2.8: Existing backup files in the GUI

Click on the Restore button if you want to start the restore operation. Alternatively, restoring can be initiated using Restore backup on the Quick Wizards page. A wizard window appears. It prompts for the target system partition (must not be the currently booted one), for selection of a backup file and for a corresponding installation image. There are also buttons for uploading a locally stored backup or image to Kernun UTM. As the last step, the recapitulation of the selected values is displayed, as shown in Figure 2.9. When you click Finish, the selected image is installed in the chosen system partition and the selected backup is unpacked. Then it is possible to do any combination of the following operations: set the newly restored partition as the default boot partition; change the partition label; reboot Kernun UTM immediately (see Figure 2.10).

2.6.2 Backup and Restoring from the Command Line

The **sysmgr**(8) utility is used to create and restore backups from the command line. A new backup file in `/data/backup` is created by the following command:

```
[root@fw ~]# sysmgr backup 2
Creating backup content database /kernun-backup.fsdb.bz2
Creating file system content database
Creating backup file
    /data/backup/backup-030000h00.200809241501.i386-2-200807281400.tbz
[root@fw ~]#
```

If a backup of the current system partition is to be created, the partition number (**2** in our example) may be omitted. A list of existing backup files is displayed by

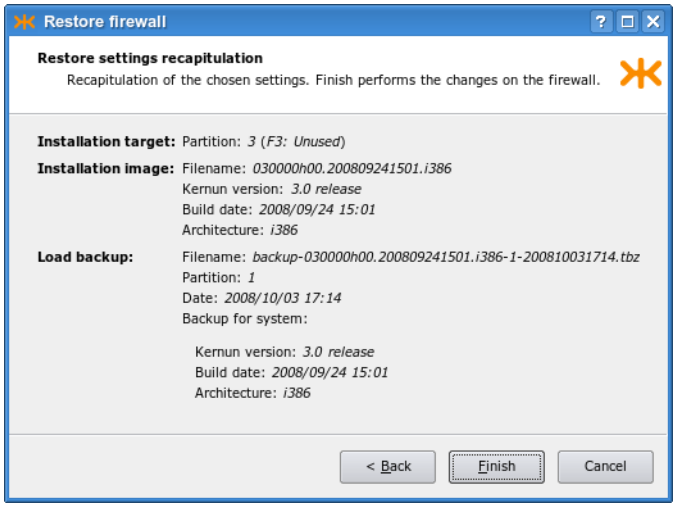

Figure 2.9: Parameters of a restore operation

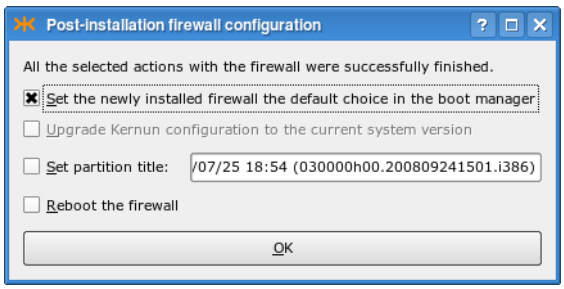

Figure 2.10: Final settings after restoring a backup

```
[root@fw ~]# sysmgr backups
backup-030000h00.200809241501.i386-1-200810031714.tbz
backup-030000h00.200809241501.i386-2-200809261822.tbz
backup-030000h00.200809241501.i386-2-200809301350.tbz
```

A backup can be restored to a selected system partition; it must not be the currently used system partition. A clean installation of an image with the correct build number should be done first.

```
[root@fw ~]# sysmgr install 2 030000h00.200809241501.i386
...
Installation successfully finished
[root@fw ~]# sysmgr restore 2 \
> backup-030000h00.200809241501.i386-1-200810031714.tbz
Processing changes of file system contents
Unpacking files from backup
Resolving conflicts
All conflicts resolved
[root@fw ~]#
```

An attempt to restore a backup in a system partition that contains a Kernun UTM instance with a different build number is detected and a warning is displayed:

```
[root@fw ~]# sysmgr restore 2 \
> backup-030000h00.200809241501.i386-1-200810031714.tbz
Backup is from different build than currently installed in /2.
Installed: 030001h00.200811142135.i386
Backup:    030000h00.200809241501.i386
It is strongly recommended to restore a backup to the Kernun build that
was used for creating the backup.
Continue anyway (y/n)? n
[root@fw ~]#
```

2.6.3 Restoring a Backup in the Standalone Installer

A backup can be restored also from the standalone installer booted from the Kernun UTM installation medium. This can be helpful after installing a new system disk or when moving a Kernun UTM installation to a new computer. First, select a system partition and install Kernun UTM from an image corresponding to the backup that is to be restored, following the procedure described in Section 2.5.1. If the backup file is not already located in `/data/dist`, you can copy it there using the emergency repair environment tools, as described in Section 2.9.

```
*** KERNUN INSTALLATION ***

Build 030001h00.200811142135.i386
```

```
1. Install Kernun
2. Check for existing Kernun installations
3. Restore backup
4. Start rescue shell
5. Mount Kernun file systems
6. Resize installer's in-memory temporary file system (current size 32m)
7. Halt
8. Power down
9. Reboot
0. Install license

Select action: 1 ❶

Detected Kernun system disk ad0
Detected Kernun data disk ad0
Repartition disks (y/n)? n
Current Kernun installations:
Boot manager on /dev/ad0
F1: Kernun 3.0 2008/10/01 07:36 (030000h00.200809241501.i386)
F2: Kernun 3.0.1 2008/11/17 16:39 (030001h00.200811142135.i386)
F3: Unused
type=Kernun 1024 B boot manager (74 character labels)
current_booted=
bootable=1 2
update=yes
default_selection=F2
Select partition for installation (1 2 3) [1]: 3 ❷
Overwrite partition /dev/ad0s3 by new Kernun installation (y/n)? y
Available installation images:
     1  030000h00.200809241501.i386
     2  030001h00.200811142135.i386
Select image to install (1-2) [2]: 1 ❸
Enter the label that will be used to identify this installation in the
boot manager. The label can be at most 44 characters long. The Kernun
build number will be appended after the entered label automatically.
Label [Kernun 3.0 2008/11/20 10:26]:
Clearing system partition 3
...
Installing kernun-030000h00.200809241501.i386.txz to system partition 3
...
Installation successfully finished

Press Enter for return to menu...
```

```
*** KERNUN INSTALLATION ***

Build 030001h00.200811142135.i386

1. Install Kernun
2. Check for existing Kernun installations
3. Restore backup
4. Start rescue shell
5. Mount Kernun file systems
6. Resize installer's in-memory temporary file system (current size 32m)
7. Halt
8. Power down
9. Reboot
0. Install license

Select action: 3 ❹

Select partition to be restored (1 2 3) [1]: 3 ❺
Available backups for build installed in partition 3:
     1  backup-030000h00.200809241501.i386-1-200810010405.tbz
     2  backup-030000h00.200809241501.i386-1-200810040604.tbz
Select backup to restore (1-2) [2]: 1 ❻
Restoring backup-030000h00.200809241501.i386-1-200810010405.tbz
to partition 3
Are you sure (y/n)? y
Conflicts resolution data in /data/restore already exist ❼
Remove old /data/restore (y/n)? y ❽
Processing changes of file system contents
Unpacking files from backup ❾
Resolving conflicts
All conflicts resolved

Press Enter for return to menu...
```

In the example above, we assume that the backup file is already stored in the `/data/backup` directory and the corresponding installation image in the `/data/dist` directory. We start the backup restoring procedure by carrying out a fresh Kernun UTM installation ❶ in an unused system partition ❷. The installation image ❸ is chosen so that it corresponds to the backup file that will be restored. After returning to the installer main menu, we select Restore backup ❹. The partition ❺ installed in the previous step should be selected. A list of backups compatible with the content of the target system partition is displayed. We choose one of the offered backup files ❻ and the restoring begins. The message ❼ indicates that there are unresolved conflicts from previous restore or upgrade operations. Usually, you should reply **n** to the question ❽. This will interrupt

the restore operation. You can restart it after you resolve the conflicts according to instructions given in Section 2.7. If you are sure that you do not need to resolve the old conflicts[9], you may reply **y** and the conflict resolution data will be deleted. The message and question concerning the old conflicts ❼❽ will not be displayed if there are no pending conflicts. Finally ❾, the backed up files are unpacked from the backup file and checked for conflicts. No conflicts should occur if the backup is restored to the same Kernun UTM build that was installed at the time the backup was created. The restored files are installed in their proper places and the restore operation successfully finishes.

2.7 Upgrade

The upgrade procedure described in this section is applicable if you want to retain as much as possible from the configuration of the old Kernun UTM instance in the new instance. If you want to configure a new Kernun UTM version from scratch, follow the installation and configuration procedures described in Section 2.5.

Upgrading to a new version or build of Kernun UTM is basically done by restoring a backup of the old version in a system partition that contains a fresh installation of the new version. The upgrade procedure comprises the following steps:

1. Normalizing the configuration.
2. Backing up the system partition with the old version.
3. Installing the new version.
4. Restoring the backup created in step 2 to the installed new version.
5. Resolving any conflicts arisen in step 4.
6. Upgrading the Kernun UTM configuration file.
7. Checking the upgraded configuration.
8. Applying the upgraded configuration.
9. Starting the new Kernun UTM version.

The syntax and semantics of the configuration files are sometimes slightly changed between versions. In order to be usable in the new Kernun version, the old configuration file must be converted in step 1. This is done automatically during the upgrade process. The configuration conversion script expects the configuration in a normalized format. Using just some formatting accepted by normal Kernun configuration tools (GUI or CML) is not sufficient. The normalization during the upgrade process is done either automatically (by GUI), or using a command in the command line upgrade.

[9] for example because they are in a system partition that is not used any more

Important

If you are upgrading from a Kernun version that does not implement automatic configuration normalization during the upgrade process, that is, from a version older than 3.3.2, you should perform the normalization manually. It can be done simply by opening and saving the configuration by either GUI, or CML. The normalization step may be skipped if the configuration has been saved recently and has not been modified outside the Kernun configuration tools, for example, by a text editor.

The upgrade operation results in a newly installed system partition that contains the new version of Kernun UTM. If we want to keep the configuration across upgrades, we need to copy the main Kernun UTM configuration file `/usr/local/kernun/conf/kernun.cml` and any other changes done in the old installation to the new one. The configuration files that have been changed, created, or deleted since the installation are found and saved when the old system partition is backed up in step 2.

Step 3 requires a full or patch installation image. Although it is possible to replace the contents of the currently used system partition with the new version, it is not recommended. You should always install an upgrade to a currently unused system partition, for two reasons. First, the old Kernun UTM instance can continue running until the upgrade is finished. Second, you can quickly return Kernun UTM to an operational state if something goes wrong with the upgrade.

Tip

The recommended practice is to use two system partitions for regular upgrades. One partition is occupied by the currently running version, while the other contains the old version and will be used for installation of the next upgrade. After each upgrade, the roles of the two partitions are switched. The third system partition can be reserved for special tasks, such as preparation of a completely new configuration.

Tip

Set the boot manager (as described in Section 2.4) default boot partition so that it always boots the currently used Kernun UTM instance. Consider disabling the automatic updating of the default boot partition or disabling the unused partitions altogether.

Restoring of the backup from step 2 in the system partition installed in step 3 effectively copies the complete configuration from the old system partition. Restoring of a backup to a build different from the one used for its creation may cause *conflicts*. These are files that cannot be restored automatically and a manual intervention of the administrator is necessary. A conflict occurs if there are two incompatible changes of the same file. The original version of the file comes from the installation image of the Kernun UTM instance that is being upgraded; we will call it "old". The second version (called "backed-up") is contained in the backup file, if the file was

changed[10] at some time between the installation of the old version and the start of the upgrade process. The third version of the file (called "new") is obtained from the installation image of the new Kernun UTM instance installed in step 3. There are two potential changes of the file. One between the old and the backed-up version, the second between the old and the new version. If only one change exists, no conflict occurs and the changed (backed-up or new) version of the file will be used. For example, `/etc/ttys` may have been changed by the administrator in the installed Kernun UTM, but remains the same in the build we are upgrading to. Another example is a proxy executable, which is modified in the new Kernun UTM version, but left unchanged by the administrator. If all three versions exist, i.e. when the backed-up and the new version differ, a conflict occurs. The automated upgrade tools are unable to handle the file and the administrator must decide whether the new file, the backed-up file, or some combination of the two should be used. For example, a third party software added in the new build creates a new user account in `/etc/master.passwd`, and the administrator has created another user account. During the upgrade, a conflict is reported for `/etc/master.passwd`. The administrator can resolve this particular conflict by merging the two versions of the file, adding both new user accounts to the resulting file.

The detected conflicts are recorded in the `/data/restore/resolve` file during step 4. The conflicting files from the backup file (the "backed-up" version) are not unpacked to the root directory tree. Instead, they are stored in corresponding locations under the `/data/restore/conflicts` directory. The root directory tree contains the files as installed (the "new" version). In step 5, the administrator specifies for each file how the conflict should be resolved, choosing from the following possibilities:

- The new version is retained and the backed-up version is deleted from `/data/restore/conflicts`.
- The backed-up version replaces the new version.
- The new or the backed-up version is used, but is modified first, for example by merging the contents of the two versions in a text editor.
- The conflict is postponed until a later iteration of conflict resolution.

The `/data/restore` directory is deleted when all conflicts are resolved. Only one upgrade procedure can be in the conflict resolution stage at a time. If a conflict resolution session is started and there is already the `/data/restore` directory with unresolved conflicts, the administrator can either cancel the second resolution, or delete the old `/data/restore` directory, thus effectively using the "new" versions of the files for all conflicts in the earlier conflict resolution session.

In step 6, a script is executed that edits the contents of the main configuration file `/usr/local/kernun/conf/kernun.cml` to make it compatible with the upgraded Kernun UTM. Sometimes, if there are complex changes in the configuration syntax and semantics between the two Kernun UTM versions, or if the configuration file contains certain

[10] By a change, we mean modification of the contents of the file, deletion of the file, a change of the file attributes (e.g., the owner or access rights), or creation of a previously nonexistent file.

advanced constructs, the script may be unable to perform a perfect conversion. It is therefore recommended to always check the result of the automatic conversion in step 7.

The new configuration file needs to be applied before the upgraded system can be put into normal operation. The low-level configuration files are generated and the configuration is applied in the context of the newly installed system using the **applycfg** command of **sysmgr**(8). If the generation or application of the configuration fails, the configuration should be corrected and applied again.

Finally, the upgraded Kernun UTM can be put into the normal production mode by rebooting to the newly installed system partition.

No modifications of the configuration (steps 6 and 7) are often required during the upgrade procedure. This is usually true when upgrading between two builds of the same version or between patch releases of the same version, for example, from 3.0 to 3.0.1 or from 3.0.1 to 3.0.2.

2.7.1 Upgrade from the GUI

An upgrade is initiated from the Quick Wizards page of the System Manager. There are two alternatives. Click Upgrade Firewall if you want to start the complete upgrade procedure. If you already have a recent backup of the system partition that you want to upgrade, you can skip the first step — creation of a backup. In this case, use the Restore backup into newer firewall button. We will describe only the former alternative; the latter is almost identical, only the backup step is missing.

The GUI assumes that we want to upgrade the currently running Kernun UTM instance. Therefore, the current system partition will be backed up. After clicking on the Upgrade Firewall button, we select the target system partition in which the upgraded Kernun UTM will be installed. Then we select the installation image of the new version. Our selections are displayed in the settings recapitulation window (Figure 2.11). Click on the Finish button to start the upgrade.

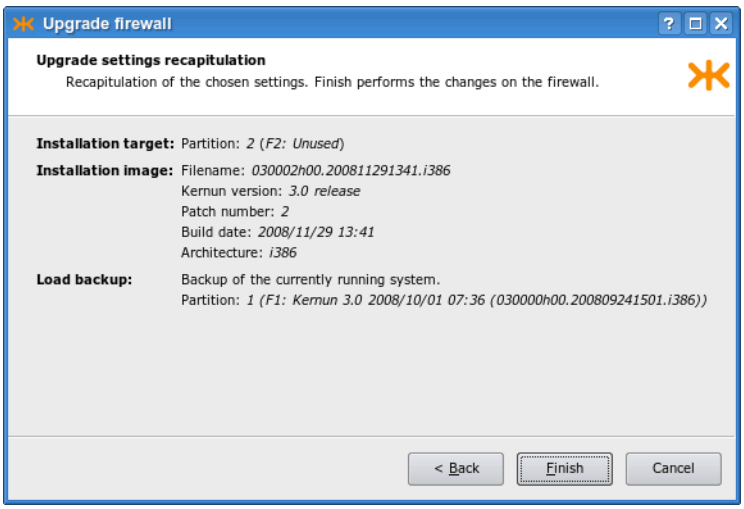

Figure 2.11: Parameters of an upgrade operation

The GUI displays the progress of the upgrade procedure. First, the current system partition

is backed up. Then, the new system partition is installed and the backup is restored in it. If there are any conflicts, the conflict resolution window is displayed, as shown in Figure 2.12. The window shows a list of conflicting files. You can determine how to resolve the conflict of a file by clicking in the Action column. The following actions are possible:

- + — uses the "backed-up" version of the file;
- . — uses the "new" version of the file, as installed from the new installation image;
- – — deletes the file;
- ! — postpones the conflict to the next iteration of conflict resolution.

It is also possible to select a file and then click a button on the right-hand side of the window to display the differences between the two versions of the file, or to open one of them in an editor.

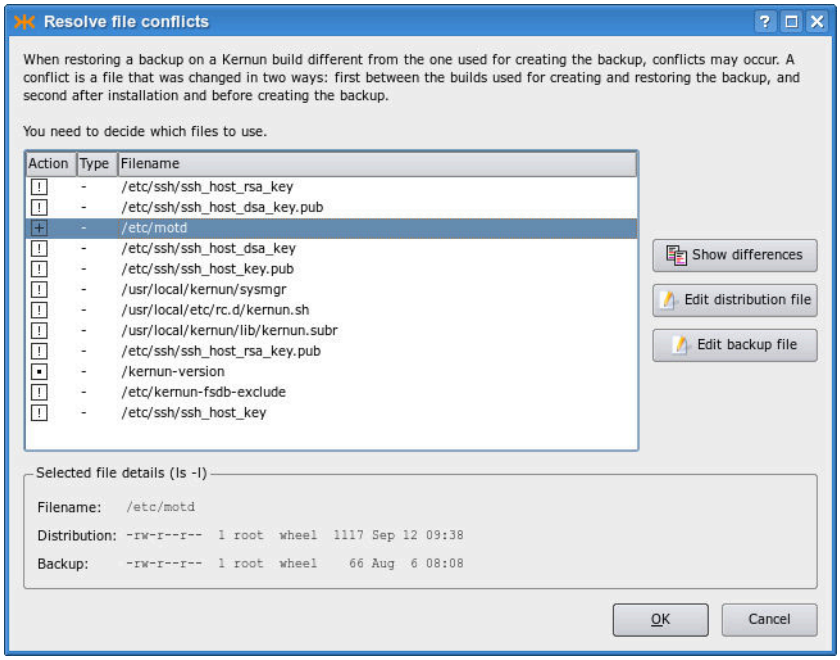

Figure 2.12: The conflict resolution window during an upgrade

After you give instructions for conflict resolution and optionally edit some conflicting files, click OK to have the conflicts resolved. Finally, a window is displayed (see Figure 2.13) that makes it possible to realise any combination of the following actions: set the newly upgraded system partition as the default boot partition; run the configuration conversion script; change the partition label; reboot Kernun UTM immediately.

2.7.2 Upgrade from the Command Line

Command line upgrades are realized using the **sysmgr**(8) utility. Unlike when using the GUI, which performs all the required steps automatically, a command line upgrade must be done step by step. An example of the upgrade procedure follows:

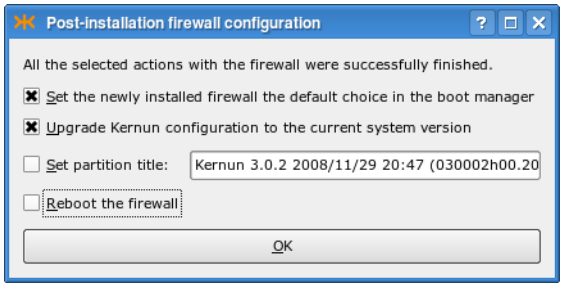

Figure 2.13: Final settings after an upgrade

```
[root@fw ~]# cml -l -f /usr/local/kernun/conf/kernun.cml ❶
RCSL-730-N File '/usr/local/kernun/conf/kernun.cml' locked for current user
CMLM-790-N RCS command completed
[root@fw ~]# sysmgr checkcfg ❷
...
Configuration is correct
[root@fw ~]# sysmgr backup ❸
Creating backup content database /kernun-backup.fsdb.bz2
Creating file system content database
Creating backup file
/data/backup/backup-030000h00.200809241501.i386-1-200811300006.tbz
[root@fw ~]# sysmgr install 2 030002h00.200811291341.i386 ❹
Clearing system partition 2
...
Installation successfully finished
[root@fw ~]# sysmgr upgrade 2 \
> backup-030000h00.200809241501.i386-1-200811300006.tbz ❺
Processing changes of file system contents
Unpacking files from backup
Resolving conflicts
There are pending conflicts, see /data/restore/resolve


*** CONFLICT RESOLUTION *** ❻

1. Resolve with easy editor (ee)
2. Resolve with editor vi
3. Do not resolve now

Select action: 2
# Conflict resolution file for system partition /2 ❼
# Each line of this file contains an instruction for one file. You
```

```
# can edit the file and then apply the instructions by running
# "sysmgr resolve". Every line contains three fields:
# - one character that defines an action to be done with the file
# - one character for file type ('d' for a directory, '-' for any
#   other type)
# - path to the file, interpreted either relative to /2 for the
#   existing file and relative to //data/restore/conflicts
#   for the file from the backup
# Procedure of conflict resolution:
# 1. Locate all lines beginning with '!'. These denote conflicting
#    files.
# 2. Optionally edit the conflicting files.
# 3. Change the character '!' to
#         + ... to use the file from backup, temporarily stored in
#               /data/restore/conflicts
#         . ... to keep the current file
#         - ... to delete the current file
#         ! ... keep the conflict for future resolution
# 4. Run "sysmgr resolve".
# 5. Repeat steps 1-4 until all conflicts are resolved.
# Merging file from the backup with the current file can be done
# either by editing the current file and specifying action '.'
# (keep) or editing the file from the backup and specifying
# action '+' (use backup).
! - ./etc/motd
! - ./etc/login.conf
...
Resolving conflicts
There are pending conflicts, see /data/restore/resolve ❽
[root@fw ~]# vi /data/restore/resolve ❾
...
[root@fw ~]# sysmgr resolve ❿
Resolving conflicts
All conflicts resolved
[root@fw ~]# sysmgr upgradecfg 2 ①
Upgrading Kernun configuration /2/usr/local/kernun/conf/kernun.cml
/2/usr/local/kernun/conf/kernun.cml,v  <--
  /2/usr/local/kernun/conf/kernun.cml
new revision: 1.2; previous revision: 1.1
done
Automatic configuration upgrade done. It is recommended to review the
configuration before returning Kernun UTM to production use.
[root@fw ~]# sysmgr applycfg 2 ②
```

```
...
System kernun applied in system partition 2
[root@fw ~]#  cml -u -f /usr/local/kernun/conf/kernun.cml ③
CMLM-790-N RCS command completed
[root@fw ~]#
```

Before upgrade, the configuration should be locked ❶[11], checked and normalized ❷. This step ensures that the configuration upgrade step ① will understand the configuration file. The upgrade procedure starts by backing up the current system partition ❸. Specify a system partition number to upgrade a currently inactive partition. If a recent backup already exists, this step can be skipped. A new Kernun UTM version is installed to an unused system partition ❹. This command also sets the default boot manager label for the newly installed partition, and makes it bootable and the default boot selection for the next booting. The backup is then restored to the newly installed system partition ❺. This command writes the list of conflicts to `/data/restore/resolve`. The conflicting files from the backup are stored in the `/data/restore/conflicts` directory. If there are conflicts, the conflict resolution menu is displayed ❻. You can either resolve the conflicts, or postpone the conflict resolution to do it later. If you choose to resolve the conflicts, the conflict resolution file `/data/restore/resolve` is opened in a text editor. Edit the file according to the displayed instructions ❼ to determine the way of resolution of individual conflicts. After the file is saved and the editor is terminated, the conflict resolution is executed in accordance with the file. If some conflicts remain unresolved, a message ❽ is printed. It is then possible to edit `/data/restore/resolve` manually ❾ and restart the conflict resolution ❿. Commands ❾ and ❿ can be repeated until all conflicts are resolved. The main configuration file is upgraded ① and applied ②. Finally, the lock is released ③. You can then reboot to the new system partition and start using the upgraded Kernun UTM.

2.8 Audit

The Kernun auditing tool **kernun-audit**(1) provides a convenient source of information about bugs discovered in the Kernun software. The auditing tool also reports when a new software version becomes available. A Kernun audit is usually executed daily by the **cron** daemon via the **periodic** command. It downloads the up-to-date auditing database, and then examines the product type, version, and architecture of the installed system. Based on these values, the relevant records are extracted from the database and reported. There are two classes of records: bugs and software updates.

Each bug that is discovered in the currently installed version of the Kernun product is reported. A bug has a unique identification number, a description, a list of versions, in which it occurs, a solution, and a workaround. The recommended solution is always a software update to a version in which the bug has been fixed (if such version is available). The workaround (if available) describes how to minimize the impact of the bug without updating the software. It should be applied if the software has not been fixed yet or if an immediate update is infeasible. Nevertheless,

[11] Steps ❶ and ③ are supported by Kernun since release 3.6. In the prior versions, these steps should be skipped.

the workaround should always be regarded as a temporary solution and the Kernun installation should be updated as soon as possible.

Software updates are reported only for the same product and architecture as in the installed system. The latest patch release from each release branch is shown. Only versions newer than the currently installed version are displayed. For example, if 3.1 is the version installed and 3.0–3.0.6, 3.1–3.1.3, and 3.2–3.2.1 are available, 3.1.3 and 3.2.1 will be the versions reported.

The initial configuration of a Kernun system runs the auditing tool daily using the `DEFAULT-CRONTAB` and `DEFAULT-PERIODIC` variables from the included crontab and the periodic configuration file `crontab.cml`. Auditing can be disabled by setting `daily_status_security_kernun_audit_enable` to `"NO"` in that file. The auditing tool **kernun-audit** can be also executed manually from the command line. The product name, version number, and architecture name are obtained from the current system, or can be specified using the command line arguments of **kernun-audit**. The identification of the current system is stored in the files `/kernun-product` (product name) and `/kernun-version` (build number, which contains the version number before the first dot and the architecture name after the second dot). If the location (local or remote) of the audit database is not specified, the database is downloaded from `download.kernun.com` by default.

The `www.kernun.com` Web site provides an online version of the Kernun auditing tool. After filling the Kernun product, version, and architecture in a form, the auditing report is generated in the same format as the one **kernun-audit** produces.

2.9 Emergency Repair Environment

> **Warning**
>
> The instructions in this section are intended for experienced administrators with profound knowledge of Kernun UTM and FreeBSD.

The Kernun UTM installer booted from the installation medium can be used to repair the system if all system partitions are unable to boot. The available functions are accessible from the installer main menu:

```
1. Install Kernun
2. Check for existing Kernun installations
3. Restore backup
4. Start rescue shell
5. Mount Kernun file systems
6. Resize installer's in-memory temporary file system (current size 32m)
7. Halt
8. Power down
9. Reboot
0. Install license
```

Option **1** is described in Section 2.5.1. Options **7**, **8**, and **9** are self-descriptive. Option **2** displays the boot manager configuration and the disk device names.

```
System disk is /dev/ad0
Boot manager on ZFS pool 'kernun'
F1: Kernun 3.11 2018/06/06 07:36 (031100h00.201806111345.amd64)
F2: Unused
F3: Unused
type=Kernun ZFS boot manager ver. 1.0
current_booted=NONE
bootable=1
update=1
default_selection=F1
```

Option **3** restores a backup selected from a list of backup files found in `/data/backup`. If the backup is stored on another medium, it must be first copied to the `/data/backup` directory, using for example the rescue shell (option **4**). For details about backup and restoring, see Section 2.6.

Option **4** starts a rescue shell (**bash**). It provides the environment for emergency maintenance of a computer with non-bootable Kernun UTM installations. The rescue shell (as well as the whole standalone installer) runs in a custom FreeBSD environment. The standard Kernun UTM kernel is used. The root file system is mounted from the installation medium and is therefore read-only. A read-write RAM disk for temporary data is mounted under `/tmp`, symlinked also from `/var/tmp`. The standard size of the RAM disk is 32 MB. It can be resized using option **6** of the installer main menu.

Warning

The content of the RAM disk is lost when the installer is terminated or when the RAM disk is resized.

Caution

Do not make the RAM disk too large, because its content is stored in the kernel memory. If the free kernel memory gets too low, the kernel may panic.

Option **5** of the menu mounts any existing Kernun UTM partitions under the directories `/1`, `/2`, `/3` (the system partitions), and `/data` (the data partition). The rescue shell provides many standard FreeBSD command line programs. Programs from a mounted Kernun UTM system partition can be run as well.

Tip

It is often useful to perform a **chroot**(8) to a mounted Kernun UTM system partition and to run commands in the chrooted environment.

2.10 Running in virtual machine environment

Kernun can be deployed in any virtualization environment with support for FreeBSD OS.

See also the notice in Section 5.24.

The following virtualiztion environments are tested for compatibility with Kernun.

2.10.1 VMware

The **emulators/open-vm-tools-nox11** is installed and staretd automatically if the VMWare environment is detected.

2.10.2 Hyper-V

The virtualization support is built in the FreeBSD OS. See also `https://docs.microsoft.com/cs-cz/windows-server/virtualization/hyper-v/Supported-FreeBSD` and `https://wiki.freebsd.org/HyperV` for more detailed information.

2.10.3 VirtualBox

The **emulators/virtualbox-ose-additions** is installed and started automatically if the VirtualBox environment is detected.

2.10.4 XEN

The **sysutils/xe-guest-utilities** is installed and started automatically if the XEN environment is detected.

Chapter 3

Kernun UTM can be administered locally, or remotely via the network. Local administrator access is usually limited to the initial installation, when the network is not yet configured, and emergency situations after a failure or misconfiguration if the network is not accessible. The administrator can access the system locally via a text system console. It provides the same set of command line tools as the remote text login via SSH. In normal operation, Kernun UTM is usually administered remotely. There are two options for remote access: a text command line interface and a graphical user interface.

An administrator can log in to Kernun UTM remotely via SSH and get shell access to the system. Administrative tools accessible from the shell include the primary Kernun UTM command line control and configuration tools **kat**(8) and **cml**(8), see also Section 3.2. Besides these two, many additional command line utilities are available, including specific Kernun UTM commands introduced in Section 3.3 and all the standard FreeBSD commands.

Kernun UTM's graphical user interface (GUI for short, described in Section 3.1) provides a similar functionality as the command line utilities, but in a more intuitive and comprehensive way. It shows the current state of all system components and can display details for each component. The GUI contains also a powerful log analyzer, a configuration editor, and a system manager, which administers installations, backups, and upgrades. There is some functionality that is unique to the GUI and cannot be accessed from the command line, such as displaying of performance graphs. The GUI runs on the administrator's local computer and communicates with Kernun UTM via the network, using SSH internally. Hence, the same prerequisites are needed for both the command line and GUI access to Kernun UTM (especially, SSH keys and the SSH protocol enabled on the way between Kernun UTM and the administrator's computer).

3.1 Graphical User Interface

In this section, we will introduce the Kernun GUI. In later chapters dealing with configuration (Chapter 4, and Chapter 5) we will assume that the administrator knows how to use the GUI.

The Kernun GUI is available in two functionally equivalent versions: for Microsoft Windows and for UNIX. The binary executables of the Kernun GUI are distributed for MS Windows and

for FreeBSD [1]. The Kernun GUI can be easily compiled also for other UNIX platforms. See the README file for instructions on how to compile the Kernun GUI.

On UNIX machines, the Kernun GUI expects OpenSSH to be installed (namely, the **ssh**, **ssh-agent**, **ssh-keygen** and **ssh-add** programs are expected to be located in a directory listed in the `PATH` environment variable).

There are no prerequisites to be installed on MS Windows. All the necessary executables are included in the Kernun GUI distribution.

The Kernun GUI provides the following functionalities:

- monitoring of the state of Kernun UTM;
- management (starting, stopping, restarting, ...) of Kernun UTM (or its particular components);
- work with logs (both current online logs and downloaded offline logs) and statistics;
- modification and application of the configuration of Kernun UTM;
- administration of Kernun UTM installations, backups, and installation images.

3.1.1 Kernun GUI Launcher

When the GUI is started, the launcher window is displayed, see Figure 3.1. The launcher provides buttons to open (and change) a local copy of the Kernun UTM configuration file (for more information on work with the configuration in the GUI see Section 3.1.4) or examine a local log file (see Section 3.1.3). However, the main purpose of the GUI launcher is to establish a new connection to Kernun UTM and launch the main GUI management window.

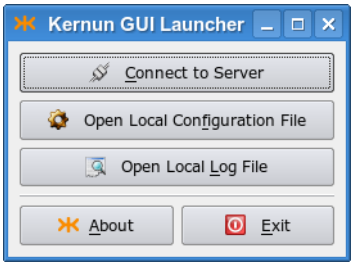

Figure 3.1: GUI Launcher

The GUI communicates with Kernun UTM via SSH connections. You therefore need to have the **sshd** service running and correctly configured (see Section 4.2.3). The parameters of the SSH connection to Kernun UTM are specified in the Connection Parameters dialog, as depicted in Figure 3.2. You need to fill in the Host name or IP address of the Kernun UTM machine, Username, Port and select the SSH key file.

[1] The binary executables are compiled for the version of FreeBSD used by Kernun UTM.

Tip

If you are connecting to Kernun UTM via SSH for the first time, you need to initialize Kernun UTM (i.e., download your private SSH key from Kernun UTM). To do this, use the dialog that appears after the Initialize new system button is pressed. Fill in the Hostname and the Password you entered during the installation. See also Section 2.5.2.

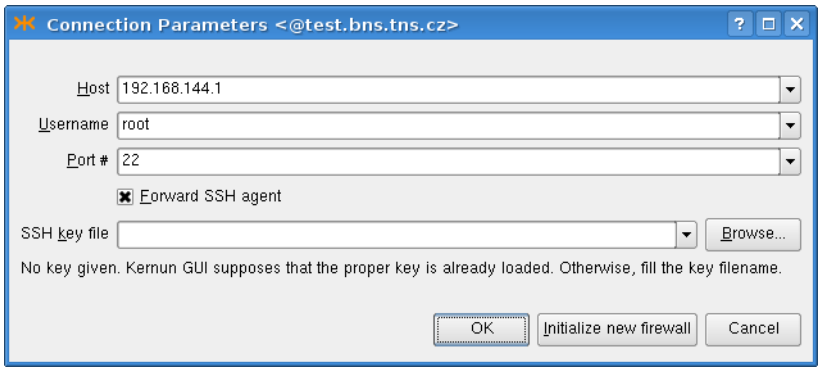

Figure 3.2: Connecting to Server

If you intend to apply the configuration to other Kernun UTM than the one you are logged in (for example, the second Kernun UTM in a cluster), you must check the Forward SSH agent check box. See **ssh**(1) for information about security risks of SSH agent forwarding.

Important

It is unsafe to leave the SSH keys loaded in the **ssh-agent** after finishing your work with Kernun UTM.
The key is deleted automatically on *UNIX*, if there was no **ssh-agent** running before the Kernun GUI was started. Otherwise, you need to unload it yourself (e.g. using **ssh-add -d *private key file***).
On *MS Windows*, the Kernun GUI instances are managed by the GUI launcher, which is set to unload the keys automatically (after a timeout) if there are no main Kernun GUI applications running. You can change this behavior or unload the key manually using the context menu of the Kernun GUI taskbar icon.

3.1.2 GKAT—Management Console

The functionality of the Kernun GUI main window, depicted in Figure 3.3, is basically equivalent to the command line administrative tool **kat**(8). It displays the states of individual Kernun UTM components and allows them to be started, stopped, and monitored.

When connected to Kernun UTM, the state of the proxies and other system components is indicated by their state icons. The states of the components are also propagated to the state icon of

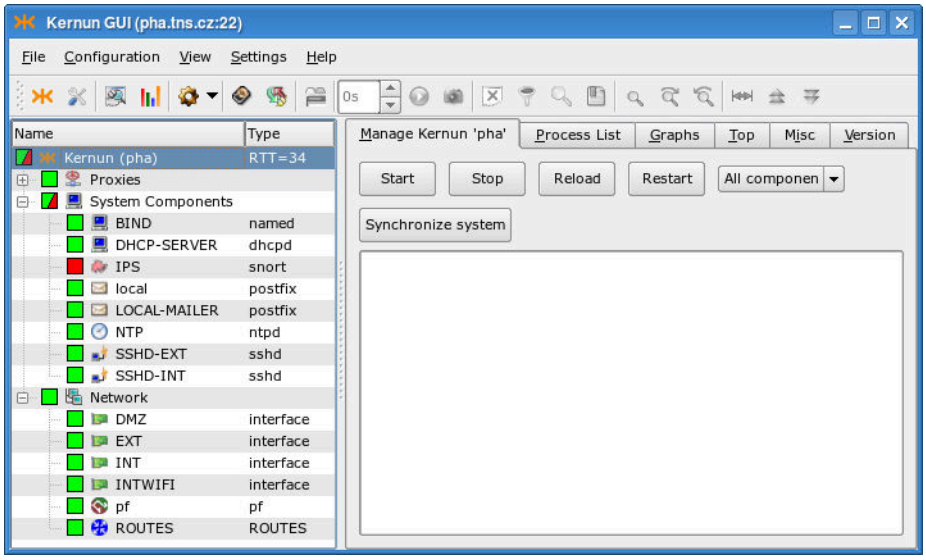

Figure 3.3: GKAT — Kernun UTM management console

their parent component groups and of the whole Kernun UTM. There are the following component groups: Proxies, System Components (such as SSH servers, mail forwarders, or DNS servers), Network (interfaces, packet filtering, routing), and Open VPN servers. A running component is denoted by green icon background ■, whereas the background of a stopped component is red ■. There are several icon overlays that indicate further information concerning the component's state:

- ◩ *Not up-to-date configuration* — the configuration of the component has changed; reload it, so that the changes can take effect.
- ◩ *Parent exiting flag* — the proxy is in a special state: it does not accept new connections, but only waits for the already active sessions to finish. For example, this state may appear when a proxy is reloaded, some sessions remain open in the old proxy instance, and only the new instance accepts new connections. Total restart of the proxy stops the old proxy instance and starts another, so no sessions remain open.
- ◩ *Not in configuration* — the system component is not in the configuration. Kill the component to solve the problem.
- ◩ *Component's state changing* — this overlay is displayed while the component's state (started/stopped/restarted/reloaded) is being changed. It disappears immediately after the action is finished.

In the situation depicted in Figure 3.3 we know that all proxies are running, even though the Proxies subtree is collapsed. The IPS component is stopped, which is why the System Components and Kernun icons are partially green and red. Kernun UTM has four network interfaces, the packet filter and the routing table running. No Open VPN server is configured. You can click on a component, group, or the whole Kernun UTM icon in the proxy tree to select it and display its details in the right-hand part of the window. The information about the *RTT* (Round Trip Time) in milliseconds between the GUI and Kernun UTM is displayed next to the Kernun UTM name.

Kernun UTM Details

On the Manage page of the whole Kernun UTM (that is, with the top-level Kernun node selected from the component list in the left-hand part of the window), the administrator can easily manage (start/stop/reload/restart) the whole Kernun UTM, as depicted in Figure 3.4.

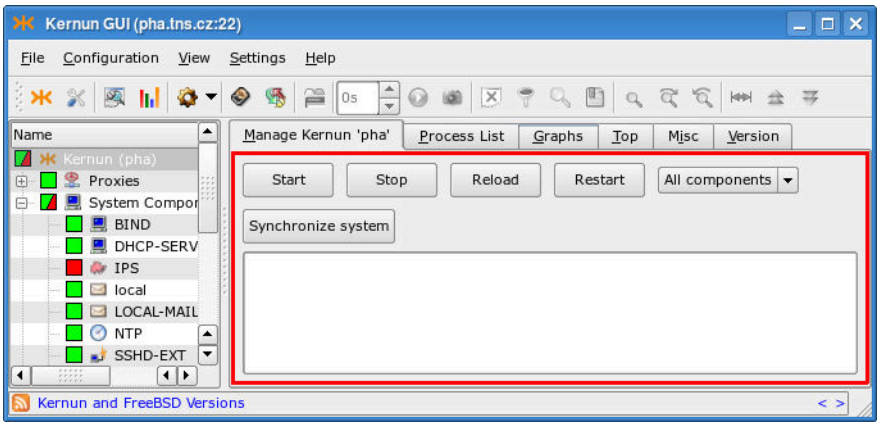

Figure 3.4: Kernun UTM Manage Page

The selected action can be applied either to all components, or to the components marked with a tag in the configuration. The tag or the All components option can be selected from a combo box. A change in the configuration takes effect only after the system state is synchronized with the updated configuration. This can be always done by rebooting, restarting, or reloading the whole Kernun UTM, but it often suffices to restart only a subset of components, while the remaining parts of Kernun UTM may be left running. The Synchronize system button automates this process. It displays a window (depicted in Figure 3.5) that lists the actions required to bring all the components into sync with the configuration. You can manually alter the proposed actions by clicking on a component in the Action column. When you are satisfied, click OK and all the selected actions will be executed.

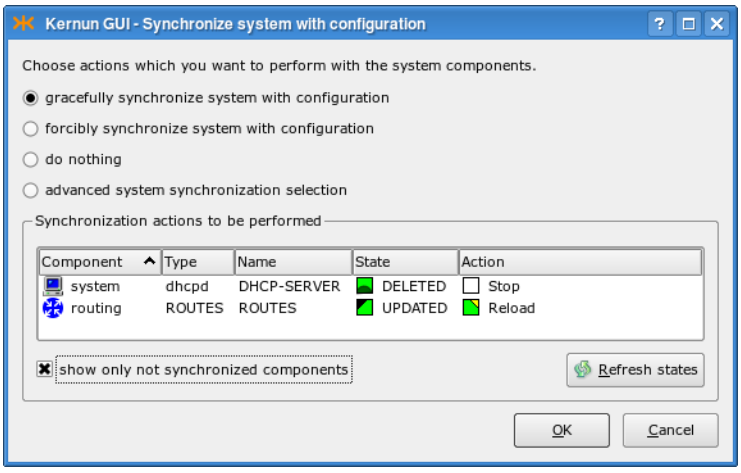

Figure 3.5: System state synchronization dialog

The Process List page (Figure 3.6) contains the list of the running *parent proxy processes*[2].

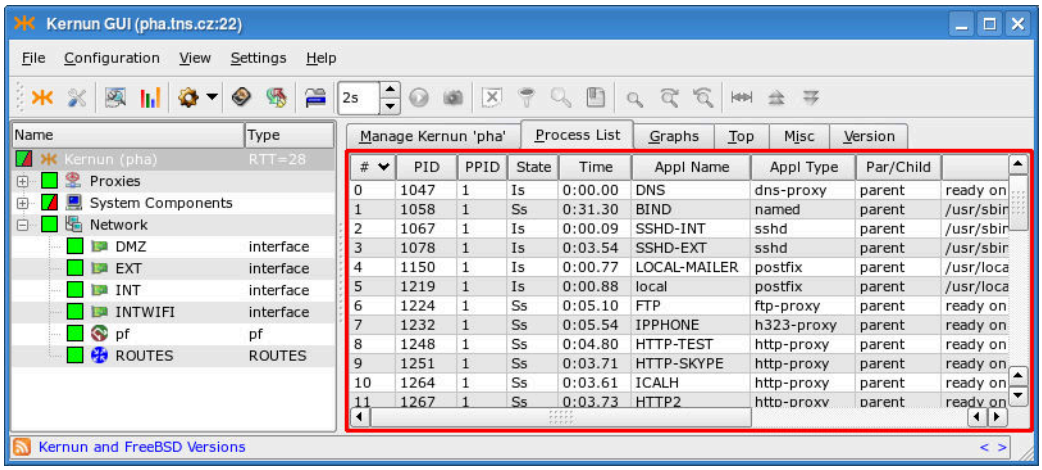

Figure 3.6: Process List page

The context menu in the process list (Figure 3.7) can be used to send the `TERM` (Kill parent process) or `KILL` (Kill -9 parent process) signals to the particular process, to copy the contents of the process list to the clipboard or to save them to a file.

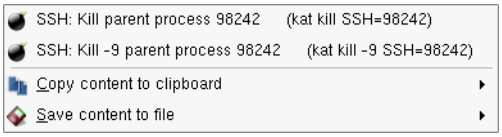

Figure 3.7: Process List context menu

The Graphs page contains graphs of various system parameters, see Figure 3.8. There are many monitored parameters of the operating system (CPU load, used memory and disk space, etc.), hardware (temperature measurement, if supported by the hardware), and cluster behavior (switches between the cluster master and backup). Kernun UTM collects parameter values and creates graphs depicting how they evolve in time, with several time scales available. The most detailed graphs show only the recent history, while coarse-grained graphs extend further into the past. Right-click on a graph to open a context menu that makes it possible i.a. to save the graph to a file or to add it to Favorite graphs.

The Top page (Figure 3.9) shows the output of the popular **top**(1) command. The Misc page (Figure 3.10) displays the output of several commands, showing i.a. the disk space (the **df -hi** command), the network state (**netstat**) or the uptime and current load of Kernun UTM (**w | head -n 1**). The Version page (Figure 3.11) shows the version of Kernun UTM and of the FreeBSD system used by Kernun UTM.

[2] Usually, there are many simultaneous requests on each Kernun UTM proxy, which need to be handled simultaneously. To do that, for every proxy there is one *parent process*, which only manages its child processes (starts and kills them), while these child processes take care of the traffic. Therefore, the number of the child processes depends on the current traffic.

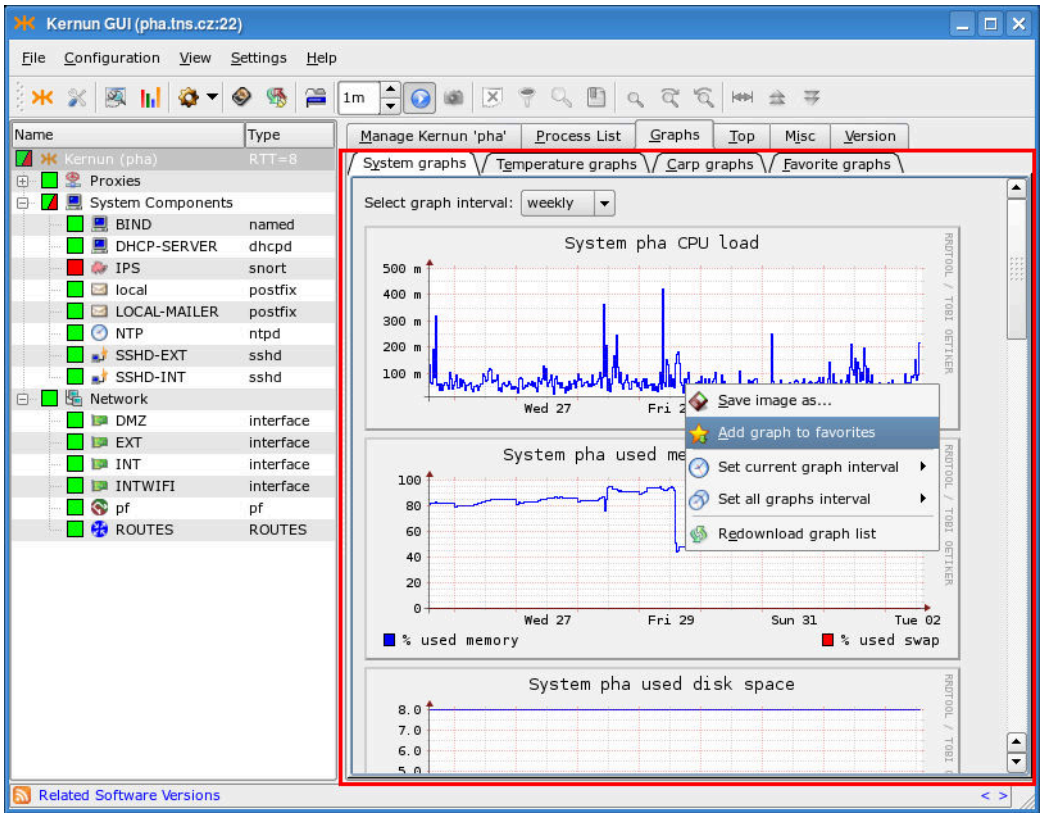

Figure 3.8: Graphs page

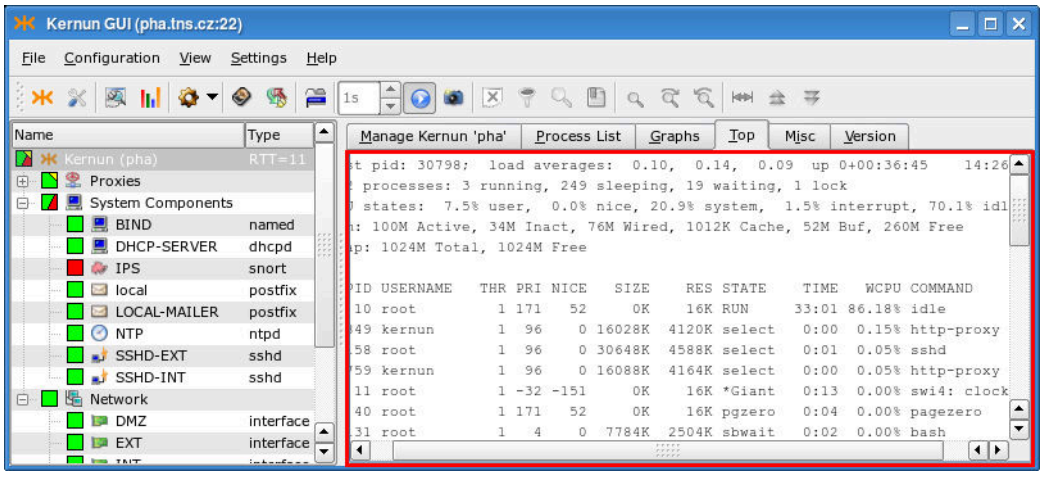

Figure 3.9: Top page

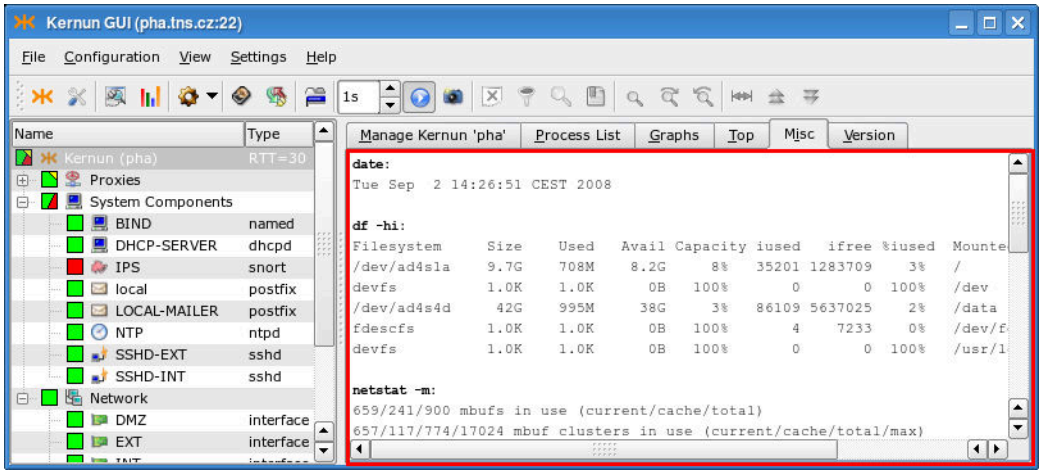

Figure 3.10: Misc page

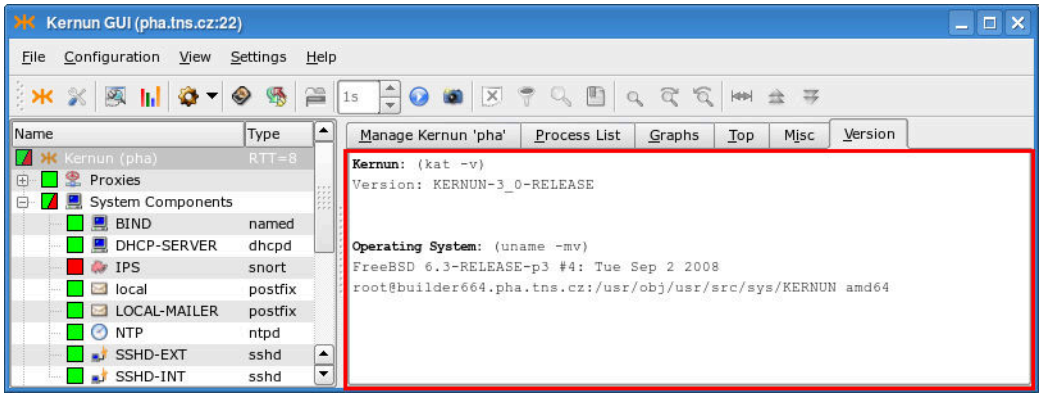

Figure 3.11: Version

Proxy Details

When the Proxies node or a particular proxy is selected in the component tree, it is possible to control and monitor all proxies or the single proxy, respectively. The set of pages is similar in both cases, but the Process List and Graphs pages are available only if a single proxy is selected.

The Manage page for all proxies is similar to the Manage page of the whole Kernun UTM (Figure 3.4), but the Synchronize system button and the combo box used to restrict operations to the tagged components are missing. For individual proxies, there are three additional buttons. Kill terminates the proxy like Stop, but works also for proxies that are removed from the configuration. The SIGUSR1 and SIGUSR2 buttons change the log level. For more information about log levels, see logging(7).

The Process List page contains the list of running processes, restricted to the processes that belong to the particular proxy; both parent and child processes are listed. The functionality is identical to the global process list (Figure 3.6), including the possibility to send signals to processes.

The Monitor page, see Figure 3.12, shows the current active sessions of all proxies, or of the selected proxy. Its content is available only for proxies, for which monitoring(5) is enabled in the configuration. Proxies can be configured to generate the monitoring information into different directories, so it is possible to select the directory, from which the GUI will read the monitoring data.

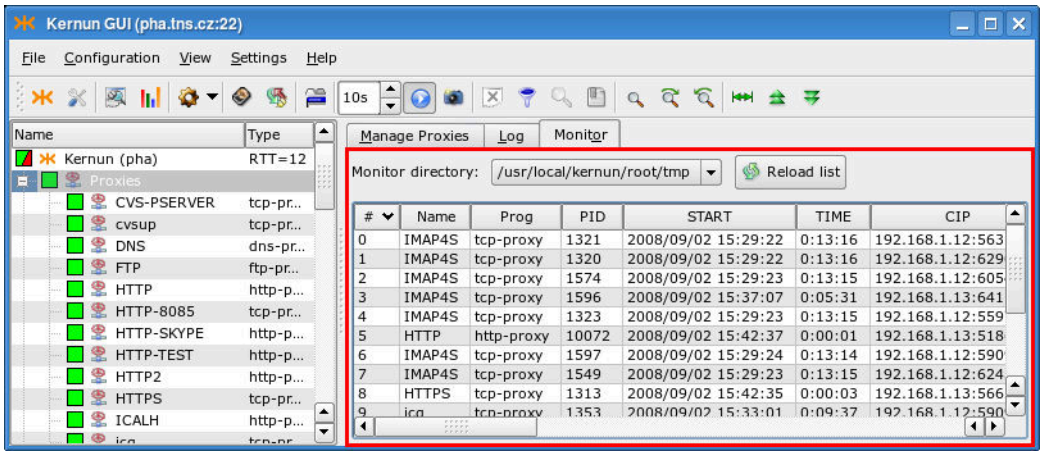

Figure 3.12: Proxy Monitor

The Graphs page's functionality is similar to the system-level one shown in Figure 3.8, but different graphs are displayed: the number of proxy child processes and the number of bytes transferred by the proxy.

The Log page shows the current log messages of the selected proxy, or of all proxies. New messages are added immediately, so the log view provides information about the current proxy activity. Select Help to *MsgId* from the context menu in the log view to open the Details window with a reference page containing information about the selected message. Click on the Row Detail tab in the Details window to display the message as a whole, rather than split into fields. Figure 3.13 shows the log page with the Details window displayed.

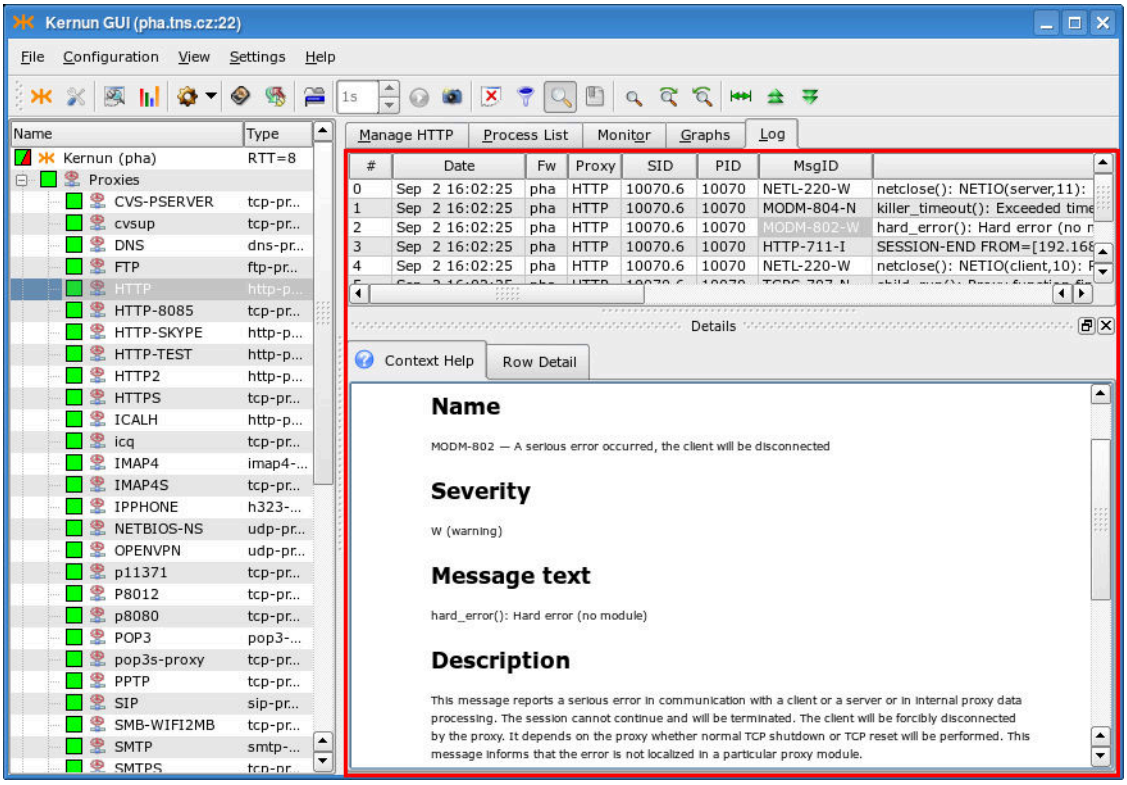

Figure 3.13: Proxy Log

Network Interface Details

The Network node in the component tree groups individual network interfaces, the routing table, and the packet filter[3].

All nodes in the Network subtree share the Manage page containing buttons for starting/stopping/restarting/reloading a component, similar to that of the whole Kernun UTM, as depicted in Figure 3.4, but without the Synchronize system button and the component tag selection combo box. Other pages differ for various types of nodes in the Network component subtree. For the Network node, individual network interface nodes, and routing there are pages containing output of the **ifconfig**(8) and **netstat**(8) commands with various parameters. The Network node provides an additional page sockstat -4 with the list of all existing IPv4 sockets. Each network interface has a Graphs page with traffic statistics graphs in bits and packets per second. Otherwise, the page behaves like its system-level counterpart (Figure 3.8). The current configuration and state table of the packet filter node can be displayed using pages containing the output of the **pfctl**(8) command.

Toolbar and Menu

The toolbar and the menu of the main GKAT window contain almost identical sets of actions. They are adapting to the currently displayed components (page) of the application, i.e., only the useful (and usable) buttons or menu items are enabled.

Figure 3.14: Main window toolbar buttons, listed together with the corresponding menu items, and sometimes keyboard shortcuts:

1. Help | About Kernun GUI — Shows the about dialog box with information about the GUI version.

2. Settings | Preferences — Opens the basic preferences dialog, in which you can set the default behavior of your GUI (e.g., your default **diff** program).

3. File | Analyze Log <Ctrl+L> — Opens a form, in which you can choose certain restrictions of the log information to be analyzed (date, time, proxy, etc.), and then downloads the log from Kernun UTM and displays it in a new window (for more information on log analyzing see Section 3.1.3).

4. File | Statistics — Opens a proxy statistics browser window. If configured, daily, weekly and monthly statistics of traffic handled by proxies are computed from the log. The browser window displays a hierarchy of available statistics ordered first by the frequency (weekly/monthly/daily), then chronologically, and finally by the proxy name. After

[3] Kernun UTM uses the PF packet filter (see **pfctl**(8) and `pf.conf`(5)), which provides stateless and stateful packet filtration, NAT, and traffic shaping.

choosing an item from the hierarchy, the corresponding statistical output (in the form of a HTML document with images containing graphs) is displayed.

5. The Configuration submenu:

   - Edit Kernun configuration — Downloads the Kernun UTM configuration and opens a window, in which you can edit it (see Section 3.1.4).
   - Configure Kernun from the remote example — Lets you choose and edit a sample Kernun UTM configuration that you can later use as your Kernun UTM configuration.
   - Import the configuration from a local file — Opens a local configuration file for editing. You can later upload it to your Kernun UTM to use it as its configuration or save it back to your computer.
   - Recent versions of the configuration file — Opens an RCS Browser displaying all the changes you have made in the Kernun UTM configuration and making it possible to switch back to any of the previous versions.

6. The Console submenu:

   - Kernun console — Opens the console with an ssh connection to Kernun UTM.
   - Add custom remote command — Opens a dialog, in which you can create custom remote commands that can be executed in the console. When created, they are added to the Console menu and can be easily invoked.

7. File | Kernun Systems Management — Provides routines to install, backup, and upgrade Kernun UTM installations. See Section 2.5, Section 2.6, and Section 2.7 for details. The system manager has also the License page, accessible also from Settings | License. It displays the current license file and provides a button for uploading a license file to Kernun UTM.

8. File | Reconnect — Restarts the connection from the GUI to Kernun UTM. It may be useful if your *RTT* gets too high or the connection is interrupted.

9. View | Tear off the tab `<Ctrl+T>` — Opens the current page in a new window and keeps it up-to-date (refreshes the selected page).

10. *Refresh Frequency* — The period, with which new data is downloaded from Kernun UTM. The lower is the period, the more traffic load Kernun UTM produces towards the GUI.

11. *Start/Stop* — Starts and stops refreshing (downloading of new information from Kernun UTM) of the current page.

12. View | Tear off a snapshot — Makes a static copy of the current page and opens it in a new window. This window is never refreshed. It can be used to analyze the snapshotted situation (e.g., the traffic log).

13. View | Clear the contents — Clears the contents of the current page.

14. View | Show/Hide Filters&Markers Window <Ctrl+M> — Shows or hides the Filters and Markers window, which is used to specify filtering and marking for the displayed log (see Section 3.1.3).

15. View | Show/Hide Detail Window <Ctrl+D> — Shows or hides the details window, which displays the details of the current row (e.g., in a log) and the reference page for the current row type.

16. View | Show/Hide Bookmarks Window <Ctrl+B> — Shows or hides the bookmarks window. Bookmarks are used to assign a name to a certain line of a snapshotted situation. You can then easily jump from one bookmarked line to another.

17. View | Find <Ctrl+F> — Opens a dialog used to search for selected text in a selected column. The view highlights all the found occurrences and jumps to the first one.

18. View | Find next marked line <Ctrl+G> — Jumps to the next line marked by the chosen marker.

19. View | Find previous marked line <Ctrl+Shift+G> — Jumps to the previous line marked by the chosen marker.

20. View | Line-Up Columns <Ctrl+I> — Adjusts the column widths according to the widths of their contents.

21. View | Jump to the beginning <Ctrl+Up> — Scrolls to the beginning of the current page.

22. Keep at end <Ctrl+Down> — Scrolls to the end of the current page and keeps scrolling so that the newly added lines are displayed. This function is most useful with the Log page, if we want to display newly added messages continuously.

3.1.3 Logs

The Kernun GUI contains a powerful log analyzer. It provides tools for filtering out undesired log entries and for marking those that are more important to you with different colors. The log window (possibly displayed as a part of another window) is displayed in several modes: you can either open a local log file, download a "historical" log file from Kernun UTM, take a snapshot of the currently displayed log, or work with the up-to-date (and changing) log.

The first setting concerning logs is the specification of the log level for the particular proxy (permanently in the configuration or temporarily using buttons on the Manage page). There are seven different log levels, as described in logging(7). The higher the level is, the more information the Kernun UTM stores (and sends to your log analysis window).

Filter and Marker Basics

Figure 3.15 shows an example of a log display window (created by selecting File | Analyze Log in the GKAT main menu). The logs are presented as tabular data. You can drag a column by its title and drop it elsewhere to change the column order. Each column has a context menu used to select which columns are to be displayed. For example, if you are analyzing a log from a

particular proxy in a single Kernun UTM network, you may want not to display the columns Fw (Kernun UTM name), Proxy and PID (as the PID is also included in the column called SID, which is a concatenation of the PID and the number of the current *session*[4] served by this process). You can use the Save column visibility settings option, so that you do not have to set it every time.

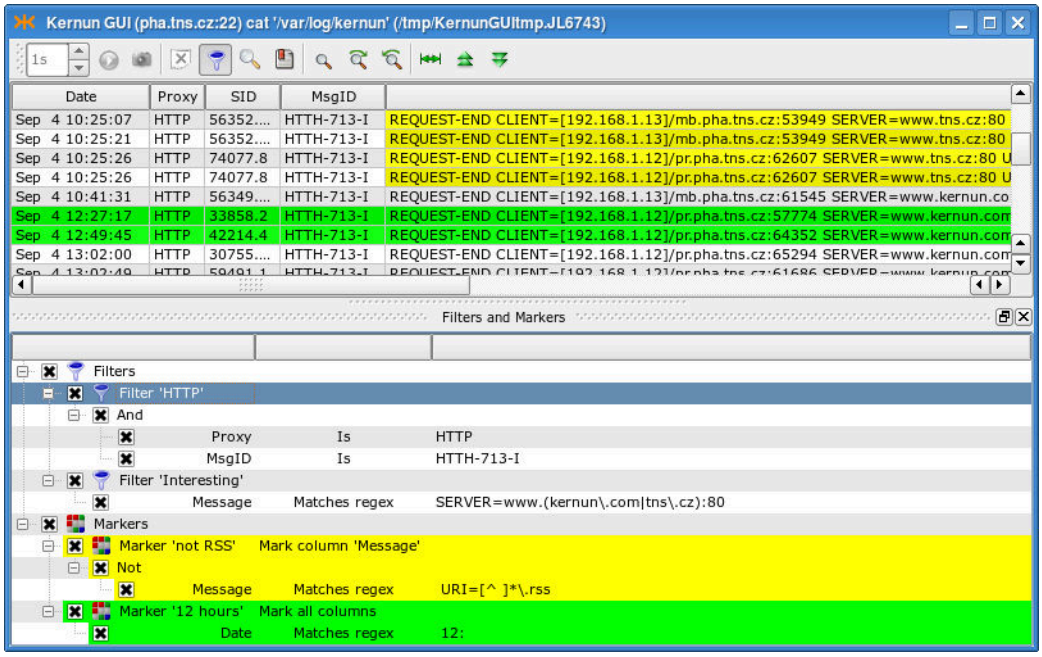

Figure 3.15: Example of a simple HTTP filter and marker set

The filters and markers are accessible using View | Show/Hide Filters&Markers Window (<Cltr+M>). Filters are used to display only the log entries that you are interested in, while the main point of markers is to highlight some special rows (columns) in the log. A selected marker can be then easily used to jump to the next/previous marked line.

Note

Both filters and markers are managed (created, deleted, moved, changed) in a similar way. From now on we will only describe management of filters. However, all of the operations work in the marker context, too.

The only difference between the management of filters and markers is that markers are applied automatically after a change, whereas filters need to be explicitly applied by selecting the Apply filter option from the context menu of the Filters node, or by clicking on the same button, which appears blinking after any change in the filter set. Moreover, the context menu of a marker contains an option used to change the highlighting color and highlighted column(s).

[4] A *session* encompasses the communication of a particular proxy process with a single client.

Important

If you work with the current live log, the filters only apply to the *newly incoming log messages*. The old ones are not filtered out even if you apply the filter. To filter out older log entries, tear off a snapshot. Markers take effect instantly in all situations.

Creating Filter Conditions

There can be many filters in the system. They are all presented in the filter tree under the Filters node. Some of the filters may be turned off (by unchecking the box next to their name). Each filter is a tree of conditions with logical operators And, Or and Not. Each condition consists of a column to be examined, a relational operator and a desired value. The relational operator defines the relation between the value in the column and the desired value that must be fulfilled in order for the row to be included in the log. The following relational operators are available: Is, Contains, Matches regex and their negations, Starts with and Ends with; they are all self-explanatory.

The easiest way to create a filter is to drag and drop a field from the log table. A new filter is created and a new condition inside this filter is initialized with the value of the field you dropped there. You can also drag a field and drop it on the relational operator (then the new condition is added as a child of the operator) or on another condition (in this case, you are asked whether you want to use And or Or to connect it with the target condition or whether you want to add the new condition to the same parent). Other useful options (including creation of a new filter) are listed in the context menu of the selected node of the filter tree.

Having created some filters, you can copy a particular node (and all its subnodes) to another place simply by dragging it there. In this way you can easily create a filter from an existing marker, and vice versa.

Saving and Loading Filters and Markers

You may want to save filters you create for future use. You can save a single filter or any subset of filters. Both saving and loading of filters is done using the context menu of the chosen filter (or the Filters node). A filter can be saved either to a local file, or to the registry. The filters (and markers, of course) in the registry are displayed in (and can be loaded from) the Load filters/markers submenu of the top-level Filters/Markers context menu.

Tip

To rename a filter/marker, double-click on its name or select Rename from the context menu.

3.1.4 GCML — Configuration

This section describes the GUI functionality for examination and modification of the Kernun UTM configuration. It does not describe the configuration language. See Section 4.1 for an introduction to the configuration syntax and semantics, and Section 4.2 for a simple configuration example.

Kernun GUI provides a GUI editor for the Kernun UTM configuration files (*kernun.cml*). The GUI CML editor is functionally equivalent to the line-oriented **cml**(8) tool. The GUI CML editor is aware of the CML syntax and semantics and highlights points of the configuration that are detected as erroneous. When connected to Kernun UTM, the Kernun GUI can automatically download the configuration from Kernun UTM, present it in the graphical CML editor, and commit it back to Kernun UTM. The GUI CML editor can also open a local file; in this case, a connection to Kernun UTM is not required.

Configuration Window Overview

Figure 3.16 shows the main GUI configuration window. The configuration is displayed in an expandable configuration tree. You can select a node to display and easily edit its details in the right-hand part of the window. The displayed details include the selected node's type and name, its full path in the configuration (as displayed by **cml**(8) in the command line prompt), a brief description, and the constraints (if there are any). There is also a form with fields (editable, if appropriate) that represent the elements of the selected node. A short description, the name of the element, and the expected data type are displayed to the right of each field. The labels of the required fields are highlighted (bold). Several field types help you fill in data of the correct data type. If you make a mistake (e.g., do not fill in all the required information or do not respect the expected data type) the background of the incorrect field is highlighted, an exclamation mark button appears next to it (click on the button to show the error message) and the error is also displayed in the configuration tree.

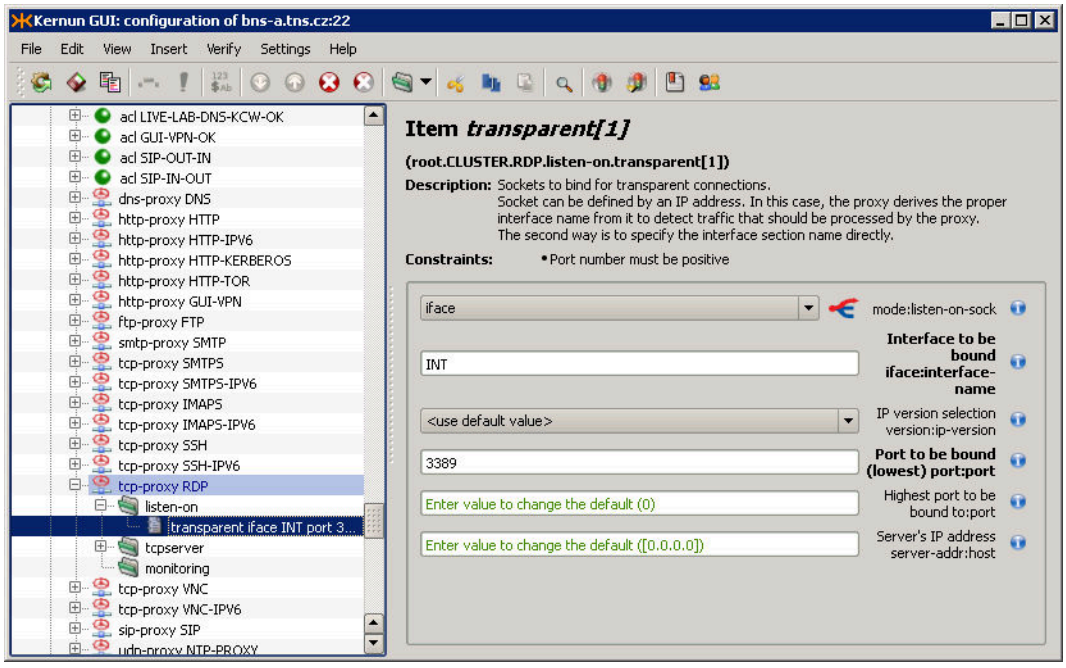

Figure 3.16: GUI CML Editor

When editing the configuration, select File | Show differences in the main menu to display the differences between the current contents of the configuration and the original configuration file

downloaded from Kernun UTM. The currently edited configuration can be saved to a local file using the File | Save the configuration as a local file menu item. In addition to online marking of incorrect configuration nodes, it is possible to invoke verification of the whole configuration using Verify | Verify the configuration.

When you finish editing of the configuration, you may want to save and apply it to Kernun UTM or save it as a local file. Both can be easily done using the File menu. Committing of the configuration (<Ctrl+S>) invokes a dialog, depicted in Figure 3.17, where you can choose actions to be done with the configuration on Kernun UTM (save/generate/apply/synchronize state, all done by default) and the configuration to apply. The results of the action are displayed in a text box. If you have made any changes to the configuration, you are asked to fill in the *RCS Log Message* describing the changes. First, the configuration file is saved and stored in a RCS file, in which the complete history of changes is kept and from which any old version of the configuration file can be retrieved. Then, low-level system and proxy configuration files are generated. The application of the configuration means copying the low-level configuration files to locations, in which Kernun UTM components look for them at the startup. Finally, the system state is synchronized with the configuration. This is the same action as the one performed by Synchronize system, including the possibility of reviewing and modifying the list of performed actions in the dialog window shown in Figure 3.5.

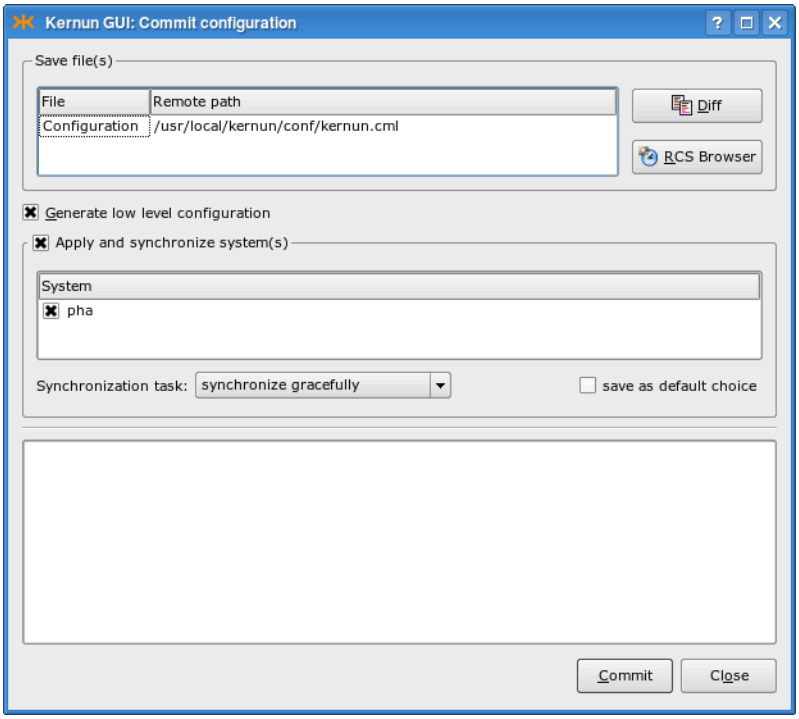

Figure 3.17: Configuration commit dialog

Interaction with the Configuration Tree

Each line of the configuration tree represents a node of the Kernun UTM configuration and has an icon according to its type. You can select a node by clicking on its name. The easiest way to move in the tree is using the cursor keys. <Up>/<Down> selects the next/previous node, <Right> expands the current section, <Left> jumps to the parent of the current node (or collapses an expanded section). A non-empty section node can also be expanded/collapsed by clicking on the +/- sign next to it. Useful commands for interaction with the configuration tree are available in the Edit menu, the context menu of the selected node, and on the toolbar. A section can be expanded, along with all of its subsections (and their subsections, etc.) by selecting Expand current node recursively <Ctrl++> (and collapsed using the Collapse ... <Ctrl+-> command). Because the order of individual sections and items matters, you can move a node up and down (<Ctrl+Up>/<Ctrl+Down>) using commands from the same menu. You can also Remove (<Ctrl+D>), Hide (<Ctrl+H>), Copy (<Ctrl+C>), Cut (<Ctrl+X>) or Paste (<Ctrl+V>, pastes the last copied/cut node under the selected one) a node.

> **Tip**
>
> If you are more familiar with the line-oriented **cml**(8) tool, you may want to see the text version of the configuration of the current node (section). This is accessible from the View | View expanded configuration menu or from the context menu of the node. A read-only window containing the output of the **/show -a** cml command is displayed.

Inserting a new node using the Insert menu is relatively easy with the GUI. First, you need to decide where you want to add the new node (as the first child, as the last child (<Ins>), or next to the current node (<Ctrl+I>)). Then you need to choose the type of the new node (section, item, data variable definition, section variable definition, section variable application, `for` loop, switch/case, include file, or comment). In the case of inserting a section, item or section variable, you need to choose the type of the section/item and its name (if appropriate). Finally, the new node is inserted and you can edit its properties in the details window.

Form Field Types

Several form field types are used to represent the properties of a node. It is important to understand each of them, in order to fill in correct data.

- *Text box* — Expects text data. However, you cannot fill in anything; you still need to preserve the data type specified in the parentheses in the field label (e.g., the *sock* type means `[`*ip*`]:`*port*). The syntax of values of various data types used in the Kernun UTM configuration is described in Section 4.1.
- *Check box* — Can either be checked (true), or unchecked (false).
- *Combo box* (selection) — You can choose exactly one of the specified options. If you want to enter its content manually, you need to use the Edit | Toggle combobox/explicit value menu

item, which allows you to write there "anything" (e.g., a variable name). If there is a "fork" icon next to the combo box, the value of the combo box decides which further form fields are displayed (e.g., in `system.smtp-proxy.delivery-acl.sender` the *error* value requires further constraints on the error, whereas the *ok* option does not).

- *List of values* — Some CML properties (e.g.`system.acl.service`) expect a list of values of a certain type (str, addr...). You need to be able to create new values, as well as change and delete the old ones. The value list field type displays each value of the list in a separate row. You can simply edit a particular value in the same way as you are used to work with the above-mentioned field types. A value can be deleted from the list using the Delete icon next to it. To insert a new value, click the Append Value button. The value lists can be multidimensional (any value list can contain sublists, which can also contain sublists, etc.). You can append a new sublist by clicking the Append Sublist button. On the right from a sublist there is a button with a submenu that allows to Remove the sublist or Exclude the sublist (creates the negation of the current sublist). List items can also be loaded from a file, in which each line represents one item of the list. To "inject" a file into a list, click on the Append File button and select the `shared-file` section name, in which the file name and the format (e.g., IP addresses, text or REGEXPs) are specified.

> **Tip**
>
> The contents of the `shared-file` can be edited in the GUI in the details of the `path` item of the `shared-file` section.

Bookmarks

As orientation in a very big configuration may become difficult, the Kernun GUI provides tools to simplify it.

First of all, you can use View | Show/Hide Bookmarks Window (`<Ctrl+B>`) to create bookmarks (like in your Web browser) to important positions in the configuration tree. The bookmark window is displayed in the top right part of Figure 3.18. To add a bookmark to the current configuration node, select the Bookmark this node option from its context menu or the Bookmark current position option from the context menu of the bookmark list in the Bookmarks window. You are asked to fill in the bookmark's title before adding it. You can jump to a bookmarked node by double-clicking the bookmark's name in the Bookmarks window. Additional bookmark operations (removing, renaming) are available from the context menu of individual bookmarks.

Relevant Nodes

In addition to simple bookmarks, the Kernun GUI provides smart bookmarks, accessible via View | Show/Hide Relevants window (`<Ctrl+R>`). Smart bookmarks are used to highlight the configuration nodes that are relevant to the chosen nodes (e.g., the ACLs for proxies) and to easily jump to them. The function of finding and highlighting of the relevant nodes is only available for proxies (it would not make much sense for other nodes), but the relevant nodes may include any

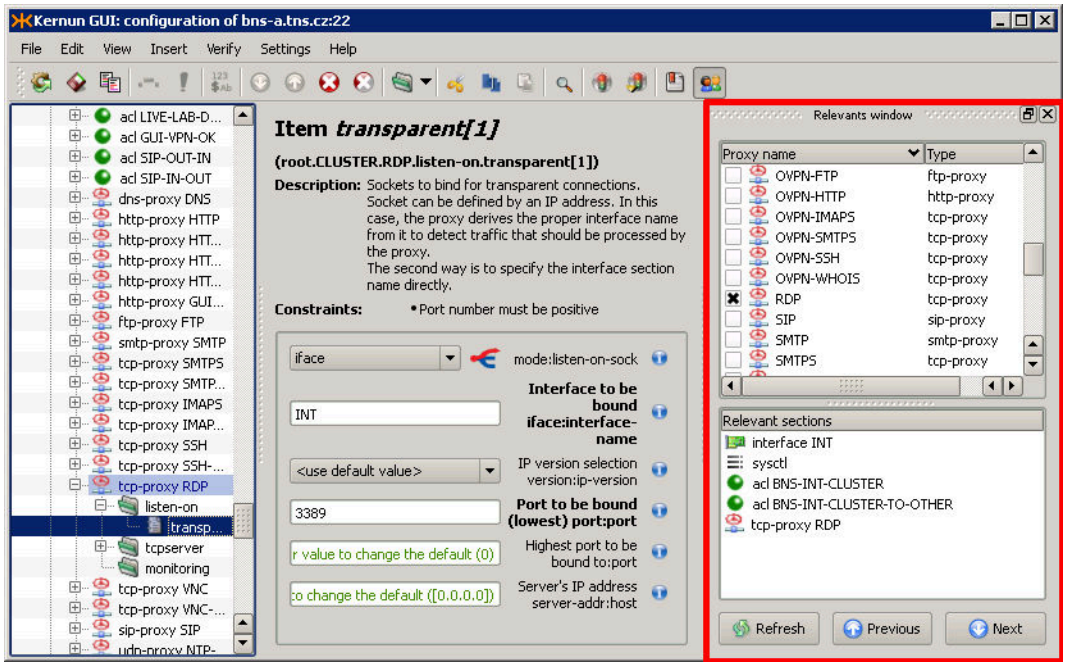

Figure 3.18: Bookmarks and relevant sections in GUI CML

sections (proxies, ACLs or other sections). A node is relevant to a proxy if a change to the node may alter the behavior of the proxy.

There are two lists of bookmarks in the Relevants window. The first one (Proxies) displays the list of all proxies in the configuration and makes it possible to select the proxies, for which the relevant nodes are to be highlighted. The other list (Relevant sections) displays bookmarks to configuration nodes that are relevant to the chosen proxies. Again, you can double-click a bookmark to jump to it in the configuration tree. After selecting the proxies in the Proxies list, the relevant nodes are highlighted in the configuration tree and bookmarks to them are displayed in the relevant sections list.

There are more actions available from the context menus of the items of the bookmarks lists, including Bookmark this... (which adds the current item to the simple bookmarks), Jump to node (works like double-clicking on the item), and several default selection templates for the proxies. Furthermore, you can choose the types of relevant nodes that you are interested in for the Relevant section items.

Tip

You might not want the relevant item list to be to refreshed automatically (after every change in the configuration and in the selection of the proxies), especially on a slower machine. This behavior can be controlled from the context menu of any of the relevant items using the Automatically refresh relevants check box in its context menu. If it is unchecked, you have to refresh the list manually by clicking the Refresh button.

Configuration Wizards

The Kernun GUI provides configuration wizards in order to simplify the configuration of complex tools (OpenVPN, IPSec). All the wizards are accessible via the Insert | Configuration wizard menu item. Select Overview of the wizards to open a dialog that describes all the wizards and the use of the created part of the configuration, and makes it possible to start the selected wizard.

Each wizard is divided into several pages that prompt the user to fill in individual fields. The form fields are the same as in the GCML, so you can use references to other parts of configuration or data variables. The fields highlighted with orange background are filled incorrectly and you need to correct their values before you are allowed to proceed. The last page of each wizard displays the part of the configuration that has been created using the wizard and the result of verification of the configuration after the changes are committed into the configuration. If you do not like the values, you can return back in the wizard, correct the respective fields and proceed to the wizard overview again. When you finish the wizard, the created configuration is committed to the main system configuration. You can cancel the wizard at any moment to undo all the changes and return back to the GCML.

The Help button located on each wizard page opens a help window next to the current page that describes all the fields on it. The contents of the help window always correspond to the current page.

> **Tip**
>
> All that can be created using wizards can be done also manually using the standard GCML interface. Another option is to create only the core part of the configuration using the wizard and then modify it using the GCML.

There are currently five wizards in the Kernun GUI. They are described in more detail in the corresponding sections of the handbook.

- Section 4.3.1 — enables connection from clients in the internal network to server(s) in the external network (e.g., access to HTTPS servers).
- Section 4.3.1 — enables access of external clients to servers in the local network (e.g., a proxy for local IMAP4S mail servers allowing employees to download their mail from home).
- *OpenVPN Remote Access Server* — configures an OpenVPN server that will accept connections from clients in the external network (road warriors).
- *OpenVPN Network to Network* — configures a local OpenVPN peer, which will connect local and remote networks (behind another OpenVPN peer).
- *IPSec* — configures IPSec in the tunnel or transport mode to secure the communication between the specified local and remote networks.

3.1.5 Locking

The Kernun GUI implements a configuration locking mechanism, which prevents changes made by different users from being accidentally overwritten. The first user to open the CML configuration

in edit mode or the System Manager acquires the configuration lock. Any other user trying to open the configuration at the same time will not be able to acquire the lock. Hence, the user will be presented with the Figure 3.19 dialog presenting the available options. The choice to break the lock must be confirmed explicitly in Figure 3.20 as a potentially dangerous operation.

The Kernun GUI also prevents the user from modifying the configuration in two different places at once. Therefore opening the System Manager while editing the configuration results in Figure 3.21.

The Kernun GUI unlocks the configuration automatically when the user closes the window, which has acquired the lock. Possible errors are reported (Figure 3.22).

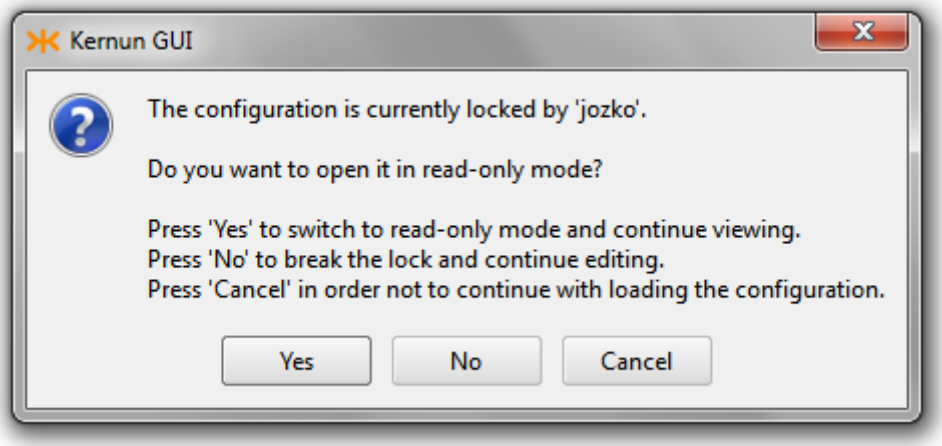

Figure 3.19: Configuration already locked by other user

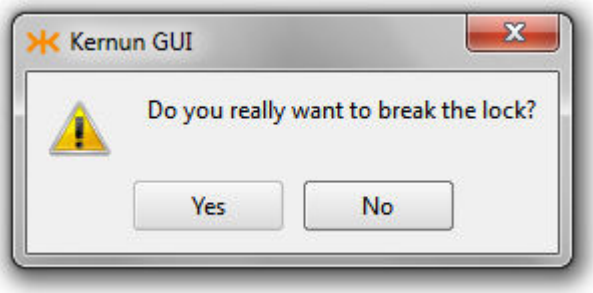

Figure 3.20: Break lock confirmation

After the locking phase, the CML configuration editor checks whether the actual contents of the configuration file have been stored in the revision control system (RCS). If not, the user will be asked whether to do so (Figure 3.23). The configuration file or the RCS version file may not even be initialized (Figure 3.24).

Before commiting the configuration to Kernun, the CML configuration editor checks whether it still owns the lock. If not, it may have either been broken and still locked (Figure 3.25) or broken and then unlocked (Figure 3.26).

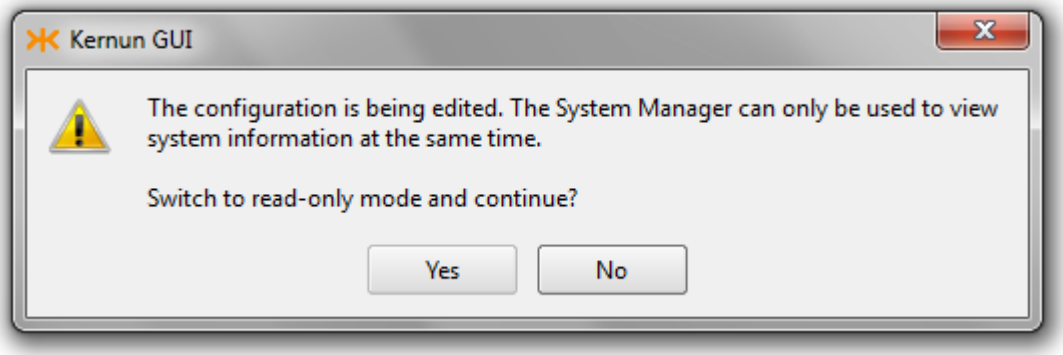

Figure 3.21: Configuration and System Manager at the same time

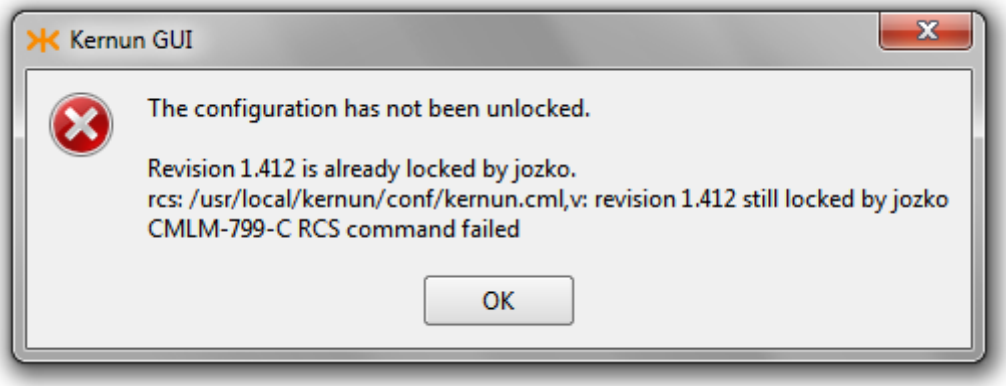

Figure 3.22: Configuration unlock failed

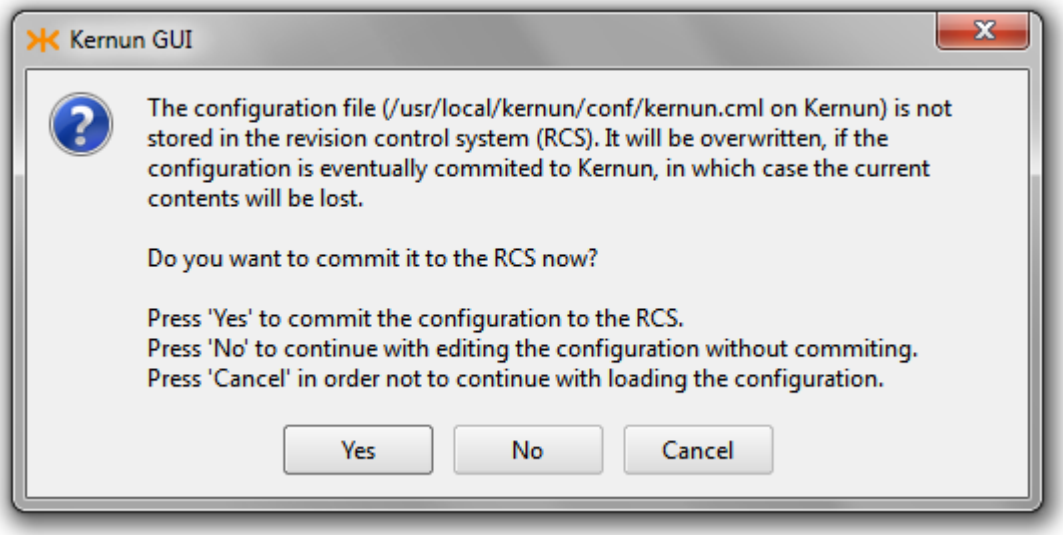

Figure 3.23: Commit configuration to RCS confirmation

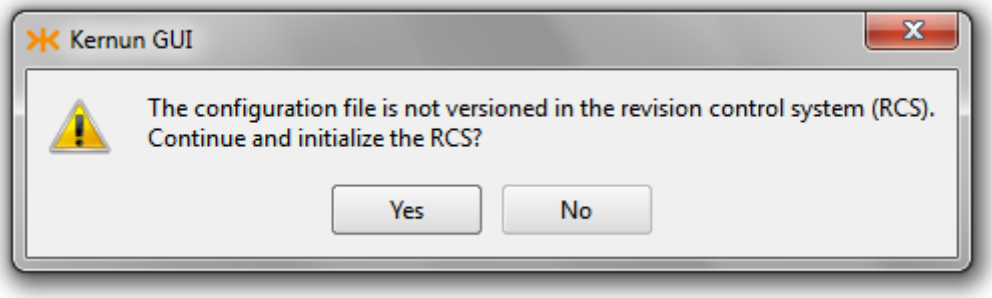

Figure 3.24: Initialize RCS confirmation

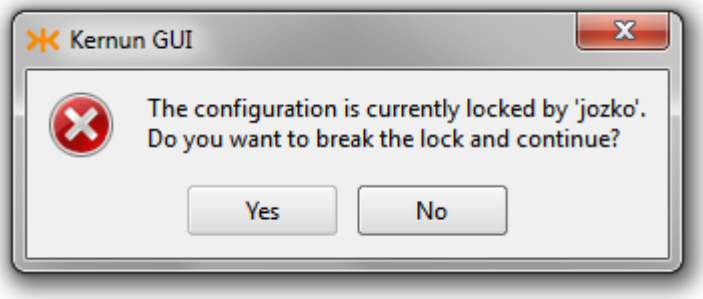

Figure 3.25: Configuration lock broken

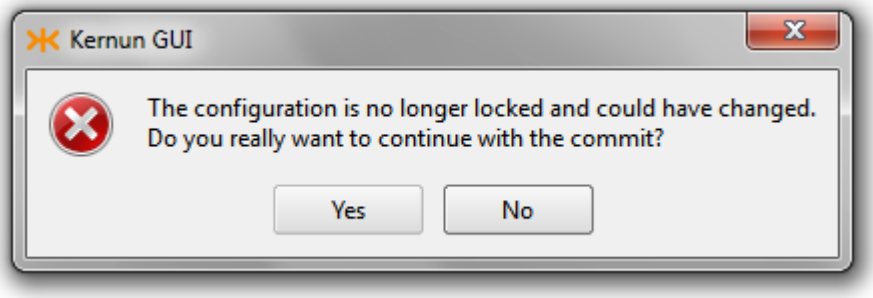

Figure 3.26: Configuration not longer locked

3.2 Command Line Interface

Besides the GUI (Section 3.1) there is another approach to Kernun UTM services, namely the text-only, or command-line interface (CLI). Its common features as well as its particular tools are described further in this section.

3.2.1 Command Line Interface Details

All command-line tools can be used in the interactive mode, which means that they display prompts and expect commands entered by the user. The user can use common features of the command-line interface, known e.g. from the **bash**(1) tool, namely:

History browsing The user can browse through the list of commands entered recently using the cursor keys, change the command text and re-enter it by pressing the `<Enter>` key.

History searching Using the `<Ctrl+R>` key combination, the user can invoke the command history searching. When s/he starts typing a part of a command, the CLI displays the most recent command containing this string. When a cursor key is pressed, the search is stopped and the selected command is copied to the command line for further processing.

Hotkey binding Most of hotkey bindings known from **bash**(1) are available in Kernun UTM's CLI tools. For example, `<Ctrl+A>` skips to the beginning of the line, `<Ctrl+E>` skips to the end of the line, `<Ctrl+U>` clears the line, `<Ctrl+D>` exits the tool etc.

```
KAT@fw> apply fw
CMLK-720-E Directory '/usr/local/kernun/conf' contains no SYSTEM
KAT@fw> cml -g
RCS file: /usr/local/kernun/conf/kernun.cml,v
...
CKGB-719-N ---- System 'fw' successfully generated
KAT@fw> <^R>
Searching for: 'ap' > apply fw
```

Warning

The command history is saved when a CLI tool is closed and made available again the next time it is started. However, there is a bug in the library used for the line-editing support and in some cases, the reading of the history file fails and the tool coredumps. The recovery is very simple: remove the history file (`~/.kat_hist` for the KAT and `~/.cml_hist` for the CML) and restart the tool.

3.2.2 C^{3}H — Command Completion and Context Help

One of the most important and most useful features implemented in Kernun UTM's CLI tools is Command Completion and Context Help, or shortly *C^{3}H*.

The C^{3}H support helps you write correct commands or proper parameters more quickly. The rule of thumb is: "If you don't know what to do now, press the <Tab> key!". Of course, it does not work absolutely perfectly in all situations, but it works e.g. when selecting a command name, a command argument name (if there is only a defined set of ones), a system name (the **apply** command in the KAT), a proxy name (application or process management in the KAT, language reference in the CML), a file name (some KAT or CML commands, a file name reference in the CML), a correct operator or delimiter (in the CML language), and many others.

```
KAT@fw> <Tab>
  apply                         # apply generated system config files
  cml                           # start configuration tool
  ...
KAT@fw> a<Tab>
KAT@fw> apply
KAT@fw> apply <Tab>
KAT@fw> apply fw
CMLK-820-N [root] Applying 'SYSTEM-fw', Source: kernun.cml,v 1.1 ...
...
System /usr/local/kernun/conf/SYSTEM-fw applied to /.
KAT@fw> start s<Tab>
SMTP        SSH         SSHD
KAT@fw> start Sm<Tab>
KAT@fw> start SMTP
CMLK-841-N [root] Executing 'start' operation of N=SMTP A= T=
...
CMLK-849-N [root] Successful end of 'start' operation
KAT@fw> ps -<Tab>
-a         -b         -d         -p         -S         -t         -T
KAT@fw> ps -p <Tab>
KAT@fw> ps -p smtp-proxy
```

3.2.3 KAT — Kernun UTM Admin Tool

The KAT utility is the core management facility for administration of Kernun UTM. It is available directly on Kernun UTM when used locally, can be used when accessing Kernun UTM remotely in the text mode via SSH, and, in fact, most of operations implemented in the GUI (Section 3.1) are realized using the KAT on the remote machine.

The complete command summary is available in **kat**(8); this section points out only the most important features and shows the typical tasks an administrator can do using the KAT tool.

Modes of Operation

The KAT can be used in two modes:

Interactive mode In this mode, the program is called without parameters. It displays its own prompt `KAT@`*`hostname`*`>` and expects commands issued by the user. In this mode, the KAT accepts also ordinary UNIX shell commands, so it can be used as a simplified UNIX shell with added commands for Kernun UTM management.

Batch mode In this mode, the program is called with a KAT command as a parameter; the command is executed and the tool exits. This mode is used for quick execution of a single KAT command without the need of starting the KAT explicitly.

```
[root@fw ~]# kat ps
12345     1 Ss     0:00.01 SMTP: parent: ready on 1 address: ...
[root@fw ~]#
```

Configuration Management

The management of the Kernun UTM configuration is one of the most important functions provided by the KAT. The configuration tool CML (see Section 3.2.4 below) can be invoked from the KAT simply by typing the command **cml**. After finishing the configuration, the KAT **apply** command must be executed to put the changes into effect.

```
KAT@fw> cml
CMLI-700-N CLI interactive mode entered
RCS file: /usr/local/kernun/conf/kernun.cml,v
...
CMLR-710-K File '/usr/local/kernun/conf/kernun.cml' loaded
CML> ... work interactively with the CML
CML> ./generate
CKGB-719-N ---- System 'fw' successfully generated
CML> ./quit
CMLI-709-N CLI interactive mode closed
KAT@fw> apply fw
CMLK-820-N [root] Applying 'SYSTEM-fw', Source: kernun.cml,v 1.1 ...
```

In the case of a mistake, the administrator can go through the log of previous revisions, compare different versions and revoke an older one. For this purpose, several commands have the `-r` option with a value of revision number, or `0`, `-1`, etc. for previous RCS versions of the configuration file.

```
KAT@fw> rlog
RCS file: kernun.cml,v
Working file: kernun.cml
head: 1.2
...
----------------------------
```

```
revision 1.1
date: ...
Initial revision
----------------------------
=============================================================
KAT@fw> rcsdiff -r 0 -r 1.1
=============================================================
RCS file: kernun.cml,v
retrieving revision 1.2
retrieving revision 1.1
diff -c -r1.2 -r1.1
...
KAT@fw> cml -r -1
CMLK-825-N [root] Revoking revision 1.1 of '/.../kernun.cml'
CMLI-700-N CLI interactive mode entered
```

Component Management

Kernun UTM consist of a set of components, and the KAT tool facilitates the work with them. The commands of this group work with the configuration currently applied in the system. Thus, applications that are not configured cannot be operated (e.g. stopped) using these commands. For such tasks, the process management (see Section 3.2.3 below) commands need to be used. The commands have several options used to select the proper set of components by type, by name etc., and also by the current state of the component's configuration (e.g. restart only the updated ones).

```
KAT@fw> restart
----> Are you sure to do this operation with the whole firewall? (y/[n]) n
KAT@fw> showapp -n
SMTP    smtp-proxy      proxy   -       :       UPDATED 50
CMLK-840-I Found 1 component(s)
KAT@fw> restart -n SMTP
CMLK-841-N [root] Executing 'restart' operation of N=SMTP A= T=
...
CMLK-849-N [root] Successful end of 'restart' operation
KAT@fw> showapp
...
SMTP    smtp-proxy      proxy   -       :       CURRENT 50
...
CMLK-840-I Found 14 component(s)
KAT@fw> restart -n SMTP
CMLK-841-N [root] Executing 'restart' operation of N=ROUTES A= T=
CMLK-740-W Component 'ROUTES' running, skipping...
```

```
CMLK-712-E No appropriate component found
KAT@fw>
```

Process management

The running applications (components) of Kernun UTM can be operated using the commands of this group. The administrator can monitor a list of application processes and send them signals.

```
KAT@fw> ps -b SMTP
12345     1 SMTP: parent: ready on 1 address: [192.168.10.1]:25
KAT@fw> kill -info SMTP *
12345     1 SMTP: parent: ready on 1 address: [192.168.10.1]:25
CMLK-830-N [root] Sending signal 29 to application 'SMTP'(), process: *
CMLK-130-N [root] Sending signal 29 to process 12345
KAT@fw>
```

3.2.4 CML — Configuration Meta Language

The CML tool is a simple syntax-driven editor of the Kernun UTM configuration. The complete command summary is available in **cml**(8); this section points out only the most important features and shows the typical configuration tasks.

The user can move within the configuration in the manner very similar to moving across the file system, with the sections of the configuration playing the role of directories. At every moment, the user is in a node of the configuration tree; the "path" to the node is shown in the prompt. To descend into a subnode, simply type the subnode's header with the opening left brace. To ascend, analogously type the closing right brace.

```
CML> sy<Tab>
CML> system <Tab>
<new name>  fw
CML> system fw {
CML.fw> routes {
CML.fw.routes> }
CML.fw>
```

In the case of a command error or incompleteness, the CML advises the proper continuation in the prompt text.

```
CML.fw> routes
CML.fw.routes> [OPEN] <Tab>
CML.fw.routes> [OPEN] {
CML.fw.routes>
```

Type an existing section to enter it for editing; type a new one to append it after the "cursor". If a new node should be placed elsewhere, the cursor position needs to be changed. Special CML commands are used for such "control" operations. To be distinguishable from the section and item names, they start with a special character sequence ("/" or "./"). The **show** command is used to display the content of a node, the **goto** command to move the cursor.

```
CML.fw> ./show
      [ 1] ## Host name without domain
      [ 2] hostname fw;
      ...
-->   [25] routes { ... }
      ...
    }
CML.fw> ./goto = acl
-->   acl INTOK { ... }
CML.fw> acl NEW {
CML.fw.NEW> }
...
CMLR-592-W .: Section 'NEW' definition not correct
CML.fw> ./show . acl
      acl INTOK { ... }
-->   acl NEW { ... }
CML.fw>
```

The above example illustrates also the "online verification" function. After completing the configuration directive, the CML tries to check whether it would be readable by the low-level configuration reader, and prospective errors are displayed.

The **show** and **goto** commands can be typed in a shortened version without names, as shown in the following example.

The rules for entering sections apply to items, too. Type a non-existent item to add it, or an existing *unrepeatable* item to change it. The only difference concerns work with repeatable items. If you type one, a new item will be added; to edit an item, use the **edit** command. The C^3H (Section 3.2.2) may help significantly, if used.

```
CML.fw> acl INTOK {
CML.fw.INTOK> ./
    acl INTOK {
      [ 1] from { ^system.INT.ipv4.net };
      [ 2] service { DNS, FTP, HTTP, HTTPS, POP3, IMAP4, SMTP, SSH };
-->   [ 3] accept;
    }
CML.fw> 1
```

```
-->   from { ^system.INT.ipv4.net };
CML.fw.INTOK> ./edit <Tab>
CML.fw.INTOK> ./edit from { ^system.INT.ipv4.net };
CML.fw.INTOK> ./edit from { $INT-ADDR };
CML.fw.INTOK> from { $EXT-ADDR };
CML.fw.INTOK> ./
    acl INTOK {
      [ 1] from { $INT-ADDR };
-->   [ 2] from { $EXT-ADDR };
      [ 3] service { DNS, FTP, HTTP, HTTPS, POP3, IMAP4, SMTP, SSH };
      [ 4] accept;
    }
```

Context-oriented online help is available at every point of the configuration using either C^{3}H (the <Tab> key), or the **info** command.

```
CML.fw.INTOK> ./info descr
  Repeatable section acl...
      General ACL definition.
  Constraints:
      Exactly one of DENY and ACCEPT must be specified.
      ...
      Item SERVICE must be specified..

  acl <name> {
    * from ...;
    * to ...;
      ...
      service ...;
  }
CML.fw.INTOK> doctype-ident-order <Tab>
  doctype-ident-order [for <for>] <order>;

  Elements:
--> [ KEY DIRECTION-SET for = * ]
      DOCTYPE-IDENT-METHOD-LIST order
```

The **delete** command can be used to remove unneeded configuration directives (the *last* removed node can be restored using the **undelete** command). The **hide** command provides a similar function. However, a hidden node remains in the configuration, even though it plays no role in the actual configuration, and can be re-enabled in the future using the **unhide** command.

```
CML.fw.INTOK> ./
```

```
    acl INTOK {
      [ 1] from { $INT-ADDR };
-->   [ 2] from { $EXT-ADDR };
      [ 3] service { DNS, FTP, HTTP, HTTPS, POP3, IMAP4, SMTP, SSH };
      [ 4] accept;
    }
CML.fw.INTOK> ./delete
-->   service { DNS, FTP, HTTP, HTTPS, POP3, IMAP4, SMTP, SSH };
CML.fw.INTOK> ./undelete
-->   from { $EXT-ADDR };
CML.fw.INTOK> ./hide
-->   hidden from { $EXT-ADDR };
CML.fw.INTOK> ./unhide
-->   from { $EXT-ADDR };
```

The CML has a clipboard, which can be used to copy and move configuration directives. Repeatable sections can be renamed when being pasted.

```
CML.fw> ./copy
CMLM-751-N 1 node(s) stored to clipboard
CML.fw> ./paste
CMLT-570-E Section 'INTOK' already defined
CML.fw> ./paste NEW
-->   acl NEW { ... }
```

When the configuration changes are finished, the configuration should be saved. Before actually saving it, the CML verifies the configuration. If the verification fails, the save operation does not proceed. The verification can be skipped (e.g. when saving temporarily), but it is strongly recommended not to do so regularly.

```
CML.fw> ./save
CKGB-711-N ---- Trying verification of system 'fw'...
CFGR-500-E FTP.NEW: Authentication method must be set
CFGR-500-E HTTP.NEW: Authentication method must be set
CKGB-718-E ---- System 'fw' verification failed
CMLM-713-E Saving cancelled
CML.fw> ./save !
CMLS-715-N Configuration saved to file '/.../kernun.cml'
/.../kernun.cml,v  <--  /.../kernun.cml
new revision: 1.3; previous revision: 1.2
enter log message, terminated with single '.' or end of file:
>> temporary saving
>> .
```

```
done
CMLT-800-N Configuration identification: kernun.cml,v 1.3 ...
```

3.3 Administrative Utilities

Kernun UTM contains many administrative utilities, which can be invoked from the shell command line. The most important of them, **kat**(8) and **cml**(8), have been described in Section 3.2. In addition to these two utilities and the standard FreeBSD commands there is a set of Kernun-specific utilities, which are introduced in this section. For detailed description, see the respective reference pages.

Note

The full contents of terminal sessions of the administrator are logged[5] in the `/var/log/session-USER-DATE-HOST.log.gz` files.

bootmgr(8) Kernun UTM boot manager configuration tool. It can be used to view and set system partition labels, toggle the bootable flag of system partitions, and select the default system partition for the next boot. Its functionality is provided also by the GUI, on the Kernun systems page of the System Manager.

diskdb(1) This program can traverse the file system and create a database file containing information about all files and directories in it. It is able to compare two such databases and display the list of changes to the file system between the times of creation of the databases. The **diskdb** program is used internally by the Kernun UTM backup, restore, and upgrade tools. It can be used also to detect unexpected changes to vital system files.

fwpasswd(1) Program for management of password files for user authentication. See Section 5.14.1 for more information.

log-ts(1) It selects messages from Kernun UTM's logs using various criteria. Automatically selects the log files that cover the required time interval.

kernun-audit(1) This tool reports bugs in the installed Kernun software and availability of new versions. It is usually run automatically at a regular interval (daily) and the reports are e-mailed to the administrator. See also Section 2.8.

mkblacklist(1)

printblacklist(1)

resolveblacklist(1) Utilities for manipulation and viewing of blacklists for **http-proxy**(8). See Section 5.18.2 for instructions on the use of blacklists in the HTTP proxy.

[5] A session is copied into a file using the **screen** (1) command invoked from `/root/.profile`.

monitor(1) A utility for online monitoring of active proxy sessions. Monitoring is provided also by the GUI on the GKAT Monitor pages for proxies. The configuration of monitoring is described in Section 5.6.3.

ooba-samba(1) This script is used by the out-of-band authentication to obtain the list of logged-in users from a Samba server. The script should be copied to the Samba server machine and run automatically, as described in Section 5.14.8.

quarc.sh(1) A tool for SMTP proxy quarantine management. In the ACLs of the **smtp-proxy**(8) it can be specified that some mail messages are not to be delivered, but stored in the quarantine instead. The quarantine management tool can be used to list the quarantined content, as well as remove or resend messages from the quarantine.

rrd(1) This is the command line interface to system parameter monitoring, as described in Section 5.6.5. It is also used by the GUI to generate graphs displayed on the Graphs page (Figure 3.8).

sum-stats(1) This script processes Kernun UTM logs and generates proxy statistics. The configuration of statistics is described in Section 5.6.4. The statistics can be displayed by the GUI in the window invoked by selecting File | Statistics from the main menu.

switchlog(1) This is a program for fast filtration of a Kernun UTM log based on proxy name and message ID matching. It can multiplex selected messages to several output streams for further processing. For example, the process of proxy statistics generation uses **switchlog** to quickly select statistical messages written by proxies and pass them to **sum-*** scripts.

triplicator(1) A tool for SMTP proxy greylisting database management. Greylisting is one of antispam techniques implemented in Kernun UTM. See Section 5.16.2 for more information.

sysmgr(8) This is the command-line alternative to the GUI System Manager window. It can perform installation, upgrade, backup, and restoring of Kernun UTM. Refer to Chapter 2 for detailed instructions on how to perform various Kernun system management tasks.

Chapter 4

In the first section of this chapter, we will introduce the principles of the Kernun UTM configuration. We will describe the syntax and semantics of the configuration language and its two different representations. The textual representation is used in the configuration file (usually `/usr/local/kernun/conf/kernun.cml`) and by the command line configuration tool **cml**(8). The graphical representation is used by the Kernun GUI. Then we will discuss the initial configuration that is generated when the newly installed Kernun UTM is booted for the first time. We will explain the meaning of individual items in this configuration and how to change them. For information about configuration of additional functionality not included in the initial configuration, such as more types of proxies, authentication, antivirus, or antispam modules, refer to Chapter 5.

Kernun UTM keeps its configuration in a text format called CML (Configuration Meta Language). The configuration is usually stored in the `/usr/local/kernun/conf/kernun.cml` file. Whenever the configuration file is saved by the GUI or command line tools, it is also saved to a RCS (Revision Control System) file called `/usr/local/kernun/conf/kernun.cml,v`. The RCS file contains the complete history of the configuration, which makes it possible to return to any past version of the configuration file, show differences between two versions, and more. The Kernun GUI and **cml**(8) provide functions for work with RCS files. For more advanced operations with RCS, you can use command line system tools, see rcsintro(7).

Speaking more precisely, there are two levels of configuration: *high* and *low*. Usually, the administrator maintains only the high-level configuration file. On the other hand, each Kernun UTM component reads its low-level configuration file upon starting. The Kernun UTM configuration handling tools generate a set of low-level configuration files[1] from the single high-level configuration file. In fact, the high-level configuration is more or less an extension of the low-level configuration of proxies. This concept makes it possible to keep a simple configuration file for each proxy and, at the same time, maintain a single file that describes the whole system in a comfortable way.

[1] They are mainly proxy configuration files and some system configuration files, e.g., `/etc/rc.conf`.

4.1 Configuration Language

The configuration has a tree structure consisting of *sections*, *items*, and *elements*. A section is a named group of logically connected configuration directives. It can contain other (sub)sections and items. An item is a named group of values—elements. The configuration structure is reflected by the view of the configuration in the GUI, see Figure 4.1, as well as in the corresponding textual configuration file:

```
## Configuration of a single Kernun UTM system ❶
system fw { ❷
  ## Host name without domain
  hostname fw; ❸
  ## Domain name
  domain pha.tns.cz;

  ## The default crontab defines typically used periodic actions
  crontab {
    $DEFAULT-CRONTAB;
  }

  ## The network interface connected to the internal network
  interface INT { ❹
    dev em0;
    ipv4 [192.168.10.1/24]; ❺
  }
```

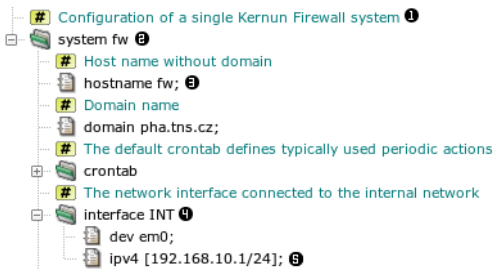

Figure 4.1: Tree structure of the configuration

In the sample configuration there is a section of type `system` named `fw` ❷. It contains several subsections and items. For example, an item `hostname` with an element (value) `fw` ❸, or a subsection `interface` named `INT` ❹ containing an item `ipv4` ❺. Note that it is possible to include comments in the configuration file. They are also displayed by the GUI ❶.

Nodes of the configuration tree are referenced by *paths*. A path lists the ancestor sections in the tree. In the above example, the `fw.INT.ipv4` path denotes the item ❺. Instead of specifying a value for an element, it is possible to use a reference to another value in the configuration with the same type. For example, the initial configuration contains

```
system fw {
  ...
  ssh-server SSHD {
    listen-sock ^system.INT.ipv4.host ❶ : 22;
    ...
  }
  ...
}
```

Here, ❶ denotes the value obtained by going up the tree to the `system` type node (section `system fw`), and then down to the node named `INT` (it is `interface INT`), then to `ipv4`, and finally we take its `host` element (value).

References ensure the consistency if the same value appears at two or more places in the configuration. If the same path is to be repeated several times, e.g., in `listen-on` of several proxies, we may avoid the use of the full path by using a *variable*.

```
system fw {
  interface INT {
    dev em0;
    ipv4 [192.168.10.1/24];
  }
  set INT_IP = ^system.INT.ipv4.host; ❶
  ...
  ## SSH daemon for remote administrator access from the internal network

  ssh-server SSHD {
    listen-sock $INT_IP ❷ : 22;
    passwd-auth;
  }
```

The previous example is changed so that the reference path is stored in a variable called `INT_IP` ❶ and the variable is used instead the full path later ❷. For the GUI view of the same configuration, see Figure 4.2. In addition to these *data variables* it is possible to define variables containing several configuration items or even whole sections. These *section variables* can be parametrized. A simple example of section variables will be shown in the next paragraph, along with file includes.

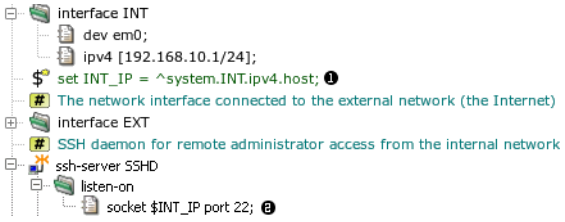

Figure 4.2: Tree structure of the configuration

A configuration file can `include` another file. After the installation there are several include files in `/usr/local/kernun/conf/samples/include`. The initial configuration includes two of them, `root-servers.cml` (a list of root DNS servers) and `crontab.cml` (the default configuration of the **cron** (8) daemon that runs periodic actions in the system, e.g., log rotation). Example: the `crontab.cml` include file contains

```
set DEFAULT-CRONTAB system.crontab {
  ...
}
set DEFAULT-PERIODIC system.periodic-conf {

    set-env daily_clean_hoststat_enable "NO";
    set-env daily_status_disks_df_flags " -i";
    set-env daily_status_mail_rejects_enable "NO";
    set-env daily_status_include_submit_mailq "NO";
    set-env daily_submit_queuerun "NO";
}
```

It is included in the main configuration file `kernun.cml`

```
include "samples/include/crontab.cml";
...
system fw {
  ...
  crontab {
    $DEFAULT-CRONTAB;
  }
```

The same excerpt of the configuration viewed in the GUI is shown in Figure 4.3. The `crontab.cml` file is included at ❶. It defines the variables `DEFAULT-CRONTAB` and `DEFAULT-PERIODIC`. The first variable is used at ❷. Note that the GUI displays the contents of included files.

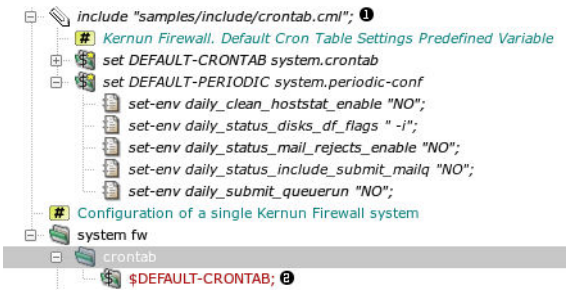

Figure 4.3: Including configuration files and using section variables

Configuration elements are of various types. All the types and their exact syntax are described in configuration(7). Here we only list the types and present the most important syntax rules.

- *Integers* — a nonnegative decimal or hexadecimal value; several types with different numbers of bits
- *Port number* — an integer or a service name from `/etc/services`
- *String* — a sequence of characters in double quotes. It is possible to omit the quotes for strings that contain only a limited set of characters.
- *Regular expression* — delimited by slashes
- *Host* — an IP address in dotted decimal format enclosed in brackets or a host name
- *Interface address*, *network address* — an IP address with mask, enclosed in brackets
- *Socket* — a host and a port number
- *Enumeration* — a set of names representing integer constants
- *List* — a sequence of values of a basic type, separated by commas and enclosed in braces. Lists can be nested.
- *Set* — a set of values of a basic type. Syntax is similar to lists with additional features— ranges, exclusion of members, and wildcards. Sets of some types allow special members, e.g., regular expressions in a set of strings or host addresses.

Important

In the GUI, the values must be entered including quoting characters, such as double quotes for strings and brackets for IP addresses.

We have introduced only the basic concepts of the Kernun UTM configuration language. The complete definition of the language syntax can be found in reference pages configuration(7) and cml(8). The semantics of the configuration is defined by the meaning of individual sections, items, and elements. Semantic rules also define the subsections and items each section may contain. There are constraints, e.g. that the existence of a configuration node requires or forbids the existence of another node. The set of possible values of a configuration element can be limited. A section or item can be `repeatable` (can occur several times, in the case of sections with different names), or `non-repeatable` (cannot occur more than once; non-repeatable sections do not have names). For example, in Figure 4.1, the named sections `system` and `interface` are repeatable, whereas the section `crontab` is non-repeatable. The semantics of the configuration is described comprehensively in section 5 of the reference pages (in the appendix of this handbook or in manual pages displayed using the **man**(1)) command.

4.2 The Initial Configuration

A newly installed Kernun UTM is configured by an interactive script during the first boot, as described in Section 2.5.2. The initial configuration is identical to the configuration sample file

/usr/local/kernun/conf/samples/cml/simple.cml. The configuration script takes this file and substitutes the values of parameters entered by the administrator, e.g., IP addresses. The configuration file simple.cml is suitable for Kernun UTM with two network interfaces, one connected to the internal (protected) network and the other to the external network (the Internet). Kernun UTM will be configured so that clients that use selected network protocols in the internal network may access servers in the external network, but access is denied for other protocols and to clients in the external network. The administrator can connect to Kernun UTM from the internal network. Although it is possible to enable proxies for some application protocols using the initial configuration script, we will assume that the administrator has chosen the more secure way, i.e., not to enable any proxy initially, but to do so later selectively using the GUI.

4.2.1 Global Level

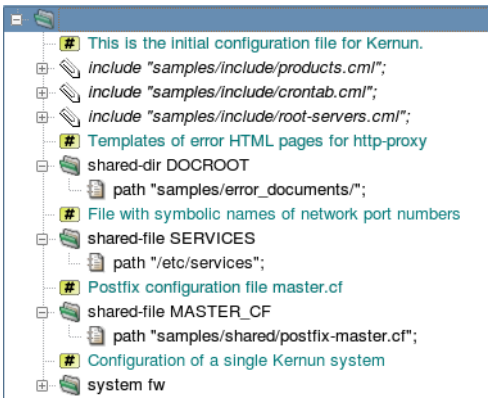

Figure 4.4: The global level of the initial configuration file

The global level of the initial configuration file is depicted in Figure 4.4. It begins with three standard include files[2]. The first file, samples/include/products.cml, defines variables usable as the system.product item. See Section 2.3 for more information about Kernun product specification in the configuration. The second file, samples/include/crontab.cml, defines variables usable as the contents of the system.crontab and system.periodic-conf sections. The third file, samples/include/root-servers.cml, contains a list of root DNS servers. Then, several shared-dir and shared-file sections specify files that are copied from the configuration directory to their target locations when the configuration is applied. The configuration can be edited on one Kernun UTM and then applied remotely to another. In this case, the shared files and directories are copied from the system where the configuration is edited to the system where it is applied.

[2] All relative paths here are relative to the configuration directory /usr/local/kernun/conf.

4.2.2 System

The rest of the sample configuration is the `system fw` section. It defines a single Kernun UTM system. The configuration of a standalone Kernun UTM usually contains only one `system` section. If several alternative configurations are used or several systems are configured from one place (e.g., in high-availability clusters), more than one `system` sections can be defined.

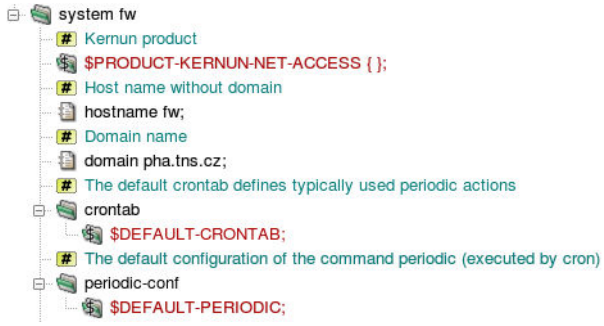

Figure 4.5: Various `system`-level definitions

First, the type of the Kernun product is specified (Figure 4.5) using one of standard product variables from `samples/include/products.cml`. The product specification controls which components can be configured. During the application of the configuration, a check is made that the product installed on the target system matches the specification. See also Section 2.3 for a more detailed explanation of the product specification in the configuration. The `system` parameters start with host and domain names. The configuration of actions executed periodically by the **cron** (8) daemon and by the **periodic** (8) command is defined in the sections `crontab` and `periodic-conf`. These sections are filled in by the application of variables defined in `crontab.cml`.

Figure 4.6: Definitions of network interfaces

The following part of the `system` section, displayed in Figure 4.6, contains the settings of two network interfaces, connected to the internal (`INT`) and external (`EXT`) network, respectively. Two values are specified for each interface: the name of the network interface device as used by the operating system (item `dev`), and the IP address with the network mask (item `ipv4`). Note that `interface` is a repeatable section, so it can occur more than twice. For example, a demilitarized zone network can be connected via a third interface in a more complex network topology.

The `system` section then defines further network-related parameters, see Figure 4.7. Domain name resolution parameters are set in the section `resolver`. In the initial configuration, the only item in this section selects a DNS server. There may be several `resolver` sections. The

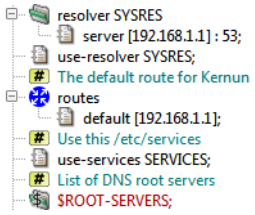

Figure 4.7: Definitions of various network parameters

section that is used globally for resolutions performed by Kernun UTM components is chosen by `use-resolver`. Each proxy can use the global resolver, or the resolver setting can be overridden for the proxy using a `use-resolver` section local to the proxy configuration section. In the initial configuration, the global resolver is shared by all proxies. Note that the resolver is used for address resolution performed by Kernun UTM itself. If clients in the internal network should have access to the DNS, which is usually the case, either a DNS server can be configured in the internal network, or the clients can resolve via a Kernun UTM **dns-proxy**.

Note

As the Kernun UTM **dns-proxy** does not cache responses obtained from DNS servers in the external network, it is inefficient to configure clients in the internal network to resolve directly via the proxy. The recommended configuration is to create a caching-only DNS server in the internal network and to set **dns-proxy** as its forwarder. The internal DNS server can be created in Kernun UTM, by adding section `system.nameserver` to the configuration.

The `routes` subsection defines static routes. In the sample configuration, only the `default` route to a router in the external network is specified. The `use-services` item selects a `shared-file`[3] that will be copied into `/etc/services`. It defines port numbers assigned to various network services. The last network-related parameter is a list of root DNS servers, section `ns-list`. It is set by the application of the section variable `ROOT-SERVERS` defined in the included file `root-servers.cml`.

4.2.3 SSH Server

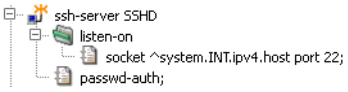

Figure 4.8: An SSH server for administrative access

Kernun UTM is usually administered remotely via the network. The administrator connects to Kernun UTM using SSH, either directly in order to obtain a shell command line, or via the

[3] Sections `shared-file` are created at the global level of the configuration file.

GUI, which uses SSH internally. Therefore, an SSH server for administrator's access should be defined in the configuration. The initial configuration contains a single SSH server section called `ssh-server SSHD`, see Figure 4.8. It listens on the standard SSH port 22 on the internal network interface only. Hence the administrator can connect to the Kernun UTM from the internal network only. If access from the external network is desired, the SSH server must be reconfigured to listen on the external interface as well.

The server allows password-based authentication of non-root users. It is used for the initial download of the root's SSH key using the special `keygen` account, as described in Section 2.5.2. The root can log in to Kernun UTM with the SSH keys listed in section `ssh-keys`. By default, this section contains the key generated by the initial configuration process and accessible via the `keygen` account.

4.2.4 Local Mail Handling

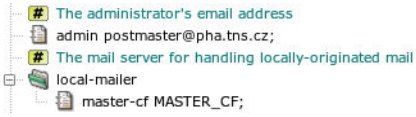

Figure 4.9: The server for handling locally-originated mail

Kernun UTM generates some e-mail messages locally, mainly as the output of periodic actions executed by **cron** (8). These messages are delivered to the address provided in the `admin` item by the mail server configured in the `local-mailer` section, see Figure 4.9. The default local mailer configuration refers to the standard file `master.cf` selected by a global `shared-file` parameter.

The server that handles local mail is Postfix-configured so that it does not listen for any incoming network connections. It handles only mail sent by the **mail** (1) command. The administrator's e-mail address is set as an alias for the `root` user in `/etc/aliases`.

4.2.5 Application Proxies and ACLs

Kernun UTM utilizes both the packet filter and application proxy firewalling technologies. The use of application proxies is preferred, because it provides a higher level of security and finer control of the network traffic. The network traffic is therefore passed through Kernun UTM mainly via proxies that handle individual application protocols.

It is possible to enable proxies for several frequently used network protocols in the initial configuration. However, in order to maintain better control over the network traffic it is advisable not to enable any proxy when the initial configuration script creates the configuration of a newly installed Kernun UTM, but rather to selectively enable and configure the proxies later, when the administrator connects to Kernun UTM via the GUI. The part of the configuration that is directly related to proxies is shown in Figure 4.10. Note that all proxy sections are *hidden*, so that proxy parameters are defined, but have no effect and the hidden proxies are not started[4].

[4] For the present, you can ignore the section `smtp-forwarder`. It will be discussed later, together with the

Figure 4.10: Hidden proxies and a global ACL

Proxies can be non-transparent or transparent, see Section 5.7 for more details. By default, all proxies in the initial configuration except `dns-proxy` are configured as transparent, which enables the use of clients in the internal network without any reconfiguration.

Access control in proxies is driven by *access control lists (ACLs)*. Each proxy has one or more layers (phases) of ACLs. Individual phases are consulted at specific moments during the processing of the network communication.

For example, in **http-proxy**, there are three phases of ACLs: the `session-acl` sections are consulted when a new connection from a client is accepted. They make early decisions based on values, such as the client's IP address. The second phase, `request-acl`, is checked after the HTTP request is received from the client. Its actions can depend e.g. on the request URI or on the values of some request headers. The third phase, `doc-acl`, is applied when the HTTP reply headers are received. On the other hand, **tcp-proxy**, which handles only the TCP data stream without deeper understanding, has a single ACL phase, `session-acl`.

The ACLs that belong to one phase are tested sequentially. The input conditions of every ACL section are evaluated and the first ACL, for which all the conditions are true, is selected. The chosen ACL can `accept` or `deny` the session, thus effectively allowing it to continue or terminating it. Various parameters that affect further handling of the session by the proxy can be set by the ACL. If no matching ACL is found, the session is terminated.

ACLs can be located at two different places in the configuration. Global ACLs defined in the `system` section belong to the first phase (`session-acl`). Each global ACL has a `service` item containing the list of proxies, to which it is applicable. The initial configuration contains one global `acl INTOK` that applies to all proxies and accepts sessions originated from clients in the internal network, as can be seen in Figure 4.10. Note that sessions initiated by clients in the external network do not match the `from` condition of `acl INTOK` and there is no other ACL that could match, hence these sessions are denied[5]. Local ACLs are defined in the configuration

smtp-proxy.

[5] In fact, such sessions do not even begin regardless of ACLs, because the proxies are configured to listen on the

sections of individual proxies. Each such ACL applies only to the proxy, in which it is defined. Phase one (`session-acl`) ACLs in a proxy are merged with the global ACLs that have this proxy in their `service` lists. For each `session-acl` in a proxy there must be a global ACL with the same name.

> **Note**
>
> The first-phase ACLs are checked in the order defined for the global ACLs. The per-proxy `session-acl` sections can add proxy-specific items to the ACLs, but do not change the order of ACLs.

4.2.6 DNS Proxy

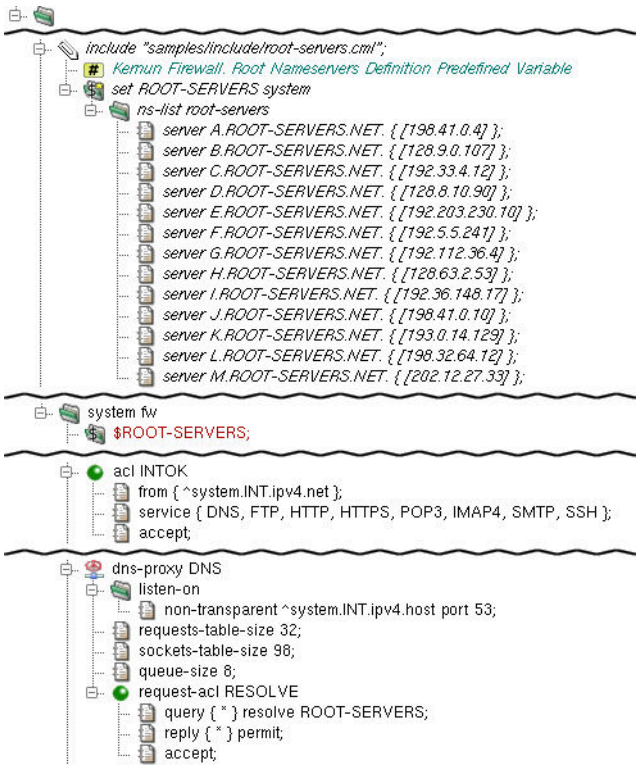

Figure 4.11: DNS Proxy

In a typical scenario, the DNS proxy provides the domain name resolution service for clients behind Kernun UTM. It is defined by its particular `dns-proxy` section which resides in the `system` section. In this example, the `listen-on` section specifies that the DNS proxy will listen non-transparently on the internal interface on port 53. This means that clients from the internal network must set the IP address of Kernun UTM's internal interface as their DNS server in order to use this proxy. The `chroot-dir` item specifies that the process will be *chrooted* to the directory `/usr/local/kernun/root`, see chroot(8).

internal interface only.

The requests-table-size and sockets-table-size items are obligatory for dns-proxy. Every domain name resolution request is stored in a table that contains all the necessary information. The size of this table is specified by the requests-table-size item. It is recommended to reserve slightly more items than the estimated number of parallel requests. Besides the number of requests, the number of simultaneously opened sockets is monitored as well. This maximum must be specified by the sockets-table-size item. See dns-proxy(5) for details.

Caution

The DNS proxy is *not* designed to be used as a name server. The typical scenario is that all clients ask another server in the internal network and this server queries the proxy (if needed) and caches the answers. See Section 5.3 for details.

dns-proxy needs at least one request-acl section providing details on what traffic is permitted or not, and the way it should be dealt with. In this example, the accept item specifies that *request-acl RESOLVE* describes an *accepting* request-acl. This means that this section defines the positive case — the conditions, under which the request is accepted. Then *query * resolve ROOT-SERVERS* directs the proxy to use the ROOT-SERVERS to resolve all the requests. Further on, it is defined that all the respective replies from the ROOT-SERVERS are permitted to be returned to the clients.

The ROOT-SERVERS item is defined in the sample file samples/include/root-servers.cml, which is included in the configuration using the include keyword. This file contains the definition of the $ROOT-SERVERS variable, which defines the ns-list section — a list of DNS root servers. In order to *apply* that definition, it is necessary to include the $ROOT-SERVERS keyword in the configuration.

Finally, dns-proxy must be referenced by at least one system ACL. In this example, the access to the DNS proxy is granted for all clients from the internal network.

4.2.7 HTTP Proxy

http-proxy is the proxy daemon for HyperText Transfer Protocol (RFCs 1945, 2616). It supports version 0.9, 1.0 and 1.1 HTTP clients and version 1.0 and 1.1 HTTP servers. The proxy also supports secure communication via the SSL/TLS protocols.

In the example depicted in Figure 4.12, http-proxy is configured to listen on the internal network interface on port 80 in the transparent mode, and on port 3128 in the non-transparent mode; see Section 4.2.5 for details about transparency. Again, the proxy is configured by chroot-dir to be chrooted into /usr/local/kernun/root/, see chroot(8).

The document-root item defines the root directory of the error document set. In this example, it refers to DOCROOT — the shared-dir section defined at system is a list of path items designating directories for further use. Note that the relative path (i.e., one that does not start with "/") is in fact relative to /usr/local/kernun/conf.

http-proxy uses three-phase ACLs.

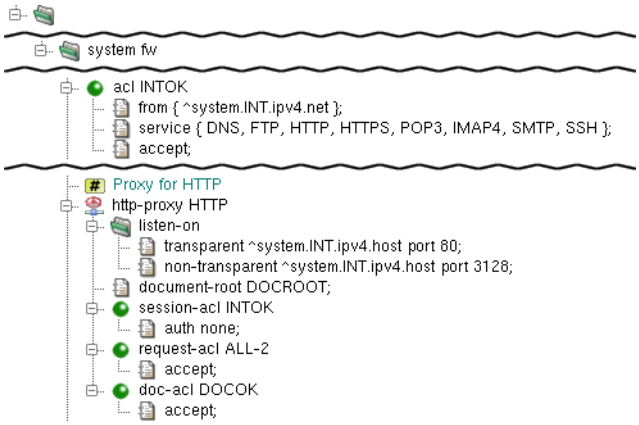

Figure 4.12: HTTP Proxy

- `session-acl` is checked once for each client connection. It permits or denies client access and sets some connection parameters.
- `request-acl` is checked for each HTTP request after the request headers are received from the client, but before anything is sent to the server. It decides whether the request is permitted or denied, and it can also set some request parameters. Note that there can be several requests per connection if persistent connections are used.
- `doc-acl` is checked for each HTTP request after the response headers are received from the server, but before the response is sent to client.

Important

The `session-acl` section of any proxy is always based on an `acl` defined at the `system` level. The items that are common for all types of proxies are defined in the common part at the `system` level. The items that are specific for each proxy are defined within the particular proxy's section [6].

In this simple example, `http-proxy` is configured not to use authentication (`auth none`). You might have noticed that we did not specify the `accept` item in `session-acl INTOK`, nor did we restrict the IP range of clients that can use `http-proxy`. The reason is that we only specify the `acl` section `INTOK` that we defined earlier. In fact, `FIREWALL.HTTP.INTOK` is a local copy of `FIREWALL.INTOK`, and we have particularized this local copy. Finally, `request-acl` and `doc-acl` are configured to accept any request or document, respectively. However, CONNECT method requests are limited to port 443 to allow SSL/TLS connections only.

See dns-proxy(5) for details.

[6] Moreover, some items can be configured at both levels. The proxy-specific one takes precedence in that case.

4.2.8 FTP Proxy

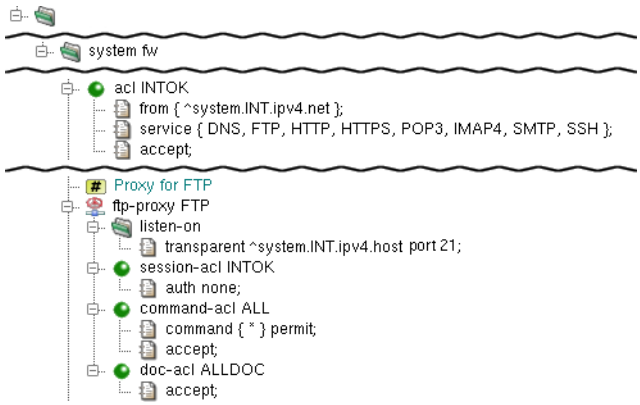

Figure 4.13: FTP proxy

`ftp-proxy` provides the proxy service for File Transfer Protocol (RFC 959) and its extensions. For crucial commands, it behaves like a server for the client and vice versa, with full syntax and semantics verification. Commands that have no impact on the session state are only recognized and simply forwarded to server.

The simple configuration of `ftp-proxy` is quite similar to the simple configuration of `http-proxy` (see Section 4.2.7). In the example at Figure 4.13, it is configured to listen on the internal network interface on port 21 in the `transparent` mode and chrooted into `/usr/local/kernun/root/` (see chroot(8)). No restrictions are configured, so everyone (from the internal network — as specified in the `INTOK` acl) is allowed to use the `ftp-proxy`. As usual, the `ftp-proxy FTP` must be present in some ACL; in this case, it is referenced in the `service` list in the `INTOK` definition.

`ftp-proxy` uses three-phase ACLs.

- `session-acl` is checked once for each client connection. It permits or denies client access and sets some connection parameters.
- `command-acl` decides how to handle particular protocol commands depending on client parameters, destination server, proxy-user, etc. Each command is checked against `command` items in the order of their appearance in the cfg file, and the first matching one is used. If no one matches, the command is denied.
- `doc-acl` is checked for each HTTP request after the response headers are received from the server, but before the response is sent to the client. It decides how to handle particular files transferred via the proxy, depending on the file name, type and transfer direction.

For more information about `ftp-proxy`, see ftp-proxy(8) and ftp-proxy(5).

4.2.9 HTTPS and SSH Proxy

Both HTTPS and SSH protocols are handled by the generic `tcp-proxy`, because they use encrypted communication and Kernun UTM therefore cannot understand the protocol data. The

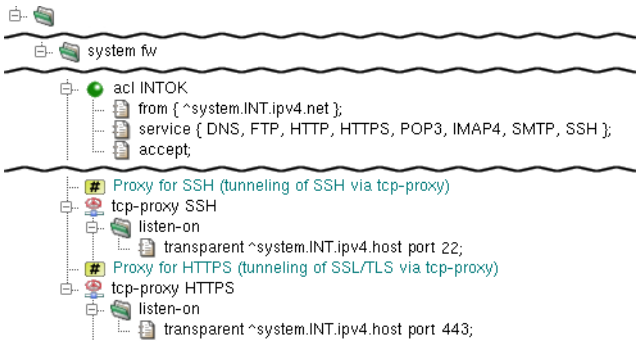

Figure 4.14: Proxies `HTTPS` and `SSH`

configurations of `tcp-proxy HTTPS` and `tcp-proxy SSH` are almost identical, see Figure 4.14. They differ only by the port they listen on—the port number is 443 for `HTTPS` and `22` for SSH.

Both proxies listen for clients connecting transparently via the internal interface of Kernun UTM (section `listen-on`). The proxies run with identity of `proxy-user kernun` and are chrooted into directory `/usr/local/kernun/root`. They do not define any ACL, therefore the global `acl INTOK` applies to them unmodified.

4.2.10 SMTP Proxy

`smtp-proxy` is the proxy daemon for Simple Mail Transfer Protocol (RFCs 2821, 2822, 2045 etc.). The SMTP protocol is used to send electronic mail from clients to mail servers as well as between mail servers. The task of `smtp-proxy` is to apply a security policy, check the incoming mail for correctness and then use some `smtp-forwarder` to queue and distribute mail. In other words, there must be an SMTP server (other than **Kernun UTM**) to deliver e-mail. The easiest way is to employ the common UNIX `postfix` daemon for this task. The configuration can be included in the **Kernun UTM** configuration and the proper configuration files for **postfix** are then generated automatically.

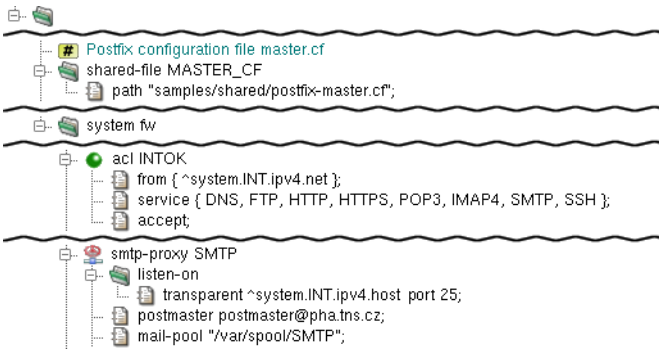

Figure 4.15: `SMTP` Proxy System Sections and Forward Agent

In the typical scenario depicted in Figure 4.15, **postfix** is configured to listen at Kernun UTM's loopback address and forward e-mails send to it by `smtp-proxy`, which listens at Kernun UTM's

internal address. Note that under this settings mail does not arrive from the Internet to the internal network. We will explain an alternative scenario in the second part of this chapter.

The **postfix** program is configured from **Kernun UTM** by the `agent` section in `smtp-forwarder`. It must contain a `master-cf` item — an anchor to `shared-file` that contains the **Postfix** `master.cf` configuration file. There is a sample in `samples/shared/postfix-master.cf`. There also must be specified at least one `server` item in `smtp-forwarder` — an address, to which the mail is to be sent.

The proxy itself listens transparently on Kernun UTM internal address port 25 (as defined by the `listen-on` section). Besides the common `chroot-dir` item there must also be present a `postmaster` item, which defines the postmaster e-mail address, and a `mail-pool` item, which defines the directory used as temporary storage for incoming mails.

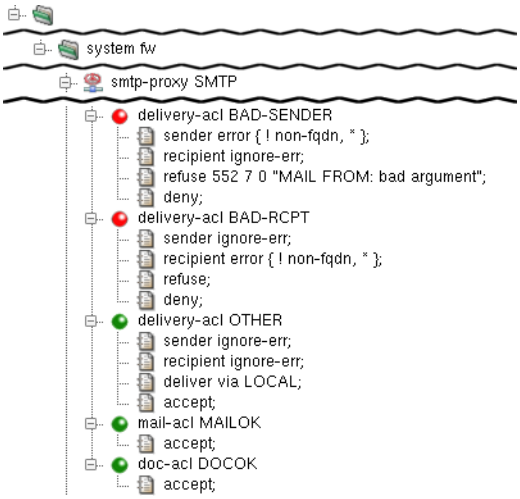

Figure 4.16: `SMTP` Proxy ACLs

`smtp-proxy` uses four types of ACLs on three levels:

- `session-acl` — (level 1) is checked once for each client connection. It either defines the general protocol behavior, or rejects the connection. In addition to the generic ACL conditions and actions, some smtp-proxy-specific conditions and parameters can be set (see smtp-proxy(5)).
- `delivery-acl` — (level 2) is checked once for each mail recipient. It either defines the response to the RCPT TO command, i.e. the way of delivery, or rejects the particular addressee.
- `mail-acl` — (level 3M) is checked once for each mail recipient. Its rules control whether the forwarding of the mail to the particular recipient should be rejected or accepted.
- `doc-acl` — (level 3D) is checked once for each recipient and document (MIME part) and defines the document processing mode (e.g. filtering, replacing etc.).

The example in Figure 4.16 shows three different `delivery-acl` items. The first one (`BAD-SENDER`) refuses mail with incorrect senders (i.e., the MAIL FROM argument), the

second one (BAD-RCPT) rejects mail with incorrect recipients and the last one (OTHER) accepts all other mail (that is, mail that matches neither of the preceding delivery-acls). The mail-acl and doc-acl items are configured to accept all e-mails.

During the delivery-acl phase, the correctness of all the commands (**HELO**, **MAIL FROM** and **RCPT TO**) and their arguments is checked and the proper decision is made.

Warning

When resolving domain names, the current resolver section setting is applied. It means that the search list (if present) is tried when some resolution fails. If this search-list contains a domain that has the *.domain MX record set, every resolution succeeds and errors such as unknown-perm never occur. This is probably not the desired behaviour. You can solve this problem by defining a new resolver section without the search item and then using this resolver in the particular **smtp-proxy** only by means of a use-resolver item in the proxy's configuration section.

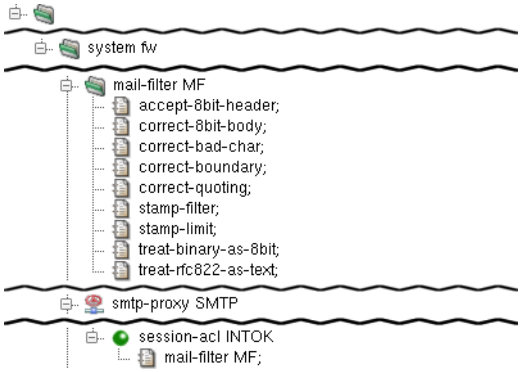

Figure 4.17: SMTP Proxy Mail Filter

smtp-proxy checks the correctness of the mail headers and MIME structure. Mail that does not conform to the RFCs is rejected. However, many clients do not respect RFCs and if the security policy allows sending of such e-mails, you can instruct the proxy to correct or even pass them. We therefore add a mail-filter item, which handles the most common cases, to the session-acl item. For details on the particular entries, see smtp-proxy(8) and mod-mail-doc(5).

In the example depicted in Figure 4.17, a reference to mail-filter is added into session-acl of smtp-proxy. The mail-filter itself is configured to:

- accept-8bit-header — accept non-US-ASCII characters in mail or MIME document headers;
- correct-8bit-body — accept and correct missing declaration of the use of non-US-ASCII characters in mail bodies or MIME documents;
- correct-bad-char — accept and correct the use of NUL char (0x00) or bare CR (not followed by LF) in mail bodies or MIME documents.

- `correct-boundary` — accept and correct the use of incorrect characters in MIME multipart boundaries.
- `correct-quoting` — accept and correct characters in SMTP and MIME headers that should be included in quotes, but are not.
- `stamp-filter` — remove 'Received:' headers that contain local-dependent information from mails. The number of the removed lines is counted and added to the e-mail as a special header X-Kernun-Loop-Info.
- `stamp-limit 30` — define the maximum number of Received-headers in mail. This check prevents mail loops.
- `treat-binary-as-8bit` — accept messages that use BINARY Content-Transfer-Encoding and treat them as 8BIT, even though it makes no sense in the current SMTP.
- `treat-rfc822-as-text` — handle the included documents as text.

Mail Server in the Internal Network

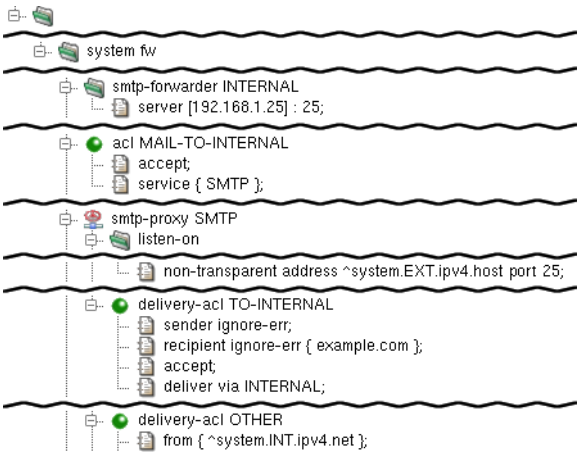

Figure 4.18: `SMTP` Proxy Mail Sever in the Internal Network

There can be a mail server in the internal network. This type of mail server usually serves as the SMTP forwarder for the internal network (that is, it forwards mail addressed outside the local network, and stores mail addressed to the internal network).

From Kernun UTM's point of view, the internal SMTP server sends some e-mails[7] to the internal interface of `smtp-proxy`. Such mail is already handled by `smtp-proxy` configured sooner in this chapter. What is new is that there can be mail from the external network addressed to someone in the local network (precisely speaking, to the mail server in the internal network). Kernun UTM should therefore listen on the external interface, accept mail addressed to the internal

[7] In particular, mail sent from the internal network addressed to someone outside the local network.

network and forward it to the internal mail server. It is possible either to create another SMTP server, or to extend the existing one. The example in Figure 4.18 shows the latter option:

Unlike other screenshots, the figure shows *only the sections and items added* to the configuration of the sample `smtp-proxy`. A new smtp forwarder `INTERNAL`, which forwards mails to the internal network (to the server 192.168.1.25), has been added. Then there is `smpt-proxy SMTP` configured to listen in the non-transparent mode on the external interface on port 25. Furthermore, there is new `delivery-acl TO-INTERNAL`, which accepts only e-mails for the selected domains (example.com) and delivers them via `smtp-forwarder INTERNAL`. Note that the order of the access control lists of the same type (`delivery-acl` in this case) is *significant*. When Kernun UTM looks for the `delivery-acl` to use, it follows the first matching one and then stops the search. The `delivery-acl TO-INTERNAL` section must be therefore created between the `BAD-RCPT` and `OTHER` sections configured earlier. The `delivery-acl OTHER` section has slightly changed; it now accepts only mail from the internal network. Note that `from { ^system.INT.ipv4.net };` must be located at the beginning of the ACL, so that the entry conditions are kept before the action (`accept` in this case). Finally, the original `delivery-acl OTHER` was restricted to accept mail from the external network (which was not necessary until the second `listen-on` item was added).

4.2.11 IMAP4 and POP3 Proxy

The IMAP4 and POP3 protocols are used to access electronic mail that is stored on mail servers. Both make it possible for client e-mail programs to download mail and present it to the user. Unlike POP3, IMAP4 enables them also to upload mail to the mailbox on the IMAP4 server.

IMAP4 Proxy

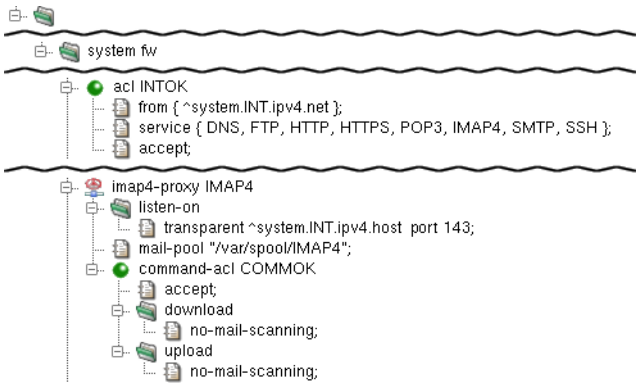

Figure 4.19: `IMAP4` Proxy

`imap4-proxy` is the proxy daemon for Internet Message Access Protocol version 4rev1 (IMAP4rev1), as defined by RFC 3501. The proxy supports secure communication via the SSL/TLS protocols.

The proxy is configured in the `imap4` section. In the sample configuration depicted in Figure 4.19, the `proxy-user` and `chroot-dir` items define the operating-system-level user,

under which the proxy should run, and the directory, to which it should be chrooted. The proxy listens transparently for requests at Kernun UTM's internal address at port 143 and, again, the proxy must be referenced by at least one ACL in the `system` section.

The `mail-pool` item defines the directory, in which e-mails are temporarily stored by the proxy. Note that this directory must be created manually by the administrator and that it must be writable by `proxy-user`. Note also that the directory can be the shared by `smtp-proxy`, `imap4-proxy` and `pop3-proxy` (or by any combination of them).

`imap4-proxy` uses three-phase ACLs. The first phase, `session-acl`, is checked once for each client connection. It permits or denies client access and sets some connection parameters. The second phase, `command-acl`, is also checked once for each connection, but it can be selected according to the client certificate if SSL/TLS is enabled by `session-acl`. Various parameters can be set in `command-acl`, e.g., the permitted sets of IMAP4 commands and capabilities, timeouts, SSL/TLS on the server connection. The third-phase ACLs are used only if mail processing is enabled in `command-acl`. Two types of these ACLs exist. `mail-acl` is checked once for each transferred e-mail. It defines the rules of the acceptance or rejection of the mail according to its content and antivirus/antispam test results. `doc-acl` is checked once for each document (MIME part) of a mail. It defines document processing, e.g., filtration or replacement by a fixed file. See **mod-mail-doc** (5).

In this example, `command-acl COMMOK` defines that no mail scanning is to be performed on either downloaded or uploaded e-mails. The item `no-mail-scanning` means, in this context, that the mail is not even opened. Were the `no-mail-scanning` item absent, the mail would be stored and unfolded to headers and attachments, etc., and various tests could be performed on any part. This alternative will be discussed in Section 5.15 and Section 5.16.

POP3 Proxy

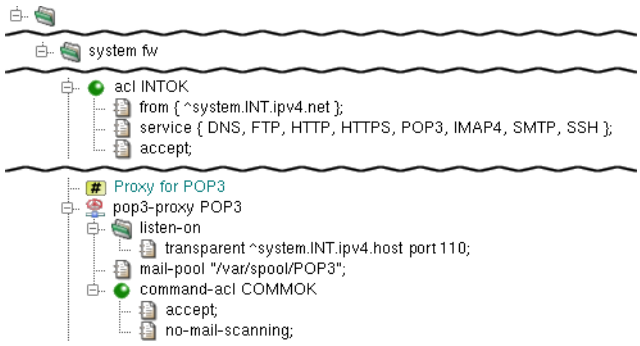

Figure 4.20: `POP3` Proxy

`pop3-proxy` is the proxy daemon for Post Office Protocol version 3 (RFCs 1939, 2449, 1734). The proxy supports secure communication via the SSL/TLS protocols.

The proxy is configured in the `pop3` section. In the sample configuration depicted in Figure 4.20, the `proxy-user` and `chroot-dir` items define the operating-system-level user, under which the proxy should run, and the directory, in which it should be chrooted. The proxy

listens transparently for requests at Kernun UTM's internal address at port 110 and, again, the proxy must be referenced by at least one ACL in the `system` section.

The `mail-pool` item defines the directory, in which the mails are temporarily stored by the proxy. Note that this directory must be created manually by the administrator and that it must be writable by `proxy-user`. Note also that the directory can be the shared by `smtp-proxy`, `imap4-proxy` and `pop3-proxy` (or by any combination of them).

`pop3-proxy` uses three-phase ACLs. The first phase, `session-acl`, is checked once for each client connection. It permits or denies client access and sets some connection parameters. The second phase, `command-acl`, is also checked once for each connection, but it can be selected according to the client certificate if SSL/TLS is enabled by `session-acl`. Various parameters can be set in `command-acl`, e.g., the permitted sets of POP3 commands and capabilities, timeouts, SSL/TLS on the server connection. The third-phase ACLs are used only if mail processing is enabled in `command-acl`. Two types of these ACLs exist. `mail-acl` is checked once for each mail transferred from the server to the client. It defines rules of the acceptance or rejection of the mail according to its content and antivirus/antispam test results. `doc-acl` is checked once for each document (MIME part) of a mail. It defines document processing, e.g., filtration or replacement by a fixed file. See **mod-mail-doc**(5).

In this example, the `no-mail-scanning` item in `command-acl COMMOK` defines that no mail scanning is to be performed. In this context, as the POP3 protocol is intended only for download of e-mails, it means that the downloaded mail is not even opened. Were the `no-mail-scanning` item absent, the mail would be stored and unfolded to headers and attachments, etc., and various tests could be performed on any part. This alternative will be discussed in Section 5.15 and Section 5.16.

4.3 Changing the Configuration

In the previous section we described the initial configuration as generated by the configuration script during the first boot after the installation of Kernun UTM. In this section we will demonstrate how to modify the configuration. We will make the following changes:

- enable **dns-proxy**;
- enable **http-proxy**;
- restrict HTTP to a subset of the internal network;
- allow only the HTTP method `GET`.

First of all, we want to permit the HTTP traffic. As the transparent mode of the proxy is used, the clients connect directly to servers. Therefore, the clients must be able to resolve server addresses, which is why we need to enable DNS as well. The following steps produce a working configuration with the required changes.

1. Start the GUI and connect to Kernun UTM, as described in Section 3.1.1. If Kernun UTM is running the unmodified initial configuration, you will see the GKAT window with only a few components running, as depicted in Figure 4.21.

2. Select Configuration | Edit configuration of the firewall in the menu to open the configuration.

3. In the left-hand part of the configuration window, look for a line containing `hidden dns-proxy DNS`. Select the line with the left mouse button, then click the right mouse button to open the context menu and select Hide/Unhide. The line will change to `dns-proxy DNS`.

4. Repeat the same action for the line containing `hidden http-proxy HTTP`.

5. Select the line `acl INTOK` and then select Add node next to this node | Section or Item from the context menu. In the Create Item/Section dialog, select `acl` as the type of the section and set the section's name, for example `NET1_DENY`. You have created a new global ACL.

6. Now select the newly created ACL and use Add new node as last child | Section or Item from its context menu repeatedly to add the `from`, `service`, and `deny` items.

7. Select the `from` item. In the right-hand part of the configuration window you can see the current value of the item, which is the empty set. Add a new value to the set by clicking on the button Append Value. Enter the address (with mask) of the subnet that is to be denied, for example, `[192.168.10.128/25]`.

8. Select the `service` item. Using Append Value, add `HTTP` to the list.

9. Now we have the new ACL denying access from a subnetwork ready. However, as the ACLs are checked in sequence and the first matching is used, we must exchange our two ACLs. This can be done in several ways. One possibility is to select `acl NET1_DENY` and click on Move Up in its context menu.

10. Finally, we want to restrict the `HTTP` proxy to allow only the `GET` method. Open the `http-proxy HTTP` section by clicking on the small plus sign. Select the `request-acl ALL2` line.

11. Add a `request-method { GET }` item to this ACL in the same way as when you modified values in `acl NET1_DENY`.

12. The configuration should now correspond to Figure 4.22.

13. Save, generate, and apply the configuration by selecting File | Commit the configuration to the firewall. The Commit configuration window is displayed, as depicted in Figure 3.17.

14. Click Commit to enter a RCS log message with a description of the change you have made.

15. The configuration is saved to Kernun UTM, recorded as a new version in the RCS file, the low-level configuration files are generated and applied (copied to the locations where Kernun UTM components will look for them).

16. The Synchronize system with configuration window is displayed, as depicted in Figure 4.23. It informs you that components `DNS` and `HTTP` have been reconfigured and need to be started for the new configuration to take effect.

17. Click OK in the synchronization dialog to finish the reconfiguration; Kernun UTM now contains running DNS and HTTP proxies, see Figure 4.24.

18. Now you can switch to the Log page of the Proxies node in the GKAT window. If you start a Web browser on a machine in the internal network and try to access a Web server in the external network, you will see proxy log messages appearing.

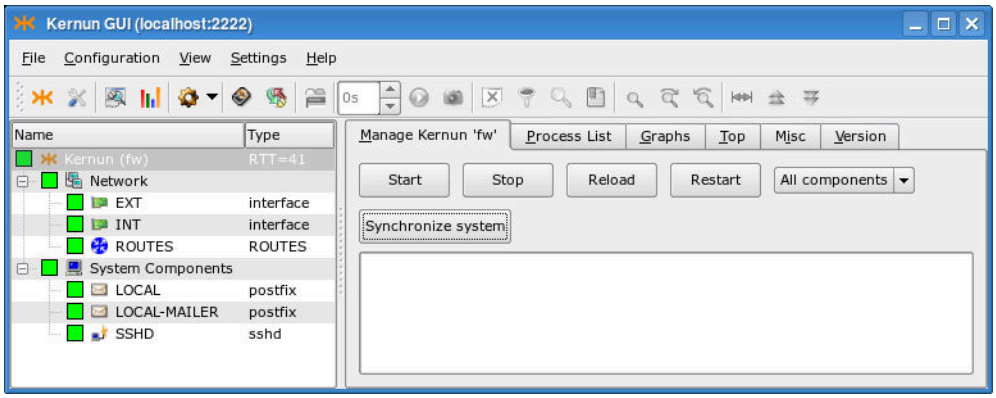

Figure 4.21: The Kernun UTM running the initial configuration

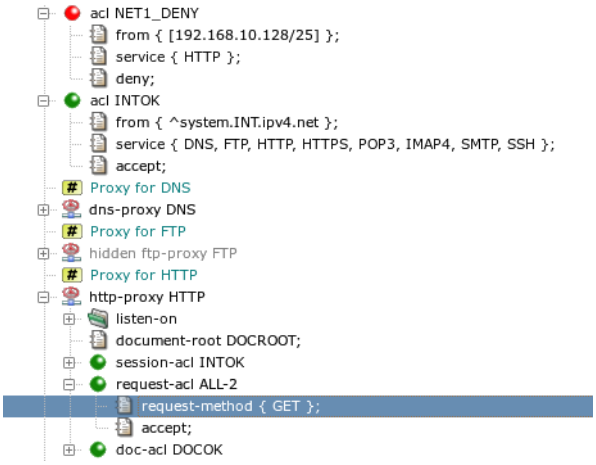

Figure 4.22: The modified configuration

4.3.1 Adding TCP Proxies

The process of adding new TCP proxies can be simplified even more using two wizards (for general information on wizards, see Section 3.1.4). When creating a TCP proxy, you first need to decide whether you want to enable the connection of clients in the internal network to a server in the external network (the server must have a public IP address and the clients connect to this address), or whether you have a server in the internal network (with a private IP address) and clients from the external network will connect to Kernun UTM, which will forward them to the server. The

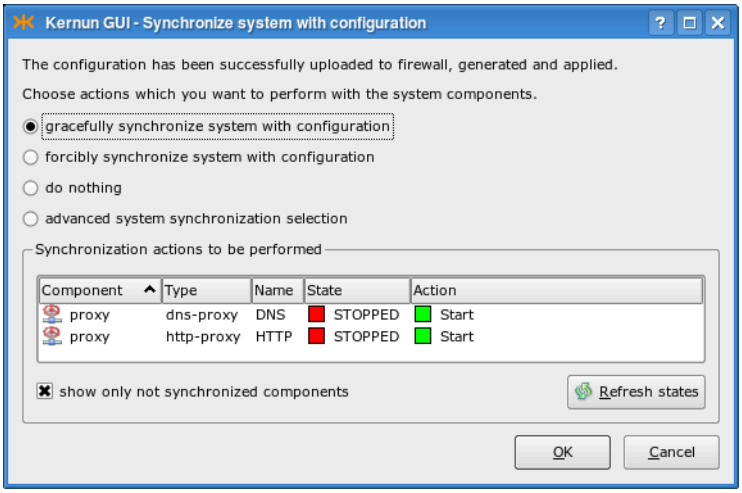

Figure 4.23: Activation of the new configuration

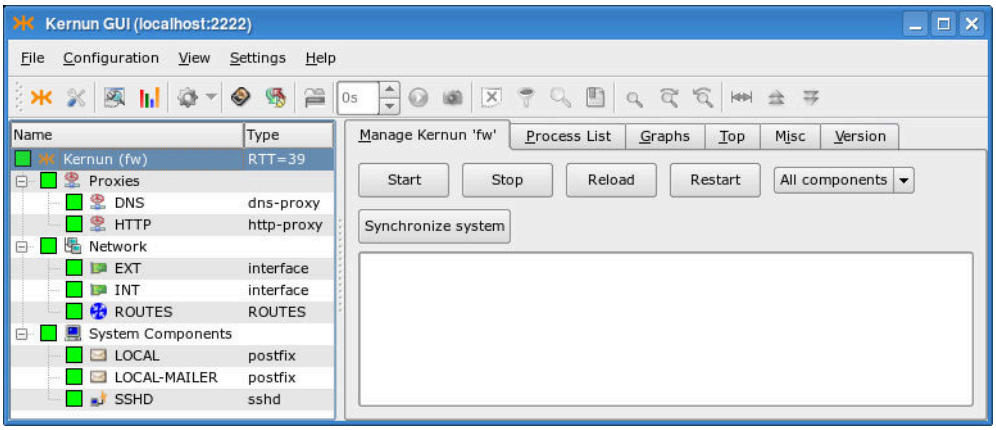

Figure 4.24: Reconfigured Kernun UTM is running

Kernun GUI provides one wizard for each of the alternatives. The following subsections describe the creation of sample proxies in both cases.

TCP Proxy to Server in External Network

As we have said before, a proxy needs to be created in order to allow communication from the internal to the external network. The proxy should be transparent, which means that clients in the local network would not be aware of the fact that they do not connect directly to the server. The TCP proxy to external network wizard will lead you through the creation of the transparent TCP proxy. In this example, we will configure the transparent `tcp-proxy` to allow some privileged local clients to shop on www.ebay.com. First of all, you need to delete or hide the `tcp-proxy HTTPS` from the initial configuration, because it would collide with the proxy we want to create. Then select Insert | Configuration wizard | TCP proxy to external network to start the wizard.

On the first page, you are asked to choose the `tcp-proxy` section name, the interface to the network that contains the clients and the port of the service that is supposed to use the proxy. In our example, we choose the `INT` interface and the `443` port, as shown in Figure 4.25.

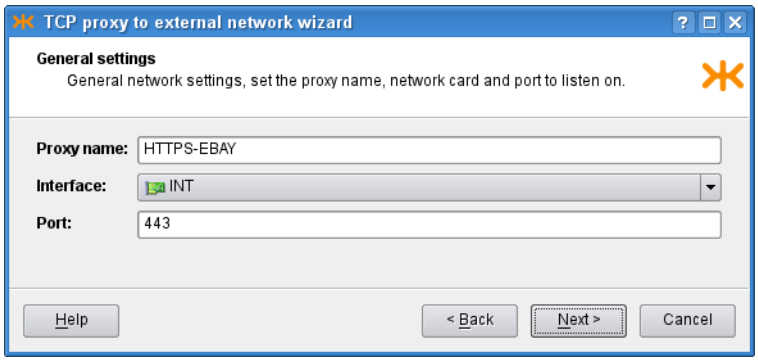

Figure 4.25: TCP proxy general network settings

On the second page, you define the clients in the internal network that are allowed to connect using this proxy and the servers in the external network, to which the clients may connect, in the form of either IP addresses or host names. If you already have an appropriate ACL created in the configuration, you may select the `Use existing accepting ACL` radio button and then choose the ACL to be used for this proxy. As we do not have any usable ACL in the system, however, we choose to create a new one, and set its name and the IP address of the privileged client that is allowed to log in to the ebay.com server. The configuration is depicted in Figure 4.26.

The third page contains more advanced connection settings, including several connection timeouts and a limit of the number of proxy children. If you do not fill in any of these fields, the corresponding item will not be included in the resulting configuration. Since we know that there will be no problems with the traffic load, we leave all the limits empty. In order to get more detailed traffic examination outputs, we choose to monitor the proxy (which i.a. produces graphs of the proxy usage, see Figure 3.8) and to generate proxy statistics (see Figure 5.26). The filled page is shown in Figure 4.27.

Finally, the last page (Figure 4.28) shows the recapitulation of the wizard settings in the form

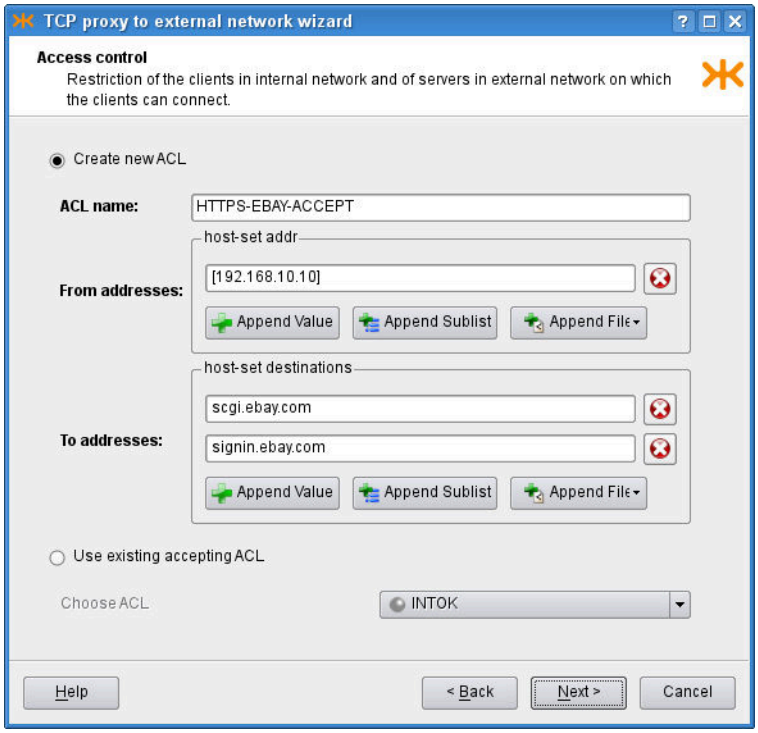

Figure 4.26: TCP proxy ACL settings

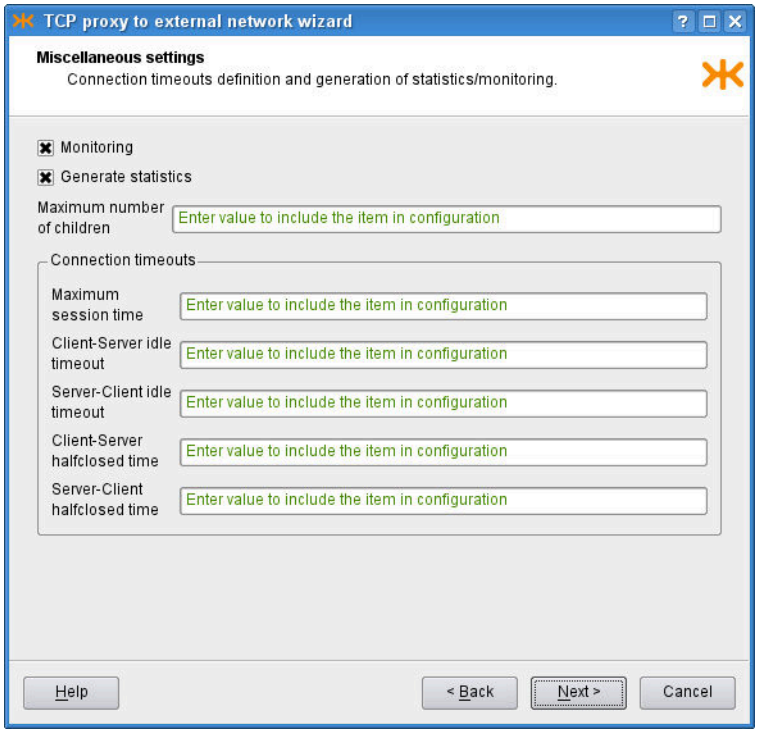

Figure 4.27: TCP proxy miscellaneous settings

of the text configuration that is to be added to the main system configuration. There is also the outcome of system verification of the configuration with the created proxy. You can click `Finish` to commit the proxy into the configuration, `Back` to return and modify the wizard settings, or `Cancel` to close the wizard and discard all the settings.

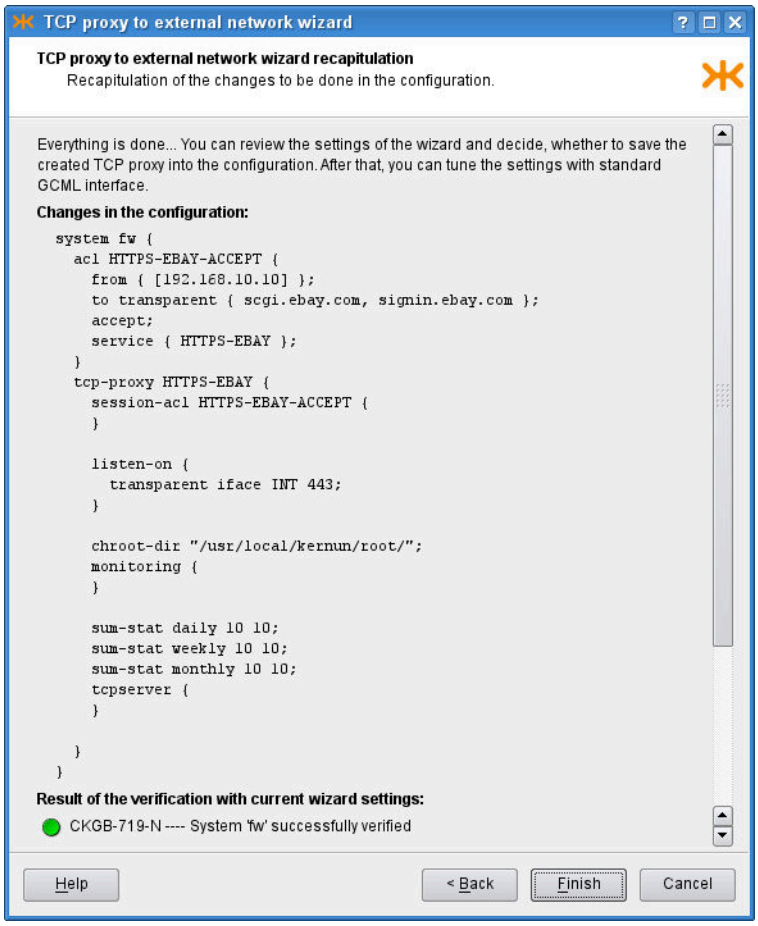

Figure 4.28: TCP proxy recapitulation

TCP Proxy to Server in Internal Network

Besides the above-mentioned wizard for a transparent proxy, the Kernun GUI also provides a wizard for a non-transparent `tcp-proxy`. In this case, clients connect to to a specified port at Kernun UTM, which forwards the communication to a server behind it. This is useful when creating servers without a public IP address in the internal network (which is a recommended practice). We will create an `IMAP4S` proxy that will allow employees at home connect to a mail server in the internal company network behind Kernun UTM and read their mail. The wizard is very similar to the Section 4.3.1 wizard; actually, only the second page differs.

On the first page, we choose the proxy name, the interface on which it is supposed to listen (this time, it is the external interface `EXT`) and the port for secure IMAP4, `993`. On the second page, we similarly define the ACL, with a slight difference in the process of its creation. We only

define restrictions concerning clients, as they always connect to Kernun UTM. After choosing the ACL, we need to specify where to plug the communication in (in our case the IP address of the IMAP4S server) and the port it listens on. This is specified in a single field as `[192.168.10.7] : 993`. Furthermore, we want the server to receive packets with the original client's IP address (rather than that of Kernun UTM), so we check the `Client source address` check box. The second page is shown in Figure 4.29.

The last two pages are the same as in the transparent proxy wizard.

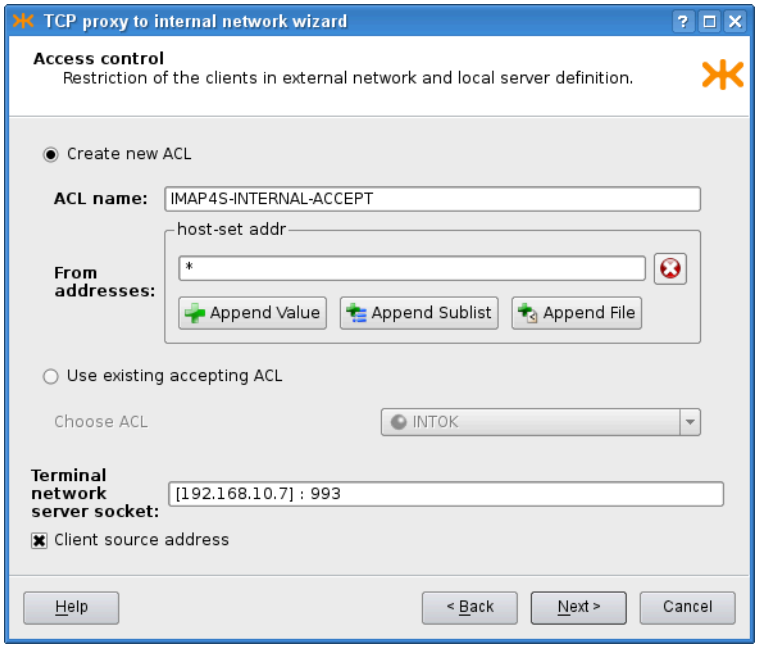

Figure 4.29: Non-transparent TCP proxy ACL settings

Chapter 5

This chapter covers the configuration of various advanced features of Kernun UTM. You can find here instructions on how to configure each feature, along with the description of the most important configuration parameters; the remaining ones are listed in the relevant parts of the reference documentation. It is assumed that you know the principles of the Kernun UTM configuration and how to use the Kernun GUI. If not, consult Chapter 4 and Section 3.1.

5.1 Packet Filter

In addition to the application layer control, Kernun UTM includes a TCP/IP packet filter with advanced features, such as stateful filtering, network address translation (NAT), traffic normalization, traffic shaping, OS fingerprinting etc. This section offers a general overview of the packet filter's capabilities, as well as examples of a few typical packet filter configurations[1].

5.1.1 Packet Flow

It is very important to consider the way packets are handled in Kernun UTM. When combining packet filter rules, traffic shaping and application proxy access control lists, we need to take into account the order, in which network traffic is processed by individual components of the system.

Upon entering the system, network packets get inspected by the packet filter at first. The packet filter itself consists of a number of components. The order, in which these components handle network traffic, is always the same:

1. State engine
2. Traffic normalization engine
3. Traffic shaping / queuing engine
4. Network address translation engine
5. Packet filtering engine

[1] The packet filter is based on *BSD pf*.

This order applies to both incoming and outgoing traffic, i.e. it is the same whether the packet is entering or leaving the system. As regards incoming packets, application proxies may take control only after they are processed by all the packet filter engines. On the other hand, traffic originated at application proxies goes through packet filter engines and then leaves the system for the network.

Important

Network address translation rules always create states. If an initial packet of a connection is translated by an NAT rule and then passed by the packet filter, it is automatically also passed on its way back, thanks to the created state.
As for packet filtering, states are created only if the rules explicitly specify so using the `keep-state` modifier. However, if the `raw` packet filter rule is specified, the state is kept by default; if you do not want to keep the state in such a case, you need to add `no state` to the raw rule.

The packet filtering rules are defined in the `system.packet-filter` section. The following list introduces its items and subsections:

- `set-option` — a repeatable item containing packet filter specific options, see pf.conf(5);
- `timeouts` — a non-repeatable section defining various filter timeouts;
- `altq` — the specification of traffic shaping rules, which assign the traffic to individual queues defined in the `system.pf-queue` sections;
- `scrub-acl` — the traffic normalization rules; by default, all incoming traffic is normalized and IP fragments are reassembled;
- `rdr-acl` — redirection NAT rules, applied to incoming traffic, change the destination address;
- `nat-acl` — mapping NAT rules, applied to outgoing traffic, modify the source address;
- `binat-acl` — bidirectional NAT rules combine both redirection and mapping;
- `filter-acl` — packet filtering, either unidirectional or bidirectional rules (both stateless and stateful);
- `load-anchor` — loading of rule subsets from files.

In the following sections we go through several typical configuration examples that involve capabilities of Kernun UTM's packet filter. We will start with the initial configuration as described in Section 4.2. The resulting packet filter configurations can be found in the configuration sample file `packet-filter.cml` in the `/usr/local/kernun/conf/samples/cml` directory.

5.1.2 Packet Filtering

Packet filtering basically means controlling (either passing, or blocking) network traffic based on basic TCP/IP attributes, including the source and destination IP addresses and ports, the network interface the packet emerges on, its direction (inbound or outbound), and a few other protocol-specific characteristics.

> **Important**
>
> Even if the traffic is passed by the packet filter, there must exist an application proxy with an ACL permitting the communication; otherwise, it is denied. In other words, Kernun UTM does not forward network packets, but instead, it attempts to transparently grab them and process them with application proxies. However, the mechanism of traffic grabbing by application proxies can be bypassed, as described in Section 5.1.4.

> **Note**
>
> By default, the packet filter rules allow all traffic, but there is only a limited set of application proxies in the initial configuration, see Section 4.2.

Blocking traffic using the packet filter may be useful in many situations. For example, we may relieve application proxies of the burden of processing traffic that we know for certain is undesired. Also, application proxies always grab connections, and only then they may selectively deny them. This means that the connection is always established at first, and then immediately closed if denied by a policy. It may be advantageous to pretend to some clients that there is no application proxy in the way, which can be achieved by resetting those connections using the packet filter. Furthermore, there are antispoofing rules to block traffic with faked source addresses, see Section 5.1.3, and it is possible to bypass the application proxy processing and forward some traffic directly to its destination, see Section 5.1.4.

Individual packet-filtering rules are located in `filter-acl` subsections within the `system.packet-filter` configuration section. The most important items in `filter-acl` are summarized here:

- `from` — A set of hosts, addresses and networks that the packet's source address must match. Optionally it may also include constraints concerning source TCP/UDP ports. If omitted, all source addresses and ports match.

- `to` — The set of hosts, addresses and networks that the packet's destination address must match. Optionally it may also include constraints concerning destination TCP/UDP ports. If omitted, all destination addresses and ports match.

- `iface` — The network interface that this rule applies to. Moreover, the direction of communication may be specified, either `in` for inbound traffic or `out` for outbound traffic. If this item is not present, the rule applies to all network interfaces and all directions.

- `protocol` — The IP protocol that this rule applies to. Protocols are accepted with their symbolic names, such as `icmp`, `udp`, `tcp`, or `esp`. Moreover, a shortcut `tcp-udp` has been added for user comfort, meaning both the TCP and UDP protocols. Similarly, the shortcut `esp-ah` is interpreted as both the ESP and AH protocols[2]. Additional protocol-specific parameters are available for the ICMP and TCP protocols, specifically `icmp-type` to match the ICMP message type, and `flags` to match the TCP flags field.

- `deny / accept` — The `deny` item blocks traffic; the `accept` item is used to pass it.

- `keep-state` — This item lets the packet filter create a state for the connection as a packet is passed. The following packets in the same connection will then be handled in exactly the same way as the first one, without the need to search the ruleset. Another advantage is that packets of the same connection in the opposite direction are implicitly passed.

Important

Port specification is available only for the TCP and UDP protocols. Thus, if a port constraint is present in a `from` or `to` item, the `protocol` item must be specified and must be one of `tcp`, `udp`, or `tcp-udp`.

Example: If we want to block the traffic coming in on our external interface to the TCP port 22, we shall use the `to`, `iface` and `deny` items, as shown in Figure 5.1.

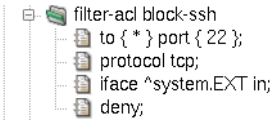

Figure 5.1: A simple blocking packet filter rule

The `deny` part of the rule means that Kernun UTM will silently discard packets coming in to port 22. However, this behavior is not very effective, for several reasons. First, everyone knows that there is a filter blocking those connections, and that can attract unwanted attention. Furthermore, the standard application will not give up if there is no response to its connection attempts. Therefore, it is customary to send back information that the port is closed. We will do so by adding a `return` item to our filter rule, see Figure 5.2.

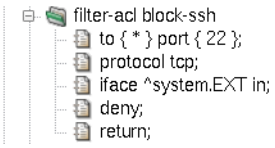

Figure 5.2: A blocking packet filter rule with `return`

[2] The ESP and AH protocols are both a part of the IPsec protocol family.

Tip

We can change the packet filter's default behavior of silently discarding packets by setting the `block-policy return` option. If we do so, it will properly react to blocked ports as if those ports were closed even if there are no `return` items in individual rules. See an example in Figure 5.3.

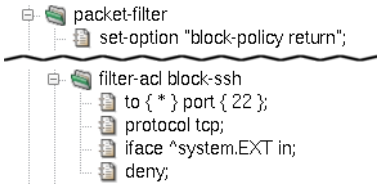

Figure 5.3: Option `block-policy` instead of `return` in rule

Should some specific clients be allowed to connect to port 22, we have to add a packet filter rule *before* the blocking rule we have just created. Rule evaluation abides by the so-called *first-match* principle. This means that rules are evaluated in the order, in which they appear in the configuration, and as soon as a matching rule is found, the evaluation stops and the remaining rules are ignored. Therefore, more specific rules must precede those with more generic matching criteria. Figure 5.4 shows how to add a rule allowing connections to port 22 to a set of IP addresses, preserving the default behavior of blocking port 22 to other clients.

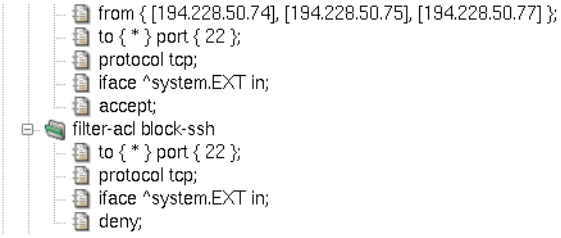

Figure 5.4: More specific rule must come first

5.1.3 Antispoofing Using Packet Filter

IP address spoofing is an attack based on counterfeiting the source IP address in order to confuse another computer system. It may be extremely dangerous if attackers from the outside pretend to have an internal source IP addresses; although they never get a response back, it may be sufficient to perform a successful denial-of-service or another kind of attack.

The goal of antispoofing is obviously to prevent spoofing attacks. We can stop intruders from the outside who pretend to have an internal source IP address quite easily. In general, internal network addresses can appear as the source of communication only on the internal network interface. As there may exist more than one protected network interface, this rule can be applied to other networks and interfaces as well.

A simple antispoofing rule consists of interface specification followed by the `antispoof` and `deny` items, see Figure 5.5. In effect, the internal network `192.168.10.0/24` is blocked when it appears as a source IP address on any other interface than `INT` [3].

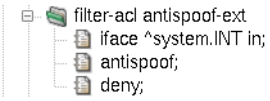

Figure 5.5: Simple antispoofing rule

This simple antispoofing rule works for networks directly connected to named interfaces. However, if our internal network is not flat, but consists of several routed networks instead, we need to involve all the internal networks in antispoofing. This can be achieved using the `routes` modifier in the `antispoof` item. The resulting rule is depicted in Figure 5.6.

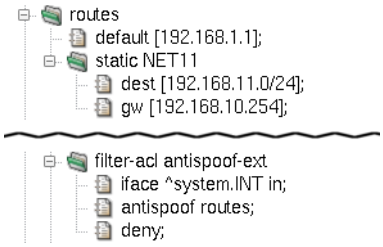

Figure 5.6: Antispoofing rule including routes

Note

The `routes` modifier has an effect only if there are some routes within the internal network. To illustrate this fact, the sample configuration file `packet-filter.cml` introduces a second internal network, `192.168.11.0/24`, specified in the `routes` section. Now, both our internal networks, `192.168.10.0/24` and `192.168.11.0/24`, get blocked in the source address field on all interfaces except the internal interface `INT`.

5.1.4 Selective Packet Forwarding

Standard routers and filtering gateways accept all network datagrams, and if they are destined for another host, they send them out in accordance with the system's routing table. This mechanism is known under the name *forwarding*.

Kernun UTM does not forward network packets by default. Only traffic either destined for the system itself or grabbed transparently by application proxies will find its way through; everything else is thrown away. See transparency(7) for more detailed information.

It is possible to bypass application proxies and control the communication only with packet filter rules. To do so, we need to inform the transparent grabbing system which packets should be

[3] The internal network is taken from the definition of interface `INT`.

left untouched. For that purpose, a special tag `NOTRANSP` has been introduced.

Note

Tagging is a feature of the packet filtering engine; network packets can be assigned a string value that will accompany those packets on their way through the network stack. Other Kernun UTM components may then check which tags, if any, are assigned to traffic they are processing.

Important

The tag name `NOTRANSP` that the transparency engine uses to recognize bypassing packets is configurable. By changing kernel sysctl variable `net.inet.ip.no_transp_tag`, we can define another tag string to be used to distinguish between standard transparent proxy traffic and bypassing datagrams. Sysctl variables (also called MIBS) are configured in `system.sysctl` configuration section, see Section 5.2.6.

To assign a tag to packets, add a `tag` item to a `filter-acl` rule inside the `packet-filter` section. Figure 5.7 illustrates a rule causing the packet flow between two hosts to bypass transparent proxy processing, forwarding them directly to the network in accordance with the system routing table. Note that we have introduced a new interface, DMZ, representing a demilitarized zone with public accessible servers. The rule `bypass-int-dmz` permits bidirectional traffic between an internal host and a host in the DMZ.

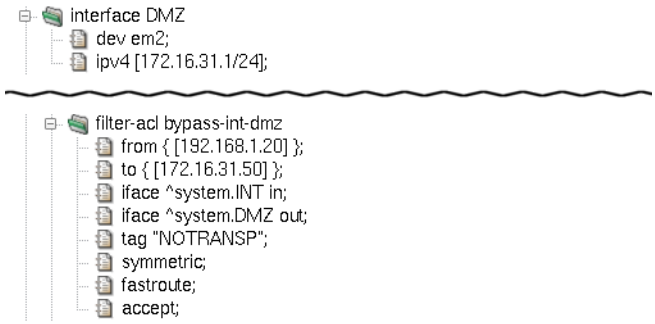

Figure 5.7: Selective packet forwarding rule

Apart from `tag`, two more important `filter-acl` items are introduced in the sample rule `bypass-int-dmz` depicted in Figure 5.7: :

- `fastroute` — This option means that packets get forwarded through Kernun UTM to their destination. It is called *selective packet forwarding*, as opposed to *global forwarding*, which is performed by the standard routers and packet filtering gateways. Without `fastroute`, packets tagged `NOTRANSP` would not reach their destinations.
- `symmetric` — Adds a second rule, allowing traffic in the opposite direction on the same interface. Source and destination IP addresses are swapped in the second rule, as well as the

traffic direction `in` or `out`. Considering the fact that we have two interface specifications in the rule, we end up with four individual packet flow permissions:

1. Incoming packets on interface `INT` going from `192.168.1.20` to `172.16.31.50` (the basic rule for `iface ^system.INT in`).
2. Outgoing traffic on interface `INT` returning back from `172.16.31.50` to `192.168.1.20` (the symmetric rule for `iface ^system.INT in`).
3. Outgoing packets on interface `DMZ` originated at `192.168.1.20` and destined for `172.16.31.50` (the basic rule for `iface ^system.DMZ out`).
4. Incoming datagrams on interface `DMZ` traveling from `172.16.31.50` to `192.168.1.20` (the symmetric rule for `iface ^system.DMZ out`).

Important

If a packet filter rule sets the `NOTRANSP` tag for a packet, a state is automatically created for the packet. This accepts all following packets of the same connection in both directions. If we want to selectively forward some communication via the `NOTRANSP` mechanism without creating a state, we need to add an explicit rule that matches the packets and does not contain `keep-state`.

5.1.5 Network Address Translation

There are three types of NAT rules: mapping rules (`nat-acl`), redirection rules (`rdr-acl`) and bidirectional NAT rules (`binat-acl`).

Mapping Rules

Mapping changes source IP addresses (and often ports) of outgoing packets. It always applies to outbound traffic, but it also creates states for backward incoming communication. The state engine fully recognizes individual TCP connections, UDP sessions and ICMP control messages that belong to them. Hence, if a state is created, only legal communication is passed and translated forth and back.

The `nat-acl` sections allow for a similar set of items as `filter-acl` rules. The item `from` is used to match source IP addresses and ports, similarly `to` is matched against destination IP addresses and ports. The interface specification may not include the `in/out` direction as mapping rules apply only to outbound traffic. The `deny` modifier does not block traffic, but effectively denies any NAT mapping if matched.

A new important item is introduced for mapping rules: `map-to`. Its purpose is to specify the final address and port combination after the translation. A sample mapping rule is depicted in Figure 5.8. It illustrates the use of the `map-to` item; it specifies an IP address (using a reference to the outgoing interface's address `^system.DMZ.ipv4.host`) and a port (0 in our example, meaning any port number available).

As always, mapping rules are implemented using the first-match principle, i.e. the first matching rule is applied immediately, without consulting the rest of the `nat-acl` rules.

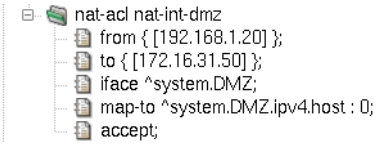

Figure 5.8: Mapping NAT rule

Redirection Rules

Unlike mapping, redirection deals with destination IP addresses and ports. Redirection rules are thus applied to the inbound traffic, creating states. The same powerful state engine is in charge of matching backward outgoing packets and changing their addresses and ports back to their original values.

Apart from the target redirection address and port combination, which is specified using the `rdr-to` modifier, all other item names and features are the same in mapping and redirection rules. The example in Figure 5.9 assumes connections from the `172.16.31.0/24` network, destined for the DMZ interface's local address `172.16.31.1` and port 80. Those connections get redirected to internal server at `192.168.1.20`, port 80.

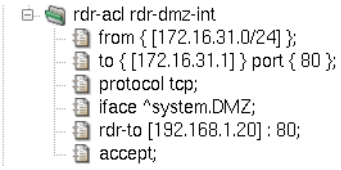

Figure 5.9: Redirection NAT rule

Bidirectional Rules

Bidirectional NAT rules are not yet fully supported by the CML language. The `binat-acl` section accepts only raw specifications of rules, in accordance with the pf.conf(5) manual page.

5.1.6 Packet Forwarding along with NAT

Imagine that we have an NAT network and want to bypass Kernun for some traffic (e.g. ICMP packets, in order to be able to ping to the internet from the local network). For that special case we need to create an NAT rule for the NAT and, at the same time, tag the traffic with the `NOTRANSP` tag to forward it, rather than give it to Kernun's proxies. The NAT rule automatically creates a state, so the reply to the ping should be delivered to the requester without the need to add any other rule.

However, there is a catch in the PF implementation. The NOTRANSP tag in combination with NAT rule gets lost and the returning packet is passed to Kernun, rather than forwarded. For this case, Kernun automatically generates a rule `pass any to any tagged NOTRANSP no state tag NOTRANSP` in the pf.conf file, in order to keep the tag. This rule permanently stores the NOTRANSP tag to the state of the tagged packet and is not applied to any other packets.

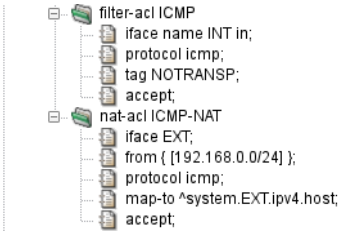

Figure 5.10: Forwarding of ICMP Packets over NAT

Figure 5.10 shows a configuration of selective forwarding of ICMP packets on Kernun UTM for clients behind NAT. The `filter-acl ICMP` section tags the packet by the NOTRANSP tag to be passed through Kernun without giving it to the proxies, while the `nat-acl ICMP-NAT` rewrites the addresses and creates a state for the reply packets to be passed back. The returned packets are first NATed, then the above-mentioned rule is applied and restores the NOTRANSP tag, and the packet is therefore forwarded into the local network.

5.1.7 Defending against DoS/DDoS Attacks

The packet filter, together with the network stack in the operating system kernel, provide some means for defense against Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. Such attacks try to overload a target computer system or network by sending huge amount of traffic. A DoS attack is originated from a single malicious computer. A DDoS attack is similar, but data are sent by many computers at the same time. It allows the attacker to magnify the number of network packets many times in comparison with a single-origin DoS, hence making the effect on the target network worse and any defense harder.

Basic protection against some (D)DoS attacks on the transport layer of the TCP/IP is built into the network stack of the operating system kernel in form of the *SYN cache* and *SYN cookies*. They are effective especially against the SYN flood attack, when the attacker sends many TCP connection requests in the form of TCP SYN segments. The SYN cache keeps information about TCP connection handshakes that have not been completed yet. A SYN cache entry occupies less memory than the full state record of an established TCP connection. Hence the system is able to withstand much more SYNs. SYN cookies take one step further, keeping no state and encoding all information necessary to complete the handshake into the SYN/ACK segment sent to the client.

The SYN cache is always enabled. By default, SYN cookies are also enabled. They can be disabled by setting the sysctl variable `net.inet.tcp.syncookies=0`, see Section 5.2.6 for instructions on setting sysctl variables. SYN cookies are used when the SYN cache becomes full. It is possible to disable the SYN cache and use only SYN cookies by setting the sysctl variable `net.inet.tcp.syncookies_only=1`.

The SYN cache and SYN cookies protect only against SYN flood attacks on TCP-based application protocols handled by a proxy or by a server running locally on the Kernun system. Additional defenses are provided by the packet filter. They are effective for communication handled by the packet filter and not passing via any proxy, but can be combined with a proxy, too. They can also block attacks that perform full TCP handshake and then send excessively large

volumes of application-layer data in order to overload a server.

The packet filter allows limiting numbers of simultaneous connections that match a filtering rule or originate from a single source addres. The limits are configured by adding per-rule options. There are two variants how to create such packet filter rules:

- A `filter-acl` is created with item `keep-state`. A raw option is added by item `option` containing limit specifications delimited by comma, for example, `option "keep state (source-track rule, max-src-nodes 100)"`. Note that "`keep state`" is specified here, in addition to the separate `keep-state` item.
- A `filter-acl` is created containing the whole packet filter rule written in a `raw` item, for example:

```
pass quick inet proto tcp from any to any  keep state (max 100)
```

Available limit specifications are:

max *number* It limits the maximum number of simultaneous states (connections) the rule may create. When this limit is reached, further connection attempts are silently dropped. New connections are allowed only after some of the existing states time out. Note that a state times out some time after the related connection is closed.

source-track rule Enables counting the states created for each individual source IP address. The per-IP limits (e.g., `max-src-nodes` and `max-src-states`) are compared to the number of states created by this rule.

source-track global Enables counting the states created for each individual source IP address. The per-IP limits are compared to the sum of states created by all rules that use this option.

max-src-nodes *number* It limits the maximum number of distinct IP addresses that can have states at the same time.

max-src-states *number* It limits the maximum number of states that can be created for a single source IP address.

max-src-states *number* It limits the maximum number of established TCP connections that can be created for a single source IP address. In contrast to `max-src-states`, this option counts only connection that completed the 3-way TCP handshake.

max-src-conn-rate *number* / *seconds* It limits the rate of establishing new TCP connections over a time interval.

overload <*table*>

overload <*table*> flush

overload <*table*> flush global If a source IP address reaches one of the limits `max-src-conn` or `max-src-conn-rate`, it will be added to a named packet filter

table. If `flush` is used, all states created by the matching rule and originating from this IP address will be deleted, effectively terminating all existing connection from the offending IP address. If `flush global` is used, all states from this IP address are deleted, regardless the rule that created them.

Example: The following rules will block any IP adress that initiates more than 100 HTTP connections per second.

```
table <dos_attack> persist
block quick from <dos_attack>
pass in proto tcp from any to any port 80 keep state \
    (source-track rule, max-src-conn-rate 100/1, overload <dos_attack> \
    flush global)
```

5.1.8 Honeypot

...

5.2 System Configuration

5.2.1 User Accounts

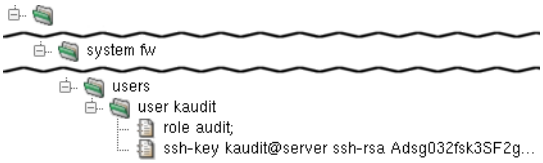

Figure 5.11: User account configuration

Additional user accounts can be allowed access to the system in the configuration. Each user is represented by a `user` section within the `users` section. The `user` section must contain a `role` item with a value `admin` or `audit`. The `admin` class has full access to the system and permissions to configure and operate all Kernun UTM components. The `audit` class can only view the system configuration and logs. The `user` section can also contain a `ssh-key` item, which generates a corresponding line in the `.ssh/authorized_keys` file for the user's ssh key.

5.2.2 Network Interfaces

The Kernun UTM configuration can hold any number of `interface` sections that correspond to a physical device. Virtual network interfaces can be created from other tools than Kernun UTM configuration utilities; these, however, need to be referenced in the Kernun UTM configuration. For example, an instance of **openvpn** creates the `tun0` interface when started. To be able to use it in the Kernun UTM configuration, include the `virtual` parameter in its `dev` item (as illustrated in the `interface VPN-PRAHA` section in Figure 5.12). Such an interface is not generated in the `rc.conf` file.

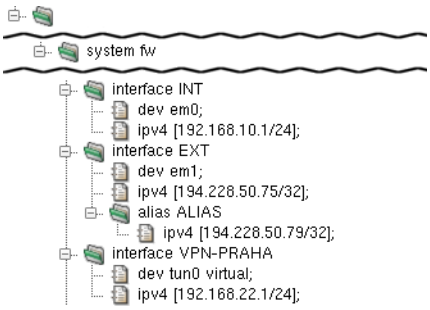

Figure 5.12: Network Interfaces

An interface can be assigned multiple IP addresses. To do so, add an arbitrary number of `alias` sections to the `interface` section. Each of them must contain an `ipv4` item that defines its IPv4 address, as depicted in Figure 5.12 in `interface EXT`.

See interface(5) for details.

5.2.3 Static Routes

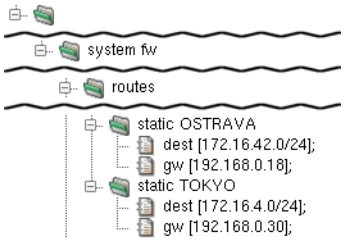

Figure 5.13: Static Routes

Static routes provide the capability to explicitly route packets for a given network to a static machine, which works as a gateway for this network when this cannot be done automatically by the system routing table management daemon (such as `routed`). See route(8) for further info.

A static route is set by a `static` section in the `routes` section. The `static` section must contain one `dest` item — the routed network address — and a `gw` item — the address of the gateway machine for the routed network.

5.2.4 Dynamic IP routing with BIRD

Kernun UTM supports dynamic IP routing via BIRD's (The BIRD Internet Routing Daemon) implementation of OSPF (Open Shortest Path First) protocol. It allows routers to automatically change routes, so that the path remain functional in case of failure of an router in the path.

BIRD uses separate configuration for IPv4 and IPv6. In the example below, `bird4` defines rules for IPv4 routing of one Kernun UTM in the role of router. This router would be one of multiple routers in network, either Kernun UTM or any other device that supports OSPF. For example, two clusters of Kernun UTM with all nodes connected with each other via VPN.

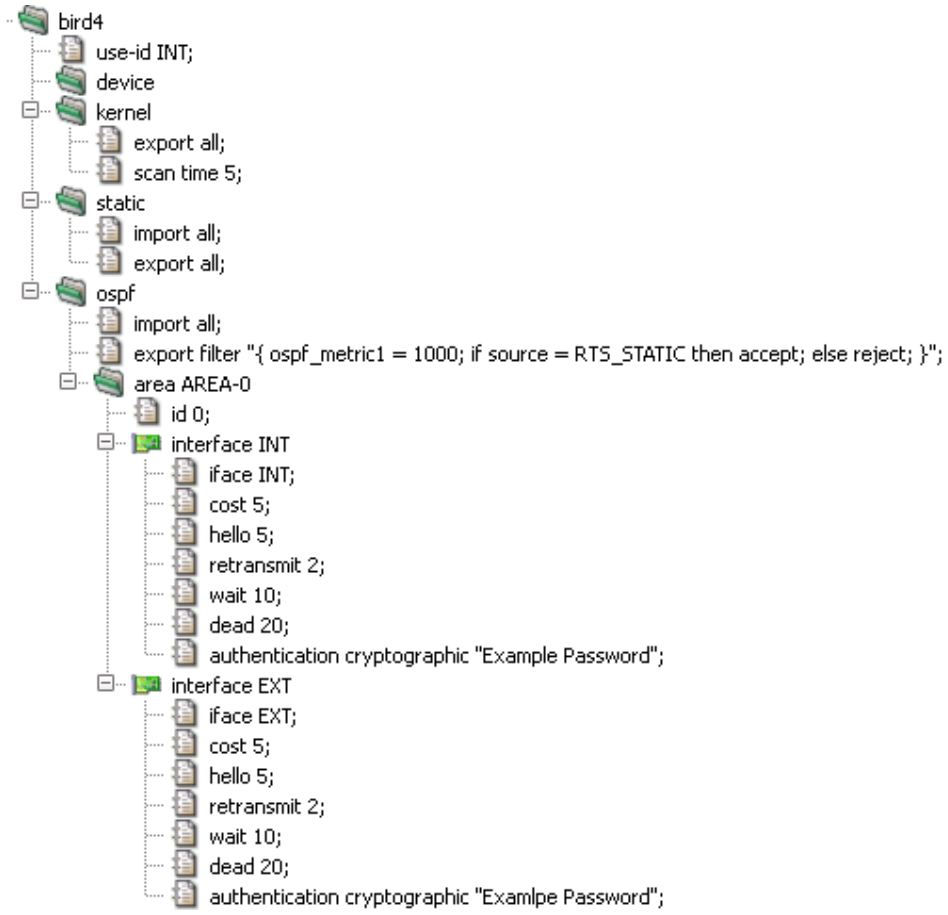

Figure 5.14: BIRD dynamic routing

BIRD, or more precisely OSPF, requires each router to be identified by an unique ID. `use-id` in the example above, uses IPv4 address of interface INT as this ID. The sections `device`, `kernel`, `static` and `ospf` defines different protocols (or pseudo-protocols) which BIRD should use for import or export of routes. The `device` protocol is not a real routing protocol. It doesn't generate any routes and it only serves as a module for getting information about network interfaces from the kernel. Except for very unusual circumstances, you probably should include this protocol in the configuration since almost all other protocols require network interfaces to be defined for them to work with. The `kernel` is also a pseudo-protocol. Instead of communicating with other routers in the network, it performs synchronization of BIRD's routing tables with the OS kernel. Basically, it sends all routing table updates to the kernel and from time to time (item `scan`) it scans the kernel tables to see whether some routes have disappeared or whether a new route has been added by someone else. The section `static` analogically sets rules for static routes defined in `system -> routes -> static`.

The `ospf` section defines, besides import/export rules, one or more areas (`area`). In OSPF, the network is divided into areas that are logical groupings of hosts and networks. Each area maintains a separate link state database whose information may be summarized towards the rest of the network by the connecting router. Thus, the topology of an area is unknown outside of the area. This reduces the routing traffic between parts of an autonomous system. An area is identified by it's `id`. "O" in this example, identifies backbone area (or area 0.0.0.0) which forms the core of an OSPF network. In `interface` sections we assign different properties for selected network interfaces. Especially, the `cost` item. The OSPF routing policies for constructing a route table are governed by link cost factors associated with each routing interface. Cost factors may be the distance of a router (round-trip time), data throughput of a link, or link availability and reliability, expressed as simple unitless numbers. Other items defines different rules for communication with other routers via given network interface.

For more information on BIRD configuration see `http://bird.network.cz/?get_doc&f=bird-3.html`

5.2.5 File `/etc/rc.conf`

The `/etc/rc.conf` configuration file is used to automatically perform various actions at the system startup. It contains information about the configuration of network interfaces, the services that should be started after a boot and optionally their parameters, the configuration of the machine's host name, console settings, etc. See rc.conf(5) for details.

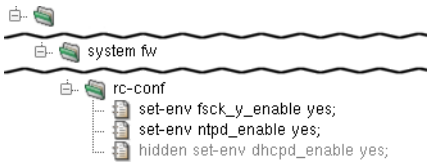

Figure 5.15: `rc.conf` configuration

The `rc.conf` file is generated by Kernun UTM and should therefore not be configured directly, but in `rc-conf` section of the Kernun UTM configuration. An entry in `rc.conf` is set through

the `set-env` item. Examples of useful items include `set-env fsck_y_enable yes` (which will make **fsck** answer itself "YES" to all questions and thus make automatic OS booting possible).

It is even possible to extend an already set value by redefinition, such as `set-env variable "$variable ..."`.

5.2.6 Kernel Parameters in `/etc/sysctl.conf`

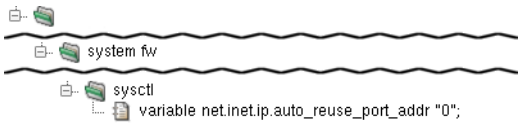

Figure 5.16: `sysctl.conf` configuration

The `/etc/sysctl.conf` file is read when the OS enters the multi-user mode and sets default settings for the kernel. Its contents are generated from the `variables` items in the `syctl` section. Each `variable` sets one kernel setting. For example, `variable net.inet.ip.forwarding "1"` turns on IP forwarding.

5.2.7 Configuration of the cron Daemon

The `crontab` is the configuration file for the **cron** daemon, which runs selected commands periodically. Each entry in `crontab` contains seven fields separated by a white space character:

- *minute* — permitted values: 0-59
- *hour* — permitted values: 0-23
- *day of month* — permitted values: 1-31
- *month* — permitted values: 1-12 (or names)
- *week* — permitted values: 0-7 (0 or 7 is Sun, or use names)
- *user* — user under whom the program should be run
- *command* — the command to be run and its parameters

A field may be set to "*", which stands for its entire range. Numerical ranges, lists and their combinations are also allowed. For example, "3-7", "1,2,7,8", "1-3,8-14,20". Step values and more complex expressions can be used as well, see crontab(5) for details.

There is a sample crontab configuration in `samples/include/crontab.cml`. It can be included in `crontab` by the application of the `$default-crontab` variable, which is set in `include samples/include/crontab.cml`.

Additional entries can be set by `plan` items in the `crontab` section. For example `plan "1 2 * * * root /usr/local/kernun/bin/upgradeFBSD.sh"` will schedule the script `upgradeFBSD.sh` (which automatically synchronizes and compiles the OS source) to be run every day at 02:01 under the user root.

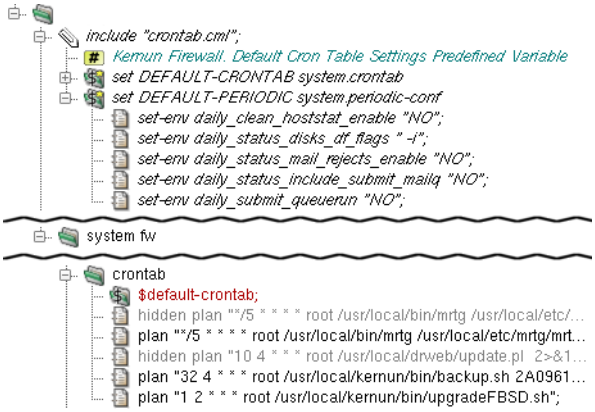

Figure 5.17: `crontab` configuration

5.3 Caching Name Server

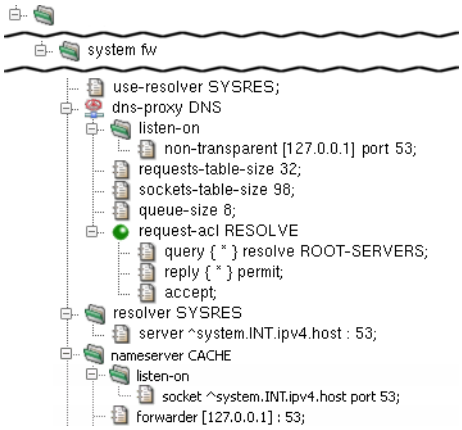

Figure 5.18: Caching Name Server configuration

Kernun UTM's `dns-proxy` is not designed to be used as a name server — it does not cache DNS queries. A possible solution is a combination of **named** and **dns-proxy**. In this scenario, **named** listens for DNS queries on the internal interface and provides the cache. It queries **dns-proxy** that is bound to the loopback interface and, in accordance with ACLs, permits or denies the query, sends a response or queries the DNS root servers.

The configuration in Figure 5.18 shows the **named** daemon configured in the `nameserver` section to listen on Kernun UTM's internal address on port 53 (`listen-sock ^system.INT.ipv4.host : 53`), while Kernun UTM uses it as its resolver (`server ^system.INT.ipv4.host : 53` in the `resolver` section). `dns-proxy` is bound to the loopback interface by the `non-transparent [127.0.0.1] : 53` item in the `listen-on` section of `dns-proxy`. See named.conf(5) for more details.

The other typical scenario is that one or more name servers exist in the internal network. In this situation, clients are configured to query the server in the internal network, which queries

dns-proxy that is configured to listen on the internal address, while Kernun UTM itself uses the internal name server as its resolver.

Note that in both of these scenarios it is necessary to have multiple name servers running in order to provide different DNS responses for different clients, because the response is cached on the name server and therefore not matched against the ACLs of **dns-proxy**. Nevertheless, it is always possible to plug requests coming from particular clients to a host with a different IP address, ignoring the DNS name in the request for every service.

5.4 DNS and DHCP Services

5.4.1 DNS Server for the Local Zone

In this scenario, **named** listens for DNS queries on the internal interface and queries **dns-proxy**, which provides the response.

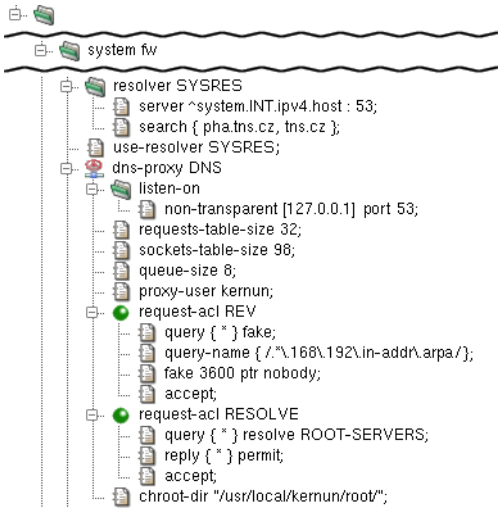

Figure 5.19: DNS Server - Proxy configuration

The configuration in Figure 5.19 shows Kernun UTM using **named** as the resolver (`server ^system.INT.ipv4.host : 53` in the `resolver` section). `dns-proxy` is bound to the loopback interface by `non-transparent [127.0.0.1] : 53` in the `listen-on` section of `dns-proxy`. See Section 5.3 for further information.

In the configuration depicted in Figure 5.20, the **named** daemon is configured in the `nameserver` section to listen on the internal address on port 53 (`listen-sock ^system.INT.ipv4.host : 53`) and to forward the request to **dns-proxy** on the loopback interface (`forwarder [127.0.0.1] : 53`).

Finally, there is the `zone pha` section. The `name pha.tns.cz` item assigns its name and the `generate` item makes KGB generate the zone data from `hosts-table`. The `reverse` item makes `zone pha-reverse` provide reverse DNS records for the local network.

The `hosts-table` section contains multiple `host` items, each defining the host name (e.g. "builder.pha.tns.cz") and the IP address (e.g. "192.168.1.101") for a certain MAC address (e.g.

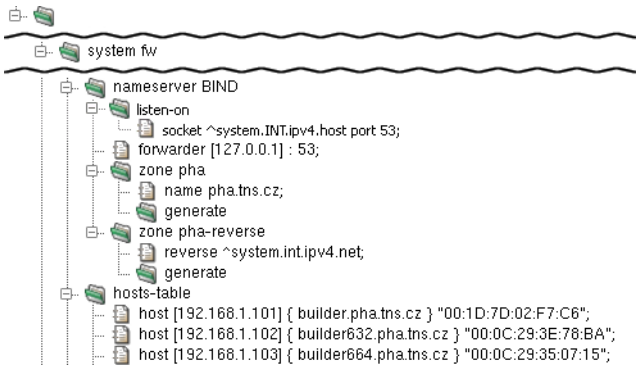

Figure 5.20: DNS Server - BIND configuration

"00:1D:7D:02:F7:C6").

It is possible to globally force the clients to use SafeSearch functionality for Google, YouTube and Bing by using `samples/include/safe-search.cml`.

5.4.2 DHCP Server for the Local Network

The Dynamic Host Configuration Protocol (DHCP) is used by a client to obtain information necessary to connect to an IP network automatically, with no need of manual administration. This information includes the client's IP address, network mask, default gateway, DNS server address, etc.

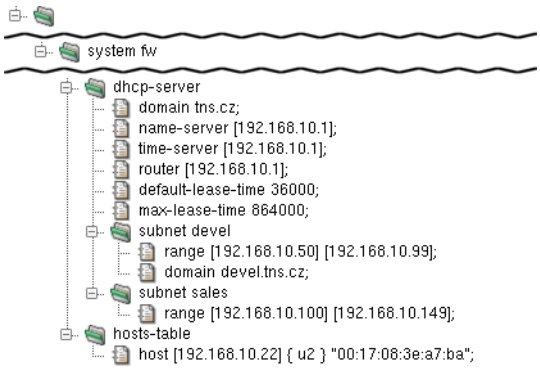

Figure 5.21: DHCP configuration

The DHCP server is configured in the `dhcp-server` section. In this example, the DNS server address pushed by the DHCP server is 192.168.10.1 (`name-server` item). The lease time is set to 10 hours (`default-lease-time`) and the maximum lease time to 1 day (`max-lease-time`). Furthermore, there is a setting for the domain name (`domain`), the router address (`router`) and the NTP server (`time-server`). There is a range of IP addresses (`range [192.168.10.50] [192.168.10.99]`) reserved for `subnet devel`, which also has the domain name altered to (`domain devel.tns.cz`). The addresses assigned by DHCP server will therefore by between 192.168.10.50 and 192.168.10.99.

The `hosts-table` section defines a single host with a predefined IP address: the host *u2* with the MAC address 00:17:08:3e:a7:ba is assigned the IP address 192.168.10.22 (`host [192.168.10.22] { u2 } "00:17:08:3e:a7:ba"`).

See dhcp-server(5) for details.

5.5 Time Synchronization with NTP

It is very important to keep the correct time and date on all computer systems, including firewalls, internal servers, routers and even workstations. Kernun UTM provides a time synchronization function by means of an NTP server.

Kernun UTM's NTP server allows two functions: to synchronize the local time with a remote NTP server, and to serve this time data to the local systems. It uses the NTP protocol version 4, but retains compatibility with versions 3, 2 and 1 of the NTP protocol. It can play the client and server role at the same time. Thus, in a typical scenario, Kernun UTM's NTP server exchanges NTP messages with a few public time servers in order to keep its own time and date synchronized, while offering time synchronization to the whole local network.

Note

Local time zone of the Kernun UTM system is set in the console during the first boot, see Section 2.5.2.

From now on, we will assume that the initial configuration file is loaded in the GUI, as shown in Section 4.2. To define NTP server parameters, we add a new `ntp` section at the `system` level, for instance after the `ssh-keys` section. As a minimum, we need to define one server in an item within the `ntp` section, `ns2.tns.cz` in our example. Furthermore, we want to let the internal systems synchronize their time with Kernun UTM, which is achieved by adding an extra `restrict` item. We use a reference `^system.INT.ipv4.net` to specify our local network; in more complex topologies, we would have to repeat the `restrict` item and specify internal networks explicitly or using variables. The `nopeer` and `noquery` flags of the `restrict` item are used to allow only client synchronization requests. Figure 5.22 shows the relevant part of configuration.

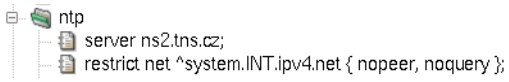

Figure 5.22: Minimum NTP server configuration

Reliance on a single external time server may lead to time synchronization outages. Therefore, it is quite common to use more than one time server. The `server` item can be repeated, which results in a stabler configuration. Moreover, if there are more firewalls or parallel network servers, it is sometimes beneficial to let them know about each other's NTP server, acting together as peers. An NTP peer is not an authoritative source of time data, but may serve for minor time corrections upon Internet blackouts. We achieve this effect by specifying another NTP server with a `peer` item within the `ntp` section, as shown in Figure 5.23.

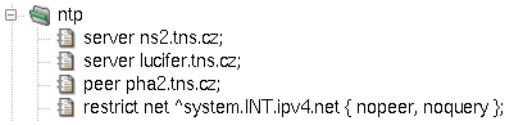

Figure 5.23: Peer for NTP server

If the `drift-file` item is used, Kernun UTM's NTP server attempts to compute the error in the intrinsic frequency of the local on-board clock. The item must be accompanied with a filename, e.g. `/var/ntp.drift`. The NTP server also includes support for local reference clocks, if available.[4] The on-board system clock itself may be used as a reference clock. An example of its use is depicted in Figure 5.24. The `clock` item accepts three parameters:

1. `Type`—number 1 for local on-board clock.
2. `Unit`—number 0 as the first (the only) unit of local on-board clock.
3. `Stratum`—the distance in hops from an accurate authoritative time source; for the local on-board clock, we use number 10 to make it higher than standard Internet NTP servers.

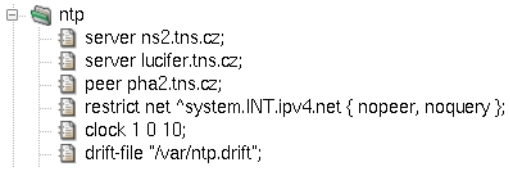

Figure 5.24: On-board clock with NTP server

The resulting configuration file is available among Kernun UTM samples under the name `ntp.cml` in `/usr/local/kernun/conf/samples/cml`.

5.6 Monitoring of Kernun UTM Operation

5.6.1 Logging Configuration

All Kernun UTM proxies write information about their state and progress into the log. The log can be used later to analyze the traffic, generate statistics, find out why a certain connection was refused, or look for system errors. Logging is switched on for all proxies by default. To alter the log settings of a proxy, add a `log` section with desired items (see below) into it.

Logging can be done either by the standard **syslog** daemon (the default is the `LOCAL4` facility, which logs into the `/var/log/kernun-debug` file), or directly into a specified file. Moreover, there is a possibility to log a small amount of the last data for each proxy into a memory-mapped file, which can be used for system failure analysis. The `facility` item is used to change the

[4] Currently, Kernun UTM appliance does not allow for optional hardware reference clock delivery, but this is subject to future changes.

facility number, the `file` item to enable logging into a file and the `mem-file` item to specify the memory-mapped file.

There are 9 levels of log messages (the lower the number, the higher the severity) and the administrator can choose the level of logging. The log level of a proxy can be set either in the configuration using the `level` item, or at runtime by sending a `SIGUSR1`/`SIGUSR2` signal to the proxy process to increase/decrease the level. The proxy logs only messages with a level lower or equal to its log level. The log message level is appended to the `MsgID` column of the message, so you can easily filter only messages of a certain log level in the log browser (Section 3.1.3).

Each log message contains the `MsgID` column, which stands for the type of the message. For every log message type, there is a manual page in the section 6. The Message ID consists of a component code (an application name, possibly with a suffix), message number and message log level. Most of the information on normal traffic can be gathered from the level `I` (information statistical messages) and level `N` (noticeable conditions), which are both logged with any log level set. For example, a simple HTTP 1.1 request is started by `TCPS-706-N` which informs about a new TCP connection and assigns the connection to a child, then a session is started (`HTTP-710-I`) and proceeds with the first phase of acl. If the session is accepted, then the HTTP request is started (`HTTH-702-I`), request-acl is evaluated (`HTTH-811-I`) and the client is connected to the server (`MNIO-710-N`). When the answer, the document, arrives from the server, doc-acl is evaluated and the message `HTTH-893-I` is added to the log. If the document was accepted, the request end message `HTTH-713-I` is added and the proxy waits for other requests on the same socket (we are communicating over the multi-request HTTP 1.1 protocol). Then, possibly, more requests are served and, eventually, the session is ended (`HTTP-711-I`), the proxy finishes the connection and waits for a new one (messages `TCPS-707-N` and `TCPS-715-N`). The log of the described situation is shown in Figure 5.25.

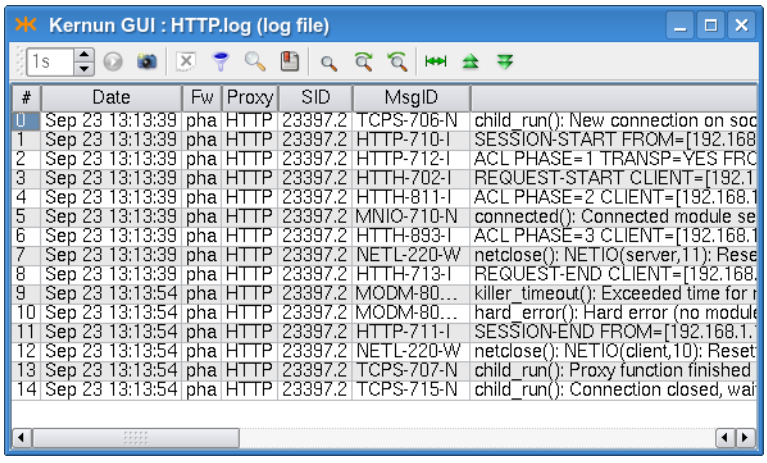

Figure 5.25: Log of an HTTP request

For more information on logging, see **logging**(7). For details concerning configuration of logging, see the **log**(5) manual page.

5.6.2 Log Rotation

The created log files grow enormously and might fill all the free disk space. Another problem of logging is that manipulation with big files is more difficult (and slower) than with smaller ones. Kernun UTM provides tools for log rotation as a method of solving the above-mentioned problems. Log files are regularly (daily, weekly or monthly) renamed and the system starts afresh with an empty log file. The old log files are thrown away after a certain period. The administrator can configure log rotation by adding an `rotate` item into the system-level `rotate-log` section for the entire Kernun UTM or into the `log` section of a particular proxy, if it logs into a separate file. In the `rotate` item you can specify the owner, group and rights of the file, the number of coexisting history logs, the compression method and the time of log creation. The logs can be rotated either periodically (daily, weekly or monthly), or when the size of the log exceeds a specified limit. The log rotation will only work if you have it scheduled in the `crontab`, as illustrated e.g. in the sample crontab file `/usr/local/kernun/conf/samples/include/crontab.cml`.

5.6.3 Monitoring of Active Sessions

Each of the connection-oriented proxies (http-, ftp-, imap4-, pop3-, smtp-, sip-, sqlnet- and tcp-proxy) and udp-proxy are able to write information about the current sessions (connections they are currently serving) into special binary files that can be used to monitor their current state. A log does not provide all the necessary information when huge files are being downloaded: we are informed that the request has started, but do not know anything until it ends. This is the right time to use the **monitor**(1) application, which collects information about the current session's progress and is able to present it to the user (either as plain text, or in the HTML format). Monitoring is best accessible from the GKAT window of the Kernun GUI via the Monitor tab in the detail of a proxy (or the `Proxies` root node). The information in the monitor include the name and type of the proxy, the PID of the proxy process, the duration of the current session, the server and client IP addresses and names, the number of bytes transferred and delivered in both directions, the connection speeds, and the URI of the document (in the case of http- or ftp-proxy). Monitoring is only available for proxies that contain the `monitor` section in their configuration.

Monitoring information is also stored in a database that is used for graph generation; for more details see Section 5.6.5.

5.6.4 Proxy Statistics Generation

Every proxy can generate statistics of its usage; these statistics are then shown in the statistics browser window (Figure 5.26), which is accessible via the File | Statistics menu in the GKAT. In the browser, the tree on the left-hand side shows the available statistics (for each of the available periods and all the proxies that generated statistics in that period). The right-hand part of the window shows the statistics of the selected period, date and proxy as an HTML page.

To enable the generation of statistics for the desired proxies or the whole system, insert `stats-daily`, `stats-weekly` and/or `stats-monthly` sections into their configurations. Inside these sections, you can specify the number of top results for each watchable parameter (if not set, the default value will be used). By default, the generation scripts for each of

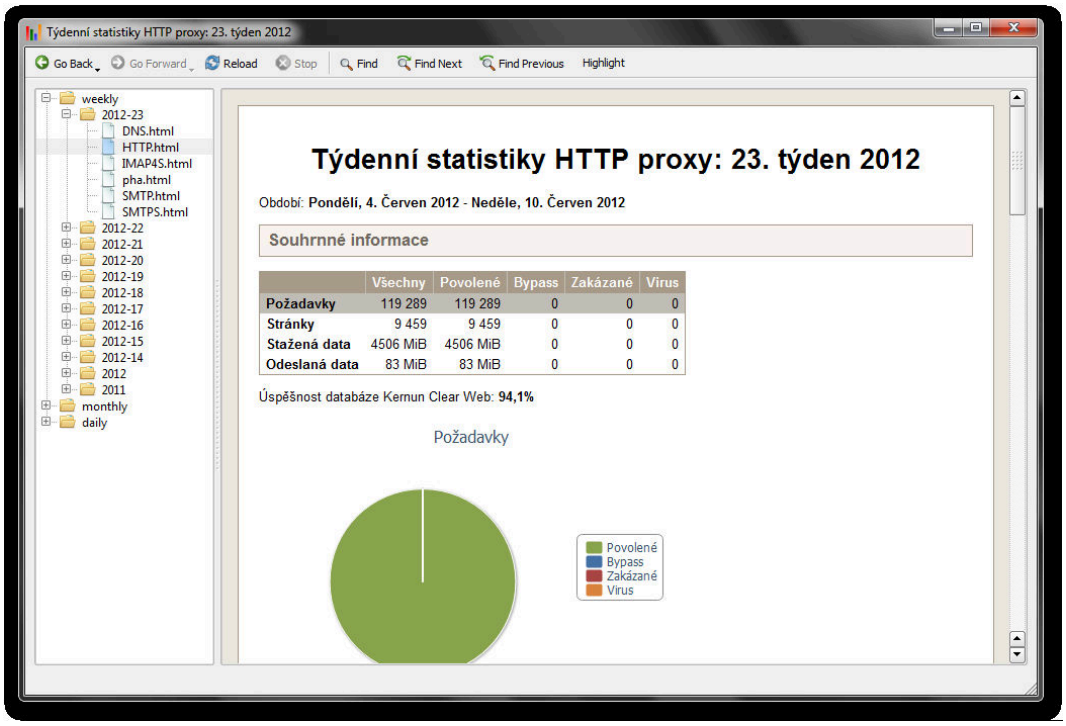

Figure 5.26: Statistics browser window

the periods are scheduled in the crontab. Therefore, no changes should be necessary. Figure 5.27 shows a sample statistics configuration for an `http-proxy`. Sample configuration with generation of statistics for most of the proxies and the whole system is included in `/usr/local/kernun/conf/samples/cml/statistics.cml`.

5.6.5 Monitoring of System Parameters

Kernun UTM is able to monitor various system parameters, from hardware temperature to proxy loads, and store their values in a database. Later it is possible to create and view graphs generated from these values and thus examine the traffic load history. The graphs are generated by the command line tool **rrd**, described in the **rrd**(1) manual page, which can be also used to view the available graph types and intervals. The easiest way to access the graphs is using the Graphs tab in the GKAT window of the Kernun GUI. It is available for all the proxies and network interfaces, and for several system components. All the system parameters are displayed in the Graphs tab of the top-level GKAT node, as depicted in Figure 3.8. While system components other than proxies generate their graphs automatically, you have to insert a `monitoring` section into the configuration of a proxy to generate its graphs.

Chosen graphs can be added to the `Favorite graphs` tab in the Graphs tab of the top-level GKAT node to make them more accessible; the period of the displayed graphs can be changed using a combo box.

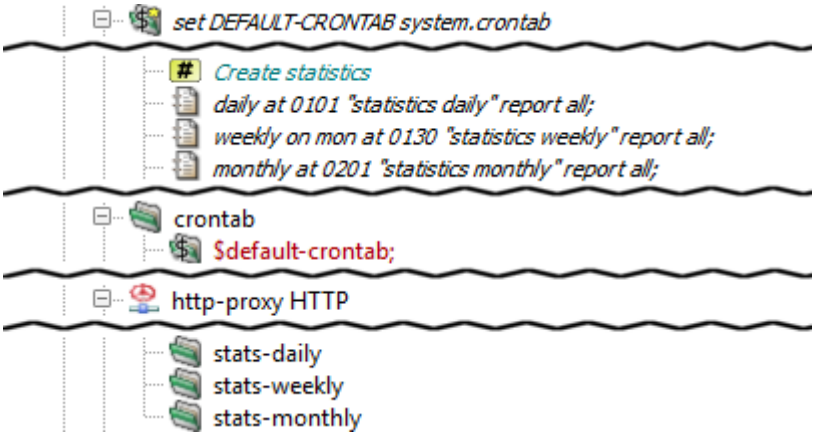

Figure 5.27: Configuration of statistics

5.7 Networking in Proxies

In general, a proxy can listen in two modes: *non-transparent* and *transparent*. In the non-transparent mode, the client must be aware of Kernun UTM, as it is connecting explicitly to some of its IP addresses. For example, the proxy address and port must be entered in the preferences dialog of the Web browser. According to the configuration of the protocol and of Kernun UTM, the proxy typically connects to a server and mediates the communication.

In the transparent mode, on the other hand, Kernun UTM captures clients' connections and hands them over to the proxies. Although the client thinks to be connected directly to the server IP address, it is in fact communicating with the proxy. Hence, the client does not have to be configured for using the proxy. The proxy typically creates the connection on its own and mediates the communication. According to the configuration, the proxy can either connect to a server (the one the client was connecting to, or even to another one), or it can react otherwise, e.g. deny the connection.

5.7.1 Transparent Proxies

Transparency for Clients

Incoming transparency (i.e., proxy listening in the transparent mode) is configured by specifying the `transparent` item in the `listen-on` section in the proxy configuration.

Let us demonstrate the transparency aspects on the example of `http-proxy`. It is configured to listen in the `transparent` mode on port 80 and in the `non-transparent` mode on port 3128, as depicted in Figure 5.28.

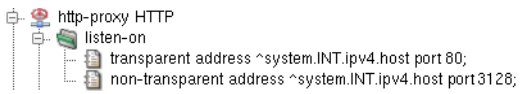

Figure 5.28: HTTP Proxy listening both in transparent and non-transparent mode

When one proxy listens in both modes simultaneously, clients can choose whether to connect in the transparent mode (by connecting directly to the Web server to its port 80), or non-transparently (by connecting to the proxy to port 3128). Examples of Web browser configuration dialogs are shown in Figure 5.29.

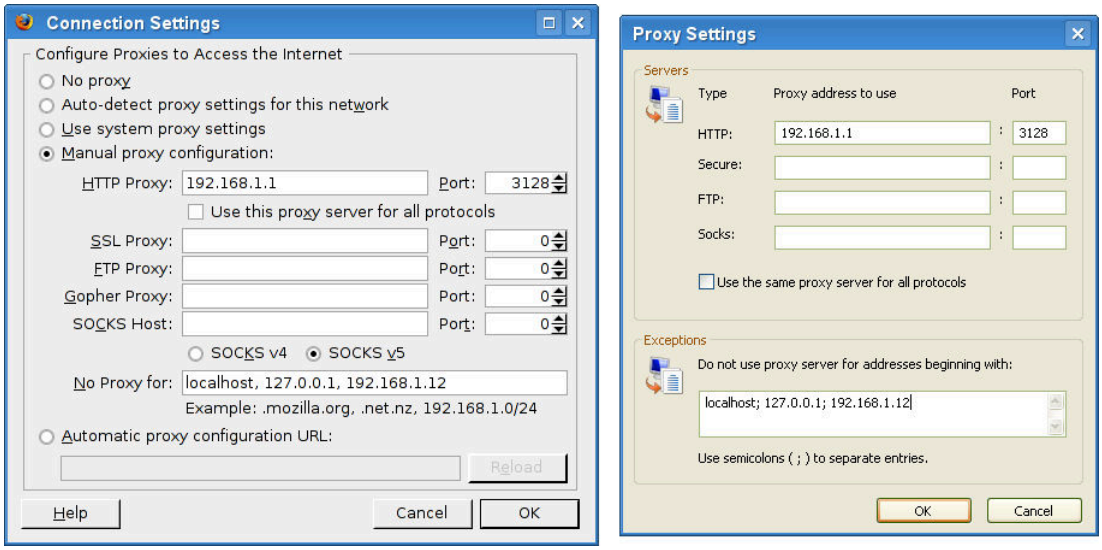

Figure 5.29: Configuration of non-transparent mode in Firefox and Microsoft Internet Explorer Web browsers

After the configuration is applied to Kernun UTM and the proxy is restarted, it starts to listen on the two given sockets. This can be verified in the Network | sockstat tab of the GKAT window, as depicted in Figure 5.30. We can see sockets listening for connection in the transparent mode (rows marked ❶) and in the non-transparent mode (rows marked ❷). These sockets occur in the list more than once as they are shared by the parent process and all the child processes. The listening sockets show *:* in the ForeignAddr column. The non-transparent listening sockets show the IP address they listen on in the LocalAddr column. The transparent listening sockets show the special symbol >> in the LocalAddr column, preceded by the name of the interface, from which they accept connections.

In this example we can also see two clients to be actually connecting through the proxy, both to the Web server at `194.228.50.79`. Client from IP `192.168.1.31` is connecting via the non-transparent proxy. We can see the connection from the client to Kernun UTM's internal IP address (row marked ❸), and the connection from Kernun UTM's external IP address to the server (row marked ❹).

Client from the IP address `192.168.1.12` is connecting transparently. We can see the connection from the client to the server, which was grabbed by Kernun UTM (row marked ❺) and the connection from Kernun UTM to the server (row marked ❻).

The realized connections can be traced in the log, as is shown in Figure 5.31. We have filtered the log messages that inform about the session—see the filters box in the screenshot. The complete track of the session is more detailed, concerning matching of ACLs, requests, documents, etc. The selected message (HTTP-710-I) denotes the start of the session in `http-proxy`, and informs

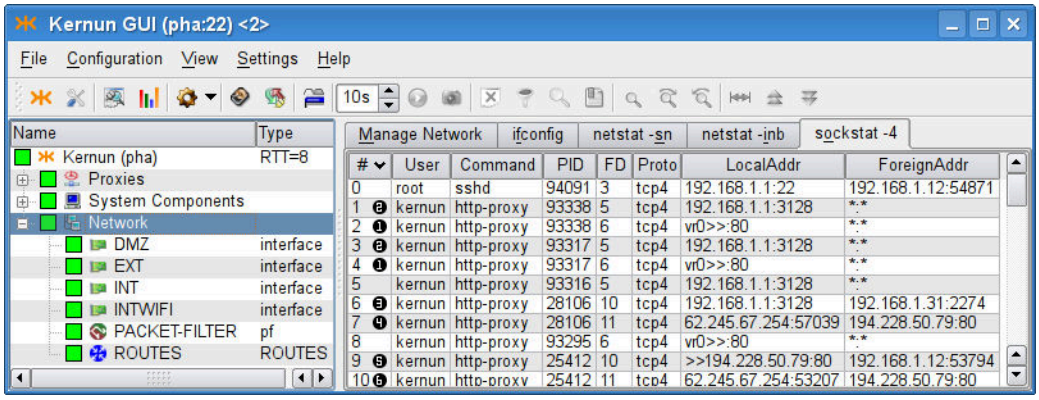

Figure 5.30: Sockstat tab showing the list of all sockets

about the facts that are known at the time the session is being established (client address, server address and the transparency flag). The session end message (HTTP-711-I) would also provide other informations concerning the session (the number of requests, the length of the session, etc.).

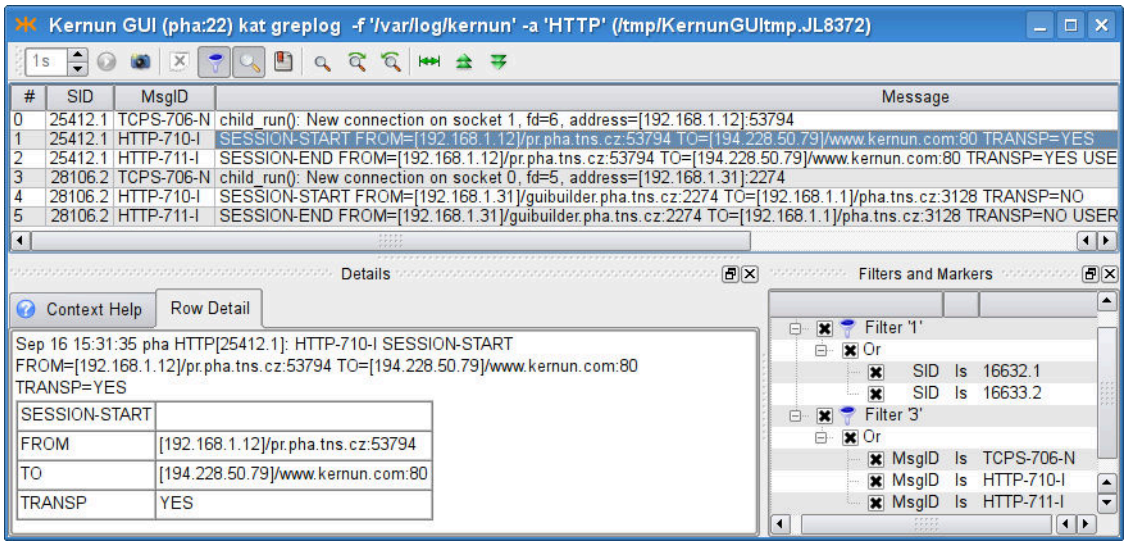

Figure 5.31: Tracking transparent and non-transparent connections in the `http-proxy` log (the log was filtered, in order to save space).

Transparency for Servers

When a client is connecting to a server, the question is what IP address the server sees as the client IP address. Different approaches can be useful in different situations.

If a network with private addressing is hidden behind Kernun UTM, Kernun UTM must connect from its own (external, public) IP address. When the client network uses public IP addresses, both approaches can be desired: either the internal addressing should be hidden (the traffic should be formally realized from Kernun UTM's IP address), or not. On the other hand, a server protected

in a DMZ may need to know the IP address of the real client.

By default, Kernun UTM proxy connects to the server from Kernun UTM's IP address (i.e. hiding the client's real IP address). However, there are two other options, which can be specified in the first-layer ACL (see Section 4.2.5 for details on ACLs):

- `source-address client` denotes that the IP address of the real client should be used as the source address;
- `source-address force` *`address`* denotes that the given *address* should be faked as the source address for the connection to the server.

The `source-address` item can be either specified on the `system` level of the configuration, in which case it influences all the proxies that reference the ACL, or in the `session-acl` section of the particular proxy.

In Figure 5.32, the real client address is configured to be used as the source address for all the named proxies, because it is specified in the `system` ACL.

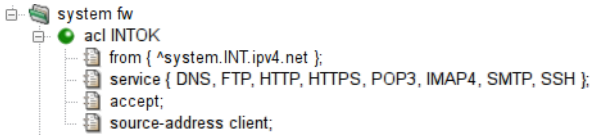

Figure 5.32: Transparency for servers (`source-address client`)

5.7.2 A Proxy and a Server on the Same Port

In Kernun UTM, a single port can be shared by several proxies or servers, provided that they do not collide in their listen sockets. For example, it is desirable to provide a transparent SSH proxy for connecting to SSH servers in the Internet and, at the same time, to provide an SSH server for administration of Kernun UTM. Both should be available from the protected network.

An example of such configuration is shown in Figure 5.33.

When a packet arrives, Kernun UTM kernel delivers it to the most specific socket that is suitable for it. Consult transparency(7) for more information, including the definition of conflicting sockets and the socket preference. This way, it is possible to provide services on their traditional ports as well as transparent proxies on the same ports.

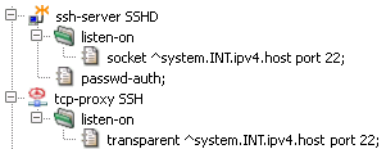

Figure 5.33: A port shared by two applications

5.7.3 Listening on a port range

Kernun UTM proxy can be configured to listen on a set of contiguous ports. This ability is also known as listening on a *port range* and is available for TCP and UDP-based proxies. If a transparent proxy listens on a port range (and ACLs do not specify otherwise), the proxy connects to the port the client was connecting to. If a non-transparent proxy listens on a port range, the situation is similar to the regular non-transparent proxy (one that listens on a single port): the port (as well as the server IP address) would be either provided in some ACL in the Kernun UTM configuration (for example, in the `plug-to` item), or it is a part of the application protocol. An example of `sip-proxy` configured to listen on a port range is shown in the Figure 5.34.

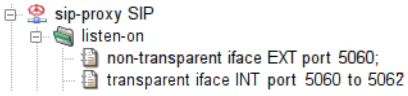

Figure 5.34: The `sip-proxy` configured to listen transparently on a port range 5060-5062

Warning

Proxies cannot listen on ranges of ports that collide with the system default port ranges. The system normally assigns ports to connections with unspecified source port from three port ranges, which can be modified by the `portrange-default`, `portrange-high`, `portrange-low` items in the `system.sysctl` section. None of those ranges may overlap with the range of ports that a proxy is listening on. In the case of a port range collision, the system would not know, which ports are free to use and which are occupied.
Note that this problem does not occur when the proxy listens on separate ports. In this case, the single ports are reserved for the use of the proxy (even if they are inside one of the system port ranges).

5.8 H.323 Proxies

`h323-proxy` and `gk-proxy` are the proxy daemons for a set of multimedia communication protocols called H.323 protocols. Kernun UTM supports them only for backward compatibility, because this protocol family is now widely replaced by the SIP protocols (see Figure 5.35 below). Therefore, this manual does not cover this topic. For more information, see h323-proxy(8), h323-proxy(5), gk-proxy(8) and gk-proxy(5).

5.9 SIP Proxy

`sip-proxy` is the proxy daemon for the Session Initiation Protocol (RFC 3261 et al.), i.e. mainly Internet telephone calls and related services.

The proxy is configured in the `sip` section. In the sample configuration depicted in Figure 5.35, `chroot-dir` defines directory into which it should be chrooted. The proxy listens transparently for requests at Kernun UTM's internal address at port 5060. However, due to the hop-by-hop logic

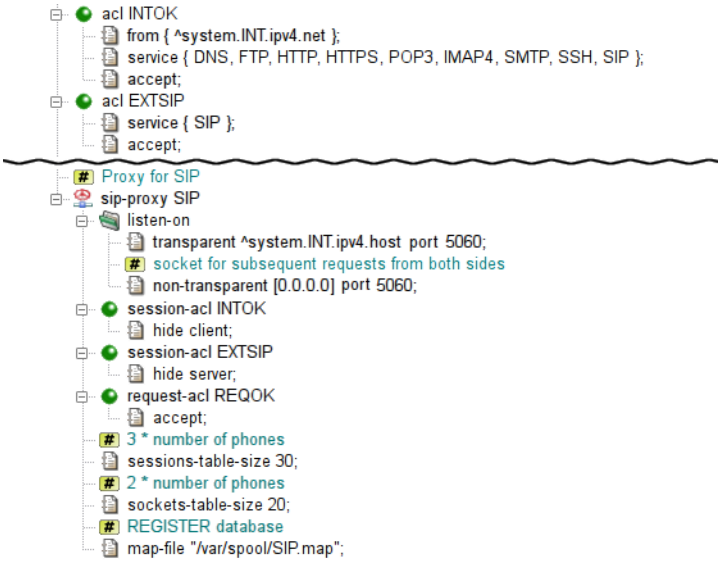

Figure 5.35: SIP Proxy

of the protocol, the proxy must listen on this port non-transparently on all internal and external interfaces as well. As usually, the proxy must be referenced by at least one ACL in the system section. In order to enable incoming calls, we will need to allow also sessions from the external network.

sip-proxy uses two-phase ACLs. The first phase, session-acl, is checked once for each client connection. It permits or denies client access and sets some connection parameters. This is where the administrator can specify that some important data that is usually stored in SIP messages headers (e.g. internal addresses) is to be hidden. The second phase, request-acl, is checked once for each request and it can be used e.g. to change the target server according to the Request-URI.

sip-proxy requires specification of the maximum of sessions and data channels used in parallel. An estimate is three times number of the phones for sessions-table-size and two times number of the phones for sockets-table-size. Finally, the proxy requires a file to store the registration data of clients (local phones).

The complete resulting configuration can be found in /usr/local/kernun/conf/samples/cml/sip-proxy.cml. See sip-proxy(8) and sip-proxy(5) to learn more about sip-proxy.

5.10 SQLNet Proxy

sqlnet-proxy is the proxy daemon for the proprietary Oracle SQL*Net Protocol. The proxy can handle features, such as session redirection, or database user checking.

The proxy is configured in the sql section. In the sample configuration depicted in Figure 5.36 chroot-dir defines directory into which it should be chrooted. The proxy listens transparently for requests at Kernun UTM's internal address at port 1521. As usually, the proxy must be

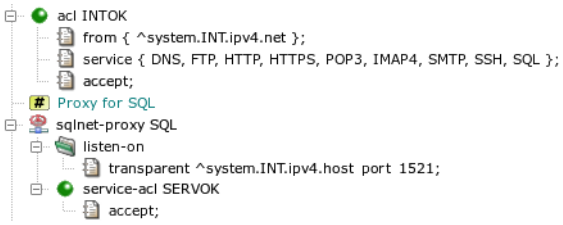

Figure 5.36: `SQL*Net` Proxy

referenced by at least one ACL in `system` section.

The `sqlnet-proxy` uses two-phase ACLs. The first phase, `session-acl`, is checked once for each client connection. It permits or denies client access and sets some connection parameters. The second phase, `service-acl`, is checked once for each CN (connect) or RD (redirect) message and it can be used e.g. to change the target server according to the SERVICE name.

The complete resulting configuration can be found in `/usr/local/kernun/conf/samples/cml/sqlnet-proxy.cml`. Since the protocol is proprietary, clients often violate it and it is necessary to avoid some checks using configuration directives, such as `connect-string-charset`. See sqlnet-proxy(8) and sqlnet-proxy(5) to learn more about `sqlnet-proxy`.

5.11 UDP Proxy

Kernun UTM provides a generic proxy for handling application protocols based on UDP. Its philosophy is similar to the generic TCP proxy, which is used for the HTTPS and SSH protocols in the initial configuration, as described in Section 4.2.9. The proxy waits on one or more ports, transparently or non-transparently, for datagrams from clients. Although UDP is a stateless protocol, the UDP proxy defines logical sessions that group together related datagrams. See **udp-proxy**(8).

We will demonstrate the UDP proxy on the DNS protocol. It is one of typical applications of the UDP proxy[5]. Kernun UTM provides the DNS proxy (refer to Figure 4.11), which is a better choice for accessing DNS servers in the Internet, because it performs thorough checks of the protocol. However, if Kernun UTM connects two trusted internal networks and clients in one of them send DNS requests to a server in the other, the use of the UDP proxy may be sufficient.

The DNS proxy in the sample configuration in Figure 5.37 was replaced by the UDP proxy. It listens on the internal network interface on the DNS UDP port 53 non-transparently. The item `max-sessions` in section `udpserver` limits the maximum number of logical sessions that can be handled by the proxy simultaneously. All DNS requests from the internal network are accepted by global `acl INTOK`. This ACL is extended in the proxy by the `plug-to` item, which forwards all requests to a single DNS server. As DNS is a request-reply protocol, we define that each logical session contains at most one request (client to server) and one response (server to client) datagram.

[5] Another standard use of the UDP proxy is forwarding the OpenVPN protocol via Kernun UTM.

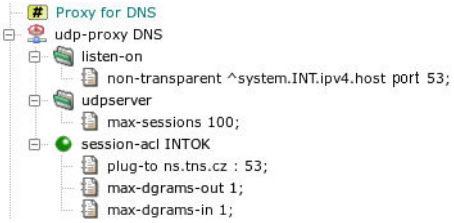

Figure 5.37: UDP proxy

The setup described in this section can be tested by specifying in `plug-to` a DNS server that accepts and recursively solves requests from Kernun UTM. The client machines must be configured to use Kernun UTM as their DNS server.

5.12 Cooperation of HTTP and FTP Proxies

Web browsers support access to FTP servers by using URLs beginning with `ftp:`. If no proxy is configured for FTP in the browser parameters, the browser accesses FTP servers using the native FTP protocol. Such communication can be mediated by a transparent FTP proxy. Another situation is the non-transparent case, when a proxy for FTP is specified in the browser settings. Then the browser communicates with the proxy using the HTTP protocol, rather than FTP. Hence, the communication is processed by the HTTP proxy, which gets a URL starting with `ftp:`. The proxy must act as a FTP client, contact an FTP server, and send the result to the browser via HTTP. The HTTP proxy does not support FTP directly. Instead, it communicates with the FTP proxy using an internal protocol[6]. The FTP proxy handles the FTP part of the communication.

Both **http-proxy**(8) and **ftp-proxy**(8) need to be reconfigured in order to cooperate in the processing of `ftp:` requests from Web browsers. In the `ftp-proxy` section, a new `non-transparent` listening socket should be added to the `listen-on` section. The two proxies run on the same system and they communicate via the loopback interface. Hence, the socket for HTTP-FTP cooperation listens on the `localhost` address and on an arbitrarily chosen port. It is necessary to allow connections from the HTTP proxy to the FTP proxy by adding a system-level `acl` and to tell the FTP proxy in the corresponding `session-acl` that it should expect requests from the HTTP proxy instead of the standard FTP. The HTTP proxy must be told to pass FTP requests to the FTP proxy by adding a `ftp-proxy` item to section `http-proxy`.

An example of FTP and HTTP configuration is shown in Figure 5.38. Port 8022 has been chosen and stored in variable `HTFTP_PORT`. A new `acl HTFTP` section has been created, which permits connections in the FTP proxy from the local host to the HTFTP port on the local host. In the `ftp-proxy FTP` section, the FTP proxy is switched (by item `htftp-mode`) to the HTFTP mode for connections accepted on the HTFTP socket. The HTTP proxy is instructed how to contact the FTP proxy by item `ftp-proxy`. The complete resulting configuration can be

[6] Kernun UTM refers to the HTTP-FTP cooperation protocol as *HTFTP*.

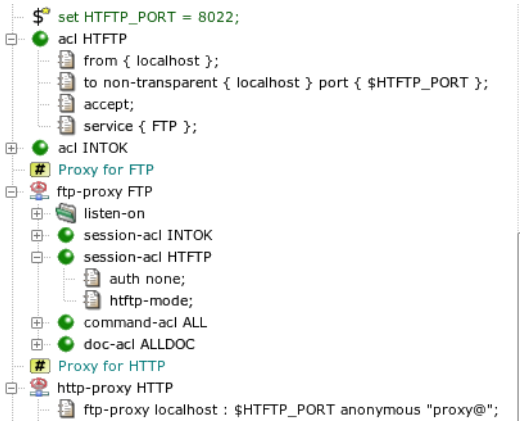

Figure 5.38: Cooperation of HTTP and FTP

found in file `/usr/local/kernun/conf/samples/cml/htftp.cml`. After applying these configuration changes and restarting the HTTP and FTP proxies, it should be possible to set the HTTP proxy as the proxy for FTP in the Web browser and to start using `ftp:` URLs.

5.13 Secure Communication Using SSL/TLS

Kernun UTM supports network communication securing using the standard SSL (Secure Sockets Layer) protocol and its newer version TLS (Transport Layer Security). As the SSL/TLS protocol ensures end-to-end security, Kernun UTM cannot read or modify data from an SSL/TLS connection between a client and a server. This means that protocols using SSL/TLS encapsulation, e.g., HTTPS, POP3S, or IMAP4S, cannot be processed by a protocol-specific proxy. Instead, generic TCP proxy is usually used to forward such communication. In the case of HTTPS, a non-transparent HTTP proxy can be utilized for tunneling HTTPS data in an HTTP connection with the `CONNECT` HTTP request method. Sometimes, however, we need to have access to encrypted data for the purposes of analysis or even modification. An example is antivirus checking of files transferred via HTTPS.

The configuration described in this section is useful for fixed secure tunnels, e.g., via **tcp-proxy**, or for securing a single HTTP server by **http-proxy**. It is unsuitable for inspecting HTTPS traffic between ordinary HTTP clients (browsers) and servers, because the proxy presents a single server certificate (containing a single fixed host name) for any server accessed by a client. Common behavior of browsers is to compare the accessed server host name with the name in the server certificate and to issue a security warning if they do not match. If you need to handle this, HTTPS inspection, introduced in Section 5.19, is a viable option.

To access data protected by SSL/TLS, a proxy must split the communication into two SSL/TLS connections. Data from a client is first sent to the proxy via the first secure connection terminated on Kernun UTM. The encrypted data stream is decrypted by the proxy, processed, encrypted again and sent to the server via the second secure connection. Data from the server is passed via the two connections in the opposite direction. It is also possible to convert the unencrypted version of an application protocol (for example, HTTP) used by a client to the encrypted version

(for example, HTTPS) accepted by a server, or vice versa.

An example of a proxy that uses SSL/TLS security is presented in Section 5.14.7, and its configuration is shown in Figure 5.51. This proxy converts a secured connection from a client to plain HTTP sent to the server. Another example is depicted in Figure 5.39. It is an HTTP proxy that enforces authentication of the remote server and also authenticates Kernun UTM to the remote server.

Figure 5.39: SSL/TLS security configuration

The parameters of an SSL/TLS connection are defined by the section `ssl-params` . All the available parameters are explained in ssl(5). In our example, this section selects the X.509 certificate with the related private key used by the proxy to authenticate itself to clients and servers (item `id`), and the list of certification authority certificates (`auth-cert`) with the related certificate revocation lists (`crl`) for verification of certificates provided by peer clients and servers. The `verify-peer` item forces the proxy to request and verify certificates of peers. If a peer does not provide a certificate or the certificate cannot be verified, the SSL/TLS handshake fails and the session is terminated.

The `client-ssl SSL` item in the `session-acl INTOK` section switches the connection from the client to the SSL/TLS mode and sets the protocol parameters. If the client establishes an encrypted connection and sends a valid certificate, it can send an HTTP request. Otherwise, the session is terminated. After receiving an HTTP request, the proxy searches for a request ACL. It selects `request-acl TNS-CERT` only if the client's certificate contains "`SSL access`" in the subject field, because of the `client-cert-match` item. The request ACL enables SSL/TLS on the connection to the server and sets the protocol parameters. The `server-cert-match` item defines conditions that must be fulfilled by the server's certificate (it must contain a common name

ending by "`kernun.com`" in our example). Note that if the `client-cert-match` item is not matched, the single ACL is skipped and the search continues at the next request ACL, whereas if the `server-cert-match` item is not matched, the request is terminated immediately.

If you apply this kind of configuration and access a Web server via the proxy, your Web browser will show that the peer certificate is that of the proxy. In the proxy log, you will see details of the client's and server's certificates as received by the proxy.

5.14 User Authentication

Kernun UTM applies different rules to network traffic depending on the identity of the communicating parties. A client or a server can be identified in several ways. The basic one is the IP address and the port number, which are available for any network connection. The IP-based identity is used very often, but it distinguishes between computers, not users. Sometimes it is necessary to apply different rules to different users, while each user can work on several machines and each machine can be utilized by more than one user. In such situations, we need to identify individual users, independently on the IP addresses of the computers they use. Kernun UTM provides various means of user authentication. An authenticated user name can be used as one of the conditions tested when searching for an applicable ACL.

Some application protocols allow the passing of user credentials to a proxy within the normal protocol data flow. In Kernun UTM, such type of authentication is supported by the HTTP and FTP proxies. Many other proxies are able to use *out-of-band (OOB) authentication*, which binds a user name to a client machine IP address. Kernun UTM also provides a special authentication mode usable for protecting access to protected HTTP servers from clients in the external network—the HTTP authentication proxy.

Examples of configuration for all variants of authentication except NTLM described in this section are summarized in the sample configuration file `/usr/local/kernun/conf/samples/cml/auth.cml`.

5.14.1 Authentication Methods

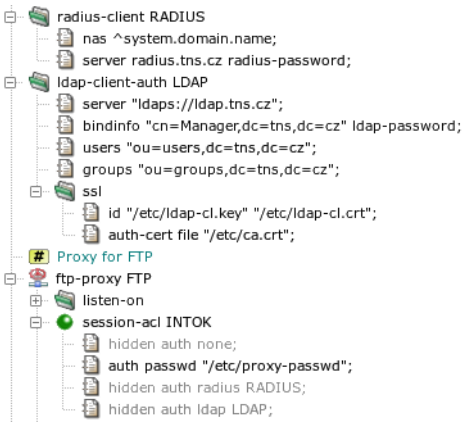

Figure 5.40: Authentication methods

Kernun UTM supports several methods, described in detail in auth(7). Section `ftp-proxy FTP` in Figure 5.40 contains sample `auth` items for different authentication methods. To switch to another authentication method, simply hide the single not hidden `auth` item and unhide another one.

None No user authentication is performed.

Password file The user provides a user name and a password, which are compared to name-password pairs in a password file stored locally on Kernun UTM. The administrator must create—by utility **fwpasswd**(1)—a file containing valid user names with encrypted passwords (`/etc/proxy-passwd` in the example configuration).

RADIUS The user name and password are verified by a RADIUS server. There are two variants: name/password (the user enters his name and the password) and challenge/response (the user enters her name, the RADIUS server generates a challenge, and the user replies by a response derived from the challenge and some secret information, e.g., using an authentication token hardware). If the `radius` method is chosen in an `auth` item, it must reference a `radius-client` section that defines the parameters needed to connect to a RADIUS server.

LDAP The user name and password are verified in an LDAP database. If the `ldap` method is chosen in an `auth` item, it must reference an `ldap-client-auth` section that defines the parameters needed to connect to an LDAP server. If a secure connection to the LDAP server is used (the LDAP server URI begins with `ldaps`), the certificates to be used must be specified in the `ldap-client-auth.ssl` subsection.

Kerberos The Kerberos authentication is specific to the **http-proxy**. If a user is authenticated in an Active Directory domain (or has a Kerberos TGT), a Web browser uses the user's Kerberos tickets and does not require any interaction with the user (name and password entry).

NTLM The NTLM authentication is specific to the **http-proxy** and HTTP clients (Web browsers) running on Microsoft Windows client operating systems. If a user is authenticated in an Active Directory domain, a Web browser uses the user's credentials known by the system since the login time and does not require any interaction with the user (name and password entry). It can be combined with other authentication methods for clients that do not support NTLM.

> **Note**
>
> If a proxy runs in chroot, the paths in `auth passwd` and `ldap-client-auth.ssl` are interpreted in the context of the chroot directory.

5.14.2 Authentication in FTP Proxy

User authentication in the FTP proxy can be enabled using the `auth` item that specifies an authentication method other than `none` in `session-acl`. For example, a proxy with the configuration shown in Section 5.14.1 will use a local password file. If authentication is enabled and the

user does not provide valid credentials, the proxy terminates the session. Otherwise, it is possible to match user name or group using a `user` item of `command-acl`. There are several ways of sending authentication data to the proxy, see **ftp-proxy**(8). An example FTP session initiation with authentication, using a command line FTP client:

```
$ ftp fw.pha.tns.cz
Connected to ftp.tns.cz.
220-fw.pha.tns.cz KERNUN FTP Proxy (Version KERNUN-3_0-RELEASE) Ready.
    Target server name/address and authentication to proxy required.
    You can use loginname in form of proxy_user@server_user@server.
220 Log in with USER and PASS command.
Name (192.168.10.1:user): guest@anonymous@ftp.tns.cz
331-USER command successful.
    Proxy user: 'guest'.
    Target-server user: 'anonymous'.
    Target server: ftp.tns.cz, port: 21.
    Enter password in form proxy_user_password@server_user_password.
331 Enter password or response, please.
Password: ****@****
230-User 'guest' succesfully authenticated.
230-pha.tns.cz KERNUN FTP Proxy (Version KERNUN-3_0-RELEASE) Ready.
230-Welcome to ftp.tns.cz FTP service.
230----- proxy connected to [194.228.50.76]/ftp.tns.cz port 21 ----
    ---- proxy connected to ftp.tns.cz port 21 ----
230-Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

A similar, but unsuccessful session initiation (due to a wrong password):

```
$ ftp fw.pha.tns.cz
Connected to ftp.tns.cz.
220-fw.pha.tns.cz KERNUN FTP Proxy (Version KERNUN-3_0-RELEASE) Ready.
    Target server name/address and authentication to proxy required.
    You can use loginname in form of proxy_user@server_user@server.
220 Log in with USER and PASS command.
Name (192.168.10.1:user): guest@anonymous@ftp.tns.cz
331-USER command successful.
    Proxy user: 'guest'.
    Target-server user: 'anonymous'.
    Target server: ftp.tns.cz, port: 21.
    Enter password in form proxy_user_password@server_user_password.
```

```
331 Enter password or response, please.
Password: ****@****
421 User 'guest' not authenticated.
ftp: Login failed.
ftp>
```

Selected log messages related to these two FTP sessions are displayed in Figure 5.41.

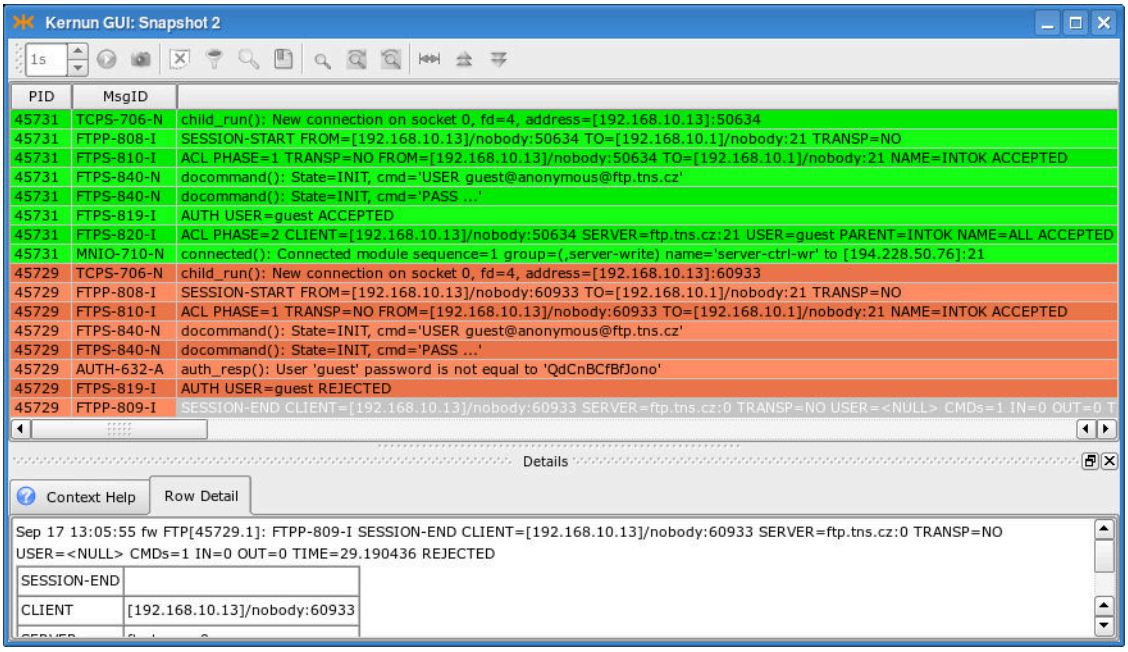

Figure 5.41: Log of authentication in FTP proxy

5.14.3 Basic Authentication in HTTP Proxy

The HTTP proxy supports proxy authentication using the Basic authentication method, i.e., a name and a password entered by the user. The technical details of the authentication are described in **http-proxy**(8). Proxy authentication in HTTP works as follows: the first request sent by the client (browser) is unauthenticated, the proxy replies with response code 407 "Proxy Authentication Required"; upon reception of this response code, the browser displays a dialog (see Figure 5.42) asking for user credentials; the browser then repeats the request with the user name and password attached.

Figure 5.43 contains an HTTP proxy configuration with user authentication. The authentication is enabled and the authentication method (a password file in this case) is selected by the `auth` item in `session-acl`. If a request contains user credentials, they are checked and if correct, the user name—and the list of groups the user belongs to—is remembered for matching with the `user` item in a `request-acl`. Unlike in the case of FTP proxy, a request containing invalid or no credentials is not rejected. The processing continues, but no `request-acl` containing a `user` item matches, except for the one with `user none`. There should be a `request-acl`

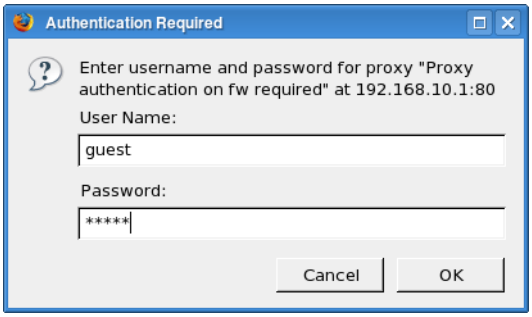

Figure 5.42: Proxy authentication dialog in a Web browser

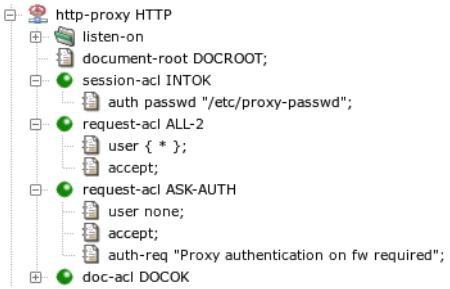

Figure 5.43: User authentication configuration in the HTTP proxy

that matches unauthenticated requests and returns the 407 response. In the example configuration, there are two `request-acl` sections. The first one, `request-acl ALL-2`, matches any successfully authenticated user. The request is processed normally, i.e., forwarded to the target server. The other, `request-acl ASK-AUTH`, matches if the request does not contain valid user credentials[7]. The 407 response is forced by the `auth-req` item. This item also sets the *realm* (prompt) that is usually displayed by the browser in the authentication dialog, as depicted in Figure 5.42.

If we apply this sample configuration to Kernun UTM and invoke a new request in a Web browser, the log will contain messages similar to those in Figure 5.44. The first request was unauthenticated. The two orange messages inform us that `request-acl ASK-AUTH` was selected and response 407 was returned to the browser. After the user filled in the authentication dialog, the request was repeated. The user name and password were correct, user `guest` was authenticated successfully, and `request-acl ALL-2` was selected, as indicated by the two green messages.

[7] This happens in the case of the first request or a wrong name or password entered by the user.

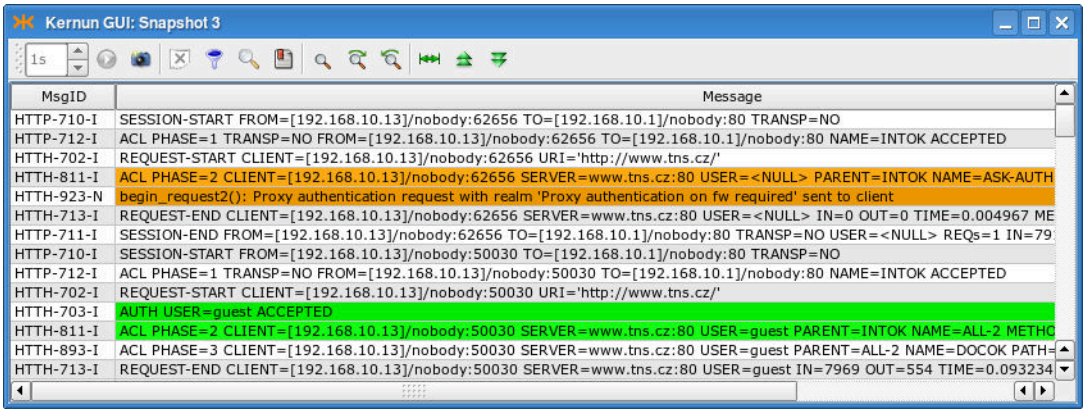

Figure 5.44: Log of user authentication in the HTTP proxy

Important

Because many Web browsers do not expect to be required to perform proxy authentication when no HTTP proxy is configured in the browser, proxy authentication may not work with a transparent proxy. It is possible in such a case that users are displayed a 407 error page instead of the authentication dialog. Use proxy user authentication with a non-transparent HTTP proxy only, unless you check that Web browsers on your client machines handle it correctly with a transparent HTTP proxy.

5.14.4 Kerberos Authentication in HTTP Proxy

The Kerberos authentication is an alternative to the Basic authentication described in Section 5.14.3. It provides a single-sign-on capability for clients authenticated in an Active Directory domain or a Kerberos realm. If a user is logged in a domain, or has a valid Kerberos ticket, a Kerberos-capable Web browser can authenticate the user to the proxy automatically, without asking the user for a name and a password.

Kerberos authentication in Active Directory is supported for Microsoft Windows server and Samba 4 acting as an Active Directory domain controller. Older Samba versions (3.x) cannot be configured as an AD domain controller.

Warning

Authentication to a Samba server uses different authentication mechanism than a MS Windows Server by default. This authentication mechanism (GSS-SPNEGO) fails and may allocate all available memory. Hence, the client (Kernun **http-proxy**) should be configured to use GSSAPI instead. This can be done by adding file `.ldaprc` to the root directory and to home directories of users involved in authentication (`root` and `kernun`). The file should contain line:

```
SASL_MECH GSSAPI
```

Important

If an HTTP proxy uses the Kerberos authentication, it cannot be run chrooted, because it needs access to some system components that are not contained in the standard Kernun chroot environment.

An example of a Kerberos configuration is stored in the sample configuration file `/usr/local/kernun/conf/samples/cml/kerberos-auth.cml`. The `system` section contains global configuration related to the Kerberos authentication in section `kerberos-auth`, Figure 5.45. Two parameters are required: the name of the Active Directory domain (or Kerberos realm) `domain` and the address of the domain controller (or the Kerberos KDC) `ad-controller`. The Kerberos authentication itself provides the name of an authenticated user to the proxy. A `request-acl` can be selected according to the user name or group membership. For the latter option, the proxy needs to know the list of groups the user belongs to. The list can be obtained from an LDAP server. As the Active Directory contains all the necessary information and provides LDAP interface, the Active Directory controller is usually used as the LDAP server. Obtaining group membership can be configured by unhiding the hidden item `ldap` and section `ldap-client-auth` in the sample configuration in Figure 5.45. The URL of the LDAP server references the Active Directory controller. The proxy authenticates itself to the LDAP server by Kerberos using the machine account of the Kernun system in the Active Directory. The Kerberos authentication for LDAP is enabled by `ldap-client-auth.kerberos`. Alternatively, the proxy can authenticate itself using a name and a password of a user with the permission to access the Active Directory contents for reading. The user's credentials must be specified in the `bindinfo` item in place of *`ADUser`*, and the corresponding password in place of *`ldap-password`*. The Active Directory stores user account data in a different format than other LDAP servers used for LDAP authentication. The Active Directory format must be selected by item `active-directory`, which also specifies the domain name.

Important

The Active Directory domain name must be written using *UPPERCASE LETTERS* in the configuration.

Warning

A user's primary group name (usually `Domain Users`) is neither logged nor matched by `request-acl.user.group`.

The Kerberos configuration can be enabled in an HTTP proxy by item `session-acl.kerberos-auth`. Outcome of the Kerberos authentication is processed like the Basic authentication. The user name, optional list of groups, or the fact that the authentication failed, can be used as conditions in `request-acl` sections. There is usually one or more request ACLs that permit access to successfully authenticated users and one request ACL that denies

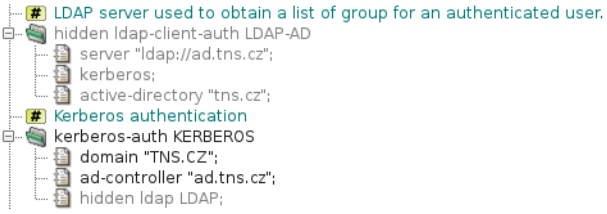

Figure 5.45: Kerberos authentication — section `system`

access to unauthenticated users and asks for authentication. The sample configuration in Figure 5.46 allows access to any authenticated user (`request-acl AUTH-OK`). Users that are not authenticated are requested to perform authentication by `request-acl AUTH-REQ`.

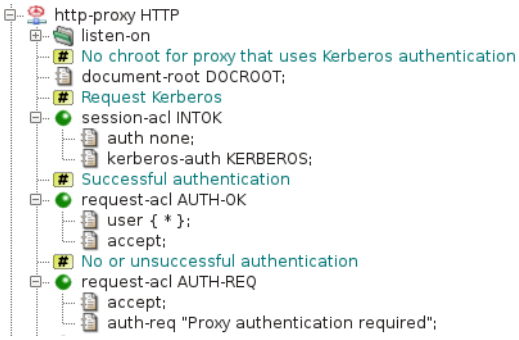

Figure 5.46: Kerberos authentication — section `http-proxy`

In the Kerberos and LDAP configurations presented so far, user and group names are matched in ACLs and written to logs in a short form, without information about domains they belongs to. This may not be sufficient in a multi-domain environment, where different domains may contain users and groups with the same names. Such a multi-domain environment can comprise either domains in a single Active Directory forest, or even domains from several forests if there are explicit trusts between forests[8]. Configuration item `kerberos-auth.user-match full` switches to using long user name in logs and ACLs in format "`user@REALM`" instead of the default short format "`user`". Likewise, adding value "`group-match domain`" to item `ldap-client-auth.active-directory` enables using long group names "`group@domain`".

The LDAP server on each Active Directory controller returns only data from its own domain. Hence, group membership of a user must be queried in the domain which the user belongs to. The proxy is able to select the LDAP server according to the domain name of the authenticated user. To do this, there must be an `ldap-client-auth` section for each domain. Section `kerberos-auth` must contain item `ldap domain`, instead of naming a single `ldap-client-auth` section. Then the `ldap-client-auth` section will be selected by its Active Directory domain name (specified in item `active-directory`).

[8] Trusts are needed because the Kernun system authenticates itself in only one Kerberos realm, but it must be able to accept authentication from other domains and also to authenticate itself to LDAP servers in other domains.

Information about membership of Kerberos-authenticated users in groups can be cached in order to decrease load of LDAP servers. Configuration of caching consists of adding the global section `oob-auth OOB` and referencing it by item `http-proxy.oob-auth-srv`. Cached group membership information for a user name expires after a timeout controlled by items `kerberos-auth.timeout-idle` (expiration after a period of inactivity) and `kerberos-auth.timeout-unauth` (unconditional expiration).

The Kerberos authentication requires some initialization steps before in order to start working. There are two supported methods of initialization.

Important

In order to work correctly, Kerberos authentication requires several conditions to be fulfilled:

- The Kernun system and the Active Directory domain controller must have their system time synchronized. By default, maximum allowable clock difference is 5 min. It is possible to synchronize Kernun system's clock to the Active Directory time by configuring the domain controller as an NTP server in the Kernun configuration.
- The Kernun system must resolve the domain name of the Active Directory domain controller to the correct IP address. The reverse resolution of this IP address must also work and yield the original domain name used for the forward resolution as the first name of the response.

Initialization by msktutil

The first method adds the Kernun system to the Active Directory domain by deleting its machine account if it already exists and creating a new machine account. Then it sets up the Kerberos keytab with all keys necessary for authentication and obtaining group membership information. A user account *user* in the Active Directory with `Domain Admins` rights is required for this task. The initialization process is performed by issuing shell commands on the Kernun system:

```
# kinit user
# ldapdelete -v -H ldap://ADC cn=COMPUTER,cn=Computers,dc=D1,dc=D2...
# msktutil -c --computer-name `hostname -s` -s HTTP/`hostname` \
-h `hostname` --server ADC --no-pac
# chown kernun /etc/krb5.keytab
```

where *ADC* is the address of the domain controller, and *D1, D2, ...* are components of the Active Directory domain name (e.g., for domain `example.com`, we use `dc=example,dc=com`). The computer account name *COMPUTER* will be the value of `-computer-name` with appended character `'$'`. If the default computer account name `` `hostname -s` `` is used, a command for automatic password renewal (**msktutil –auto-update**) will be added to `/etc/crontab`. It is possible to choose another computer account name. A non-default name must be configured in the **http-proxy** by setting `kerberos-auth.kinit` to "*computername*$". In this case, the machine account password renewal must be handled manually, or (better) password expiration must be turned off for the account by option `-dont-expire-password` of **msktutil**.

Important

The computer account name must be entered to the configuration with the appended character `'$'`.

Example: The initialization commands will be, for the domain administrator `admin`, the Active Directory domain `EXAMPLE.COM`, the domain controller `ad.example.com`, and the Kernun system `kernun.example.com`:

```
# kinit admin
# ldapdelete -v -H ldap://ad.example.com \
'cn=kernun$,cn=Computers,dc=example,dc=com'
# msktutil -c --computer-name `hostname -s` -s HTTP/`hostname` \
-h `hostname` --server ad.example.com --no-pac
# chown kernun /etc/krb5.keytab
```

If clients should access the proxy via a name different from the fully qualified host name of the Kernun system, that name must be specified in the `-s` argument of the **msktutil** command. The same name must be configured in `kerberos-auth.proxy-host` or `session-acl.kerberos-auth`.

Instead of saving the created keytab into the default location `/etc/krb5.keytab`, another file name can by set by adding option `-k` *`keytab_file`* to the **msktutil** command line. The custom keytab location must be configured as a `shared-file` referenced from `kerberos-auth.keytab`.

If the system is to be removed from the domain later (when Kerberos authentication is no more required or if the system will be moved to another domain), remove file `/etc/krb5.keytab` and delete the machine account on the Active Directory Controller.

Initialization by Generating a Keytab Manually

The second method of Kerberos initialization comprises generating a keytab on the Active Directory controller, transferring the keytab to the Kernun system, and configuring Kernun to use the keytab. The initialization procedure consists of several steps:

1. Create a user account in the Active Directory. *The account must be configured so that its password never expires and it must have enough rigths to read user and group information from the Active Directory* (usually, membership in the `Domain Users` group suffices).

2. Generate the keytab using the command line on the domain controller:

   ```
   C:\> ktpass /out KEYTAB /princ HTTP/PROXY@AD_DOMAIN
   /mapuser USER@AD_DOMAIN /pass * /crypto All
   /ptype KRB5_NT_PRINCIPAL
   ```

   where the parameters are:

   KEYTAB The file name for the created keytab file.

PROXY The proxy name used by clients. If it it not the fully qualified host name of the Kernun system, the name must be configured in `kerberos-auth.proxy-host` or `session-acl.kerberos-auth`.

AD_DOMAIN The Active Directory domain name, in upper case.

USER The name of the user created in step 1. The command will request entering this user's password. Alternatively, the password may be specified on the command line by `/pass` *PASSWD* instead of `/para *`.

3. Copy the keytab to the Kernun system or to the computer running Kernun GUI.

4. Configure the keytab as a `shared-file` and add a reference to this shared file into `kerberos-auth.keytab`. The keytab shared-file should be stored in a location included in backup and upgrade, for example, `/usr/local/kernun/conf`.

5. Set `kerberos-auth.kinit` to the proxy principal name `HTTP/`*PROXY*.

Example: The keytab creation command will be, for the Kernun user account `kernun`, the Active Directory domain `EXAMPLE.COM`, the Kernun system `kernun.example.com`, and the output keytab file `krb5.keytab`:

```
C:\> ktpass /out krb5.keytab /princ HTTP/kernun.example.com@EXAMPLE.COM
/mapuser kernun@EXAMPLE.COM /pass * /crypto All
/ptype KRB5_NT_PRINCIPAL
```

> **Note**
>
> Configuring a keytab as a shared file makes easy distribution of the keytab to cluster nodes.

If Kerberos authentication is no more required, remove the keytab from the Kernun system and delete the user account from the Active Directory controller.

5.14.5 Kerberos Authentication in Transparent HTTP Proxy

As was already mentioned, web browsers usually refuse proxy authentication on a transparent HTTP proxy. This problem can be solved by redirecting an unauthenticated transparent HTTP request to an authentication server. The server performs authentication, remembers the association of an IP address to a user name in the out-of-band authentication table, and redirects the request back to the origin server. As the Kerberos authentication is done in the background, it is invisible to the user.

An example of a Kerberos transparent proxy configuration is stored in the sample configuration file `/usr/local/kernun/conf/samples/cml/kerberos-auth-transp.cml`. The proxy must be configured as an OOB authentication server by adding the `oob-auth` global configuration section and referencing it by item `http-proxy.oob-auth-srv`. Kerberos and optional LDAP configuration is like in the non-transparent case, see Figure 5.45.

The HTTP proxy configuration is shown in Figure 5.47 and Figure 5.48. Global `acl INTOK-AUTH` and the corresponding `session-acl INTOK-AUTH` handle transparent

requests to origin servers. These requests are authenticated using out-of-band authentication. Authenticated users are passed by `request-acl HTTP-OK`. If a user is not authenticated, the request will be redirected to the authentication server by `request-acl HTTP-RDR`. Redirected requests are handled by global `acl HTTP-AUTH` and the corresponding `session-acl HTTP-AUTH`. The client is asked for Kerberos authentication by `request-acl AUTH-REQ`. Authentication is performed by the browser without asking the user for a name or a password. Then `request-acl AUTH-OK` stores the IP address of the successfully authenticated user into the out-of-band table and redirects the browser to the originally requested URI. Further requests use OOB authentication and are handled without redirection to the authentication server.

```
# Transparently authenticated user
acl INTOK-AUTH
    service { HTTP };
    from { ^system.INT.ipv4.net };
    to transparent { * } port { 80 };
    accept;
# Authentication server
acl HTTP-AUTH
    service { HTTP };
    from { ^system.INT.ipv4.net };
    to non-transparent { * } port { 8080 };
    accept;
# Proxy for HTTP
http-proxy HTTP
    listen-on
        transparent ^system.INT.ipv4.host port 80;
        non-transparent ^system.INT.ipv4.host port 8080;
    document-root DOCROOT;
    oob-auth-srv OOB;
    # Transparently authenticated user (by OOB)
    session-acl INTOK-AUTH
        auth oob OOB;
    # Authentication server
    session-acl HTTP-AUTH
        auth none;
        kerberos-auth KERBEROS;
        authenticate-at server;
```

Figure 5.47: Transparent Kerberos authentication — session ACLs

```
# Transparent HTTP access, successful OOB authentication
request-acl HTTP-OK
    session-acl { INTOK-AUTH };
    user { * };
    accept;
# Transparent HTTP access, not authenticated
request-acl HTTP-RDR
    session-acl { INTOK-AUTH };
    user none;
    accept;
    rewrite /(.*)/ "http://fw.pha.tns.cz:8080/auth?$1" redirect temporary-same-method;
# Authentication successful
request-acl AUTH-OK
    session-acl { HTTP-AUTH };
    user { * };
    accept;
    rewrite /.*\?(.*)/ "$1" redirect temporary-same-method;
    oob-add
# Request authentication
request-acl AUTH-REQ
    session-acl { HTTP-AUTH };
    user none;
    accept;
    auth-req "Proxy authentication required";
```

Figure 5.48: Transparent Kerberos authentication — request ACLs

Important

In Firefox, Kerberos authentication must be explicitly enabled for each server, in our example `fw.pha.tns.cz`. It can be done by typing a special uri `about:config` into the address line to show advanced settings, and then setting the server name to option `network.negotiate-auth.trusted-uris`.

Important

Internet Explorer, Chrome and Opera users may be prompted for a password to authenticate to the authentication server. To avoid it they must configure the browser for automatic logon. To do this

- Go to the `Internet Options` dialog box, on the `Security` tab select `Local intranet`, and then click `Custom Level`.
- In the `Security Settings` dialog box, under `Logon`, select `Automatic logon only in Intranet zone`, and then click `OK`.
- In the `Internet Options` dialog box on the `Security Settings` tab with `Local intranet` still selected, click `Sites`.
- In the `Local intranet` dialog box, click `Advanced`.
- In the next dialog box (also titled `Local intranet`), type the URL of the authentication server http://fw.pha.tns.cz:8080/ in the `Add this Web site to the zone box`, and then click `Add`.

5.14.6 NTLM Authentication in HTTP Proxy

The NTLM authentication is an alternative to the Basic authentication described in Section 5.14.3. It provides a single-sign-on capability for Microsoft Windows clients authenticated in an Active Directory domain. If a user is logged in a domain, an NTLM-capable Web browser can authenticate the user to the proxy automatically, without asking the user for a name and a password. The NTLM authentication can be used instead of the Basic authentication, or both authentication methods can be used together. In an NTLM-only configuration, clients incapable of the NTLM authentication cannot authenticate. In a configuration with both authentication methods, an NTLM-capable client uses NTLM and other clients use Basic.

Important

If an HTTP proxy uses the NTLM authentication, it cannot be run chrooted, because it needs access to some system components that are not contained in the standard Kernun chroot environment.

Important

The DNS domain name of the Active Directory controller must correspond to the Active Directory domain name. For example, a domain controller for the domain `tns.cz` should be named like `ad.tns.cz`.

An example of an NTLM configuration is stored in the sample configuration file `/usr/local/kernun/conf/samples/cml/ntlm-auth.cml`. The `system` section contains global configuration related to the NTLM authentication in section `ntlm-auth`, see Figure 5.49. Two parameters are required: the name of the Active Directory domain (`domain`) and the address of the domain controller (`ad-controller`). The NTLM authentication itself provides the name of an authenticated user to the proxy. A `request-acl` can be selected according to the user name or group membership. For the latter option, the proxy needs to know the list of groups the user belongs to. The list can be obtained from an LDAP server. As the Active Directory contains all the necessary information and provides LDAP interface, the Active Directory controller is usually used as the LDAP server. Obtaining group membership can be configured by unhiding the hidden item `ldap` and section `ldap-client-auth` in the sample configuration in Figure 5.49. The URL of the LDAP server references the Active Directory controller. A user with the permission to access the Active Directory contents for reading must be specified in the `bindinfo` item in place of *`ADUser`*, and the corresponding password in place of *`ldap-password`*. The Active Directory stores user account data in a different format than other LDAP servers used for LDAP authentication. The Active Directory format must be selected by item `active-directory`, which also specifies the domain name.

```
# LDAP server used to obtain a list of group for an authenticated user.
hidden ldap-client-auth LDAP-AD
    server "ldap://ad.tns.cz";
    bindinfo "cn=ADUser,cn=Users,dc=tns,dc=cz" "ldap-password";
    active-directory "tns.cz";
# NTLM authentication
ntlm-auth NTLM
    domain "tns.cz";
    ad-controller "ad.tns.cz";
    hidden ldap LDAP;
```

Figure 5.49: NTLM authentication — section `system`

The NTLM configuration can be enabled in an HTTP proxy by item `session-acl.ntlm-auth`. Outcome of the NTLM authentication is processed like the Basic authentication. The user name, optional list of groups, or the fact that the authentication failed, can be used as conditions in `request-acl` sections. There is usually one or more request ACLs that permit access to successfully authenticated users and one request ACL that denies access to unauthenticated users and asks for authentication. The sample configuration in Figure 5.50 allows access to any authenticated user (`request-acl AUTH-OK`). Users that are not authenticated are requested to perform authentication by `request-acl AUTH-REQ`.

Administrators often want to enable the more user-friendly NTLM authentication and to provide an alternative for clients that are not capable of the NTLM authentication. Such a configuration can be created by setting both items `ntlm-auth` and `auth` in the same `session-acl`,

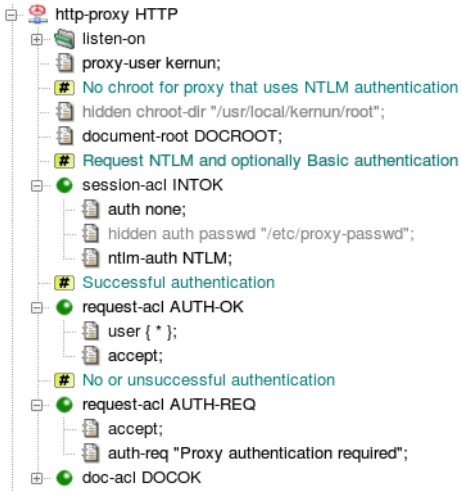

Figure 5.50: NTLM authentication — section `http-proxy`

for example, by unhiding the line "`hidden auth passwd ...`" in Figure 5.50. The proxy will offer both NTLM and Basic authentication schemes. The client can choose any of them[9].

After the NTLM authentication is configured and the configuration applied to a Kernun system, it is necessary to add the Kernun system to the Active Directory domain. A user account *`user`* in the Active Directory with `Domain Admins` rights is required for this task. The Kernun system is added to the domain by the shell command

```
# net ads join -U user
```

The system should then be rebooted in order to initialize and start the system components that provide an interface between the proxy and the Active Directory controller[10]. If the NTLM authentication is no more required, or if the Active Directory domain or the domain controller changes, the system should be removed from the domain by the shell command

```
# net ads leave -U user
```

It can be later added to another domain by **`net ads join`**. Membership of a Kernun system in an Active Directory domain can be tested by the shell command **`wbinfo -t`**. If the system has been successfully added to a domain, we get:

```
# wbinfo -t; echo $?
checking the trust secret via RPC calls succeeded
0
```

Otherwise, the output is:

[9] The client should choose the strongest implemented authentication scheme, in this case NTLM.

[10] Kernun uses Samba for communication related to the NTLM authentication between a proxy and the Active Directory.

```
# wbinfo -t; echo $?
checking the trust secret via RPC calls failed
error code was NT_STATUS_NO_TRUST_SAM_ACCOUNT (0xc000018b)
Could not check secret
1
```

Results of NTLM authentication can be cached by out-of-band authentication (see also Section 5.14.8), in order to decrease load of Active Directory and LDAP servers. Each new client is authenticated by NTLM. The combination of the client IP address, the user name and the list of groups is remembered in the OOB session table. Following requests from the same IP address will be authenticated as the same user and groups, without contacting the AD controller and the LDAP server. Computer accounts (i.e. users whose name ends with a dollar) are not added to the OOB session table.

Configuration of NTLM caching consists of adding the global section `oob-auth OOB`, referencing it by item `http-proxy.oob-auth-srv`, and adding item `auth oob OOB` to each `session-acl` that contains item `ntlm-auth`. Cached user and group information for a client IP address expires after a timeout controlled by items `ntlm-auth.timeout-idle` (expiration after a period of inactivity) and `ntlm-auth.timeout-unauth` (unconditional expiration).

Tip

The preferred authentication method in Active Directory environment is Kerberos, see Section 5.14.4.

5.14.7 HTTP Authentication Proxy

The HTTP proxy can be used to control access to a Web server in the protected network[11] from clients in the external network. A typical situation is a Web interface to a corporate mail server accessed by employees from machines in the Internet, for example from their homes. In such a case, the security policy often requires the use of sessions with limited lifetime authenticated by a strong mechanism, such as a challenge-response protocol that utilizes one-time passwords generated by hardware authentication tokens. Also, as we mentioned in Section 5.14.3, HTTP proxy authentication does not often work if the proxy is not configured in the browser explicitly. For such situations, there is a special operation mode of the HTTP proxy, called the *HTTP authentication proxy*, or *AProxy* for short.

A typical AProxy configuration is displayed in Figure 5.51. It is a configuration for access from clients in the Internet to the server `intweb.tns.cz` in the internal network. As the Internet is an untrusted network, the connections from clients are secured by TLS, which is enabled by the `ssl-params APROXY-SSL` section and the `client-ssl` item in `session-acl APROXY-EXT`. The internal network is considered secure, therefore plain unencrypted HTTP is used. (For detailed explanation of SSL/TLS in Kernun UTM, see Section 5.13.) The configuration section

[11] usually in the internal network or in the demilitarized zone

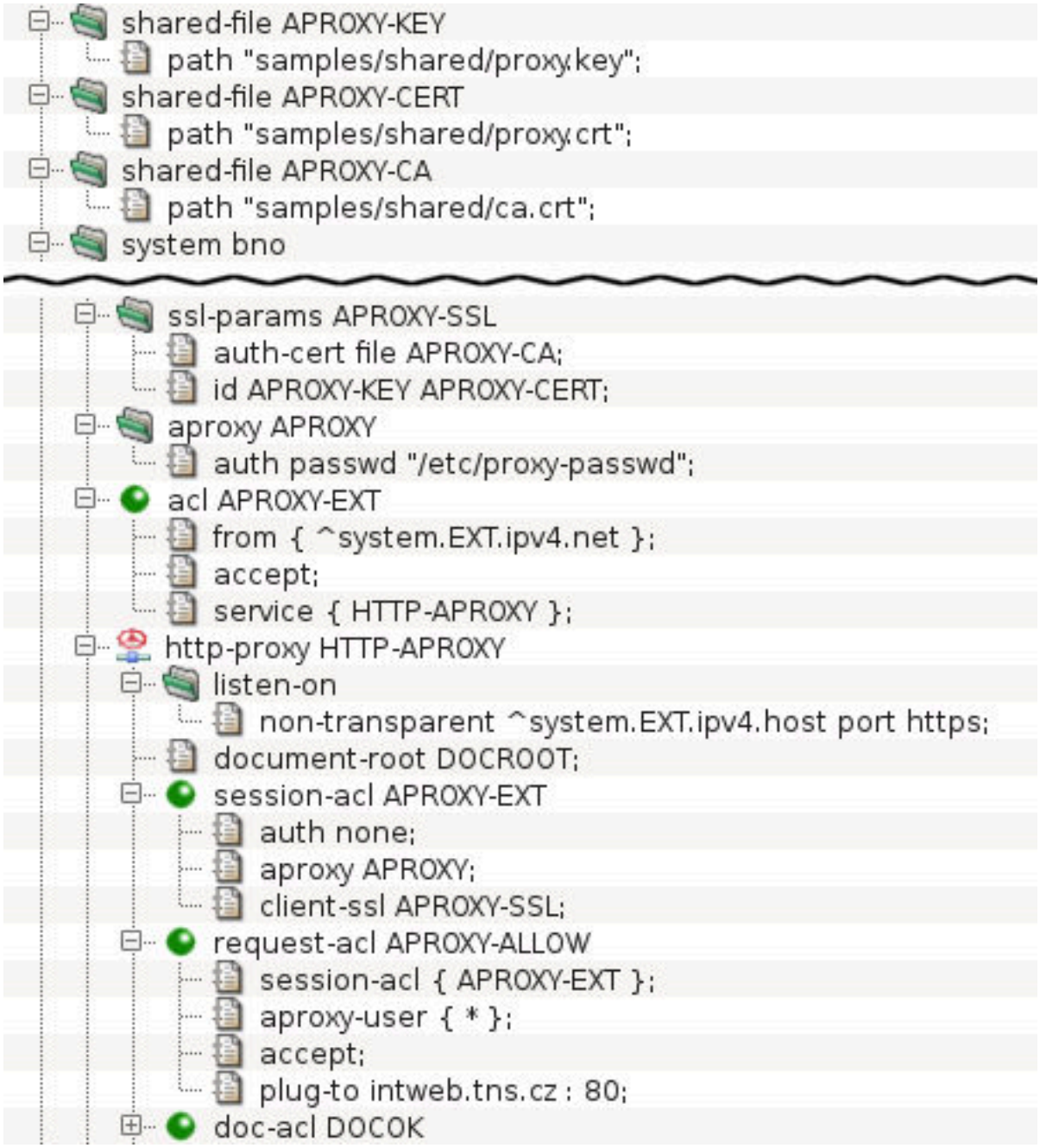

Figure 5.51: Configuration of the HTTP authentication proxy

`aproxy APROXY` defines various parameters of the AProxy. The example contains only the `auth` item, which defines that the AProxy will verify user credentials against a local password file. The default maximum session time after a successful authentication is one hour. After that time, the user is logged out and must authenticate again. Also, if the session is idle—the client does not issue any request—for more than 5 minutes, the user must reauthenticate. These default timeout values can be changed by items `timeout-unauth` and `timeout-idle` of section `aproxy`. Further configurable AProxy parameters are described in http-proxy(5). The AProxy mode is activated by the `aproxy` item of a `session-acl`. The user name obtained during AProxy authentication can be used to select a `request-acl`, by matching an `aproxy-user` item. In the sample configuration, a client accesses the AProxy by connecting to the external IP address of Kernun UTM. If authentication succeeds, the HTTP requests from the client are passed to the internal server defined by `plug-to`.

Warning

An AProxy session cookie is a sensitive piece of information and should be kept secret, because an attacker could use it to steal the identity of an authenticated user. Hence, the AProxy is usually configured so that the connections between clients and the proxy are secured by SSL/TLS (see Section 5.13). It is possible to operate the AProxy using plain HTTP by enabling `aproxy.insecure-cookies` and not configuring `client-ssl` on the client side of the proxy, but it is strongly recommended not to so unless the clients are connected to Kernun UTM via a trusted network.

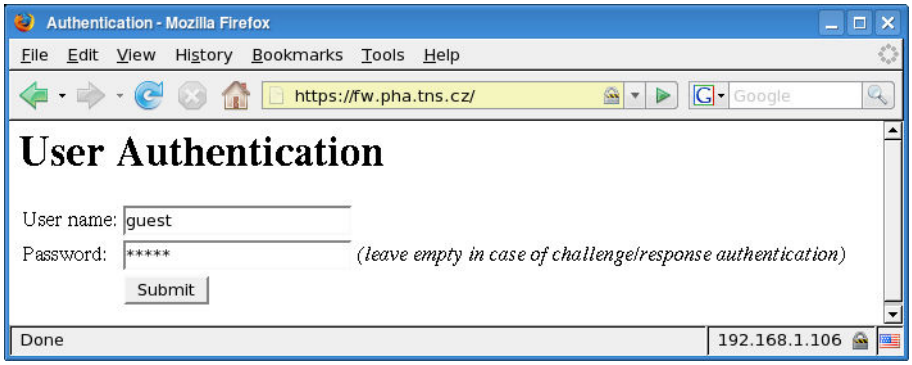

Figure 5.52: The user authentication dialog of the HTTP authentication proxy

A user starts by entering an AProxy URL to a Web browser. The AProxy returns an authentication dialog, as displayed in Figure 5.52. The user fills in their login name and password and clicks Submit. If a challenge-response authentication protocol is used[12], the user must fill in the login name only. The AProxy then displays a challenge and the user enters the response (obtained, e.g., from a hardware authentication token). If the authentication succeeds, a new session is initiated and the AProxy forwards the original HTTP request to the destination server.

[12] implemented only for authentication method RADIUS

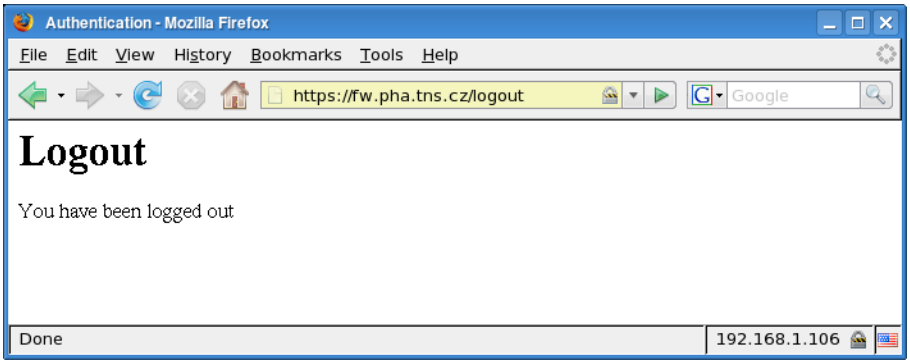

Figure 5.53: The logout confirmation message of the HTTP authentication proxy

Further requests do not require reauthentication and are transparently passed to the server. The user must authenticate again only if the session expires because of a timeout. It is also possible to log out and terminate the session explicitly by accessing a special logout URL, as shown in Figure 5.53.

The HTML authentication forms displayed by the AProxy are very simple, but their appearance is easy to modify. Form templates are stored in the `/usr/local/kernun/conf/samples/error_documents` directory. The template files are: `aproxy-password-form.html` (the form with user name and password fields), `aproxy-password-form.html` (the form with a response field for challenge-response authentication), and `aproxy-logout.html` (the logout message).

Important

The AProxy uses cookies to track sessions. The cookies are added to HTTP response messages by the proxy, but they appear to the client as coming from the server. The session cookies are then extracted by the proxy from the requests. As a cookie received from a server is never sent to other servers, it is impractical to use the AProxy for authentication of clients in the internal network accessing Web servers in the Internet.

5.14.8 Out of Band Authentication

Many application protocols do not support user authentication on a proxy. Even in protocols with a proxy authentication mechanism it may be undesirable to use it in some situations. Nevertheless, we may still want to perform access control and network usage monitoring for individual users. The *out-of-band authentication (OOBA)* provides a solution in such cases. It is based on the fact that for any network protocol, the proxy knows the IP address of the client machine. The OOBA simply binds user names with IP addresses. There is an OOBA server, which maintains a table containing pairs of user names and IP addresses. When a proxy accepts a new network connection, it sends the client IP address to the authentication server and receives the corresponding user name or the information that no user is authenticated from that IP address. As is the case with other authentication methods, the user name can be used as a condition for ACL selection.

Note

As the out-of-band authentication cannot distinguish connections from a single IP address, more than one user cannot be authenticated on a single client machine at the same time. For this reason, computer accounts (i.e. users whose name ends with a dollar) are not automatically added to the OOB session table. On the other hand, one user can be authenticated on many machines simultaneously.

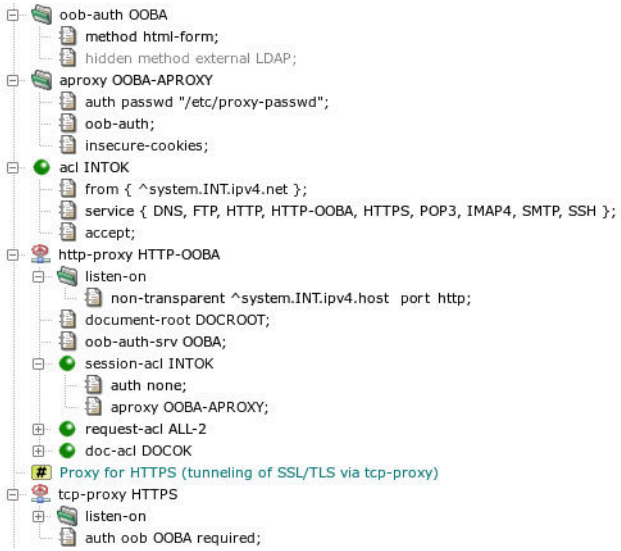

Figure 5.54: Configuration of the out-of-band authentication

A sample configuration of the OOB authentication is displayed in Figure 5.54, with `http-proxy HTTP-OOBA` serving as the OOB authentication server, and `tcp-proxy HTTPS`, which allows access only to users authenticated by the OOBA. The `oob-auth OOBA` section defines parameters of the OOB authentication. Only the authentication method is specified in this example. See auth(5) for description of all available parameters. This session is referenced in `http-proxy HTTP-OOBA` by the `oob-auth-srv` item. The HTTP proxy is switched to the OOBA mode by adding the `oob-auth` item to an `aproxy` section and referencing this section by a `session-acl`. In the sample configuration, a password file is used to check user names and passwords. As the OOBA is performed in the internal network, which is assumed to be secure, plain HTTP without SSL/TLS is used, which is why `insecure-cookies` needs to be added to the `aproxy` section.

The `html-form` OOB authentication method is selected in the sample configuration. It authenticates users interactively in the same way as the AProxy (Section 5.14.7). The alternative option, `external` (marked as `hidden` in the sample configuration), is to import the list of authenticated users from a server in the network. The external OOBA is fully transparent. Users do not have to do anything special, they just log in to the server that provides data to the Kernun UTM OOB authentication server. Out of the box, Kernun UTM contains the **ooba-samba**(1) script, which implements integration with Samba.

The `tcp-proxy HTTPS` proxy uses the OOB authentication and accepts authenticated users only. The OOBA is activated by setting the `oob` method in the `auth` item. The OOBA server and proxies that provide OOBA communicate via a session table file. Therefore, they must all reference the same file in the `oob-auth` sessions they use. If no file is specified, as in our example, a common default file will be used. The authenticated user name could be matched by a `session-acl`. Another way to reject unauthenticated users without creating a denying ACL for them is to add the `required` parameter to the `auth` item. It causes immediate termination of sessions initiated by unauthenticated users, even before ACL matching begins.

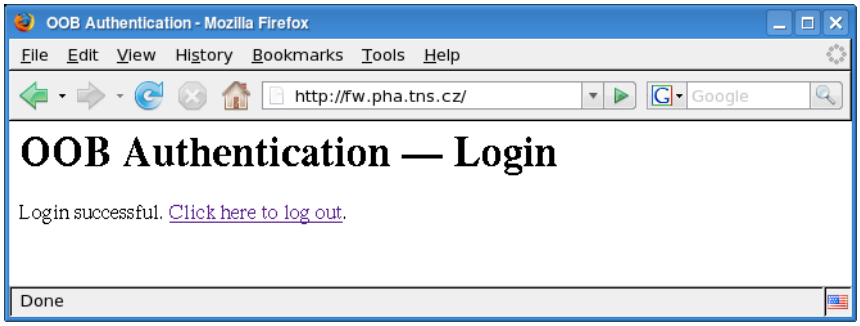

Figure 5.55: Login to the OOB authentication server

When the `html-form` OOBA method is used, a user must access the authentication page (Figure 5.52) by opening the OOBA server URL in a Web browser. After entering a valid name and password, a login confirmation page is displayed, see Figure 5.55. It contains a link that can be used to log the user out. Like in the case of AProxy, the authentication pages displayed by the OOB authentication server can be changed by modifying template files in `/usr/local/kernun/conf/samples/error_documents`. There are two more OOBA authentication forms, `ooba-login-confirm.html` (login confirmation) and `ooba-logout-confirm.html` (logout confirmation), in addition to the AProxy ones, which are reused.

5.15 Antivirus Checking of Data

Kernun UTM can cooperate with the Dr.WEB or ClamAV antivirus programs and any other antivirus program that supports ICAP protocol, e.g. Kaspersky Anti-Virus for Proxy Server. This tutorial describes the configuration of Kernun UTM only (that is, not the installation and configuration of the antivirus programs). The complete configuration file is located in `/usr/local/kernun/conf/samples/cml/antivirus.cml`.

5.15.1 Connecting with ClamAV

The **ClamAV** antivirus program can be installed either directly on Kernun UTM or on any other machine connected via the network. The administrator of Kernun UTM can choose either of two ways to transport files between Kernun UTM and ClamAV:

- `clamav-net`: Files to be checked by the antivirus are sent to the antivirus via the TCP connection.

- `clamav-file`: Files to be checked by the antivirus are stored on the local file system. This option can be only used if the antivirus program is running on the same machine as Kernun UTM. The directory where the files are stored is defined by the `comm-dir` element (which defaults to `/data/tmp/antivirus`). Kernun UTM does not create the directory (it must be explicitly created by the administrator), and proper permissions need to be set (i.e., the directory must be writable by `proxy-user`).

The IP address and the port the antivirus program listens on are specified in the section `antivirus` on the `system` level of Kernun UTM's configuration, as shown in Figure 5.56 (we suppose ClamAV listens on localhost's port 3310). It is possible to limit the size (`max-checked-size`) of files scanned by the antivirus program. In the sample configuration, it is 1 MB. Larger files are not scanned and the antivirus module immediately reports the result as `SKIPPED`.

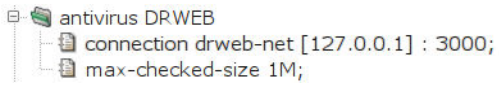

Figure 5.56: Configuration of the connection to the antivirus program

5.15.2 Connecting via ICAP protocol

The Internet Content Adaptation Protocol (ICAP) is a lightweight HTTP-like protocol used to communicate with antivirus program. Kernun UTM makes use of it's Response Modification Mode to send data to be checked. Antivirus program sends back it's test results.

The IP address and the port the antivirus ICAP server listens on are specified in the section `antivirus` on the `system` level of Kernun UTM's configuration, as shown in Figure 5.57 (in this example it's `10.0.0.33` and standard ICAP port, `1344`). The third parameter of `connection` item (`/av/respmod` in this example) is an URI of the antivirus' Response Modification Mode handler. Again, it is possible to limit the size (`max-checked-size`) of files scanned by the antivirus program.

Figure 5.57: Configuration of the connection to the antivirus program via ICAP

5.15.3 Antivirus Results

There are five possible results of a check of content by the antivirus:

Free The antivirus has scanned the data and has not found any virus.

Found At least one virus has been found.

Skipped The antivirus has not scanned the data. Either the antivirus itself has decided not to scan, or the file has been larger than the limit specified by `max-checked-size`. No virus has been found, but the antivirus has not confirmed that the data is virus-free.

Unknown The proxy has received a result from the antivirus, but does not understand it. It is not known whether there is any virus in the data, or not.

Error The proxy cannot communicate with the antivirus. This is usually caused by the antivirus not running or by misconfigured antivirus connection in the proxy.

5.15.4 Antivirus in Proxies

The antivirus program can be used for online scanning of the content transferred via `ftp-proxy`, `http-proxy`, `imap4-proxy`, `pop3-proxy` and `smtp-proxy`. There is a slight difference between mail-processing proxies (`imap4-proxy`, `pop3-proxy` and `smtp-proxy`) and the other two (`http-proxy` and `ftp-proxy`). In the latter case, a special functionality is implemented that prevents clients from reaching a timeout while very long files are being scanned. The client is fed with chunks of the file at specified intervals until the scanning of the file is completed.

Document scanning for the HTTP and FTP proxies is configured in their `doc-acl`. The `antivirus` item specifies the name of the `antivirus` section to be used. The `interval`, `chunk` and `limit` items can be used optionally to specify that if scanning takes more than a certain time (5 seconds in the example), a chunk of the (as yet unscanned) file of a certain size (up to 2,000 bytes) is to be sent to the client at a certain interval (5 seconds). By default, only documents for which the antivirus returns the result `free` are passed by the proxy. The `doc-acl.accept-antivirus-status` item can be used to specify additional result codes, for which the checked data are to be passed, in addition to `free`.

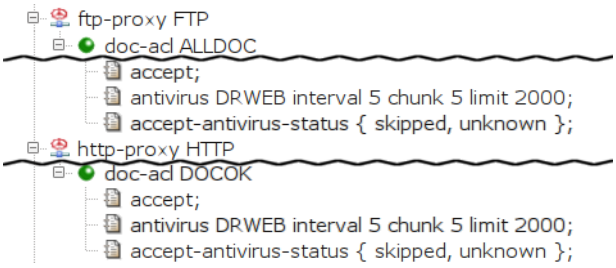

Figure 5.58: Use of antivirus in the FTP and HTTP proxies

5.15.5 SMTP Proxy: Discarding Infected Mails

We will describe two variants of `smtp-proxy` configuration, differing in the way they handle infected mail. In this section, we show the case when infected mail is discarded, i.e. not sent to the recipient(s), but stored in quarantine.

In smtp-proxy, the item use-antivirus is used to define antivirus.

Infected mail is detected in the mail-acl sections. One of them, a mail-acl accepting all e-mails (MAILOK), already exists in the Kernun UTM configuration. We will create more to process messages with various antivirus scan results (see Figure 5.59). When smtp-proxy processes mail, the first matching ACL is used. We must therefore place new mail-acl sections before the existing MAILOK.

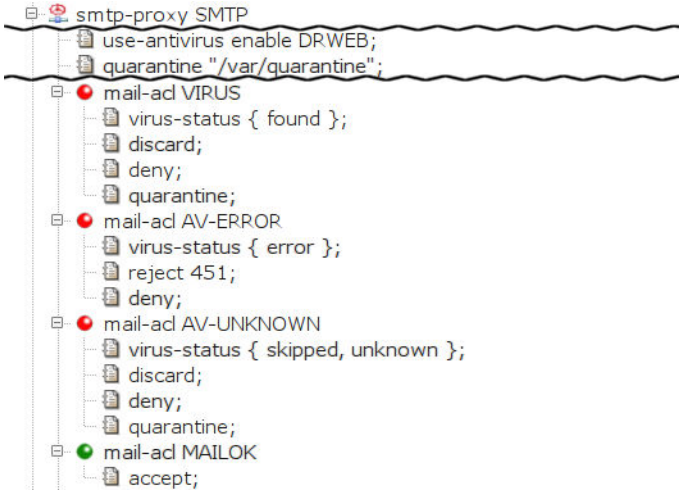

Figure 5.59: Discarding infected messages in the SMTP proxy

The first added mail-acl, VIRUS, discards infected mail and puts it in quarantine. Its only entry condition is virus-status. It is matched if the virus-status of the message is found (i.e., it contains a document that is infected by a virus). We define three actions: discard the message (i.e., the sender is not returned an error), deny it (the message is not sent to the addressee) and store the e-mail in quarantine.

The quarantine directory must be defined (the quarantine item in the smtp-proxy section) and created in the file system (e.g., /usr/local/kernun/root/var/quarantine, keeping in mind the smtp-proxy runs in chroot), and writable by kernun-user. For more information about ways of handling mail stored in quarantine, see quarc.sh(1).

There are two more mail-acl sections. AV-ERROR reports a transient error to the client if the the antivirus is unable to check the message. The client can retry sending the mail later. AV-UNKNOWN discards and quarantines the message if the antivirus does not scan it or if the proxy cannot understand the antivirus' reply.

5.15.6 SMTP Proxy: Replacing Infected Documents

Contrary to the previous example, this section describes a way to prevent infected mail from being discarded. This example refers to smtp-proxy SMTP-2, which is marked hidden in the sample file[13].

[13] Keep either SMTP, or SMTP-2 hidden—they are not designed to work in parallel.

This time, `smtp-proxy` removes the infected documents from the message and delivers the e-mail to the addressee, sending a BCC copy to a special (administrator's) e-mail address. Moreover, the subject is changed to make it obvious that a virus was removed from the message. `smtp-proxy` also stores infected mail in quarantine. The configuration is depicted in Figure 5.60

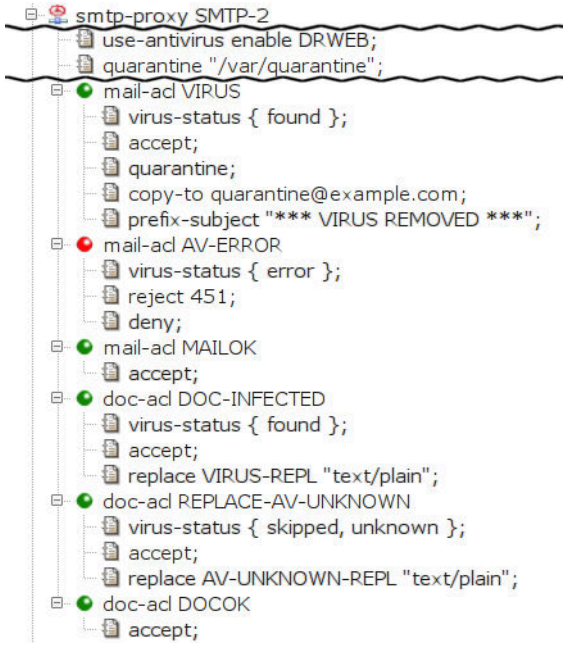

Figure 5.60: Replacing infected documents in the SMTP proxy

The removal of infected document is ensured by creating a special `doc-acl DOC-INFECTED` section that matches the infected documents. We place it in front of the existing `doc-acl DOCOK` in order to match before the more generic `doc-acl DOCOK`. The entry condition `virus-status found` limits the acl scope to the infected documents (documents marked by the antivirus program as containing a virus). We add the action item `replace` so that the matched (and therefore infected) document is replaced by the given file. The administrator must explicitly create this file if it does not exist. Because the proxy is configured to run in chroot environment, the file must be stored in the particular subdirectory of the directory specified as `chroot-dir` (in this case, `/usr/local/kernun/root/etc/shared/error_documents/`).

The special behavior for infected e-mails (i.e., the messages that contain an infected document) is defined in `mail-acl VIRUS`. The entry condition `virus-status found` limits the acl scope to infected e-mails. The `accept` item specifies that the e-mail should be delivered to the addressee. We specify that the message is to be stored in the quarantine, that a BCC copy be sent to a special e-mail address (the `copy-to` item) and that the subject be prefixed with a specified text (the `prefix-subject` item).

Like in `SMTP`, there is an additional `mail-acl AV-ERROR` that returns a transient error to the client if the mail cannot be checked by the antivirus. Finally, `doc-acl REPLACE-AV-UNKNOWN` replaces documents that are skipped by the antivirus.

5.15.7 Antivirus in POP3 and IMAP4 Proxies

In this section, we show the POP3 and the IMAP4 proxies configured to replace infected documents. See the Figure 5.61 and Figure 5.62

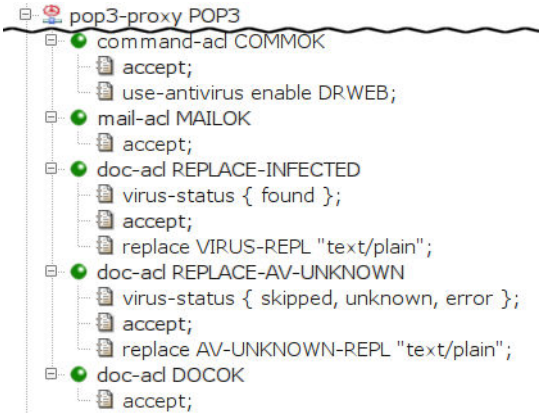

Figure 5.61: Replacing infected documents in the POP3 proxy

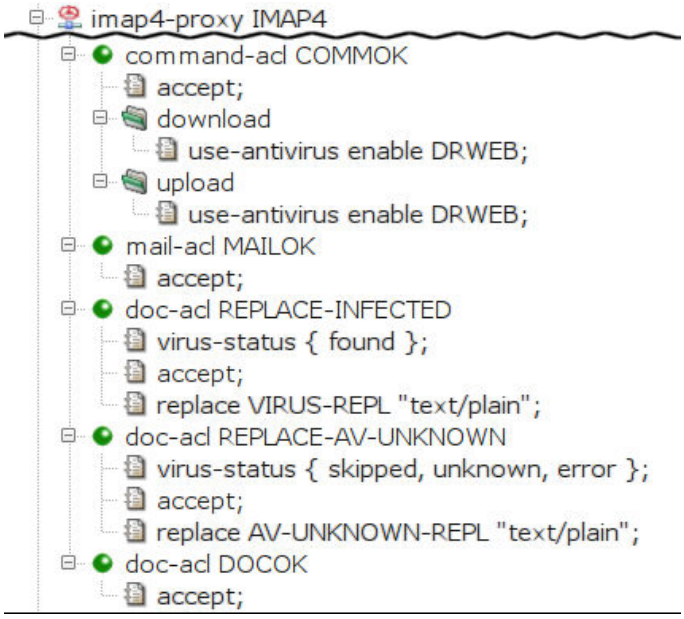

Figure 5.62: Replacing infected documents in the IMAP4 proxy

First, we instruct Kernun UTM to scan the documents transferred through the proxy with the antivirus by specifying the item `use-antivirus`. The item is specified directly in `command-acl` for the POP3 proxy, whereas for IMAP4 proxy, it is specified separately for the `download` and for the `upload` case.

In the second-level acl (`mail-acl MAILOK`), we specify that all e-mails should be accepted. In the third-level acl, we specify that we want to replace infected (`doc-acl REPLACE-INFECTED`)

and unscanned (`doc-acl REPLACE-AV-UNKNOWN`) documents, and to accept all other documents (`DOCOK`)[14].

5.16 Antispam Processing of E-mail

Kernun UTM can cooperate with the SpamAssassin antispam software to reject or mark spam. Three more methods can be used in addition to the SpamAssassin protection: black-listing (rejection of mail from senders with addresses listed in an external database), white-listing (authorisation of the sender on his or her domain server) and grey-listing (an automatic method based on a local database). We will discuss the configuration of antispam first, and then briefly describe the individual methods in the next sub-chapter.

5.16.1 Antispam Engine

Like the Dr.WEB or ClamAV antivirus programs, you can install SpamAssassin either directly on Kernun UTM or on any other machine connected via the network. The antispam configuration in Kernun UTM is analogous to the antivirus configuration. We define the `antispam` section on the `system` level, and we reference this section in the mail processing proxies (`smtp-proxy`, `pop3-proxy`, and `imap4-proxy`). We can use the results of the antispam check in the `spam-score` item[15] in the `mail-acl` section of the spam-checked proxy.

Antispam with POP3

Having configured the `SPAMASSASSIN` section, we now reference it in the proxies. We replace the item `no-mail-scanning` (in the `command-acl` section) with `use-antispam enable SPAMASSASSIN 100K`.

> **Tip**
>
> Antispam checking is a very demanding operation, and typical spam mails are quite small. You may therefore want to check only messages below a certain limit size, such as 100 kB in our case.

Because we deleted the `no-mail-scanning` item, Kernun UTM will search for matching `mail-acl` and `doc-acl` sections and we must therefore create them. (Otherwise, Kernun UTM's default reaction would be to reject the mail.) We create a `mail-acl` section that will mark spammed e-mails, but deliver them to the recipient(s). We do so by adding an acl condition `spam-score` and setting it to accept e-mails with a spam score exceeding 5.0 (we use the relational

[14] Note that the file referenced in the `replace` item must exist in the system (if it does not exist, it must be created).

[15] In fact, the score returned by SpamAssassin is multiplied by 1000, because the Kernun UTM configuration does not handle floats. This feature also allows the future compatibility with other antispam software.

operator `ge`; the Kernun UTM spam score of 5000 corresponds to SpamAssassin's score of 5.0 multiplied by 1000). We specify that such messages will be accepted, but define a `prefix-subject` item that adds a text prefix to their subject. Finally, we add accepting `mail-acl` and `doc-acl` and we get the configuration shown in Figure 5.63.

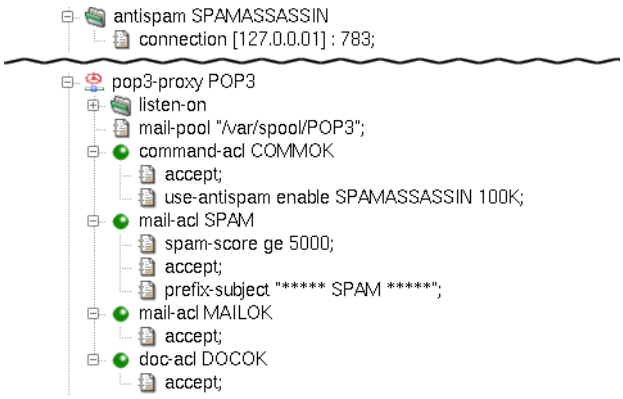

Figure 5.63: Antispam protection for POP3 proxy

Antispam with IMAP4

Analogously, we can add antispam support to the `IMAP4` proxy. We add the `use-antispam` items to both `download` and `upload` sections, and then create the `mail-acl` and `doc-acl` sections. Figure 5.64 shows the antispam configuration of `imap4-proxy`.

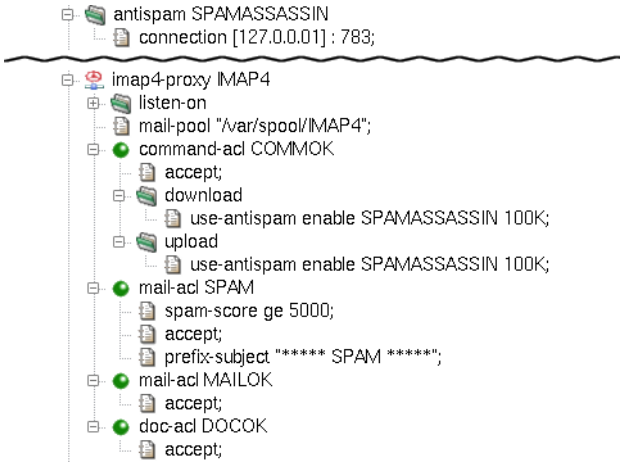

Figure 5.64: Antispam protection for IMAP4 proxy

Antispam with SMTP

The configuration of `smtp-proxy` is, again, similar to the previous cases, only this time we define the `use-antispam` item right inside the proxy. In this example we show how to deny mails with too high a spam score and store them in quarantine on the server. We do so by

specifying a `quarantine` item in the proxy and setting it to the path of the quarantine directory. Remember that we work under a chrooted environment, so we need to create that directory inside the chroot (in our example, `/usr/local/kernun/root/var/quarantine`). We add a `mail-acl SPAMQUARANTINE` with items `deny` to not deliver messages with SpamAssassin score exceeding 10, `discard` to not inform the sender about the delivery failure and `quarantine` to place the mail into the quarantine directory. Another `mail-acl`, `SPAMMARK`, is used to pass e-mails with the score of 5-10, but mark it with a subject prefix. The relevant part of the SMTP proxy configuration with antispam is shown in Figure 5.65.

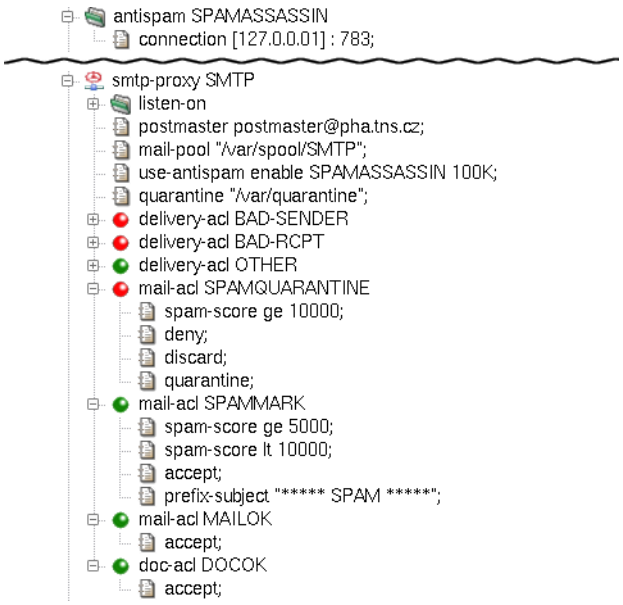

Figure 5.65: Antispam protection for SMTP proxy

The resulting configuration of mail proxies with antispam is available in the `/usr/local/kernun/conf/samples/cml/antispam.cml` sample file. For more information on antispam configuration, see the mod-antispam(5) manual page.

5.16.2 White-, Grey-, and Blacklists

Kernun UTM's `smtp-proxy` provides configurable client identity checking as protection against unsolicited mail. White-listing implemented in Kernun UTM is based on the Sender Policy Framework (`http://www.openspf.org`), which checks if the sender is allowed in the sender policy of the return path domain and thus authorizes the sender. After inserting a `white-listing` item into `session-acl`, the result of the matching can be used in the `delivery-acl` section using the `spf` condition. On the other hand, the black-listing method checks the sender's address against an external database of forbidden IP addresses. The sender's presence in such a database (for example `http://www.spamhaus.org`) means that the e-mail is denied during the `session-acl` phase. The configuration of black- and white-listing is depicted in Figure 5.66. The list of the main black-listing databases is defined in `/usr/local/kernun/conf/samples/include/smtp-blacklist.cml`.

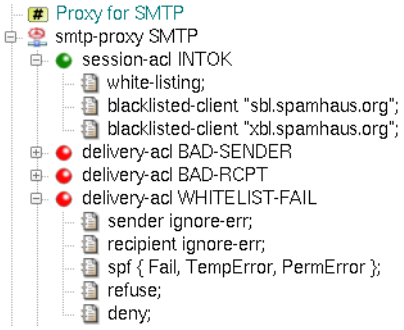

Figure 5.66: White- and black-listing for SMTP proxy

Grey-listing is a mail-filtering method that does not rely on any external database or domain information. Instead, it creates its own database of triplets (sender IP, sender e-mail address, recipient e-mail address) and behaves according to the state of triplet in the database. New e-mail is saved as `blocked` and a temporary error answer is sent to the sender. If the sender tries to redeliver the mail within a predefined period (e.g., after some time, but not too late), the state of the triplet changes to `granted` and all mail with the same triplet will then be passed without any blocking time. The grey-listing method is configured by the `grey-listing` section in the `delivery-acl`, and the `grey-listing` section with a `path` item set to the filename of the grey-listing database in `smtp-proxy`. A sample grey-listing configuration is shown in Figure 5.67. For more detailed information about the method, see the description of the triplicator(1) tool.

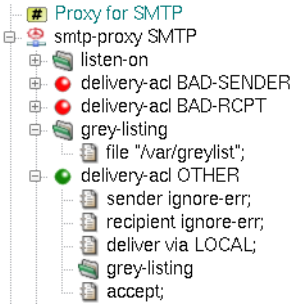

Figure 5.67: Grey-listing for SMTP proxy

5.17 Content Processing

Kernun UTM provides tools to investigate and filter the contents of the passing packets. These tools make it possible for the administrator to filter the contents of the HTML and mail traffic.

5.17.1 Content Type Detection

The third-level ACL, `doc-acl`, has as an entry condition `mime-type` specification (see access-control(7), configuration(7) manual pages). The content type detection can be performed

using several different methods; the administrator can set the order of the used methods globally for the whole proxy, or separately for each of its doc-acls.

The following content-type detection methods are available:

content-type (Original Content-Type) — the proxy uses the type declared by the document originator in the Content-Type header;

extension (File Name Extension Mapping) — the proxy tries to guess the MIME type from the name of the document (if specified);

magic (Magic Number Recognition) — the proxy reads the initial block of the document and tries to guess the file type from it with help of the magic number file (see magic(5) manual page).

Note

Method magic is skipped in the http-proxy when the response is partial and does not contain its beginning. Partial HTTP response either contains HTTP header Range or has content-type multipart/byteranges. The method magic is always skipped when the content-type is multipart/byteranges. Item request-acl.delete-req-hdr-range can be used to make the server send the entire response.

The common usage of mime-type conditions is in the http-proxy, where the administrator can forbid selected types of documents (e.g., video, applications). In our example, we will start with the initial configuration file, as shown in Section 4.2, and deny all documents recognized as any of the *video* mime-types for the http-proxy HTTP.

First, we need to specify the order of the used Content-Type detection methods. We add the doctype-identification section into the http-proxy HTTP, insert the order item into it and select the intended methods in its detail. In order to do so, we append three values magic, extension and content-type into the doctype-ident-method-list order field, leaving the field direction-set for unchecked (it can be used to define different order for each of the traffic directions — upload and download).

The magic and extension methods need further configuration: the former a magic file and the latter a file that contains the extension to the content-type mapping database. We specify them by adding two shared-file items pointing to the corresponding configuration files. In our example, we will use the sample configuration file for magic located in samples/shared/magic and the extension configuration file located in samples/shared/mime.types. We configure the selected methods to use these shared files by inserting two new items into the added doctype-identification section. The magic item points to the name of the magic configuration shared-file and the mime-types item to the name of the extension configuration shared-file.

Having configured the order of the used Content-Type detection methods, we can proceed to the filtering of all video documents. We do so by inserting new doc-acl called VIDEO into the http-proxy HTTP section. We will add it right above the doc-acl DOCOK section, so

that VIDEO takes precedence. We restrict the ACL to the video MIME type by inserting the mime-type option. We define the set of matching MIME types in str-set type. In this example we insert a single item, video. In a more complex situation, regular expressions can be used to define all the types to be matched for the ACL. Finally, we need to add the deny item into the inserted doc-acl VIDEO. Figure 5.68 shows the relevant part of configuration.

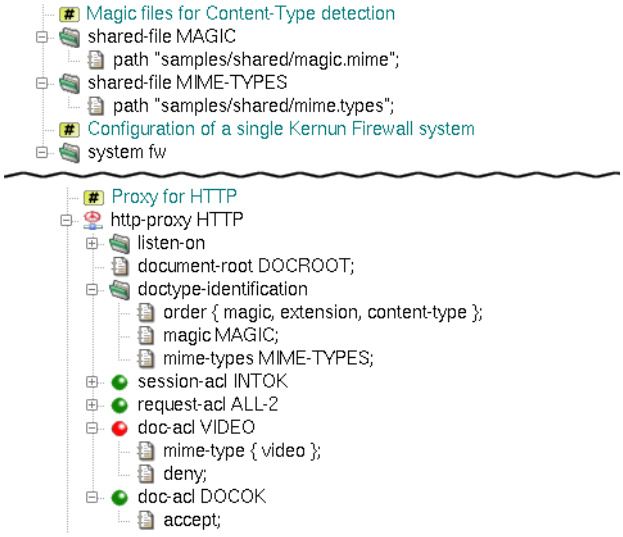

Figure 5.68: Content Type detection configuration for HTTP proxy

The resulting configuration file can be found among Kernun UTM samples under the name doctype-detection.cml in /usr/local/kernun/conf/samples/cml. For more detailed information on Content Type detection, see the doctype-identification(7) manual page.

5.17.2 HTML Filtering

Besides other tools used to filter whole packets and documents (URL filtering, antivirus checking, etc.), Kernun UTM provides an HTML filter that filters (or replaces) elements and attributes inside an HTML document. The filter is applied to every passing document on the third level of ACL processing, so it is applicable with all proxies that have the doc-acl section (http-proxy, imap4-proxy, pop3-proxy, etc.). The filter gets the whole HTML document, and deletes (or replaces) undesirable elements and attributes according to the specified rules. If there are no rules in the HTML filter, the document will be passed "as-is". If we create a rule of some type, all the matching elements (or attributes) will be accepted/denied according to the matching rule. If no rule matches, but there is a rule of the element/attribute type specified, the element/attribute will be denied.

For example, the administrator may want to deny all the Adobe Flash animations and replace all the URIs that refer to "suspicious" Web sites. We create a system-level html-filter section and name it HTMLFILTER. Now we need to add rules that will filter the elements and attributes of the document. First, we want to filter all the Flash animations. These can be contained in two HTML elements, embed and object, so we need to delete these two elements, but not all

of them – only those with the `application/x-shockwave-flash` Content-Type. We add an `embed-tag-type` item, which represents a filter rule that applies to `embed` tags with the corresponding content type. Now, we need to specify the content types we want to filter (either as whole Content-Type names or using regexps) and the action that is to be done (`accept` or `deny`). We append one value, "application/x-shockwave-flash" to `str-set val` and select `deny` from the `action` combo box. Now, we have filtered all the `embed` HTML elements with flash animations. We can do the same with the `object` elements using the `object` filter item set to the same values.

The items added so far filter out the undesirable contents. However, it may be useful to warn the users that we have changed the documents they are viewing. We can do so using the `replace-` filter items, which define the content that will replace a deleted element/attribute. For elements that can appear in both the `head` and `body` HTML elements (such as `embed` and `object`) we can set the replace text separately for each of the two cases. In our sample, we will create a `replace-body-embed-tags` item and set its value to "Flash embed tag DENIED" and a `replace-body-object-tags` item and set its value to "Flash object tag DENIED ".

We must not forget to create accepting rules for both `object` and `embed-tag-type` that will accept all the Content Types. Otherwise, the previously added rules would delete all the `embed` and `object` elements.

Clickjacking protection can be implemented by filtering IFRAME elements. This is done by adding `iframe-tag-src` rules, which follow the convention stated above. The SRC of the IFRAME element is being matched as a regexp (`/^http:\/\/([^.]+\.)*kernun.com\//` in the example matches the domain `kernun.com` and its subdomains). The replacement text can be set by `replace-iframe-tags`.

The HTML filter provides another type of useful rules: attribute filters. These can be used to delete (or replace) whole attributes from the document. In our example, we want to replace all the "suspicious" URI attributes with a neutral one. We will add a new `uri` item rule and set it to `deny` the undesired URI regexps, in our case `/.*photo.*/`, `/.*video.*/` and `/.*warez.*/`. Again, as we have created a new filter rule, we need to allow all other URIs that do not match this rule. We add a new `uri` item and set it to `accept` all the URIs (`*`). Now we add the replacement for the whole URI attribute (not only the value that is matched in the `uri` item) by inserting a new `replace-uri` item rule and setting it to a neutral URI, for example `"href='http://www.kernun.com'"`[16].

Having created the whole HTML filter, we can use it in any proxy that has a third-level ACL `doc-acl`. We tell the proxy to use an HTML filter by inserting an `html-filter` item with the filter's name into the proxy's `doc-acl`. The use of a slightly more complex HTML filter blocking all the Flash documents and the specified URIs in the `http-proxy HTTP` is depicted in Figure 5.69.

The complete resulting configuration can be found in `/usr/local/kernun/conf/samples/cml/html-filter.cml`. More complex HTML filter rules are created in `/usr/local/kernun/conf/samples/include/html-filter.cml`.

[16] This rule might replace attributes other than `href` (such as `src` or `action`) with a `href`, but such a change will not "damage" the document more than simple deletion of the attribute would.

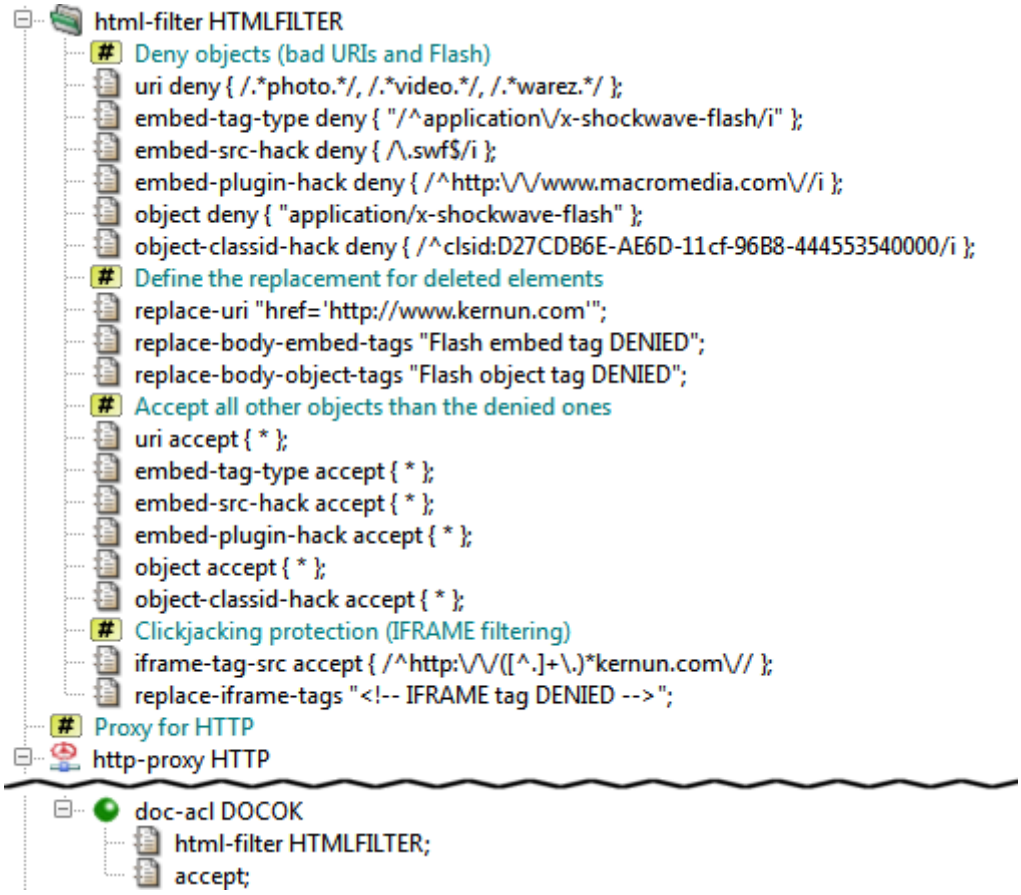

Figure 5.69: HTML filter example

For more information on HTML filtration, see the mod-html-filter(5) manual page.

5.17.3 MIME Processing

In Section 4.2, we created simple mail-handling proxies (`smtp-proxy`, `imap4-proxy` and `pop3-proxy`), which just passed the e-mail and did not check it in any way. Kernun UTM provides tools to decode the e-mail MIME structure and examine each document. The mail may be scanned for viruses (see Section 5.15) and for spam content (see Section 5.16). These scans are added to the proxy by inserting a `use-antivirus` and/or `use-antispam` item into the proxy and setting them to the respective antivirus/antispam section name.

Kernun UTM has also a mail filter, which can be used to repair the MIME structure and headers of the e-mail according to corresponding RFCs, so that other mail servers could process the mail and possibly deliver it to the recipient(s). Errors in e-mails that are to be corrected can be specified in the mail filter. The relevant part of a sample configuration with a mail filter is shown in Figure 5.70; the entire configuration can be found in `/usr/local/kernun/conf/samples/cml/mime-processing.cml`.

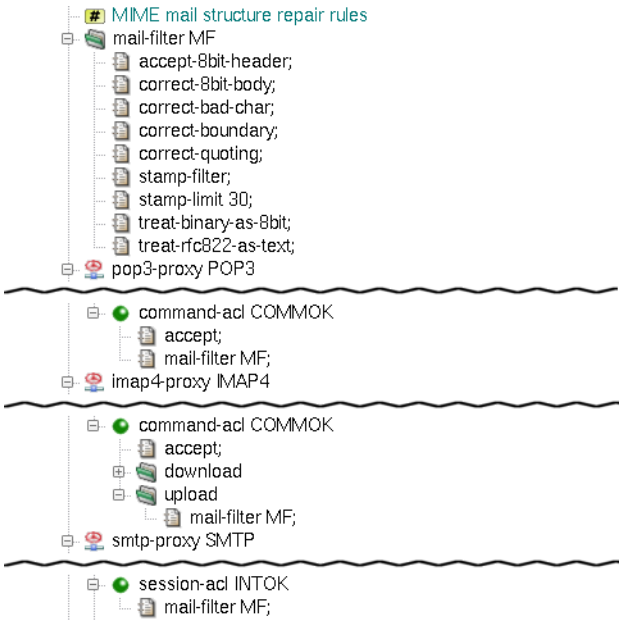

Figure 5.70: Mail filter example in use with IMAP4, POP3 and SMTP proxies

For more information on MIME filtration, see the mod-mail-doc(5) manual page.

5.18 Filtering HTTP Requests by URI

Kernun UTM's HTTP proxy provides several methods that can be used to change requestURIs passed to servers and make decisions based on URIs. A URI can be matched against regular expressions, searched in a local database (blacklist), or evaluated by the Kernun Clear Web DataBase or by an external Web filter. The sample configuration

`/usr/local/kernun/conf/samples/cml/url-filter.cml` contains an HTTP proxy with all URI filtering methods.

5.18.1 URL Matching and Rewriting

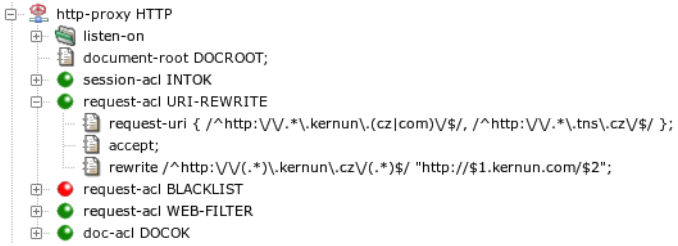

Figure 5.71: Request URI matching and rewriting

An example of the HTTP proxy that matches HTTP requests by the request URI is depicted in Figure 5.71. If a request URI matches one of the regular expressions in the string specified in the `request-uri` item of `request-acl URI-REWRITE`, this ACL will be selected. It accepts the request and applies the `rewrite` item(s). There can be several `rewrite` items. Each of them contains a regular expression and a string. A request URI that matches the regular expression in a `rewrite` clause will be rewritten to the string. Pairs of characters dollar+digit (`$1`, `$2`...) in the string will be replaced by parts of the original URI matched by parenthesized parts of the regular expressions. The `rewrite` item in Figure 5.71 matches any URI containing `kernun.cz` and rewrites `cz` to `com`, leaving the rest of the URI unchanged.

In a `request-acl`, there are two more types of URI-matching conditions. We have already seen `request-uri`, which matches the whole URI. The other conditions are: `request-scheme` (matches the scheme part of the URI) and `request-path` (matches the path part of the URI).

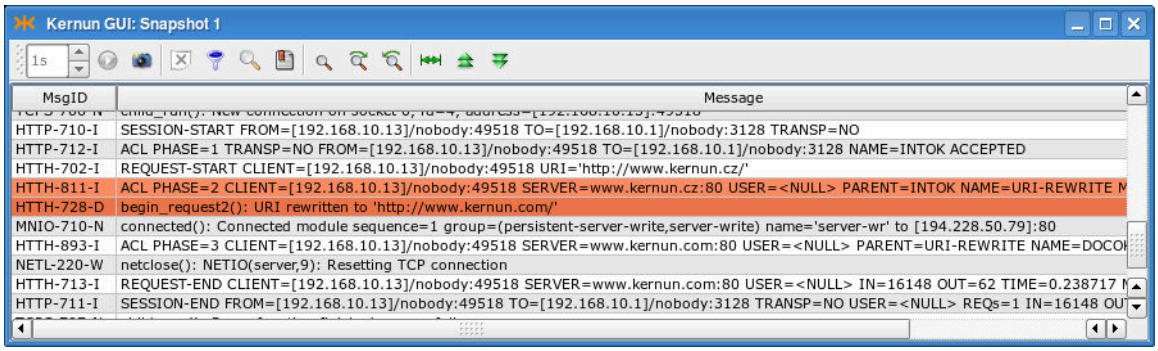

Figure 5.72: Log of URI matching and rewriting

Figure 5.72 shows a log of an HTTP proxy running with the configuration from Figure 5.71 and processing a request with URI `http://www.kernun.cz/`. The two most important messages are marked. The `HTTH-811-I` message informs that `request-acl URI-REWRITE` was selected. The following message confirms that the URI was rewritten to `http://www.kernun.com/`.

5.18.2 Blacklists in HTTP Proxy

A security policy may require to deny access to large groups of Web servers. Adding a very large number of servers to a `request-acl.request-uri` item may slow down ACL processing significantly. A better strategy is to create a *blacklist* of forbidden servers in a special format processed more efficiently by the HTTP proxy. A blacklist must be prepared in a text file in the format described in **mkblacklist**(1). The textual blackist is then transferred to a binary form usable by the HTTP proxy by utilities **resolveblacklist**(1) and **mkblacklist**(1). A binary blacklist file can be converted back to the text format by **printblacklist**(1).

A blacklist contains a list of Web server addresses (domain names or IP addresses) with an optional initial part of a request path. A list of categories is assigned to each address. It is possible to select a `request-acl` based on a matching `blacklist` item with the set of categories assigned to the request URI by the blacklist.

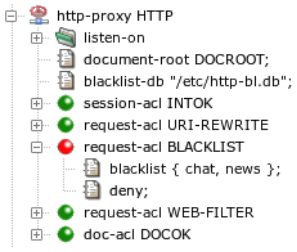

Figure 5.73: A blacklist in the HTTP proxy

An HTTP proxy will use the blacklist database file specified by `blacklist-db` in the `http-proxy` section, see Figure 5.73. If a request URI belongs to the category `chat` or `news`, then `request-uri BLACKLIST` will be selected and the request will be denied. Other requests will be processed according to `request-uri WEB-FILTER`, described in the next section.

Note

If a proxy runs in chroot, the path `blacklist-db` will be interpreted in the context of the chroot directory.

A log of an HTTP proxy configured according to Figure 5.73 is displayed in Figure 5.74. The log messages are related to two requests. The first request to server `www.example.com` does not match categories selected in `request-acl BLACKLIST` and the request is accepted by `request-acl WEB-FILTER`. The second request to `news.example.com` matches a category from `request-acl BLACKLIST`, which rejects the request. Note the ACL names reported in `ACL PHASE=2` messages and the log markers that colorize accepting and rejecting messages.

5.18.3 Kernun Clear Web DataBase

Continuous updating of a blacklist so that it covers as many prospective unwanted Web servers as possible is a challenging task, greatly exceeding the capabilities of a typical company network administrator. Therefore, there are commercially available services that provide regularly updated

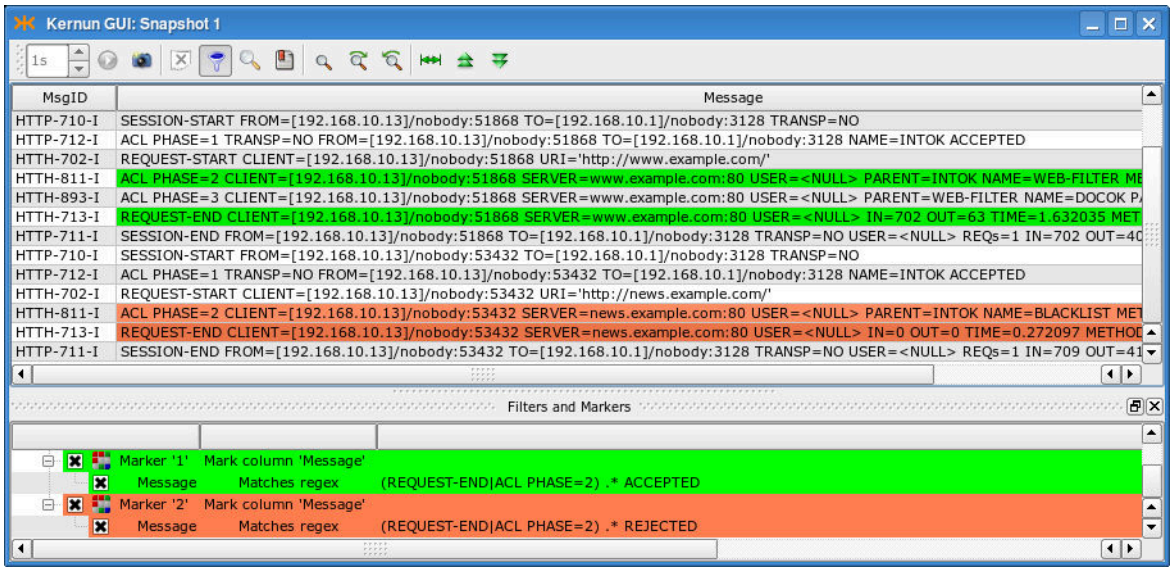

Figure 5.74: A log of blacklist usage in the HTTP proxy

databases of Web servers classified into many categories according to content. Kernun UTM contains one such service—the Kernun Clear Web DataBase.

Configuration of the Kernun Clear Web DataBase consists of two parts. One is used to detect URL categories and select ACLs according to the categories, the other deals with updating of the database of URLs.

Clear Web DataBase in HTTP Proxy

The Clear Web DataBase takes a request URI and assigns to it a set of categories. The categories can be matched by item `request-acl.clear-web-db-match` in the process of `request-acl` selection during an HTTP request processing. Two examples of sections `request-acl` with condition `clear-web-db-match` are shown in Figure 5.75. There are three possible modes of category matching, selectable in each condition item:

- `any` — at least one category of the request URI matches the set of categories in the condition;
- `all` — all categories in the condition are contained in the set of categories which the request URI belongs to;
- `exact` — the request URI belongs to all the categories from the condition and does not belong to any additional category.

If the selected `request-acl` contains a `clear-web-db-match` condition (`CLEAR-WEB` in the sample configuration) and the `deny` item, the proxy will return an error HTML page containing information that the request has been denied by the Clear Web DataBase and the list of matching categories.

As an alternative to denying a request URI matching given categories absolutely, the proxy provides the *bypass* feature, a sort of "soft deny". It is activated by item

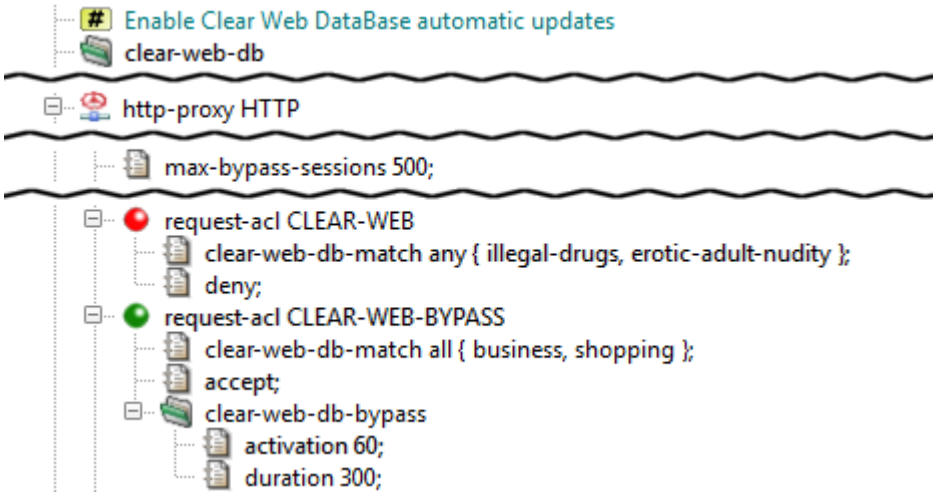

Figure 5.75: Kernun Clear Web DataBase in the HTTP proxy

`request-acl.clear-web-db-bypass` (used in `CLEAR-WEB-BYPASS` in the sample configuration). When a user attempts to access a Web server restricted by the bypass, an error page is returned by the proxy, containing a link, through which the user can obtain limited-time access to the requested server and all Web servers that belong to the same categories. After the bypass expiration time, which can be changed using item `duration`, the error page will be displayed again. The user can then reactivate the bypass. Each bypass activation is logged.

Bypass sessions activated by users are tracked by an internal table managed by the proxy. The maximum number of simultaneously active bypass sessions is controlled by `http-proxy.max-bypass-sessions`. If this number is set to 0, the number of bypass sessions will become unlimited, and the sessions will be tracked by cookies. In this case, an activated bypass is valid for the target server and all servers in the same domain, but not for other servers, even if they belong to the same categories.

Internal Servers in the Clear Web DataBase

Clients often use the HTTP proxy also for access to intranet WWW servers. These internal servers can be located in the internal network, using private IP addresses from ranges defined by RFC 1918 (networks `10.0.0.0/8`, `172.16.0.0/12`, and `192.168.0.0/16`). Or they may be accessible from the Internet, but restricted to internal users.

Such internal servers are not contained in the Clear Web DataBase and would be categorized as `unknown`. In order to distinguish internal servers from public web servers with unknown categories, host names or IP addresses of the internal servers can be listed in the configuration by item `clear-web-db.internal-servers`. Private IP addresses from RFC 1918 are included by specifying `internal-servers private-ip`, so they need not be written explicitly.

If a server from an HTTP request matches some element of the list of internal servers, it will be assigned category `internal-servers`. This category is not used for any public web server included in the standard Clear Web DataBase.

Updating the Clear Web DataBase

The Clear Web DataBase file location in controlled by `clear-web-db.db`, which defaults to `/data/var/clear-web-db` if not set. The database file is copied (or hardlinked, if possible) automatically to any chroot directories by the database update script and when the configuration is applied.

Although it is possible to update the Clear Web DataBase data manually by copying an updated `clear-web.db` file to the database directory, it is recommended to use the provided update script `clear-web-db-update.sh` for this purpose. It downloads the most appropriate full database and/or applies any incremental updates to get the most recent version of the database, minimizing the amount of downloaded data. To use this feature, you need to have valid download credentials. These are configured by `clear-web-db.credentials`. If not set, the serial number from the license file is used as both the username and the password.

The Clear Web DataBase data update script can be executed either manually by running `clear-web-db-update.sh` from the root console (Kernun console in the GUI), or automatically by cron in periodic intervals. Periodic updates are enabled by placing the `clear-web-db` (usually empty) section in the system configuration.

Automatic Categorization of Web Servers

If no record is found for a requested URL in the Clear Web DataBase, the "unknown" result is reported and matched in ACLs. A valid list of categories will be returned for repeated requests to the same URL only after the server becomes a part of the Clear Web DataBase downloaded in some future update. As an option, there is a faster way how to obtain categories for servers not yet known. It is possible to enable the **cwcatd** daemon that performs automatic local categorization of web pages.

If categories for a requested URL are not found in the Clear Web DataBase, the URL is appended to a queue that is processed by daemon **cwcatd**. The daemon reads uncategorized URLs from the queue. For each URL, it tries to download the referenced web page. The downloaded page is passed to an automatic categorizer, which tries to assign categories according to heuristics applied to the page content. If it succeeds, the result is stored in a local database. Future requests to a locally categorized server will get categories assigned according to the local database. If categories of the server appear in the downloaded Clear Web database in its periodic update, the result of the automatic local categorization will not be used any more.

Categories discovered automatically by the **cwcatd** daemon are not as correct and reliable as the contents of the Clear Web DataBase maintained by experienced human operators. But on the other hand, results of the automatic categorization are available almost immediately after the access to a web page with unknown categories.

Automatic categorization of web servers is enabled by adding subsection `local-db` to the configuration section `clear-web-db`. It instructs the **http-proxy** and the **icap-server** to enqueue URLs with unknown categories for processing by **cwcatd** and to use the results of automatic categorization. The **cwcatd** daemon is enabled by adding section `cwcatd`.

Clear Web DataBase category list

The Clear Web DataBase or the local database contain the following set of categories:

Advertisement Advertising agencies, servers providing advertising banners

Alcohol / Tobacco Alcohol, tobacco, breweries

Arts Galleries, theaters, exhibitions

Banking Banks, Internet banking

Brokers Stock exchange information, stocks purchase and sale

Building / Home Construction, architecture, e-shops with construction materials and equipment

Business Corporate websites, offering actual services and products, Web pages referring to companies (parked domains)

Cars / Vehicles Web pages of motorists, motorcyclists, aviators, sailors, producers, vendors and enthusiasts of transportation vehicles

Chats / Blogs / Forums Chats, personal blogs, discussion forums

Communication Information and communication technology, services of Internet providers, telephone operators, SMS gateways, phonebooks

Crime Committing crime, crime protection

Education Education, science, research

Entertainment Entertainment and sports centers

Environment Weather forecasts, live webcams, environment protection, ecology

Erotic / Adult / Nudity Erotic acts, half-naked up to naked bodies, erotic tales

Extreme / Hate / Violence Attacks against individuals or groups of people, deviations (sadism, masochism, etc.), violence on humans or animals

Fashion / Beauty Clothing, body care, make-up

Food / Restaurants Restaurants, recipes

Foundations / Charity / Social Services Help to handicapped, addicted, underprivileged people or people with special needs, families and children

Gambling Gambling for money

Games Online and offline games

Government Government organizations, agencies, ministries, communes

Hacking / Phishing / Fraud Web pages of cracking groups, malicious software, exploitation guides. Does not include anti-hacking and anti-phishing protection

Health / Medicine Electronic pharmacies, hospitals, relaxation, massage, spa

Hobbies Thematic hobby/interests pages

Humor / Cool Humor, jokes, pranks

Illegal Drugs Drugs, narcotics

Instant Messaging Real-time message transfer

Insurance Insurance companies

Internal Servers Internal web server, includes private (RFC 1918) IP addresses and servers declared as internal in the configuration

IT / Hardware / Software Hardware and software retail, service, support

IT Services / Internet IT and Internet products, services, domain names

Job / Career Human resource agencies, job offers

Kids / Toys / Family Advice, instructions, websites for kids and family

Military / Guns Historic information, army-shops

Mobile Phones / Operators Mobile phones and accessories, telephone operators

Money / Financial General financial consultancy, financial servers

Music / Radio / Cinema / TV Web pages of music bands, pages about music and movies, TV guides, cinema programme

News / Magazines Live news, tabloids, journals. Does not include Web pages aggregating news from other sources

Peer-to-peer Peer-to-peer networks for data transfer (software, music, movies)

Personal / Dating / Live Styles Personal Web pages, pages of musicians and bands, gossips

Politics / Law Web pages of political parties, pages about politics and law

Pornography Videos for download, photos, e-shops

Portals / Search Engines Internet search engines, covering either a large number of areas or specific fields. Does not include Web pages searching their own site only

Proxies Servers that provide access to other Web pages to their clients, e.g. in order to hide the client's identity or to circumvent network access restrictions

Real Estate Real estate servers, companies

Regional Web pages of municipalities and regions

Religious / Spirituality Churches, religious themes, esoteric pages

Sale / Auctions Auctions, bazaars, sale and purchase

Sects Sects

Sex Education Professional information about sex

Shopping Stores, advertising sites, e-shops with a shopping cart, which must be operated at the top-level domain

Social Networks Social structures made up of people connected by friendship, family, common interests and other relations

Sports All sports and disciplines, including extreme sports

Streaming / Broadcasting Internet TV and radio

Swimwear / Intimate Lingerie, underwear, swimsuits, e-shops, producers and sellers

Translation Services Online translation, translational services

Travelling / Vacation Timetables, travel agencies, public transportation companies, castles, chateaus

Uploading / Downloading Online data storage, photo albums, file-sharing servers

Warez / Piracy Illegal distribution of copyright-protected content, links to software, serial numbers, movies, password exchange forums

Web Based Mail Free Web-based mail providers

Web Hosting Web sites providing space on their servers to third parties

5.18.4 Using External Web Filter

In addition to using the internal Kernun Clear Web DataBase, it is possible to cooperate with an external Web filtering service. Only one type of such a service is supported—the Proventia Web Filter. It is considered a legacy feature and Kernun Clear Web DataBase is recommended as replacement.

Proventia Web Filter runs separately from Kernun UTM on a machine with a Microsoft Windows, Linux, or FreeBSD (with Linux emulation) operating system. It is configured by its own graphical management console, available for Microsoft Windows. In order to work with Kernun UTM, Proventia Web Filter must be configured as described in section Web Filter of manual page **http-proxy**(8). Especially, ICAP Integration must be enabled. Moreover, if the Web filter rules take user names into account, User Profile Support must be enabled.

The relevant part of HTTP proxy configuration is depicted in Figure 5.76. The parameters of the connection to the Proventia Web Filter server are specified by section `web-filter`. Note the port number 1344, which is used by default for the ICAP protocol by Proventia Web Filter. The sample configuration contains also a `fail-ok` item, which means that all requests are to

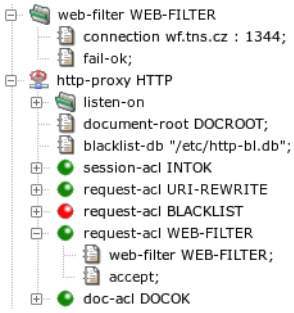

Figure 5.76: An HTTP proxy configured for use of an external Web filter

be accepted if Kernun UTM cannot communication with the Web filter. Without this item, all requests would be rejected until the connection to the Web filter would be restored. Requests that are `accept`-ed by a `request-acl` with a `web-filter` item will be forwarded to the Web filter server. If the Web filter accepts the request, the proxy will continue with the normal request processing[17]. If the Web filter rejects the request, the proxy will terminate the processing of the request and return an error HTML page to the client. An example of a Web site blocked by the Web filter is shown in Figure 5.77.

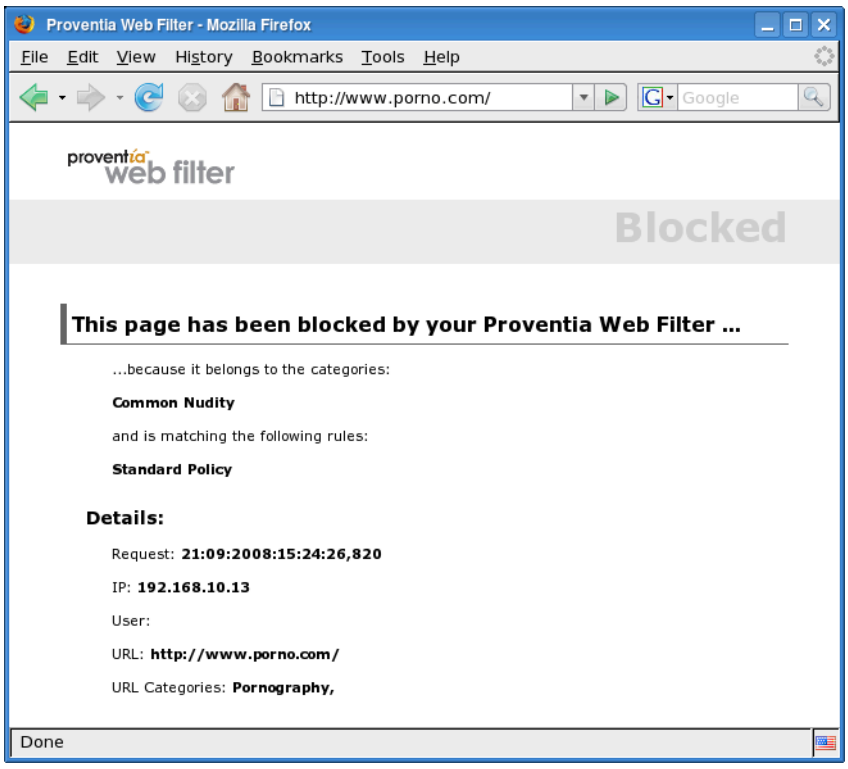

Figure 5.77: A Web server blocked by the Web filter

[17] The request will be sent to the destination server and its response will be returned to the client.

5.19 HTTPS Inspection

Kernun UTM, namely the `http-proxy`, provides means for inspection of the HTTPS traffic. HTTPS inspection is available for both transparent and non-transparent connection mode.

An example of the `http-proxy` configured for HTTPS inspection is available in the `/usr/local/kernun/conf/samples/cml/https-insp.cml` file. The example demonstrates the following scenario:

- Both HTTP and HTTPS requests are handled by the proxy.
- Both transparent and non-transparent mode is provided.
- The server certificate is verified by the proxy.
- There is a list of server exceptions, for which the inspection is not performed.
- There is a list of client exceptions, for which the inspection is not performed.

The configuration described in this section is intended for providing access to an a priori unknown set of HTTPS servers. For example, access of clients in the internal network to HTTPS servers in the Internet. If you need a secure tunnel between two points or HTTPS to a single server, the SSL/TLS configuration from Section 5.13 may be more appropriate.

5.19.1 Certificates

HTTPS uses certificates as a means to verify the server identity. HTTPS inspection (by its nature) breaks into the connection between the client and the server, which would be detected by the client and reported as an "Untrusted connection" warning to the user. To prevent this issue, the proxy generates a new certificate which mimics the original certificate. It is issued by the certification authority (CA) controlled by the Kernun UTM. By importing this CA to the list of trusted CAs of the client browser the above mentioned "Untrusted connection" warning is prevented.

There are two CAs needed for proper work of the HTTPS inspection: GOOD and BAD. The HTTP proxy selects one of them for signing the certificate that is presented to the client as the server certificate.

The GOOD CA is used for servers, whose certificates were successfully verified by the proxy. The clients are expected to import the GOOD certificate as the trusted certification authority. The BAD CA is used for servers, whose certificates cannot be verified by the proxy. The clients MUST NOT import the BAD certificate.

As the result, the web pages, whose server certificate was succsessfully verified by the proxy are presented with certificates signed by the GOOD CA to the client. The client accepts them without complains, because the client trusts the GOOD CA.

By contrast: web pages, whose server certificate cannot be verified by the proxy, trigger the "Untrusted connection" alarm, since they are presented to the client with the certificate signed by the BAD CA, which the client does not trust.

The configuration is shown in figure Figure 5.78. The GOOD CA and the BAD CA are provided to the proxy in the section `fake-cert` as item `fake-cert.auth-ca` (the GOOD

CA), and item `fake-cert.fail-ca` (the BAD CA). Besides the two CA, there is also a private key needed for signing the faked certificates. The key is specified in section `fake-cert.key`.

The proxy uses the list of the trusted CAs for verifying the server certificates. The list is presented in the section `SSL-INSP-SERVER-SSL`. The default list of the trusted CA is provided by Kernun in `/usr/local/kernun/conf/samples/cert/dist_auth_cert.pem`. This list is the root certificate bundle with certificates merged from Internet Explorer and the Mozilla Project. Main Czech and Slovak certification authorities are included as well. If you need to modify the list, copy the file to another location and change the path for the `shared-file SSL-INSP-AUTH` appropriately. The file contains concatenation of certificates in the PEM format, so you can delete existing certificates or append new ones using any text editor.

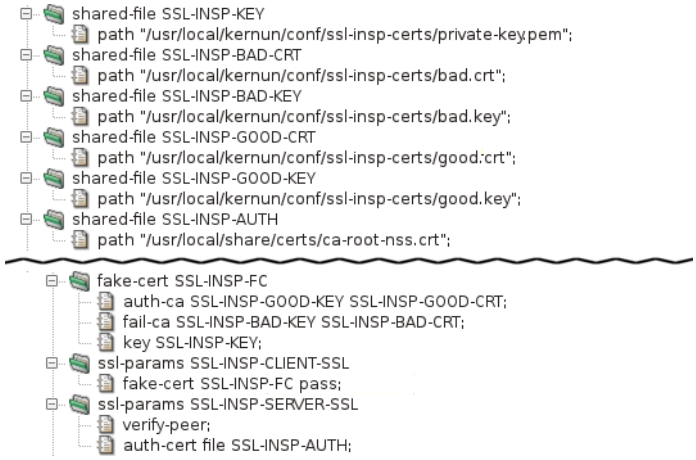

Figure 5.78: Certification authorities for HTTPS inspection

Note

The CA can be created by command **mkcert.sh**, for example:

```
mkdir /usr/local/kernun/conf/ssl-insp-certs
cd /usr/local/kernun/conf/ssl-insp-certs
mkcert.sh -g . -c "Kernun SSL Inspection" -F good.crt -K good.key
mkcert.sh -g . -c "Server verification failed!" -F bad.crt -K bad.key
```

The private key needed can be created by command **openssl**, for example:

```
cd /usr/local/kernun/conf/ssl-insp-certs
openssl genrsa -out private-key.pem 2048 -nodes
```

5.19.2 HTTPS inspection ACL flow

When both transparent and non-transparent HTTPS is used in a single `http-proxy`, several `session-acls` and `request-acls` are used to internally transform the non-transparent traffic to transparent as well as to transform the HTTPS to HTTP by inspecting the SSL, or to apply ac-

cess control without inspecting the SSL. After the transformations, a single set of `request-acls` is used to implement the security policy.

HTTPS traffic can be:

- Accepted without HTTPS inspection. The proxy establishes TCP tunnel between client and server, ending the ACL processing at `request-acl` level.
- Blocked without HTTPS inspection. In transparent mode, TCP connection to the client is reset before connecting to the server. In non-transparent mode, an error message is sent to the client.
- Accepted in HTTPS inspection. The proxy decrypts the SSL/TLS, processes the HTTP communication and re-encrypts it afterwards. The decrypted HTTP communication is processed the same way as (both transparent and non-transparent) HTTP. When different behaviour is desired, `session-acl` item `connect-session-acl` can be used to detect the `session-acl` that accepted the HTTPS connection.
- Denied in HTTPS inspection. Proxy decrypts and re-encrypts the HTTP communication. Proxy can either send the client properly encrypted error document or send the requested server response signed by BAD CA as described by Section 5.19.1.

Figure 5.79 provides the schema of ACL flow during the processing of HTTP and HTTPS traffic in both transparent and non-transparent mode.

5.19.3 Transparent mode

The `http-proxy` by default expects the client to send an HTTP request, it can be told to expect TLS by specifying either `sni-insp` or `client-ssl` in the `session-acl`.

In transparent HTTPS, server hostname is specified in HTTP protocol which is encrypted. However, it is usually also specified in SNI in the first TLS message which is not encrypted yet. See Section 5.19.5 for more details. In order to obtain the server hostname from TLS layer before HTTPS inspection, item `sni-insp` is used. If an exception from HTTPS inspection takes effect, the proxy establishes TCP tunnel to the server.

If the traffic is to be HTTPS inspected, item `capture-connect` is used to make the proxy virtually close the current session and reexute it. Afterwards, HTTPS inspection is turned on by referencing the `client-ssl` section which defines item `fake-cert`. Subsequent `request-acl` processes the request as if it was plain HTTP.

The configuration of both transparent and non-transparent modes is shown in Figure 5.80. The `session-acl S-HTTPS-TRANSP` uses the item `sni-insp` to obtain server name as well as to validate the first SSL/TLS message. Exceptions from SSL inspection are defined in `request-acls R-HTTPS-NO-INSP-CLIENTS` and `R-HTTPS-NO-INSP-SERVERS`. Item `request-method` with value `CONNECT` can be used here because of item `sni-insp` which simulates the `CONNECT` method. Item `capture-connect` in `request-acl R-HTTPS-INSP` makes the proxy virtually close the current session and reexute it, which results in `session-acl S-HTTPS-INSP` getting matched because of item `captured-connect`. The `session-acl S-HTTPS-INSP` uses the `client-ssl SSL-INSP-CLIENT-SSL` and

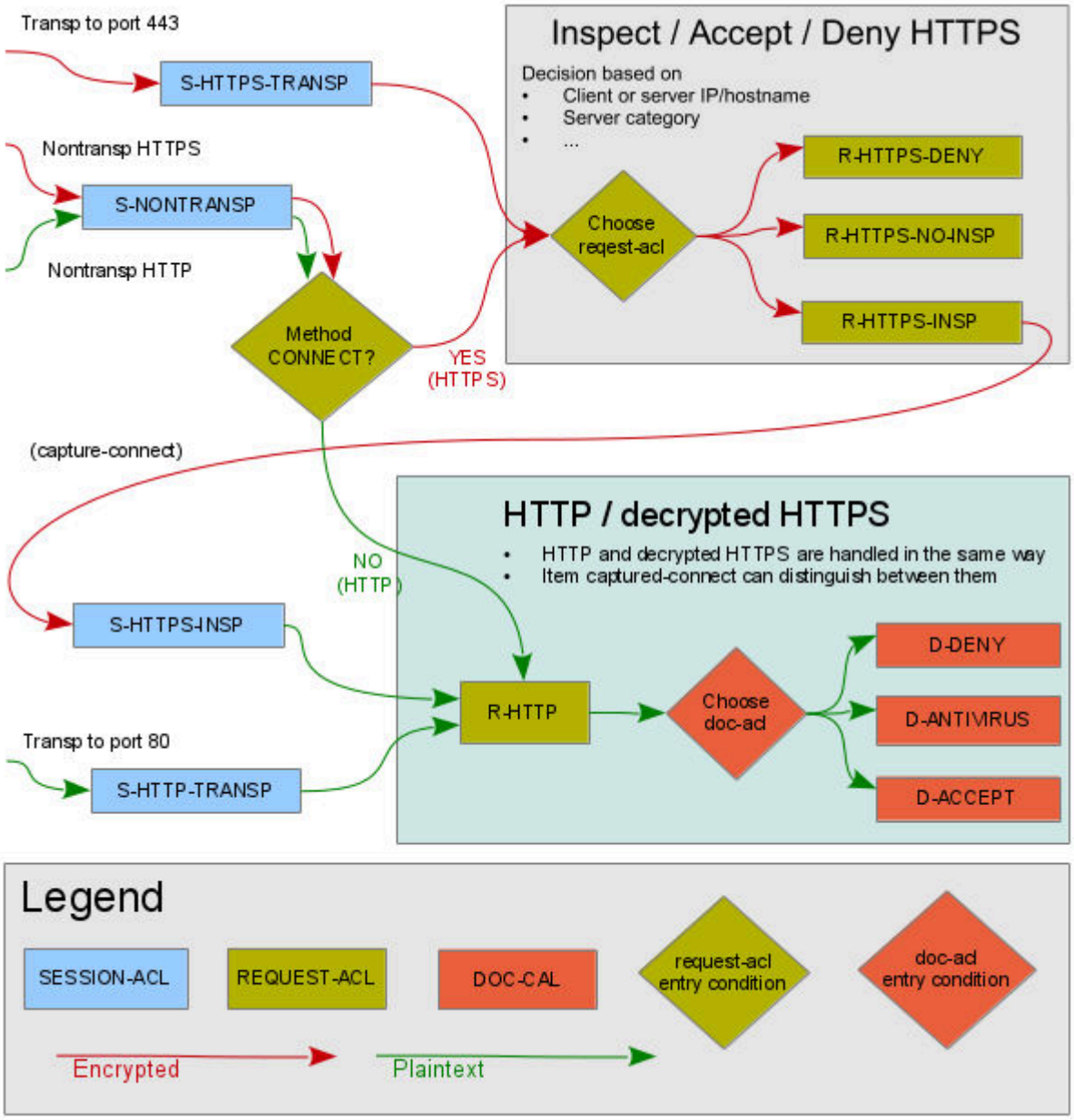

Figure 5.79: HTTPS inspection ACL flow

`server-ssl SSL-INSP-SERVER-SSL`. See Section 5.19.1 for details on these `ssl-params` sections.

5.19.4 Non-transparent mode

In non-transparent mode, HTTPS is recognized by HTTP request method CONNECT. Server hostname is specified in the enveloping HTTP so SNI inspection is not needed. Special item `capture-connect` is used for handling CONNECT requests. Afterwards, it is handled by the means described above in Section 5.19.3.

5.19.5 SNI inspection in HTTPS

This section describes Server Name Indication (SNI) inspection that can be used to obtain server name from TLS handshake for usage in request ACLs. It is intended to be used in transparent HTTPS, because in non-transparent HTTPS, server hostname is specified in the enveloping HTTP.

When a client connects to an HTTPS server, the first message (SSL/TLS ClientHello) usually contains server hostname (SNI) that the server uses for detecting service requested by the client, because multiple hostnames can be hosted on a single IP address. SNI was introduced in TLS 1.0 and is optional, it is missing for example when user accesses an HTTPS web server using IP address instead of a hostname.

When SNI inspection is configured in session ACL, the first message from client is parsed. When it does contain SNI, server hostname and URI is changed from an IP address to a hostname for usage in request ACL. Afterwards, as described at Section 5.19.2, HTTPS inspection can be performed, the connection can be accepted without HTTPS inspection or the connection can be denied.

The result of SNI inspection can be used in request ACL item `sni-result` which has values:

`specified` The request was a TLS 1.0 or higher ClientHello that contained SNI. Server hostname and URI for the subsequent request ACL was changed from an IP address to the hostname from SNI.

`unspecified` The request was a TLS 1.0 or higher ClientHello that did not contain SNI. Server hostname and URI remained unchanged for request ACL matching.

`ssl3` The request was a SSLv3 ClientHello that did not contain SNI because it is not in the SSLv3. Server hostname and URI remained unchanged for request ACL matching.

`skype` The request was traffic specific for Skype. Server hostname and URI remained unchanged for request ACL matching. The request was either an invalid empty TLS 1.0 handshake message or TLS 1.0 ClientHello in SSLv2 record layer. Skype uses these requests for registration to the Skype network and for authentication. For the communication itself, Skype uses standard HTTPS so both `skype` and `specified` must be permitted in order for Skype to work. Skype can handle HTTPS inspection as well.

`unknown-protocol` The request was an unknown protocol or malformed request so the SNI was not inspected. Server hostname and URI remained unchanged for request ACL matching.

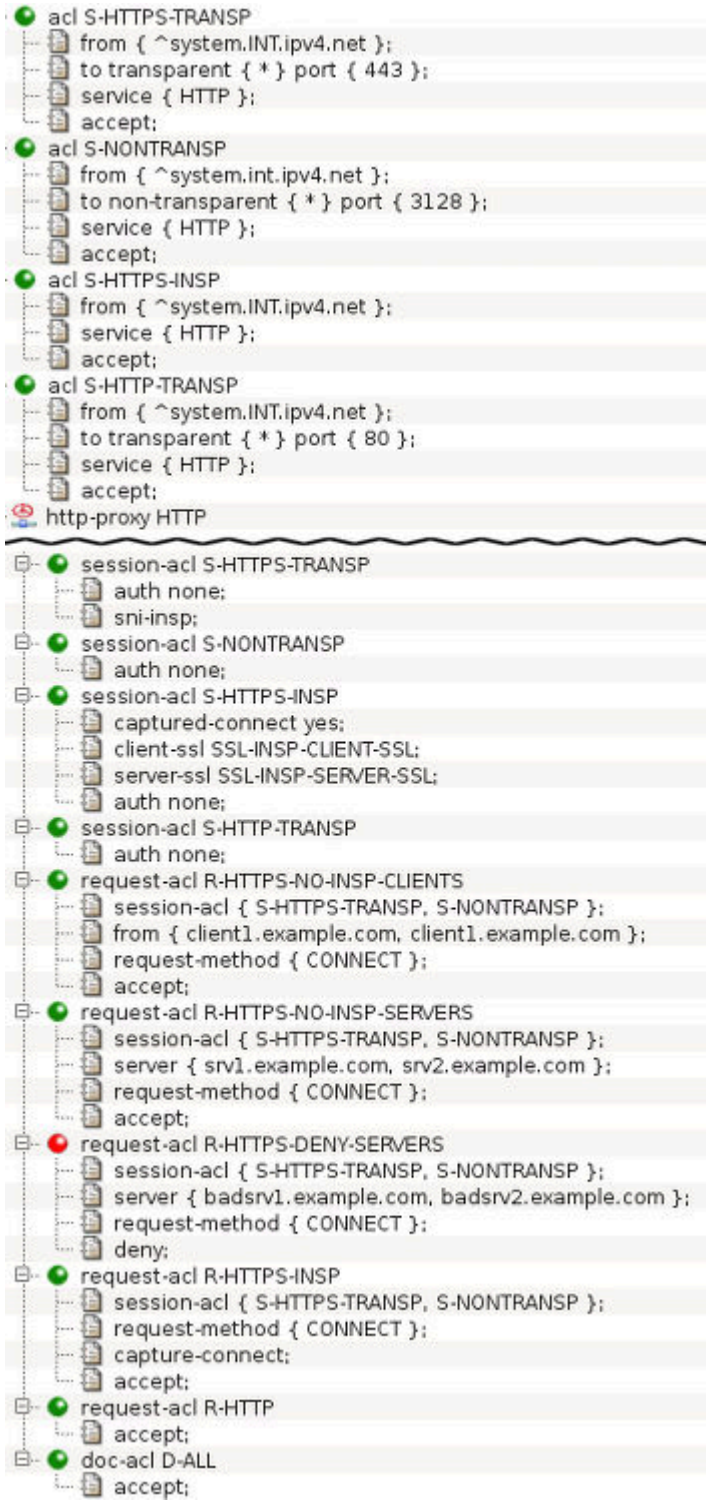

Figure 5.80: HTTPS inspection configuration

5.19.6 TLS termination

The `http-proxy` can be used to provide access to an internal web server for clients in the Internet. When HTTPS is used, the proxy responds to the client with a TLS server certificate immediately after receiving the first message from the client, the ClientHello. If the proxy provides access to multiple internal web servers, it is necessary to select the correct server certificate for sending. That can be achieved by inspecting the SNI received in the ClientHello.

An example of reverse HTTPS proxy configuration is shown in Figure 5.81 and Figure 5.82. The full configuration is located in `/usr/local/kernun/conf/samples/cml/https-termination.cml`. It describes an `http-proxy` that provides access to two web servers, server A and server B, located in internal network for clients in Internet. The clients are forced to use HTTPS when communicating, the proxy uses HTTP with server A and HTTPS with server B.

The proxy listens on the external network interface for connections from clients in the Internet. When a client connects, `session-acl HTTPS-REVERSE-SNI-INSP` performs SNI inspection. After the SNI inspection, the following `request-acls` have its value available in item `server`. The proxy chooses a `request-acl` from `CAPTURE-CONNECT-CRT-A` or `CAPTURE-CONNECT-CRT-B` which both perform `capture-connect`.

During the `capture-connect`, the proxy virtually closes the session and reexecutes it, entering `session-acl` phase once again. One of `session-acls HTTPS-REVERSE-OK-CRT-A` and `HTTPS-REVERSE-OK-CRT-B` is selected based on the `request-acl` that performed the capture. The selected `session-acl` also responds to the client's ClientHello with the specified server certificate and finishes the TLS handshake.

Now the proxy has TLS layer terminated and it is processing plain HTTP. It selects either `request-acl A-PERMIT` or `B-PERMIT` based on the address specified in HTTP request line. The selected `request-acl` relays the HTTP request to one of the servers located in the internal network. The server can be either HTTP or HTTPS. In the example server A is HTTP while server B is HTTPS.

5.20 Adaptive Firewall

Kernun UTM provides module Adaptive Firewall that detects and blocks suspicious traffic.

According to the configuration, it can either detect and log suspicious network traffic (in IDS mode), or also block it (in IPS mode). For detailed description of the IDS/IPS system, see adaptive-firewall(7) and ips(7).

An example of Adaptive Firewall configuration is shown in Figure 5.83. The complete sample configuration is available in the `/usr/local/kernun/conf/samples/cml/ids.cml` file.

Adaptive Firewall consists of two parts, `ids` and `ips`. IDS is composed of several possible detectors, `agent`, `honeypot` and `watchdog`. Both IDS and IPS have their own databases that contain detected IP addresses (in case of IDS) or blocked IP addresses (in case of IPS).

IDS `agent` is an application that performs advanced inspection of network traffic by using complex rules downloaded from a central server to monitor traffic on interfaces specified by item `iface`. The rules are configured in section `system.adaptive-firewall.ids.agent.rules` while

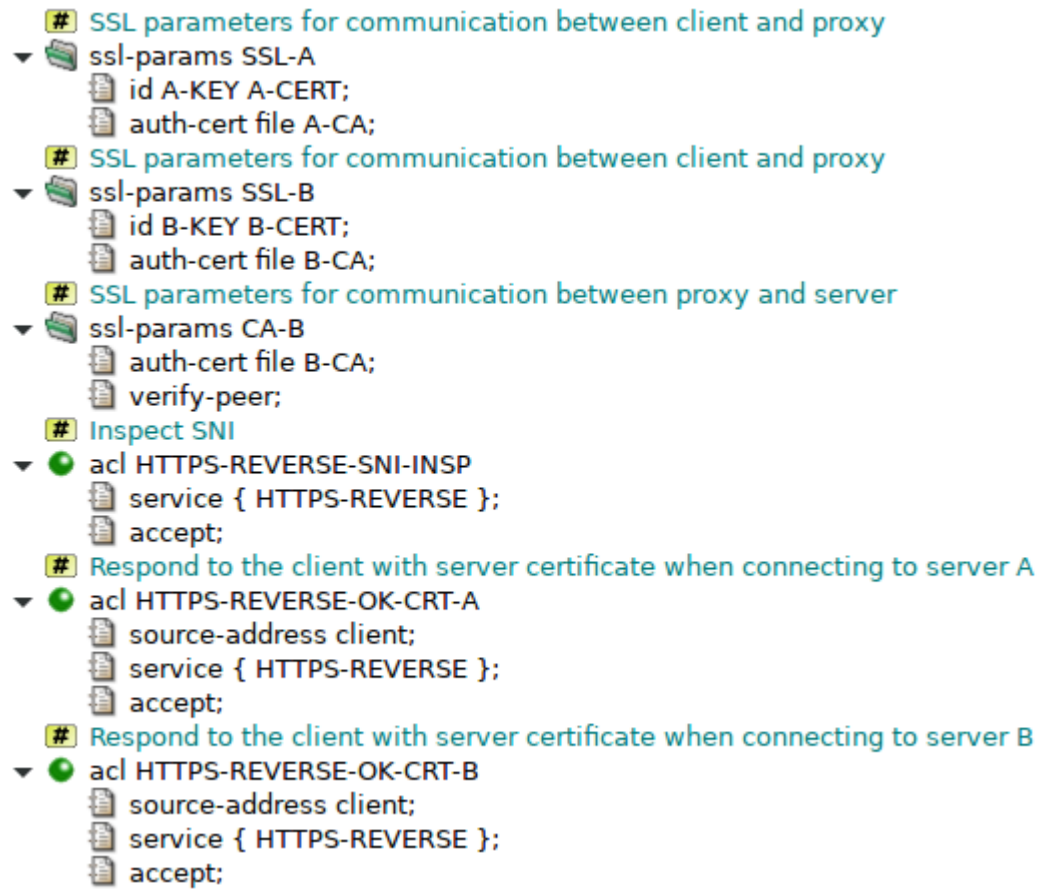

Figure 5.81: HTTPS termination global configuration

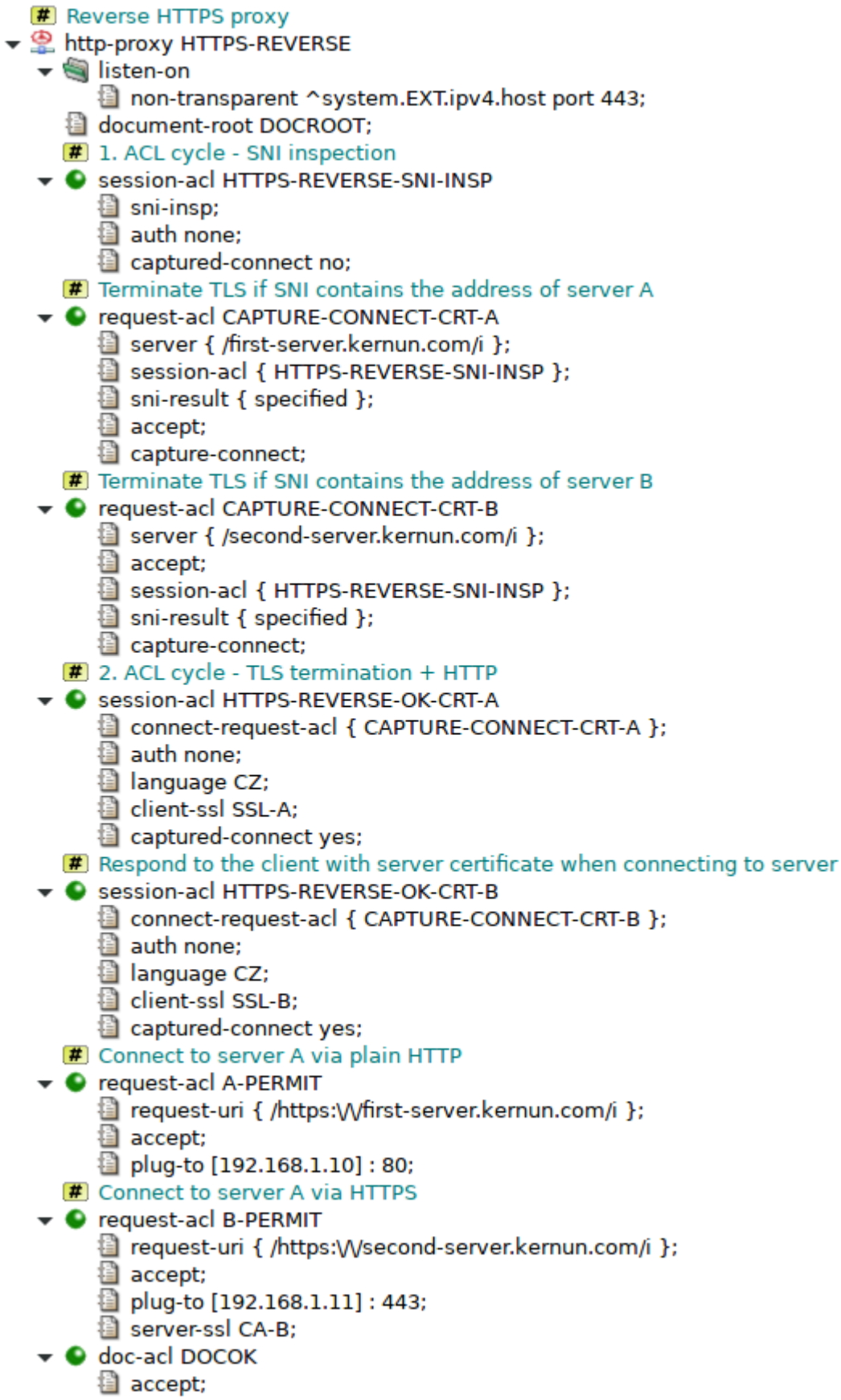

Figure 5.82: HTTPS termination proxy configuration

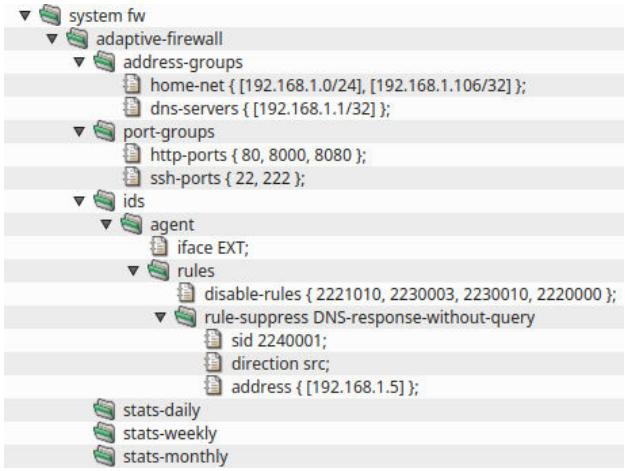

Figure 5.83: Adaptive Firewall

the rules download is configured in section `system.update`. IDS agent uses configuration file `samples/shared/ids-agent.yaml` by default. It is possible to provide a custom configuration file by specifying a shared-file `agent.engine-cfg-file`. Note that the file is processed by Kernun to propagate CML configuration, namely it adds logging, specified interfaces and path to the downloaded rules.

`Honeypot` is a detector that listens on given IP address that is not used for any other purpose. A client that tries to connect to this IP address is assumed to be an attacker and is reported to the IPS part of Adaptive Firewall.

`Watchdog` is a detector that monitors given files for occurences of given string patterns. It can be used for example to detect attackers that are trying to brute-force an SSH autentication. Configuration of this scenario can be seen in `samples/include/sshd-watchdog.cml`

IPS is enabled by the presence of section `adaptive-firewall.ips`. When enabled, the IP addresses reported by various IDS detectors are blocked by packet filter. Every minute, IPS decides which addresses to add from the IDS database to the IPS database and which addresses to remove from the IPS database because they were not seen for a long enough time, which can be controlled by item `record-lifetime`.

5.20.1 IDS agent variables

In order to increase the success rate of rules, it is possible to tell the IDS agent more about the network it is operating in. Sections `agent.address-groups` and `agent.port-groups` are designed for this purpose. Most rules refer to variables that can be defined here.

5.20.2 Rules update

Traffic analysis is based on rules that describe suspicious traffic. It is desirable to update the rules regularly. An automatic rule download system is available in Kernun UTM. This option can be configured with the `rules-download` section, which describes the rules download policy. By default, it is enabled and it uses Kernun rules. Use item `schedule` to define the update schedule.

Item `source` defines the source of the rules. More items can be specified in order to finetune the rules download.

If you are using Kernun rules, it is highly recommended to enable `feedback-upload` to send the feedback containing matched rules back to Kernun server. The matched rules are used to improve the Kernun rules database in order to achieve high success rate of the IPS module.

5.20.3 Rules modification

Even though the rules provided by Kernun are thoroughly tested to eliminate false positive matches, they can still happen. Therefore, the administrator can modify the downloaded rules by items in the `rules` section. It is also possible add custom rules by specifying items `add-rule` or `include-rules`.

A rule that is distributed as disabled can be enabled by item `enable-rules`. A disabled rule is commented out in the downloaded rule file so the IPS engine would otherwise ignore it.

When disabling a rule, the administrator has more options. A rule can be disabled unconditionally by item `disable-rule` or only for certain IP addresses by item `rule-suppress`. It is also possible to disable all rules for certain IP addresses by item `global-suppress`.

In IPS mode, it is sometimes desired to change the rule action from `alert` to `drop` or `reject`. This can be done by items `change-rules-to-drop` and `change-rules-to-reject`.

Items `rule-rate-filter` and `global-rate-filter` can be used to change the rule action after the rule matched a certain number of times within a specified time frame. Similarly, items `rule-threshold` and `global-threshold` alter the rule so it is applied only after it matches a certain number of times within a specified time frame.

When the above methods are not sufficient, it is also possible to modify a certain rule by providing a regular expression and a replacement string in item `modify-rules`.

5.21 Traffic Shaping

Kernun UTM can control the amount of bandwidth consumed by various types of network communication. This feature is known as *traffic shaping*. Traffic shaping is based on a division of the available network bandwidth into queues of various capacity. The traffic shaping rules then assign outgoing network packets to queues. A queue controls how fast a packet will be sent out. Traffic shaping in Kernun UTM is implemented using the altq(4) framework.

> **Important**
>
> Traffic shaping works only for packets sent by Kernun UTM to a network, internal or external. It is not possible to limit the rate of reception of incoming packets. It is therefore impossible to limit the download speed from the external to the internal network by queuing received packets on the external interface; packets sent by the internal interface must be queued instead.

We will describe a simple traffic shaping configuration aimed at limiting the bandwidth usage by SSH communication of clients in the internal network with servers in the external network, and

limiting the download speed of selected HTTP communication. We assume that the maximum speed of both the internal and external network interfaces is 100 Mbits/s.

Let us specify our sample objectives more precisely: 1. For any SSH connection, i.e., a connection to a server in the external network on port 22, we want to limit the data flow to 10 Mbits/s in each direction. 2. Clients from a subset of the internal network accessing HTTP servers in the external network are to have the download speed limited to 10 Mbit/s for selected request URIs at certain time intervals. The 10 Mbit/s limit will be shared by the SSH and HTTP communication.

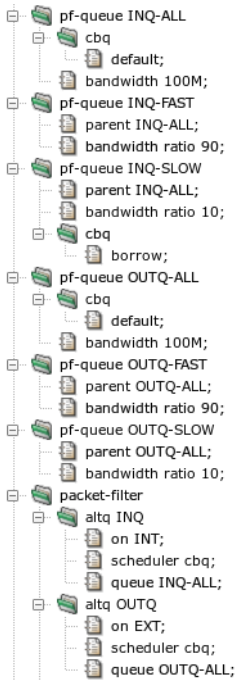

Figure 5.84: Configuration of traffic shaping queues

The first step in traffic shaping configuration is to define the set of queues. This is done by adding sections `system.pf-queue` and `system.packet-filter.altq`, see Figure 5.84. All our queues use the *cbq (Class Based Queueing)* scheduler, i.e., the bandwidth of a network interface is divided into a hierarchically organized tree of queues. We define a parent queue on the internal network interface by section `pf-queue INQ-ALL` and subsection `altq` of section `packet-filter`. The `pf-queue` section selects the cbq scheduler and specifies (using the `default` parameter) that the queue will get all packets that are not explicitly assigned to another queue. The queue can use all 100 Mbits/s available for the interface. Subsection `altq` specifies that the parent queue, and also its child queues, will handle packets sent to the internal network via the `INT` interface. The bandwidth is split between two child queues defined by `pf-queue INQ-FAST` (90 Mbits/s) and `pf-queue INQ-SLOW` (10 Mbits/s). Moreover, queue `INQ-SLOW` can `borrow` unused bandwidth from other queues. Similarly, there are three queues `OUTQ-ALL`, `OUTQ-FAST`, and `OUTQ-SLOW`[18] for handling packets sent to the external network via the `EXT` interface.

[18] Unlike `INQ-SLOW`, this queue cannot borrow bandwidth.

The `/usr/local/kernun/conf/samples/cml/altq.cml` file contains the complete sample configuration. For detailed description of ALTQ parameters, refer to packet-filter(5) and pf.conf(5).

Traffic shaping can be done at two levels, using either packet filter, or queue selection in proxy ACLs. We will use each of the levels for one of our sample objectives.

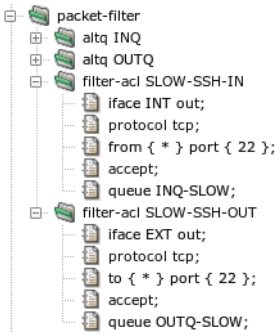

Figure 5.85: Traffic shaping by packet filter ACLs

The SSH communication speed can be limited using packet filter ACLs, as depicted in Figure 5.85. Packets in the direction from the server to the client are matched by `filter-acl SLOW-SSH-IN` as they are sent by `tcp-proxy SSH` to the internal network. This `filter-acl` matches packets sent (direction `out`) on interface `INT`, from any server IP address and from the server port 22. The packets are accepted and queued via `queue INQ-SLOW`. Similarly, `filter-acl SLOW-SSH-OUT` matches packets in the direction from the client to the server as they are sent by `tcp-proxy SSH` to the external network. It matches packets sent (direction `out`) on interface `EXT`, to any server IP address and to the server port 22. The packets are accepted and queued via `queue OUTQ-SLOW`.

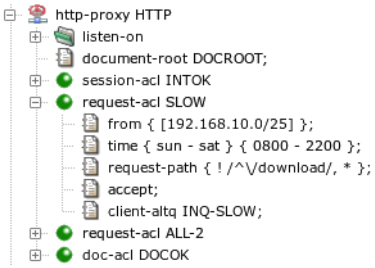

Figure 5.86: Traffic shaping by proxy ACLs

The HTTP communication speed cannot be limited by the packet filter, because the desired conditions for queue selection cannot be expressed in a `filter-acl`. Hence, we utilize the possibility to specify a queue in an ACL of a proxy. In Figure 5.86, `request-acl SLOW` matches response data returned by Web servers for requests issued by clients from the subnet `192.168.10.0/25` each day between 8 and 22 hours. Requests with URI path beginning with the string `/download` are exempted from traffic shaping. The matching response data are queued to `client-altq INQ-SLOW` as they are sent to the client by the proxy. If we wanted to limit

speed of request data sent by the client, we would use `server-altq OUTQ-SLOW` to queue data when they are sent to the server by the proxy.

There are several ways how to check that traffic shaping is working correctly. The command **pfctl -s queue -v** executed on the shell command line displays the numbers of packets and bytes passed via individual queues. The proxy online monitoring, described in Section 5.6.3, reports the current data transmission rates of all active sessions. The amount of transferred data and the session duration are reported in a `SESSION-END` log message for each terminated session.

5.22 Virtual Private Networks — OpenVPN

Kernun UTM supports creation of virtual private networks using OpenVPN. There can be any number of the virtual networks configured for a single Kernun UTM system. Each instance is described in its own `openvpn` section of the configuration. When applied to the system, each instance is displayed in the GKAT as a running application.

Each OpenVPN instance uses an interface, TUN or TAP. The interface, referenced from the `openvpn` section by item `openvpn.interface`, is dedicated to the particular openvpn instance. When applied to the system, the interface is displayed separately from the openvpn application, among other interfaces in the system. Data that comes from the virtual network and is sent to the virtual network appears at this interface (or is sent to it). An example of GKAT window with OpenVPN-related components is depicted in Figure 5.87. There is an OpenVPN instance `OPENVPN-RAS` and its related interface `OPENVPN-RAS-IF`, which serves as the Remote Access Server (RAS). Another OpenVPN instance is `OVPN-BRANCH`, which provides a virtual link to *branch*.

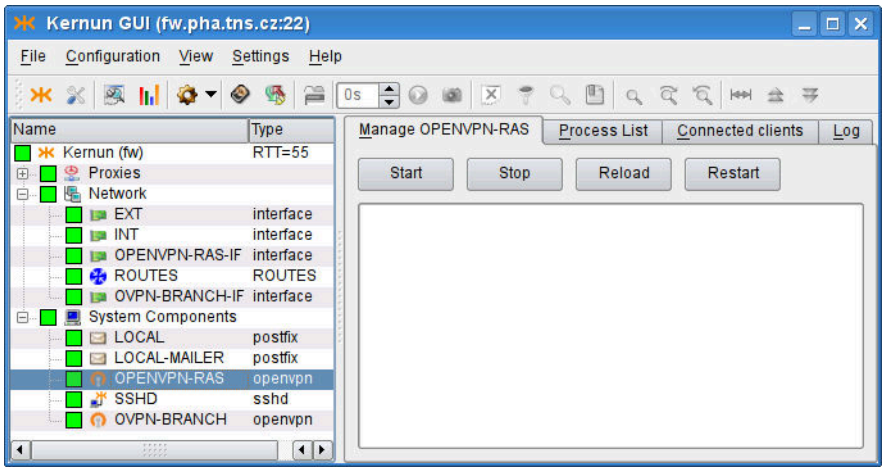

Figure 5.87: OpenVPN components in a GKAT window

Examples of the OpenVPN configuration described here are available in the sample configuration file `/usr/local/kernun/conf/samples/cml/openvpn.cml`.

5.22.1 Remote Access Server

Remote Access Server is an OpenVPN server that allows creation of Virtual Private Network between clients outside the company (Road Warriors) and the company server. These clients, if properly configured, can access internal resources (servers) that are not accessible via the external interface, as if the clients were located inside the company. In this chapter we will configure OpenVPN Remote Access Server to create the virtual network using a wizard, which can be started using Insert | Configuration wizard | OpenVPN Remote Access Server. Ways of granting road warriors access to the internal resources are outlined in Section 5.22.3.

OpenVPN creates a virtual device, and all the VPN communication goes through this device and is encrypted/decrypted. After choosing the section names, we need to decide whether to use the `tap`, or the `tun` interface. TAP works with Ethernet frames and is used to create a network bridge, while TUN works with IP packets and is used with routing. We opt for `tun` and choose an unused device number, 1 in our case. Then we need to define the virtual network, in which the server and clients will communicate. We choose `[10.8.0.1/24]`, thus assigning the server (us) IP address 10.8.0.1. It is very important to choose a network address that does not collide with any other network, from which the client would connect. Finally, we specify the virtual IP address that will be assigned to the other point of the tunnel; we choose `10.8.0.2`. The result is shown in Figure 5.88.

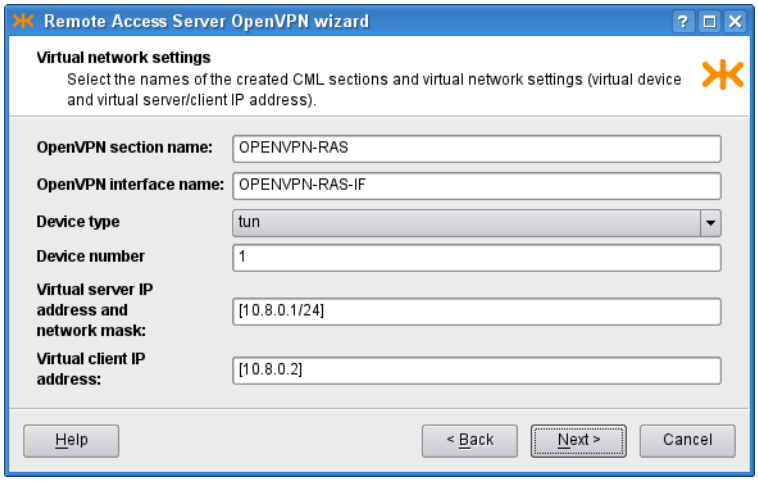

Figure 5.88: RAS: Virtual network settings

On the second page we choose the IP address to bind the openvpn server to. We choose the option `any`, which listens on all the interfaces in the system. The other option, `addr`, allows the user to specify a single address for binding. We leave the port option set to the default value, `1194`. OpenVPN is typically used with the `UDP` protocol. The result is shown in Figure 5.89.

OpenVPN is used to create secure connection, and therefore needs to authenticate the connected clients. We need to define a certificate of the used certification authority, a server certificate and a key signed by this authority. As we have these certificates in separate files in the `PEM` format, we check the second radio button on the third page of the wizard and create new shared files by clicking the  icon and selecting the Create new shared-file option. A dialog depicted

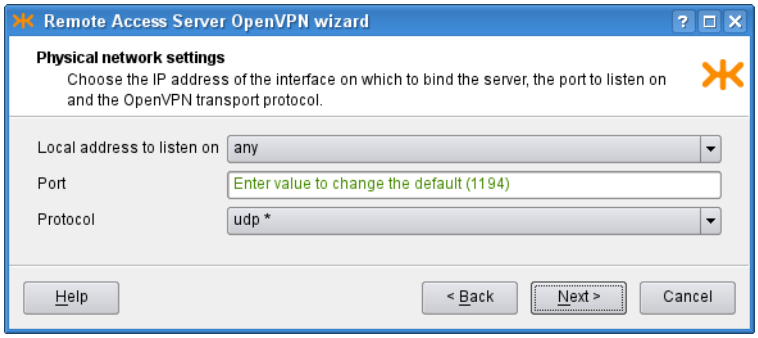

Figure 5.89: RAS: Physical network settings

in Figure 5.90 opens. There we set the `shared-file` name and upload a local certificate file to Kernun UTM. We do not have any certificate revocation list (a list of compromised client certificates) and we leave the Diffie Hellman parameters unchanged (the parameters are already distributed with Kernun UTM and this parameter is not a secret). Finally, the third page should look like Figure 5.91.

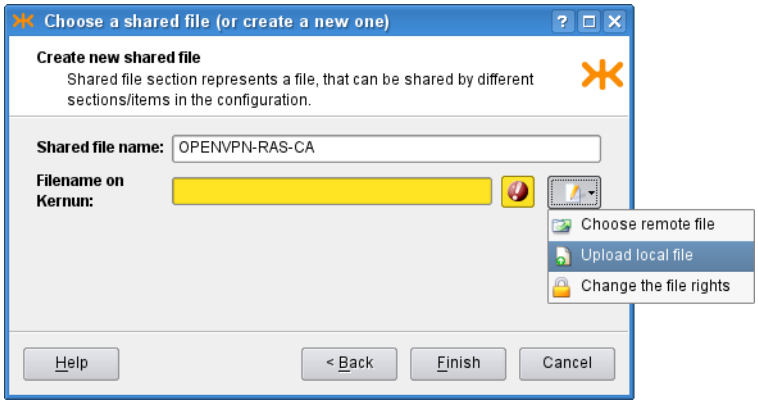

Figure 5.90: Shared file creation dialog

On the Client settings page we can control the interaction between the clients and the server, and among clients. We allow clients to connect among each other and we specify that we accept only connections from clients configured on the clients. In the bottom part of the window, we configure two clients, `client1` and `client2`, and set the IP addresses to be pushed to them. If we do not specify the IP addresses, they will be assigned automatically. On the first page, we chose the `tun` interface, which creates a `/30` network for each client. In each of these networks, the 0. address is the network address, the 3. address is the broadcast address and the middle two remain for the server and client. Therefore, we need to choose one of these middle two IP addresses for the clients, for example `[10.8.0.10]` and `[10.8.0.14]`. The `tap` interface does not create such networks, so if we chose it, we would be able to choose the IP addresses freely from the specified [10.8.0.0/24] network (except the network, broadcast and the server ones). In the middle part of the page there are several IP addresses that can be pushed to each connected client; we choose to push only the default gateway. The resulting page is depicted in Figure 5.92.

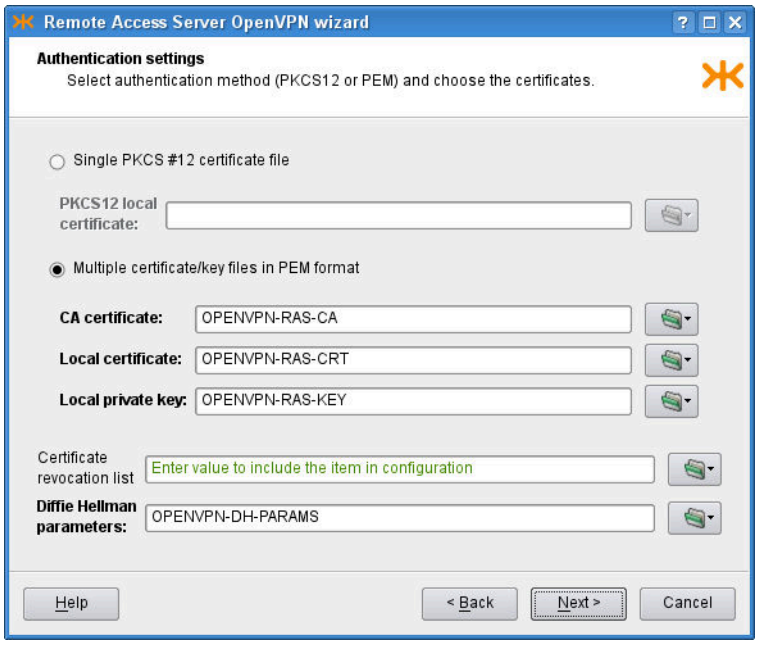

Figure 5.91: RAS: Authentication settings

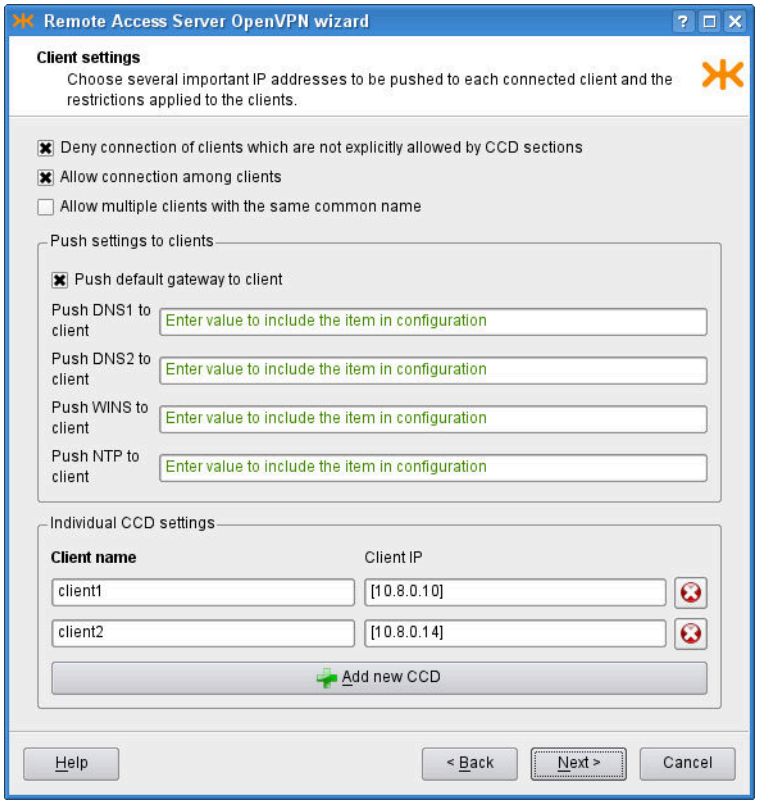

Figure 5.92: RAS: Client settings

Finally, on the Miscellaneous page, we can modify the predefined timeouts, encryption algorithm, compression and log level. In our example, however, we leave the default values of these parameters and proceed to the recapitulation, where we can check the selected choices and either finish the wizard, or return back to previous pages and modify them. The resulting OpenVPN RAS along with the OpenVPN Net2Net configuration can be found in the sample file `openvpn.cml` in the `/usr/local/kernun/conf/samples/cml` directory. The resulting configuration is shown in Figure 5.93

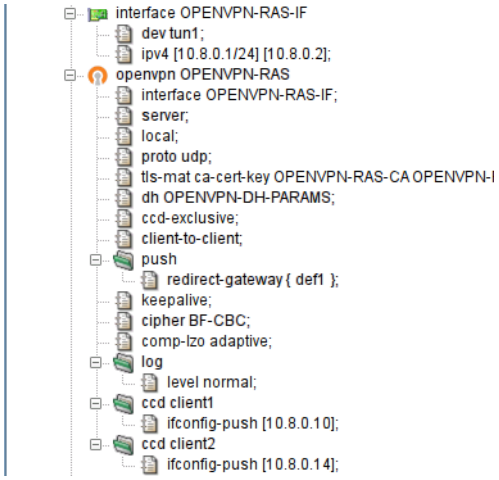

Figure 5.93: RAS OpenVPN wizard: The resulting configuration

RAS Client Configuration

In order to create a configuration for the client to make it able to connect to the created RAS server, it is first necessary to create the client certificate and key with the certification authority that is used for the OpenVPN, and get the certificate of the authority and the two created files to the client computer. Then we create the configuration file of the openvpn program and start the openvpn daemon with this configuration file on the client. An example of the configuration file:

```
# we are a client
client

# name of the device on the client computer, needs
# to be the same type as on the server (tun/tap)
dev tun1

# communicaton protocol
proto udp

# remote access server hostname (or IP address) and port
remote kernun.kernun.com 1194
```

```
# certificate of the Certification Authority (the same
# as on the server
ca /etc/openvpn/keys/ca.crt
# client certificate file and key signed by the CA
cert /etc/openvpn/keys/lj.crt
key /etc/openvpn/keys/lj.key

# additional options
resolv-retry infinite
nobind
comp-lzo adaptive
```

5.22.2 Network to Network

OpenVPN can be used to create a secure point-to-point connection either between two Kernun UTM systems, or between a Kernun UTM system and another system. A virtual point-to-point tunnel interface `tun` is created in the system.

This chapter covers the procedure of configuring the point-to-point connection using the Openvpn Network to Network wizard (Insert | Configuration wizard | Openvpn Network to Network wizard).

Suppose we want to establish an OpenVPN connection between systems *fw* and *branch*. Let us show the configuration for *fw*; the other side is to be configured complementarily.

We start with describing the parameters of the virtual network to be created (see Figure 5.94). We choose a name for the `openvpn` section (`OVPN-BRANCH`) and for the respective `interface` (`OVPN-BRANCH-IF`), the ID of the `tun` interface (`5`), and the IP addresses to be assigned to the virtual ends (`[192.168.155.1/32]` and `[192.168.155.2]`). Note that the local end is specified with the /32 mask.

Important

The IP addresses of the virtual endpoints can be chosen almost arbitrarily. However, the address should be chosen so that it does not collide with any of the used/accessible networks.

Important

The TUN interface should be used in the point-to-point mode. In this mode, the mask /32 should be used for the local peer virtual address, which denotes that no subnet should be routed into this interface by default. If a different mask was specified, an implicit route for the given network would be created.

On the following page of the wizard (see Figure 5.95) we set the physical network parameters: the protocol to be used (`UDP`), the local address and port the OpenVPN server would listen on (the external IP address `^system.EXT.ipv4.host` and port `2194`). We also set the remote host to be `branch.tns.cz`, port `1194`. Finally, we add the network `[192.168.25.0/24]` to be routed through this tunnel[19].

[19] Given that this is the network that is used in *branch*.

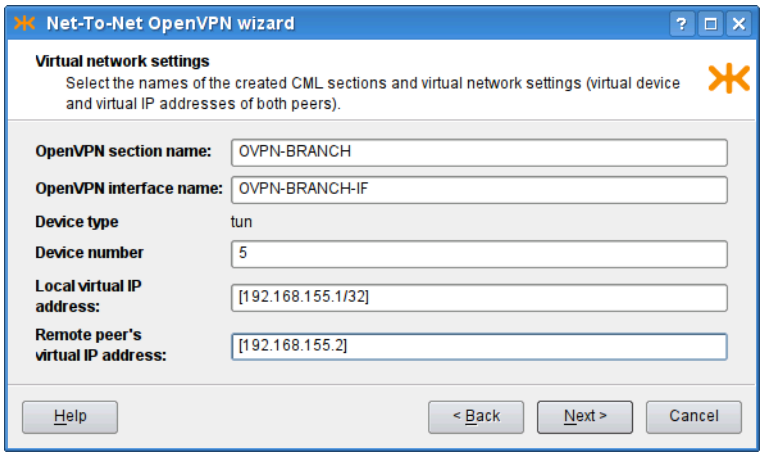

Figure 5.94: Net-to-Net OpenVPN wizard: Virtual network settings page

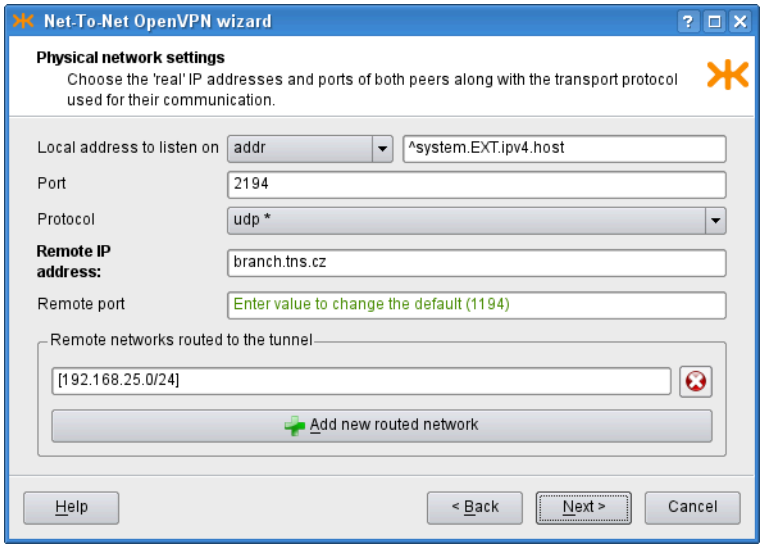

Figure 5.95: Net-To-Net OpenVPN wizard: Physical network settings page

The following page of the wizard (see Figure 5.96) is used to specify authentication options. We can choose between the TLS mode[20] and the Preshared static key file[21]. We have chosen to use the TLS-Server mode and provided the CA certificate, the Local certificate and the Local private key. Finally, on the last page, we can adjust miscellaneous settings; we keep the default values.

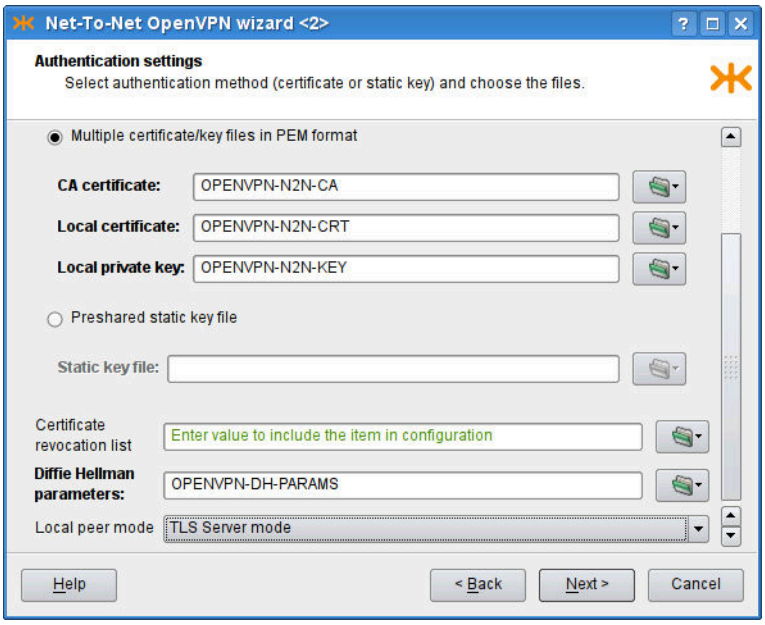

Figure 5.96: Net-To-Net OpenVPN wizard: Authentication settings page

The resulting configuration is depicted in Figure 5.97. We have shown the configuration for one of the peers; the complementary configuration should be used on the other.

5.22.3 Accessing the virtual network

We have shown how to create virtual connections between peers (1:N in Section 5.22.1 or 1:1 in Section 5.22.2). If the network traffic is supposed to "go through" the Kernun UTM system, it must be explicitly configured so. There are several ways to achieve this.

One way to set the policy for the network traffic that comes through the virtual network is to provide a proxy for each service that should be accessible from the other side of the virtual network. The proxy would be configured like any other regular proxy.

Another method of interconnecting two (or more) networks is to arrange ip-forwarding among them. See Section 5.1.4 for more details.

[20] Either a single PKCS12 or the tuple of PEM files (certification authority's certificate, local peer's certificate and the local peer's private key) should be specified. In this case, one peer must be specified to be the TLS server, and the second to be TLS client.

[21] The same secret key is supposed to be used for both peers. It can be generated from the "Create new shared key" dialog.

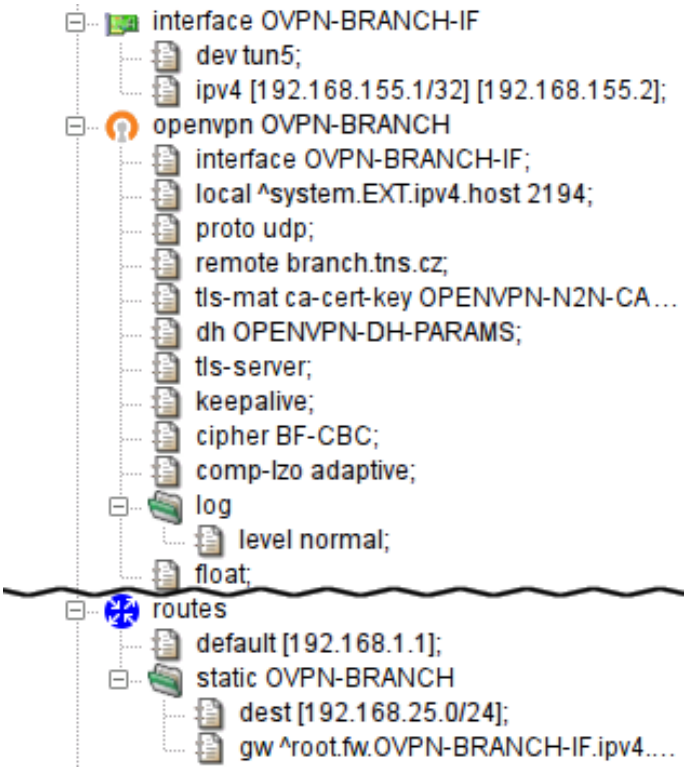

Figure 5.97: Net-To-Net OpenVPN wizard: The resulting configuration

5.22.4 Logs

Logs of the openvpn process are stored in the system log (`/var/log/messages`). Logs generated by the Kernun OpenVPN parent and other auxilliary processes are stored in the Kernun log (`/var/log/kernun-debug`).

5.23 Virtual Private Networks — IPsec

Kernun supports IPsec in the transport and tunnel modes, using the ESP protocol. In the transport mode, Kernun requires a tunnel (GIF or GRE) interface and uses IPsec to encrypt the encapsulated traffic passed via this interface. In the tunnel mode, IPsec encrypts traffic between the specified networks. IPsec in Kernun consists of several parts, represented by components in GKAT. IPsec packets are handled by the operating system kernel according to the SPD (Security Policy Database) and SAD (Security Association Database). The SPD entries for individual IPsec tunnels are controlled by the ipsec Kernun components. Security associations are created by ISAMKP daemon Racoon, presented also as a separate component.

A GKAT window with IPsec-related components is depicted in Figure 5.98. There is a GIF tunnel network interface called `GIF-IPSEC`, which is used by the IPsec tunnel represented by the `TRANSPORT` component. Another IPsec tunnel, the `TUNNEL` component, uses IPsec in the tunnel mode (without a related tunnel interface). The `RACOON` component is the ISAKMP daemon.

The examples of IPsec configuration described here are available in the sample configuration

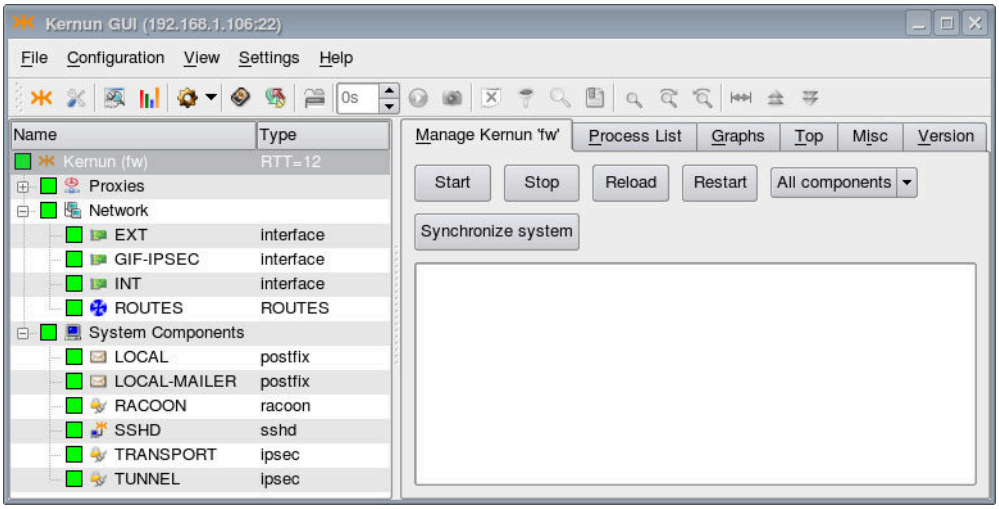

Figure 5.98: IPsec components in a GKAT window

file `/usr/local/kernun/conf/samples/cml/ipsec.cml`.

5.23.1 IPsec Configuration

```
interface GIF-IPSEC
    dev gif0;
    ipv4 [192.168.30.1/24] [192.168.31.1];
    tunnel ^system.EXT.ipv4.host [192.168.1.1];
ipsec-global
ipsec TRANSPORT
    transport-mode GIF-IPSEC;
    phase1
        authentication x509 IPSEC-CA IPSEC-CERT IPSEC-KEY;
    phase2
ipsec TUNNEL
    tunnel-mode
        local { [192.168.20.0/24] };
        protocol { tcp, udp, icmp };
        remote { [192.168.21.0/24] };
        sa-mode network;
        tunnel ^system.EXT.ipv4.host [192.168.1.1];
    phase1
        authentication psk secret;
    phase2
```

Figure 5.99: IPsec configuration

The resulting IPSec configuration in both modes is shown in Figure 5.99. When applied, IPsec-related components `GIF-IPSEC`, `RACOON`, `TRANSPORT`, and `TUNNEL` will be created and become visible in GKAT (see Figure 5.98). For IPsec in the transport mode, the configuration contains the tunnel interface `GIF-IPSEC`. Its configuration section defines the interface device name, local and remote logical IP addresses, and the tunnel addresses (the physical IP addresses used for the encapsulation of tunneled packets). Section `ipsec-global` can contain global parameters of the ISAKMP daemon. There is a section for each IPsec tunnel. Section `ipsec TRANSPORT` config-

ures IPsec in the transport mode for encryption of packets travelling via interface GIF-IPSEC. The authentication utilizes X.509 certificates. Other ISAKMP phase 1 and phase 2 parameters have the default values. Section ipsec TUNNEL sets up IPsec in the tunnel mode. It is not related to any network interface, hence the networking parameters — local, remote, and tunnel (physical) addresses, set of protocols handled by IPsec, and security association mode — are defined inside the ipsec section. A pre-shared secret key for authentication is specified. Again, the remaining ISAKMP parameters have the default values.

5.24 High Availability Clusters

To reduce the risk of system failure, Kernun UTM allows to build hot stand-by clusters. As the name suggests, apart from the main Kernun UTM system, there is a copy of it, usually equipped with the same features and configuration, ready to step up and start handling the communication automatically if the main node fails.

Important

When running Kernun in the virtual environment (like VMWare, VirtualBox, Hyper-V etc.), the network adapter to be used for the shared IP address MUST be set the Promiscusous Mode.
VirtualBox: Settings -> Network -> Adapter -> Promiscuous Mode: Allow All
Microsoft Hyper-V: Settings -> Network Adapter -> Advanced Settings -> Enable Spoofing of MAC Addresses
VMWare vSphere: ESXi/ESX Host -> Configuration -> Hardware -> Networking -> Properties -> Edit -> Security tab -> Promiscuous mode -> Accept
Citrix Xen 6.5: no special settings needed

In this chapter, we assign the systems labels “node A” and “node B”. The system that is currently in charge of traffic control is called *master node* within the cluster, while its idle peer is *backup node*. Under normal circumstances, node A is master and node B is backup. Naturally, both nodes must be connected to the same networks, so we assume that the number of active network interfaces on both nodes is the same. Figure 5.100 illustrates a typical cluster topology consisting of nodes fw-a and fw-b, securing communication between an internal network (let us call it INT) and the external network—Internet (EXT).

As we can see, at least three IP addresses are needed for each network to build a two-node cluster; each node must have its own IP address, and a shared cluster IP address must be present. For example, in the network INT, node A has got IP address 10.0.2.1, node B 10.0.2.2, and the shared cluster address is 10.0.2.3. In the external network, node A has been assigned the address 192.0.2.1, node B 192.0.2.2, and the shared address is 192.0.2.3.

There also needs to be a special “heart-beat” link (simple ethernet connection between nodes) for cluster nodes to see each other. A dedicated deamon pikemon listens on this interface on both nodes and informs each other of their state and priority. In this example node A has address 169.254.1.1 and node B has address 169.254.1.0.

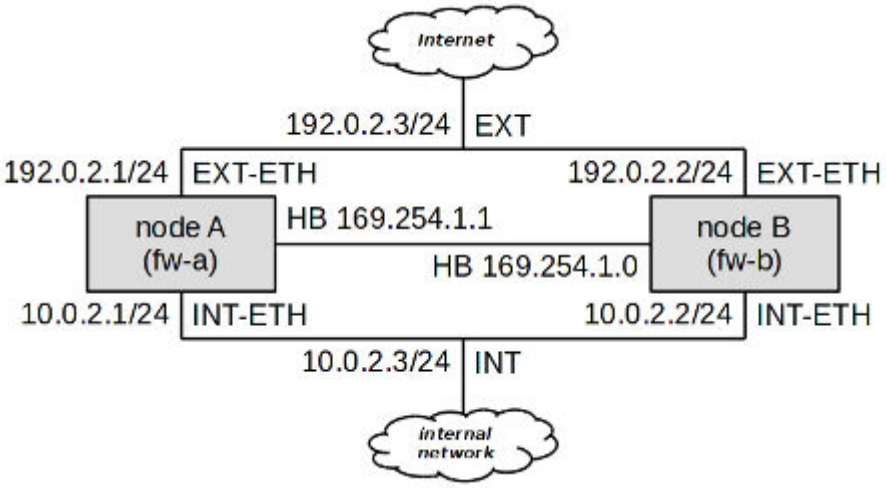

Figure 5.100: Simple cluster with two nodes and two networks

Important

The `pikemon` daemon determines his peer address on heart-beat network by performing XOR on the last bit of his own IP address. If unsure which addresses to choose, use addresses from this sample configuration.

It would be tedious to keep configurations of both cluster members up-to-date. To simplify cluster administration, Kernun UTM makes it possible to specify multiple `system` sections in the same configuration file. Normally, such a configuration file resides one of the nodes, say node A in our example. When the configuration is to be applied, we need to recognize which of the systems is local and which is remote, and how to reach the remote system. The `apply-host` configuration directive serves this purpose. If not present, the system configuration is applied locally. Otherwise, its parameter specifies the host name or address and the SSH daemon's port the system configuration should be applied to. Figure 5.101 illustrates the use of the `apply-host` option.

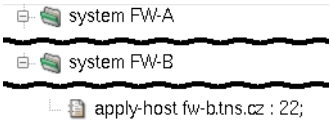

Figure 5.101: `apply-host` is used to distinguish the local system from the remote one

The existence of two system configurations in the same file would not relieve us of the burden of keeping both system configurations up-to-date. We shall use a *section variable* (as defined in Section 4.1) to simplify cluster administration further.

First, let us create a system variable called `CLUSTER`. Figure 5.102 shows its definition. Apart from its name, there are four parameters that distinguish individual nodes of the cluster:

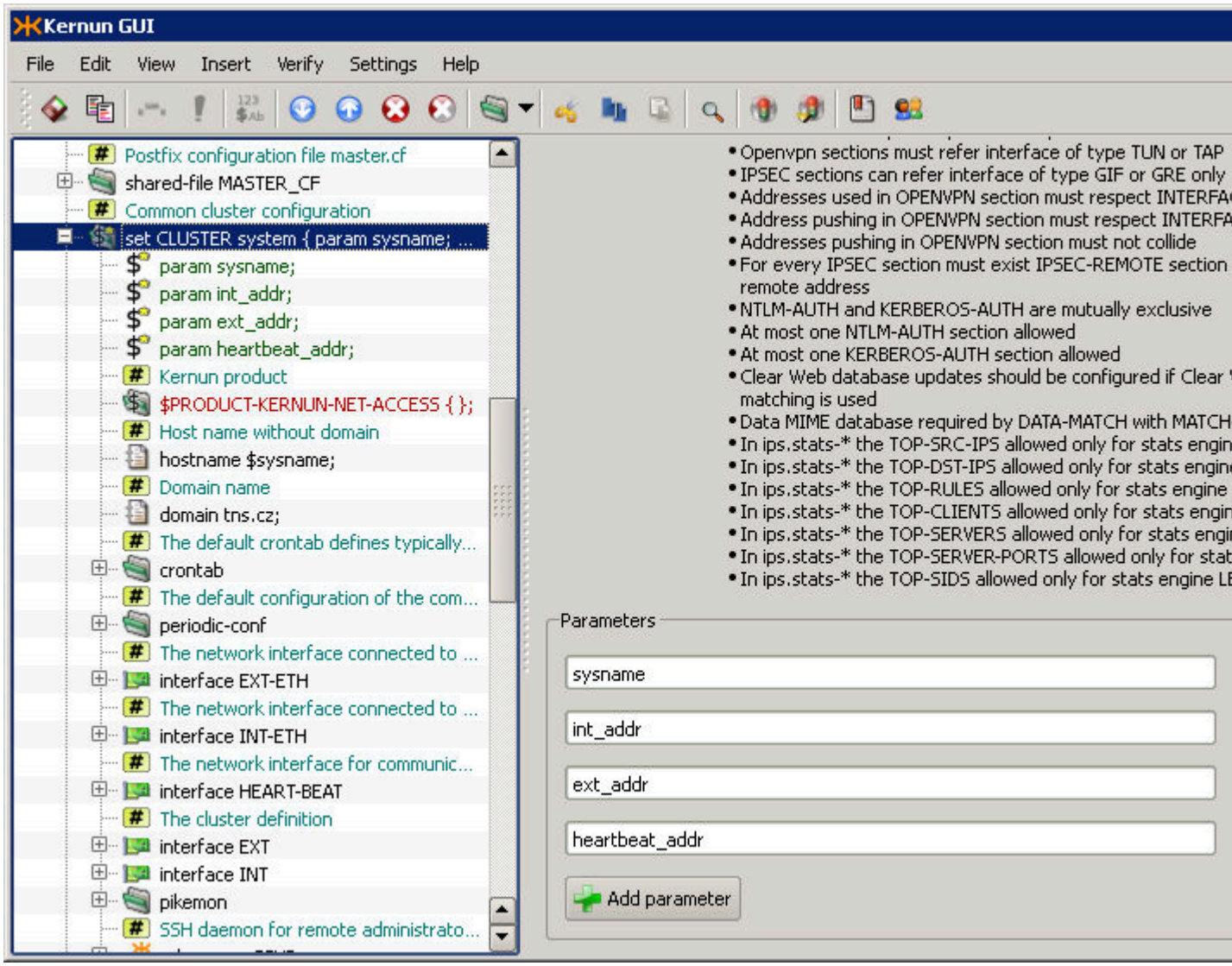

Figure 5.102: Definition of system variable CLUSTER

- `sysname` — The name of the system.
- `int_addr` — The internal IP address of the system (not the shared one).
- `ext_addr` — The external IP address of the system (not the shared one).
- `heartbeat_addr` — The heart-beat IP address of the system.

The system variable `CLUSTER` includes what both nodes have in common, which is practically the whole of their configuration. They differ only in the four parameters described above, and in the way the configuration is applied. The parameters are used on their appropriate places within the `CLUSTER` definition, as shown in Figure 5.103. For instance, the `$sysname` parameter is used within the `hostname` directive to define the system's host name (the row marked ❶). The real internal address, specified in parameter `$int_addr`, is used inside the interface `INT-ETH` definition (row ❷). Similarly, parameter `$ext_addr` is used to specify the IP address of interface `EXT-ETH` (row ❸) and `$heartbeat_addr` is used for interface `HEART-BEAT` (row ❹).

Figure 5.103: Usage of parameters inside `CLUSTER` definition

The section variable `CLUSTER` defined above needs to be referenced to use its contents to define a real Kernun UTM system. Upon referencing the variable, its parameter values must be filled in.

In our example, we use the variable CLUSTER twice to define two nodes: system fw-a and system fw-b, as depicted in Figure 5.104.

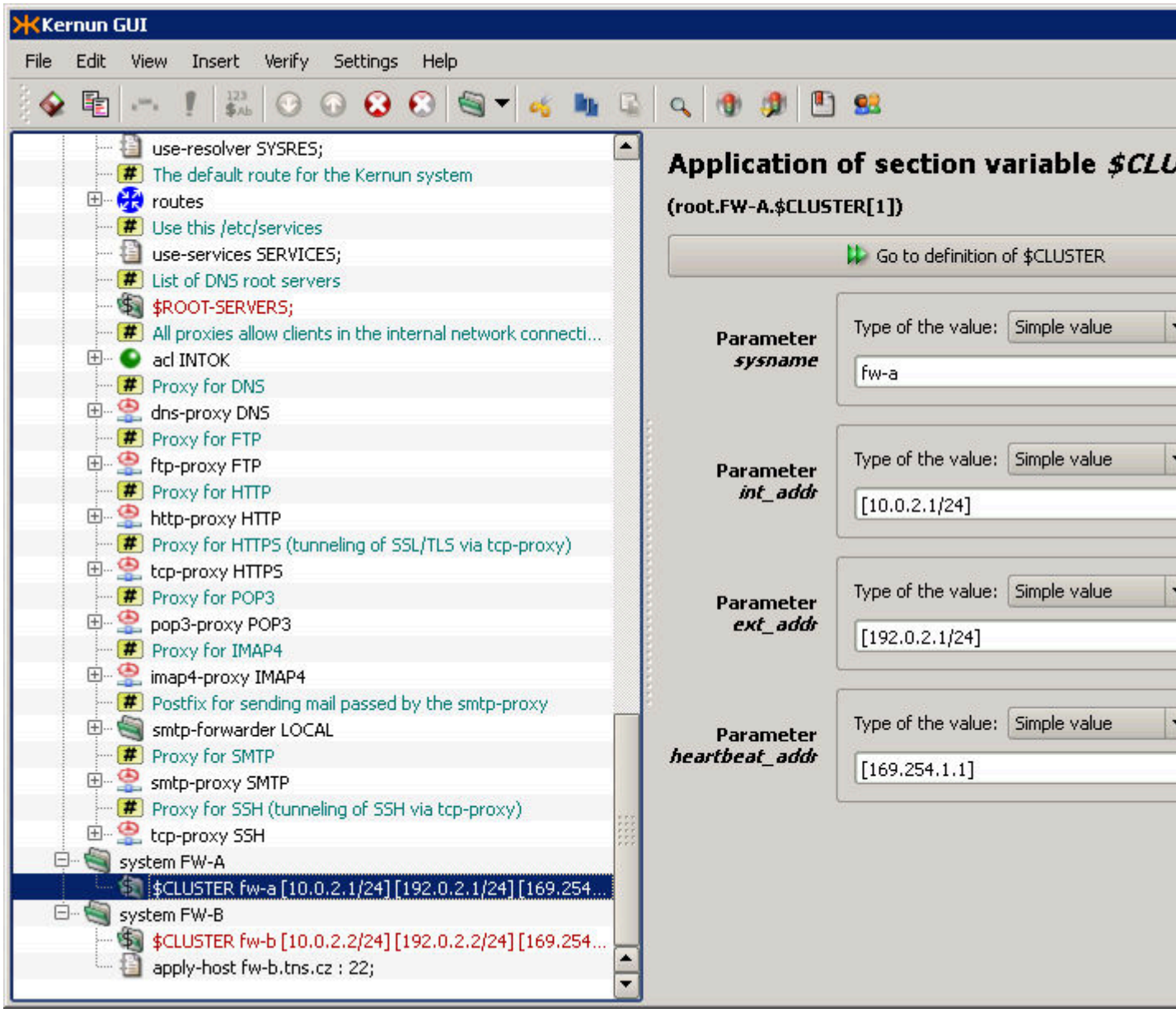

Figure 5.104: Use of variable CLUSTER to define two nodes

Cluster functionality, as described above, ensures that if node A fails, node B takes over automatically, causing minimal communication blackout (a few seconds at the most). However, if the failure is related only to one of the network interfaces (i.e. switch port fault, cable or network interface failure), say on interface EXT of node A, we would end up with node A still being master for network INT, but node B being master for network EXT. This is incorrect, as packets going from internal clients out would go through node A (master for the internal network), which is not properly connected to the external network.

To handle such conditions, Kernun UTM makes it possible to tie cluster interfaces together. A self-detector monitors the state of its interfaces by pinging other hosts on all relevant networks. By pinging a remote host, Kernun UTM makes sure that the network interface is working properly, including cabling and switch port.

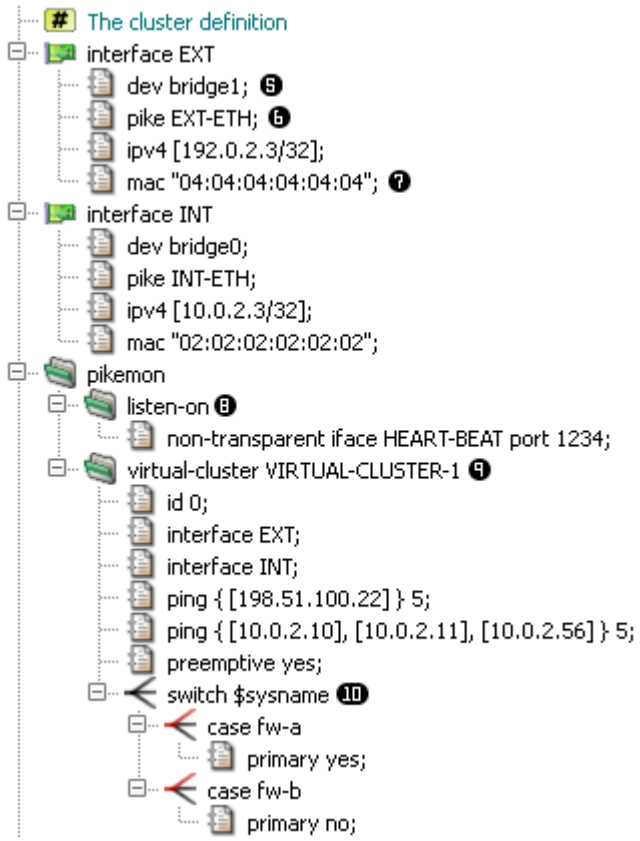

Figure 5.105: Definition of `pikemon` and virtual clusters

Shared IP addresses (alias "virtual ip addresses" or "cluster addresses") are not bound to any physical network interface, instead they are assigned to a virtual `bridge` interface (row ❺) [22]. The `pike` item (row ❻), binds particular virtual interface to a physical one. As the virual IP can migrate from one node of cluster to another, it is necessary for both nodes to advertise same MAC address for this IP, as defined in `mac` item (row ❼).

Important

For the shared IP address to migrate properly, it is necessary on some switched to allow one MAC address to be present on multiple switch ports. On Cisco switches, this requires setting the `port-security` option off for those ports.

The `pikemon` section defines one or more virtual clusters and a listen socket for it's daemon to listen on (row ❽). You can choose any free port on the HEART-BEAT interface.

[22] The interface is created automatically when used in an `interface.dev` item.

The `virtual-cluster` section (row ❾) defines a logical group of virtual IP addresses and their properties. In this example we want the EXT and INT interface to be part of one virtual cluster, so that in case of disruption of communication of one node with internal network, the node also give up it's master role on EXT interface. Such disruption is detected by timeout of ping request for defined group of hosts. When more addresses are defined in one `ping` item, all of these addresses must fail to respond to ping request for a cluster node to lower it's priority. In this example, unavailability of `198.51.100.22` will result in a node loosing master role, on the contrary unavailability of `10.0.2.11` would not, because all of the 3 hosts in the second `ping` item must be down.

Tip

When choosing hosts for cluster monitors, take into consideration that the remote host should be always up. The best choice is a router or switch, network servers being the next best option.

Next, we define a `switch` command (row ❿) which enables us to use different configuration for each node (fw-a and fw-b). We set node A (fw-a) to by primary node of this sample cluster.

In the `virual-cluster` section, it is possible to influence another aspect of cluster behavior. Suppose node A had a failure, and node B has become master. Once node A is recovered, node B could either remain the master node, or hand the master status back to node A. By default, the former is true, but we may force the latter by adding the `preemptive` directive.

Tip

It is highly recommended to use the `preemptive` option in all cluster configurations.

Important

In cluster configuration, both transparent and non-transparent proxies need to listen on the virtual `bridge` interface (INT and EXT interfaces in this sample). The same applies to all `packet-filter` rules except for `nat-acl`s which must be set on physical network interface (INT-ETH or EXT-ETH). It is also suitable for some services to listen on physical interface addresses. for example SSH daemon should listen on physical IP address, so it would be possible to connect to both nodes. See `/usr/local/kernun/conf/samples/cml/cluster.cml` for more examples.

5.24.1 Controling multiple systems from GUI

In order to control more systems from a single GUI, set up the following daemons: icamd(8) and icasd(8). The master (icamd) allows the slaves (icasd) to be controlled. The situation is shown in Figure 5.106: the GUI is connected to the system pha (bold font in the tree view). The systems pha-b and bno can be controlled, because they are connected to the system pha via icamd/icasd.

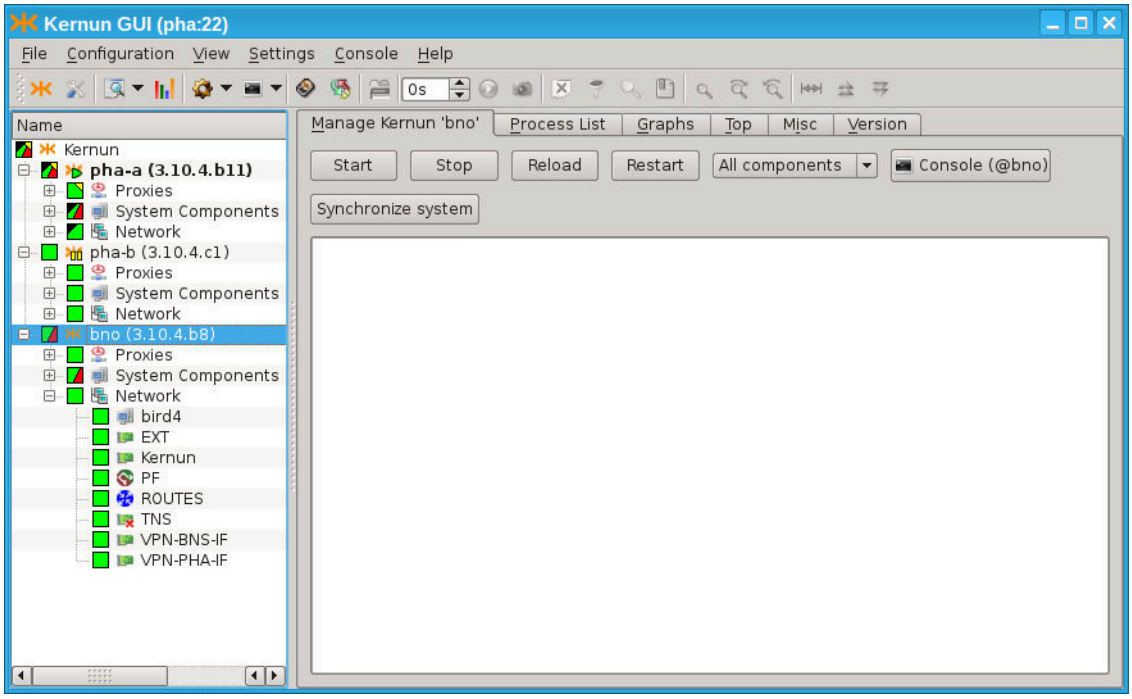

Figure 5.106: Multiple systems can be controlled using icamd / icasd daemons

There is a simple way to setup icamd/icasd daemons for the cluster of two systems. Both daemons are configured by the single item `SYSTEM.ica-auto`. See Figure 5.107. The rsa key pair is provided for mutual authentication. The communication port is provided.

The more complicated topologies are described in the configuration explicitly by the configuration of the icamd/icasd daemons, as shown in Figure 5.108: The configuration for two systems (pha-a and pha-b) is provided. They can control each other. Morover, there is a third system (bno), which is also connected via icamd/icasd (but system bno does not share the configuration with pha-a and pha-b). The configuration is symetric for pha-a and pha-b. Both define the icamd (for being able to control the other systems), and icasd (for being controlled by the other systems). The ssh-key pair is provided for each system's icamd and each system's icasd (6 key pairs in total: 3 key pairs for icamd daemons, 3 key pairs for icasd daemons). The listen port is provided for the icamd daemons. The address and port is provided for each icasd's master section.

5.24.2 Sharing the configuration among systems

Should the configuration be shared among several Kernun systems (for example in the clusters (Section 5.24) or KBA (Section 5.25), the `SYSTEM.config-sync` configuration item should be used. This item implies that the configuration is automatically synchronized to the remote system, whe being remotely applied.

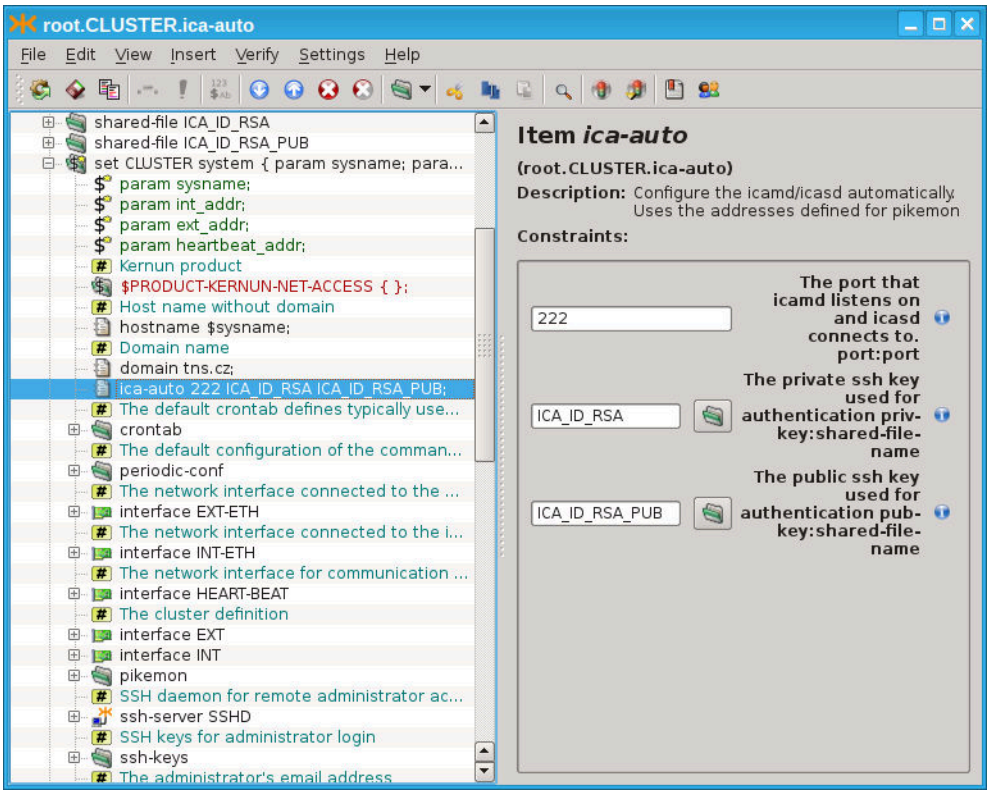

Figure 5.107: Simplified icamd/icasd configuration using item `ica-auto`

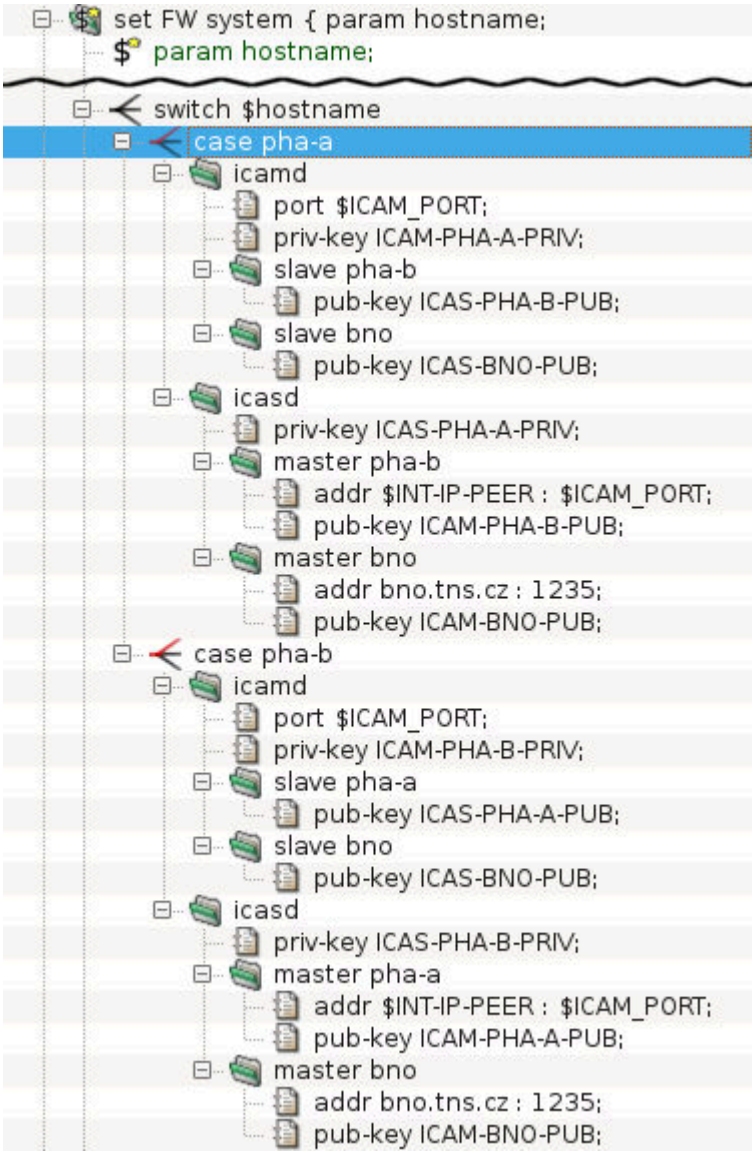

Figure 5.108: Full icamd/icasd configuration

5.25 Kernun Branch Access

Kernun Branch Access is a device with the primary task to provide a remote branch of a company with a secure connection to a central Kernun UTM. In addition, Kernun Branch Access provides another security features for the perimeter of the network such as network packet filtering and VPN. Optionally, the device can also serve as a Remote Access Server for VPN. Kernun Branch Access is designed to be used together with Kernun UTM and should not be used separately.

5.25.1 Description and Plug-in

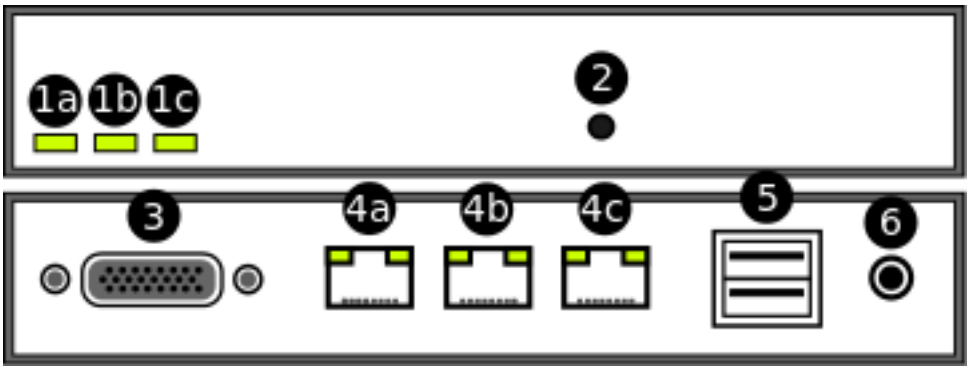

Figure 5.109: A schema of the front and the back face of Kernun Branch Access

1. Status LEDs (1a, 1b, 1c) — Three LEDs, which are intended to indicate the status of some key functions of the device.
2. Button — Serves for confirming configuration application and starting the Remote Help Service. In order to press the button, use a paperclip or a similar tool.
3. A serial console connector
4. Ethernet ports (4a, 4b, 4c) — Kernun Branch Access is equipped with three Ethernet ports. The ports serve for connecting the device to LAN and WAN.
5. USB ports — Kernun Branch Access is equipped with two USB 2.0 ports.
6. Power supply input

In order to power Kernun Branch Access on and connect it to the network properly, the following steps should be taken:

- Plug-in a power supply connector to the power supply input (6)
- Plug-in an Ethernet cable connected to LAN into an Ethernet port (4b)
- Plug-in an Ethernet cable connected to WAN into an Ethernet port (4a)

5.25.2 Installation

In order to install Kernun Branch Access, you need a Kernun Branch Access installation medium[23]. Since the device is equipped with serial console output only, it is also necessary to modify installation process according to the guide described in Section 2.5.5. After this modification, the operating system will output to serial console while booting and the installation will be similar to the installation of Kernun UTM from standalone installer described in Section 2.5.1.

5.25.3 Configuration

Configuration of Kernun Branch Access can be performed in two different ways:

1. Initial configuration using a USB flash drive, which is typically applied when Kernun Branch Access is configured for the first time.

2. Standard configuration applied remotely from a central Kernun UTM.

Both of the configuration options will be described in the following sections.

Initial Configuration

Kernun Branch Access is usually supplied in the form of a compact device configured to default factory settings. Since the supplied device has no active network connection available yet, no remote configuration technique similar to the one described in Section 5.24 can be used. In order to configure the Kernun Branch Access for the first time, it is necessary to apply the configuration using a USB flash drive.

The procedure consists of two steps. Firstly, the configuration for Kernun Branch Access is defined on the central Kernun UTM and then, using a series of commands, it is exported into a special configuration file. Secondly, the exported file is copied to an arbitrary USB flash disk and the disk inserted into one of USB ports on the device. The whole procedure is described in details in the following paragraphs:

1. Exporting a configuration file from a central Kernun UTM

   - Defining configuration for Kernun Branch Access on central Kernun UTM

     The configuration is defined as a single `system` section (see Section 4.2.2 for more details on `system` configuration basics).

   - Generating and exporting the `system` configuration into a file using the **kat**(8) command line tool

     The tool can be launched either directly using an SSH remote access to the terminal of the central Kernun UTM or by pressing the GUI Console icon (marked as number 6 in Figure 3.14).

[23] You can use `dd` on Linux / BSD or `https://github.com/openSUSE/kiwi/downloads` on Windows to copy the USB flash drive image to the device.

```
KAT@central-utm> cml
CMLR-710-K File '/usr/local/kernun/conf/kernun.cml' loaded
CML> /generate
...
CKGB-710-N ---- Preparing files for system 'kba-office'...
...
CKGB-719-N ---- System 'kba-office' successfully generated
CML> /quit
CMLI-709-N CLI interactive mode closed
KAT@central-utm> export kba-office
CMLK-821-N [root] Exporting 'SYSTEM-kba-office' configuration
Exporting configuration to /root/SYSTEM-kba-office.tar
tar: Removing leading '/' from member names
KAT@central-utm> quit
```

After applying the `export` function, the configuration file named `SYSTEM-kba-office.tar` should be exported to the home directory of the user who launched the command.

- Transporting the generated file to the administrator manipulating the Kernun Branch Access.
  In case the administrator who generated the configuration file does not have physical access to Kernun Branch Access, the file must be transported to the administrator manipulating the device itself. It might, for example, be sent by an e-mail.

2. Preparing and inserting the configuration USB flash drive into Kernun Branch Access

   Once the configuration file is exported and delivered to the administrator having physical access to Kernun Branch Access, the following steps should be taken:

   - Copy the configuration file (e.g. `SYSTEM-kba-office.tar`) to the root directory of an arbitrary USB flash drive
   - Insert the USB flash drive into one of USB ports of Kernun Branch Access (5)
     After inserting, the device will beep once shortly and then wait 30 seconds for confirming the application by pressing the button (2). If the timeout runs out, the device will beep nine times shortly and no configuration will be applied. In such case, the USB flash drive must be reinserted and confirmed.
   - Confirm the configuration by pressing the button (2) with a paperclip or a similar tool
     If the button is pressed, the device beeps three times shortly and automatically starts the configuration process. If the process succeeds, the device beeps once longly, otherwise it beeps three times longly.
   - After successful configuration, the network connection with central Kernun UTM should be established and Kernun Branch Access can be further configured remotely using the process described in the following section.

> **Note**
>
> The way of configuration using a USB flash drive is primarily intended for setting the very first configuration, when Kernun Branch Access has no network connection with the central Kernun UTM available. However, it may be reused arbitrarily when needed in the future.

Remote Configuration

If the initial configuration using a USB flash drive has already been successfully applied, Kernun Branch Access can be further configured remotely. The desired configuration can be prepared as a standalone system on the central Kernun UTM with `apply-host` configuration directive applied. Instructions for proper setting and applying the `apply-host` directive can be found at Section 5.24.

5.25.4 Diagnostics and Troubleshooting

Kernun Branch Access is intended to operate without a monitor or any kind of a administrator's console. Therefore, it has only limited possibilities to indicate its current state or communicate with users and administrators. Three build-in LEDs serve for basic diagnostics of the key functions of Kernun Branch Access. In this section the meaning and the purpose of each LED will be described. It will also be explained how the Remote Help Service can be started and what its purpose is. Finally, steps to be taken in order to reset the device back into the default factory settings will be described.

Status LEDs

Kernun Branch Access is equipped with three LEDs, which are intended to indicate the status of some key functions of the device. The meaning of each LED is as follows:

- The left LED (1a)

  Indicates whether Kernun Branch Access is powered on. If so, the diode should light continuously without interruptions.

- The middle LED (1b)

  Indicates a status of all configured KBA components after a boot of the system. After the system boots, the diode starts to blink, which indicates that the KBA components are being verified. If all the components are successfully started, Kernun Branch Access beeps once long and the diode starts to light continuously. If not, Kernun Branch Access beeps three times long and the diode turns off.

- The right LED (1c)

  Serves for indicating a status of the Remote Help Service. When the service is started by pressing a Remote Help Service button (see Section 5.25.4), the diode will blink quickly until the service is ready (the process should not last longer then 60 seconds). If the service was started successfully, the diode should begin to light continuously, if not, it turns off.

Similarly, when the service shuts down, the diode begins to blink until the process is finished and then it turns off.

Remote Help Service Button

The Remote Help Service is a service allowing the technicians of the manufacturer of Kernun UTM temporary access to Kernun Branch Access in order to diagnose and maintain it remotely. The service can be both started and stopped by pressing the button located on the front side of the device. The entire starting procedure consists of the following steps:

- Press the button (2) with a paperclip or a similar tool for 5 seconds.
- The right LED (1c) starts to blink.
- When the right LED begins to light continuously, the Remote Help Service was started successfully. The diode will light all the time the service will be active.

5.26 IPv6

Kernun UTM supports IPv6 since version 3.5. An example of an IPv6-enabled configuration can be found in the sample configuration file `/usr/local/kernun/conf/samples/cml/ipv6.cml`. It is a dual-stack configuration with IPv4 and IPv6 enabled both in the internal and the external network. Clients from the internal network can access HTTP and SSH servers in the external network. For a transparent (HTTP or SSH) proxy, the target server IP address is taken from the destination address of the client's connection. Hence the same IP version is used by the client and the server. In the case of the non-transparent HTTP proxy, the target server name is passed by the client to the proxy in the request URI. The proxy resolves the name and establishes connection to the server. Hence an IPv4 client can access an IPv6 server and vice versa.

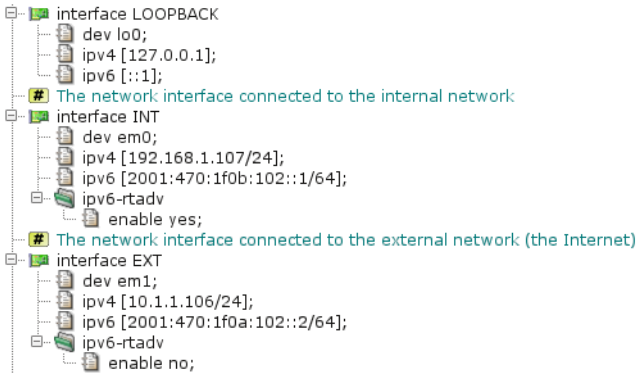

Figure 5.110: IPv6 interfaces

The configuration of network interfaces is depicted in Figure 5.110. There are three interfaces: `LOOPBACK` is the standard system loopback interface; `INT` and `EXT` are the internal and the external Ethernet interfaces, respectively. IPv6 support in Kernun UTM is enabled if at least

one IPv6 address is assigned to any network interface in the configuration[24]. IPv6 addresses are defined by the `ipv6` item. It is also possible to specify IPv6 aliases by the `alias` section. By default, Kernun UTM operates as an IPv6 router, but it does not send router advertisements. Router advertisements can be turned on and their parameters can be set for an interface in the `ipv6-rtadv` section.

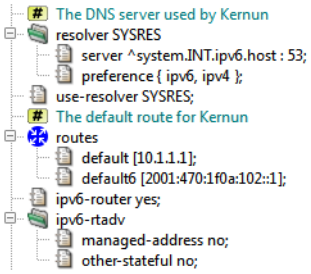

Figure 5.111: IPv6 networking parameters

Some global IPv6-related networking parameters are depicted in Figure 5.111. The `resolver` section defines the name server address[25]. A single domain name can be resolved to an IPv4 or an IPv6 address. The `preference` item selects, which resolved addresses will be used. There are four possibilities: use IPv4 and ignore IPv6, use IPv6 and ignore IPv4, use both and prefer IPv4, or use both and prefer IPv6. The last choice is used in the sample configuration. The default is to prefer IPv4. The `routes` section defines the default IPv4 and IPv6 routes. It is also possible to add static routes to IPv4 and IPv6 networks by the `static` item. The `ipv6-router` item enables or disables forwarding of IPv6 packets. IPv6 forwarding is enabled by default, so this item can be omitted. The `ipv6-rtadv` section defines default values for IPv6 router advertisements. These defaults can be overriden by an `interface.ipv6-rtadv` section. In the example configuration, router advertisements are configured so that they provide address and default route autoconfiguration and DHCPv6 is not used.

The sample IPv6 configuration contains HTTP and TCP proxies, depicted in Figure 5.112. Their configurations are similar to IPv4-only proxies. The only differences are IPv6 addresses in the `listen-on` items and in `acl INTOK`.

IPv6 addresses can be used in place of IPv4 addresses in many other places in the Kernun UTM configuration. It is possible to use other proxies for IPv6 communication, as well as define IPv6 packet filter rules and IPv6 IPsec VPN connections. We do not show configurations of these components here, because they are essentially the same as the respective IPv4-only configurations. DHCPv6 server is configured in the `dhcp6-server` section, in a way similar to DHCPv4 server in the `dhcp-server` section. Fixed IPv6 addresses and AAAA DNS records can be defined by entries in the `hosts-table` section.

There are also some important differences and limitations of IPv6. Destination IPv6 address

[24] Note that in addition to explicitly assigned addresses, each interface has a link-local IPv6 address assigned automatically by the operating system.

[25] In this sample configuration, a local caching nameserver chained to the DNS proxy is used.

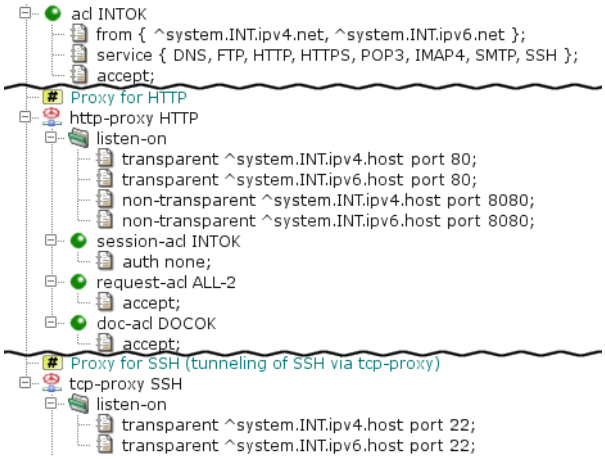

Figure 5.112: Proxies with IPv6 support

must not be set for a point-to-point network interface (TUN, GIF, GRE). The GRE network interface type does not support IPv6 tunnel addresses. OpenVPN cannot use IPv6 transport. The `mac` value of an IPv6 entry of the `hosts-table` is interpreted as the host's DUID.

5.27 Honeypot

The idea behind honeypot is to provide a resource valuable to attacker and by monitoring attackers behaviour counteract his attempts to unauthorized use of information systems. In an example scenario, we dedicate one public IP adress as "honeypot" address. This IP address should not be referred in any DNS record and must not by referred in any web site or elsewhere, so that we can assume only attackers automatically scanning all IPs attempts to connect to this honeypot address. Automatic scans function generally in two modes. Some scanners send TCP SYN packets to a range of IP addresses, immediately forgeting about it a detecting open ports by simply monitoring incomming TCP SYN+ACK packets (see `http://en.wikipedia.org/wiki/Transmission_Control_Protocol#Connection_establishment` for explanation of TCP connection establishment). Other scanners attempts to complete TCP handshake and eventually continue the scan on application level. Kernun UTM logs all attempts to connect to honeypot and blacklists all IP addresses that completed TCP handshake. Once on blacklist, any traffic (any protocol) on any network interface (not just honeypot address) from that IP address is blocked.

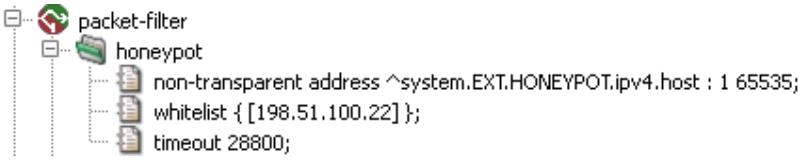

Figure 5.113: Honeypot

The section `honeypot` is part of the `packet-filter`. The item `non-transparent` defines

on which address and ports should we listen for attacker's connections. In this example it's all available ports. In `whitelist` we defined IP address of our monitoring server which should not be placed on blacklist even if it connects to honeypot address. Finally `timeout` defines time period (in seconds) for which IP address should remain in blacklist, after it became silent, i.e. after there is no new connection from that IP address on any of our network interfaces.

Appendix A

NAME

HtmlMatchPasswd.pm — encapsulates the databases of the HTML form value control tool html-match-db(1) for storing two couples of credentials (internal username and password and external username and password). It also keeps the logs of the actions over the particular accounts.

SYNOPSIS

HtmlMatchPasswd.pm -f *root_dir* [-bpSv] [-u *iu*] [-d *log_level*] { -a| -C| -l| -L| -r| -R| -s| -n }

DESCRIPTION

Utility **HtmlMatchPasswd.pm** is used to manage password files (pairs of internal/external usernames/passwords database) used by the data matching and processing module, see also html-match-db(1), data-matching(7). It provides functions for adding, deleting, listing users, setting the password, etc. It provides a perl package interface for accessing the database by other programs (i.e., www user interface for changing the password by the users). While the command-line interface is intended to be used by the administrator, the perl package interface also provides functionality intended for the end users (check the old password before it can be changed, etc).

Two password files are kept: `var/passwd.db` and `var/passwd`. The first one (`passwd.db`) keeps the database in the form it is used by the http-proxy (see data-matching(7)). The other file (`passwd`) keeps more information (in particular, it stores the unencrypted local usernames), so it is possible to change credentials, remove the user without knowledge of the internal password, etc.

The activity of users is logged. Each user's activity is logged in file `log/internal_username`. You can view the log using `-L`.

Options

-a Add an user. Internal username, internal password, external username and external password are asked to be entered.

-C Check the consistency of the two databases (see above). Use `-R` to repair inconsistency.

-l Print the list of internal usernames.

-L Show the user log. Use `-u` to limit the log to a single user. Without `-u`, the log of all users is printed.

-r Remove a user. Only internal username is asked.

-R Checks the consistency of the two databases (see above). If an inconsistency is found, it is repaired (HASH found only in `passwd.db` is removed from it, HASH found only in `passwd` is added into `passwd.db`).

-s Set the credentials for an existing user. Internal username, new internal password, new external username and new external password are asked.

-n Print the license limitations: used, free and total.

-f Specify the root of the **HtmlMatchPasswd.pm** file infrastructure. Directories *root_dir*/var, *root_dir*/var/log, and files files *root_dir*/var/lock, *root_dir*/var/passwd, *root_dir*/var/passwd.db, *root_dir*/var/log/*internal_username* are eventually used. The var directory is assumed to exist; the remaining ones are created if they do not exist. All of them must be writable for any user that manipulates the databases.

-b Batch mode. Does not display prompts when asking questions, etc.

-d Set the log level explicitly. Otherwise, environmental variable LOG LEVEL is used. Acceptable values: 0–9.

-p When asking for password, the terminal ECHO is normally switched off so it is read silently, without being displayed. This can be overriden by specifying -b.

-S When asking for the new internal password, its strength is not checked (and even a weak password is accepted).

-u *iu* When displaying the user log (see option -L), limit it to the particular user.

-v More verbose mode.

Exit Code

The program returns exit code 0 if the command is successfully performed, and a nonzero if an error occurs.

SEE ALSO

html-match-db(1), data-matching(7), http-proxy(8)

NAME

clear-web-db-update.sh — tool for updating the Clear Web DataBase

SYNOPSIS

clear-web-db-update.sh [-w] [-x]

DESCRIPTION

Utility **clear-web-db-update.sh** updates the Clear Web operational database, which is used by the http-proxy(8) to categorize web pages. It relies on the clear-web-db(1) utility to manipulate the database. It is executed periodically by cron(8) when automatic updates are configured.

Options

-w Waits random number of seconds (0..RANDOM_WAIT_MAX) before downloading the database.

-x Skips downloading the database, only updates the appropriate files in proxy chroot directories.

Exit Status

The program returns the exit status 0 on success and nonzero after any error.

FILES

/data/var/clear-web-db/clear-web.db The standard location of the Clear Web DataBase.

SEE ALSO

clear-web-db(1), http-proxy(8)

NAME

clear-web-db — tool for managing the Clear Web DataBase

SYNOPSIS

clear-web-db { -d| -n| -m }

clear-web-db { -o| -u| -s| -S } [*db*]

clear-web-db -l [-f] [*db*]

clear-web-db -c [*locale*]

clear-web-db -a [*db*] *incr*

clear-web-db -i [*db*] *db2 incr*

DESCRIPTION

Utility **clear-web-db** can be used to create, modify, and view the Clear Web operational database, which is used by http-proxy(8) to categorize Web pages. It is usually called by program clear-web-db-update.sh (1), which handles automatic periodic updates of the database.

URLs used by program **clear-web-db** consist of a server domain name followed by an optional path, including neither the method, nor the port number. Example: `www.tns.cz/index.html` is used instead of `http://www.tns.cz:80/index.html`.

Options

-o Reads the input textual data from the standard input and writes the operational database into file *db*.

-u Reads the input textual data from the standard input and updates the operational database in file *db*.

-i Creates the incremental update data between the operational databases stored in files *db* and *db2* and writes the result into file *incr*.

-a Applies the incremental updates from *incr* to database *db*.

-s Reads URLs, one per line, from the standard input, searches for them in database *db*, and writes corresponding categories to the standard output. Each output line contains the numeric bitmap of categories followed by a space-delimited list of category names.

-S Reads URLs, one per line, from the standard input, searches for them in database *db*, and writes corresponding categories to the standard output along with the part of the URL that was used to determine the categories. Each output line contains the numeric bitmap of categories followed by the matching URL substring.

-l Lists the contents of database *db*. Each line of output contains a hash value, a bitmap of categories, and a list of category names. When listing an incremental update data, records to be deleted are listed as a hash value followed by the character "−". If option `-f` is specified, flags are displayed between the hash value and the bitmap of categories. A flags value is an OR-combination of constants: 0x1 = there may be a record with a longer domain name suffix, 0x2 = there may be a record with a longer path prefix, 0x4 = the record contains categories (without it, the record is used only to continue with a longer domain or path according to 0x1 or 0x2), 0x8 = denotes a record to be deleted (in an incremental update only).

-d Reads URLs, one per line, from the standard input, and writes digests (hash values) to the standard output.

-c Writes all known categories (numbers and names) to the standard output. If the *locale* argument is specified, the category name is localized into given locale.

-n Reads category bitmaps, one per line, from the standard input and writes the corresponding lists of category names to the standard output.

-m Reads lists of category names from the standard input and writes the corresponding category bitmaps to the standard output.

Exit Status

The program returns exit code 0 if the command is successfully performed, and a nonzero if an error occurs.

FILES

/data/var/clear-web-db/clear-web.db The default database file used if argument *db* is not specified on the command line.

SEE ALSO

clear-web-db-update.sh(1), http-proxy(8)

NAME

cluster-sync — tool for synchronizing files between cluster members

SYNOPSIS

cluster-sync [-n] [-d *debug_level*] -c *cfg*

DESCRIPTION

This script performs file synchronization between exactly two (cluster) members.

This script is expected to be executed regularly (i.e. from crontab). It connects to the remote host, looks for changes made on the watched files and transfers the changes from one host to the other. If the file is changed on both hosts, it is considered as the conflict. It is specified in the configuration, which side is prefered in conflicts (CONFLICT_WINNER) below.

The hosts are expected to be interconnected by icamd/icasd daemons. See REMOTE_HOST below.

Options

The values given on the command line have precedence over values given in the configuraiton file.

-c The configuration file to be used.

-d The debug level. Default is 6 (LOG INFO). Increase it for more verbose logging. The script logs to stdout and to /var/log/kernun-debug.

-n Perform a 'dry run'. No sync actions are performed, just log what would be done.

Configuration file

At least REMOTE_HOST and INCLUDE must be given. If the option can be specified both in configuration file and on command line, the command line value takes precedence.

REMOTE_HOST=*TARGET* The ssh target to be used when invoking commands on the other host.

Be sure to use the name defined in /etc/sshd/ssh_config.

There are many commands executed on the REMOTE host, so it is necessary for them to be executed quickly. Daemons icamd/icasd provide fast connections since it takes advantage of the ssh ControlPath feature.

INCLUDE=*"PATH1"[, "PATH2", ...]* The list of the included files, directories or patterns. The absolute path is given, and it must be preceeded by symbol '.'.

Example:

```
INCLUDE="./var/log/kernun-stats.*.bz2", "./data/fake-cert/fake_ca/C*.*"
```

The values must be written in a comma separated list on a single row.

PRECMD=*"command"* The command is executed to determine whether perform the synchronization or not. Use PRECMD to prevent synchronization when it is not desired. For example when the cluster is degraded.

The command can either be a script name or a simple shell script. It is passed to the perl 'system' call.

No PRECMD is performed by default.

MINIMAL_AGE=*seconds* Only files that are older that MINIMAL AGE are taken in account. This should prevent repetitious copying of the files that are being written for longer period (for examples the zipped logs).

Defaults to 0 (the files are synchronized regardless of their age).

LOCK=*filename* The lock file to be locked in order to start the synchronisation. If the lock cannot be (exclusively) locked, the scripts exits immediately.

By default, the lock prevents the script to run more than once in parallel. It can also be used by other programs not to run simultaneously.

CONFLICT_WINNER={LOCAL|REMOTE} Should the conflict appear, this option specifies whos change wins. Defaults to LOCAL.

RM_STALED={0|1} Whether the staled temporary file should be automatically removed. Defaults to 1.

DRY_RUN={0,1} Perform a 'dry run'. No sync actions are performed, just log what would be done.

DEBUG_LEVEL=*num* The debug level. Default is 6 (LOG INFO). Increase it for more verbose logging. The script logs to stdout and to /var/log/kernun-debug.

SEE ALSO

icamd(8), icasd(8),

NAME

diskdb — tool for creating and querying file system content database

SYNOPSIS

diskdb *cmd* [-1CcDFMqrStvx] -d *db* [-d *db*] [-I *filename* ...] [-i *pattern* ...] [-E *filename* ...] [-e *pattern* ...] [-L *filename* ...] [-P *filename* ...] [-R *dir*] [path ...]

DESCRIPTION

Program **diskdb** manages a database of information about the contents of a file system. The database contains a list of files and directories in the file system. For each file (or directory), the remembered attributes are its name, metadata, size, and optionally checksum (only for regular files, using cryptographic hash SHA256). The metadata include the file type, permissions, owner, group, times (access, modify, change, create), file flags, and ACLs. For symbolic links, the link target is also recorded. The database is in a textual format.

It is possible to select a subset of the whole file system directory in several ways:

- One or more *path*s can be specified on the command line or read from a file when creating a database. The operation of the program is then limited to subtrees of the file system hierarchy rooted at these paths. If a path is a directory, its whole subtree is processed by default. If the path arguments are omitted, an empty database is created. If at least one path argument or path file is specified for commands other than **create**, the operation of the program is limited to these paths (either from arguments or from the file), without recursion for paths denoting directories.

- Option `-1` stops at a file system boundary, that is, mount points are not crossed when creating a database.

- A list of inclusion and exclusion patterns can be defined. Each path encountered when traversing a file system subtree is compared to the list. The first matching pattern controls whether the path will be processed, or ignored. If no pattern matches, the path is processed, unless `-x` is used. The pattern can contain shell pattern matching characters, but they behave more like `-path` of find(1), that is, slash and dot characters (at a file name beginning) do not have to be matched explicitly.

  The list of patterns is composed of patterns defined by options `-i` and `-e` and patterns read from files selected by options `-I` and `-E`. The ordering of options defines the ordering of patterns in the list.

  Inclusion and exclusion patterns work recursively, that is, including/excluding a directory includes/excludes also its contents.

When a database content is displayed in textual form (by commands `list`, `diff`, and all other database comparison commands), the output is sorted by paths component-wise. As this is

not the same as the lexicographic order, two such outputs cannot compared by command comm(1) without being sorted by sort(1) first.

Commands

create Creates a new database by examining the current contents of the file system. This command requires one `-d` option.

diff Compares a database with the current contents of the file system or two databases, and writes the differences to the standard output. Each line of the output contains a single character followed by a space and the file path. The first character is '+' for added files (those existing only in the current file system or in the second database), '−' for deleted files (those existing only in the (first) database), '`!`' for modified files (different in the first database and the file system or the second database), '`d`' for files changed from non-directory to directory (directory in the second database and some other file type in the first database), '`f`' for files changed from directory to non-directory (directory in the first database and some other file type in the second database). This command requires two `-d` options.

diffequal Like `diff`, but reports (with prefix '=') also files that are equal in both databases.

added Like `diff`, but reports only added files, one path per line.

deleted Like `diff`, but reports only deleted files, one path per line.

changed Like `diff`, but reports only modified files, one path per line.

equal Like `diff`, but reports only equal files, one path per line.

dir Like `diff`, but reports only files changed from non-directory to directory, one path per line.

nondir Like `diff`, but reports only files changed from directory to non-directory, one path per line.

setmeta Creates directories that are in the database, but do not exist in the filesystem. Directories are created as empty. Sets file and directory metadata according to the database. This command requires one `-d` option.

show Outputs all information for selected files from the database in a textual format. This command requires one `-d` option.

list Outputs paths (and no other information) for selected files from the database in a textual format. This command requires one `-d` option.

Options

-1 When recursively descending a directory tree, it does not go into directories mounted from different devices. It means that file system boundaries at mount points are not crossed.

-C When comparing two databases, it checks only file contents: size, checksum (modification time is used instead of the checksum if `-c` is set), or target file (for symbolic links). File system objects that are not regular objects or symbolic links are always treated as having equal contents. By default, both contents and metadata are compared. File types are always compared and files of different types are treated as different. This option is allowed only for commands that compare two databases.

-c File checksums are not computed when used together with the `create` command and are not checked when used with the `diff` or `changed` command. If a database does not contain checksums, an attempt to use it for comparison without `-M` or `-c` set leads to an error.

-D Operate only on directories. This option cannot be used for command `create` and is mutually exclusive with option `-F`.

-d *db* The name of the created or processed database file. This option is required once or twice, depending on the command.

-E *filename* Adds patterns from file *filename* (or the standard input if *filename* is '-') to the pattern list as exclusion patterns. The file contains one pattern per line. Blank lines and leading/trailing whitespace are ignored. Lines whose first non-space character is a pound-sign (#) are comments, and are ignored.

-e *pattern* Adds *pattern* to the pattern list as an exclusion pattern.

-F Operate only on file types other than directories. This option cannot be used for command `create` and is mutually exclusive with option `-D`.

-I *filename* Adds patterns from file *filename* (or the standard input if *filename* is '-') to the pattern list as inclusion patterns. The file contains one pattern per line.

-i *pattern* Adds *pattern* to the pattern list as an inclusion pattern.

-L *filename* Adds patterns from file *filename* (or the standard input if *filename* is '-') to the pattern list. The file contains one pattern per line. Inclusion patterns are preceded by characters `"+ "` (plus sign and space). Exclusion patterns are preceded by `"- "` (minus sign and space). Blank lines and leading/trailing whitespace are ignored. Lines whose first non-space character is a pound-sign (#) are comments, and are ignored.

-M When comparing files, it checks only metadata. By default, both file contents and metadata are checked. Files types are always compared and files of different types are treated as different. This option is allowed only for commands that compare two databases.

-P *filename* Adds paths from file *filename* (or the standard input if *filename* is '-') to the path list. There is one path per line.

-q File names containing newlines are silently ignored when creating a database. If this option not set and there are some file names that contain the newline character, a warning message is reported at the end, and the program finishes with an error status.

Files that do not exist are silently ignored by command `setmeta`. If this option is not set, missing files are reported. Missing directories are always created, regardless of `-q`.

-R ***dir*** Relative *path*s are relative to directory *dir* instead of the current directory. Used by commands `create` and `setmeta`.

-r Turns off recursive descent into directories when creating a database. Only files specified explicitly in the *path* list are processed.

-S Include sockets in the created database or output. Sockets are skipped by default, because programs such as tar(1) cannot process them.

-T Check no file times. Normally, modification times (or all times with `-c`) are compared when comparing file metadata. This option skips the time test, and only the remaining metadata types are tested.

-t Check all file times. If not set, only the modification time is checked when comparing databases.

-v Verbose operation, prints each processed path to `stderr`.

-x Paths that do not match any include and exclude pattern will be ignored. Without `-x`, they would be processed.

EXIT STATUS

The **diskdb** program terminates with 0 status if successul, and with a nonzero if an error occurs.

If the requested information cannot be obtained for a file when creating the database, an error message containing the failed path is written to `stderr`, the file is ignored, and the processing continues with the next file. The final exit status will be nonzero.

ENVIRONMENT VARIABLES

TMPDIR The directory used to store temporary databases; if it is not set, `/tmp` is used.

SEE ALSO

find(1)

BUGS

Files with names containing the newline character are skipped.

NAME

fwpasswd — create and update password authentication files

SYNOPSIS

fwpasswd *passwd_file user_name*

DESCRIPTION

Program **fwpasswd** is used to create and update files, in which usernames and passwords for password authentication in Kernun firewall are stored. The program has two parameters: the password file name and the user name. The user's password is read from the standard input. If the password file does not exist, it is created. Otherwise, the file is updated. If the user name is already contained in the file, a new password is set. Otherwise, the user is added with an empty list of groups.

The return value of **fwpasswd** is 0 if the password file is successfully created or updated. If an error occurs, the password file remains unchanged and **fwpasswd** returns nonzero.

Each line of the password file contains colon-separated information about one user: user name, password (encrypted by `crypt`(3) function), and optionally a list of (comma-separated) groups the user belongs to. Program **fwpasswd** uses a temporary file named as the password file with additional suffix `.tmp`.

For existing users, **fwpasswd** modifies only the password. Any text editor can be used to rename or delete a user or to change the list of groups for a user.

RESTRICTIONS

- Usernames may not include the character '**:**'.
- The length of a password file line (username, encrypted password, and list of groups) is limited to 200 characters.
- Passwords may not include the character '`@`'. (This is because of **ftp-proxy**, in which both proxy password and server password can be given at once, delimited with the character '`@`'.)

SEE ALSO

auth(7)

NAME

grep-debug — tool for selecting messages from Kernun logs

DESCRIPTION

See **grep-debug help** for usage information.

SEE ALSO

grep-stats(1), grep-debug(1), sum-stats(1), switchlog(1), logging(7)

NAME

grep-stats — tool for selecting messages from Kernun logs

DESCRIPTION

See **grep-stats help** for usage information.

SEE ALSO

grep-stats(1), grep-debug(1), sum-stats(1), switchlog(1), logging(7)

NAME

html-match-db — controls databases of HTML form values used by the generic data matching module

SYNOPSIS

html-match-db -h { -a| -d| -s| -l } *db_file*

html-match-db -r { -a| -A| -d| -D| -s| -l| -n } *db_file*

html-match-db -r { -k| -e }

DESCRIPTION

Utility **html-match-db** is used to manage database files used by the generic data matching and processing module, see also data-matching(7). It provides functions for adding, deleting, searching, and listing records in a database file. All accesses to a database are properly locked, therefore **html-match-db** can be executed while the database file is used by a proxy.

Options

-h The provided database file is expected to be a database used by the `html-hash` and `html-alert` tests of the data matching module.

-r The provided database file is expected to be a database used by the `html-replace` test.

-a Adds a new record to the database. If used together with `-h`, a single line from the input is read, its hash computed and stored in the database. If used with `-r`, the input contains lines corresponding to the values expected in an HTML form (the original values), followed by the same number of lines with the replacement values.

-A Adds a new record to the HTML form replacement database. It expects two lines of input with the database key and encrypted value in the hexadecimal format as displayed by options `-k` and `-e`, respectively.

-d Deletes a record from the database. If used with `-h`, the value to be deleted is expected on a single input line. If used with `-r`, a sequence of input lines contains the original form values. The corresponding encrypted replacement values are deleted from the database.

-D Deletes a record from the HTML form replacement database. It expects a single input line with the database key in the format as displayed by option `-k`.

-s Searches the database for a record. Together with option `-h`, it reads a single input line containing the value to be searched for and terminates with zero exit code if the corresponding hash is found in the database, or returns a nonzero exit code otherwise. Together with `-r`, it reads a sequence of input lines containing the original form values and displays the same number of lines with the replacement values.

-l Lists all records from the database, either individual lines containing stored hashes of values for option -h, or pairs of lines containing keys and corresponding encrypted values in format as displayed by options -k and -e.

-n Reports the number of records in the database.

-k Reads a sequence of input lines containing the original form values and displays the corresponding database key obtained by hashing the input values.

-e Reads a sequence of input lines corresponding to the original values, followed by the same number of lines with the replacement values. It then displays the encrypted database value, i.e., the replacement values encrypted by a key constructed from the original values.

db file Name of the database file.

Exit Code

The program returns exit code 0 if a record has been successfully added, deleted, or found, or if options -k or -e have been specified. Exit code 1 is returned if a record cannot be added because the maximum number of records permitted by the product license has been reached, or if a record cannot be found in search and delete operations. Exit code 2 is returned if the program fails, for example, if it cannot access the specified database file.

SEE ALSO

data-matching(7), http-proxy(8)

NAME

kernun-audit — checks for bugs and new versions of the Kernun software

SYNOPSIS

kernun-audit [-butgGhsx] [-p *product*] [-v *version*] [-a *arch*] [-d *database*]

DESCRIPTION

Program **kernun-audit** reports bugs and availability of new versions of the Kernun software. It obtains information from an audit database either stored locally, or downloaded from network. This program should be run regularly in order to get timely warnings about newly discovered software bugs and announcements of available software updates. The preferred way is to execute it from cron(8) via periodic(8). The program uses information about the currently installed Kernun product and its version and selects only bugs and updates related to this version.

Options

If neither of the options -b, -u, -t, -g, and -G is specified, -bu is assumed.

-b Report bugs.

-u Report available software updates.

-U Report build numbers of available software updates, one per line. Produces empty output if there is no update.

-t Test database validity. Checking of database validity is done also with -b and -u.

-g Get full content of the audit database and display it on the standard output.

-G Display normalized content of the audit database on the standard output.

-h Display short help and exit.

-s Do not check the GPG digital signature of the audit database.

-x Display bugs and available updates in XML.

-p *product* Create report for the specified Kernun product. If not set, the product name is obtained from the currently running system.

-v *version* Create report for the specified Kernun version (given either as the full build number, or just the version number in the format used in the build number). If not set, the product name is obtained from the currently running system.

-a *arch* Create report for the specified architecture. This option must not be used if -v is set to a build number (which contains also the architecture).

-d *database* Local file name or URL of the audit database. The file name of the database digital signature is *database*`.asc`.

EXIT STATUS

The exit status is zero if the database has been successfully downloaded, has correct contents, no bug has been detected (if bug checking was selected by command line options), and no software updates are available (if update checking was selected). Otherwise, the exit status is nonzero. If XML output is selected by option `-x` or if a build number list is selected by option `-U`, the exit status is zero even if some bugs are detected or some software updates are available.

FILES

`/kernun-product` The Kernun product identifier.

`/kernun-version` The build number of the currently installed version.

SEE ALSO

cron(8), periodic(8)

NAME

license — tool for checking Kernun license file

SYNOPSIS

license [-d| -c| -p| -x]

DESCRIPTION

Command **license** checks the validity of a Kernun license file. If the file is valid, i.e., it has the right format and contains a valid digital signature, the command outputs the list of licensed components and parameters.

The license file in standard location (`/usr/local/kernun/license.dat`) is processed. An alternative license file can be selected by setting the environment variables `KERNUN_LICENSE_FILE` or `KERNUN_DIR`.

Options

-d Prints debugging messages, mainly concerning errors found in the license file.

-c Prints only the names of licensed components, one per line.

-p Prints only the names of licensed parameters, one per line.

-x Prints license content and validity in XML.

Exit Status

The command exit status is zero if the license file is valid, and nonzero if the license file cannot be read, has incorrect format or its digital signature is not valid.

FILES

/usr/local/kernun/license.dat The standard location of the Kernun license file

NAME

log-ts — tool for selecting messages from Kernun logs

DESCRIPTION

See **log-ts help** for usage information.

SEE ALSO

grep-stats(1), grep-debug(1), sum-stats(1), switchlog(1), logging(7)

NAME

mkblacklist — tool for converting http-proxy blacklists into DB format

SYNOPSIS

mkblacklist *db_file*

DESCRIPTION

The http-proxy can limit access to some servers according to a blacklist. The blacklist can exist in textual or DB formats. The proxy uses only the DB format, which provides much faster search for entries. Utility **mkblacklist** reads the blacklist in text format from the standard input and writes entries into a DB file given as an argument.

If the DB database file already exists, its content is preserved and new entries are added to it. If an entry with a key already existing in the database is to be added, a warning message is written to the standard error and the original entry is left in the DB file.

Blacklist textual format

Each line defines a single blacklist entry consisting of a server address (hostname or IP address) with an optional path and a list of categories. Individual categories are separated among themselves and from the address by whitespace. For example:

```
warez.xyz.com warez hacking
10.0.0.1/multimedia audio video
```

define two entries of a blacklist. The first one assigns categories `warez` and `hacking` to all content on server `warez.xyz.com`. The second line assigns categories `audio` and `video` to pages in subtree specified by path `/multimedia` on the server with the IP address `10.0.0.1`.

Notes

- There are no port numbers in the blacklist, because matching is always done regardless of the port the server is running on.
- Empty and comment lines (those with '#' as the first non-whitespace character) in the blacklist are ignored.
- Matching of server addresses is performed as text, i.e., hostname matches only with hostname and IP address with IP address. Utility resolveblacklist(1) can be used to automatically add all IP addresses for each host in the blacklist.

SEE ALSO

printblacklist(1), resolveblacklist(1), http-proxy(8)

NAME

monitor — report current status of Kernun proxies

SYNOPSIS

monitor [-g [-f]] [-d *dir*] [-o *file*] [-w *sec*] [-r *sec*] [-c *cols*] [-t *n* [-T *col*]] [-s *col*] [-R *type*] [application ...]

DESCRIPTION

Utility **monitor** provides a user interface to the Kernun application runtime monitoring facility. It reads the monitoring communication files created by running applications, processes the data, and generates reports. A report can be generated once or periodically, see option -r. The format of the report is either plain text or an HTML document. The report can be sorted by various criteria and contains information about all active proxy sessions or other entities, or only about a subset of them. It is possible to select applications by name (argument *proxy*) or location of communication files (-d). Another alternative is to show only top values, for example, only the 10 sessions with the largest amount of data downloaded from a server.

HTML output is intended for remote access to data monitoring. Remote access requires an HTTP server (secure and properly configured) on the firewall. If the server does not support CGI scripts or the security policy forbids them, monitor may be started as a daemon with -h, -r and other arguments that choose a fixed format of displayed information. The monitor then periodically generates a report as a static HTML page. If the monitor is used as a CGI script, it is possible to augment the report (-f) with a form that allows setting some report parameters. Option -r in both versions of HTML output instructs the user's browser to periodically refresh the page.

Arguments

-c *cols* Selects the displayed colums, cols is a comma-separated list of column types (see below).

-d *dir* Specifies a directory containing monitoring communication files. If not set, the current working directory is used by default.

-f Adds a form for selecting some parameters to HTML output, processes input from the form.

Textual output is generated if neither -g nor -f is used.

-g Generates HTML output.

-o *file* Stores the output to a file instead of sending it to the standard output.

-r *sec* Refreshes the report with given period (in seconds). HTML output will contain a Refresh header in order to automatically reload the page in a browser. If started as a CGI script, monitor always exits after the first report is generated, regardless of -r.

-R *type* Print data from records of given type (see below).

-s *col* The column type used for sorting the output. If not set, `sin` is used.

-t *n* Shows only top `n` entries with the highest values (or the oldest time).

-T *col* The column type used for `-t`. If not set, `sin` is used. Only times, byte counts, and speeds may be used here.

-w *sec* Sets timeout (in seconds) for waiting for monitor-dump. When reading a communication file, monitor-dump must sometimes wait until the proxy finishes modification of the file. This options prevents indefinitely long waiting in the case of a synchronization error.

application... Reports active data of these applications only. If no application is specified, all applications are reported.

Record Types and Column Names

The column set depends on particular record type.

SESSION record type This type is a default type and is used by proxies and the **atrmon** application.

name Proxy name, the name of `section *-proxy` in the configuration

prog Proxy type, the name of the proxy executable

pid PID of the process handling the session

start Session start time (hour:min:sec), prefixed with date if not today

time Current session duration time (hour:min:sec)

cip Client numeric IP address and port

cname Client name and port; IP to name resolution is done by proxy

sip Server numeric IP address and port

sname Server name and port; IP to name resolution is done by proxy

cout Bytes received from the client

sout Bytes sent to the server

sin Bytes received from the server

cin Bytes sent to the client

int Measurement interval (in seconds) for communication speed evaluation

cos Current speed of data receiving from the client (bytes per second)

sos Current speed of data sending to the server (bytes per second)

sis Current speed of data receiving from the server (bytes per second)

cis Current speed of data sending to the client (bytes per second)

trunc Flags indicating truncation of additional variable-length data (`user`, `auser` and `file`)

user User name as authenticated by the proxy (not the user on the final server)

auser AProxy user name authenticated by http-proxy(8)

file Name of the file that is being currently downloaded/uploaded by ftp-proxy(8) or the current request URI in http-proxy(8)

all All columns

def A default set of columns: `name,time,cname,sname,sout,sin,sos,sis`

HOSTMON record type This type is used by host monitoring applications like **atrmon** and **pikemon**.

name Application name

entity Entity for which the monitoring is relevant, i.e. a `VIRTUAL-CLUSTER` name for **pikemon** and a `REQUEST-ACL.ADDRESS` for **atrmon**

group The name of a group with ping target hosts

target_ip Particular target host to ping

err_tot Total number of unsuccessful ping attempts

sent_tot Total number of ping sent

rtt_tot An average RTT of total responses received

age1 Age of the last ping response

rtt1 Round trip time of the last ping

period Length of the last monitored period

errp Number of unsuccessful ping attempts during the last period

sentp Number of ping sent during the last period

rttp An average RTT of responses received during the last period

PIKEMON record type This type is used by cluster monitoring application **pikemon**.

name Application name

vcname The `VIRTUAL-CLUSTER` name

host_prio This host priority (`Primary` vs. `Secondary`)

host_state This host state (`Master` vs. `Backup`)

host_ready This host health status (`Up` vs. `Down`)

sensors Number of live sensors (ping groups and/or watched interfaces) and number of all sensors

peer_prio Peer host priority (`Primary` vs. `Secondary`)

peer_state Peer host state (`Master` vs. `Backup`)

peer_ready Peer host health status (`Up` vs. `Down`)

last_hello An age of the last HELO packet received from the peer

CONFIGURATION

It is possible to configure colors used by the monitor in HTML output and the description of the abbreviated column names. See instructions at the beginning of the **monitor** script.

SEE ALSO

monitoring(7), ftp-proxy(8), http-proxy(8)

NAME

ooba-acs — uses Cisco ACS log to update out of band authentication user list

SYNOPSIS

ooba-acs [-v] [-p *pidfile*] [-s] [-a *ca*] [-c *cert*] [-k *key*] *host*:*port*

DESCRIPTION

Script **ooba-acs** provides communication between Cisco ACS log and a **http-proxy**(8) acting as an out of band (OOB) authentication server. The script reads and parses the log of Cisco ACS (expected in STDIN) and passes the information about the logged users the **http-proxy**. This way, users declared to be authenticated in the Cisco ASA log are seen as authenticated by proxies that use OOB authentication.

For each Accounting log message, the appropriate update request is sent to the **http-proxy**. At most one user can be bounded to certain IP address at a time. The newer record remains.

The following accounting messages are recoginezed:

Acct-Status-Type=Start

Acct-Status-Type=Interim-Update The user is bounded to the IP address.

Acct-Status-Type=Stop The user is unbounded from the IP address.

IP address is taken from field *Framed-IP-Address*.

The user name is taken from field *User-Name*. The following special forms of file name are expected:

ANY\\USERNAME

ANY\USERNAME

ANY/USERNAME The *USERNAME* is used as User Name. The *ANY* part is ignored.

USERNAME@DOMAIN The *USERNAME* is used as User Name. The *DOMAIN* part is ignored.

UNRESPONSIVE The special User Name, that is completely ignored. No update is sent to the **http-proxy** in this case.

USERNAME If not any of the preceeding options, the username is used *as is*.

Options

-v Increases the verbosity level. Logs a message about every event sent to the **http-proxy**.

-p *pidfile* Writes process id into *pidfile*.

-s Use a secure connection (SSL/TLS) for communication with the OOB authentication server.

-a *ca* A file containing a certificate of a trusted certification authority for verification of OOB authentication server certificate

-c *cert* A file containing a certificate used for communication with the OOB authentication server

-k *key* A file containing a private key for the certificate *cert*

host Address of the OOB authentication server

port Port of the OOB authentication server

Configuration of http-proxy

The **http-proxy** must be configured as an OOB authentication server using *ext-mod* method of authentication:

- A section `aproxy` must exist, contain item `oob-auth`, and be referenced by a `session-acl`.
- The section `http-proxy` must contain item `oob-auth-srv` that references a section `oob-auth` with `method ext-mod`.
- If `oob-auth.method.ldap` is set, **http-proxy** looks for group membership information in an LDAP database.
- If `oob-auth.method.even-no-group` is set, the user is treated as being authenticated, even though ldap check for the user failed.
- It is recommended to use SSL/TLS for communication between **ooba-samba** and the OOB authentication server.

SEE ALSO

http-proxy(8), http-proxy(5), http-proxy.cfg(5), auth(7)

Samba documentation at `http://www.samba.org`

NAME

ooba-samba — uses a Samba server to update the out of band authentication user list

SYNOPSIS

ooba-samba [-d] [-p *pidfile*] [-t *sec*] [-s] [-a *ca*] [-c *cert*] [-k *key*] *host port*

DESCRIPTION

Script **ooba-samba** provides communication between a Samba server and a **http-proxy**(8) acting as an out of band (OOB) authentication server. The script reads the list of users currently logged on the Samba server and passes them to the **http-proxy**. This way, users authenticated on the Samba server are seen as authenticated by proxies that use OOB authentication.

For each user logged on the Samba server, **ooba-samba** sends to the **http-proxy** the user name, the IP address of the user's machine, and the group the user belongs to. An updated list of users is sent to the **http-proxy** each time a user logs in or out of the Samba server. Additionally, updates are sent periodically (every 5 minutes by default) in order to synchronize the list in case of a failed login/logout update.

Options

-d Prints some debugging information.

-p *pidfile* Writes process id into *pidfile*.

-t *sec* Sets the period (in seconds, the default is 5 minutes) of sending the user list to the OOB authentication server in addition to updates triggered by Samba preexec/postexec.

-s Use a secure connection (SSL/TLS) for communication with the OOB authentication server.

-a *ca* A file containing a certificate of a trusted certification authority for verification of OOB authentication server certificate

-c *cert* A file containing a certificate used for communication with the OOB authentication server

-k *key* A file containing a private key for the certificate *cert*

host Address of the OOB authentication server

port Port of the OOB authentication server

Configuration of http-proxy

The **http-proxy** must be configured as an OOB authentication server using *external* method of authentication:

- A section `aproxy` must exist, contain item `oob-auth`, and be referenced by a `session-acl`.
- The section `http-proxy` must contain item `oob-auth-srv` that references a section `oob-auth` with `method external`.
- Information about user membership in groups is also passed to **http-proxy** by **ooba-samba**. Alternatively, if `oob-auth.method.ldap` is set, **http-proxy** looks for group membership information in an LDAP database.
- It is recommended to use SSL/TLS for communication between **ooba-samba** and the OOB authentication server.

Configuration of Samba Server

- Script **ooba-samba** must be installed on the machine running the Samba server.
- The script must be configured to run all the time Samba is running. The best method is to start it from a `/etc/rc.d` or `/usr/local/etc/rc.d` script.
- The script must be configured to send the user list to the *host* and *port* where the **http-proxy** acting as an OOB authentication server listens.
- The following lines must be added to `smb4.conf` to a section defining a share, to which all users connect:

```
root preexec=kill -USR1 `cat pidfile`
root postexec=kill -USR1 `cat pidfile`
```

  where *pidfile* is a file that contains **ooba-samba** process id, as set by option `-p` of **ooba-samba**.

SEE ALSO

http-proxy(8), http-proxy(5), http-proxy.cfg(5), auth(7)

Samba documentation at `http://www.samba.org`

NAME

oobctl — tool for creating and querying file oob database

SYNOPSIS

oobctl [-rv] [-d *delim*] [-g *gdelim*] { -t *table-fn*| -f *http-proxy-cfg-fn* }

DESCRIPTION

Program **oobctl** can be used for creating, viewing and changing the OOB table.

The OOB file name is specified either by option `-t` (the file name of the OOB file) or by option `-f` (the OOB file name is taken from the http-proxy configuraiton file).

If the file does not exist yet, it can be created (option `-r`).

The program is interactive, the following commands can be used:

? [*username* | *IP*] prints the current contents of the OOB table (with particular filter)

0 clears the OOB table

+ IP user group1 group2 ... Adds/replaces a record to/in the OOB table

- { *username* | *IP* } removes the (newest) record from the OOB table

EOF the program exits

Options

-f *http-proxy-fn* The low level configuration file for http-proxy. The file name of the OOB table is taken from the configuration, or the default value is used.

-t *table-fn* The file name of the OOB table to be used

-d *delim* The field delimiter for the output

-g *gdelim* The delimitter for groups

-v print verbose information about the internals of the table to stdout

-r Recreate the table if necessary. The table is created if it does not exist. The table is recreated if the parameters from the configuration do not match the parametrs of the existing table (the following parameters are checked: *max-users*, *max-groups* and *max-sessions*).

It is only safe to create the table when using the configuration file as the file name (option `-f`).

SEE ALSO

auth(7) http-proxy(8) http-proxy(5)

NAME

printblacklist — tool for converting http-proxy blacklists into textual format

SYNOPSIS

printblacklist *db_file*

DESCRIPTION

The http-proxy(8) can limit access to some servers according to a blacklist. The blacklist can exist in textual or DB formats. The proxy uses only the DB format, which provides much faster search for entries. Utility **printblacklist** converts the content of a DB file given as an argument into text format, which is written to the standard output. Then it can be edited in any text editor and converted back to DB format by mkblacklist(1).

SEE ALSO

mkblacklist(1), resolveblacklist(1), http-proxy(8)

NAME

quarc.sh — mail quarantine control tool

SYNOPSIS

quarc.sh *command* [*options*] [*tag*]

DESCRIPTION

Utility **quarc.sh** provides a user interface to the Kernun smtp-proxy(8) quarantine management operations.

Commands

list Display short info about each selected e-mail in quarantine.

- The first line of each e-mail's block contains: MSGID, the e-mail's size, date and time of receipt and the sender.
- The second line contains the number of recipients and the first recipient.
- The third line contains *reason-tag*s describing the reasons why the e-mail has been put into the quarantine. These tags have form of *couple* `type:`*reason*, e.g. `ACL3M:virus-xy` means that the e-mail was sent to the quarantine by the mail-acl named virus-xy.
- The fourth line contains the e-mail's Subject (if any).

info Display full info about each selected e-mail in quarantine.

In fact, this info is the full content of the e-mail's quarantine control file. The lines that begin with letter '`C`' contain verbatim command lines used by the client. The lines that begin with letter '`I`' contain internal info; the next keyword at the line specifies the type of information:

RCVD Receipt date and time.

PRXY Proxy name.

QTAG Reason tag (see above).

ACL1 SESSION-ACL decision criteria and result.

HELO HELO/EHLO command argument and RFC check flags.

MAIL MAIL FROM command argument and RFC check flags, mail size and domain (7bit vs. 8bit).

HDRS Mail headers info, currently only Subject header (if present).

NODE MIME node info (number, type, size, viruses).

VIRN Virus name found in current MIME node.

ACL2 DELIVERY-ACL decision criteria and result: original recipient address and RFC check flags, new recipient address (copy-to or deliver-to), ACL name.

ACL3 MAIL-ACL name and set of DOC-ACL names.

RESP Final recipient result (response returned by proxy to RCPT command or response received by proxy from the forwarder).

send Send selected e-mails from quarantine to the smtp-proxy *proxy* given by the `-p` option.

In this case, the proxy must be specified as a *string* (not regexp) and the `-q` option must not be used.

If the proxy does not listen on "quarantine" port (see proxy-level `quarantine` directive in smtp-proxy(5) manual page), the operation fails. If the proxy listens on the proper port, e-mails from quarantine can be distinguished using the `from-quarantine` item in level 3 ACLs.

The *tag* argument can specify the content of the e-mail's header line `X-Kernun-Quarantine-Tag` that is added to the beginning of the e-mail. The value of this tag can be matched against the value of the `from-quarantine` item in level 3 ACLs.

remove Remove selected e-mails from quarantine.

Options

-q *dir* Define quarantine directory.

If used, the `-p` option can be omitted and the tool will operate on all e-mails regardless the proxy name. In this mode, the **send** operation is not allowed.

If not used, the `-p` option must define the proxy *name* (not regexp).

-p *proxy* Restrict operation only to e-mails with proxy name matching the *proxy* regexp pattern, or name (if the `-q` option is not used).

-i *msgid* Restrict operation only to e-mails with MSGID matching the *msgid* regexp pattern.

-d +*days* Restrict operation only to e-mails received earlier than *days* ago.

-d -*days* Restrict operation only to e-mails received later than *days* ago.

-R *recipient* Restrict operation only to e-mails with at least one recipient matching the *recipient* regexp pattern.

-s +*bytes* Restrict operation only to e-mails with size (in bytes) greater than or equal to the *bytes* value.

-s -*bytes* Restrict operation only to e-mails with size (in bytes) less than or equal to the *bytes* value.

-S *sender* Restrict operation only to e-mails with sender matching the *sender* regexp pattern.

-t *tag* Restrict operation only to e-mails with reason-tag matching the *tag* regexp pattern.

-v Generate more verbose output.

SEE ALSO

Kernun: mod-mail-doc(5), smtp-proxy(5), smtp-proxy(8),

NAME

resolveblacklist — tool for resolving hostnames in http-proxy blacklists

SYNOPSIS

resolveblacklist

DESCRIPTION

The http-proxy(8) can limit access to some servers according to a blacklist. The matching of server addresses is performed as text, i.e., a hostname in a request URI matches only with a hostname in the blacklist and an IP address matches only with an IP address. Utility **resolveblacklist** reads a blacklist from the standard input and writes each entry back to the output. If the server in an entry is specified by its hostname, it is resolved and entries corresponding to all IP addresses of the server are written to the output blacklist, following the original entry containing hostname.

Utility **resolveblacklist** works with blacklists in textual format. The http-proxy(8) reads blacklists in DB database format. The mkblacklist(1) and printblacklist(1) utilities perform conversions between textual and DB formats of blacklists.

SEE ALSO

mkblacklist(1), printblacklist(1), http-proxy(8)

NAME

rrd — system parameter watching

SYNOPSIS

rrd [-v] { time| class| list| values| create| update| monitor| proxy| lsgraph }

rrd [-v] graph *class.instance.graph.time* ...

DESCRIPTION

Utility **rrd** provides a command line interface to the online system parameter watching facilities of the Kernun Firewall. The history of values of various parameters is stored in a RRD (Round-Robin Database) and can be displayed in the form of graphs on demand.

Program **rrd** categorizes available parameters into named *classes* (for example, `netif` for network interface parameters, `proxy` for proxies, or `sys` for system). Each class can have several *instances* (individual network interfaces for class `netif`, proxies for class `proxy`, or single instance named by hostname for class `sys`. In order to be able to watch proxy parameters, monitoring must be enabled for the proxy, see monitoring(7).

For each instance, one or more parameters can be watched and stored in an RRD file. Graphical presentation of data can be generated in various time scales, for example, last day or month.

Options

-v Verbose output for debugging and help

time Lists identifiers of available time scales of graphs. The identifiers are usable in `graph` command. If called with `-v`, a short description of each time scale is displayed.

class Lists all classes of parameters that can be watched. The identifiers are usable in `graph` command. If called with `-v`, a short description of each parameter is displayed.

list Lists all existing instances of all parameter classes. Each line is prefixed by two characters, which are either '-', '`A`' (the instance is active, that is, currently existing in the system), or '`D` (there is a RRD file for the instance). RRD files for inactive instances are not removed automatically, which makes it possible to display graphs for removed proxies or network interfaces.

values Displays the current values of all watched parameters.

create Creates RRD files for all active instances.

update Obtains the current values of all watched parameters and stores them in the RRD. This should be called periodically, for example, once a minute by **cron**.

monitor Updates the RRD continuously. Periodically calls `update`, waits for some time (1 minute by default) and updates again.

proxy Deletes monitoring files of terminated proxies and stores data from them in `/data/rrd/proxy.sum`. This should be called periodically to delete obsolete monitoring files if periodic `update` is not performed and monitoring is switched on in at least one proxy.

lsgraph Lists available graph types. If called with `-v`, a short description of each graph is displayed.

graph Generates graphs. Each argument following `graph` defines one graph to generate by specifying data class, instance, graph type, and graph time scale (delimited by periods).

FILES

/data/rrd Directory used to store RRD database files and generated graphs

. The current directory is used if `/data/rrd` does not exist.

/data/rrd/*.rrd A RRD database file for a single instance

/data/rrd/*.png A graph generated from RRD data

/data/rrd/proxy.sum Values of parameters taken from monitoring files of terminated proxies

BUGS

RRD database files are not portable between architectures. If you migrate a Kernun system from i386 to amd64 or vice versa, you cannot copy your RRD files. If you do, they will not be updated and graphs will not be generated from them. It is possible to move a RRD file from one architecture to another in a portable XML format. Export the binary data to XML by command **rrdtool dump** on the original architecture, then import XML data into the binary format by command **rrdtool restore** on the target architecture.

SEE ALSO

monitoring(7)

NAME

sum-stats — generates proxy usage statistics from Kernun logs

SYNOPSIS

sum-stats [-p *period*] [-t *type*] [-n *name*] [-l *field=limit ...*] [-filter *field=filter ...*] [-spam-threshold *value*] [-shift *time_offset*] [-start *time_spec ...*] [-finish *time_spec*] [-entitle *label*] [-info *list*] [-db] -o *outfile*

DESCRIPTION

The **sum-stats** script reads a Kernun log from the standard input and generates proxy usage statistics. The exact contents of the output depend on the proxy type. However, the generated output always retains the following structure:

- Summary: totals + Kernun Clear Web database hit-rate (for `http-proxy` and `icap-server`)
- Histograms: per-hour, per-day, per-weekday (depends on *period*)
- Hitparades: per-client, per-server, ... (depends on *type*)

Options

-p *period* Sets the *period* (`daily`, `weekly`, `monthly`). Log items outside the date interval based on this period are filtered out.

Use `-shift` for specifing which period to be generated. The current period (day/week/month) is generated by default. For example, use `-p weekly -shift -1w` for generating the statistics for the last week.

-t *type* Sets the *type* of the proxy. If not set, the default value is `proxy` (does not assume any particular proxy type). A list of recognized proxy types can be found below.

-n *name* Sets the *name* of the proxy (altname) to be included in the statistics (other proxies are filtered out). If not set, all proxies are included.

-l *field=limit* Sets the *limit* for the given *field* (top N clients, servers, ...). If not set, the field is excluded from the statistics.

The special value 0 means not to limit this field at all, All the values are included in the statistics, regardless of their total count. Note that using field limit 0 can result in a VERY BIG statistics that can lead to problems when viewing them.

A list of available fields can be found below.

-filter *field=limit* Sets the *filter* for the given *field* (clients, servers, ...). If set, only the log records that match the filter are taken into account. If set, the statistics for the field that is being filtered are supressed, since it would be degenerate.

-spam-threshold *value* Sets the *spam-threshold*; mails with spam score above this level are considered SPAM. If not set, the default value is `5000`.

-shift *time_offset* Behaves as if the processing day was executed earlier/later, given by `time_offset`. The form of the `time_offset` is *[<SIGN>]<COUNT>[<UNIT>][<ROUND>]*

- *SIGN*: '-' for shift to the history, + for shift to the future. Defaults to '+'
- *COUNT*: the number of days/weeks/months. Can be 0 for no shift, which can be useful in conjunction with *ROUND*.
- *UNIT*: 'h' for hours, 'd' for days, 'w' for weeks, 'm' for months.

  If ommited, *UNIT* default depends on the period selected by `-period`: 'm' for monthly period, 'w' for weekly period and 'd' for daily period. If no period is selected, 'd' is used as the default value for UNIT.
- *ROUND*: if given, the result is rounded up or down within the given unit. Use 'up' for round up, 'down' for round down.

For example, `-shift -2w_up` shifts two weeks back, to the Sunday 23:59:59. The option can be given more than once in which case the time in sequence shifted more times.

See also environmental variable `TIME` Setting the environmental variable `TIME` has the similar effect as using `-shift`. The time is given as the system time when the script is executed by default. This can be overriden by the `TIME` environmental variable. The resulting value is then used as the base for the `-shift` options.

-start *time_spec*, -finish *time_spec* Explicitly sets the time interval to be used. The *timespec* is one of the following:

- *iso timestamp*: one of *YYYY-MM-DDTHH:MM:SS*, *YYYY-MM-DDTHH:MM*, *YYYY-MM-DDTHH*, *YYYY-MM-DD*
- *unix timestamp*: the number of seconds since 1970
- *time offset*: time is given as an offset to the current time (possibly affected by option `shift`

Options `-start` and `-finish` are mutually exclusive with option `period`, which sets the interval implicitly.

-info *list* Instead of creating the statistics, reports some information, given as a comma separated list of desired info:

- `fields`: print the fields valid for the particular type
- `types`: print the available types

- `results`: print the available results
- `interval`: print the time interval that would be used
- `log_files`: list the filenames that likely contain the desired time interval without the eventual compression suffix.
- `log_files`: list the filenames that likely contain the desired time interval.
- `log_files_ts`: print the shell script that cats the files that likely contain the desired time interval.
- `period_inst_name`: period instance name. Prints the suggested name of the periodic statistics, if generated with the current arguments. Based on the beginning of the interval, it is used 'YYYYMM' for monthly, 'YYYYWW' for weekly and 'YYYYMMDD' for daily statistics.
- `oldest_log`: print the timestamp of the oldest line in the available logs.

-db If present, the newly created statistics is also indexed in the statistics index database.

-o *outfile* The output will be saved to *outfile*.html, accompanied by its data file *outfile*.json.

Proxy Types

proxy Fields: `client`, `user`, `server`

http-proxy, icap-server Fields: `client`, `user`, `group`, `server`, `category`

smtp-proxy Fields: `client`, `server`, `sender`, `recipient`, `mime`

dns-proxy Fields: `client`, `server`, `qname`, `qtype`

sip-proxy Fields: `caller`, `receiver`

ENVIRONMENT VARIABLES

TIME The timestamp used to calculate the interval of dates to be included in the statistics (affected by the period, shift). If not set, the current time is used.

NOTES

Computing per-client, per-server, ... statistics (hitparades) can consume a large amount of memory. Memory usage can only be reduced by turning off individual fields (skip `-l` *`field`*, or set `-l` *`field`*`=0`). Mere setting the number of top values reported does not reduce memory consumption.

SEE ALSO

log-ts(1), switchlog(1), logging(7)

NAME

switchlog — distribute messages from Kernun log according to message id and proxy name

SYNOPSIS

switchlog [{-v|-V} [*lines*]] *cfgfile* [*logfile*]

DESCRIPTION

Program **switchlog** reads a log file in the Kernun log format from the standard input or from a file specified by the `logfile` parameter. Individual messages are then written to different output files or sent to other programs through pipes, according to the configuration file *cfgfile*.

Options

{-v|-V} *[lines]* When reading log, report how many lines have been processed (`-v` prints a sequence of messages, `-V` rewrites the same message using backspace characters). Parameter *lines* is the number of lines, after which a message will be generated (if not present, print a message after each 10000 lines).

cfgfile Name of the configuration file.

logfile Log file to be processed. If not present, the standard input is read.

Syntax of Configuration File

Empty lines and lines beginning with '#' are ignored.

out *out_id* [!]>*file* Messages written to output id *out_id* will be appended to *file*. If the "!>" form is used, the output will be flushed after each line.

out *out_id* [!]|*program* *[args...]* Messages written to output id *out_id* will be processed by *program*. The program will be run only once and messages will be passed to its standard input via a pipe. If the "!|" form is used, the output will be flushed after each line.

log_id altname out_id Messages with matching log id and proxy name will be sent to output id *out_id*. Value of *log_id* can be either log id (e.g., `TCPP-202`), or the word `unknown` (matching messages with unknown — not present in switchlog's hash table — log id), or `default`. Value of *name* is either a proxy name, or '*' meaning "any".

Matching

The log id and altname are extracted from each message read from the input file. The log id is then used for lookup in a hash table compiled into **switchlog**. If the id is found in the table, the configuration lines with corresponding *log_id* are searched for matching proxy name (which can also be '*', meaning "any"). If the id is not found, lines beginning with `unknown` are searched for

altname. If no log id, proxy name pair has been found so far, lines beginning with `default` are searched for proxy name. Finally, if a matching configuration line is found, the message is sent to the output id specified in that line. Otherwise, the message is ignored.

NOTES

Processing logs by scripts, such as sum-stats(1) is a time-consuming task. Moreover, if statistics of several proxies are to be computed, summarization scripts would read the same log file again and again. Program **switchlog** can reduce the log processing time by quickly selecting only the messages relevant for further processing. Also, the log file is read only once even if further processing is performed by several scripts.

SEE ALSO

log-ts(1), sum-stats(1), logging(7)

NAME

triplicator — SMTP Grey-listing Triplet Database Manipulator

SYNOPSIS

triplicator [-hv] [-d *debuglev*] -f *cfgfile* -c *command*

DESCRIPTION

Utility **triplicator** provides a user interface to Kernun's smtp-proxy(8) grey-listing triplet database management operations.

The main task of this tool is to clean the database. For this purpose, it is recommended to include the following call of **triplicator** in a **cron** plan:

```
system FIREWALL {
  crontab {
    ...
    plan "0 * * * * kernun path/triplicator -f smtp-cfg -c clean"
  }
}
```

where *smtp-cfg* is the configuration file name of **smtp-proxy** that uses the grey-listing method and *path* is the path to the Kernun binaries directory (usually `/usr/local/kernun/bin`).

Grey-listing method

Grey-listing (`http://projects.puremagic.com/greylisting`) is a spam blocking method based on the fact that most spam sources do not behave in the same way as "normal" mail systems and do not repeat delivery attempt in the case of temporary rejection. Thus, `smtp-proxy` with grey-listing configured temporarily rejects every new e-mail it has never seen, keeps this information and, under certain conditions, allows reception of this mail in the future. More precisely speaking: the proxy looks at three pieces of information (called *triplet*) for any particular mail delivery attempt:

- The IP address of the host attempting the delivery
- The envelope sender address
- The envelope recipient address

Each triplet can be in one of the following states:

blocked If we have never seen this triplet before, then refuse this delivery and any others that may come within a certain period of time with a temporary failure.

released If the triplet has passed the initial blocking period, we expect repeated delivery within a certain period of time. Any e-mail with the identical triplet coming within this time period will change the triplet's state to *granted*. If no such delivery occurs, the triplet is forgotten.

granted If a triplet has been successfully acknowledged, any mail with the identical triplet will be delivered without delay for a certain period of time. Every new delivery attempt for the triplet will restart this time period. If the time period passes without any delivery attempt, the triplet is forgotten.

Proper functionality of the method can be set up using three basic parameters that control the above-mentioned time periods:

`block-time` The initial delay of a previously unseen triplet.

Default: 1 Hour

For this period of time, a new triplet is in state *blocked*.

`retry-time` The lifetime of triplets that have not yet allowed an e-mail to pass and wait for confirmation by another delivery attempt.

Default: 4 Hours

The total time the client has to retry the delivery attempt. If this period of time elapses and no mail with the particular triplet has come, the triplet is forgotten. Notice that this time includes also the initial `block-time` amount of time. Thus, in fact, the length of the time period, during which a triplet is in state *released*, is `retry-time` - `block-time`.

`guard-time` The lifetime of auto-whitelisted triplets that allow mail to pass.

Default: 36 Days

For this time (after any successful delivery), the triplet is guarded and mails with particular triplet are *granted* to pass.

All the above parameters are part of the `grey-listing` section of the `smtp-proxy` section.

The proxy saves the information about triplet states in a local database in a file (its name must be set in `smtp-proxy.grey-listing`). For every triplet, it holds the state and the time of its expiration. Using the `triplicator` tool, you can manage this database - clean it (i.e. remove forgotten triplets), display and change triplet data.

Commands

`stat` Display the number of triplets in the database per states.

`list` Display all triplets in the database, each with its state and the time of the expiration of the state.

Example:

```
<10.1.1.1, , root@tns.cz> 1=ts_blocked, 2000/02/04 01:00:00, 10800
```

clean Remove all expired triplets.

add Add/change triplet data.

Command synopsis: `add` *`state date time <sender> <recipient>`*

state Triplet state, possible values: 'b+*next*', 'r' and 'g'.

The *next* parameter for the "blocked" state defines the time period (in seconds), for which the triplet will stay in the "released" state after reaching the deadline of the "blocked" state. In fact, it is the difference between `retry-time` and `block-time`.

date Expiration date, format: *year*/*month*/*day*

time Expiration time, format: *hh*:*mm*:*ss*

Example:

```
add b+3600 2000/2/4 01:00:00 10.1.1.1 <> <root@tns.cz>
```

purge Rebuild the database file (implemented as backup + restore).

backup Dump non-expired triplets.

Command synopsis: `backup [`*`filename`*`]`

filename Output file name (stdout if omitted).

restore Rebuild database from a backup file.

Command synopsis: `restore [`*`filename`*`]`

filename Input file name (stdin if omitted).

Program options

-h Display usage information and exit.

-v Print version information and exit.

-d *dbglev* Set debugging level.

-f *cfgfile* Read configuration from *cfgfile*.

-c *command* Command to execute.

SEE ALSO

Kernun: smtp-proxy(5), smtp-proxy(8),

Appendix B

NAME

acl — format of acl component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **acl** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **acl** configuration directives:

direction (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

week-day (see **time**(5))

month (see **time**(5))

auth-method (see **auth**(5))

virus-status (see **antivirus**(5))

source-address-mode (see **source-address**(5))

transparency (name-usage obligatory)

Transparency mode.

non-transparent

transparent

user-auth-spec (name-usage obligatory)

Firewall user authentication mode.

none

No user authentication is presented or required.

name

Authorized users can be specified in the configuration.

doctype-ident-method (name-usage obligatory)

Methods for document type recognition.

content-type

MIME type defined in Content-Type header is used.

extension

Type is derived from URI/filename suffix.

magic

Type is recognized according to real content.

header-op (name-usage obligatory)

Header modification operation

delete

Header(s) will be removed.

add

New header will be added. If header is not repeatable, old occurence will be removed.

replace

Header(s) will be removed and new one will be added.

ITEMS AND SECTIONS

Configuration of **acl** library component consists of following prototypes:
* user ... ;
  plug-to ... ;
  hand-off ... ;
* doctype-ident-order ... ;
* acl *name* { ... }
* acl-1 *name* { ... }
* acl-2 *name* { ... }
* acl-3 *name* { ... }

Description:

user none;

user [name] [*name* **[group** *group***]];**

User and group specification.

<**branching element**> **(type: user-auth-spec, optional, default: name)**
***name* (type: str-set, optional, default: *)**

user name (authenticated on firewall)

group *group* **(type: str-set, optional, default: *)**

list of groups, if present, both NAME and GROUP must match

plug-to *addr*;

Final destination server.

addr **(type: sock)**

Address/port of final destination server.

If port is zero, then original port is used.

hand-off *addr*;

Next-hop proxy.

addr **(type: sock)**

doctype-ident-order [for *for***]** *order*;

Order of document type recognition methods.

This item defines order in which different methods of document type recognition methods are used. Item can be defined at several places - globally for the proxy and in some ACLs. The most specific occurence is used, if no specification is found, just CONTENT-TYPE method is used.

for *for* **(type: direction-set, optional, default: *)**

Document transfer direction set.

This element defines directions for which the order is specified by this item.

For some proxies, both directions can be used while for others either direction is not applicable; consult proxy man page.

order **(type: doctype-ident-method-list)**

Methods are used in given order unless type is recognized.

For some proxies, some methods are not applicable, consult proxy man page.

Constraints:

Only 3 methods can be specified.

acl *name* **{**

* from ... ;

* to ... ;

* server ... ;

* user ... ;

* time ... ;

time-period-set { ... }

* parent-acl ... ;

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

}

Access Control List.

General form of specification of firewall services limitation. In each proxy configurations, this general model is adapted and renamed, even more than once - if it needs different approach for different phases of its work. For this purpose, more precized prototypes ACL-1 and ACL-2 are derived from this general prototype.

In general, ACL consists of several categories of limitations (or entry conditions) controling which connections or operations will be handled according to particular ACL. Then, ACL defines wheter connections or operations being handled by this ACL will be accepted or denied by the proxy (items ACCEPT and DENY). Finally, ACL defines details of protocol behavior.

This prototype defines basic entry conditions applicable to most of proxies. Each proxy can exclude some features from this general concept and add several proxy specific limitations - see proxy configuration man page to check these changes.

General entry conditions:

- FROM (connection/request source IP address/name)
- TO (connection/request destination IP address/name)
- SERVER (logical target server's IP address/name)
- USER (proxy-authenticated user name)
- TIME or TIME-PERIOD-SET (actual time)
- PARENT-ACL (name of acl used in previous phase)

Each category can be used more than once in one ACL; then they are checked in disjunction (OR). If omitted, category is not checked in particular ACL.

Different categories are checked in conjunction (AND).

Constraints:

Exactly one of DENY and ACCEPT must be specified.

TIME and TIME-PERIOD-SET are mutually exclusive.

Items & subsections:

`from` *addr*;

Entry condition - client (source) address.

***addr* (type: `host-set`)**

Set of client IP addresses or hostnames.

Constraints:

Regexps are not allowed in host set.

to *mode destinations* [**port** *port*];

Entry condition - physical destination address.

This item is used to match the TCP connection or UDP request destination address. In the transparent case, this is actual server targeted by the client, while otherwise this is an address and port at the firewall where particular proxy is bound.

mode (type: **transparency**)

Select mode allowed for connections/requests.

destinations (type: **host-set**)

Set of destination adresses/names.

In transparent case, destination is equal to target server. In non-transparent case, destination is equal to proxy address/port.

port *port* (type: **port-set**, optional, default: *)

Set of destination service names/port numbers.

Constraints:

Regexps are not allowed in host set.

server [*addr* [**port** *port*]];

Entry condition - logical target server.

This item is used to match the logical target of service requested by the client. In the transparent case, it should be equal to the physical destination server. However, in many proxies the server address or name is known to the proxy only when it recognizes an initial protocol command sequence.

addr (type: **host-set**, optional, default: *)

Set of logical target IP addresses or hostnames.

port *port* (type: **port-set**, optional, default: *)

Set of logical target service names/port numbers.

user none;

user [**name**] [*name* [**group** *group*]];

Entry condition - proxy-user name.

<branching element> (type: **user-auth-spec**, optional, default: name)

name (type: **str-set**, optional, default: *)

user name (authenticated on firewall)

group *group* (type: **str-set**, optional, default: *)

list of groups, if present, both NAME and GROUP must match

time [**day** *day*] [**month** *month*] [*wday* [*hhmm*]];

Entry condition - date/time.

day *day* (type: **uint8-set**, optional, default: *)

day of month (1 - 31)

month *month* **(type: month-set, optional, default: *)**

month (Jan - Dec or 1 - 12)

wday **(type: week-day-set, optional, default: *)**

week-day (Sun - Sat or 0 - 6)

hhmm **(type: time-set, optional, default: *)**

time (in form hhmm)

time-period-set {

exclude ... ;

* time-spec *name* { ... }

}

Entry condition - date/time.

The time-period-set section is derived from time-period-set section prototype. For detail description of it, see time(5).

parent-acl *name*;

Entry condition - parent ACL name.

This item is used only for proxies with multi-phase ACL, name of this item is changed to real name of previous phase ACL. See proxy man page for details.

name **(type: str-set)**

(name of ACL used in the previous phase)

deny;

This item is obligatory if particular ACL is to deny connections or operations satisfying entry conditions. Specific proxy ACLs derived from this prototype add some details of denial procedure depending on protocol.

accept;

This item is obligatory if particular ACL is to accept connections or operations satisfying entry conditions. Specific proxy ACLs derived from this prototype add some details of futher behavior depending on protocol.

doctype-ident-order [for *for***]** *order*;

Order of document type recognition methods.

This item defines order in which different methods of document type recognition methods are used. Item can be defined at several places - globally for the proxy and in some ACLs. The most specific occurence is used, if no specification is found, just CONTENT-TYPE method is used.

for *for* **(type: direction-set, optional, default: *)**

Document transfer direction set.

This element defines directions for which the order is specified by this item.

For some proxies, both directions can be used while for others either direction is not applicable; consult proxy man page.

order **(type: doctype-ident-method-list)**

Methods are used in given order unless type is recognized.

For some proxies, some methods are not applicable, consult proxy man page.

Constraints:

Only 3 methods can be specified.

`rule` *rule*;

The identifier of the high-level rule which is implemented by this acl.

rule **(type: `str`)**

The rule identifier

[End of section `acl` description.]

`acl-1` *name* {

* from ... ;

* to ... ;

* user ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

auth ... ;

idle-timeout ... ;

idle-timeout-peer ... ;

source-address ... ;

plug-to ... ;

}

Access Control List, Phase 1.

This prototype is derived from the general ACL by excluding some attributes not used in initial phase of proxy operation. Besides, several general phase 1 features are added:

- AUTH (authentication mode used)
- IDLE-TIMEOUT (forcing of session idle-timeout)
- SOURCE-ADDRESS (forcing of source address towards server)
- PLUG-TO (forcing of destination server)

The `acl-1` section is derived from `acl` section prototype. For detail description of it, see above.

Changes to the `acl-1` section:

Item `server` is not valid.

Item `parent-acl` is not valid.

DENY and AUTH are mutually exclusive.

DENY and IDLE-TIMEOUT are mutually exclusive.

DENY and SOURCE-ADDRESS are mutually exclusive.

DENY and PLUG-TO are mutually exclusive.

Added items & subsections:

`auth none`;

`auth passwd` *file*;

`auth radius` *client*;

`auth ldap` *ldap*;

`auth ext` *file*;

`auth oob` *oob* [*mode* [**`loose`**]];

Authentication method and attributes specification.

For more details, see auth(7).

<**branching element**> (**type: `auth-method`**)

file (**type: `str`**)

Password/utility file name.

client (**type: `name` of `radius-client`**, see radius(5))

RADIUS client configuration name.

ldap (**type: `name` of `ldap-client-auth`**, see ldap(5))

LDAP client configuration parameters.

oob (**type: `name` of `oob-auth`**, see auth(5))

OOB authentication parameters.

mode (**type: `obligation`, optional, default: required**)

`loose` (**type: `key`, optional**)

`idle-timeout` [*seconds*];

Session inactivity timeout.

If no data is transmitted for this session in the period of idle-timeout seconds, the connection is closed.

If omitted, value of proxy.idle-timeout is used.

seconds (**type: `uint31`, optional, default: 0**)

Timeout for datagrams in any direction (any packet resets the timer), zero means unlimited.

`idle-timeout-peer` [*client* [*server*]];

Peer inactivity timeout.

If no data is transmitted by peer in the period of idle-timeout seconds, the connection is closed.

If omitted, the inactivity is controlled by the idle-timeout item.

client **(type: uint31, optional, default: 0)**

Timeout for datagrams from client to server (in seconds, zero means unlimited).

server **(type: uint31, optional, default: 0)**

Timeout for datagrams from server to client (in seconds, zero means unlimited).

source-address [client] [addr4 *addr4***] [addr6** *addr6***] cluster [***cluster***];**
source-address [client] [addr4 *addr4***] [addr6** *addr6***] [physical];**
source-address [client] [addr4 *addr4***] [addr6** *addr6***] no-fallback;**

Source address for outgoing connections to servers.

If omitted, the proper address of the proxy will be used, i.e. in the case of a cluster, the cluster address will be used.

If not specified by the SOURCE-PORT item, a generic port will be used.

The elements entered within this item will be used by the proxy until the first of them is applicable:

- The CLIENT keyword means the original client IP address is used. This mode will be succesful in all cases except mismatch of IP address families.

- The ADDR4/ADDR6 keyword-value pairs mean that the specified address is used for a connection of corresponding address family.

- The CLUSTER keyword means that one of cluster addresses will be used. By default, the main address of the bridge is used, however, any preferred alias address can be listed in the cluster list.- The PHYSICAL option means that the address of the physical interface is used instead of the cluster one.

- The DEFAULT option means the default behavior - i.e. using of the physical address.

- The NO-FALLBACK option means that if no other way of setting the address is acceptable, the session is rejected. Without this option, the system tries to find a suitable source IP address automatically.

client (type: key, optional)

addr4 *addr4* **(type: host, optional, default: [0.0.0.0])**

addr6 *addr6* **(type: host, optional, default: [::])**
<branching element> (type: source-address-mode, optional, default: physical)
cluster **(type: host-list, optional, default: {})**

Constraints:

Address family must respect the element's address family..

plug-to *addr*;

Final destination server.

addr **(type: sock)**

Address/port of final destination server.

If port is zero, then original port is used.

[End of section `acl-1` description.]

acl-2 *name* **{**

* from ... ;

* server ... ;

* user ... ;

* time ... ;

time-period-set { ... }

* parent-acl ... ;

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

}

Access Control List, Phase 2.

This prototype is derived from the general ACL by excluding some attributes not used for proxy command control.

The `acl-2` section is derived from `acl` section prototype. For detail description of it, see above.
Changes to the `acl-2` section:

Item `to` is not valid.

acl-3 *name* **{**

* from ... ;

* server ... ;

* user ... ;

* time ... ;

time-period-set { ... }

* parent-acl ... ;

deny ... ;

accept ... ;

rule ... ;

direction ... ;

* size ... ;

* content-type ... ;

* mime-type ... ;

virus-status ... ;

* modify-header ... ;

force-doctype-ident ... ;

replace ... ;

html-filter ... ;

}

Access Control List, Phase 3.

This prototype is derived from the general ACL by excluding some attributes not used in document-processing phase of proxy operation. Besides, several general phase 3 features are added:

- DIRECTION (entry condition - document transfer direction)
- SIZE (entry condition - document size)
- MIME-TYPE (entry condition - document type)
- VIRUS-STATUS (entry condition - antivirus check result)
- REPLACE (accepting action - replacing document by file)
- HTML-FILTER (accepting action - filtering document)

The `acl-3` section is derived from `acl` section prototype. For detail description of it, see above.
Changes to the `acl-3` section:

Item `to` is not valid.

Item `doctype-ident-order` is not valid.

Filtration items (REPLACE, HTML-FILTER, FORCE-DOCTYPE-IDENT) are not allowed if DENY is on.

Items HTML-FILTER/FORCE-DOCTYPE-IDENT and REPLACE are mutually exclusive.

Added items & subsections:

`direction` [*dir*];

Entry condition - document transfer direction.

dir **(type: `direction-set`, optional, default: *)**

`size unknown`;

`size lt` *limit*;

`size le` *limit*;

`size eq` *limit*;

`size ne` *limit*;

`size gt` *limit*;

`size ge` *limit*;

`size in` *lower upper*;

size ni *lower upper*;

Entry condition - document size.

<**branching element**> (**type: range-op**)

limit (**type: uint64**)

Tested value limitation.

lower (**type: uint64**)

Tested value lower bound.

upper (**type: uint64**)

Tested value upper bound.

Constraints:

Value UNKNOWN is not allowed.

content-type *type*;

Entry condition - original Content-Type.

type (**type: str-set**)

Set of type/subtype string definition.

If a regexp is part of the set, then this regexp is checked to match with type/subtype specification. Beware of escaping the slash, if present (write /...\/.../).

If a string is part of the set, then it must contain at most one slash. If the slash is not present, string is compared with document type only (not the subtype). If the slash is present, then pattern is checked to match with type/subtype specification.

mime-type *type*;

Entry condition - recognized MIME type.

type (**type: str-set**)

Set of type/subtype string definition.

If a regexp is part of the set, then this regexp is checked to match with type/subtype specification. Beware of escaping the slash, if present (write /...\/.../).

If a string is part of the set, then it must contain at most one slash. If the slash is not present, string is compared with document type only (not the subtype). If the slash is present, then pattern is checked to match with type/subtype specification.

virus-status [*status*];

Entry condition - virus detection status.

status (**type: virus-status-set**, **optional**, **default: ***)

modify-header delete *names*;

modify-header add *name text*;

modify-header replace *name text*;

Document headers modified.

<**branching element**> (**type: header-op**)

Action to be done with header(s).

names (type: **str-set**)
Names of headers to be deleted.

name (type: **str**)
Name of header(s) to be modified or added.

text (type: **str**)
New header text.

Constraints:
Header modification text must comply with RFC.

force-doctype-ident [*apply*];
Change Content-Type header to recognized one.

This item forces change of original Content-Type header in the document to the new one recognized by means of the DOCTYPE-IDENTIFICATION tool.

apply (type: **str-set**, optional, default: *)
Set of original Content-Type values that will be changed by this item. Values NOT MATCHING tis set will be preserved.

replace *filename* [*mime-type*];
Document is removed and replaced by new text.

filename (type: **name** of **shared-file**, see common(5))
Replacement file.

mime-type (type: **str**, optional, default: "text/plain")
Document Content-Type.

Constraints:
MIME type of replacement must comply with RFC.

html-filter *htmlf*;
Document is filtered.

htmlf (type: **name** of **html-filter**, see mod-html-filter(5))

[End of section `acl-3` description.]

SEE ALSO

configuration(7), antivirus(5), auth(5), common(5), ldap(5), mod-html-filter(5), radius(5), source-address(5), time(5), access-control(7), auth(7), host-matching(7), time-matching(7)

NAME

adaptive-firewall — format of adaptive-firewall component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **adaptive-firewall** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **adaptive-firewall** configuration directives:

yes-no (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

report-mode (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

task-frequency (see **common**(5))

listen-on-sock (see **listen-on**(5))

ids-agent-log-level (**name-usage obligatory**)

IDS log level

none

No messages are logged

emergency

Only emergency messages are logged

alert

Alert messages and above are logged

critical

Critical messages and above are logged

error

Error messages and above are logged

warning

Warning messages and above are logged

notice

Notice messages and above are logged

info

Informational messages and above are logged

perf

Performance messages and above are logged

config

Configuration messages and above are logged

debug

All possible messages are logged

ids-agent-detection-direction (name-usage obligatory)

Which address to detect as suspicious

src

Report source address

dst

Report destination address

both

Report both source and destination addresses

ids-agent-protocol (name-usage obligatory)

IDS protocols to inspect

any

Scan any protocol

tcp

Scan TCP protocol

udp

Scan UDP protocol

dcerpc

Scan DCERPC protocol

dhcp

Scan DHCP protocol

dns

Scan DNS protocol

ftp

Scan FTP protocol

http

Scan HTTP protocol

icmp

Scan ICMP protocol

ikev2

Scan IKEV2 protocol

imap

Scan IMAP protocol

krb5

Scan KRB5 protocol

msn

Scan MSN protocol

nfs

Scan NFS protocol

ntp

Scan NTP protocol

smtp

Scan SMTP protocol

ssh

Scan SSH protocol

tls

Scan TLS protocol

ids-agent-rule-action (name-usage obligatory)

IPS rule action

alert

Generate an alert but do not block the traffic.

The alerts are sent to Kernun base station for further analysis.

pass

Ignore the packet

block

Generate an alert and block the traffic.

IPS mode needs to be enabled by specifying section ADAPTIVE-FIREWALL.IPS, otherwise this action behaves like ALERT.

Note that it may take up to a minute for the traffic to be blocked.

ids-agent-threshold-type (name-usage obligatory)

IDS rules threshold type

threshold

Sets a minimum threshold for a rule before it generates an alert.

A threshold setting of COUNT means on the COUNT-th time the rule matches an alert is generated.

limit

If set to limit COUNT, it alerts at most COUNT times.

both

Applies both thresholding and limiting.

ids-agent-threshold-track-by (name-usage obligatory)

IDS rules threshold track by

src

Track the policy rule by source.

dst

Track the policy rule by destination.

ids-agent-rate-filter-track-by (name-usage obligatory)

IDS rules filter track by

src

Track the policy rule by source.

The tracking is done per IP-address.

dst

Track the policy rule by destination.

The tracking is done per IP-address.

rule

Track the policy rule globally for the rule.

both

Track the policy rule by a pair of source and destination.

The tracking is done per IP-address. Packets going to opposite directions between same addresses tracked as the same pair.

ids-agent-suppress-direction (name-usage obligatory)

IDS rules suppress direction

src

Suppress the IDS rule for given source addresses.

dst

Suppress the IDS rule for given destination addresses.

any

Suppress the IDS rule for given addresses (source or destination).

policy-level (name-usage obligatory)

Adaptive Database Record Levels

medium

high

highest

ITEMS AND SECTIONS

Configuration of **adaptive-firewall** library component consists of following prototypes:
* ids-watchdog *name* { ... }
  ids-agent-from-to ... ;
* ids-agent-rule-def *name* { ... }
* ids-agent-base-rule-policy *name* { ... }
* ids-agent-base-rate-filter *name* { ... }
* ids-agent-base-threshold *name* { ... }
* ids-agent-base-suppress *name* { ... }
  ids-agent-base-change-rule ... ;
  ids-agent { ... }
  adaptive-firewall { ... }

Description:

ids-watchdog *name* **{**

id ... ;

file ... ;

* pattern ... ;

* threshold ... ;

record-lifetime ... ;

blocking ... ;

max-entries ... ;

}

A detector that monitors files for patterns.

It watches lines being added to given file and searches for given patterns.

Constraints:

Watchdog identification must be specified.

At least one pattern must be specified.

Items & subsections:

`id` *key*;

Watchdog Identification.

key **(type: `str`)**

Source ID.

`file` *path*;

Path to the file being monitored.

path **(type: `str`)**

`pattern` *pat*;

Pattern being searched for.

pat **(type: `regexp`)**

Searched pattern.

IP adress position should be marked by parenthesis.

`threshold` *count sec*;

Thresholds for watchdog failures.

If given number of attempts is found within given time period in the file, the client IP address is reported by the watchdog.

count **(type: `uint8`)**

sec **(type: `uint32`)**

Constraints:

Maximum COUNT value is 10..

`record-lifetime` [*sec*];

Address record lifetime.

Addresses not seen within this period are removed from the IDS database.

sec **(type: `uint32`, optional, default: 86400)**

`blocking` [*mode*];

Address blocking configuration.

mode **(type: `yes-no`, optional, default: yes)**

`max-entries` [*size*];

Maximum number of IPS table entries held in PF.

size **(type: `uint32`, optional, default: 200000)**

Maximum table size.

[End of section `ids-watchdog` description.]

ids-agent-from-to [*hosts* [*ports*]];

hosts (type: **host-set**, optional, default: *)

Set of hosts to apply the rule to

ports (type: **port-set**, optional, default: *)

Set of ports to apply the rule to

ids-agent-rule-def *name* {

action ... ;

protocol ... ;

src ... ;

dst ... ;

options ... ;

}

IDS rule to be added

Constraints:

ACTION must be specified.

OPTIONS must be specified.

Items & subsections:

action *action*;

Rule action.

action (type: **ids-agent-rule-action**)

protocol [*proto*];

Protocol to be scanned. Defaults to any protocol if omited.

The available protocols depend on the agent configuration. The protocols that are listed here are available in the default agent configuration.

proto (type: **ids-agent-protocol**, optional, default: any)

src [*hosts* [*ports*]];

Source addresses and ports. Defaults to any address and any port if omitted.

hosts (type: **host-set**, optional, default: *)

Set of hosts to apply the rule to

ports (type: **port-set**, optional, default: *)

Set of ports to apply the rule to

dst [*hosts* [*ports*]];

Destination addresses and ports. Defaults to any address and any port if omitted.

hosts (type: **host-set**, optional, default: *)

Set of hosts to apply the rule to

ports **(type: `port-set`, optional, default: *)**

Set of ports to apply the rule to

`options` *options*;

Rule definition.

See suricata documentation for the options syntax.

For example: "msg:\"Testing rule\"; flow:to server,established; content:\"TEST\"; sid:1999999; classtype:unknown;"

Note that SID numbers have to be unique, range between 1500000 and 1999999 can be used for custom rules.

options **(type: `str`)**

[End of section `ids-agent-rule-def` description.]

`ids-agent-base-rule-policy` *name* **{**

* sid ... ;

}

IDS rule policy base.

Items & subsections:

`sid` *sid*;

Signature identifier

sid **(type: `uint64`)**

[End of section `ids-agent-base-rule-policy` description.]

`ids-agent-base-rate-filter` *name* **{**

* sid ... ;

track-by ... ;

count ... ;

seconds ... ;

new-action ... ;

timeout ... ;

}

IDS rule rate filter base.

The `ids-agent-base-rate-filter` section is derived from `ids-agent-base-rule-policy` section prototype. For detail description of it, see above.

Changes to the `ids-agent-base-rate-filter` section:

Item TRACK-BY required.

Item COUNT required.

Item SECONDS required.

Item NEW-ACTION required.

Item TIMEOUT required.

Added items & subsections:

`track-by src;`

`track-by dst;`

`track-by rule;`

`track-by both;`

How to track the exception.

<**branching element**> (**type: `ids-agent-rate-filter-track-by`**)

`count` *count*;

Number of rule hits before this exception is activated.

count (**type: `uint64`**)

`seconds` *seconds*;

Time period within which the COUNT needs to be reached to activate this exception.

seconds (**type: `uint64`**)

`new-action` *action*;

The rule action is changed to this action when this exception matches.

action (**type: `ids-agent-rule-action`**)

`timeout` *timeout*;

Time in seconds during which this exception remains active

timeout (**type: `uint64`**)

[End of section `ids-agent-base-rate-filter` description.]

`ids-agent-base-threshold` *name* **{**

* sid ... ;

type ... ;

track-by ... ;

count ... ;

seconds ... ;

}

Threshold base.

The `ids-agent-base-threshold` section is derived from `ids-agent-base-rule-policy` section prototype. For detail description of it, see above.

Changes to the `ids-agent-base-threshold` section:

Item TYPE required.

Item TRACK-BY required.

Item COUNT required.

Item SECONDS required.

Added items & subsections:

`type threshold;`

`type limit;`

`type both;`

Threshold type.

<branching element> (type: **`ids-agent-threshold-type`**)

`track-by src;`

`track-by dst;`

How to track the threshold.

<branching element> (type: **`ids-agent-threshold-track-by`**)

`count` *count*;

The COUNT as described by TYPE.

count (type: **`uint64`**)

`seconds` *seconds*;

Time period within which the COUNT needs to be reached to activate this threshold.

seconds (type: **`uint64`**)

[End of section `ids-agent-base-threshold` description.]

`ids-agent-base-suppress` *name* {

* sid ... ;

direction ... ;

address ... ;

}

Exception base.

The `ids-agent-base-suppress` section is derived from `ids-agent-base-rule-policy` section prototype. For detail description of it, see above.

Added items & subsections:

`direction src;`

`direction dst;`

`direction [any];`

Direction of the suppression.

<branching element> (type: **ids-agent-suppress-direction**, optional, default: any)

address [*address*];

The addresses and ports to base the suppression on.

address (type: **host-set**, optional, default: *)

Set of hosts to apply the rule to

[End of section `ids-agent-base-suppress` description.]

ids-agent-base-change-rule *sid*;

Change rule base.

sid (type: **uint64-list**)

Constraints:

SID list must not be empty.

ids-agent {

phase ... ;

* tag ... ;

* iface ... ;

record-lifetime ... ;

max-entries ... ;

rules { ... }

blocking { ... }

engine { ... }

rotate-log ... ;

}

An IDS application for advanced inspection of network traffic.

It uses complex rules downloaded from a central server to monitor traffic on given interfaces.

Constraints:

At least one IFACE has to be specified.

Items & subsections:

phase [*number*];

Application Startup Phase.

number (type: **uint8**, optional, default: 50)

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

iface *name*;

Network interfaces watched by the IDS agent.

Warning!:

For all interfaces IDS-AGENT listens on, it is necessary to disable various hardware offloadings by adding flags -rxcsum -tso -toe -lro to ifconfig.

Otherwise, IDS-AGENT will set these flags when starting and unset them when stopping which will cause the interface to be restarted.

This is done automatically for all hardware interfaces and for interfaces of types vlan, vmx and lagg.

It is therefore recommended to perform a reboot after adding, changing or removing item IDS-AGENT.IFACE, especially if you have a cluster.

name **(type: `name of interface`, see interface(5))**

record-lifetime [*sec*];

Address record lifetime.

Hosts not seen within this period are removed from the DB.

sec **(type: `uint32`, optional, default: 86400)**

max-entries [*size*];

Maximum number of IPS table entries held in PF.

size **(type: `uint32`, optional, default: 200000)**

Maximum table size.

rules {

* add-rule *name* { ... }

* include-rules ... ;

* modify-rules ... ;

enable-rules ... ;

disable-rules ... ;

change-rules-to-block ... ;

* rule-rate-filter *name* { ... }

* global-rate-filter *name* { ... }

* rule-threshold *name* { ... }

* global-threshold *name* { ... }

* rule-suppress *name* { ... }

* global-suppress *name* { ... }

}

IDS rules configuration and modification.

Items & subsections:

`add-rule` *name* **{**

action ... ;

protocol ... ;

src ... ;

dst ... ;

options ... ;

}

Custom IDS rule

The `add-rule` section is derived from `ids-agent-rule-def` section prototype. For detail description of it, see above.

`include-rules` *file*;

Include raw-defined rules from a file.

file (type: **`name`** of **`shared-file`**, see **common**(5))

File to include raw-defined rules from

`modify-rules` *sid replace-regex replace-with*;

Modify IDS rules.

sid (type: **`uint64-list`**)

replace-regex (type: **`regexp`**)

replace-with (type: **`str`**)

Constraints:

SID list must not be empty.

`enable-rules` *sid*;

Enable IDS rules

sid (type: **`uint64-list`**)

Constraints:

SID list must not be empty.

`disable-rules` *sid*;

Disable IDS rules.

A disabled rule is not passed to the agent.

sid (type: **`uint64-list`**)

Constraints:

SID list must not be empty.

change-rules-to-block *sid*;

Change the action of IDS rules to block

sid (type: **uint64-list**)

Constraints:

SID list must not be empty.

rule-rate-filter *name* {

```
* sid ... ;
  track-by ... ;
  count ... ;
  seconds ... ;
  new-action ... ;
  timeout ... ;
}
```

Conditionally change the action of selected IDS rules

The rule-rate-filter section is derived from ids-agent-base-rate-filter section prototype. For detail description of it, see above.

Changes to the rule-rate-filter section:

At least one SID must be entered.

global-rate-filter *name* {

```
  track-by ... ;
  count ... ;
  seconds ... ;
  new-action ... ;
  timeout ... ;
}
```

Conditionally change the action of all IDS rules

The global-rate-filter section is derived from ids-agent-base-rate-filter section prototype. For detail description of it, see above.

Changes to the global-rate-filter section:

Item `sid` is not valid.

rule-threshold *name* {

```
* sid ... ;
  type ... ;
  track-by ... ;
  count ... ;
  seconds ... ;
}
```

Conditionally limit the selected IDS rules

The rule-threshold section is derived from ids-agent-base-threshold section prototype. For detail description of it, see above.

Changes to the `rule-threshold` section:

At least one SID must be entered.

`global-threshold` *name* **{**

type ... ;

track-by ... ;

count ... ;

seconds ... ;

}

Conditionally limit all IDS rules

The `global-threshold` section is derived from `ids-agent-base-threshold` section prototype. For detail description of it, see above.

Changes to the `global-threshold` section:

Item `sid` is not valid.

`rule-suppress` *name* **{**

* sid ... ;

direction ... ;

address ... ;

}

Conditionally suppress selected IDS rules.

A suppressed rule acts the same as a rule with action "pass", which means:

- it doesn't generate an alert,

- it prevents other rules from matching on that packet.

The `rule-suppress` section is derived from `ids-agent-base-suppress` section prototype. For detail description of it, see above.

Changes to the `rule-suppress` section:

At least one SID must be entered.

Item ADDRESS requires item DIRECTION to be specified..

Item DIRECTION requires atleast one item ADDRESS to be specified..

`global-suppress` *name* **{**

direction ... ;

address ... ;

}

Conditionally suppress all IDS rules.

A suppressed rule acts the same as a rule with action "pass", which means:

- it does not generate an alert,

- it prevents other rules from matching on that packet.

The `global-suppress` section is derived from `ids-agent-base-suppress` section prototype. For detail description of it, see above.

Changes to the `global-suppress` section:

Item `sid` is not valid.

At least one address must be specified..

[End of section `ids-agent.rules` description.]

`blocking {`

direction ... ;

alerts ... ;

log ... ;

block ... ;

}

IDS agent blocking configuration.

IDS agent has two levels of detecting suspicious traffic:

- alerts, which are only mildly suspicious and thus are only logged, and

- blocks, which are severe alerts that are also reported to the IPS module for blocking

Items & subsections:

`direction src`;

`direction dst`;

`direction [both]`;

Which addresses to block when IPS mode is enabled.
<**branching element**> **(type: `ids-agent-detection-direction`, optional, default: both)**

`alerts [`*val*`]`;

Whether to consider rules with action ALERT to have action BLOCK.

val **(type: `yes-no`, optional, default: no)**

`log [`*val*`]`;

Whether to log IDS blocks to /var/log/kernun-ids-agent

val **(type: `yes-no`, optional, default: yes)**

`block [`*val*`]`;

Whether to actually block the addresses or just log what would be blocked.
This item makes difference only when IPS is configured. It can be used to disable the blocking mechanism of IDS-AGENT without having to disable the entire IPS.

val **(type: `yes-no`, optional, default: yes)**

[End of section `ids-agent.blocking` description.]

`engine {`

flags ... ;

cfg-file ... ;

log-level ... ;

}

IDS agent engine configuration

Items & subsections:

`flags` ***flags*;**

Flags to be passed to the engine upon start.

flags (type: **str**)

cfg-file *file*;

The configuration file for the agent.

Default value is "samples/shared/ids-agent.yaml" from distribution. CML modifies this file according to its purpose, i.e:

- CML overwrites variable definitions according to ADDRESS-GROUPS and PORT-GROUPS

- CML overwrites the path and generates the content of threshold file if atleast one item GLOBAL-RATE-FILTER, RULE-RATE-FILTER, GLOBAL-THRESHOLD, RULE-THRESHOLD, GLOBAL-SUPPRESS or RULE-SUPPRESS is specified

- CML overwrites the pcap section according to IFACE item

- CML appends the path to the downloaded rules to the rule-files list when item RULES-DOWNLOAD is specified

file (type: **name** of **shared-file**, see common(5))

log-level none;

log-level emergency;

log-level alert;

log-level critical;

log-level error;

log-level warning;

log-level [notice];

log-level info;

log-level perf;

log-level config;

log-level debug;

Engine log level

<branching element> (type: **ids-agent-log-level**, optional, default: **notice**)

[End of section `ids-agent.engine` description.]

rotate-log [user *user*] **[group** *group*] **[mode** *mode*] **[count** *count*] **[size** *size*] [*when* [*zip*]];

Log file rotation description.

Use the SIZE elem if log file size criterion required. Use the WHEN elem if periodical rotation required. If used both SIZE and WHEN elems, the log file is rotated at a proper time only if size limit is reached.

user *user* (type: **str**, optional, default: <NULL>)

Log file owner - user.

group *group* (type: **str**, optional, default: "wheel")

Log file owner - group.

mode *mode* (type: **uint16**, optional, default: 640)

Log file permissions.

`count` *count* (type: `uint16`, optional, default: 31)
Number of days being archived.

`size` *size* (type: `uint16`, optional, default: 0)
Size limit for rotation in KB (ignore log file size if omitted).

when **(type: `time-cond`, optional, default: anytime)**
Rotation periodicity (use SIZE condition if omitted).

zip **(type: `zip-mode`, optional, default: bzip2)**
Zipping mode.

Constraints:
Use either size criterion or defined periodicity.

[End of section `ids-agent` description.]

`adaptive-firewall {`

ids-agent { ... }
* watchdog *name* { ... }
honeypot { ... }
auto-blocking { ... }
adaptive-database { ... }
address-groups { ... }
port-groups { ... }
whitelist ... ;
blacklist ... ;
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
}

Configuration of adaptive IDS/IPS system.

Constraints:
IDS-AGENT requires non-empty ADDRESS-GROUPS.HOME-NET.

Items & subsections:

`ids-agent {`

phase ... ;
* tag ... ;
* iface ... ;
record-lifetime ... ;
max-entries ... ;
rules { ... }

blocking { ... }
engine { ... }
rotate-log ... ;
}

The `ids-agent` section is derived from `ids-agent` section prototype. For detail description of it, see above.

`watchdog` *name* {

id ... ;
file ... ;
* pattern ... ;
* threshold ... ;
record-lifetime ... ;
blocking ... ;
max-entries ... ;
}

The `watchdog` section is derived from `ids-watchdog` section prototype. For detail description of it, see above.

`honeypot` {

* non-transparent ... ;
record-lifetime ... ;
blocking ... ;
max-entries ... ;
}

A detector that provides a dummy TCP server which listens on an address that is not assigned to a real host.

A client attempting to connect to this server is assumed to be a port scanner.

The `honeypot` section is derived from `listen-on` section prototype. For detail description of it, see listen-on(5).

Changes to the `honeypot` section:

Item `transparent` is not valid.

Added items & subsections:

`record-lifetime` [*sec*];

Address record lifetime.

Addresses not seen within this period are removed from the IDS database.

sec **(type: `uint32`, optional, default: 86400)**

`blocking` [*mode*];

Address blocking configuration.

mode **(type: `yes-no`, optional, default: yes)**

max-entries [*size*];

Maximum number of IPS table entries held in PF.

size **(type: uint32, optional, default: 200000)**

Maximum table size.

[End of section `adaptive-firewall.honeypot` description.]

auto-blocking {

record-lifetime ... ;

save-delay ... ;

refresh ... ;

cleanup-time ... ;

}

Autonomous blocking guard parameters.

It blocks addresses added to the blacklist by internal detectors (honeypot, watchdog etc.).

Items & subsections:

record-lifetime [*sec*];

Blacklist record lifetime.

Addresses not seen within this period are removed from the database.

sec **(type: uint32, optional, default: 86400)**

save-delay [*sec*];

SQL transaction maximum duration.

sec **(type: uint32, optional, default: 1)**

refresh daily [**time** *time*] [**report** *report*];

refresh hourly [**minute** *minute*] [**report** *report*];

refresh [**every**] [**period** *period*] [**at** *at*] [**report** *report*];

refresh raw raw *raw* [**report** *report*];

refresh manually;

Schedule refresh of internal blocking rules.

<**branching element**> **(type: task-frequency, optional, default: every)**

raw *raw* **(type: str)**

Raw line to be placed into crontab. First 5 columns (the time specification) must be specified.

minute *minute* **(type: time, optional, default: 0)**

Starting time of task (mm, hour ignored).

time *time* **(type: time, optional, default: 415)**

Starting time of task (hhmm).

period *period* **(type: uint8, optional, default: 1)**

Run the task every PERIOD minutes (mm, hours ignored).

at *at* **(type: uint8, optional, default: 0)**

Starting time of task (mm, hours ignored)

report *report* **(type: report-mode, optional, default: nothing=0)**

Task output (stdout and stderr) delivery.

cleanup-time [*hhmm***];**

Time of day when the database cleanup is done.

At time given in this item, records for address not seen within particular RECORD-LIFETIME period are removed.

hhmm **(type: time, optional, default: 303)**

[End of section `adaptive-firewall.auto-blocking` description.]

adaptive-database {

policy ... ;

max-entries ... ;

}

Kernun Adaptive Database application.

Items & subsections:

policy [from *from***];**

Address blocking policy.

The higher the policy is, the more IP addresses are blocked.

from *from* **(type: policy-level, optional, default: high)**

max-entries [*size***];**

Maximum number of Adaptive Database table entries held in PF.

size **(type: uint32, optional, default: 100000)**

Maximum table size.

[End of section `adaptive-firewall.adaptive-database` description.]

address-groups {

home-net ... ;

external-net ... ;

http-servers ... ;

sql-servers ... ;

smtp-servers ... ;

dns-servers ... ;

}

Address variables.

Defining these variables according to your network will increase the accuracy of the Adaptive Firewall.

Items & subsections:

home-net [*home-net*];

Set of addresses in your network that are to be protected by the Adaptive Firewall. These addresses are by default added to the whitelist and thus cannot end up on the blacklist.

Note that when running IDS agent on external interface (or any interface with NAT), it is necessary to include the external address in this set because otherwise the traffic will not get matched by most rules.

About 75 % of IDS rules use this variable.

home-net (type: **net-list**, optional, default: {})

external-net *external-net*;

Set of addresses that are not in your network.

Defaults to negation of HOME-NET when unspecified.

About 85 % of IDS rules use this variable.

external-net (type: **net-list**)

http-servers *http-servers*;

Set of DNS servers used in your network.

Defaults to HOME-NET when unspecified.

About 25 % of IDS rules use this variable.

http-servers (type: **net-list**)

sql-servers *sql-servers*;

Set of SQL servers used in your network.

Defaults to HOME-NET when unspecified.

About 2 % of IDS rules use this variable.

sql-servers (type: **net-list**)

smtp-servers *smtp-servers*;

Set of SMTP servers used in your network.

Defaults to HOME-NET when unspecified.

About 1 % of IDS rules use this variable.

smtp-servers (type: **net-list**)

dns-servers *dns-servers*;

Set of DNS servers used in your network.

Defaults to HOME-NET when unspecified.

Less than 1 % of IDS rules use this variable.

dns-servers (type: **net-list**)

[End of section `adaptive-firewall.address-groups` description.]

port-groups {

http-ports ... ;

oracle-ports ... ;

shellcode-ports ... ;

ssh-ports ... ;

}

Port variables.

Defining these variables according to your network will increase the accuracy of the Adaptive Firewall.

Items & subsections:

`http-ports` *http-ports*;

Set of HTTP ports used in your network.

Defaults to { 80, 8080 } when unspecified.

About 25 % of IDS rules use this variable.

http-ports (type: **`port-set`**)

`oracle-ports` *oracle-ports*;

Set of SSH ports used in your network.

Defaults to { 1521 } when unspecified.

About 1 % of IDS rules use this variable.

oracle-ports (type: **`port-set`**)

`shellcode-ports` *shellcode-ports*;

Set of SSH ports used in your network.

Defaults to ! { 80 } when unspecified.

Less than 1 % of IDS rules use this variable.

shellcode-ports (type: **`port-set`**)

`ssh-ports` *ssh-ports*;

Set of SSH ports used in your network.

Defaults to { 22 } when unspecified.

Less than 1 % of IDS rules use this variable.

ssh-ports (type: **`port-set`**)

[End of section `adaptive-firewall.port-groups` description.]

`whitelist` [**`no-home-net`**] [**`no-servers`**] [*list*];

Whitelisted addresses.

These addresses are never blocked by IPS but their traffic can still be blocked due to the other side of communication getting blocked by IPS.

The addresses can get detected by some IDS detectors but most will not even report them.

`no-home-net` (type: **`key`**, optional)

Whether to put addresses from ADDRESS-GROUPS.HOME-NET on the whitelist.

It can be useful to disable this if you want to allow IPS to potentially block some addresses in your network.

`no-servers` (type: **`key`**, optional)

Whether to put addresses from the following items on the whitelist:

- ADDRESS-GROUPS.DNS-SERVERS
- ADDRESS-GROUPS.HTTP-SERVERS

- ADDRESS-GROUPS.SMTP-SERVERS
- ADDRESS-GROUPS.SQL-SERVERS

It can be useful to disable this if you want these addresses to be temporarily blockable when someone is spoofing them.

list **(type: `host-set`, optional, default: {})**

`blacklist` [*list*];

Blacklisted addresses.

These addresses are always blocked by IPS.

list **(type: `net-list`, optional, default: {})**

`stats-daily` {

top-clients ... ;

top-servers ... ;

top-src-ips ... ;

top-dst-ips ... ;

top-rules ... ;

}

The `stats-daily` section is derived from `summary` section prototype.
For detail description of it, see application(5).
Changes to the `stats-daily` section:

Item `top-users` is not valid.
Item `top-groups` is not valid.
Item `top-categories` is not valid.
Item `top-senders` is not valid.
Item `top-recipients` is not valid.
Item `top-mime-types` is not valid.
Item `top-qnames` is not valid.
Item `top-qtypes` is not valid.
Item `top-callers` is not valid.
Item `top-receivers` is not valid.
Item `top-sids` is not valid.
Item `top-server-ports` is not valid.
Item `spam-threshold` is not valid.
Section `activity-report` is not valid.

`stats-weekly` {

top-clients ... ;

top-servers ... ;

top-src-ips ... ;

top-dst-ips ... ;

top-rules ... ;

}

The `stats-weekly` section is derived from `summary` section prototype. For detail description of it, see application(5).
Changes to the `stats-weekly` section:

Item `top-users` is not valid.

Item `top-groups` is not valid.

Item `top-categories` is not valid.

Item `top-senders` is not valid.

Item `top-recipients` is not valid.

Item `top-mime-types` is not valid.

Item `top-qnames` is not valid.

Item `top-qtypes` is not valid.

Item `top-callers` is not valid.

Item `top-receivers` is not valid.

Item `top-sids` is not valid.

Item `top-server-ports` is not valid.

Item `spam-threshold` is not valid.

Section `activity-report` is not valid.

`stats-monthly` {

top-clients ... ;

top-servers ... ;

top-src-ips ... ;

top-dst-ips ... ;

top-rules ... ;

}

The `stats-monthly` section is derived from `summary` section prototype. For detail description of it, see application(5).
Changes to the `stats-monthly` section:

Item `top-users` is not valid.

Item `top-groups` is not valid.

Item `top-categories` is not valid.

Item `top-senders` is not valid.

Item `top-recipients` is not valid.

Item `top-mime-types` is not valid.

Item `top-qnames` is not valid.

Item `top-qtypes` is not valid.

Item `top-callers` is not valid.

Item `top-receivers` is not valid.

Item `top-sids` is not valid.

Item `top-server-ports` is not valid.

Item `spam-threshold` is not valid.

Section `activity-report` is not valid.

[End of section `adaptive-firewall` description.]

SEE ALSO

configuration(7), application(5), common(5), interface(5), listen-on(5)

NAME

alertd — format of alertd component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **alertd** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **alertd** configuration directives:

yes-no (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

ITEMS AND SECTIONS

Configuration of **alertd** library component consists of following prototypes:

alertd { ... }

Description:

alertd {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

* snmp-manager *name* { ... }

}

SNMP Alert Daemon configuration.

The `alertd` section is derived from `alone-application` section prototype. For detail description of it, see application(5).
Added items & subsections:

`snmp-manager` *name* {

host ... ;

port ... ;

community ... ;

}

SNMP Manager Definition.

Items & subsections:

`host` *host*;

SNMP Manager Host.

host **(type: `host`)**

`port` [*port*];

SNMP Manager Port.

port **(type: `port`, optional, default: 162)**

`community` [*community*];

SNMP Community.

community **(type: `str`, optional, default: "public")**

[End of section `alertd.snmp-manager` description.]

[End of section `alertd` description.]

SEE ALSO

configuration(7), application(5), common(5), log(5)

NAME

alertd.cfg — format of alertd program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **alertd.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **alertd.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

ip-version (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

log-in-vain-proto (see **sysctl**(5))

blackhole-proto (see **sysctl**(5))

ITEMS AND SECTIONS

Program **alertd** recognizes following items and sections:

* resolver *name* { ... }
  sysctl { ... }
  use-resolver ... ;
  alertd { ... }
  ipv6-mode ... ;

Description:

resolver *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

sysctl {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

use-resolver *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

alertd {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

* snmp-manager *name* { ... }

}

The `alertd` section is derived from `alertd` section prototype. For detail description of it, see alertd(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), alertd(8), alertd(5), common(5), log(5), resolver(5), sysctl(5)

NAME

altq — format of altq component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **altq** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

ITEMS AND SECTIONS

Configuration of **altq** library component consists of following prototypes:

altq ... ;
altq2 ... ;

Description:

altq *altq*;

ALTQ queue selection for traffic shaping (UDP).

altq (type: **name** of **pf-queue**, see pf-queue(5))

queue name

altq2 *altq* [**paltq** *paltq*];

ALTQ queue selection for traffic shaping (TCP).

altq (type: **name** of **pf-queue**, see pf-queue(5))

queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), **optional**, **default: NULL**)

priority queue name (if set, used for TCP ACK without data)

SEE ALSO

configuration(7), pf-queue(5)

NAME

antivirus — format of antivirus component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **antivirus** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **antivirus** configuration directives:

`enabling` (see **common**(5))

`antivirus-protocol` (name-usage obligatory)

Which antivirus software is used (selects communication protocol).

`clamav-file`

Clam AntiVirus communicating via file

`icap`

generic antivirus communicating via ICAP

`kav-debug-level` (name-usage optional)

Debug level of the Kaspersky antivirus

`L0` (0)

Logging off.

`L1` (1)

Scanning status only, no tracing data is output.

`L2` (2)

Error messages and critical faults.

`L3` (3)

Warning messages.

`L4` (4)

Informational messages.

L5 (5)

Detailed informational messages.

L6 (6)

Informational messages with extra details.

L7 (7)

Application traces.

L8 (8)

Enhanced tracing.

L9 (9)

Debug output. This is a recommended level for bug reports.

L10 (10)

Full tracing detail. This is a maximum supported level of logging detail.

virus-status (name-usage obligatory)

Antivirus detection status. This enumeration is used when checking results of an antivirus run.

found

Mail or document scanned, at least one virus found.

free

Mail or document scanned, no virus was found.

skipped

Mail or document not scanned or antivirus disabled.

unknown

Antivirus returned an unknown response.

error

Antivirus failed.

database-source (name-usage obligatory)

Antivirus database source.

none

No database is used.

file

Path to a file in the filesystem.

ITEMS AND SECTIONS

Configuration of **antivirus** library component consists of following prototypes:
* antivirus *name* { ... }

antivirus-keepalive ... ;
use-antivirus ... ;
antivirus-mode ... ;
accept-antivirus-status ... ;

Description:

antivirus *name* **{**

connection ... ;

sock-opt { ... }

timeout ... ;

comm-dir ... ;

altq ... ;

max-checked-size ... ;

icap-pass-200-with-pure-body ... ;

persistent-stream ... ;

clamav-agent { ... }

}

Settings of antivirus checking.

Constraints:

ICAP Antivirus engine is not available as agent.

CLAMAV-AGENT are allowed only for CLAMAV-* antiviruses.

Items & subsections:

connection clamav-file *inet-socket*;

connection icap *inet-socket* [*uri*];

Connection to antivirus (socket and protocol).

<**branching element**> (**type: antivirus-protocol**)

inet-socket (**type: sock**)

Server IP address/hostname

uri (**type: str**, **optional, default: "/av"**)

URI for ICAP GET request. The scheme, host and port may be omitted so the URI can be written as an absolute path.

sock-opt {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;
}

Connection to antivirus options.

The `sock-opt` section is derived from `sock-opt` section prototype.
For detail description of it, see netio(5).
Changes to the `sock-opt` section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`timeout` [*sec*];

Total timeout for checking of one document.

sec **(type: `uint16`, optional, default: 300)**

`comm-dir` [*path*];

Directory used for communication with antivirus.

path **(type: `str`, optional, default: "/data/tmp/antivirus")**

`altq` *altq* [`paltq` *paltq*];

ALTQ queues for data sent to antivirus.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

`paltq` *paltq* (type: `name` of `pf-queue`, see pf-queue(5), optional, default: NULL)

priority queue name (if set, used for TCP ACK without data)

`max-checked-size` *bytes* [`skip`];

Maximum size of document sent to antivirus engine.

If the document is larger, only the first part of given size is checked. If a virus is found, the appropriate status is returned. If the document is clean, the rest of document is forwarded without checking.

An alternative behavior can be configured when oversized documents are not checked and status SKIPPED is returned.

bytes **(type: `uint64`)**

Size limit in bytes.

`skip` (type: `key`, optional)

This flag causes skipping check for oversized documents.

`icap-pass-200-with-pure-body` [*status*];

ICAP server option - handle 200 OK response with pure document body (without HTTP error response header) as virus-free response. Without this option, all 200 OK responses are considered to be virus-found ones.

status **(type: `enabling`, optional, default: enable)**

`persistent-stream`;

Keeping the antivirus connection alive between several attempts of checking the same file.

clamav-agent {

phase ... ;

* tag ... ;

exclude-pua ... ;

* clamd-raw ... ;

custom-db-source ... ;

* custom-db-url ... ;

* freshclam-raw ... ;

}

ClamAV antivirus engine component.

If used, this section defines parameters of a local agent listening on antivirus connection addresses and executing antivirus scanning.

Items & subsections:

phase [*number*];

Application Startup Phase.

***number* (type: uint8, optional, default: 40)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

***value* (type: str)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

exclude-pua [*list*];

ExcludePUA configuration option values.

***list* (type: str, optional, default: "Packed,PUA.Win.Packer,EncryptedDoc")**

clamd-raw *line*;

Raw lines of clamd.cfg file.

***line* (type: str)**

custom-db-source none;

custom-db-source [file] [*file*];

Source of virus database URL set.

<branching element> (type: database-source, optional, default: file)

***file* (type: str, optional, default: "/usr/local/kernun/license.clamav.dat")**

custom-db-url *url*;

Additional custom URL of virus database.

url (type: **str**)

freshclam-raw *line*;

Raw lines of freshclam.cfg file.

line (type: **str**)

[End of section `antivirus.clamav-agent` description.]

[End of section `antivirus` description.]

antivirus-keepalive *channel* [**interval** *interval*] [**chunk** *chunk*] [**limit** *limit*];

Antivirus usage mode.

Check document by antivirus, with settings for passing initial part of unchecked data through the antivirus module during antivirus checking.

channel (type: **name-list of antivirus**, see above)

Name of ANTIVIRUS global section used.

interval *interval* (type: **uint16**, optional, default: **0**)

Seconds between passing blocks of unchecked data (0 = do not send unchecked data).

chunk *chunk* (type: **uint32**, optional, default: **0**)

Size of each block of unchecked data.

limit *limit* (type: **uint32**, optional, default: **0**)

Maximum size of unchecked data passed before antivirus check is completed. Remaining data will be passed only after successful checking.

use-antivirus disable;

use-antivirus enable *channel*;

Antivirus usage mode.

If omitted, or disabled, no antivirus is enabled. In this case, neither any ANTIVIRUS global section can be present nor any MAIL-ACL and DOC-ACL can have VIRUS item specified.

<**branching element**> (type: **enabling**)

channel (type: **name-list of antivirus**, see above)

antivirus-mode disable [**interval** *interval*] [**chunk** *chunk*] [**limit** *limit*];

antivirus-mode enable *channel* [**interval** *interval*] [**chunk** *chunk*] [**limit** *limit*];

Antivirus usage mode.

If omitted, or disabled, no antivirus is enabled. In this case, neither any ANTIVIRUS global section can be present nor any ACL can have VIRUS item specified.

If enabled, it can be configured for passing initial part of unchecked data to the client before the antivirus check is completed. In this case, if a virus is found later, the connection to the client is broken.

<branching element> (type: **enabling**)

channel (type: **name-list** of **antivirus**, see above)

interval *interval* (type: **uint16**, optional, default: 0)

Seconds between passing blocks of unchecked data (0 = do not send unchecked data).

chunk *chunk* (type: **uint32**, optional, default: 0)

Size of each block of unchecked data.

limit *limit* (type: **uint32**, optional, default: 0)

Maximum size of unchecked data passed before antivirus check is completed. Remaining data will be passed only after successful checking.

accept-antivirus-status *status*;

Defines set of antivirus status codes (in addition to FREE) that allow further passing of data. Other status codes cause termination of data transfer. If not set, data are passed only if the antivirus returns status FREE.

status (type: **virus-status-set**)

SEE ALSO

configuration(7), common(5), netio(5), pf-queue(5)

NAME

application — format of application component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **application** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **application** configuration directives:

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

lock-type (see **ipc**(5))

source-address-mode (see **source-address**(5))

doctype-ident-method (see **acl**(5))

listen-on-sock (see **listen-on**(5))

proc-priority (name-usage obligatory)

Process priority type.

normal

realtime

ITEMS AND SECTIONS

Configuration of **application** library component consists of following prototypes:

priority ... ;
doctype-identification { ... }
graph ... ;
summary { ... }
alone-application { ... }
* clone-application *name* { ... }
* proxy *name* { ... }

Description:

priority [normal];

priority realtime *realtime*;

Process priority setting.

<**branching element**> **(type: proc-priority, optional, default: normal)**

realtime **(type: uint8)**

Realtime priority (parameter of rtprio() call).

Accepted values between 0 and 31; 0 is the highest priority.

Constraints:

Priority value must be between 0 and 31.

doctype-identification {

* order ... ;

mime-types ... ;

magic ... ;

}

Document type recognition attributes.

This section defines attributes and order of different methods of document type recognition methods.

Items & subsections:

order [for *for***]** *order*;

Default order of methods usage.

If omitted, only Content-Type defines document type.

for *for* **(type: direction-set, optional, default: *)**

Document transfer direction set.

This element defines directions for which the order is specified by this item.

For some proxies, both directions can be used while for others either direction is not applicable; consult proxy man page.

order **(type: doctype-ident-method-list)**

Methods are used in given order unless type is recognized.

For some proxies, some methods are not applicable, consult proxy man page.

Constraints:

Only 3 methods can be specified.

mime-types *filename*;

EXTENSION method attributes.

filename **(type: name of shared-file, see** common**(5))**

Extensions to MIME types mapping file.

magic [*filename* **[***scan-size***]];**

MAGIC method attributes.

filename **(type: name of shared-file, see** common**(5), optional, default: NULL)**

Magic numbers to MIME types mapping file.

If omitted, system default file is used.

scan-size **(type: uint32, optional, default: 4096)**

Size of initial part of data used for type recognition.

[End of section `doctype-identification` description.]

graph *top*;

top **(type: uint16)**

summary {

top-clients ... ;

top-users ... ;

top-groups ... ;

top-servers ... ;

top-categories ... ;

top-senders ... ;

top-recipients ... ;

top-mime-types ... ;

top-qnames ... ;

top-qtypes ... ;

top-callers ... ;

top-receivers ... ;

top-sids ... ;

top-server-ports ... ;

spam-threshold ... ;

activity-report { ... }

top-src-ips ... ;

top-dst-ips ... ;

top-rules ... ;

}

General definition of graph parameters for periodic summary lists.

Items & subsections:

`top-clients` [*top*];

Top clients in statistics.

top **(type: `uint16`, optional, default: 20)**

`top-users` [*top*];

Top users in statistics.

top **(type: `uint16`, optional, default: 20)**

`top-groups` [*top*];

Top groups in statistics.

top **(type: `uint16`, optional, default: 10)**

`top-servers` [*top*];

Top servers in statistics.

top **(type: `uint16`, optional, default: 20)**

`top-categories` [*top*];

Top Clear Web categories in statistics.

top **(type: `uint16`, optional, default: 10)**

`top-senders` [*top*];

Top mail senders in statistics.

top **(type: `uint16`, optional, default: 20)**

`top-recipients` [*top*];

Top mail recipients in statistics.

top **(type: `uint16`, optional, default: 20)**

`top-mime-types` [*top*];

Top attachment MIME types in statistics.

top **(type: uint16, optional, default: 10)**

top-qnames [*top***];**

Top query names in statistics.

top **(type: uint16, optional, default: 20)**

top-qtypes [*top***];**

Top query types in statistics.

top **(type: uint16, optional, default: 10)**

top-callers [*top***];**

Top call initiators in statistics.

top **(type: uint16, optional, default: 20)**

top-receivers [*top***];**

Top call receivers in statistics.

top **(type: uint16, optional, default: 20)**

top-sids [*top***];**

Top SIDs (ids-agent rule identifiers) in statistics.

top **(type: uint16, optional, default: 20)**

top-server-ports [*top***];**

Top server ports in statistics.

top **(type: uint16, optional, default: 20)**

spam-threshold [*value***];**

Spam score threshold for a mail to be considered SPAM.

value **(type: uint16, optional, default: 5000)**

activity-report {

server-max ... ;

}

Generate a detailed report of client/user activity.

Items & subsections:

server-max [*val***];**

The number of characters displayed from the end of a long server name in a client/user activity report. If the item has value 0, the whole server name will be displayed.

val **(type: uint16, optional, default: 40)**

[End of section `summary.activity-report` description.]

top-src-ips [*top***];**

Top source IP addresses.

top **(type: uint16, optional, default: 20)**

top-dst-ips [*top*];

Top destination IP addresses.

***top* (type: uint16, optional, default: 20)**

top-rules [*top*];

Top Snort rules.

***top* (type: uint16, optional, default: 20)**

[End of section `summary` description.]

alone-application {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

}

This section defines general TNS-wide nonrepeatable application attributes.

Items & subsections:

phase [*number*];

Application Startup Phase.

***number* (type: uint8, optional, default: 50)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

`log-debug {`

level ... ;

mem-level ... ;

facility ... ;

file ... ;

rotate ... ;

mem-file ... ;

syslog-failure ... ;

data-limit ... ;

dump-hold-time ... ;

}

The `log-debug` section is derived from `log` section prototype. For detail description of it, see log(5).

`log-stats {`

level ... ;

mem-level ... ;

facility ... ;

file ... ;

rotate ... ;

mem-file ... ;

syslog-failure ... ;

data-limit ... ;

dump-hold-time ... ;

}

The `log-stats` section is derived from `log` section prototype. For detail description of it, see log(5).

`use-resolver` *name*;

Resolver Specification.

This item defines resolver configuration used for this proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`cfg-resolution` [*max-addrs* [*min-ttl* [*def-ttl* [*max-ttl* [*hosts-ttl* [*pool-dir*]]]]]];

Attributes for resolution of domain names in configuration.

max-addrs (type: **uint8**, optional, default: **10**)

Maximum of addresses per a single domain name.

min-ttl (type: **uint32**, optional, default: **10**)

Minimum TTL accepted, used instead of too small TTL values (e.g. 0).

def-ttl (type: **uint32**, optional, default: **1m**)

Default TTL used in case of unsuccessful DNS resolution.

max-ttl (type: **uint32**, optional, default: **1d**)

Maximum TTL accepted, used instead of large TTL values.

hosts-ttl (type: **uint32**, optional, default: **1d**)

TTL used for names in /etc/hosts.

pool-dir (type: **str**, optional, default: **"/tmp"**)

Directory for temporary files used to share results.

monitoring {

disabled ... ;

comm-dir ... ;

interval ... ;

user ... ;

aproxy-user ... ;

data ... ;

}

The `monitoring` section is derived from `monitoring` section prototype. For detail description of it, see monitoring(5).

stats-daily {

}

The `stats-daily` section is derived from `summary` section prototype. For detail description of it, see above.
Changes to the `stats-daily` section:

Item `top-clients` is not valid.

Item `top-users` is not valid.

Item `top-groups` is not valid.

Item `top-servers` is not valid.

Item `top-categories` is not valid.

Item `top-senders` is not valid.

Item `top-recipients` is not valid.

Item `top-mime-types` is not valid.

Item `top-qnames` is not valid.

Item `top-qtypes` is not valid.

Item `top-callers` is not valid.

Item `top-receivers` is not valid.

Item `top-sids` is not valid.

Item top-server-ports is not valid.

Item spam-threshold is not valid.

Section activity-report is not valid.

Item top-src-ips is not valid.

Item top-dst-ips is not valid.

Item top-rules is not valid.

stats-weekly {

}

The stats-weekly section is derived from summary section prototype. For detail description of it, see above.
Changes to the stats-weekly section:

Item top-clients is not valid.

Item top-users is not valid.

Item top-groups is not valid.

Item top-servers is not valid.

Item top-categories is not valid.

Item top-senders is not valid.

Item top-recipients is not valid.

Item top-mime-types is not valid.

Item top-qnames is not valid.

Item top-qtypes is not valid.

Item top-callers is not valid.

Item top-receivers is not valid.

Item top-sids is not valid.

Item top-server-ports is not valid.

Item spam-threshold is not valid.

Section activity-report is not valid.

Item top-src-ips is not valid.

Item top-dst-ips is not valid.

Item top-rules is not valid.

stats-monthly {

}

The stats-monthly section is derived from summary section prototype. For detail description of it, see above.
Changes to the stats-monthly section:

Item top-clients is not valid.

Item top-users is not valid.

Item top-groups is not valid.

Item top-servers is not valid.

Item top-categories is not valid.

Item top-senders is not valid.

Item `top-recipients` is not valid.
Item `top-mime-types` is not valid.
Item `top-qnames` is not valid.
Item `top-qtypes` is not valid.
Item `top-callers` is not valid.
Item `top-receivers` is not valid.
Item `top-sids` is not valid.
Item `top-server-ports` is not valid.
Item `spam-threshold` is not valid.
Section `activity-report` is not valid.
Item `top-src-ips` is not valid.
Item `top-dst-ips` is not valid.
Item `top-rules` is not valid.

nodaemon;

Do not daemonize itself.

singleproc;

Do not fork any child processes.

app-user [*name*]**;**

User to run the program as.

If the program is started by root, it changes its identity. Otherwise, the program must be started by named user.

name **(type: str, optional, default: "kernun")**

idle-timeout [*seconds*]**;**

If no data is transmitted for a session within the period of specified amount of seconds, connection (TCP case) or logical connection (UDP case) is closed.

Value of 0 (zero) means 'no limitation'.

seconds **(type: uint32, optional, default: 3600)**

run-block-sigalrm [*val*]**;**

Block SIGALRM while a module runs.

val **(type: yes-no, optional, default: yes)**

[End of section `alone-application` description.]

clone-application *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

}

This section defines general TNS-wide repeatable application attributes.

Items & subsections:

`phase` [*number*];

Application Startup Phase.

number **(type: `uint8`, optional, default: 50)**

Phase number; the lower one, the earlier start.

`tag` *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

`log-debug` {

level ... ;

mem-level ... ;

facility ... ;

file ... ;

rotate ... ;

mem-file ... ;

syslog-failure ... ;

data-limit ... ;

dump-hold-time ... ;

}

The `log-debug` section is derived from `log` section prototype. For detail description of it, see log(5).

`log-stats` {

level ... ;

mem-level ... ;

facility ... ;

file ... ;

rotate ... ;

mem-file ... ;

syslog-failure ... ;

data-limit ... ;

dump-hold-time ... ;

}

The `log-stats` section is derived from `log` section prototype. For detail description of it, see log(5).

`use-resolver` *name*;

Resolver Specification.

This item defines resolver configuration used for this proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`cfg-resolution` [*max-addrs* [*min-ttl* [*def-ttl* [*max-ttl* [*hosts-ttl* [*pool-dir*]]]]]];

Attributes for resolution of domain names in configuration.

max-addrs **(type: `uint8`, optional, default: 10)**

Maximum of addresses per a single domain name.

min-ttl **(type: `uint32`, optional, default: 10)**

Minimum TTL accepted, used instead of too small TTL values (e.g. 0).

def-ttl **(type: `uint32`, optional, default: 1m)**

Default TTL used in case of unsuccessful DNS resolution.

max-ttl **(type: `uint32`, optional, default: 1d)**

Maximum TTL accepted, used instead of large TTL values.

hosts-ttl **(type: `uint32`, optional, default: 1d)**

TTL used for names in /etc/hosts.

pool-dir **(type: `str`, optional, default: "/tmp")**

Directory for temporary files used to share results.

`monitoring` {

disabled ... ;

comm-dir ... ;

interval ... ;

user ... ;

aproxy-user ... ;

data ... ;

}

The `monitoring` section is derived from `monitoring` section prototype. For detail description of it, see monitoring(5).

`stats-daily` {

}

The `stats-daily` section is derived from `summary` section prototype. For detail description of it, see above.
Changes to the `stats-daily` section:

Item `top-clients` is not valid.

Item `top-users` is not valid.

Item `top-groups` is not valid.

Item `top-servers` is not valid.

Item `top-categories` is not valid.

Item `top-senders` is not valid.

Item `top-recipients` is not valid.

Item `top-mime-types` is not valid.

Item `top-qnames` is not valid.

Item `top-qtypes` is not valid.

Item `top-callers` is not valid.

Item `top-receivers` is not valid.

Item `top-sids` is not valid.

Item `top-server-ports` is not valid.

Item `spam-threshold` is not valid.

Section `activity-report` is not valid.

Item `top-src-ips` is not valid.

Item `top-dst-ips` is not valid.

Item `top-rules` is not valid.

`stats-weekly` {

}

The `stats-weekly` section is derived from `summary` section prototype. For detail description of it, see above.
Changes to the `stats-weekly` section:

Item `top-clients` is not valid.

Item `top-users` is not valid.

Item `top-groups` is not valid.

Item `top-servers` is not valid.

Item `top-categories` is not valid.

Item `top-senders` is not valid.

Item `top-recipients` is not valid.

Item `top-mime-types` is not valid.

Item `top-qnames` is not valid.

Item `top-qtypes` is not valid.

Item `top-callers` is not valid.

Item `top-receivers` is not valid.

Item `top-sids` is not valid.

Item `top-server-ports` is not valid.

Item `spam-threshold` is not valid.

Section `activity-report` is not valid.

Item `top-src-ips` is not valid.

Item `top-dst-ips` is not valid.

Item `top-rules` is not valid.

`stats-monthly` {

}

The `stats-monthly` section is derived from `summary` section prototype. For detail description of it, see above.
Changes to the `stats-monthly` section:

Item `top-clients` is not valid.

Item `top-users` is not valid.

Item `top-groups` is not valid.

Item `top-servers` is not valid.

Item `top-categories` is not valid.

Item `top-senders` is not valid.

Item `top-recipients` is not valid.

Item `top-mime-types` is not valid.

Item `top-qnames` is not valid.

Item `top-qtypes` is not valid.

Item `top-callers` is not valid.

Item `top-receivers` is not valid.

Item `top-sids` is not valid.

Item `top-server-ports` is not valid.

Item `spam-threshold` is not valid.

Section `activity-report` is not valid.

Item `top-src-ips` is not valid.

Item `top-dst-ips` is not valid.

Item `top-rules` is not valid.

`nodaemon`;

Do not daemonize itself.

singleproc;

Do not fork any child processes.

app-user [*name*];

User to run the program as.

If the program is started by root, it changes its identity. Otherwise, the program must be started by named user.

name **(type: str, optional, default: "kernun")**

idle-timeout [*seconds*];

If no data is transmitted for a session within the period of specified amount of seconds, connection (TCP case) or logical connection (UDP case) is closed.

Value of 0 (zero) means 'no limitation'.

seconds **(type: uint32, optional, default: 3600)**

run-block-sigalrm [*val*];

Block SIGALRM while a module runs.

val **(type: yes-no, optional, default: yes)**

[End of section `clone-application` description.]

proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

udpserver { ... }

source-address ... ;

doctype-identification { ... }

}

This section defines general TNS-wide proxy attributes.

The `proxy` section is derived from `clone-application` section prototype. For detail description of it, see above.
Changes to the `proxy` section:

Addresses to listen on must be specified.

Added items & subsections:

`listen-on` {

* non-transparent ... ;

* transparent ... ;

}

The `listen-on` section is derived from `listen-on` section prototype. For detail description of it, see listen-on(5).
Changes to the `listen-on` section:

At least one address to listen on must be specified.

`tcpserver` {

queue-size ... ;

init-children ... ;

max-children ... ;

max-children-per-ip ... ;

min-idle ... ;

max-idle ... ;

parent-cycle ... ;

info-cycle ... ;

min-start-rate ... ;

max-start-rate ... ;

kill-rate ... ;

fork-wait ... ;

fork-retries ... ;

lock ... ;

alt-lock ... ;

listener ... ;

conn-rate ... ;

conn-rate-per-ip ... ;

conn-rate-table ... ;

terminate-wait ... ;

}

The `tcpserver` section is derived from `tcpserver` section prototype.
For detail description of it, see tcpserver(5).

`udpserver` {

max-sessions ... ;

}

The `udpserver` section is derived from `udpserver` section prototype.
For detail description of it, see udpserver(5).

`source-address` [`client`] [`addr4` *addr4*] [`addr6` *addr6*] `cluster` [*cluster*];
`source-address` [`client`] [`addr4` *addr4*] [`addr6` *addr6*] [`physical`];
`source-address` [`client`] [`addr4` *addr4*] [`addr6` *addr6*] `no-fallback`;

Source address for outgoing connections to servers.

If omitted, the proper address of the proxy will be used, i.e. in the case of a cluster, the cluster address will be used.

If not specified by the SOURCE-PORT item, a generic port will be used.

The elements entered within this item will be used by the proxy until the first of them is applicable:

- The CLIENT keyword means the original client IP address is used. This mode will be succesful in all cases except mismatch of IP address families.

- The ADDR4/ADDR6 keyword-value pairs mean that the specified address is used for a connection of corresponding address family.

- The CLUSTER keyword means that one of cluster addresses will be used. By default, the main address of the bridge is used, however, any preferred alias address can be listed in the cluster list.- The PHYSICAL option means that the address of the physical interface is used instead of the cluster one.

- The DEFAULT option means the default behavior - i.e. using of the physical address.

- The NO-FALLBACK option means that if no other way of setting the address is acceptable, the session is rejected. Without this option, the system tries to find a suitable source IP address automatically.

`client` (type: `key`, optional)

`addr4` *addr4* (type: `host`, optional, default: [0.0.0.0])

`addr6` *addr6* (type: `host`, optional, default: [::])
<branching element> (type: `source-address-mode`, optional, default: physical)
***cluster* (type: `host-list`, optional, default: {})**

Constraints:

Address family must respect the element's address family..

doctype-identification {

* order ... ;

mime-types ... ;

magic ... ;

}

The `doctype-identification` section is derived from `doctype-identification` section prototype. For detail description of it, see above.

[End of section `proxy` description.]

SEE ALSO

configuration(7), acl(5), common(5), ipc(5), listen-on(5), log(5), monitoring(5), netio(5), resolver(5), source-address(5), tcpserver(5), udpserver(5)

NAME

atr — format of atr component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **atr** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **atr** configuration directives:

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

auth-method (see **auth**(5))

source-address-mode (see **source-address**(5))

transparency (see **acl**(5))

user-auth-spec (see **acl**(5))

doctype-ident-method (see **acl**(5))

listen-on-sock (see listen-on(5))

atr-strategy (name-usage obligatory)

Strategy for address selection.

all

All available addresses added to response.

first

First available address sent in response.

highest

Available address with highest ratio is sent in response.

cyclic

Available addresses are alternated in a circle.

random

Available addresses are alternated randomly by ratio.

atr-fallback (name-usage obligatory)

Fallback mode for no available address.

no-data

Response with NoError code and no ANSWER is returned.

first

Choose first address of requested type despite state.

ITEMS AND SECTIONS

Configuration of **atr** library component consists of following prototypes:
* atrmon *name* { ... }

Description:

atrmon *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

run-block-sigalrm ... ;

listen-on { ... }

client-conn { ... }

* session-acl *name* { ... }

* request-acl *name* { ... }

}

Adaptive Transport Routing Monitor configuration.

The `atrmon` section is derived from `proxy` section prototype. For detail description of it, see application(5).

Changes to the `atrmon` section:

Section `stats-daily` is not valid.

Section `stats-weekly` is not valid.

Section `stats-monthly` is not valid.

Item `idle-timeout` is not valid.

Section `tcpserver` is not valid.

Section `udpserver` is not valid.

Item `source-address` is not valid.

Section `doctype-identification` is not valid.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one REQUEST-ACL must be specified.

Section `monitoring` (see monitoring(5))

Item `user` is not valid.

Item `aproxy-user` is not valid.

Item `data` used as `query`.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 53.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 53.

Added items & subsections:

`client-conn` {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Client connection options.

The `client-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

Item `recv-bufsize` (see netio(5))

Element `bytes` is optional, default: 512.

Input buffer size must be at least 512B.

Item `send-timeout` (see netio(5))

Element `seconds` is optional, default: 60.

Item `send-bufsize` (see netio(5))

Output buffer size must be at least 512B.

`session-acl` *name* {

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

idle-timeout-peer ... ;

source-address ... ;

neg-resp-ttl ... ;

}

The first level ACL decides only between acceptation and denial of the incoming datagram/connection.

The `session-acl` section is derived from `acl-1` section prototype. For detail description of it, see acl(5).

Changes to the `session-acl` section:

Item `user` is not valid.

Item `auth` is not valid.

Item `idle-timeout` is not valid.

Item `plug-to` is not valid.

Added items & subsections:

`neg-resp-ttl` [*seconds*];

TTL for negative responses.

If ATR monitor sends NXDomain response code for name from known domain, it can send a SOA record in AUTHORITY section. This record causes caching of

this negative answer in clients (nameservers) for the time used as the TTL of the SOA RR. This value can be defined by this item.
Setting the TTL to zero means switching this feature off. Use this with care because it can cause ineffectivity of DNS service.

seconds **(type: `uint32`, optional, default: 60)**

[End of section `atrmon.session-acl` description.]

`request-acl` ***name*** **{**

* from ... ;
* time ... ;
  time-period-set { ... }
* session-acl ... ;
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
  client-altq ... ;
  name ... ;
* nameserver ... ;
* address *name* { ... }
  strategy ... ;
  fallback ... ;
  neg-resp-ttl ... ;

}

The second level ACL decides how to handle particular DNS query/notify request.

The `request-acl` section is derived from `acl-2` section prototype.
For detail description of it, see acl(5).
Changes to the `request-acl` section:

Item `server` is not valid.
Item `user` is not valid.
Item `parent-acl` used as `session-acl`.
NAME must be specified.
At least one ADDRESS or NAMESERVER must be specified.
ADDRESS and NAMESERVER are mutually exclusive.

Added items & subsections:

`client-altq` ***altq*** **[`paltq`** ***paltq*****];**

ALTQ queues for data sent to client.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**
queue name

paltq *paltq* **(type: name of pf-queue, see pf-queue(5), optional, default: NULL)**

priority queue name (if set, used for TCP ACK without data)

name *name*;

Entry condition - Query domain name.

name **(type: str)**

nameserver *ttl host*;

NS RR data.

ttl **(type: uint32)**

Time-to-live value of DNS RR.

host **(type: str)**

address *name* {

data ... ;

ratio ... ;

* ping-group *name* { ... }

down-timeout ... ;

up-timeout ... ;

}

Single address for resolution and availability check.

Constraints:

Host DATA must be specified.

Items & subsections:

data *ttl addr*;

Data for particular answer.

ttl **(type: uint32)**

Time-to-live value of DNS RR.

addr **(type: host)**

IPv4/6 address of A/AAAA DNS RR.

ratio [*prty*];

Priority (relative frequency) of this address in responses.

prty **(type: uint8, optional, default: 100)**

ping-group *name* {

timeout ... ;

* host ... ;

}

Group of hosts being pinged.

Every defined group within an ADDRESS section must be alive to add this address to DNS responses.

The `ping-group` section is derived from `ping-group` section prototype. For detail description of it, see ping(5).

`down-timeout` [*sec*];

Cluster down timeout.

At least one tested IP group must be inaccessible for this time in order to switch the cluster interfaces "down".

sec **(type: `uint32`, optional, default: 0)**

Timeout in seconds, zero means immediate action.

`up-timeout` [*sec*];

Cluster up timeout.

All tested IP groups must be accessible for this time in order to switch the cluster interfaces "up".

sec **(type: `uint32`, optional, default: 0)**

Timeout in seconds, zero means immediate action.

[End of section `atrmon.request-acl.address` description.]

`strategy` [*mode*];

Address selection strategy.

mode **(type: `atr-strategy`, optional, default: all)**

`fallback` [*mode*];

Response policy when no address alive.

mode **(type: `atr-fallback`, optional, default: no-data)**

`neg-resp-ttl` [*seconds*];

TTL for negative responses.

If ATR monitor sends negative QUERY responses (NoError response code with no answer records or NXDomain response code), it can send a SOA record in AUTHORITY section. This record causes caching of this negative answer in clients (nameservers) for the time used as the TTL of the SOA RR. This value can be defined by this item.

Setting the TTL to zero means switching this feature off. Use this with care because it can cause ineffectivity of DNS service.

seconds **(type: `uint32`, optional, default: 60)**

[End of section `atrmon.request-acl` description.]

[End of section `atrmon` description.]

SEE ALSO

configuration(7), acl(5), application(5), auth(5), common(5), listen-on(5), log(5), monitoring(5), netio(5), pf-queue(5), ping(5), source-address(5), time(5)

NAME

atrmon.cfg — format of atrmon program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **atrmon.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **atrmon.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see **radius**(5))

ldap-tls-reqcert-mode (see **ldap**(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

atr-strategy (see atr(5))

atr-fallback (see atr(5))

ITEMS AND SECTIONS

Program **atrmon** recognizes following items and sections:

admin ... ;
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }

sysctl { ... }

use-resolver ... ;
* atrmon *name* { ... }

ipv6-mode ... ;

Description:

admin *system* [*contact*];

Firewall administrator and contact e-mail addresses.

system **(type: str)**

The technical administrator(s) of the system; an address or set of comma separated adresses of persons responsible for system maintenance.

contact **(type: str, optional, default: <NULL>)**

The policy administator; an address of person responsible for system configuration. If not defined, the technical administration is used instead.

Constraints:

Administrator contact must comply with RFC.

interface *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The interface section is derived from interface section prototype.
For detail description of it, see interface(5).

ldap-client-auth *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

```
}
```

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

`oob-auth` *name* {

```
  method ... ;
  max-sessions ... ;
  max-user ... ;
  max-groups ... ;
  truncate-groups ... ;
  file ... ;
  lock ... ;
}
```

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

`pf-queue` *name* {

```
  parent ... ;
  bandwidth ... ;
  priority ... ;
  qlimit ... ;
  cbq { ... }
  priq { ... }
  hfsc { ... }
}
```

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

`radius-client` *name* {

```
  nas ... ;
  groups ... ;
* server ... ;
}
```

The `radius-client` section is derived from `radius-client` section prototype. For detail description of it, see radius(5).

resolver *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

shared-dir *name* {

path ... ;

}

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

shared-file *name* {

path ... ;

format ... ;

}

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

sysctl {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`atrmon` *name* {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

run-block-sigalrm ... ;

listen-on { ... }

client-conn { ... }

* session-acl *name* { ... }

* request-acl *name* { ... }

}

The `atrmon` section is derived from `atrmon` section prototype. For detail description of it, see atr(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), atrmon(8), acl(5), atr(5), auth(5), common(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), pf-queue(5), radius(5), resolver(5), source-address(5), sysctl(5), time(5)

NAME

auth — format of auth component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **auth** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **auth** configuration directives:

obligation (see **common**(5))

lock-type (see **ipc**(5))

auth-method (name-usage obligatory)

Authentication Method.

This type is used to specify authentication method used by proxy.

none

Free access, no authentication needed.

passwd

Access controlled by password file.

radius

Access controlled by radius client/server.

ldap

Access controlled by ldap client/server.

ext

Access controlled by external utility.

oob

Access controlled by out-of band authentication.

oob-authentication-method (name-usage obligatory)

Out-of-band authentication method.

html-form

A user fills in an authentication form in a web browser.

external

A list of authenticated users is provided by an external source, e.g., a Samba server.

ext-mod

Modifications of the list of authenticated users is controlled by an external source.

user-match-mode (name-usage obligatory)

ACL matching modes of authenticated usernames.

short

Only username w/o domainname/realm is being matched in ACLs.

long

Full username with domainname/realm is being matched in ACLs.

ITEMS AND SECTIONS

Configuration of **auth** library component consists of following prototypes:
* oob-auth *name* { ... }
auth ... ;
user-match ... ;

Description:

oob-auth *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

Parameters of OOB authentication.

Items & subsections:

method [html-form];

method external [*ldap* **[even-no-group]];**

method ext-mod [*ldap* **[even-no-group]];**

OOB authentication method

<branching element> (type: **oob-authentication-method**, optional, default: html-form)
ldap (type: **name** of **ldap-client-auth**, see ldap(5), optional, default: NULL)
Ask an LDAP server for a list of groups each user belongs to.

even-no-group (type: **key**, optional)
Add the users even if the ldap search fails for the user.

max-sessions [*val*];

Maximum number of simultaneously active OOB authentication sessions.

val (type: **uint16**, optional, default: 1500)

Constraints:
MAX-SESSIONS must be nonzero.

max-user [*val*];

Maximum length of a user name

val (type: **uint16**, optional, default: 48)

Constraints:
MAX-USER must be nonzero.

max-groups [*val*];

Maximum space used by a list of groups for a single user. Each group name of length L takes L+1 characters from this space.

val (type: **uint16**, optional, default: 2048)

Constraints:
MAX-GROUPS must be nonzero.

truncate-groups;

If used, a too long list of groups is truncated. If unused, the user cannot authenticate if its list of groups does not fit to space allocated according to MAX-GROUPS.

file [*path*];

OOB session table file.

path (type: **str**, optional, default: "/tmp/oob-auth")

lock none;

lock semaphore;

lock lock2 [*path*];

lock [**multilock2**] [*path*];

An alternative implemetation of locks.
<branching element> (type: **lock-type**, optional, default: multilock2)
path (type: **str**, optional, default: <NULL>)
If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is

the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

[End of section `oob-auth` description.]

`auth none`;

`auth passwd` *file*;

`auth radius` *client*;

`auth ldap` *ldap*;

`auth ext` *file*;

`auth oob` *oob* [*mode* [**`loose`**]];

Authentication method and attributes specification.

For more details, see auth(7).

<**branching element**> (**type: `auth-method`**)

file (**type: `str`**)

Password/utility file name.

client (**type: `name` of `radius-client`**, see **radius**(5))

RADIUS client configuration name.

ldap (**type: `name` of `ldap-client-auth`**, see **ldap**(5))

LDAP client configuration parameters.

oob (**type: `name` of `oob-auth`**, see above)

OOB authentication parameters.

mode (**type: `obligation`**, optional, default: required)

`loose` (**type: `key`**, optional)

`user-match` [*mode*];

ACL matching mode of authenticated usernames.

mode (**type: `user-match-mode`**, optional, default: short)

SEE ALSO

configuration(7), common(5), ipc(5), ldap(5), radius(5), auth(7)

NAME

clear-web-db — format of clear-web-db component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **clear-web-db** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **clear-web-db** configuration directives:

lock-type (see **ipc**(5))

clear-web-db-category (name-usage obligatory)

Categories of web servers recognized by the Clear Web DataBase.

advertisement

alcohol-tobacco

arts

cars-vehicles

banking

brokers

building-home

business

chats-blogs-forums

communications

crime

education

entertainment

environment

erotic-adult-nudity

extreme-hate-violence

fashion-beauty

food-restaurants

foundations-charity-social-services

gambling

games

government

hacking-phishing-fraud

health-medicine

hobbies

humour-cool

it-hardware-software

it-services-internet

illegal-drugs

instant-messaging

insurance

job-career

kids-toys-family

military-guns

mobile-phones-operators

music-radio-cinema-tv

news-magazines

peer-to-peer

personal-dating-lifestyle

politics-law

pornography

portals-search-engines

proxies

real-estate

regional

religious-spirituality

sale-auctions

sects

sex-education

shopping

social-networks

sports

streaming-broadcasting

swimwear-intimate

translation-services

travelling-vacation

uploading-downloading

warez-piracy

web-based-mail

web-hosting

money-financial

internal-servers

unknown

clear-web-db-match-mode (name-usage obligatory)

How to match Clear Web DataBase categories.

any

At least one category of the request URI matches the condition.

all

All categories in the condition are matched by categories of the request URI.

subset

All categories of the request URI match the condition.

exact

Categories of the request URI are exactly those in the condition.

ITEMS AND SECTIONS

Configuration of **clear-web-db** library component consists of following prototypes:

clear-web-db { ... }
cwcatd-wakeup ... ;
cwcatd-retry ... ;
clear-web-db-bypass { ... }
* clear-web-db-match ... ;

Description:

clear-web-db {

internal-servers ... ;

db ... ;

lock ... ;

local-db { ... }

}

Global settings for web filtration based on URL. Enables periodic updates of the database.

Items & subsections:

internal-servers [private-ip] *server*;

List of servers, that are categorized as INTERNAL-SERVERS, in addition to RFC 1918 IP addresses. Domain names will be resolved to IP addresses and compared to the real IP address of the server. Regexp will be compared to the target hostname (without resolving). See more information in host-matching(7).

private-ip (type: key, optional)

Categorize private IP addresses from RFC 1918 (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16) as internal servers.

server **(type: host-set)**

Categorize matching servers as internal. See host-matching(7).

db [*dir***];**

Local directory used to store Clear Web DataBase data.

dir **(type: str, optional, default: "/data/var/clear-web-db")**

lock none;

lock semaphore;

lock lock2 [*path***];**

lock [multilock2] [*path***];**

An alternative implemetation of locks.

<branching element> (type: lock-type, optional, default: multi-lock2)

path **(type: str, optional, default: <NULL>)**

If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

local-db {

file ... ;

timeout-search ... ;

timeout-enqueue ... ;

timeout-cwcatd ... ;

}

Locally created Clear Web database. It contains categories for automatically categorized URLs. If this section is missing in the configuration, a proxy will not use the local database.

Items & subsections:

`file` [*path*];

Name of the database file

***path* (type: `str`, optional, default: "/data/var/clear-web-local/clear-web-local.sqlite")**

`timeout-search` [*sec*];

Timeout for searching the database. A proxy will wait up to this number of seconds when searching for categories of an URL and another process keeps the database locked.

***sec* (type: `fract`, optional, default: 0.100)**

`timeout-enqueue` [*sec*];

Timeout for inserting an URL to be categorized into the categorizer queue. A proxy will wait up to this number of seconds when inserting an URL and another process keeps the database locked.

***sec* (type: `fract`, optional, default: 0.100)**

`timeout-cwcatd` [*sec*];

The cwcatd daemon will wait up to this number of seconds before aborting the current operation when another process keeps the database locked.

***sec* (type: `fract`, optional, default: 5)**

[End of section `clear-web-db.local-db` description.]

[End of section `clear-web-db` description.]

`cwcatd-wakeup` [*sec*];

Period (in seconds) of waking up of the categorization daemon and checking the queue of categorization requests. In addition, the daemon is awaken by a signal immediately after a new request is enqueued.

***sec* (type: `uint16`, optional, default: 60)**

`cwcatd-retry` [*sec*];

Time (in seconds) after which a failed automatic categorization will be retried.

***sec* (type: `uint32`, optional, default: 3600)**

`clear-web-db-bypass` {

status ... ;

cookie ... ;

activation ... ;

duration ... ;

}

Enable the bypass functionality (time-limited access to a page blocked by the Clear Web DataBase).

Items & subsections:

`status` **[*code*];**

Status code returned when the bypass is inactive. Default is 403 Forbidden.

code **(type: `uint16`, optional, default: 403)**

`cookie` **[*name*];**

Use cookies for bypass management. Default is to used a table of client IP addresses with enabled bypass.

name **(type: `str`, optional, default: "Kernun-ClearWebDB-Bypass")**

bypass cookie name

`activation` **[*sec*];**

Maximum time for clicking on bypass activation link after the bypass activation page is shown. If the user clicks the link later, the proxy will return the activation page again.

sec **(type: `uint8`, optional, default: 30)**

`duration` **[*sec*];**

Duration of allowed access.

sec **(type: `uint16`, optional, default: 60)**

[End of section `clear-web-db-bypass` description.]

`clear-web-db-match` **[`any`]** ***categories-set*;**

`clear-web-db-match` **`all`** ***categories-list*;**

`clear-web-db-match` **`subset`** ***categories-set*;**

`clear-web-db-match` **`exact`** ***categories-list*;**

Clear Web Matching Control.

This item is used as an ACL entry condition for a URL based on Clear Web category matching.

<branching element> (type: `clear-web-db-match-mode`, optional, default: any)

categories-set **(type: `clear-web-db-category-set`)**

categories-list **(type: `clear-web-db-category-list`)**

SEE ALSO

configuration(7), ipc(5), host-matching(7)

NAME

common — format of common component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **common** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **common** configuration directives:

enabling (name-usage obligatory)

General Enabling/Disabling Enumeration.

disable

enable

yes-no (name-usage obligatory)

General Yes/No Enumeration

no

yes

language (name-usage obligatory)

National Language Support - Language Setting.

EN

English

CZ

Czech (UTF-8)

nls (name-usage obligatory)

National Language Support - Language and Charset.

EN

English

CZ

Czech (default charset)

CZ-ASCII

Czech, without diacritics

CZ-ISO-8859-2

Czech, ISO-Latin-2

CZ-WINDOWS-1250

Czech, Windows-1250

on-off (name-usage optional)

Features switching on/off.

off (0)

on (1)

genesis (name-usage obligatory)

General Genesis (Static/Dynamic) Enumeration.

static

dynamic

permission (name-usage obligatory)

Permission/Denial Methods.

permit

particular option is permitted

deny

particular option is rejected but ignored

abort

particular option is rejected and session is aborted

max-setting (name-usage obligatory)

Ways to Set Maximum.

max

particular limitation will be set, values have maximum

any

particular limitation will be ignored, any value is valid

direction (name-usage obligatory)

General Traffic Direction Enumeration.

download

Transfer from server to client.

upload

Transfer from client to server.

name-selection (name-usage obligatory)

Methods to select object.

any

Setting of particular object is not required, anyone is correct.

name

Setting of object by its name in configuration.

destination (name-usage obligatory)

Destination (remote peers or nets).

host

net

default

ip-version (name-usage optional)

IP version.

ipv4 (4)

ipv6 (6)

address-family (name-usage obligatory)

Socket Address Family.

inet

inet6

unix

osi4-proto (name-usage obligatory)

Transport Layer Protocol.

default

tcp

udp

tcp-udp

in-out (name-usage obligatory)

Interface inbound/outbound Direction.

in

out

both

report-mode (name-usage optional)

Process stdout/stderr control.

nothing (0)

out (1)

err (2)

all (3)

periodicity (name-usage obligatory)

Time period types.

daily

weekly

monthly

time-cond (name-usage obligatory)

Time condition types.

anytime

No condition on time applied

daily

weekly

monthly

zip-mode (name-usage obligatory)

Logfile zipping mode.

plain

gzip

bzip2

obligation (name-usage obligatory)

Modes of special features usage.

This enumeration is used when some feature (like authentication, SSL etc.) can be required or only allowed by admin's decision.

required

Feature is mandatory

allowed

Feature is optional

range-op (name-usage obligatory)

Range Comparison Operator.

unknown

Tested value is not known.

lt

Tested value is lower than the configuration limit.

le

Tested value is lower than or equal to the configuration limit.

eq

Tested value is equal to the configuration limit.

ne

Tested value is not equal to the configuration limit.

gt

Tested value is greater than the configuration limit.

ge

Tested value is greater than or equal to the configuration limit.

in

Tested value is in between the configuration limits (borders OK).

ni

Tested value is not in between the configuration limits (borders OK).

inline-file-format (name-usage obligatory)

In-line File Formats.

text

Regular text, lines will be trimmed and quoted.

raw

Raw text, lines are only quoted, no comments allowed.

native

Native CML values, lines are used as-is.

ip-addr

IP addresses with or without mask, but without brackets.

regexp

Regular expressions without slashes

yes-no-always (name-usage obligatory)

Yes/No Enumeration with Always option.

Represents a YES-NO value that is tied to a certain condition, usually to a component or function being configured.

no

Always NO, even when the condition is true

yes

YES when the condition is true, NO when the condition is false

always

Always YES, even when the condition is false

task-frequency (name-usage obligatory)

Task frequency

daily

Run the task once a day.

hourly

Run the task once an hour.

every

Run the task every PERIOD minutes.

raw

Raw crontab period specification.

manually

No automatically scheduled refresh.

ITEMS AND SECTIONS

Configuration of **common** library component consists of following prototypes:

admin ... ;
ipv6-mode ... ;
phase ... ;
* cfg-tag ... ;
* range-cond ... ;
* set-var ... ;
* mime-type-check ... ;
* shared-file *name* { ... }
* shared-dir *name* { ... }

rotate-file ... ;
cron-schedule ... ;

Description:

admin *system* [*contact*];

Firewall administrator and contact e-mail addresses.

system **(type: `str`)**

The technical administrator(s) of the system; an address or set of comma separated adresses of persons responsible for system maintenance.

contact **(type: `str`, optional, default: <NULL>)**

The policy administator; an address of person responsible for system configuration. If not defined, the technical administration is used instead.

Constraints:

Administrator contact must comply with RFC.

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

`phase` [*number*];

Application Startup Phase.

number **(type: `uint8`, optional, default: 50)**

Phase number; the lower one, the earlier start.

`cfg-tag` *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

`range-cond unknown`;

`range-cond lt` *limit*;

`range-cond le` *limit*;

`range-cond eq` *limit*;

`range-cond ne` *limit*;

`range-cond gt` *limit*;

`range-cond ge` *limit*;

range-cond in *lower upper*;

range-cond ni *lower upper*;

Range Testing Condition.

<**branching element**> (**type: range-op**)

limit (**type: uint64**)

Tested value limitation.

lower (**type: uint64**)

Tested value lower bound.

upper (**type: uint64**)

Tested value upper bound.

set-var *name value*;

Shell-like variable setting.

name (**type: str**)

Variable name.

value (**type: str**)

Variable value.

Constraints:

Variable name must contain alphanumeric chars only.

mime-type-check *type*;

Document MIME Type and Subtype Testing Checking.

type (**type: str-set**)

Set of type/subtype string definition.

If a regexp is part of the set, then this regexp is checked to match with type/subtype specification. Beware of escaping the slash, if present (write /...\/.../).

If a string is part of the set, then it must contain at most one slash. If the slash is not present, string is compared with document type only (not the subtype). If the slash is present, then pattern is checked to match with type/subtype specification.

shared-file *name* {

path ... ;

format ... ;

}

Shared file definition.

Constraints:

Pathname must be specified.

Items & subsections:

path *name*;

Path specification.

This path is valid in the environment, where applied:

- within CML it means path on the filesystem where run; if relative, it is related to the configuration directory
- within firewall configuration files it means path on the firewall (cannot be relative).

Thus, value of this item can differ between source CML file and target CFG files and CML command /GENERATE copies these files into destination SYSTEM-* tree.

name **(type: str)**
Path to the file.

format [*type*];

Inline file format.

If the shared file is used as inline file ("< NAME" in list) this item defines line modifications.

type **(type: inline-file-format, optional, default: text)**

[End of section `shared-file` description.]

shared-dir *name* **{**

path ... ;

}

Shared directory definition.

Constraints:

Pathname must be specified.

Items & subsections:

path *name*;

Path specification.

This path is valid in the environment, where applied:

- within CML it means path on the filesystem where run; if relative, it is related to the configuration directory
- within firewall configuration files it means path on the firewall (cannot be relative).

Thus, value of this item can differ between source CML file and target CFG files and CML command /GENERATE copies these directories into destination SYSTEM-* tree.

name **(type: str)**
Path to the directory.

[End of section `shared-dir` description.]

rotate-file [**user** *user*] [**group** *group*] [**mode** *mode*] [**count** *count*] [**size** *size*] [*when* [*zip*]];

Log file rotation description.

Use the SIZE elem if log file size criterion required. Use the WHEN elem if periodical rotation required. If used both SIZE and WHEN elems, the log file is rotated at a proper time only if size limit is reached.

user *user* (type: **str**, optional, default: <NULL>)

Log file owner - user.

group *group* (type: **str**, optional, default: "wheel")

Log file owner - group.

mode *mode* (type: **uint16**, optional, default: 640)

Log file permissions.

count *count* (type: **uint16**, optional, default: 31)

Number of days being archived.

size *size* (type: **uint16**, optional, default: 0)

Size limit for rotation in KB (ignore log file size if omitted).

when (type: **time-cond**, optional, default: anytime)

Rotation periodicity (use SIZE condition if omitted).

zip (type: **zip-mode**, optional, default: bzip2)

Zipping mode.

Constraints:

Use either size criterion or defined periodicity.

cron-schedule daily [**time** *time*] [**report** *report*];

cron-schedule hourly [**minute** *minute*] [**report** *report*];

cron-schedule [**every**] [**period** *period*] [**at** *at*] [**report** *report*];

cron-schedule raw raw *raw* [**report** *report*];

cron-schedule manually;

Parameters for scheduling a cron task.

<branching element> (type: **task-frequency**, optional, default: every)

raw *raw* (type: **str**)

Raw line to be placed into crontab. First 5 columns (the time specification) must be specified.

minute *minute* (type: **time**, optional, default: 0)

Starting time of task (mm, hour ignored).

`time` *time* (type: `time`, optional, default: 415)

Starting time of task (hhmm).

`period` *period* (type: `uint8`, optional, default: 15)

Run the task every PERIOD minutes (mm, hours ignored).

`at` *at* (type: `uint8`, optional, default: 0)

Starting time of task (mm, hours ignored)

`report` *report* (type: `report-mode`, optional, default: nothing=0)

Task output (stdout and stderr) delivery.

SEE ALSO

configuration(7), logging(7)

NAME

cwcatd.cfg — format of cwcatd program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **cwcatd.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **cwcatd.cfg** configuration directives:

lock-type (see **ipc**(5))

enabling (see **common**(5))

yes-no (see **common**(5))

ip-version (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

source-address-mode (see **source-address**(5))

log-in-vain-proto (see **sysctl**(5))

blackhole-proto (see **sysctl**(5))

ITEMS AND SECTIONS

Program **cwcatd** recognizes following items and sections:

clear-web-db { ... }
* resolver *name* { ... }
sysctl { ... }
use-resolver ... ;

* cwcatd *name* { ... }

Description:

clear-web-db {

internal-servers ... ;

db ... ;

lock ... ;

local-db { ... }

}

The clear-web-db section is derived from clear-web-db section prototype. For detail description of it, see clear-web-db(5).

resolver *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The resolver section is derived from resolver section prototype. For detail description of it, see resolver(5).

sysctl {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`cwcatd` *name* {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

wakeup ... ;

retry ... ;

sitemarker ... ;

}

The `cwcatd` section is derived from `cwcatd` section prototype. For detail description of it, see above.

SEE ALSO

configuration(7), cwcatd(8), clear-web-db(5), common(5), ipc(5), log(5), resolver(5), source-address(5), sysctl(5)

NAME

dhcp-server — format of dhcp-server component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **dhcp-server** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

ITEMS AND SECTIONS

Configuration of **dhcp-server** library component consists of following prototypes:

* dhcp-peer ... ;
* dhcp-domain ... ;
  dhcp-server-common { ... }
  dhcp-server { ... }
  dhcp6-server { ... }

Description:

dhcp-peer *host*;

Peer host offered by DHCP to clients.

host (**type: host**)

dhcp-domain *name*;

Domain name search (option DOMAIN-NAME/DHCP6.DOMAIN-SEARCH).

name (**type: str**)

dhcp-server-common {

phase ... ;

* tag ... ;

lease-file ... ;

default-lease-time ... ;

max-lease-time ... ;

* domain ... ;

* name-server ... ;

* time-server ... ;

* router ... ;

* raw ... ;

* subnet *name* { ... }

failover { ... }

}

DHCP server configuration.

Global parameters are defined here, MAC to IP address assignment and special options are set within HOSTS-TABLE section. There are separate servers for IPv4 (section DHCP-SERVER) and IPv6 (section DHCP6-SERVER)

For configuration attributes details, see dhcpd.conf(5).

Items & subsections:

`phase` **[***number***];**

Application Startup Phase.

number **(type: `uint8`, optional, default: 40)**

Phase number; the lower one, the earlier start.

`tag` *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

`lease-file` *path*;

Lease file location.

path **(type: `str`)**

`default-lease-time` *seconds*;

Default lease time.

seconds **(type: `uint32`)**

`max-lease-time` *seconds*;

Maximum lease time.

seconds **(type: uint32)**

domain *name*;

Domain name search (option DOMAIN-NAME/DHCP6.DOMAIN-SEARCH).

If omitted the system domain name is used.

name **(type: str)**

name-server *host*;

Nameserver (option DOMAIN-NAME-SERVERS/DHCP6.NAME-SERVERS).

host **(type: host)**

time-server *host*;

Time server (option TIME-SERVERS).

host **(type: host)**

router *host*;

Router (option ROUTERS).

host **(type: host)**

raw *line*;

Raw global line.

line **(type: str)**

subnet *name* {

address ... ;

* range ... ;

* domain ... ;

* name-server ... ;

* time-server ... ;

* router ... ;

disable-failover ... ;

* raw ... ;

}

Subnet definition.

Constraints:

Address must be specified.

Items & subsections:

address *net*;

net **(type: net)**

range *lo hi*;

lo **(type: host)**

hi **(type: host)**

domain *name*;

Domain name search (option DOMAIN-NAME/DHCP6.DOMAIN-SEARCH). If omitted the system domain name is used.

name **(type: str)**

name-server *host*;

Nameserver (option DOMAIN-NAME-SERVERS/DHCP6.NAME-SERVERS).

host **(type: host)**

time-server *host*;

Time server (option TIME-SERVERS).

host **(type: host)**

router *host*;

Router (option ROUTERS).

host **(type: host)**

disable-failover;

Disable DHCP failover.

raw *line*;

Raw subnet line.

line **(type: str)**

[End of section `dhcp-server-common.subnet` description.]

failover {

primary ... ;

secondary ... ;

max-response-delay ... ;

max-unacked-updates ... ;

mclt ... ;

split ... ;

lbms ... ;

* raw ... ;

}

DHCP failover parameters.

Constraints:

PRIMARY and SECONDARY must be specified.

Items & subsections:

primary *addr* **[port** *port***];**

Primary server definition.

addr **(type: host)**

Listening address of primary server.

port *port* **(type: port, optional, default: 519)**

Listening port of secondary server.

secondary *addr* [**port** *port*];
Secondary server definition.
addr (type: **host**)
Listening address of secondary server.
port *port* (type: **port**, optional, default: **520**)
Listening port of secondary server.

max-response-delay [*sec*];
Peer dead timeout (local parameter).
sec (type: **uint16**, optional, default: **60**)

max-unacked-updates [*number*];
Unresponded messges limit (remote parameter).
number (type: **uint16**, optional, default: **10**)

mclt [*sec*];
Maximum client lead time (primary parameter).
sec (type: **uint16**, optional, default: **3600**)

split [*ratio*];
Load balancing split (primary parameter).
ratio (type: **uint16**, optional, default: **128**)
Constraints:
Maximum value 256 is allowed.

lbms [*sec*];
Load balance maximum seconds.
sec (type: **uint16**, optional, default: **3**)

raw *line*;
Raw failover line.
line (type: **str**)

[End of section `dhcp-server-common.failover` description.]

[End of section `dhcp-server-common` description.]

dhcp-server {
phase ... ;
* tag ... ;
lease-file ... ;
default-lease-time ... ;
max-lease-time ... ;
* domain ... ;
* name-server ... ;
* time-server ... ;

* router ... ;

* raw ... ;

* subnet *name* { ... }

  failover { ... }

}

The `dhcp-server` section is derived from `dhcp-server-common` section prototype. For detail description of it, see above.
Item `lease-file` (see above)

Element `path` is optional, default: "/data/var/db/dhcpd.leases".

`dhcp6-server` {

  phase ... ;

* tag ... ;

  lease-file ... ;

  default-lease-time ... ;

  max-lease-time ... ;

* domain ... ;

* name-server ... ;

* raw ... ;

* subnet *name* { ... }

}

The `dhcp6-server` section is derived from `dhcp-server-common` section prototype. For detail description of it, see above.
Changes to the `dhcp6-server` section:

Item `time-server` is not valid.

Item `router` is not valid.

Section `failover` is not valid.

Item `lease-file` (see above)

Element `path` is optional, default: "/data/var/db/dhcpd6.leases".

Section `subnet` (see above)

Item `time-server` is not valid.

Item `router` is not valid.

Item `disable-failover` is not valid.

SEE ALSO

configuration(7), dhcpd.conf(5)

NAME

dns-proxy — format of dns-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **dns-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **dns-proxy** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

auth-method (see **auth**(5))

source-address-mode (see **source-address**(5))

transparency (see **acl**(5))

user-auth-spec (see **acl**(5))

doctype-ident-method (see acl(5))

dns-type (see resolver(5))

dns-opcode (see resolver(5))

dns-response (see resolver(5))

dns-qaction (see resolver(5))

dns-raction (see resolver(5))

dns-fake (see resolver(5))

xfr-mode (see resolver(5))

listen-on-sock (see listen-on(5))

dns-name-type (name-usage obligatory)

forward

reverse

ITEMS AND SECTIONS

Configuration of **dns-proxy** library component consists of following prototypes:
* dns-proxy *name* { ... }

Description:

dns-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

doctype-identification { ... }

queue-size ... ;

edns ... ;

dnssec ... ;

cache { ... }

request-timeout ... ;

response-timeout ... ;

query-timeout ... ;

server-dead ... ;

server-retry ... ;

server-proto ... ;

requests-table-size ... ;

sockets-table-size ... ;

internal-request-depth ... ;

adr-reply-limit ... ;

ptr-reply-limit ... ;

client-conn { ... }

server-conn { ... }

* session-acl *name* { ... }

* request-acl *name* { ... }

}

This section defines DNS-proxy attributes.

The `dns-proxy` section is derived from `proxy` section prototype. For detail description of it, see application(5).

Changes to the `dns-proxy` section:

Section `tcpserver` is not valid.

Section `udpserver` is not valid.

Item `source-address` is not valid.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one REQUEST-ACL must be specified.

EDNS must be enabled for DNSSEC.

Item `phase` (see common(5))

Element `number` is optional, default: 20.

Section `monitoring` (see monitoring(5))

Item `user` is not valid.

Item `aproxy-user` is not valid.

Item `data` used as `query`.

Item `idle-timeout` (see application(5))

Element `seconds` is optional, default: 120.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 53.

DNS proxy cannot bind address [0.0.0.0] : 53.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 53.

Added items & subsections:

`queue-size` [*value*];

Queue length for listen(2) syscall.

value **(type: `uint8`, optional, default: 10)**

`edns` [*support*];

EDNS support.

support **(type: `enabling`, optional, default: enable)**

`dnssec` [*support*];

DNSSEC support.

support **(type: `enabling`, optional, default: disable)**

`cache` {

cleanup-period ... ;

max-domains ... ;

max-hosts ... ;

}

Internal nameserver cache attributes.

This cache is used by the proxy for internal purposes only.

Items & subsections:

`cleanup-period` [*seconds*];

Cache cleanup period.

Cache is periodically cleaned, items not used within last cleanup period are removed.

seconds **(type: `uint16`, optional, default: 900)**

max-domains [*items*];

Maximum of domains stored in the cache.

If reached, non-periodical cleanup is done.

items (type: **uint16**, optional, default: **1000**)

max-hosts [*items*];

Maximum of nameservers stored in the cache.

If reached, non-periodical cleanup is done.

items (type: **uint16**, optional, default: **2000**)

[End of section `dns-proxy.cache` description.]

request-timeout [*seconds*];

Timeout to close uncompleted requests and reply by ServFail.

seconds (type: **uint16**, optional, default: **120**)

response-timeout *tmout*;

Timeout to send responses (after expiring no response is sent).

tmout (type: **fract**)

query-timeout [*seconds*];

Initial timeout for querying another server.

seconds (type: **uint16**, optional, default: **2**)

server-dead [*seconds*];

Timeout to assume server is dead.

seconds (type: **uint16**, optional, default: **15**)

server-retry [*seconds*];

Timeout to wait before retry dead server.

seconds (type: **uint16**, optional, default: **15**)

server-proto *ver*;

Allowed protocol for resolving from the root.

If not used, no restriction is applied.

ver (type: **ip-version**)

requests-table-size [*number*];

Number of simultaneous requests.

Requests table contains both client requests (UDP and TCP) and internal requests (nameserver resolving and CNAME querying).

number (type: **uint16**, optional, default: **256**)

Constraints:

Number of requests must not be zero.

sockets-table-size [*number*];

Number of simultaneously opened sockets.

Sockets table contains each socket used for solving of any request in parallel.

number **(type: uint16, optional, default: 0)**

Number of sockets, default is REQUESTS-TABLE-SIZE * 2

internal-request-depth [*number*];

Number of internal requests generated recursively.

number **(type: uint16, optional, default: 50)**

adr-reply-limit [*bytes*];

Size limit for A and AAAA RRs in answer.

Older clients' resolver routines crashes on too long replies. Reasonable value is default one, use zero to switch test off.

bytes **(type: uint16, optional, default: 8192)**

ptr-reply-limit [*bytes*];

Size limit for PTR RRs in answer.

Older clients' resolver routines crashes on too long replies. Reasonable value is default one, use zero to switch test off.

bytes **(type: uint16, optional, default: 512)**

client-conn {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Client connection options.

The client-conn section is derived from sock-opt section prototype. For detail description of it, see netio(5).

Item recv-bufsize (see netio(5))

Element `bytes` is optional, default: 4100.

Input buffer size must be at least 512B.

Item send-timeout (see netio(5))

Element `seconds` is optional, default: 60.

Item send-bufsize (see netio(5))

Output buffer size must be at least 512B.

server-conn {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Server connection options.

The `server-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

Item `recv-bufsize` (see netio(5))

Input buffer size must be at least 512B.

Item `send-timeout` (see netio(5))

Element `seconds` is optional, default: 60.

Item `send-bufsize` (see netio(5))

Element `bytes` is optional, default: 4100.

Output buffer size must be at least 512B.

`session-acl` *name* {

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

source-address ... ;

}

The first level ACL decides only between acceptation and denial of the incoming datagram/connection.

The `session-acl` section is derived from `acl-1` section prototype. For detail description of it, see acl(5).

Changes to the `session-acl` section:

Item `user` is not valid.

Item `auth` is not valid.

Item `idle-timeout` is not valid.

Item `idle-timeout-peer` is not valid.

Item `plug-to` is not valid.

`request-acl` *name* {

* from ... ;

* time ... ;

time-period-set { ... }

* session-acl ... ;
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
* query-name ... ;
* request-type ... ;
* query ... ;
  notify ... ;
* reply ... ;
* fake ... ;
  edns ... ;
  ignore-void-rr ... ;
  ignore-missing-aa ... ;
  ignore-trailer ... ;
  rr-limit ... ;
  xfr-format ... ;
  neg-resp-ttl ... ;
  client-altq ... ;
  server-altq ... ;
}

The second level ACL decides how to handle particular DNS query/notify request.

The `request-acl` section is derived from `acl-2` section prototype.
For detail description of it, see acl(5).
Changes to the `request-acl` section:

Item `server` is not valid.

Item `user` is not valid.

Item `parent-acl` used as `session-acl`.

At least one QUERY/NOTIFY must be specified.

DENY and QUERY/NOTIFY are mutually exclusive.

QUERY/NOTIFY and REPLY set must be consistent.

Cannot mix CNAME and other RR types.

Added items & subsections:

`query-name` [`forward`] ***names*****;**

`query-name reverse` ***nets*****;**

Entry condition - request domain name.

If no QUERY-NAME item is used, ACL matches for all requests independently of domain name.

<branching element> (type: `dns-name-type`, optional, default: forward)

names (type: **str-set**)
Set of domain names and/or regexps.
Query name matches the set if

- matches any regexp-member of the set (regexp must respect a dot appended to every tested name), or
- any name-member of the set is equal to the last part of the query name.

nets (type: **net-set**)
Set of networks.
Query name matches the set if it is a reverse query (*.in-addr.arpa or ip6.arpa) and the corresponding host matches the set.

request-type *op-types* [*rr-types*];
Entry condition - request type.
If no REQUEST-TYPE item is used, ACL matches for all requests independently of request type.

op-types (type: **dns-opcode-set**)
Set of DNS operation types (QUERY and NOTIFY supported).

rr-types (type: **dns-type-set**, optional, default: *)
Set of RR types requested.

query *types* **abort**;

query *types* **deny** [*error-code*];

query *types* **resolve** *ns-list*;

query *types* **forward** *ns-list* [**clear-aa**];

query *types* **fake**;
Allow/deny particular query types for matching domain names. Each query received is checked against QUERY items in order of their occurence in cfg file, and the first matching one is used. If no one matches, request fails.

types (type: **dns-type-set**)
Set of query types matching this QUERY section.

<**branching element**> (type: **dns-qaction**)
DNS operation requested for matching queries.

ns-list (type: **name** of **ns-list**, see resolver(5))
Nameserver address list used resolving or forwarding.

clear-aa (type: **key**, optional)
Clear AA flag in responses being forwarded.

error-code (type: **dns-response**, optional, default: **Refused**=5)
Response code for denied queries.

notify abort;

notify deny [*error-code*];

notify resolve *ns-list*;

notify forward *ns-list*;

notify fake;

Allow/deny NOTIFY requests for matching domain names. If NOTIFY item is not used, request fails.

<branching element> (type: **dns-qaction**)

DNS operation requested by this item.

ns-list (type: **name** of **ns-list**, see resolver(5))

Nameserver addresses.

error-code (type: **dns-response**, optional, default: Refused=5)

Response code used.

Constraints:

NOTIFY requests cannot be handled by RESOLVE or FAKE.

reply *types* **abort;**

reply *types* **deny** [*error-code*];

reply *types* **permit;**

reply *types* **remove;**

Allow/deny particular reply RR types for particular domain names. Each record received is checked against REPLY items in order of their appearance in cfg file, and the first matching one is used. If no one matches, query is denied (FormErr).

types (type: **dns-type-set**)

Set of RR types.

<branching element> (type: **dns-raction**)

DNS operation requested by this item.

error-code (type: **dns-response**, optional, default: Refused=5)

Response code used.

fake *ttl* **0;**

fake *ttl* **a** *addr*;

fake *ttl* **ns** *name*;

fake *ttl* **3;**

fake *ttl* **4;**

fake *ttl* **cname** *name*;

fake *ttl* **6;**

fake *ttl* **7;**

fake *ttl* **8;**

fake *ttl* **9;**

fake *ttl* **10;**

fake *ttl* **11;**

fake *ttl* **ptr** *name*;

fake *ttl* **13;**

fake *ttl* **14;**

fake *ttl* **mx** *prty name*;

fake *ttl* **16**;

fake *ttl* **17**;

fake *ttl* **18**;

fake *ttl* **19**;

fake *ttl* **20**;

fake *ttl* **21**;

fake *ttl* **22**;

fake *ttl* **23**;

fake *ttl* **24**;

fake *ttl* **25**;

fake *ttl* **26**;

fake *ttl* **27**;

fake *ttl* **aaaa** *addr*;

Fake particular RRs in answer.

ttl **(type: uint32)**

Time-to-live value of DNS RR.

<**branching element**> **(type: dns-fake)**

prty **(type: uint16)**

Priority value of MX DNS RR.

name **(type: str)**

Hostname value of various DNS RRs.

addr **(type: host)**

IPv4 address of address type DNS RR.

edns [udp-payload *udp-payload***];**

EDNS support parameters.

udp-payload *udp-payload* **(type: uint16, optional, default: 4096)**

Maximum UDP payload.

ignore-void-rr [*status***];**

Ignore RRs with name having no relationship to the question.

Without this item, the dns-proxy rejects replies having RRs with name that does not appear in other RR, e.g. due to incorrect using of CNAMEs. The server is marked as untrusted.

Some nameservers send more A RRs in the ADDITIONAL section than NS RRs in the AUTHORITY section. This item can prevent to mark them as unusable.

status **(type: enabling, optional, default: enable)**

ignore-missing-aa [*status***];**

Ignore missing AUTHORITATIVE flag in the response.

The algorithm used by the proxy is based on a strategy to find recursively a server for particular domain that admits the fact that it is an authoritative server for this domain. However, some authoritative servers does not set this flags so this algorithm fails.

This flag allows to ignore missing AA flag (for domains where this behavior is known) and trust the answer even without the flag.

status **(type: `enabling`, optional, default: enable)**

`ignore-trailer` [*status*]**;**

Ignore meaningless appendices.

Some servers write errorneous responses containing meaningless byte stream at the end. Proxy ignores this appendix by default. This item can disable this feature and errorneous responses will be treated as errors.

status **(type: `enabling`, optional, default: enable)**

`rr-limit` [*records*]**;**

Maximal allowed number of RRs in response to request.

This limitation is used for DoS prevention, you can set it accordingly to the request type. For zone transfer, this limitation respects total number of RRs in all separated DNS messages belonging to the request.

records **(type: `uint32`, optional, default: 1000)**

`xfr-format` [*mode*]**;**

Zone transfer format.

mode **(type: `xfr-mode`, optional, default: keep)**

`neg-resp-ttl` [*seconds*]**;**

TTL for negative responses.

If the proxy sends negative QUERY responses (NoError response code with no answer records or NXDomain response code), e.g. by faking replies, by filtering out of incoming response records, or by denying of particular query/response, it can send a SOA record in AUTHORITY section. This record causes caching of this negative answer in clients (nameservers) for the time used as the TTL of the SOA RR. This value can be defined by this item.

Setting the TTL to zero means switching this feature off. Use this with care because it can cause ineffectivity of DNS service.

seconds **(type: `uint32`, optional, default: 3600)**

`client-altq` *altq* [**`paltq`** *paltq*]**;**

ALTQ queues for data sent to client.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

`paltq` *paltq* **(type: `name` of `pf-queue`, see pf-queue(5), optional, default: NULL)**

priority queue name (if set, used for TCP ACK without data)

`server-altq` *altq* [**`paltq`** *paltq*]**;**

ALTQ queues for data sent to server.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

`paltq` *paltq* **(type: `name` of `pf-queue`, see pf-queue(5), optional, default: NULL)**

priority queue name (if set, used for TCP ACK without data)

[End of section `dns-proxy.request-acl` description.]

[End of section `dns-proxy` description.]

SEE ALSO

configuration(7), listen(2), acl(5), application(5), auth(5), common(5), listen-on(5), log(5), monitoring(5), netio(5), pf-queue(5), resolver(5), source-address(5), time(5)

NAME

dns-proxy.cfg — format of dns-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **dns-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **dns-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see **radius**(5))

ldap-tls-reqcert-mode (see **ldap**(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

dns-type (see resolver(5))

dns-opcode (see resolver(5))

dns-response (see resolver(5))

dns-qaction (see resolver(5))

dns-raction (see resolver(5))

dns-fake (see resolver(5))

xfr-mode (see resolver(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

dns-name-type (see dns-proxy(5))

ITEMS AND SECTIONS

Program **dns-proxy** recognizes following items and sections:

admin ... ;
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* ns-list *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }

* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* dns-proxy *name* { ... }
  ipv6-mode ... ;

Description:

`admin` ***system*** **[*contact*];**

Firewall administrator and contact e-mail addresses.

system **(type: `str`)**

The technical administrator(s) of the system; an address or set of comma separated adresses of persons responsible for system maintenance.

contact **(type: `str`, optional, default: <NULL>)**

The policy administator; an address of person responsible for system configuration. If not defined, the technical administration is used instead.

Constraints:

Administrator contact must comply with RFC.

`interface` ***name*** **{**

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`ldap-client-auth` *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

`ns-list` *name* {

* server ... ;

}

The `ns-list` section is derived from `ns-list` section prototype. For detail description of it, see resolver(5).

`oob-auth` *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

`pf-queue` *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

`radius-client` *name* {

nas ... ;

groups ... ;

* server ... ;

}

The `radius-client` section is derived from `radius-client` section prototype. For detail description of it, see radius(5).

`resolver` *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`shared-dir` *name* {

path ... ;

}

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

`shared-file` *name* {

path ... ;

format ... ;

}

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`dns-proxy` *name* {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;
monitoring { ... }
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
doctype-identification { ... }
queue-size ... ;
edns ... ;
dnssec ... ;
cache { ... }
request-timeout ... ;
response-timeout ... ;
query-timeout ... ;
server-dead ... ;
server-retry ... ;
server-proto ... ;
requests-table-size ... ;
sockets-table-size ... ;
internal-request-depth ... ;
adr-reply-limit ... ;
ptr-reply-limit ... ;
client-conn { ... }
server-conn { ... }
* session-acl *name* { ... }
* request-acl *name* { ... }
}

The `dns-proxy` section is derived from `dns-proxy` section prototype.
For detail description of it, see dns-proxy(5).

ipv6-mode [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: enabling, optional, default: enable)**

SEE ALSO

configuration(7), dns-proxy(8), acl(5), auth(5), common(5), dns-proxy(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), pf-queue(5), radius(5), resolver(5), source-address(5), sysctl(5), time(5), host-matching(7)

NAME

ftp-proxy — format of ftp-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ftp-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **ftp-proxy** configuration directives:

yes-no (see **common**(5))

nls (see **common**(5))

permission (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

auth-method (see **auth**(5))

virus-status (see antivirus(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

listen-on-sock (see listen-on(5))

pass-remove (name-usage obligatory)

Passing/removing features.

remove

pass

data-type (name-usage obligatory)

Data connection method used to server.

auto

No method is preferred by this particular configuration item.

active

Use active method (PORT command).

passive

Use passive method (EPSV or PASV in case of error).

ftp-cmd (name-usage obligatory)

FTP commands

NONE

ABOR

ACCT

ADAT

ALLO

APPE

AUTH

BNB

CCC

CDUP

CLNT

CONF

CPSV

CWD

DELE

ENC

EPRT

EPSV

FEAT

HELP

LANG

LIST

LPRT

LPSV

MDTM

MIC

MKD

MLSD

MLST

MODE

MFMT

MFCT

MFF

MAIL

MLFL

MSAM

MSND

MSOM

MRCP

MRSQ

NLST

NOOP

OPEN

OPTS

PASS

PASSERVE

PASV

PBSZ

PORT

PROT

PWD

QUIT

REIN

REST

RETR

RMD

RNFR

RNTO

SITE

SIZE

SMNT

SSCN

STAT

STOR

STOU

STRU

SYST

TYPE

USER

XCWD

XCUP

XMKD

XPWD

XRMD

UNKNOWN

This "command" setting will be used for all unknown commands.

ITEMS AND SECTIONS

Configuration of **ftp-proxy** library component consists of following prototypes:
* ftp-proxy *name* { ... }

Description:

ftp-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-ctrl { ... }

server-ctrl { ... }

client-data { ... }

server-data { ... }

init-timeout ... ;

init-cmdlimit ... ;

* data-transfer ... ;

retry-data ... ;

* session-acl *name* { ... }

* command-acl *name* { ... }

* doc-acl *name* { ... }

}

This section defines FTP-proxy attributes.

The `ftp-proxy` section is derived from `proxy` section prototype. For detail description of it, see application(5).
Changes to the `ftp-proxy` section:

Section `udpserver` is not valid.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one COMMAND-ACL must be specified.

At least one DOC-ACL must be specified.

Section `monitoring` (see monitoring(5))

Item `aproxy-user` is not valid.

Item `data` used as `file`.

Item `idle-timeout` (see application(5))

Element `seconds` is optional, default: 900.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 21.

Element `proto` is optional, default: tcp.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 21.

Element `proto` is optional, default: tcp.

Item `doctype-identification.order` (see acl(5))

Only EXTENSION and MAGIC allowed for doctype identification.

Added items & subsections:

`client-ctrl` {

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Client control connection options.

The `client-ctrl` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `client-ctrl` section:

Item `conn-timeout` is not valid.

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

Item `recv-bufsize` (see netio(5))

Element `bytes` is optional, default: 1536.

`server-ctrl` {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Server control connection options.

The `server-ctrl` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `server-ctrl` section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

Item `send-bufsize` (see netio(5))

Element `bytes` is optional, default: 1536.

`client-data` {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Client data connection options.

The `client-data` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `client-data` section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`server-data` {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Server data connection options.

The `server-data` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `server-data` section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`init-timeout` [*seconds*];

Initialization timeout.

seconds **(type: `uint16`, optional, default: 120)**

`init-cmdlimit` [*number*];

Maximum of initialization commands.

number **(type: `uint16`, optional, default: 10)**

`data-transfer` *type* [*list*];

Data transfer method for particular servers.

type **(type: `data-type`)**

(AUTO means here that connection method is learned from client)

list **(type: `host-set`, optional, default: *)**

`retry-data` [*attempts*];

After succesfull write of one block of data, try several attempts to transfer other ones without checking control connection.

attempts **(type: `uint8`, optional, default: 0)**

(0 means don't try data, always check control connection

`session-acl` *name* **{**

* from ... ;
* to ... ;
* time ... ;
  time-period-set { ... }
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
  auth ... ;
  idle-timeout ... ;
  source-address ... ;
  plug-to ... ;
  language ... ;
  msgs { ... }
  hand-off ... ;
  data-port ... ;
  htftp-mode ... ;

}

The first level ACL decides how to handle incoming connections (namely communication language, authentication procedure, forwarding connection to other server etc.).

The `session-acl` section is derived from `acl-1` section prototype.
For detail description of it, see acl(5).

Changes to the `session-acl` section:

Item `user` is not valid.

Item `idle-timeout-peer` is not valid.

Authentication method must be set.

Item `doctype-ident-order` (see acl(5))

Only EXTENSION and MAGIC allowed for doctype identification.

Item `auth` (see auth(5))

OOB authentication mode cannot be ALLOWED.

Added items & subsections:

`language` *code*;

Language and charset of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

code **(type: `nls`)**

`msgs` {

welcome ... ;

hello-conn ... ;

hello-autr ... ;

hello-aunt ... ;

hello-user ... ;

}

Messages used by FTP-proxy.

Items & subsections:

`welcome` *text*;

Initial message, part one: introducing the host.

text **(type: `str`)**

`hello-conn` *text*;

Initial message, part two: remote user and host required.

text **(type: `str`)**

`hello-autr` *text*;

Initial message, part two: authentication and remote user required.

text **(type: `str`)**

`hello-aunt` *text*;

Initial message, part two: authentication user, remote user and host required.

text **(type: `str`)**

`hello-user` *text*;

Initial message, part two: remote user required.

text **(type: str)**

[End of section `ftp-proxy.session-acl.msgs` description.]

hand-off *addr cmd* [*data*];

Forwarding next-hop proxy.

addr **(type: sock)**

Proxy address:port.

cmd **(type: str)**

Proxy command name (USER or alias of SITE).

data **(type: data-type, optional, default: auto)**

Data transfer method to proxy.

(AUTO means here that no exclusive data transfer mode is required by next-hop proxy)

data-port *port*;

Port used for active data connections to clients.

If omitted, generic port is used.

port **(type: port)**

(non-generic port number/service name)

htftp-mode;

Client is served in HTTP<->FTP mode.

[End of section `ftp-proxy.session-acl` description.]

command-acl *name* **{**

* from ... ;
* server ... ;
* user ... ;
* time ... ;

time-period-set { ... }

* session-acl ... ;

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

enable-port ... ;

* command ... ;
* feature ... ;

control-client-altq ... ;

control-server-altq ... ;

data-client-altq ... ;

data-server-altq ... ;

}

The second level ACL decides how to handle particular protocol commands depending on client parameters, destination server, proxy-user etc.

The `command-acl` section is derived from `acl-2` section prototype.
For detail description of it, see acl(5).
Changes to the `command-acl` section:

Item `parent-acl` used as `session-acl`.

Command configuration must be set.

Item `doctype-ident-order` (see acl(5))

Only EXTENSION and MAGIC allowed for doctype identification.

Added items & subsections:

`enable-port`;

Allow user to specify port.

If omitted, only default port can be used.

`command` *names* `permit` [`size` *size*];

`command` *names* `deny`;

`command` *names* `abort`;

Allow/deny particular commands, set size limits.

Each command is checked against COMMAND items in order of their appearance in cfg file, and the first matching one is used. If no one matches, command is denied.

names **(type: `ftp-cmd-set`)**

(set of commands)

<**branching element**> **(type: `permission`)**

(command permission)

`size` *size* (type: `uint64`, optional, default: 0)

(command size limit, 0 = no limit)

`feature` *names* [`param` *param*] *policy*;

Allow/deny particular features offered by server as a response to the FEAT command.

Each feature found in the response is checked against FEATURE items in order of their appearance in cfg file, and the first matching one is used. If the feature has a parameter then also one is checked against PARAM elem additional to the particular FEATURE items.

If no FEATURE item matches, a default behavior hardcoded in the proxy is used. The strategy is strict: pass only features surely supported by the proxy. The current version of the proxy passes following features: LANG, MDTM, MLST, REST, SIZE, TVFS, TYPE, UTF8.

names **(type: `str-set`)**

(set of features)

`param` *param* (type: `str-set`, optional, default: *)

(feature parameter criterion)

policy (type: **pass-remove**)
(feature passing/removal)

control-client-altq *altq* [**paltq** *paltq*];
ALTQ queues for data sent to client on control connection.

altq (type: **name** of **pf-queue**, see pf-queue(5))
queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: NULL)
priority queue name (if set, used for TCP ACK without data)

control-server-altq *altq* [**paltq** *paltq*];
ALTQ queues for data sent to server on control connection.

altq (type: **name** of **pf-queue**, see pf-queue(5))
queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: NULL)
priority queue name (if set, used for TCP ACK without data)

data-client-altq *altq* [**paltq** *paltq*];
ALTQ queues for data sent to data on control connection.

altq (type: **name** of **pf-queue**, see pf-queue(5))
queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: NULL)
priority queue name (if set, used for TCP ACK without data)

data-server-altq *altq* [**paltq** *paltq*];
ALTQ queues for data sent to server on data connection.

altq (type: **name** of **pf-queue**, see pf-queue(5))
queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: NULL)
priority queue name (if set, used for TCP ACK without data)

[End of section `ftp-proxy.command-acl` description.]

doc-acl *name* {
* from ... ;
* server ... ;
* user ... ;
* time ... ;
time-period-set { ... }
* command-acl ... ;
deny ... ;
accept ... ;
rule ... ;
direction ... ;

* mime-type ... ;
  force-doctype-ident ... ;
  html-filter ... ;
* filename ... ;
  antivirus ... ;
  accept-antivirus-status ... ;
  control-client-altq ... ;
  control-server-altq ... ;
  data-client-altq ... ;
  data-server-altq ... ;
}

The third level ACL decides how to handle particular files transferred via proxy (denial, antivirus check or filtering) depending on file name, type (guessed from the file name) and transfer direction.

WARNING! Items FILENAME and MIME-TYPE are two different kinds of items. According to general Kernun ACL matching rules they are completely independent and if both present, file must match both conditions to match particular DOC-ACL.

The `doc-acl` section is derived from `acl-3` section prototype. For detail description of it, see acl(5).

Changes to the `doc-acl` section:

Item `parent-acl` used as `command-acl`.

Item `size` is not valid.

Item `content-type` is not valid.

Item `virus-status` is not valid.

Item `modify-header` is not valid.

Item `replace` is not valid.

Item ANTIVIRUS not allowed if DENY is on.

Item ACCEPT-ANTIVIRUS-STATUS not allowed if DENY is on.

Added items & subsections:

`filename` *names*;

Entry condition - name of transferred file.

names **(type: `str-set`)**

Only last part of file name (without path) is used for matching

`antivirus` *channel* [`interval` *interval*] [`chunk` *chunk*] [`limit` *limit*];

Antivirus usage mode.

Check document by antivirus, with settings for passing initial part of unchecked data through the antivirus module during antivirus checking.

channel **(type: `name-list` of `antivirus`, see antivirus(5))**

Name of ANTIVIRUS global section used.

`interval` *interval* (type: `uint16`, optional, default: 0)

Seconds between passing blocks of unchecked data (0 = do not send unchecked data).

chunk *chunk* **(type: uint32, optional, default: 0)**
Size of each block of unchecked data.

limit *limit* **(type: uint32, optional, default: 0)**
Maximum size of unchecked data passed before antivirus check is completed. Remaining data will be passed only after successful checking.

accept-antivirus-status *status*;
Defines set of antivirus status codes (in addition to FREE) that allow further passing of data. Other status codes cause termination of data transfer. If not set, data are passed only if the antivirus returns status FREE.

status **(type: virus-status-set)**

control-client-altq *altq* [**paltq** *paltq*];
ALTQ queues for data sent to client on control connection.

altq **(type: name of pf-queue, see** pf-queue**(5))**
queue name

paltq *paltq* **(type: name of pf-queue, see** pf-queue**(5), optional, default: NULL)**
priority queue name (if set, used for TCP ACK without data)

control-server-altq *altq* [**paltq** *paltq*];
ALTQ queues for data sent to server on control connection.

altq **(type: name of pf-queue, see** pf-queue**(5))**
queue name

paltq *paltq* **(type: name of pf-queue, see** pf-queue**(5), optional, default: NULL)**
priority queue name (if set, used for TCP ACK without data)

data-client-altq *altq* [**paltq** *paltq*];
ALTQ queues for data sent to data on control connection.

altq **(type: name of pf-queue, see** pf-queue**(5))**
queue name

paltq *paltq* **(type: name of pf-queue, see** pf-queue**(5), optional, default: NULL)**
priority queue name (if set, used for TCP ACK without data)

data-server-altq *altq* [**paltq** *paltq*];
ALTQ queues for data sent to server on data connection.

altq **(type: name of pf-queue, see** pf-queue**(5))**
queue name

paltq *paltq* **(type: name of pf-queue, see** pf-queue**(5), optional, default: NULL)**
priority queue name (if set, used for TCP ACK without data)

[End of section `ftp-proxy.doc-acl` description.]

[End of section `ftp-proxy` description.]

SEE ALSO

configuration(7), acl(5), antivirus(5), application(5), auth(5), common(5), ipc(5), listen-on(5), log(5), monitoring(5), netio(5), pf-queue(5), source-address(5), time(5)

NAME

ftp-proxy.cfg — format of ftp-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ftp-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **ftp-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

nls (see **common**(5))

permission (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see ipc(5))

radius-attr (see radius(5))

ldap-tls-reqcert-mode (see ldap(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

antivirus-protocol (see antivirus(5))

virus-status (see antivirus(5))

database-source (see antivirus(5))

source-address-mode (see source-address(5))

accept-deny (see mod-html-filter(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

pass-remove (see ftp-proxy(5))

data-type (see ftp-proxy(5))

ftp-cmd (see ftp-proxy(5))

ITEMS AND SECTIONS

Program **ftp-proxy** recognizes following items and sections:

admin ... ;
* antivirus *name* { ... }
* html-filter *name* { ... }
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }

sysctl { ... }
use-resolver ... ;
* ftp-proxy *name* { ... }

ipv6-mode ... ;

Description:

admin *system* **[***contact***];**

Firewall administrator and contact e-mail addresses.

system **(type: str)**

The technical administrator(s) of the system; an address or set of comma separated adresses of persons responsible for system maintenance.

contact **(type: str, optional, default: <NULL>)**

The policy administator; an address of person responsible for system configuration. If not defined, the technical administration is used instead.

Constraints:

Administrator contact must comply with RFC.

antivirus *name* **{**

connection ... ;

sock-opt { ... }

timeout ... ;

comm-dir ... ;

altq ... ;

max-checked-size ... ;

icap-pass-200-with-pure-body ... ;

persistent-stream ... ;

clamav-agent { ... }

}

The `antivirus` section is derived from `antivirus` section prototype. For detail description of it, see antivirus(5).

`html-filter` *name* {

* script-tag-language ... ;

replace-head-script-tags ... ;

replace-body-script-tags ... ;

* style-tag-type ... ;

replace-style-tags ... ;

* iframe-tag-src ... ;

replace-iframe-tags ... ;

* intrinsic-language ... ;

* intrinsic-hack ... ;

replace-intrinsic ... ;

* macro-language ... ;

* macro-hack ... ;

replace-macros ... ;

* uri ... ;

replace-uri ... ;

* embed-tag-type ... ;

* embed-src-hack ... ;

* embed-plugin-hack ... ;

replace-head-embed-tags ... ;

replace-body-embed-tags ... ;

* applet ... ;

replace-applets ... ;

* object ... ;

* object-classid-hack ... ;

* object-data-hack ... ;

replace-head-object-tags ... ;

replace-body-object-tags ... ;

* param-tags ... ;

replace-param ... ;

script-end-hack ... ;

}

The `html-filter` section is derived from `html-filter` section prototype. For detail description of it, see mod-html-filter(5).

`interface` *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`ldap-client-auth` *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

oob-auth *name* **{**

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The oob-auth section is derived from oob-auth section prototype. For detail description of it, see auth(5).

pf-queue *name* **{**

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The pf-queue section is derived from pf-queue section prototype. For detail description of it, see pf-queue(5).

radius-client *name* **{**

nas ... ;

groups ... ;

* server ... ;

}

The radius-client section is derived from radius-client section prototype. For detail description of it, see radius(5).

resolver *name* **{**

* server ... ;

search ... ;

```
    preference ... ;
    edns ... ;
    conf-timeout ... ;
    initial-timeout ... ;
    final-timeout ... ;
    conn-timeout ... ;
    disable-deresolution ... ;
}
```

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`shared-dir` *name* {

```
    path ... ;
}
```

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

`shared-file` *name* {

```
    path ... ;
    format ... ;
}
```

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`sysctl` {

```
* variable ... ;
    portrange-default ... ;
    portrange-high ... ;
    portrange-low ... ;
    portrange-reserved ... ;
    somaxconn ... ;
    log-in-vain ... ;
    blackhole ... ;
}
```

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`ftp-proxy` *name* {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-ctrl { ... }

server-ctrl { ... }

client-data { ... }

server-data { ... }

init-timeout ... ;

init-cmdlimit ... ;

* data-transfer ... ;

  retry-data ... ;

* session-acl *name* { ... }

* command-acl *name* { ... }

* doc-acl *name* { ... }

}

The `ftp-proxy` section is derived from `ftp-proxy` section prototype.
For detail description of it, see ftp-proxy(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), ftp-proxy(8), acl(5), antivirus(5), auth(5), common(5), ftp-proxy(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), mod-html-filter(5), pf-queue(5), radius(5), resolver(5), source-address(5), sysctl(5), time(5), host-matching(7)

NAME

gk-proxy — format of gk-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **gk-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **gk-proxy** configuration directives:

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

week-day (see **time**(5))

month (see **time**(5))

auth-method (see **auth**(5))

source-address-mode (see **source-address**(5))

source-port-mode (see **source-address**(5))

transparency (see **acl**(5))

user-auth-spec (see **acl**(5))

doctype-ident-method (see **acl**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

listen-on-sock (see **listen-on**(5))

ITEMS AND SECTIONS

Configuration of **gk-proxy** library component consists of following prototypes:
* gk-proxy *name* { ... }

Description:

gk-proxy *name* {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

udpserver { ... }

source-address ... ;

doctype-identification { ... }

map-file ... ;

* session-acl *name* { ... }

}

H.323 GateKeeper Proxy configuration.

The gk-proxy section is derived from proxy section prototype. For detail description of it, see application(5).
Changes to the gk-proxy section:

Section `tcpserver` is not valid.

Section UDPSERVER required.

RAS Yellow Pages File name must be specified.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

Section `monitoring` (see monitoring(5))

Monitoring is not functional in H.323 proxies in this version.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 1719.

Element `proto` is optional, default: udp.

GK proxy cannot bind address [0.0.0.0].

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 1719.

Element `proto` is optional, default: udp.

Added items & subsections:

`map-file` *name*;

RAS Yellow Pages File.

This file name must be identical with one defined in H.323 Proxy.

***name* (type: `str`)**

`session-acl` *name* {

* from ... ;
* to ... ;
* time ... ;
  time-period-set { ... }
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
  idle-timeout ... ;
  idle-timeout-peer ... ;
  source-address ... ;
  plug-to ... ;
  session-timeout ... ;
  register ... ;
  h323-address ... ;
  client-altq ... ;
  server-altq ... ;

}

The `session-acl` section is derived from `acl-1` section prototype. For detail description of it, see acl(5).

Changes to the `session-acl` section:

Item `user` is not valid.

Item `auth` is not valid.

Item H323-ADDRESS required.

Item REGISTER required.

Item `idle-timeout` (see acl(5))

Element `seconds` is optional, default: 120.

Added items & subsections:

`session-timeout` [*seconds*];

Maximum duration of session.

seconds **(type: `uint31`, optional, default: 0)**

Duration in seconds (0 = unlimited).

`register client`;

`register [force]` *addr*;

Address to be registered on gatekeeper.

<branching element> (type: `source-port-mode`, optional, default: force)

addr **(type: `sock`)**

Use specified address.

`h323-address` *addr*;

Listening Address of H.323 Proxy

addr **(type: `sock`)**

`client-altq` *altq*;

ALTQ queue for data sent to client.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

`server-altq` *altq*;

ALTQ queue for data sent to server.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

[End of section `gk-proxy.session-acl` description.]

[End of section `gk-proxy` description.]

SEE ALSO

configuration(7), acl(5), application(5), auth(5), common(5), listen-on(5), log(5), monitoring(5), pf-queue(5), source-address(5), time(5), h323-proxy(8)

NAME

gk-proxy.cfg — format of gk-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **gk-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **gk-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

inline-file-format (see **common**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see **radius**(5))

ldap-tls-reqcert-mode (see **ldap**(5))

ldap-search-scope (see **ldap**(5))

ldap-group-match (see **ldap**(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

source-address-mode (see source-address(5))

source-port-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

dbglev (see log(5))

logfail-mode (see log(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

ITEMS AND SECTIONS

Program **gk-proxy** recognizes following items and sections:
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* gk-proxy *name* { ... }
  ipv6-mode ... ;

Description:

interface *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

ldap-client-auth *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

oob-auth *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

```
  file ... ;
  lock ... ;
}
```

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

`pf-queue` *name* {

```
  parent ... ;
  bandwidth ... ;
  priority ... ;
  qlimit ... ;
  cbq { ... }
  priq { ... }
  hfsc { ... }
}
```

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

`radius-client` *name* {

```
  nas ... ;
  groups ... ;
* server ... ;
}
```

The `radius-client` section is derived from `radius-client` section prototype. For detail description of it, see radius(5).

`resolver` *name* {

```
* server ... ;
  search ... ;
  preference ... ;
  edns ... ;
  conf-timeout ... ;
  initial-timeout ... ;
  final-timeout ... ;
  conn-timeout ... ;
```

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`shared-dir` *name* {

path ... ;

}

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

`shared-file` *name* {

path ... ;

format ... ;

}

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`gk-proxy` *name* {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

udpserver { ... }

source-address ... ;

doctype-identification { ... }

map-file ... ;

* session-acl *name* { ... }

}

The `gk-proxy` section is derived from `gk-proxy` section prototype. For detail description of it, see gk-proxy(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), gk-proxy(8), acl(5), auth(5), common(5), gk-proxy(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), pf-queue(5), radius(5), resolver(5), source-address(5), sysctl(5), time(5), host-matching(7), h323-proxy(8)

NAME

h323-proxy — format of h323-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **h323-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **h323-proxy** configuration directives:

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

auth-method (see **auth**(5))

source-address-mode (see **source-address**(5))

transparency (see **acl**(5))

user-auth-spec (see **acl**(5))

doctype-ident-method (see **acl**(5))

listen-on-sock (see **listen-on**(5))

ITEMS AND SECTIONS

Configuration of **h323-proxy** library component consists of following prototypes:
* h323-proxy *name* { ... }

Description:

h323-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

doctype-identification { ... }

client-ctrl { ... }

server-ctrl { ... }

data-channel { ... }

map-file ... ;

* session-acl *name* { ... }

max-channel-ports ... ;

}

This section defines H.323-proxy attributes.

The `h323-proxy` section is derived from `proxy` section prototype. For detail description of it, see application(5).

Changes to the `h323-proxy` section:

Section `udpserver` is not valid.

Item `source-address` is not valid.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

Section `monitoring` (see monitoring(5))

Monitoring is not functional in H.323 proxies in this version.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 1720.

Element `proto` is optional, default: tcp.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 1720.

Element `proto` is optional, default: tcp.

Added items & subsections:

`client-ctrl` {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Client H.225/H.245 connection options.

The `client-ctrl` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

`server-ctrl` {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Server H.225/H.245 connection options.

The `server-ctrl` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

`data-channel` {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Multimedia data channel options.

The `data-channel` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

`map-file` *name*;

RAS Yellow Pages File.

This file name must be identical with one defined in Gatekeeper Proxy.

name **(type: `str`)**

`session-acl` *name* {

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

idle-timeout ... ;

source-address ... ;

plug-to ... ;

client-altq ... ;

server-altq ... ;

ras ... ;

allow-peer ... ;

}

The `session-acl` section is derived from `acl-1` section prototype. For detail description of it, see acl(5).

Changes to the `session-acl` section:

Item `user` is not valid.

Item `auth` is not valid.

Item `idle-timeout-peer` is not valid.

Added items & subsections:

`client-altq` *altq* **[`paltq`** *paltq*`]`**;**

ALTQ queues for data sent to client.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

`paltq` *paltq* **(type: `name` of `pf-queue`, see pf-queue(5), optional, default: NULL)**

priority queue name (if set, used for TCP ACK without data)

`server-altq` *altq* **[`paltq`** *paltq*`]`**;**

ALTQ queues for data sent to server.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

`paltq` *paltq* **(type: `name` of `pf-queue`, see pf-queue(5), optional, default: NULL)**

priority queue name (if set, used for TCP ACK without data)

`ras`;

Use this ACL for RAS-driven connections.

`allow-peer` *peers*;

Allow additional peers for data channels.

Without this item, just client/server addresses can be used as data channel targets. Any other address offered by peers will be refused.

peers **(type: `host-set`)**

[End of section `h323-proxy.session-acl` description.]

`max-channel-ports` **[***limit***]**;

Maximum of per-session logical channel ports.

limit **(type: `uint16`, optional, default: 16)**

[End of section `h323-proxy` description.]

SEE ALSO

configuration(7), acl(5), application(5), auth(5), common(5), ipc(5), listen-on(5), log(5), monitoring(5), netio(5), pf-queue(5), source-address(5), time(5), gk-proxy(8)

NAME

h323-proxy.cfg — format of h323-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **h323-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **h323-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see **radius**(5))

ldap-tls-reqcert-mode (see **ldap**(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

ITEMS AND SECTIONS

Program **h323-proxy** recognizes following items and sections:
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* h323-proxy *name* { ... }
  ipv6-mode ... ;

Description:

interface *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

ldap-client-auth *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

oob-auth *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

`pf-queue` *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

`radius-client` *name* {

nas ... ;

groups ... ;

* server ... ;

}

The `radius-client` section is derived from `radius-client` section prototype. For detail description of it, see radius(5).

`resolver` *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`shared-dir` *name* {

path ... ;

}

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

`shared-file` *name* {

path ... ;

format ... ;

}

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name (type: **name** of **resolver**, see resolver(5))

h323-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

doctype-identification { ... }

client-ctrl { ... }

server-ctrl { ... }

data-channel { ... }

map-file ... ;

* session-acl *name* { ... }

max-channel-ports ... ;

}

The h323-proxy section is derived from h323-proxy section prototype. For detail description of it, see h323-proxy(5).

ipv6-mode [*status*];

Enabling/Disabling IPv6 Mode.

status (type: **enabling**, optional, default: enable)

SEE ALSO

configuration(7), h323-proxy(8), acl(5), auth(5), common(5), h323-proxy(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), pf-queue(5), radius(5), resolver(5), source-address(5), sysctl(5), time(5), host-matching(7), gk-proxy(8)

NAME

http-cache — format of http-cache component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **http-cache** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **http-cache** configuration directives:

ip-version (see **common**(5))

osi4-proto (see **common**(5))

listen-on-sock (see **listen-on**(5))

ITEMS AND SECTIONS

Configuration of **http-cache** library component consists of following prototypes:

http-cache { ... }

Description:

http-cache {

phase ... ;

* tag ... ;

listen-on { ... }

hand-off ... ;

cache-size ... ;

max-object-size ... ;

* raw ... ;

}

HTTP Cache Daemon configuration.

Currently, the Squid daemon is used for Kernun caching.

Constraints:

Addresses to listen on must be specified.

HTTP cache size must be specified.

Items & subsections:

phase [*number*];

Application Startup Phase.

number **(type: uint8, optional, default: 40)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: str)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

listen-on {

* socket ... ;

}

The listen-on section is derived from listen-on section prototype.
For detail description of it, see listen-on(5).
Changes to the listen-on section:

Item `non-transparent` used as `socket`.

Item `transparent` is not valid.

At least one address to listen on must be specified.

Item socket (see listen-on(5))

Element `proto` is optional, default: tcp.

hand-off *addr*;

Next-hop proxy.

See CACHE_PEER configuration item of squid.conf.

addr **(type: sock)**

cache-size *size*;

Disc cache size.

See CACHE_DIR configuration item of squid.conf.

size **(type: uint64)**

max-object-size *size*;

Maximum object size to be kept in cache.

See MAX OBJECT SIZE configuration item of squid.conf.

size (type: **uint64**)

raw *line*;

Raw line to be written to squid.conf configuration file.

line (type: **str**)

[End of section `http-cache` description.]

SEE ALSO

configuration(7), common(5), listen-on(5), squid(8)

NAME

http-control — format of http-control component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **http-control** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

ITEMS AND SECTIONS

Configuration of **http-control** library component consists of following prototypes:

deny-msg ... ;

Description:

deny-msg [template *template*] [*msg*];

The error page returned when a request is denied by ACL. If not set, a generic "denied by security policy" page is returned, except when the denying ACL contains item CLEAR-WEB-DB-MATCH. In this case, the returned page will display the categories of the request URI that match the CLEAR-WEB-DB-MATCH item. See also http-proxy(8) for the instructions how to create custom error pages.

template *template* **(type: str, optional, default: "")**

error page template file name, without the .html.LANGUAGE-CHARSET suffix, relative to the DOCUMENT-ROOT directory

msg **(type: str, optional, default: "")**

a message inserted into the error page template

SEE ALSO

configuration(7), http-proxy(8)

NAME

http-proxy — format of http-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **http-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **http-proxy** configuration directives:

yes-no (see **common**(5))

language (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

auth-method (see **auth**(5))

virus-status (see **antivirus**(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

listen-on-sock (see listen-on(5))

clear-web-db-category (see clear-web-db(5))

clear-web-db-match-mode (see clear-web-db(5))

replace-authorization-mode (name-usage obligatory)

Variants of authorization replace mode

simple

Replace the authorization data using a lookup table

radius

Replace the authorization data using a lookup table. Moreover, use a radius for authentication.

proxy-via (name-usage obligatory)

Values for controlling Via header.

pass

If a request or reply contains a Via header, it is passed through unchanged.

add

Each request or reply will get a Via header line added for the firewall host.

full

Each generated Via: header line will additionally have the http-proxy identification version shown as a Via comment field.

block

Every proxy request or reply will have all its Via header lines removed. No new Via header will be generated.

fake

All Via headers will be removed and then a fake header will be added.

http-protocol (name-usage obligatory)

Version of HTTP

HTTP-OTHER-VER

HTTP v. other than 0.9, 1.0, or 1.1

HTTP-0-9

HTTP v. 0.9

HTTP-1-0

HTTP v. 1.0

HTTP-1-1

HTTP v. 1.1

http-scheme (name-usage obligatory)

Scheme in HTTP URI

HTTP

http://...

FTP

ftp://...

cookie-table-clean (name-usage obligatory)

When to clean cookie table.

never

Table is persistent, must be deleted manually if corrupted.

on-restart

Table is deleted and recreated when the proxy is restarted, but not when reloaded.

always

Table is deleted when the proxy is restarted, reloaded, or stopped.

accept-gzip (name-usage obligatory)

How to handle the Accept-Encoding header.

no

Do not modify the Accept-Encoding header.

yes

Always add "Accept-Encoding: gzip" to the request.

client

Keep the Accept-Encoding header, but remove all encodings except gzip/x-gzip and identity. Do not add if missing.

client-add

Keep the Accept-Encoding header, but remove all encodings except gzip/x-gzip and identity. Add empty if missing.

content-gzip (name-usage obligatory)

How to handle the Content-Encoding header. Controls conversion of gzip and x-gzip values in the header. The same operation is applied also to compress and x-compress values.

keep

Do not modify Content-Encoding.

gzip

Convert x-gzip to gzip.

x-gzip

Convert gzip to x-gzip.

http

As X-GZIP for HTTP/1.0 clients and GZIP for HTTP/1.1 clients.

accept

Use the same form (gzip or x-gzip) as used by the client in the Accept-Encoding request header. Works as HTTP if Accept-Encoding is not present or does not contain gzip nor x-gzip.

http-redirect (name-usage optional)

Status code for HTTP redirection.

permanent (301)

HTTP status 301 Moved Permanently

temporary (302)

HTTP status 302 Found

temporary-same-method (307)

HTTP status 307 Temporary Redirect

kerberos-user-match (name-usage obligatory)

How a Kerberos user name is matched in ACL and logged.

short

Match and log only the user name without the @REALM part.

full

Match and log the whole user@REALM name.

ldap-select (name-usage obligatory)

How a LDAP-CLIENT-AUTH section is selected.

name

Select by section name.

domain

Select by domain name.

auth-headers (name-usage obligatory)

Which authentication-related headers and responses are used.

proxy

Uses proxy authentication headers (Proxy-Authenticate, Proxy-Authorization) and responses (407 Proxy Authentication Required).

server

Uses origin server authentication headers (WWW-Authenticate, Authorization) and responses (401 Unauthorized).

sni-result (name-usage obligatory)

SNI inspection result.

specified

TLS 1.0 or higher with specified SNI so host and URI were changed to value from TLS ClientHello.

unspecified

TLS 1.0 or higher without specified SNI so host and URI remained unchanged.

ssl3

SSLv3 detected, therefore SNI inspection was skipped so host and URI remained unchanged.

skype

Skype protocol detected, therefore SNI inspection skipped so host and URI remained unchanged. Note that Skype uses HTTPS as well so only part of Skype communication will have this result.

unknown-protocol

Unknown protocol detected, therefore SNI inspection was skipped so host and URI remained unchanged.

ITEMS AND SECTIONS

Configuration of **http-proxy** library component consists of following prototypes:

via-mode ... ;
* start-line-check ... ;
* header-check ... ;
* aproxy *name* { ... }
* web-filter *name* { ... }
* ntlm-auth *name* { ... }
* kerberos-auth *name* { ... }
* http-proxy *name* { ... }

Description:

via-mode pass;

via-mode add [*faked*];

via-mode full [*faked*];

via-mode [**block**];

via-mode fake [*faked*];

Processing Via headers.

<**branching element**> (**type: proxy-via, optional, default: block**)

faked (**type: str, optional, default: ""**)

full Via header value if FAKE, replacement for firewall hostname for ADD, FULL

start-line-check *text-match max-len*;

Check of the start-line of a HTTP message.

text-match (**type: str-set**)

The line must match this.

max-len (**type: uint32**)

Maximum number of characters in the line

header-check *name-match val-match max-val-len*;

Check of a HTTP message header.

name-match (**type: str-set**)

The header name must match this.

val-match (**type: str-set**)

The header value must match this.

max-val-len (**type: uint32**)

Maximum number of characters of the header value

aproxy *name* {

auth ... ;

insecure-cookies ... ;

oob-auth ... ;

cookie-name ... ;

logout ... ;

timeout-idle ... ;

timeout-unauth ... ;

bufsz ... ;

}

Settings of authentication proxy for HTTP servers and of authentication server for out-of-band authentication.

Constraints:

Authentication method must be set.

Items & subsections:

auth none;

auth passwd *file*;

auth radius *client*;

auth ldap *ldap*;

auth ext *file*;

auth oob *oob* [*mode* [**loose**]];

Authentication method and attributes specification.

For more details, see auth(7).

<**branching element**> (**type: auth-method**)

file (**type: str**)

Password/utility file name.

client (**type: name of radius-client**, see radius(5))

RADIUS client configuration name.

ldap (**type: name of ldap-client-auth**, see ldap(5))

LDAP client configuration parameters.

oob (**type: name of oob-auth**, see auth(5))

OOB authentication parameters.

mode (**type: obligation, optional, default: required**)

loose (**type: key, optional**)

Constraints:

Out-of-band authentication cannot be used here.

insecure-cookies;

Allow client to send session cookie using insecure (non-SSL/TLS) connections.

oob-auth;

This is out-of-band authentication server, not an HTTP authentication proxy.

cookie-name [*val*];

Name of session cookie.

val (**type: str, optional, default: "KernunAProxySession"**)

logout [*path*];

If request URI contains this path, user is logged out.

path (**type: str, optional, default: "/logout"**)

timeout-idle [*sec*];

Number of seconds until authentication expires when the client is idle (sends no requests).

sec **(type: uint32, optional, default: 360)**

timeout-unauth [*sec*];

Maximum time after which authentication expires.

sec **(type: uint32, optional, default: 3600)**

bufsz [*bytes*];

Input buffer size for mod-aproxy.

bytes **(type: uint32, optional, default: 16384)**

[End of section aproxy description.]

web-filter *name* {

connection ... ;

fail-ok ... ;

sock-opt { ... }

}

External WebFilter configuration.

Items & subsections:

connection *socket*;

Network address of the web filter

socket **(type: sock)**

fail-ok;

Allow request if communication with the web filter fails

sock-opt {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection to web filter options.

The sock-opt section is derived from sock-opt section prototype.
For detail description of it, see netio(5).
Changes to the sock-opt section:

Item recv-timeout is not valid.

Item send-timeout is not valid.

[End of section web-filter description.]

ntlm-auth *name* **{**

domain ... ;

workgroup ... ;

* ad-controller ... ;

interfaces { ... }

ldap ... ;

timeout ... ;

timeout-idle ... ;

timeout-unauth ... ;

}

Constraints:

Active Directory domain name must be specified.

Active Directory controller address must be specified.

INTERFACES must be specified.

Items & subsections:

domain *name*;

Active Directory domain.

name **(type: str)**

workgroup *name*;

Workgroup name. If not set, the first component of the DOMAIN name (before the first period) is used.

name **(type: str)**

ad-controller *addr*;

Active Directory Domain Controller address.

addr **(type: host)**

interfaces {

* interface ... ;

* network ... ;

}

Selects the interfaces for communication with the Active Directory Domain Controller. If not set, all interfaces can be used. It is possible to specify either an INTERFACE section names, or network IP addresses with masks.

Constraints:

NTLM-related communication must be limited by INTERFACE or NETWORK.

Items & subsections:

interface *name*;

Communicate on this interface.

name **(type: name of interface, see interface(5))**

network *addr*;

Communicate on this network.

addr **(type: net)**

[End of section `ntlm-auth.interfaces` description.]

ldap *name*;

Ask an LDAP server for a list of groups each NTLM-authenticated user belongs to.

name **(type: name of ldap-client-auth, see ldap(5))**

timeout [*sec*];

Timeout for communicating with the NTLM helper program (0 = unlimited).

sec **(type: uint16, optional, default: 5)**

timeout-idle [*sec*];

Number of seconds until cached OOB authentication expires when the client is idle (sends no requests).

sec **(type: uint32, optional, default: 360)**

timeout-unauth [*sec*];

Maximum time after which cached OOB authentication expires.

sec **(type: uint32, optional, default: 3600)**

[End of section `ntlm-auth` description.]

kerberos-auth *name* **{**

domain ... ;

user-match ... ;

kinit ... ;

keytab ... ;

proxy-host ... ;

* ad-controller ... ;

ldap ... ;

timeout-idle ... ;

timeout-unauth ... ;

lock ... ;

lock-ldap ... ;

one-per-session ... ;

}

Constraints:

Active Directory domain name must be specified.

Active Directory controller address must be specified.

Items & subsections:

domain *name*;

Active Directory domain.

name (type: **str**)

user-match [*match*];

How a Kerberos user name is matched in ACL and logged.

match (type: **kerberos-user-match**, optional, default: **short**)

kinit *principal*;

The Kerberos principal used to get a TGT for access to the LDAP. If not set, the Kernun system account in the Active Directory will be used, that is, the host name without domain followed by the dollar sign (host$). Empty principal means that the proxy should not try to obtain a ticket.

principal (type: **str**)

keytab *path*;

The keytab file used for Kerberos authentication. If not specified, the default keytab file /etc/krb5.keytab will be used.

path (type: **name** of **shared-file**, see common(5))

proxy-host *name*;

The proxy host name expected in authentication data from clients. This is also the proxy host name that users should set in their browser configuration. The specified host name will be used in the proxy Kerberos principal (HTTP/proxy-host). If not set, the host name of the Kernun system will be used.

name (type: **str**)

ad-controller *addr*;

Active Directory Domain Controller address.

addr (type: **host**)

ldap [**name**] *name*;

ldap domain;

Ask an LDAP server for a list of groups each Kerberos-authenticated user belongs to.

<branching element> (type: **ldap-select**, optional, default: **name**)

name (type: **name** of **ldap-client-auth**, see ldap(5))

LDAP-CLIENT-AUTH section name

timeout-idle [*sec*];

Number of seconds until cached group membership expires when the client is idle (sends no requests).

sec **(type: uint32, optional, default: 7200)**

timeout-unauth [*sec*];

Maximum time after which cached group membership expires.

sec **(type: uint32, optional, default: 86400)**

lock none;

lock semaphore;

lock lock2 [*path*];

lock [**multilock2**] [*path*];

An alternative implemetation of locks.

<**branching element**> **(type: lock-type, optional, default: multilock2)**

path **(type: str, optional, default:** <**NULL**>**)**

If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

lock-ldap [*val*];

Whether getting groups from LDAP should be protected by LOCK. Locking may be used if the proxy is slowed down by many child processes performing Kerberos authentication to a LDAP server at the same time.

val **(type: yes-no, optional, default: no)**

one-per-session [*val*];

Whether Kerberos authentication should be performed only once per session. Authentication headers in requests following successful authentication are ignored.

val **(type: yes-no, optional, default: yes)**

[End of section `kerberos-auth` description.]

http-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

document-root ... ;

hdr-line-len ... ;

blacklist-db ... ;

connect-data-mime-db ... ;

ftp-proxy ... ;

max-aproxy-sessions ... ;

max-bypass-sessions ... ;

oob-auth-srv ... ;

ssl-session-cache { ... }

aproxy-lock ... ;

cookie-table { ... }

extended-status ... ;

* session-acl *name* { ... }

* request-acl *name* { ... }

* doc-acl *name* { ... }

}

HTTP proxy configuration.

The `http-proxy` section is derived from `proxy` section prototype. For detail description of it, see application(5).

Changes to the `http-proxy` section:

Section `udpserver` is not valid.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one REQUEST-ACL must be specified.

At least one DOC-ACL must be specified.

Document root path required.

BLACKLIST-DB required by REQUEST-ACL.BLACKLIST and DOC-ACL.BLACKLIST.

Section `monitoring` (see monitoring(5))

Item `data` used as `uri`.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 80.

Element `proto` is optional, default: tcp.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 80.

Element `proto` is optional, default: tcp.

Item `tcpserver.init-children` (see tcpserver(5))

Element `value` is optional, default: 50.

Item `tcpserver.max-children` (see tcpserver(5))

Element `value` is optional, default: 1500.

Item `tcpserver.min-idle` (see tcpserver(5))

Element `value` is optional, default: 50.

Item `tcpserver.max-idle` (see tcpserver(5))

Element `value` is optional, default: 70.

Added items & subsections:

`client-conn` {

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection from client options.

The `client-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

Changes to the `client-conn` section:

Item `conn-timeout` is not valid.

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

server-conn {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection to server options.

The server-conn section is derived from sock-opt section prototype. For detail description of it, see netio(5).

Changes to the server-conn section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

document-root *path*;

Directory containing error documents and forced responses.

path (type: **name** of **shared-dir**, see common(5))

hdr-line-len [*bytes*];

Maximum (multi)line length in HTTP message headers (sets buffer size in header processing modules)

bytes (type: **uint32**, optional, default: **12Ki**)

blacklist-db *fname*;

Name of DB file with a blacklist

fname (type: **str**)

connect-data-mime-db *filename*;

CONNECT data MIME type mapping file.

filename (type: **name** of **shared-file**, see common(5))

ftp-proxy *addr anon-usr anon-pass*;

Parameters for ftp-proxy used for ftp scheme requests

addr (type: **sock**)

address of ftp-proxy

anon-usr (type: **str**)

username for anonymous FTP

anon-pass (type: **str**)

password for anonymous FTP

max-aproxy-sessions [*val*];

Maximum number of simultaneously active sessions.

val (type: **uint16**, optional, default: **100**)

max-bypass-sessions [*num*];

Maximum number of simultaneous Clear Web DataBase bypass sessions. If set to 0, then bypass session management will use cookies instead, which limits a bypass to a single domain only.

num (type: **uint32**, optional, default: **1000**)

oob-auth-srv *name*;

Parameters of OOB authentication server.

name (type: **name of oob-auth**, see auth(5))

ssl-session-cache {

capacity ... ;

dir ... ;

lock ... ;

}

The ssl-session-cache section is derived from ssl-session-cache section prototype. For detail description of it, see ssl(5).

aproxy-lock [*path*];

Lock for exclusive access to the authentication proxy and OOB authentication session tables.

path (type: **str**, optional, default: <**NULL**>)

If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

cookie-table {

file ... ;

clean ... ;

size ... ;

max-age ... ;

}

Table of modified cookies.

Items & subsections:

file [*path*];

Database file for storing the table.

path (type: **str**, optional, default: "/tmp/http-proxy.cookie-table")

clean [*when*];

When to clean then table.

when (type: **cookie-table-clean**, optional, default: on-restart)

size [*sz*];

Maximum number of stored cookies.

sz (type: **uint32**, optional, default: 1024)

Constraints:

Cookie table size must be at least 1.

max-age [*sec*];

Cookie lifetime (seconds). If a cookie itself specifies shorter lifetime, the shorter value is used instead.

sec (type: **uint32**, optional, default: 36000)

Constraints:

Cookie lifetime must not be zero.

[End of section `http-proxy.cookie-table` description.]

extended-status [*val*];

Permit status codes in statistical messages other than ACCEPTED and REJECTED

val (type: **yes-no**, optional, default: yes)

session-acl *name* **{**

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

auth ... ;

idle-timeout ... ;

source-address ... ;

plug-to ... ;

captured-connect ... ;

* connect-session-acl ... ;

* connect-request-acl ... ;

* ip-tos-from-client ... ;

linger-time ... ;

client-keepalive ... ;

server-keepalive ... ;

language ... ;

hand-off ... ;

```
  client-ssl ... ;
  server-ssl ... ;
* client-cert-match ... ;
* server-cert-match ... ;
  simulate-connect ... ;
  sni-insp ... ;
  aproxy ... ;
  ntlm-auth ... ;
  kerberos-auth ... ;
  authenticate-at ... ;
  acl-error-status ... ;
  server-from-tcp ... ;
}
```

The `session-acl` section is derived from `acl-1` section prototype.
For detail description of it, see acl(5).
Changes to the `session-acl` section:

Item `user` is not valid.

Item `idle-timeout-peer` is not valid.

SSL/TLS required on connection in order to match client certificates.

Authentication method must be set.

CONNECT-SESSION-ACL requires CAPTURED-CONNECT.

CONNECT-REQUEST-ACL requires CAPTURED-CONNECT.

Only one of NTLM-AUTH and KERBEROS-AUTH may be used.

Only one of SIMULATE-CONNECT and SNI-INSP may be used.

CLIENT-SSL and SNI-INSP are mutually exclusive.

SERVER-SSL and SNI-INSP are mutually exclusive.

Item `auth` (see auth(5))

Element `mode` is optional, default: allowed.

Added items & subsections:

`captured-connect` [*test*];

A test if this session handles a captured CONNECT request (instead of creating a tunnel to the destination server).

test **(type: `yes-no`, optional, default: no)**

`connect-session-acl` *name*;

SESSION-ACL used to handle the captured CONNECT request.

name **(type: `str-set`)**

`connect-request-acl` *name*;

REQUEST-ACL used to handle the captured CONNECT request.

name **(type: `str-set`)**

ip-tos-from-client *val*;

Testing an IP TOS value of received packets.

val (type: **uint8-set**)

linger-time [*seconds*];

Read end of client connection will linger for EOF after closing the write end.

seconds (type: **uint32**, optional, default: 1)

client-keepalive [**req** *req*] [**idle** *idle*];

Parameters of client connection keep-alive.

req *req* (type: **uint16**, optional, default: 100)

max. number of requests on single connection (0 = unlimited, 1 = persistent client connections not used)

idle *idle* (type: **uint16**, optional, default: 15)

max. idle time before the first request or between requests on single connection (0 = unlimited)

server-keepalive [**req** *req*] [**idle** *idle*] [**conn** *conn*];

Parameters of server connection keep-alive.

req *req* (type: **uint16**, optional, default: 100)

max. number of requests on single connection (0 = unlimited, 1 = persistent client connections not used)

idle *idle* (type: **uint16**, optional, default: 15)

max. idle time between requests (0 = unlimited)

conn *conn* (type: **uint16**, optional, default: 4)

max. number of simultaneously open persistent connections to servers

language [*lang*];

Language of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

lang (type: **language**, optional, default: **EN**)

hand-off *addr*;

Next-hop proxy.

addr (type: **sock**)

client-ssl *params*;

Use SSL/TLS on the connection from a client.

params (type: **name of ssl-params**, see ssl(5))

server-ssl *params*;

Parameters of SSL/TLS for the connection to a server.

This item is valid only if the client SSL-PARAMS section forces faking of the server certificate for the client.

params (type: **name of ssl-params**, see ssl(5))

client-cert-match [**subject** *subject*] [**issuer** *issuer*];

Requirements for client certificate.

subject *subject* (type: **str-set**, optional, default: *)
acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)
acceptable certificate issuers

server-cert-match [**subject** *subject*] [**issuer** *issuer*];
Requirements for server certificate.

subject *subject* (type: **str-set**, optional, default: *)
acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)
acceptable certificate issuers

simulate-connect;
Behave as if the client issued a CONNECT request to the client connection destination address. Establish a TCP tunnel to the server. Note that the item CAPTURE-CONNECT cannot be used afterwards; use SNI-INSP instead of SIMULATE-CONNECT in such case.

sni-insp;
Obtain server name identifier for use in item sni-result of request ACLs.

aproxy *name*;
Act as an authentication proxy for HTTP servers.

name (type: **name** of **aproxy**, see above)

ntlm-auth *name*;
Enable NTLM authentication of clients.

name (type: **name** of **ntlm-auth**, see above)

kerberos-auth *name* [*proxy-host*];
Enable Kerberos authentication of clients.

name (type: **name** of **kerberos-auth**, see above)

proxy-host (type: **str**, optional, default: "")
The proxy host name expected in authentication data from clients. This is also the proxy host name that users should set in their browser configuration. The specified host name will be used in the proxy Kerberos principal (HTTP/proxy-host). If not set, the host name of the Kernun system will be used. This value overrides a value set in the referenced KERBEROS-AUTH section.

authenticate-at [*val*];
Whether the proxy will ask the client to perform proxy authentication (the default), or authentication at the origin server.

val (type: **auth-headers**, optional, default: proxy)

acl-error-status *code*;
Status code returned when a request is denied by ACL. If not set, default is 403 Forbidden.

code (type: **uint16**)

server-from-tcp [*val*];

Specifies which destination IP address is used for connecting to the server, in transparent sessions. By default the destination IP address of the original client TCP connection is used. If set to 'NO', the address specified inside the HTTP protocol (HTTP request) is used, i.e. the Host header or the host part of the request URI.

val **(type: yes-no, optional, default: yes)**

[End of section `http-proxy.session-acl` description.]

request-acl *name* **{**

* from ... ;
* server ... ;
* user ... ;
* time ... ;

time-period-set { ... }

* session-acl ... ;

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

source-address ... ;

plug-to ... ;

* request-method ... ;
* request-scheme ... ;
* request-uri ... ;
* request-path ... ;
* referer ... ;
* blacklist ... ;
* request-version ... ;
* user-agent ... ;
* client-cert-match ... ;
* aproxy-user ... ;
* clear-web-db-match ... ;

clear-web-db-bypass { ... }

web-filter ... ;

host-hdr-transp ... ;

* rewrite ... ;

http-host ... ;

uri-decode ... ;

hand-off ... ;

select-optimization ... ;

allow-req-hdr ... ;
delete-req-hdr-range ... ;
allow-resp-hdr ... ;
* add-req-hdr ... ;
* add-resp-hdr ... ;
ftp-force-utf-8 ... ;
* req-line-check ... ;
* req-hdr-check ... ;
* status-line-check ... ;
* resp-hdr-check ... ;
accept-gzip ... ;
content-gzip ... ;
request-via ... ;
response-via ... ;
request-time ... ;
language ... ;
auth-req ... ;
oob-add { ... }
max-bytes ... ;
dechunk-ignore-eof ... ;
server-ssl ... ;
* server-cert-match ... ;
sni-result ... ;
client-altq ... ;
server-altq ... ;
ip-tos-to-client { ... }
ip-tos-to-server { ... }
file-response ... ;
program-response ... ;
library-response { ... }
request-end-program ... ;
* program-env ... ;
connect-data-filter-client ... ;
connect-data-filter-server ... ;
capture-connect ... ;
* modify-cookies ... ;
delete-cookies ... ;
request-body-match ... ;

request-body-max-size ... ;

replace-authorization ... ;

acl-error-status ... ;

deny-msg ... ;

}

The `request-acl` section is derived from `acl-2` section prototype.
For detail description of it, see acl(5).
Changes to the `request-acl` section:

Item `parent-acl` used as `session-acl`.

SSL/TLS required on connection in order to match server certificates.

FILE-RESPONSE cannot be combined with HAND-OFF or PLUG-TO.

PROGRAM-RESPONSE cannot be combined with HAND-OFF or PLUG-TO.

LIBRARY-RESPONSE cannot be combined with HAND-OFF or PLUG-TO.

Only one of FILE-RESPONSE, PROGRAM-RESPONSE, and LIBRARY-RESPONSE can be set.

CAPTURE-CONNECT cannot be combined with AUTH-REQ, HAND-OFF, PLUG-TO, FILE-RESPONSE, PROGRAM-RESPONSE and LIBRARY-RESPONSE.

Items DENY and CLEAR-WEB-DB-BYPASS are mutually exclusive.

Item CLEAR-WEB-DB-BYPASS requires CLEAR-WEB-DB-MATCH.

Item DENY-MSG requires DENY.

Added items & subsections:
`source-address` [`client`] [`addr4` *addr4*] [`addr6` *addr6*] `cluster` [*cluster*];
`source-address` [`client`] [`addr4` *addr4*] [`addr6` *addr6*] [`physical`];
`source-address` [`client`] [`addr4` *addr4*] [`addr6` *addr6*] `no-fallback`;

Source address for outgoing connections to servers.

If omitted, the proper address of the proxy will be used, i.e. in the case of a cluster, the cluster address will be used.

If not specified by the SOURCE-PORT item, a generic port will be used.

The elements entered within this item will be used by the proxy until the first of them is applicable:

- The CLIENT keyword means the original client IP address is used. This mode will be succesful in all cases except mismatch of IP address families.

- The ADDR4/ADDR6 keyword-value pairs mean that the specified address is used for a connection of corresponding address family.

- The CLUSTER keyword means that one of cluster addresses will be used. By default, the main address of the bridge is used, however, any preferred alias address can be listed in the cluster list.- The PHYSICAL option means that the address of the physical interface is used instead of the cluster one.

- The DEFAULT option means the default behavior - i.e. using of the physical address.

- The NO-FALLBACK option means that if no other way of setting the address is acceptable, the session is rejected. Without this option, the system tries to find a suitable source IP address automatically.

client (type: **key**, optional)

addr4 *addr4* (type: **host**, optional, default: [0.0.0.0])

addr6 *addr6* (type: **host**, optional, default: [::])
<branching element> (type: **source-address-mode**, optional, default: physical)
cluster (type: **host-list**, optional, default: {})

Constraints:

Address family must respect the element's address family..

plug-to *addr*;

Final destination server.

addr (type: **sock**)

Address/port of final destination server.

If port is zero, then original port is used.

request-method *val*;

Request methods to which this ACL is applicable.

val (type: **str-set**)

request-scheme *val*;

Matching scheme from request URI. If URI does not contain scheme, SCHEME-HTTP is assumed.

val (type: **http-scheme-set**)

request-uri *val*;

Matching the whole request URI. Proxy URIs have form <SCHEME>://<HOST>[:PORT]<PATH>[?<QUERY>], e.g., http://www.tns.cz:80/kernun/index.html.

val (type: **str-set**)

request-path *val*;

Matching request URI path.

val (type: **str-set**)

referer *val*;

Matching Referer HTTP header.

val (type: **str-set**)

blacklist *categories*;

Select this ACL if request URI (server and possibly initial part of path) is found in the blacklist with at least one matching category.

categories (type: **str-set**)

request-version *val*;

HTTP protocol version in request.

val (type: **http-protocol-set**)

user-agent *val*;

Select an ACL according to the User-AgentHTTP header.

val (type: **str-set**)

client-cert-match [**subject** *subject*] [**issuer** *issuer*];

Select an ACL according to a client certificate.

subject *subject* (type: **str-set**, optional, default: *)

acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)

acceptable certificate issuers

aproxy-user none;

aproxy-user [**name**] [*name* [**group** *group*]];

User and group specification.

<branching element> (type: **user-auth-spec**, optional, default: name)

name (type: **str-set**, optional, default: *)

user name (authenticated on firewall)

group *group* (type: **str-set**, optional, default: *)

list of groups, if present, both NAME and GROUP must match

clear-web-db-match [**any**] *categories-set*;

clear-web-db-match all *categories-list*;

clear-web-db-match subset *categories-set*;

clear-web-db-match exact *categories-list*;

Clear Web Matching Control.

This item is used as an ACL entry condition for a URL based on Clear Web category matching.

<branching element> (type: **clear-web-db-match-mode**, optional, default: any)

categories-set (type: **clear-web-db-category-set**)

categories-list (type: **clear-web-db-category-list**)

clear-web-db-bypass {

status ... ;

cookie ... ;

activation ... ;

duration ... ;

}

The **clear-web-db-bypass** section is derived from **clear-web-db-bypass** section prototype. For detail description of it, see clear-web-db(5).

web-filter *name*;

Enables web filter and selects web filter server

name (type: **name** of **web-filter**, see above)

host-hdr-transp [enable-loopback];

Enables transparent proxy behavior for non-transparent connections. If the request URI contains only path and no server address, uses the content of the Host header as the server address.

enable-loopback (type: **key**, optional)

Permits connections to local addresses.

rewrite *match subst* **[redirect** *redirect***];**

Change request URI.

match (type: **regexp**)

URIs matching this regular expression

subst (type: **str**)

will be substituted with this ($1..$9 are references to parenthesized subexpressions of MATCH, $$ means $)

redirect *redirect* (type: **http-redirect**, optional, default: undefined)

the proxy will return HTTP redirect with the rewritten URI instead of directly serving the rewritten URI

http-host *addr*;

Change request URI host and Host header.

addr (type: **sock**)

uri-decode [*val***];**

Decode unreserved characters in a request URI encoded as %XX. This item must be set to NO for some clients or servers that require their URIs to be passed unmodified. Matching of REQUEST-URI and REQUEST-PATH is always done with decoded unreserved characters, regadless the value of this item.

val (type: **yes-no**, optional, default: no)

hand-off *addr*;

Next-hop proxy.

addr (type: **sock**)

select-optimization [*c2s* **[***s2c***]];**

Optimization of read/write/select operations.

c2s (type: **uint32**, optional, default: 0)

max. number of client->server reads/writes without calling select

s2c (type: **uint32**, optional, default: 0)

max. number of server->client reads/writes without calling select

allow-req-hdr *name*;

Pass only these request headers. If not used, all request headers.

name (type: **str-set**)

names of allowed headers

delete-req-hdr-range;

Remove request header Range.

allow-resp-hdr *name*;

Pass only these response headers.

name (type: **str-set**)

names of allowed headers

add-req-hdr *name value*;

Add a request header. The header is added literaly and does not replace any already existing header of the same name. The header is not checked for compliance with RFC.

name (type: **str**)

Header name.

value (type: **str**)

Header value.

add-resp-hdr *name value*;

Add a response header. The header is added literaly and does not replace any already existing header of the same name. The header is not checked for compliance with RFC.

name (type: **str**)

Header name.

value (type: **str**)

Header value.

ftp-force-utf-8;

When retrieving and displaying FTP directory, use UTF-8.
It means sending the "OPTS UTF8 ON" command to the server, and adding the "charset=UTF-8" info to the web page displayed.

req-line-check *text-match max-len*;

Check of request line.

text-match (type: **str-set**)

The line must match this.

max-len (type: **uint32**)

Maximum number of characters in the line

req-hdr-check *name-match val-match max-val-len*;

Check of request headers.

name-match (type: **str-set**)

The header name must match this.

val-match (type: **str-set**)

The header value must match this.

max-val-len (type: **uint32**)

Maximum number of characters of the header value

status-line-check *text-match max-len*;

Check of response status line.

text-match (type: **str-set**)
The line must match this.

max-len (type: **uint32**)
Maximum number of characters in the line

resp-hdr-check *name-match val-match max-val-len*;
Check of response headers.

name-match (type: **str-set**)
The header name must match this.

val-match (type: **str-set**)
The header value must match this.

max-val-len (type: **uint32**)
Maximum number of characters of the header value

accept-gzip [*gzip-only*];
Modify Accept-Encoding header so that only identity and gzip encodings will be accepted. This is useful if the response will be passed to HTML filter later.

gzip-only (type: **accept-gzip**, optional, default: client-add)

content-gzip [*mode*];
Convert between gzip and x-gzip in the Content-Encoding response header.

mode (type: **content-gzip**, optional, default: accept)

request-via pass;

request-via add [*faked*];

request-via full [*faked*];

request-via [**block**];

request-via fake [*faked*];
Processing Via headers in requests.

<branching element> (type: **proxy-via**, optional, default: block)

faked (type: **str**, optional, default: "")
full Via header value if FAKE, replacement for firewall hostname for ADD, FULL

response-via pass;

response-via add [*faked*];

response-via full [*faked*];

response-via [**block**];

response-via fake [*faked*];
Processing Via headers in responses.

<branching element> (type: **proxy-via**, optional, default: block)

faked (type: **str**, optional, default: "")
full Via header value if FAKE, replacement for firewall hostname for ADD, FULL

request-time *seconds*;

Limited time of single request.

seconds **(type: uint32)**

language *lang*;

Language of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

lang **(type: language)**

auth-req *realm*;

Send authentication request back to client.

realm **(type: str)**

realm sent to clients

oob-add {

timeout-idle ... ;

timeout-unauth ... ;

}

If this is an OOB authentication server and the user has been authenticated successfully, add the user to the OOB authentication session table.

Items & subsections:

timeout-idle [*sec***];**

Number of seconds until authentication expires when the client is idle (sends no requests).

sec **(type: uint32, optional, default: 360)**

timeout-unauth [*sec***];**

Maximum time after which authentication expires.

sec **(type: uint32, optional, default: 3600)**

[End of section `http-proxy.request-acl.oob-add` description.]

max-bytes [*cout* **[***cin* **[***sout* **[***sin***]]]];**

Maximum number of transferred bytes (0 = unlimited).

cout **(type: uint64, optional, default: 0)**

(received from client)

cin **(type: uint64, optional, default: 0)**

(sent to client)

sout **(type: uint64, optional, default: 0)**

(sent to server)

sin **(type: uint64, optional, default: 0)**

(received from server)

dechunk-ignore-eof [*val***];**

Ignore premature connection close by a server when waiting for a chunk (behave as if there was a terminating zero-sized chunk).

val (type: **yes-no**, optional, default: **no**)

server-ssl *params*;

Use SSL/TLS on the connection to a server.

params (type: **name** of **ssl-params**, see ssl(5))

server-cert-match [**subject** *subject*] [**issuer** *issuer*];

Requirements for server certificate.

subject *subject* (type: **str-set**, optional, default: *)

acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)

acceptable certificate issuers

sni-result [*result*];

Result of the SNI inspection. Default value is "specified".

result (type: **sni-result-set**, optional, default: {})

client-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to client.

altq (type: **name** of **pf-queue**, see pf-queue(5))

queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: **NULL**)

priority queue name (if set, used for TCP ACK without data)

server-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to server.

altq (type: **name** of **pf-queue**, see pf-queue(5))

queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: **NULL**)

priority queue name (if set, used for TCP ACK without data)

ip-tos-to-client {

fixed ... ;

received ... ;

other ... ;

}

The ip-tos-to-client section is derived from ip-tos-to-client section prototype. For detail description of it, see netio(5).

ip-tos-to-server {

fixed ... ;

received ... ;

other ... ;

}

The `ip-tos-to-server` section is derived from `ip-tos-to-server` section prototype. For detail description of it, see netio(5).

`file-response` [`status-code` *status-code*] [`mime-type` *mime-type*] [*path* [`request-uri`]];

Do not contact a server and send a local file obtained by concatenating the document-root path and the specified file path or the request URI path.

`status-code` *status-code* (type: `uint16`, optional, default: 200)

status code of response

`mime-type` *mime-type* (type: `str`, optional, default: "text/html")

Document Content-Type.

***path* (type: `str`, optional, default: "")**

Path of the response file (under DOCUMENT-ROOT).

`request-uri` (type: `key`, optional)

Append request URI path after request (return file selected by request URI in subdirectory PATH of DOCUMENT-ROOT).

`program-response` *path* [*runtime*];

Do not contact a server and start a program. The program is passed the complete request to the standard input. The standard output of the program is sent to the client. The program mustgenerate a complete response beginning with a response lineand headers.

***path* (type: `name` of `shared-file`, see common(5))**

Program name.

***runtime* (type: `uint32`, optional, default: 10)**

Maximum allowed runtime of the program (seconds). After expiration, the program is signalled by SIGTERM. If zero, the allowed runtime is unlimited.

`library-response` {

lib ... ;

* param ... ;

}

Constraints:

Item LIB must be specified.

Items & subsections:

`lib` *path*;

Path to the shared library.

***path* (type: `str`)**

`param` *name value*;

Optional parameters for the library.

***name* (type: `str`)**

Parameter name.

value **(type: str)**

Parameter value.

[End of section `http-proxy.request-acl.library-response` description.]

request-end-program *path*;

A program executed asynchronously by the proxy when a request processing finishes.

path **(type: name of shared-file, see** common**(5))**

Program name.

program-env *name val*;

Additional environment variables passed to programs executed PROGRAM-RESPONSE and REQUEST-END-PROGRAM.

name **(type: str)**

Name of the environment variable.

val **(type: str)**

Value of the environment variable.

connect-data-filter-client *rules*;

Client CONNECT data filtering.

rules **(type: name of data-match, see** mod-match**(5))**

connect-data-filter-server *rules*;

Server CONNECT data filtering.

rules **(type: name of data-match, see** mod-match**(5))**

capture-connect [stats-log *stats-log*];

In the case of a CONNECT request, restart the session and handle the communication itself instead of creating a TCP tunnel to the destination server.

stats-log *stats-log* **(type: yes-no, optional, default: no)**

Whether statistics log message is produced for capture-connect request itself. Stats messages for the requests performed later in this 'caputre-connected' session are not affected by this value, they are logged anyway.

modify-cookies *name* **[params** *params*] **[max-age** *max-age*] **[any-client] [keep-not-found];**

Matching cookies will be modified so that they are useless outside the internal network.

name **(type: str-set)**

Modify only cookies with matching name.

params *params* **(type: str-set, optional, default: *)**

Modify only cookies with matching parameters. Is matched against the string containing all cookie parameters following the cookie NAME and VALUE.

max-age *max-age* **(type: uint32, optional, default: 0)**

Cookie lifetime (seconds). If zero or not set, the value from HTTP-PROXY.COOKIE-TABLE.MAX-AGE is used. If a cookie itself specifies shorter lifetime, the shorter value is used instead.

any-client (type: **key**, optional)

Modify a cookie even if it comes from other client address than it was previously sent to.

keep-not-found (type: **key**, optional)

Keep the cookie value unchanged if not found in the cookie table. Otherwise, values of unknown cookies are set to the empty string.

delete-cookies [ip-from-query];

Delete all cookies for the client IP in the cookie table.

ip-from-query (type: **key**, optional)

Use IP address written in the query part of the request URI instead of the client IP address.

request-body-match *rules*;

Rules for matching content of the request body.

rules (type: **name of data-match**, see mod-match(5))

request-body-max-size *bytes*;

Maximum size of the request body.

bytes (type: **uint64**)

replace-authorization [simple] *db-file* **[keep-not-found] [replace-not-found** *replace-not-found***];**
replace-authorization radius *db-file* **[keep-not-found] [replace-not-found** *replace-not-found***] radius** *radius*
radius-delimiter *radius-delimiter*;

Replaces user name and password in a request Authorization header.

<**branching element**> (type: **replace-authorization-mode**, optional, default: **simple**)

Mode of the authorization replacement

db-file (type: **str**)

A database file for replacement. It must be in the format used by action HTML-REPLACE in REQUEST-BODY-MATCH.RULES. Each record of the replacement database must contain two values: user name and password.

keep-not-found (type: **key**, optional)

If set and replacement values are not found in the database, pass the selected HTML form controls unchanged. Otherwise, values of the selected controls are deleted.

replace-not-found *replace-not-found* (type: **str**, optional, default: <**NULL**>)

If set and replacement values are not found in the database, replace values of the selected HTML form controls by this value.

radius *radius* (type: **name of radius-client**, see radius(5))

Radius client configuration to be used for authentication.

radius-delimiter *radius-delimiter* (type: **str**)

A single character used as delimiter of the internal password and the radius password. Last occurence of the delimiter is used. If not present in the particular password, all the password text is interpreted as the radius password.

Constraints:

Only one of elements KEEP-NOT-FOUND and REPLACE-NOT-FOUND may be specified.

acl-error-status *code*;

Status code returned when a request is denied by ACL. If not set, default is 403 Forbidden.

code **(type: uint16)**

deny-msg [template *template*] [*msg*];

The error page returned when a request is denied by ACL. If not set, a generic "denied by security policy" page is returned, except when the denying ACL contains item CLEAR-WEB-DB-MATCH. In this case, the returned page will display the categories of the request URI that match the CLEAR-WEB-DB-MATCH item. See also http-proxy(8) for the instructions how to create custom error pages.

template *template* **(type: str, optional, default: "")**

error page template file name, without the .html.LANGUAGE-CHARSET suffix, relative to the DOCUMENT-ROOT directory

msg **(type: str, optional, default: "")**

a message inserted into the error page template

[End of section `http-proxy.request-acl` description.]

doc-acl *name* {

* from ... ;
* server ... ;
* user ... ;
* time ... ;

time-period-set { ... }

* request-acl ... ;

deny ... ;

accept ... ;

rule ... ;

* content-type ... ;
* mime-type ... ;

force-doctype-ident ... ;

html-filter ... ;

* request-scheme ... ;
* referer ... ;
* request-path ... ;
* blacklist ... ;

set-mime-type ... ;

replace-response ... ;

jpeg-scan-sz ... ;
* filter-images ... ;
antivirus ... ;
accept-antivirus-status ... ;
response-body-match ... ;
client-altq ... ;
server-altq ... ;
deny-msg ... ;
}

The `doc-acl` section is derived from `acl-3` section prototype. For detail description of it, see acl(5).
Changes to the `doc-acl` section:

Item `parent-acl` used as `request-acl`.
Item `direction` is not valid.
Item `size` is not valid.
Item `virus-status` is not valid.
Item `modify-header` is not valid.
Item `replace` is not valid.
SET-MIME-TYPE and REPLACE-RESPONSE are mutually exclusive.
SET-MIME-TYPE and FORCE-DOCTYPE-IDENT are mutually exclusive.
REPLACE-RESPONSE and FORCE-DOCTYPE-IDENT are mutually exclusive.
Item DENY-MSG requires DENY..

Added items & subsections:

`request-scheme` *val*;

Matching scheme from request URI. If URI does not contain scheme, SCHEME-HTTP is assumed.

val **(type: `http-scheme-set`)**

`referer` *val*;

Matching Referer HTTP header.

val **(type: `str-set`)**

`request-path` *val*;

Matching request URI path. Proxy URIs have form [<SCHEME>://<HOST>[:PORT]]<PATH>[?<QUERY>], e.g., http://www.tns.cz:80/kernun/index.html

val **(type: `str-set`)**

`blacklist` *categories*;

Select this ACL if request URI (server and possibly initial part of path) is found in the blacklist with at least one matching category.

categories **(type: `str-set`)**

`set-mime-type` *val*;

Set Content-Type header sent to the client.

val **(type: str)**

replace-response *status-code mime-type path*;

Send a local file instead of a server response.

status-code **(type: uint16)**

status code of response

mime-type **(type: str)**

Document Content-Type.

path **(type: str)**

file to send (relative to document-root if relative path)

jpeg-scan-sz [*sz*];

Maximum length of the initial part of an JPEG image file scanned for image type and size.

sz **(type: uint16, optional, default: 1024)**

filter-images *width height path*;

Send a local file instead of some images (MIME types image/gif, image/jpeg, image/png). Matching is done according to image size, image which has invalid format orunknown size is treated as 0 x 0 image for matching.

width **(type: uint16-set)**

images width

height **(type: uint16-set)**

images height

path **(type: str)**

file to send (relative to document-root if relative path), an extension according to type (.gif/.jpeg/.png) will be appended

antivirus *channel* [**interval** *interval*] [**chunk** *chunk*] [**limit** *limit*];

Antivirus usage mode.

Check document by antivirus, with settings for passing initial part of unchecked data through the antivirus module during antivirus checking.

channel **(type: name-list of antivirus, see** antivirus**(5))**

Name of ANTIVIRUS global section used.

interval *interval* **(type: uint16, optional, default: 0)**

Seconds between passing blocks of unchecked data (0 = do not send unchecked data).

chunk *chunk* **(type: uint32, optional, default: 0)**

Size of each block of unchecked data.

limit *limit* **(type: uint32, optional, default: 0)**

Maximum size of unchecked data passed before antivirus check is completed. Remaining data will be passed only after successful checking.

accept-antivirus-status *status*;

Defines set of antivirus status codes (in addition to FREE) that allow further passing of data. Other status codes cause termination of data transfer. If not set, data are passed only if the antivirus returns status FREE.

status (type: **virus-status-set**)

response-body-match *rules*;

Rules for matching content of the response body.

rules (type: **name of data-match**, see mod-match(5))

client-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to client.

altq (type: **name of pf-queue**, see pf-queue(5))

queue name

paltq *paltq* (type: **name of pf-queue**, see pf-queue(5), optional, default: NULL)

priority queue name (if set, used for TCP ACK without data)

server-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to server.

altq (type: **name of pf-queue**, see pf-queue(5))

queue name

paltq *paltq* (type: **name of pf-queue**, see pf-queue(5), optional, default: NULL)

priority queue name (if set, used for TCP ACK without data)

deny-msg [**template** *template*] [*msg*];

The error page returned when a request is denied by ACL. If not set, a generic "denied by security policy" page is returned, except when the denying ACL contains item CLEAR-WEB-DB-MATCH. In this case, the returned page will display the categories of the request URI that match the CLEAR-WEB-DB-MATCH item. See also http-proxy(8) for the instructions how to create custom error pages.

template *template* (type: **str**, optional, default: "")

error page template file name, without the .html.LANGUAGE-CHARSET suffix, relative to the DOCUMENT-ROOT directory

msg (type: **str**, optional, default: "")

a message inserted into the error page template

[End of section `http-proxy.doc-acl` description.]

[End of section `http-proxy` description.]

SEE ALSO

configuration(7), acl(5), antivirus(5), application(5), auth(5), clear-web-db(5), common(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), mod-match(5), monitoring(5), netio(5), pf-queue(5), radius(5), source-address(5), ssl(5), tcpserver(5), time(5), auth(7), http-proxy(8)

NAME

http-proxy.cfg — format of http-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **http-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **http-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

language (see **common**(5))

nls (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see ipc(5))

radius-attr (see radius(5))

ldap-tls-reqcert-mode (see ldap(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

antivirus-protocol (see antivirus(5))

virus-status (see antivirus(5))

database-source (see antivirus(5))

source-address-mode (see source-address(5))

accept-deny (see mod-html-filter(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

proc-priority (see application(5))

ssl-ver (see ssl(5))

extension-op (see ssl(5))

veri-fail-action (see ssl(5))

auth-cert-type (see ssl(5))

distrusted-cert-type (see ssl(5))

data-match-action (see mod-match(5))

clear-web-db-category (see clear-web-db(5))

clear-web-db-match-mode (see clear-web-db(5))

replace-authorization-mode (see http-proxy(5))

proxy-via (see http-proxy(5))

http-protocol (see http-proxy(5))

http-scheme (see http-proxy(5))

cookie-table-clean (see http-proxy(5))

accept-gzip (see http-proxy(5))

content-gzip (see http-proxy(5))

http-redirect (see http-proxy(5))

kerberos-user-match (see http-proxy(5))

ldap-select (see http-proxy(5))

auth-headers (see http-proxy(5))

sni-result (see http-proxy(5))

ITEMS AND SECTIONS

Program **http-proxy** recognizes following items and sections:

admin ... ;
* antivirus *name* { ... }
* aproxy *name* { ... }

clear-web-db { ... }
* data-match *name* { ... }
* fake-cert *name* { ... }
* html-filter *name* { ... }
* interface *name* { ... }
* kerberos-auth *name* { ... }
* ldap-client-auth *name* { ... }
* ntlm-auth *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
* ssl-params *name* { ... }

sysctl { ... }
use-resolver ... ;
* web-filter *name* { ... }
* http-proxy *name* { ... }
ipv6-mode ... ;

Description:

admin *system* [*contact*];

Firewall administrator and contact e-mail addresses.

system **(type: str)**

The technical administrator(s) of the system; an address or set of comma separated adresses of persons responsible for system maintenance.

contact **(type: str, optional, default: <NULL>)**

The policy administator; an address of person responsible for system configuration. If not defined, the technical administration is used instead.

Constraints:

Administrator contact must comply with RFC.

antivirus *name* {

connection ... ;
sock-opt { ... }
timeout ... ;
comm-dir ... ;
altq ... ;
max-checked-size ... ;
icap-pass-200-with-pure-body ... ;
persistent-stream ... ;
clamav-agent { ... }
}

The antivirus section is derived from antivirus section prototype.
For detail description of it, see antivirus(5).

aproxy *name* {

auth ... ;
insecure-cookies ... ;
oob-auth ... ;

```
    cookie-name ... ;
    logout ... ;
    timeout-idle ... ;
    timeout-unauth ... ;
    bufsz ... ;
  }
```

The `aproxy` section is derived from `aproxy` section prototype. For detail description of it, see http-proxy(5).

`clear-web-db` {

```
    internal-servers ... ;
    db ... ;
    lock ... ;
    local-db { ... }
  }
```

The `clear-web-db` section is derived from `clear-web-db` section prototype. For detail description of it, see clear-web-db(5).

`data-match` *name* {

```
    max-size ... ;
    init-match ... ;
    max-match ... ;
    step-size ... ;
    step-match ... ;
  * test ... ;
  }
```

The `data-match` section is derived from `data-match` section prototype. For detail description of it, see mod-match(5).

`fake-cert` *name* {

```
    key ... ;
    auth-ca ... ;
    fail-ca ... ;
  * extension ... ;
    purge ... ;
```

}

The `fake-cert` section is derived from `fake-cert` section prototype. For detail description of it, see ssl(5).

`html-filter` *name* {

* script-tag-language ... ;
  replace-head-script-tags ... ;
  replace-body-script-tags ... ;
* style-tag-type ... ;
  replace-style-tags ... ;
* iframe-tag-src ... ;
  replace-iframe-tags ... ;
* intrinsic-language ... ;
* intrinsic-hack ... ;
  replace-intrinsic ... ;
* macro-language ... ;
* macro-hack ... ;
  replace-macros ... ;
* uri ... ;
  replace-uri ... ;
* embed-tag-type ... ;
* embed-src-hack ... ;
* embed-plugin-hack ... ;
  replace-head-embed-tags ... ;
  replace-body-embed-tags ... ;
* applet ... ;
  replace-applets ... ;
* object ... ;
* object-classid-hack ... ;
* object-data-hack ... ;
  replace-head-object-tags ... ;
  replace-body-object-tags ... ;
* param-tags ... ;
  replace-param ... ;
  script-end-hack ... ;

}

The `html-filter` section is derived from `html-filter` section prototype. For detail description of it, see mod-html-filter(5).

`interface` *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`kerberos-auth` *name* {

domain ... ;

user-match ... ;

kinit ... ;

keytab ... ;

proxy-host ... ;

* ad-controller ... ;

ldap ... ;

timeout-idle ... ;

timeout-unauth ... ;

lock ... ;

lock-ldap ... ;

one-per-session ... ;

}

The `kerberos-auth` section is derived from `kerberos-auth` section prototype. For detail description of it, see http-proxy(5).

`ldap-client-auth` *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

`ntlm-auth` *name* {

domain ... ;

workgroup ... ;

* ad-controller ... ;

interfaces { ... }

ldap ... ;

timeout ... ;

timeout-idle ... ;

timeout-unauth ... ;

}

The `ntlm-auth` section is derived from `ntlm-auth` section prototype. For detail description of it, see http-proxy(5).

`oob-auth` *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

```
}
```

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

```
pf-queue name {
    parent ... ;
    bandwidth ... ;
    priority ... ;
    qlimit ... ;
    cbq { ... }
    priq { ... }
    hfsc { ... }
}
```

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

```
radius-client name {
    nas ... ;
    groups ... ;
  * server ... ;
}
```

The `radius-client` section is derived from `radius-client` section prototype. For detail description of it, see radius(5).

```
resolver name {
  * server ... ;
    search ... ;
    preference ... ;
    edns ... ;
    conf-timeout ... ;
    initial-timeout ... ;
    final-timeout ... ;
    conn-timeout ... ;
    disable-deresolution ... ;
}
```

The **resolver** section is derived from **resolver** section prototype. For detail description of it, see resolver(5).

shared-dir *name* {

path ... ;

}

The **shared-dir** section is derived from **shared-dir** section prototype. For detail description of it, see common(5).

shared-file *name* {

path ... ;

format ... ;

}

The **shared-file** section is derived from **shared-file** section prototype. For detail description of it, see common(5).

ssl-params *name* {

versions ... ;

ciphers ... ;

tcp-eof ... ;

id ... ;

* auth-cert ... ;

distrusted-certs ... ;

dont-check-crl ... ;

* crl ... ;

verify-peer ... ;

cache-timeout ... ;

use-ticket ... ;

enable-renegotiation ... ;

fake-cert ... ;

prefer server ciphers ... ;

enable-ecdh ... ;

}

The **ssl-params** section is derived from **ssl-params** section prototype. For detail description of it, see ssl(5).

sysctl {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

use-resolver *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

web-filter *name* **{**

connection ... ;

fail-ok ... ;

sock-opt { ... }

}

The `web-filter` section is derived from `web-filter` section prototype. For detail description of it, see http-proxy(5).

http-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

document-root ... ;

hdr-line-len ... ;

blacklist-db ... ;

connect-data-mime-db ... ;

ftp-proxy ... ;

max-aproxy-sessions ... ;

max-bypass-sessions ... ;

oob-auth-srv ... ;

ssl-session-cache { ... }

aproxy-lock ... ;

cookie-table { ... }

extended-status ... ;

* session-acl *name* { ... }

* request-acl *name* { ... }

* doc-acl *name* { ... }

}

The `http-proxy` section is derived from `http-proxy` section prototype. For detail description of it, see http-proxy(5).

ipv6-mode [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: enabling, optional, default: enable)**

SEE ALSO

configuration(7), http-proxy(8), acl(5), antivirus(5), application(5), auth(5), clear-web-db(5), common(5), http-proxy(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), mod-html-filter(5), mod-match(5), pf-queue(5), radius(5), resolver(5), source-address(5), ssl(5), sysctl(5), time(5), host-matching(7)

NAME

ica — format of ica component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ica** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **ica** configuration directives:

ip-version (see **common**(5))

osi4-proto (see **common**(5))

listen-on-sock (see **listen-on**(5))

ITEMS AND SECTIONS

Configuration of **ica** library component consists of following prototypes:

icamd { ... }
icasd { ... }

Description:

icamd {

phase ... ;

* tag ... ;

listen-on { ... }

priv-key ... ;

* slave *name* { ... }

}

Kernun inter-node communication master. ICAM allows this node to control other KERNUN device(s), which run the ICAS daemon.

Constraints:

Listen-on must be specified.

Private key must be specified.

Items & subsections:

phase [*number*];

Application Startup Phase.

number **(type: uint8, optional, default: 20)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: str)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

listen-on {

* socket ... ;

}

The listen-on section is derived from listen-on section prototype.
For detail description of it, see listen-on(5).
Changes to the listen-on section:

Item `non-transparent` used as `socket`.

Item `transparent` is not valid.

At least one address to listen on must be specified.

Item socket (see listen-on(5))

Element `version` is optional, default: ipv4=4.

Element `proto` is optional, default: tcp.

priv-key *id-rsa*;

Private SSH key used by the icamd daemon

id-rsa **(type: name of shared-file**, see common(5))

slave *name* {

pub-key ... ;

}

ICAS slave allowed to connect to this master

Constraints:

Public key must be specified.

Items & subsections:

pub-key *id-rsa-pub*;

Public SSH key of the slave used to check its identity

id-rsa-pub (type: **name** of **shared-file**, see common(5))

[End of section `icamd.slave` description.]

[End of section `icamd` description.]

icasd {

phase ... ;

* tag ... ;

priv-key ... ;

* master *name* { ... }

}

Kernun inter-node communication slave. ICAS allows this node to be controlled by other KERNUN device(s), which run the ICAM daemon.

Constraints:

Private key must be specified.

Items & subsections:

phase [*number*];

Application Startup Phase.

number (type: **uint8**, optional, default: 20)

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value (type: **str**)

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

priv-key *id-rsa*;

Private SSH key used by the icas daemon

id-rsa (type: **name** of **shared-file**, see common(5))

master *name* {

addr ... ;

pub-key ... ;

}

ICAS master to connect to

Constraints:

Address of the icas daemon must be specified.

Public key must be specified.

Items & subsections:

addr *addr*;

Address and port to connect to

addr **(type: sock)**

pub-key *id-rsa-pub*;

Public SSH key of the master used to check its identity

id-rsa-pub **(type: name of shared-file, see common(5))**

[End of section `icasd.master` description.]

[End of section `icasd` description.]

SEE ALSO

configuration(7), common(5), listen-on(5)

NAME

icap-server — format of icap-server component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **icap-server** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **icap-server** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

language (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

auth-method (see **auth**(5))

user-match-mode (see auth(5))

virus-status (see antivirus(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

listen-on-sock (see listen-on(5))

clear-web-db-category (see clear-web-db(5))

clear-web-db-match-mode (see clear-web-db(5))

ITEMS AND SECTIONS

Configuration of **icap-server** library component consists of following prototypes:
* icap-server *name* { ... }

Description:

icap-server *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

doctype-identification { ... }

client-conn { ... }

document-root ... ;

hdr-line-len ... ;

preview ... ;

blacklist-db ... ;

max-bypass-sessions ... ;

ssl-session-cache { ... }

ldap-cache { ... }

* session-acl *name* { ... }

* service-acl *name* { ... }

* request-acl *name* { ... }

* doc-acl *name* { ... }

}

This section defines ICAP server attributes.

The `icap-server` section is derived from `proxy` section prototype.
For detail description of it, see application(5).
Changes to the `icap-server` section:

Section `udpserver` is not valid.

Item `source-address` is not valid.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one SERVICE-ACL must be specified.

At least one REQUEST-ACL must be specified.

At least one DOC-ACL must be specified.

Document root path required.

Section `monitoring` (see monitoring(5))

Item `aproxy-user` is not valid.

Item `data` used as `uri`.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 1344.

Element `proto` is optional, default: tcp.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 1344.

Element `proto` is optional, default: tcp.

Added items & subsections:

client-conn {

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection from client options.

The client-conn section is derived from sock-opt section prototype. For detail description of it, see netio(5).

Changes to the client-conn section:

Item `conn-timeout` is not valid.

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

document-root *path*;

Directory containing error documents and forced responses.

path **(type: name of shared-dir, see common(5))**

hdr-line-len [*bytes*];

Maximum (multi)line length in ICAP message headers (sets buffer size in header processing modules).

bytes **(type: uint32, optional, default: 12Ki)**

preview disable;

preview [enable] *bytes*;

Default document preview mode and size.

If used, this setting is valid whenever no re-setting is defined in a particular SERVICE-ACL.

If not used, just the particular SERVICE-ACL setting is significant for the preview mode and size.

<branching element> (type: enabling, optional, default: enable)

bytes **(type: uint32)**

blacklist-db *fname*;

Blacklist categorization database file.

fname **(type: str)**

max-bypass-sessions [*num*];

Maximum number of simultaneous Clear Web DataBase bypass sessions.

num **(type: uint32, optional, default: 1000)**

ssl-session-cache {

capacity ... ;

dir ... ;

lock ... ;

}

The ssl-session-cache section is derived from ssl-session-cache section prototype. For detail description of it, see ssl(5).

ldap-cache {

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

timeout ... ;

file ... ;

lock ... ;

}

Use cache for LDAP groups results.

Items & subsections:

max-sessions [*val*];

Maximum number of simultaneously active LDAP user sessions.

***val* (type: uint16, optional, default: 200)**

Constraints:

MAX-SESSIONS must be nonzero.

max-user [*val*];

Maximum length of a user name.

***val* (type: uint16, optional, default: 48)**

Constraints:

MAX-USER must be nonzero.

max-groups [*val*];

Maximum space used by a list of groups for a single user.

Each group name of length L takes L+1 characters from this space.

***val* (type: uint16, optional, default: 1024)**

Constraints:

MAX-GROUPS must be nonzero.

truncate-groups;

Too long group list handling flag.

If used, a too long list of groups is truncated.

If omitted, the user cannot authenticate if his/her list of groups does not fit to the space allocated according to MAX-GROUPS.

timeout [*sec*];

User record expiration timeout.

sec **(type: uint32, optional, default: 3600)**

file [*path*];

LDAP caching file name.

path **(type: str, optional, default: "/tmp/ldap-cache")**

lock none;

lock semaphore;

lock lock2 [*path*];

lock [**multilock2**] [*path*];

An alternative implemetation of locks.

<branching element> (type: lock-type, optional, default: multilock2)

path **(type: str, optional, default: <NULL>)**

If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

[End of section `icap-server.ldap-cache` description.]

session-acl *name* **{**

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

auth ... ;

idle-timeout ... ;

source-address ... ;

linger-time ... ;

client-keepalive ... ;

language ... ;

client-ssl ... ;

* client-cert-match ... ;

}

The `session-acl` section is derived from `acl-1` section prototype.
For detail description of it, see acl(5).
Changes to the `session-acl` section:

Item `user` is not valid.

Item `idle-timeout-peer` is not valid.

Item `plug-to` is not valid.

SSL/TLS required on connection in order to match client certificates.

Item `auth` (see auth(5))

Element `mode` is optional, default: allowed.

Added items & subsections:

`linger-time` *seconds*;

Read end of client connection will linger for EOF after closing the write end.

seconds **(type: `uint32`)**

`client-keepalive` [`req` *req*] [`idle` *idle*];

Parameters of client connection keep-alive.

`req` *req* (type: `uint16`, optional, default: 100)

Max. number of requests on single connection (0 = unlimited, 1 = persistent client connections not used).

`idle` *idle* (type: `uint16`, optional, default: 15)

Max. idle time before the first request or between requests on single connection (0 = unlimited).

`language` [*lang*];

Language of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

lang **(type: `language`, optional, default: EN)**

`client-ssl` *params*;

Use SSL/TLS on the connection from a client.

params **(type: `name` of `ssl-params`, see ssl(5))**

`client-cert-match` [`subject` *subject*] [`issuer` *issuer*];

Requirements for client certificate.

`subject` *subject* (type: `str-set`, optional, default: *)

acceptable certificate subjects

`issuer` *issuer* (type: `str-set`, optional, default: *)

acceptable certificate issuers

[End of section `icap-server.session-acl` description.]

`service-acl` *name* {

* from ... ;

* server ... ;

* user ... ;

```
* time ... ;
  time-period-set { ... }
* session-acl ... ;
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
* request-method ... ;
* request-uri ... ;
* request-path ... ;
* user-agent ... ;
* client-cert-match ... ;
  use-antivirus ... ;
  request-time ... ;
  language ... ;
  preview ... ;
  send-opt-clearweb ... ;
  auth-req ... ;
  user-match ... ;
  ldap-groups ... ;
  max-bytes ... ;
  flush ... ;
  client-altq ... ;
}
```

The ICAP service sublevel of the second level of ACL.

This sublevel decides about particular ICAP service, i.e. types of document inspection required. It exploits data from ICAP protocol level.

The `service-acl` section is derived from `acl-2` section prototype.

For detail description of it, see acl(5).

Changes to the `service-acl` section:

Item `parent-acl` used as `session-acl`.

Added items & subsections:

`request-method` *val*;

Entry condition - ICAP request method.

***val* (type: `str-set`)**

`request-uri` *val*;

Entry condition - ICAP request URI.

ICAP URIs have form icap://[<USER>@]<HOST>[:<PORT>]<PATH>[?<QUERY>] (e.g., icap://icap.tns.cz/av-scan).

val (type: **str-set**)

request-path *val*;

Entry condition - ICAP URI path.

val (type: **str-set**)

user-agent *val*;

Entry condition - User-Agent ICAP header.

val (type: **str-set**)

client-cert-match [**subject** *subject*] [**issuer** *issuer*];

Entry condition - client certificate.

subject *subject* (type: **str-set**, optional, default: *)

acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)

acceptable certificate issuers

use-antivirus disable [**interval** *interval*] [**chunk** *chunk*] [**limit** *limit*];
use-antivirus enable *channel* [**interval** *interval*] [**chunk** *chunk*] [**limit** *limit*];

Antivirus usage mode.

If omitted, or disabled, no antivirus is enabled. In this case, neither any ANTIVIRUS global section can be present nor any ACL can have VIRUS item specified.

If enabled, it can be configured for passing initial part of unchecked data to the client before the antivirus check is completed. In this case, if a virus is found later, the connection to the client is broken.

<**branching element**> (type: **enabling**)

channel (type: **name-list of antivirus**, see antivirus(5))

interval *interval* (type: **uint16**, optional, default: **0**)

Seconds between passing blocks of unchecked data (0 = do not send unchecked data).

chunk *chunk* (type: **uint32**, optional, default: **0**)

Size of each block of unchecked data.

limit *limit* (type: **uint32**, optional, default: **0**)

Maximum size of unchecked data passed before antivirus check is completed. Remaining data will be passed only after successful checking.

request-time *seconds*;

Limited time of single request.

seconds (type: **uint32**)

language *lang*;

Language of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

lang (type: **language**)

preview disable;

preview [enable] *bytes*;

Document preview mode and size.

If used, this setting is valid for the OPTIONS request response belonging to this SERVICE-ACL.

If not used and the global setting (ICAP-SERVER level) is defined, the global setting is valid.

If no PREVIEW setting is specified on neither ICAP-SERVER nor SERVICE-ACL level, the OPTIONS response is derived by the icap-server according to services offered (document type identification by the magic library, antivirus scanning, etc.).

<**branching element**> (type: **enabling**, optional, default: **enable**)

bytes (type: **uint32**)

send-opt-clearweb;

OPTIONS response control.

If used, the server includes clearweb categories into the response to the OPTIONS request.

auth-req *realm*;

Send authentication request back to client.

realm (type: **str**)

realm sent to clients

user-match [*mode***];**

ACL matching mode of authenticated usernames.

mode (type: **user-match-mode**, optional, default: **short**)

ldap-groups [*name***];**

Get list of user groups by LDAP.

name (type: **name** of **ldap-client-auth**, see ldap(5), optional, default: **NULL**)

Default LDAP connection for the case of no user domain sent by the ICAP client.

max-bytes [*cout* **[***cin***]];**

Maximum number of transferred bytes (0 = unlimited).

cout (type: **uint64**, optional, default: **0**)

Bytes received from client.

cin (type: **uint64**, optional, default: **0**)

Bytes sent to client.

flush client;

Force immediate data flushing to network.

client (type: **key**)

Flush data to client.

client-altq *altq* **[paltq** *paltq***];**

ALTQ queues for data sent to client.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

`paltq` *paltq* (type: `name` of `pf-queue`, see pf-queue(5), optional, default: NULL)

priority queue name (if set, used for TCP ACK without data)

[End of section `icap-server.service-acl` description.]

`request-acl` *name* {

* from ... ;
* server ... ;
* user ... ;
* time ... ;

time-period-set { ... }

* service-acl ... ;

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

* request-method ... ;
* request-uri ... ;
* request-scheme ... ;
* request-path ... ;
* blacklist ... ;
* clear-web-db-match ... ;

clear-web-db-bypass { ... }

replace-response ... ;

deny-msg ... ;

language ... ;

client-altq ... ;

}

The HTTP request sublevel of the second level of ACL.

This sublevel decides about particular document inspection methods according to the encapsulated document HTTP data.

The `request-acl` section is derived from `acl-2` section prototype.
For detail description of it, see acl(5).
Changes to the `request-acl` section:

Item `parent-acl` used as `service-acl`.

Items DENY and CLEAR-WEB-DB-BYPASS are mutually exclusive.

Item CLEAR-WEB-DB-BYPASS requires CLEAR-WEB-DB-MATCH.

Items REPLACE-RESPONSE and CLEAR-WEB-DB-BYPASS are mutually exclusive.

Item DENY-MSG requires DENY.

Added items & subsections:

request-method *val*;

Entry condition - HTTP request method.

val (type: **str-set**)

request-uri *val*;

Entry condition - HTTP message URI.

Encapsulated HTTP messages have URI in form <SCHEME>://<HOST>[:PORT]<PATH>[?<QUERY>] (e.g., http://www.tns.cz:80/kernun/index.html).

val (type: **str-set**)

request-scheme *val*;

Entry condition - HTTP message URI scheme.

If URI does not contain scheme, scheme HTTP is assumed.

val (type: **str-set**)

request-path *val*;

Entry condition - HTTP message URI path.

val (type: **str-set**)

blacklist *categories*;

Entry condition - HTTP message URI blacklist category.

If used, this ACL is selected if request URI (server and possibly initial part of path) is found in the blacklist with at least one matching category.

categories (type: **str-set**)

clear-web-db-match [any] *categories-set*;

clear-web-db-match all *categories-list*;

clear-web-db-match subset *categories-set*;

clear-web-db-match exact *categories-list*;

Entry condition - HTTP message URI Clear Web category.

If used, this ACL is selected if the request URI is found in the Clear Web DataBase with at least one matching category.

<**branching element**> (type: **clear-web-db-match-mode**, optional, default: any)

categories-set (type: **clear-web-db-category-set**)

categories-list (type: **clear-web-db-category-list**)

clear-web-db-bypass {

status ... ;

cookie ... ;

activation ... ;

duration ... ;

}

The `clear-web-db-bypass` section is derived from `clear-web-db-bypass` section prototype. For detail description of it, see clear-web-db(5).

`replace-response` *code reason mime-type path*;

Send a local file instead of a server response.

code **(type: `uint16`)**

Status code of response.

reason **(type: `str`)**

Reason phrase.

mime-type **(type: `str`)**

Document Content-Type.

path **(type: `str`)**

File to send (absolute, or relative to document-root).

Constraints:

Reply-code must be 4xx or 5xx.

`deny-msg` [`template` *template*] [*msg*];

The error page returned when a request is denied by ACL. If not set, a generic "denied by security policy" page is returned, except when the denying ACL contains item CLEAR-WEB-DB-MATCH. In this case, the returned page will display the categories of the request URI that match the CLEAR-WEB-DB-MATCH item. See also http-proxy(8) for the instructions how to create custom error pages.

`template` *template* (type: `str`, optional, default: `""`)

error page template file name, without the .html.LANGUAGE-CHARSET suffix, relative to the DOCUMENT-ROOT directory

msg **(type: `str`, optional, default: `""`)**

a message inserted into the error page template

`language` *lang*;

Language of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

lang **(type: `language`)**

`client-altq` *altq* [`paltq` *paltq*];

ALTQ queues for data sent to client.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

`paltq` *paltq* (type: `name` of `pf-queue`, see pf-queue(5), optional, default: NULL)

priority queue name (if set, used for TCP ACK without data)

[End of section `icap-server.request-acl` description.]

`doc-acl` *name* {

* from ... ;

```
* server ... ;
* user ... ;
* time ... ;
  time-period-set { ... }
* request-acl ... ;
  deny ... ;
  accept ... ;
  rule ... ;
* content-type ... ;
* mime-type ... ;
  virus-status ... ;
  force-doctype-ident ... ;
  html-filter ... ;
  replace-response ... ;
  deny-msg ... ;
  client-altq ... ;
}
```

The `doc-acl` section is derived from `acl-3` section prototype. For detail description of it, see acl(5).

Changes to the `doc-acl` section:

Item `parent-acl` used as `request-acl`.

Item `direction` is not valid.

Item `size` is not valid.

Item `modify-header` is not valid.

Item `replace` is not valid.

Item DENY-MSG requires DENY.

Added items & subsections:

`replace-response` ***code reason mime-type path*;**

Send a local file instead of a server response.

code **(type: `uint16`)**

Status code of response.

reason **(type: `str`)**

Reason phrase.

mime-type **(type: `str`)**

Document Content-Type.

path **(type: `str`)**

File to send (absolute, or relative to document-root).

Constraints:

Reply-code must be 4xx or 5xx.

deny-msg [template *template*] [*msg*];

The error page returned when a request is denied by ACL. If not set, a generic "denied by security policy" page is returned, except when the denying ACL contains item CLEAR-WEB-DB-MATCH. In this case, the returned page will display the categories of the request URI that match the CLEAR-WEB-DB-MATCH item. See also http-proxy(8) for the instructions how to create custom error pages.

template *template* **(type: str, optional, default: "")**

error page template file name, without the .html.LANGUAGE-CHARSET suffix, relative to the DOCUMENT-ROOT directory

msg **(type: str, optional, default: "")**

a message inserted into the error page template

client-altq *altq* **[paltq** *paltq*];

ALTQ queues for data sent to client.

altq **(type: name of pf-queue, see pf-queue(5))**

queue name

paltq *paltq* **(type: name of pf-queue, see pf-queue(5), optional, default: NULL)**

priority queue name (if set, used for TCP ACK without data)

[End of section `icap-server.doc-acl` description.]

[End of section `icap-server` description.]

SEE ALSO

configuration(7), acl(5), antivirus(5), application(5), auth(5), clear-web-db(5), common(5), ipc(5), ldap(5), listen-on(5), log(5), monitoring(5), netio(5), pf-queue(5), source-address(5), ssl(5), time(5), http-proxy(8)

NAME

icap-server.cfg — format of icap-server program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **icap-server.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **icap-server.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

language (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see radius(5))

ldap-tls-reqcert-mode (see ldap(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

user-match-mode (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

antivirus-protocol (see antivirus(5))

virus-status (see antivirus(5))

database-source (see antivirus(5))

source-address-mode (see source-address(5))

accept-deny (see mod-html-filter(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

ssl-ver (see ssl(5))

extension-op (see ssl(5))

veri-fail-action (see ssl(5))

auth-cert-type (see ssl(5))

distrusted-cert-type (see ssl(5))

clear-web-db-category (see clear-web-db(5))

clear-web-db-match-mode (see clear-web-db(5))

ITEMS AND SECTIONS

Program **icap-server** recognizes following items and sections:

admin ... ;
* antivirus *name* { ... }
clear-web-db { ... }
* fake-cert *name* { ... }
* html-filter *name* { ... }
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
* ssl-params *name* { ... }
sysctl { ... }
use-resolver ... ;
* icap-server *name* { ... }
ipv6-mode ... ;

Description:

admin *system* [*contact*];

Firewall administrator and contact e-mail addresses.

system **(type: str)**

The technical administrator(s) of the system; an address or set of comma separated adresses of persons responsible for system maintenance.

contact **(type: str, optional, default: <NULL>)**

The policy administator; an address of person responsible for system configuration. If not defined, the technical administration is used instead.

Constraints:

Administrator contact must comply with RFC.

antivirus *name* **{**

connection ... ;

sock-opt { ... }

timeout ... ;

comm-dir ... ;

altq ... ;

max-checked-size ... ;

icap-pass-200-with-pure-body ... ;

persistent-stream ... ;

clamav-agent { ... }

}

The `antivirus` section is derived from `antivirus` section prototype. For detail description of it, see antivirus(5).

`clear-web-db` {

internal-servers ... ;

db ... ;

lock ... ;

local-db { ... }

}

The `clear-web-db` section is derived from `clear-web-db` section prototype. For detail description of it, see clear-web-db(5).

`fake-cert` *name* {

key ... ;

auth-ca ... ;

fail-ca ... ;

* extension ... ;

purge ... ;

}

The `fake-cert` section is derived from `fake-cert` section prototype. For detail description of it, see ssl(5).

`html-filter` *name* {

* script-tag-language ... ;

replace-head-script-tags ... ;

replace-body-script-tags ... ;

* style-tag-type ... ;

replace-style-tags ... ;

* iframe-tag-src ... ;

replace-iframe-tags ... ;

* intrinsic-language ... ;
* intrinsic-hack ... ;
  replace-intrinsic ... ;
* macro-language ... ;
* macro-hack ... ;
  replace-macros ... ;
* uri ... ;
  replace-uri ... ;
* embed-tag-type ... ;
* embed-src-hack ... ;
* embed-plugin-hack ... ;
  replace-head-embed-tags ... ;
  replace-body-embed-tags ... ;
* applet ... ;
  replace-applets ... ;
* object ... ;
* object-classid-hack ... ;
* object-data-hack ... ;
  replace-head-object-tags ... ;
  replace-body-object-tags ... ;
* param-tags ... ;
  replace-param ... ;
  script-end-hack ... ;

}

The `html-filter` section is derived from `html-filter` section prototype. For detail description of it, see mod-html-filter(5).

`interface` *name* {

dev ... ;
ipv4 ... ;
ipv6 ... ;
mac ... ;
aggregate ... ;
pike ... ;
vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`ldap-client-auth` *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

`oob-auth` *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

pf-queue *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The pf-queue section is derived from pf-queue section prototype. For detail description of it, see pf-queue(5).

radius-client *name* {

nas ... ;

groups ... ;

* server ... ;

}

The radius-client section is derived from radius-client section prototype. For detail description of it, see radius(5).

resolver *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The resolver section is derived from resolver section prototype. For detail description of it, see resolver(5).

shared-dir *name* {

path ... ;

}

The shared-dir section is derived from shared-dir section prototype. For detail description of it, see common(5).

shared-file *name* {

path ... ;

format ... ;

}

The shared-file section is derived from shared-file section prototype. For detail description of it, see common(5).

ssl-params *name* {

versions ... ;

ciphers ... ;

tcp-eof ... ;

id ... ;

* auth-cert ... ;

distrusted-certs ... ;

dont-check-crl ... ;

* crl ... ;

verify-peer ... ;

cache-timeout ... ;

use-ticket ... ;

enable-renegotiation ... ;

fake-cert ... ;

prefer server ciphers ... ;

enable-ecdh ... ;

}

The ssl-params section is derived from ssl-params section prototype. For detail description of it, see ssl(5).

sysctl {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

use-resolver *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

icap-server *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

doctype-identification { ... }

client-conn { ... }

document-root ... ;

hdr-line-len ... ;

preview ... ;

blacklist-db ... ;

max-bypass-sessions ... ;

ssl-session-cache { ... }

ldap-cache { ... }

* session-acl *name* { ... }

* service-acl *name* { ... }

* request-acl *name* { ... }

* doc-acl *name* { ... }

}

The `icap-server` section is derived from `icap-server` section prototype. For detail description of it, see icap-server(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), icap-server(8), acl(5), antivirus(5), auth(5), clear-web-db(5), common(5), icap-server(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), mod-html-filter(5), pf-queue(5), radius(5), resolver(5), source-address(5), ssl(5), sysctl(5), time(5), host-matching(7)

NAME

imap4-proxy — format of imap4-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **imap4-proxy** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **imap4-proxy** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

nls (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

auth-method (see **auth**(5))

virus-status (see antivirus(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

listen-on-sock (see listen-on(5))

mime-header-check-type (see mod-mail-doc(5))

imap4-cmd (name-usage obligatory)

IMAP4 commands

UNKNOWN

command unknown to the proxy

CAPABILITY

NOOP

LOGOUT

STARTTLS

AUTHENTICATE

LOGIN

SELECT

EXAMINE

CREATE

DELETE

RENAME

SUBSCRIBE

UNSUBSCRIBE

LIST

LSUB

STATUS

APPEND

CHECK

CLOSE

EXPUNGE

SEARCH

FETCH

STORE

COPY

UID-COPY

UID-FETCH

UID-STORE

UID-SEARCH

SETACL

DELETEACL

GETACL

LISTRIGHTS

MYRIGHTS

SETQUOTA

GETQUOTA

GETQUOTAROOT

IDLE

UID-EXPUNGE

ID

UNSELECT

imap4-capa (name-usage obligatory)

IMAP4 capabilities

UNKNOWN

capability unknown to the proxy

IMAP4rev1

STARTTLS

LOGINDISABLED

AUTH-PLAIN

AUTH-GSSAPI

ACL

QUOTA

LITERALPLUS

IDLE

UIDPLUS

ID

MULTIAPPEND

BINARY

UNSELECT

ITEMS AND SECTIONS

Configuration of **imap4-proxy** library component consists of following prototypes:
* imap4-proxy *name* { ... }

Description:

imap4-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

ssl-session-cache { ... }

mail-pool ... ;

* session-acl *name* { ... }

* command-acl *name* { ... }

* mail-acl *name* { ... }

* doc-acl *name* { ... }

}

IMAP4 proxy configuration.

The `imap4-proxy` section is derived from `proxy` section prototype. For detail description of it, see application(5).
Changes to the `imap4-proxy` section:

Section `udpserver` is not valid.

MAIL-POOL must be specified.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one COMMAND-ACL must be specified.

Section `monitoring` (see monitoring(5))

Item `aproxy-user` is not valid.

Item `data` used as `uri`.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 143.

Element `proto` is optional, default: tcp.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 143.

Element `proto` is optional, default: tcp.

Added items & subsections:

`client-conn` {

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection to client options.

The `client-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `client-conn` section:

Item `conn-timeout` is not valid.

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`server-conn` {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection to server options.

The `server-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `server-conn` section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`ssl-session-cache` {

capacity ... ;

dir ... ;

lock ... ;

}

The `ssl-session-cache` section is derived from `ssl-session-cache` section prototype. For detail description of it, see ssl(5).

`mail-pool` *name*;

Mail pool directory.

name **(type: `str`)**

`session-acl` *name* {

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

auth ... ;

idle-timeout ... ;

source-address ... ;

plug-to ... ;

client-ssl ... ;

* client-cert-match ... ;

language ... ;

}

The first level ACL decides how to handle incoming connections.

The `session-acl` section is derived from `acl-1` section prototype. For detail description of it, see acl(5).
Changes to the `session-acl` section:

Item `user` is not valid.

Item `idle-timeout-peer` is not valid.

SSL/TLS required on in order to match client certificates.

IDLE-TIMEOUT has no use without SSL/TLS.

Item `auth` (see auth(5))

Element `mode` is optional, default: allowed.

Only out-of-band authentication is supported in this proxy.

Added items & subsections:

`client-ssl` *params*;

Use SSL/TLS on the connection from a client.

params **(type: `name` of `ssl-params`, see ssl(5))**

`client-cert-match` [`subject` *subject*] [`issuer` *issuer*];

Requirements for client certificate.

`subject` *subject* (type: `str-set`, optional, default: *)

acceptable certificate subjects

`issuer` *issuer* (type: `str-set`, optional, default: *)

acceptable certificate issuers

`language` [*code*];

Language and charset of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

code **(type: `nls`, optional, default: EN)**

[End of section `imap4-proxy.session-acl` description.]

`command-acl` *name* {

* from ... ;
* server ... ;
* user ... ;
* time ... ;
  time-period-set { ... }
* session-acl ... ;
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
  source-address ... ;
  plug-to ... ;
* client-cert-match ... ;
  server-ssl ... ;
* server-cert-match ... ;
  language ... ;
  max-bytes-in ... ;
  max-bytes-out ... ;
  max-mail-in ... ;

max-mail-out ... ;

max-time ... ;

idle-timeout ... ;

commands ... ;

capabilities ... ;

upload { ... }

download { ... }

client-altq ... ;

server-altq ... ;

}

The second level ACL sets parameters of the connection to the server and decides about handling individual commands.

The `command-acl` section is derived from `acl-2` section prototype.
For detail description of it, see acl(5).
Changes to the `command-acl` section:

Item `parent-acl` used as `session-acl`.

SSL/TLS required on in order to match server certificates.

Added items & subsections:
`source-address` [`client`] [`addr4` *addr4*] [`addr6` *addr6*] `cluster` [*cluster*];
`source-address` [`client`] [`addr4` *addr4*] [`addr6` *addr6*] [`physical`];
`source-address` [`client`] [`addr4` *addr4*] [`addr6` *addr6*] `no-fallback`;

Source address for outgoing connections to servers.

If omitted, the proper address of the proxy will be used, i.e. in the case of a cluster, the cluster address will be used.

If not specified by the SOURCE-PORT item, a generic port will be used.

The elements entered within this item will be used by the proxy until the first of them is applicable:

- The CLIENT keyword means the original client IP address is used. This mode will be succesful in all cases except mismatch of IP address families.

- The ADDR4/ADDR6 keyword-value pairs mean that the specified address is used for a connection of corresponding address family.

- The CLUSTER keyword means that one of cluster addresses will be used. By default, the main address of the bridge is used, however, any preferred alias address can be listed in the cluster list.- The PHYSICAL option means that the address of the physical interface is used instead of the cluster one.

- The DEFAULT option means the default behavior - i.e. using of the physical address.

- The NO-FALLBACK option means that if no other way of setting the address is acceptable, the session is rejected. Without this option, the system tries to find a suitable source IP address automatically.

`client` (type: `key`, optional)

addr4 *addr4* (type: **host**, optional, default: [0.0.0.0])

addr6 *addr6* (type: **host**, optional, default: [::])
<branching element> (type: **source-address-mode**, optional, default: physical)
cluster (type: **host-list**, optional, default: {})

Constraints:

Address family must respect the element's address family..

plug-to *addr*;

Final destination server.

addr (type: **sock**)

Address/port of final destination server.

If port is zero, then original port is used.

client-cert-match [**subject** *subject*] [**issuer** *issuer*];

Select an ACL according to a client certificate.

subject *subject* (type: **str-set**, optional, default: *)

acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)

acceptable certificate issuers

server-ssl *params*;

Use SSL/TLS on the connection to a server.

params (type: **name** of **ssl-params**, see ssl(5))

server-cert-match [**subject** *subject*] [**issuer** *issuer*];

Requirements for server certificate.

subject *subject* (type: **str-set**, optional, default: *)

acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)

acceptable certificate issuers

language *code*;

Language and charset of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

code (type: **nls**)

max-bytes-in *bytes*;

Maximum number of bytes from server to client in a session.

bytes (type: **uint64**)

max-bytes-out *bytes*;

Maximum number of bytes from client to server in a session.

bytes (type: **uint64**)

max-mail-in *bytes*;

Maximum size of any single mail transferred from client to server.

bytes (type: **uint64**)

max-mail-out *bytes*;

Maximum size of any single mail transferred from server to client.

bytes (type: **uint64**)

max-time *seconds*;

Maximum time of session

seconds (type: **uint32**)

idle-timeout [*seconds*];

If no data transmitted for this session in the period of idle-timeout seconds, connection is closed.

If omitted, value of proxy session-acl.idle-timeout is used.

seconds (type: **uint32**, optional, default: 0)

commands [*cmd*];

Set of allowed IMAP4 commands.

cmd (type: **imap4-cmd-set**, optional, default: *)

capabilities [*cap*];

Set of allowed IMAP4 capabilities (sent in response to command.

cap (type: **imap4-capa-set**, optional, default: *)

upload {

mail-filter ... ;
use-antispam ... ;
use-antivirus ... ;
no-mail-scanning ... ;
}

Settings for uploading mail from client to server.

Constraints:

MAIL-FILTER, USE-ANTISPAM, and USE-ANTIVIRUS cannot be used together with NO-MAIL-SCANNING.

Items & subsections:

mail-filter *name*;

Filter for mails

name (type: **name** of **mail-filter**, see mod-mail-doc(5))

use-antispam disable;

use-antispam enable *channel* [*limit*];

Antispam usage.

This section defines type of antispam daemon used and mode of antispam checking operation.

<**branching element**> (type: **enabling**)

channel **(type: `name` of `antispam`, see mod-antispam(5))**

Name of antispam global section used.
Referred section defines the way how to communicate with the antispam daemon (see above).

limit **(type: `uint64`, optional, default: 0)**

Size limit (in bytes) for antispam check.
Antispam checking used to be very exhausting operation, and typical spam mails used to be not very large (both for passing by size limit filters and for being able to send a lot of copies). That's why it can be desired to avoid checking of very large mails.
Setting of this limit says antispam module not to check mails larger than given limit and declare their spam score to zero.
Setting this limit to zero disables this feature and enables using of antispam to all mails. Be prepared for high machine load and noticeable delay in delivery if used so.

`use-antivirus disable;`

`use-antivirus enable` *channel*;

Antivirus usage mode.
If omitted, or disabled, no antivirus is enabled. In this case, neither any ANTIVIRUS global section can be present nor any MAIL-ACL and DOC-ACL can have VIRUS item specified.

<branching element> (type: `enabling`)

channel **(type: `name-list` of `antivirus`, see antivirus(5))**

`no-mail-scanning;`

Pass mail to the client without checking.

[End of section `imap4-proxy.command-acl.upload` description.]

`download {`

```
mail-filter ... ;
use-antispam ... ;
use-antivirus ... ;
no-mail-scanning ... ;
}
```

Settings for downloading mail from server to client.

Constraints:

MAIL-FILTER, USE-ANTISPAM, and USE-ANTIVIRUS cannot be used together with NO-MAIL-SCANNING.

Items & subsections:

`mail-filter` *name*;

Filter for mails

name (type: **name** of **mail-filter**, see mod-mail-doc(5))

use-antispam disable;

use-antispam enable *channel* [*limit*];

Antispam usage.

This section defines type of antispam daemon used and mode of antispam checking operation.

<**branching element**> (type: **enabling**)

channel (type: **name** of **antispam**, see mod-antispam(5))

Name of antispam global section used.

Referred section defines the way how to communicate with the antispam daemon (see above).

limit (type: **uint64**, optional, default: 0)

Size limit (in bytes) for antispam check.

Antispam checking used to be very exhausting operation, and typical spam mails used to be not very large (both for passing by size limit filters and for being able to send a lot of copies). That's why it can be desired to avoid checking of very large mails.

Setting of this limit says antispam module not to check mails larger than given limit and declare their spam score to zero.

Setting this limit to zero disables this feature and enables using of antispam to all mails. Be prepared for high machine load and noticeable delay in delivery if used so.

use-antivirus disable;

use-antivirus enable *channel*;

Antivirus usage mode.

If omitted, or disabled, no antivirus is enabled. In this case, neither any ANTIVIRUS global section can be present nor any MAIL-ACL and DOC-ACL can have VIRUS item specified.

<**branching element**> (type: **enabling**)

channel (type: **name-list** of **antivirus**, see antivirus(5))

no-mail-scanning;

Pass mail to the client without checking.

[End of section `imap4-proxy.command-acl.download` description.]

client-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to client.

altq **(type: name of pf-queue, see pf-queue(5))**
queue name

paltq *paltq* **(type: name of pf-queue, see pf-queue(5), optional, default: NULL)**
priority queue name (if set, used for TCP ACK without data)

server-altq *altq* **[paltq** *paltq***];**
ALTQ queues for data sent to server.

altq **(type: name of pf-queue, see pf-queue(5))**
queue name

paltq *paltq* **(type: name of pf-queue, see pf-queue(5), optional, default: NULL)**
priority queue name (if set, used for TCP ACK without data)

[End of section `imap4-proxy.command-acl` description.]

mail-acl *name* **{**

* from ... ;
* time ... ;
  time-period-set { ... }
* command-acl ... ;
  deny ... ;
  accept ... ;
  rule ... ;
  direction ... ;
* content-type ... ;
  virus-status ... ;
* modify-header ... ;
  replace ... ;
* spam-score ... ;
* header ... ;
  prefix-subject ... ;

}

The first ACL on the third level decides how to handle the whole mail.

The mail-acl section is derived from mail-acl section prototype.
For detail description of it, see mod-mail-doc(5).
Changes to the mail-acl section:

Item `parent-acl` used as `command-acl`.
Item `size` is not valid.
Item `sender` is not valid.
Item `recipient` is not valid.
Item `recipients` is not valid.
Item `from-quarantine` is not valid.

doc-acl *name* {

* from ... ;
* time ... ;
  time-period-set { ... }
* command-acl ... ;
  deny ... ;
  accept ... ;
  rule ... ;
  direction ... ;
* size ... ;
* content-type ... ;
* mime-type ... ;
  virus-status ... ;
* modify-header ... ;
  force-doctype-ident ... ;
  replace ... ;
  html-filter ... ;
* spam-score ... ;
* header ... ;
* filename ... ;
  add-virus-names ... ;

}

The `doc-acl` section is derived from `mail-doc-acl` section prototype. For detail description of it, see mod-mail-doc(5).
Changes to the `doc-acl` section:

Item `parent-acl` used as `command-acl`.
Item `sender` is not valid.
Item `recipient` is not valid.
Item `from-quarantine` is not valid.

[End of section `imap4-proxy` description.]

SEE ALSO

configuration(7), acl(5), antivirus(5), application(5), auth(5), common(5), ipc(5), listen-on(5), log(5), mod-antispam(5), mod-mail-doc(5), monitoring(5), netio(5), pf-queue(5), source-address(5), ssl(5), time(5)

NAME

imap4-proxy.cfg — format of imap4-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **imap4-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **imap4-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

nls (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see radius(5))

ldap-tls-reqcert-mode (see ldap(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

antivirus-protocol (see antivirus(5))

virus-status (see antivirus(5))

database-source (see antivirus(5))

source-address-mode (see source-address(5))

accept-deny (see mod-html-filter(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

ssl-ver (see ssl(5))

extension-op (see ssl(5))

veri-fail-action (see ssl(5))

auth-cert-type (see ssl(5))

distrusted-cert-type (see ssl(5))

mail-reaction (see mod-mail-doc(5))

mail-fallback (see mod-mail-doc(5))

mime-header-check-type (see mod-mail-doc(5))

imap4-cmd (see imap4-proxy(5))

imap4-capa (see imap4-proxy(5))

ITEMS AND SECTIONS

Program **imap4-proxy** recognizes following items and sections:

* antispam *name* { ... }
* antivirus *name* { ... }
* fake-cert *name* { ... }
* html-filter *name* { ... }
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* mail-filter *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
* ssl-params *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* imap4-proxy *name* { ... }
  ipv6-mode ... ;

Description:

antispam *name* **{**

connection ... ;

sock-opt { ... }

altq ... ;

}

The antispam section is derived from antispam section prototype. For detail description of it, see mod-antispam(5).

antivirus *name* **{**

connection ... ;

sock-opt { ... }

timeout ... ;

comm-dir ... ;

altq ... ;

max-checked-size ... ;

icap-pass-200-with-pure-body ... ;

persistent-stream ... ;

clamav-agent { ... }

}

The `antivirus` section is derived from `antivirus` section prototype. For detail description of it, see antivirus(5).

`fake-cert` *name* {

key ... ;

auth-ca ... ;

fail-ca ... ;

* extension ... ;

purge ... ;

}

The `fake-cert` section is derived from `fake-cert` section prototype. For detail description of it, see ssl(5).

`html-filter` *name* {

* script-tag-language ... ;

replace-head-script-tags ... ;

replace-body-script-tags ... ;

* style-tag-type ... ;

replace-style-tags ... ;

* iframe-tag-src ... ;

replace-iframe-tags ... ;

* intrinsic-language ... ;

* intrinsic-hack ... ;

replace-intrinsic ... ;

* macro-language ... ;

* macro-hack ... ;

replace-macros ... ;

* uri ... ;

replace-uri ... ;

* embed-tag-type ... ;

* embed-src-hack ... ;

* embed-plugin-hack ... ;

replace-head-embed-tags ... ;

replace-body-embed-tags ... ;

* applet ... ;

replace-applets ... ;

* object ... ;

* object-classid-hack ... ;

* object-data-hack ... ;

replace-head-object-tags ... ;

replace-body-object-tags ... ;

* param-tags ... ;

replace-param ... ;

script-end-hack ... ;

}

The `html-filter` section is derived from `html-filter` section prototype. For detail description of it, see mod-html-filter(5).

`interface` *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

ldap-client-auth *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The ldap-client-auth section is derived from ldap-client-auth section prototype. For detail description of it, see ldap(5).

mail-filter *name* {

stamp-limit ... ;

stamp-filter ... ;

* unflagged-8bit ... ;

* bad-end-of-line ... ;

* invalid-header ... ;

* long-header-lines ... ;

* invalid-chars ... ;

* header-8bit-chars ... ;

* bad-boundary-chars ... ;

* bad-boundary-length ... ;

* long-body-lines ... ;

* long-encoded-lines ... ;

enc-line-len ... ;

* bad-mime-struct ... ;

* invalid-encoding ... ;

treat-rfc822-as-text ... ;

}

The mail-filter section is derived from mail-filter section prototype. For detail description of it, see mod-mail-doc(5).

oob-auth *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The oob-auth section is derived from oob-auth section prototype. For detail description of it, see auth(5).

pf-queue *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The pf-queue section is derived from pf-queue section prototype. For detail description of it, see pf-queue(5).

radius-client *name* {

nas ... ;

groups ... ;

* server ... ;

}

The radius-client section is derived from radius-client section prototype. For detail description of it, see radius(5).

resolver *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`shared-dir` *name* {

path ... ;

}

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

`shared-file` *name* {

path ... ;

format ... ;

}

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`ssl-params` *name* {

versions ... ;

ciphers ... ;

tcp-eof ... ;

id ... ;

* auth-cert ... ;

distrusted-certs ... ;

dont-check-crl ... ;

* crl ... ;

verify-peer ... ;

cache-timeout ... ;

use-ticket ... ;

enable-renegotiation ... ;

fake-cert ... ;

prefer server ciphers ... ;

enable-ecdh ... ;

}

The `ssl-params` section is derived from `ssl-params` section prototype. For detail description of it, see ssl(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`imap4-proxy` *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

ssl-session-cache { ... }

mail-pool ... ;

* session-acl *name* { ... }

* command-acl *name* { ... }

* mail-acl *name* { ... }

* doc-acl *name* { ... }

}

The `imap4-proxy` section is derived from `imap4-proxy` section prototype. For detail description of it, see imap4-proxy(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), imap4-proxy(8), acl(5), antivirus(5), auth(5), common(5), imap4-proxy(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), mod-antispam(5), mod-html-filter(5), mod-mail-doc(5), pf-queue(5), radius(5), resolver(5), source-address(5), ssl(5), sysctl(5), time(5), host-matching(7)

NAME

interface — format of interface component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **interface** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **interface** configuration directives:

yes-no (see **common(5)**)

lagg-protocol (name-usage obligatory)

Line Aggregation Protocols.

none

failover

fec

lacp

loadbalance

roundrobin

ITEMS AND SECTIONS

Configuration of **interface** library component consists of following prototypes:

ipv4 ... ;
ipv6 ... ;
ipv6-rtadv { ... }
tunnel ... ;
* interface *name* { ... }

Description:

ipv4 *addr* [*dest*];

IPv4 address definition.

addr (type: **addr**)

Interface address and mask.

dest (type: **host**, optional, default: [0.0.0.0])

Point-to-point destination address.

Methods:

net (type: **net**)

Method to get network address.

host (type: **host**)

Method to get address w/o mask.

ipv6 *addr*;

IPv6 address definition.

addr (type: **addr**)

Interface address with prefix.

Methods:

net (type: **net**)

Method to get network address.

host (type: **host**)

Method to get address w/o prefix.

ipv6-rtadv {

enable ... ;

managed-address ... ;

other-stateful ... ;

* raw ... ;

}

Settings of IPv6 router advertisements.

Items & subsections:

enable [*val*];

Enables or disables IPv6 router advertisements on this interface.

val (type: **yes-no**, optional, default: no)

managed-address [*val*];

Sets Managed address configuration flag bit.

val (type: **yes-no**, optional, default: no)

other-stateful [*val*];

Sets Other stateful configuration flag bit.

val **(type: yes-no, optional, default: no)**

raw *field*;

Raw configuration field.

It must be written in the format specified in rtadvd.conf(5). The FIELD value of this item is written (separated properly by colons) to the end of the rtadvd.conf 'default' entry (if used in the global RTADVD section), or just before the terminal 'tc=default' field of a particular interface entry (if used in an INTERFACE section).

field **(type: str)**

[End of section `ipv6-rtadv` description.]

tunnel *addr dest*;

Pair of tunnel addresses.

addr **(type: addr)**

Interface address with prefix.

dest **(type: host)**

Tunnel destination address.

Constraints:

Tunnel addresses must have the same family..

interface *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

Interface description.

There are two main reasons for defining of interfaces:

- All interfaces except VIRTUAL ones will be added into operating system startup and a formal Kernun component will be added to ease its management.

- All interfaces (their names) can be used for proxy listen-on socket definition.

Constraints:

Device name must be specified.

Tunnel addresses needed and allowed only for GIF and GRE.

Destination address needed and allowed only for IPv4 in GIF, GRE and TUN..

Tunnel IPv6 addresses not allowed for GRE.

AGGREGATE is obligatory item of LAGG interfaces.

VLAN is obligatory item of VLAN interfaces.

PIKE is obligatory item of PIKE interfaces.

Items & subsections:

`dev` *name* **[`virtual`] [`media`** *media***] [`mediaopt`** *mediaopt***];**

Device description.

name **(type: `str`)**

Device name.

`virtual` **(type: `key`, optional)**

Virtual device, do not include to rc.conf.

`media` *media* **(type: `str`, optional, default: <NULL>)**

Device media type.

`mediaopt` *mediaopt* **(type: `str`, optional, default: <NULL>)**

Device media options.

Constraints:

Media options can be set only if media is set, too.

`ipv4` *addr* **[***dest***];**

Interface base IPv4 address.

addr **(type: `addr`)**

Interface address and mask.

dest **(type: `host`, optional, default: [0.0.0.0])**

Point-to-point destination address.

Methods:

`net` **(type: `net`)**

Method to get network address.

`host` **(type: `host`)**

Method to get address w/o mask.

`ipv6` *addr*;

Interface base IPv6 address.

addr (type: **addr**)

Interface address with prefix.

Methods:

net (type: **net**)

Method to get network address.

host (type: **host**)

Method to get address w/o prefix.

mac *addr*;

Hardware address.

addr (type: **str**)

aggregate [**proto** *proto*] *iface*;

Aggregated interface parameters definition.

proto *proto* (type: **lagg-protocol**, optional, default: failover)

iface (type: **name-list** of **interface**, see above)

pike *iface* [**nomadic**];

PIKE interface parameters definition.

iface (type: **name** of **interface**, see above)

Real interface.

nomadic (type: **key**, optional)

Flag to hide address in backup state.

vlan *id parent*;

VLAN interface parameters definition.

id (type: **uint16**)

VLAN ID

parent (type: **name** of **interface**, see above)

Parent interface

tunnel *addr dest*;

Pair of tunnel addresses.

addr (type: **addr**)

Interface address with prefix.

dest (type: **host**)

Tunnel destination address.

Constraints:

Tunnel addresses must have the same family..

dhcp-client;

DHCP configuration mode definition.

If used, the interface will be configured via DHCP.

If used together with the IPv4 item, the address MUST be assigned statically and interface behaves as normal interface except the dhclient daemon running.

If the address is assigned by the DHCP server randomly, the IPv4 item must not be used. In this case, the interface cannot be referenced by the name in non-transparent listen-on case.

ipv6-rtadv {

enable ... ;

managed-address ... ;

other-stateful ... ;

* raw ... ;

}

The ipv6-rtadv section is derived from ipv6-rtadv section prototype. For detail description of it, see above.

alias *name* **{**

ipv4 ... ;

ipv6 ... ;

}

Interface aliases definition.

Items & subsections:

ipv4 *addr* **[***dest***];**

IPv4 address definition.

addr **(type: addr)**

Interface address and mask.

dest **(type: host, optional, default: [0.0.0.0])**

Point-to-point destination address.

Methods:

net (type: net)

Method to get network address.

host (type: host)

Method to get address w/o mask.

ipv6 *addr***;**

IPv6 address definition.

addr **(type: addr)**

Interface address with prefix.

Methods:

net (type: net)

Method to get network address.

host (type: host)

Method to get address w/o prefix.

[End of section `interface.alias` description.]

`tag` ***value*;**

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

[End of section `interface` description.]

SEE ALSO

configuration(7), common(5), rtadvd.conf(5)

NAME

ipc — format of ipc component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ipc** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **ipc** configuration directives:

`lock-type` (name-usage obligatory)

Type of an alternative lock.

`none`

No locking done.

`semaphore`

Lock implemented by a SysV semaphore.

`lock2`

Two levels of locks with a single lock file.

`multilock2`

Two levels of locks with multiple lock files.

ITEMS AND SECTIONS

Configuration of **ipc** library component consists of following prototypes:

lock ... ;
alt-lock ... ;

Description:

`lock` [*path*];

Lock file used as a lock.

path **(type: `str`, optional, default: <NULL>)**

If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

`alt-lock none`;

`alt-lock semaphore`;

`alt-lock lock2` [*path*];

`alt-lock` [**`multilock2`**] [*path*];

An alternative implemetation of locks.

<branching element> (type: `lock-type`, optional, default: multilock2)

path **(type: `str`, optional, default: <NULL>)**

If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

SEE ALSO

configuration(7)

NAME

ipsec — format of ipsec component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ipsec** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **ipsec** configuration directives:

ipsec-encryption1 (name-usage obligatory)

Available encryption algorithms for ISAKMP phase 1.

aes

des3

cast128

blowfish

ipsec-encryption2 (name-usage obligatory)

Available encryption algorithms for ISAKMP phase 2.

aes

des3

cast128

blowfish

rc5

rc4

idea

twofish

ipsec-hash1 (name-usage obligatory)

Available hash algorithms for ISAKMP phase 1.

md5

sha1

sha256

sha384

sha512

ipsec-auth2 (name-usage obligatory)

Available authentication algorithms for ISAKMP phase 2.

hmac_md5

hmac_sha1

hmac_sha256

hmac_sha384

hmac_sha512

ipsec-dh-group (name-usage optional)

Diffie-Hellman group for ISAKMP.

modp768 (1)

modp1024 (2)

modp1536 (5)

modp2048 (14)

modp3072 (15)

modp4096 (16)

modp6144 (17)

modp8192 (18)

ipsec-tunnel-sa-mode (name-usage obligatory)

Mode of creating IPsec security associations in the tunnel mode.

network

There will be a single SA for each pair of networks.

host

There will be a separate SA for each pair of communicating hosts.

ipsec-auth-method (name-usage obligatory)

Method of ISAKMP phase 1 authentication

psk

Pre-shared key.

x509

X.509 certificate.

ipsec-protocol (name-usage optional)

Protocols handled by IPsec in tunnel mode.

any (0)

icmp (1)

ipencap (4)

gif (4)

tcp (6)

udp (17)

gre (47)

ipsec-remote-mode (name-usage obligatory)

Remote host definition mode.

address

Remote address is defined directly.

tunnel

Remote address is taken from INTERFACE.TUNNEL.

ipsec-rekey-mode (name-usage obligatory)

Automatic renegotiation of expired phase1 modes.

off

No automatic rekeying.

on

Rekeying bound to DPD monitoring.

force

Rekeying unconditional.

ITEMS AND SECTIONS

Configuration of **ipsec** library component consists of following prototypes:
ipsec-global { ... }
* ipsec *name* { ... }
* ipsec-remote *name* { ... }

Description:

ipsec-global {

phase ... ;

* tag ... ;

}

Global parameters of IPsec.

Items & subsections:

phase [*number*];

ISAKMP daemon startup phase.

***number* (type: `uint8`, optional, default: 40)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

***value* (type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

[End of section `ipsec-global` description.]

ipsec *name* {

phase ... ;

* tag ... ;

transport-mode ... ;

tunnel-mode { ... }

phase2 { ... }

}

Definition of a single IPsec tunnel.

Constraints:

Either TRANSPORT-MODE or TUNNEL-MODE must be specified.

TRANSPORT-MODE and TUNNEL-MODE are mutually exclusive.

Section PHASE2 required.

Items & subsections:

phase [*number*];

Application Startup Phase.

***number* (type: `uint8`, optional, default: 40)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

`transport-mode` *interface*;

Use IPsec in transport mode to secure a GIF or GRE tunnel.

interface **(type: `name of interface`, see interface(5))**

IPsec will be used for traffic on this tunnel interface.

`tunnel-mode` {

tunnel ... ;

local ... ;

remote ... ;

protocol ... ;

sa-mode ... ;

}

Use IPsec in tunnel mode.

Constraints:

Tunnel addresses must be specified.

LOCAL networks must be specified.

REMOTE networks must be specified.

PROTOCOL must be specified.

Items & subsections:

`tunnel` *addr dest*;

Pair of tunnel addresses.

addr **(type: `addr`)**

Interface address with prefix.

dest **(type: `host`)**

Tunnel destination address.

Constraints:

Tunnel addresses must have the same family..

Remote IP address must be specified..

`local` *nets*;

Addresses of local networks that communicate via this tunnel.

nets **(type: `net-list`)**

Constraints:

At least one LOCAL network must be specified.

remote *nets*;

Addresses of remote networks that communicate via this tunnel.

nets **(type: net-list)**

Constraints:

At least one REMOTE network must be specified.

protocol *proto*;

List of protocols handled by IPsec in this tunnel.

proto **(type: ipsec-protocol-list)**

Constraints:

At least one PROTOCOL must be specified.

sa-mode [*mode*];

Mode of creating IPsec security associations in the tunnel mode.

mode **(type: ipsec-tunnel-sa-mode, optional, default: network)**

[End of section `ipsec.tunnel-mode` description.]

phase2 {

lifetime ... ;

* encryption ... ;

auth-alg ... ;

pfs-group ... ;

}

ISAKMP phase 2 parameters.

Items & subsections:

lifetime [*sec*];

Lifetime of a SA (in seconds).

sec **(type: uint32, optional, default: 43200)**

Constraints:

Lifetime must not be zero.

encryption *alg* [*bits*];

The encryption algorithms for the phase 2 proposals. If not set, AES256 will be used.

alg **(type: ipsec-encryption2-list)**

bits **(type: uint16, optional, default: 0)**

Key length in bits of the encryption algorithm. The value, if nonzero, applies to all algorithm in the list. The value 0 means the default length for each selected algorithm.

Constraints:

At least one algorithm required.

auth-alg *alg*;

The authentication algorithms for the phase 2 proposals.If not set, HMAC-SHA1 will be used.

alg (type: **ipsec-auth2-list**)

Constraints:

At least one algorithm required.

pfs-group *group*;

The group of Diffie-Hellman exponentiations. If not set, PFS will not be used.

group (type: **ipsec-dh-group**)

Constraints:

Bad DH group number.

[End of section `ipsec.phase2` description.]

[End of section `ipsec` description.]

ipsec-remote *name* {

peer ... ;

lifetime ... ;

encryption ... ;

hash ... ;

dh-group ... ;

authentication ... ;

dpd ... ;

rekey ... ;

ike-frag ... ;

esp-frag ... ;

}

ISAKMP phase 1 parameters for remote host.

Constraints:

Remote peer must be specified.

Authentication method must be specified.

Items & subsections:

peer address *peer*;

peer tunnel *iface*;

Remote peer address definition.

<**branching element**> (type: **ipsec-remote-mode**)

iface (type: **name of interface**, see interface(5))

Tunnel interface used for ipsec to this host.

peer (type: **host**)

Remote host address.

lifetime [*sec*];

Lifetime proposed in the phase 1 negotiations (in seconds).

sec (type: **uint32**, optional, default: **3600**)

Constraints:

Lifetime must not be zero.

encryption [*alg* [*bits*]];

The encryption algorithm used for the phase 1 negotiations.

alg (type: **ipsec-encryption1**, optional, default: **aes**)

bits (type: **uint16**, optional, default: **0**)

Key length in bits of the encryption algorithm. The value 0 means the default length for the selected algorithm.

hash [*alg*];

The hash algorithm used for the phase 1 negotiations.

alg (type: **ipsec-hash1**, optional, default: **sha1**)

dh-group [*group*];

The group used for the Diffie-Hellman exponentiations.

group (type: **ipsec-dh-group**, optional, default: **modp1024=2**)

Constraints:

Bad DH group number.

authentication psk *psk*;

authentication x509 *ca cert key*;

Method and data for authentication.

<branching element> (type: **ipsec-auth-method**)

psk (type: **str**)

The pre-shared key.

ca (type: **name** of **shared-file**, see common(5))

Root CA certificate.

cert (type: **name** of **shared-file**, see common(5))

A certificate.

key (type: **name** of **shared-file**, see common(5))

A private key.

dpd [*delay* [*retry* [*maxfail*]]];

DPD enabling and parameters setting.

delay (type: **uint16**, optional, default: **0**)

Time between two proofs of liveness.

By default, the DPD monitoring is disabled.

retry (type: **uint16**, optional, default: **5**)

Proof of liveness timeout.

maxfail (type: **uint16**, optional, default: **5**)

Maximum number of proof retry.

rekey [*mode*];

Automatic phase1 renegotiation.

mode (type: **ipsec-rekey-mode**, optional, default: **on**)

ike-frag *mode*;

Receiver-side IKE fragmentation.

mode (type: **ipsec-rekey-mode**)

esp-frag *fraglen*;

Forcing ESP over UDP of fragmented packets instead of fragmented ESP over UDP packets.

fraglen (type: **uint16**)

[End of section `ipsec-remote` description.]

SEE ALSO

configuration(7), ipsec(4), common(5), interface(5), racoon.conf(5), racoon(8), setkey(8)

NAME

kernun.cml — format of Kernun configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the whole Kernun configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **kernun.cml** configuration directives:

yes-no (see **common**(5))

language (see **common**(5))

nls (see **common**(5))

on-off (see **common**(5))

genesis (see **common**(5))

permission (see **common**(5))

direction (see **common**(5))

name-selection (see **common**(5))

destination (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

in-out (see **common**(5))

report-mode (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see common(5))

inline-file-format (see common(5))

yes-no-always (see common(5))

task-frequency (see common(5))

week-day (see time(5))

month (see time(5))

lock-type (see ipc(5))

radius-attr (see radius(5))

ldap-tls-reqcert-mode (see ldap(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

user-match-mode (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

antivirus-protocol (see antivirus(5))

virus-status (see antivirus(5))

database-source (see antivirus(5))

source-address-mode (see source-address(5))

source-port-mode (see source-address(5))

accept-deny (see mod-html-filter(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

product-type (see license(5))

component-group (see license(5))

component-type (see license(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

user-type (see system(5))

route-flag (see system(5))

usb-auto-setup-policy (see system(5))

dbglev (see log(5))

logfail-mode (see log(5))

dns-type (see resolver(5))

dns-opcode (see resolver(5))

dns-response (see resolver(5))

dns-qaction (see resolver(5))

dns-raction (see resolver(5))

dns-fake (see resolver(5))

xfr-mode (see resolver(5))

udp-session-type (see udpserver(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

proc-priority (see application(5))

pf-osi4-proto (see packet-filter(5))

icmp-type (see packet-filter(5))

pf-scheduler (see packet-filter(5))

pf-proc-mode (see packet-filter(5))

ids-agent-log-level (see adaptive-firewall(5))

ids-agent-detection-direction (see adaptive-firewall(5))

ids-agent-protocol (see adaptive-firewall(5))

ids-agent-rule-action (see adaptive-firewall(5))

ids-agent-threshold-type (see adaptive-firewall(5))

ids-agent-threshold-track-by (see adaptive-firewall(5))

ids-agent-rate-filter-track-by (see adaptive-firewall(5))

ids-agent-suppress-direction (see adaptive-firewall(5))

policy-level (see adaptive-firewall(5))

ids-agent-rules-download-type (see update(5))

forward (see nameserver(5))

atr-strategy (see atr(5))

atr-fallback (see atr(5))

pike-control-type (see pike(5))

ntp-rest-flag (see ntp(5))

ovpn-protocols (see openvpn(5))

ovpn-remote-proto (see openvpn(5))

ovpn-comp-lzo-mode (see openvpn(5))

ovpn-cert-types (see openvpn(5))

ovpn-cipher-algs (see openvpn(5))

ovpn-redirect-gateway-flags (see openvpn(5))

ovpn-dhcp-option (see openvpn(5))

ovpn-topology (see openvpn(5))

ovpn-local-scope (see openvpn(5))

tls-mat-variants (see openvpn(5))

ipsec-encryption1 (see ipsec(5))

ipsec-encryption2 (see ipsec(5))

ipsec-hash1 (see ipsec(5))

ipsec-auth2 (see ipsec(5))

ipsec-dh-group (see ipsec(5))

ipsec-tunnel-sa-mode (see ipsec(5))

ipsec-auth-method (see ipsec(5))

ipsec-protocol (see ipsec(5))

ipsec-remote-mode (see ipsec(5))

ipsec-rekey-mode (see ipsec(5))

snmpd-disk-mode (see snmpd(5))

snmpd-source-mode (see snmpd(5))

snmpd-view-type (see snmpd(5))

snmpd-security-level (see snmpd(5))

snmpd-auth-hash (see snmpd(5))

snmpd-encr-alg (see snmpd(5))

ssh-key-type (see ssh(5))

ssh-proto (see ssh(5))

export-import-mode (see router(5))

ospf-authentication (see router(5))

ospf-area-id-mode (see router(5))

ssl-ver (see ssl(5))

extension-op (see ssl(5))

veri-fail-action (see ssl(5))

auth-cert-type (see ssl(5))

distrusted-cert-type (see ssl(5))

data-match-action (see mod-match(5))

dns-name-type (see dns-proxy(5))

pass-remove (see ftp-proxy(5))

data-type (see ftp-proxy(5))

ftp-cmd (see ftp-proxy(5))

clear-web-db-category (see clear-web-db(5))

clear-web-db-match-mode (see clear-web-db(5))

replace-authorization-mode (see http-proxy(5))

proxy-via (see http-proxy(5))

http-protocol (see http-proxy(5))

http-scheme (see http-proxy(5))

cookie-table-clean (see http-proxy(5))

accept-gzip (see http-proxy(5))

content-gzip (see http-proxy(5))

http-redirect (see http-proxy(5))

kerberos-user-match (see http-proxy(5))

ldap-select (see http-proxy(5))

auth-headers (see http-proxy(5))

sni-result (see http-proxy(5))

smtp-error (see mod-mail-doc(5))

mail-reaction (see mod-mail-doc(5))

mail-fallback (see mod-mail-doc(5))

mime-header-check-type (see mod-mail-doc(5))

imap4-cmd (see imap4-proxy(5))

imap4-capa (see imap4-proxy(5))

pop3-cmd (see pop3-proxy(5))

pop3-capa (see pop3-proxy(5))

peer (see sip-proxy(5))

smtp-size-usage (see smtp-proxy(5))

ssl-startup-mode (see smtp-proxy(5))

postfix-security-level (see smtp-proxy(5))

postfix-transport-map-mode (see smtp-proxy(5))

smtp-err-switch (see smtp-proxy(5))

spf-result (see smtp-proxy(5))

spf-modes (see smtp-proxy(5))

redirection-mode (see sqlnet-proxy(5))

session-protocol (see proxy-ng(5))

json-type (see proxy-ng(5))

http-version (see proxy-ng(5))

ITEMS AND SECTIONS

Program **cml** recognizes following items and sections:
* shared-file *name* { ... }
* shared-dir *name* { ... }
* system *name* { ... }

Description:

`shared-file` *name* **{**

path ... ;

format ... ;

}

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`shared-dir` *name* **{**

path ... ;

}

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

`system` *name* **{**

product ... ;

admin ... ;

hostname ... ;

domain ... ;

kernun-root ... ;

usb-auto-setup ... ;

apply-host ... ;

config-sync ... ;

users { ... }

sysctl { ... }

* interface *name* { ... }

ipv6-router ... ;

ipv6-addrctl { ... }

pikemon { ... }

routes { ... }

rc-conf { ... }

hosts-table { ... }

* rotate-log *name* { ... }

ntp { ... }

dhcp-server { ... }

dhcp6-server { ... }

crontab { ... }

periodic-conf { ... }

local-mailer { ... }

* ssh-server *name* { ... }

ssh-keys { ... }

ica-auto ... ;

icamd { ... }

icasd { ... }

watch { ... }

* acl *name* { ... }

use-services ... ;

use-resolver ... ;

* resolver *name* { ... }

* nameserver *name* { ... }

* ns-list *name* { ... }

* atrmon *name* { ... }

* pf-queue *name* { ... }

packet-filter { ... }

adaptive-firewall { ... }

alertd { ... }

bird4 { ... }

bird6 { ... }

rtadvd { ... }

* ssl-params *name* { ... }

* fake-cert *name* { ... }

* html-filter *name* { ... }

* mail-filter *name* { ... }

* aproxy *name* { ... }
* radius-client *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* antivirus *name* { ... }
* antispam *name* { ... }
* smtp-forwarder *name* { ... }
* web-filter *name* { ... }

  clear-web-db { ... }
* openvpn *name* { ... }

  ipsec-global { ... }
* ipsec-remote *name* { ... }
* ipsec *name* { ... }
* data-match *name* { ... }
* ntlm-auth *name* { ... }
* kerberos-auth *name* { ... }

  cwcatd { ... }

  snmpd { ... }

  http-cache { ... }

  update { ... }

  feedback { ... }

  stats { ... }

  stats-daily { ... }

  stats-weekly { ... }

  stats-monthly { ... }
* tcp-proxy *name* { ... }
* udp-proxy *name* { ... }
* dns-proxy *name* { ... }
* ftp-proxy *name* { ... }
* gk-proxy *name* { ... }
* h323-proxy *name* { ... }
* http-proxy *name* { ... }
* icap-server *name* { ... }
* imap4-proxy *name* { ... }

* pop3-proxy *name* { ... }

* sip-proxy *name* { ... }

* smtp-proxy *name* { ... }

* sqlnet-proxy *name* { ... }

* proxy-ng *name* { ... }

  proxy-ng-transp-ports ... ;

}

The `system` section is derived from `system` section prototype. For detail description of it, see system(5).

SEE ALSO

configuration(7), acl(5), adaptive-firewall(5), antivirus(5), application(5), atr(5), auth(5), clear-web-db(5), common(5), dns-proxy(5), ftp-proxy(5), http-proxy(5), imap4-proxy(5), interface(5), ipc(5), ipsec(5), ldap(5), license(5), listen-on(5), log(5), mod-html-filter(5), mod-mail-doc(5), mod-match(5), nameserver(5), ntp(5), openvpn(5), packet-filter(5), pf-queue(5), pike(5), pop3-proxy(5), proxy-ng(5), radius(5), resolver(5), router(5), sip-proxy(5), smtp-proxy(5), snmpd(5), source-address(5), sqlnet-proxy(5), ssh(5), ssl(5), sysctl(5), system(5), time(5), udpserver(5), update(5)

NAME

ldap — format of ldap component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ldap** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **ldap** configuration directives:

ldap-tls-reqcert-mode (name-usage obligatory)

Specifies what checks to perform on LDAP server certificates in a TLS session, if any.

never

The client (fw) will not request or check any server certificate.

demand

The server certificate is requested. If no certificate is provided, or bad certificate is provided, authentication immediately fails.

ldap-search-scope (name-usage obligatory)

Specifies the scope for searching users and groups.

subtree

Search the subtree of the given node.

onelevel

Search the level that is directly below the given node.

ldap-group-match (name-usage obligatory)

How a group name obtained from LDAP is matched in ACL and logged.

short

Match and log only the first component (CN) of the group name.

domain

Match and log the first component (CN) of the group name with appended '@DOMAIN' (extracted from the DC components of the group distinguished name), for example, Users@EXAMPLE.COM.

ITEMS AND SECTIONS

Configuration of **ldap** library component consists of following prototypes:
* ldap-client-auth *name* { ... }

Description:

ldap-client-auth *name* **{**

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

LDAP Client authorisation attributes.

Server identification and description of its attributes.

Constraints:

Item server required.

Items BINDINFO a KERBEROS are mutually exclusive.

Item BINDINFO or KERBEROS required.

Pair of items USERS and GROUPS or item ACTIVE-DIRECTORY required.

Item ACTIVE-DIRECTORY is mutually exclusive with USERS and GROUPS.

Items & subsections:

server *uri* [**timeout** *timeout*];

Definition of LDAP server location.

uri **(type: str-list)**

URI of the ldap server. If more than one server is configured, the first accessible server will be used.

timeout *timeout* **(type: uint32, optional, default: 2)**

Timeout for ldap operations; in seconds.

ssl {

id ... ;

auth-cert ... ;

tls-reqcert ... ;

}

Items & subsections:

id *private-key certificate*;

Private key and certificate.

private-key (type: **name** of **shared-file**, see common(5))

The file that contains the private key that matches the certificate stored in the 'certificate' file. The private key must not be protected with a password, so it is of critical importance that the key file is protected carefully.

certificate (type: **name** of **shared-file**, see common(5))

The file that contains the client certificate.

auth-cert [**file** *file*] [**dir** *dir*];

Certificates of trusted certification authorities.

file *file* (type: **name** of **shared-file**, see common(5), optional, default: NULL)

The file that contains certificates for all of the Certificate Authorities the LDAP client will recognize.

dir *dir* (type: **name** of **shared-dir**, see common(5), optional, default: NULL)

The directory that contains Certificate Authority certificates in separate individual files. File is always used before dir.

tls-reqcert [*tls-reqcert-mode*];

tls-reqcert-mode (type: **ldap-tls-reqcert-mode**, optional, default: demand)

Specifies what checks to perform on LDAP server certificates in a TLS session, if any.

[End of section `ldap-client-auth.ssl` description.]

bindinfo *binddn bindpasswd*;

binddn (type: **str**)

Distinguished name (dn) of the user for accessing the LDAP directory.

bindpasswd (type: **str**)

Password for accessing the LDAP directory.

kerberos;

Use Kerberos authentication for accessing the LDAP directory. A valid Kerberos ticket is needed.

users *dnusers* [**uname-attr** *uname-attr*] [**gidnum-attr** *gidnum-attr*] [**search-scope** *search-scope*];

Definition of user list properties.

dnusers (type: **str**)

Distinguished name (dn) of the node where the user list is stored within LDAP directory.

uname-attr *uname-attr* (type: **str**, optional, default: "uid")

Attribute name where user name is stored within the user definition node.

gidnum-attr *gidnum-attr* (type: **str**, optional, default: "gidNumber")

Attribute name where default group is stored within the user definition node.

search-scope *search-scope* **(type: ldap-search-scope, optional, default: onelevel)**

Specifies where to search for the users (directly below the dnusers or in the whole subtree of dnusers).

groups *dngroups* **[gidnum-attr** *gidnum-attr***] [gmember-attr** *gmember-attr***] [gname-attr** *gname-attr***] [search-scope** *search-scope***];**

Distinguished name (dn) of the node where the group list is stored within LDAP directory.

dngroups **(type: str)**

Distinguished name (dn) of the node where group list is stored within LDAP directory.

gidnum-attr *gidnum-attr* **(type: str, optional, default: "gidNumber")**

Attribute name where group ID is stored within the group definition node.

gmember-attr *gmember-attr* **(type: str, optional, default: "memberUID")**

Attribute name where members of the group are stored within the group definition node.

gname-attr *gname-attr* **(type: str, optional, default: "cn")**

Attribute name where group name is stored within the group definition node.

search-scope *search-scope* **(type: ldap-search-scope, optional, default: onelevel)**

Specifies where to search for the groups (directly below the dngroups or in the whole subtree of dngroups).

active-directory *domain* **[group-match** *group-match***] [users-search-base** *users-search-base***] [users-object** *users-object***] [username-attribute** *username-attribute***] [member-of-attribute** *member-of-attribute***];**

The LDAP server is Microsoft Windows Active Directory.

domain **(type: str)**

Domain name used by the AD.

group-match *group-match* **(type: ldap-group-match, optional, default: short)**

How a group name obtained from LDAP is matched in ACL and logged.

users-search-base *users-search-base* **(type: str, optional, default: "")**

Starting point for the search of users within the AD; if omitted, USERS-SEARCH-BASE is created from DOMAIN.

users-object *users-object* **(type: str, optional, default: "user")**

Name of the object class for users

username-attribute *username-attribute* **(type: str, optional, default: "sAMAccountName")**

Attribute where the username is stored

member-of-attribute *member-of-attribute* **(type: str, optional, default: "memberOf")**

Attribute where the group membership is stored within the user object

[End of section `ldap-client-auth` description.]

SEE ALSO

configuration(7), common(5)

NAME

license — format of license component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **license** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **license** configuration directives:

product-type (name-usage obligatory)

Type of a Kernun product.

unspecified

Product type not specified, KERNUN (Kernun Net Access) assumed, but not checked during application of the configuration.

kernun

Products based on the Kernun Net Access code.

component-group (name-usage obligatory)

Groups of licensed components.

adaptive-kernun

kernun

kernun-net-access

kernun-mail-access

kernun-vpn-access

kernun-office-access

kernun-web-access

kernun-business-intelligence

kernun-clear-web

kernun-secure-box

kernun-secure-box-retail

kernun-atr

kernun-antivirus

modules-data-scanning

modules-secure-box

modules-web-filter

modules-clear-web-db

component-type (name-usage obligatory)

Types of licensed components.

product-kernun

product-kernun-net-access

product-kernun-mail-access

product-kernun-vpn-access

product-kernun-office-access

product-kernun-web-access

product-kernun-clear-web

product-kernun-secure-box

product-kernun-secure-box-retail

atc

atr

clear-web-db

dns-proxy

ftp-proxy

gk-proxy

h323-proxy

http-cookie

http-proxy

icap-server

imap4-proxy

isds

isds-retail

ips

kernun-business-intelligence

mod-antivirus

mod-antivirus-agent

mod-antispam

mod-kav-httpd

mod-match

mod-match-replace

mod-pwf

pop3-proxy

sip-proxy

smtp-proxy

sqlnet-proxy

tcp-proxy

udp-proxy

ITEMS AND SECTIONS

Configuration of **license** library component consists of following prototypes:

SEE ALSO

configuration(7)

NAME

listen-on — format of listen-on component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **listen-on** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **listen-on** configuration directives:

ip-version (see **common**(5))

osi4-proto (see **common**(5))

listen-on-sock (name-usage obligatory)

address
: Listening socket defined by IP address.

iface
: Listening socket defined by INTERFACE section.

ITEMS AND SECTIONS

Configuration of **listen-on** library component consists of following prototypes:

listen-on { ... }

Description:

listen-on {

* non-transparent ... ;

* transparent ... ;

}

: Addresses to listen on (both for transparent and non-transparent traffic).

Items & subsections:

non-transparent [address] *addr* **[version** *version*] **port** *port* **[to** *to*] **[proto** *proto*];

non-transparent iface *iface* **[alias** *alias*] **[version** *version*] **port** *port* **[to** *to*] **[proto** *proto*];

Sockets to bind for non-transparent connections.

Socket can be defined by an IP address. However, in this case, the proxy cannot be started if the interface is not ready in the time when the proxy is started.

The second way is to specify the interface section name. In this case, the proxy takes interface address and bind it.

Using of an alias address instead of the interface main one can be specified by an ALIAS element.

<**branching element**> (**type: listen-on-sock**, **optional**, **default: address**)

addr (**type: host**)

Address to be bound

iface (**type: name of interface**, see interface(5))

Interface to be bound

alias *alias* (**type: str-set**, **optional**, **default: {}**)

Aliases names

version *version* (**type: ip-version**, **optional**, **default: undefined**)

IP version selection

port *port* (**type: port**)

Port to be bound (lowest)

to *to* (**type: port**, **optional**, **default: 0**)

Highest port to be bound

proto *proto* (**type: osi4-proto**, **optional**, **default: default**)

TCP/UDP selection

Constraints:

Port number must be positive.

Port upper bound must be higher than lower one.

transparent [address] *addr* **[version** *version*] **port** *port* **[to** *to*] **[proto** *proto*] **[server-addr** *server-addr*];

transparent iface *iface* **[version** *version*] **port** *port* **[to** *to*] **[proto** *proto*] **[server-addr** *server-addr*];

Sockets to bind for transparent connections.

Socket can be defined by an IP address. In this case, the proxy derives the proper interface name from it to detect traffic that should be processed by the proxy.

The second way is to specify the interface section name directly.

<**branching element**> (**type: listen-on-sock**, **optional**, **default: address**)

addr (**type: host**)

Address to be bound

iface (type: **name** of **interface**, see interface(5))
: Interface to be bound

version *version* (type: **ip-version**, optional, default: undefined)
: IP version selection

port *port* (type: **port**)
: Port to be bound (lowest)

to *to* (type: **port**, optional, default: 0)
: Highest port to be bound

proto *proto* (type: **osi4-proto**, optional, default: default)
: TCP/UDP selection

server-addr *server-addr* (type: **host**, optional, default: [0.0.0.0])
: Server's IP address

Constraints:
: Port number must be positive.

: Port upper bound must be higher than lower one.

[End of section `listen-on` description.]

SEE ALSO

configuration(7), common(5), interface(5)

NAME

log — format of log component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **log** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **log** configuration directives:

time-cond (see **common**(5))

zip-mode (see **common**(5))

dbglev (name-usage optional)

Log Debugging Level.

error (3)

Only errors and statistical messages are logged.

warning (4)

Errors, warnings and statistical messages are logged.

normal (6)

Normal level of logging, all operation messages are logged, no debugging.

debug (7)

Debugging level, firewall operations are logged in detail.

trace (8)

Tracing level, firewall routines calls can be traced.

full (9)

Full debug level, full data flow is logged.

logfail-mode (name-usage obligatory)

Logging to syslog failure mode.

This enumeration describes proxy behavior in case of syslog daemon is not operating.

ignore

Ignore syslog write failure.

file

Switch logging to file.

stop

Stop proxy immediately.

ITEMS AND SECTIONS

Configuration of **log** library component consists of following prototypes:
log { ... }

Description:

log {

level ... ;

mem-level ... ;

facility ... ;

file ... ;

rotate ... ;

mem-file ... ;

syslog-failure ... ;

data-limit ... ;

dump-hold-time ... ;

}

Firewall logging parameters.

If omitted, default values of all attributes are used.

Constraints:

FACILITY and FILE are mutually exclusive.

Cannot use ROTATE without FILE.

Items & subsections:

level [*value*];

Log debugging level.

value **(type: dbglev, optional, default: normal=6)**

Constraints:

Logging of error messages cannot be switched off.

Maximal logging level is 9 (full).

mem-level *value*;

Log level for transient logging to memory.

value (type: **dbglev**)

Constraints:

Logging of error messages cannot be switched off.

Maximal logging level is 9 (full).

facility *value*;

Syslog LOCALn facility number.

value (type: **uint8**)

Constraints:

Maximal facility number is 7.

file *name* [**usec**];

Filename used for logging instead of syslogd daemon.

If omitted, syslogd is used.

name (type: **str**)

usec (type: **key**, optional)

Log time with microseconds.

rotate [**user** *user*] [**group** *group*] [**mode** *mode*] [**count** *count*] [**size** *size*] [*when* [*zip*]];

Log file rotation description.

If user not specified, PROXY-USER is used.

user *user* (type: **str**, optional, default: <**NULL**>)

Log file owner - user.

group *group* (type: **str**, optional, default: "wheel")

Log file owner - group.

mode *mode* (type: **uint16**, optional, default: 640)

Log file permissions.

count *count* (type: **uint16**, optional, default: 31)

Number of days being archived.

size *size* (type: **uint16**, optional, default: 0)

Size limit for rotation in KB (ignore log file size if omitted).

when (type: **time-cond**, optional, default: anytime)

Rotation periodicity (use SIZE condition if omitted).

zip (type: **zip-mode**, optional, default: bzip2)

Zipping mode.

Constraints:

Use either size criterion or defined periodicity.

mem-file *name* [*size*];

File name (.PID will be added) and size for logging to memory.

name (type: **str**)

size (type: **uint32**, optional, default: **16384**)

syslog-failure [ignore];

syslog-failure file *file*;

syslog-failure stop;

Proxy behavior in case of syslog failure.

<branching element> (type: **logfail-mode**, optional, default: **ignore**)

file (type: **str**)

data-limit [*bytes***];**

Per block data limit for full log.

This limit is used as a default value, it can be redefined by means of SOCK-OPT.LOG-LIMIT (see netio(5) manual page).

bytes (type: **uint32**, optional, default: **128**)

dump-hold-time [*seconds***];**

Extensive log dump hold time.

Proxies log in various situations very extensive dumps. Logging of this dump very often is both space consuming and meaningless, so the frequence of such dumps can be controlled by this item.

seconds (type: **uint32**, optional, default: **60**)

[End of section `log` description.]

SEE ALSO

configuration(7), common(5), netio(5), logging(7)

NAME

mod-antispam — format of mod-antispam component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **mod-antispam** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **mod-antispam** configuration directives:

`enabling` (see **common**(5))

ITEMS AND SECTIONS

Configuration of **mod-antispam** library component consists of following prototypes:
* antispam *name* { ... }
  use-antispam ... ;

Description:

`antispam` *name* **{**

connection ... ;

sock-opt { ... }

altq ... ;

}

Channel to antispam daemon.

This global section defines the way to communicate with selected antispam daemon. Name of such section is to be used in particular proxy configuration when defining mode of operation.

The current version of antispam module has implemented usage of the only antispam daemon, namely SpamAssassin (spamd). However, the antispam score is multiplied by 1000 for future compatibility. For the same reason, the negative values are changed to zero.

If the check fails, the SPAM-SCORE value is set to a special value (-2) which is later matched by special value of UNKNOWN (see common(5)).

Constraints:

Connection parameters of SpamAssassin server must be defined.

Items & subsections:

`connection` *socket*;

Socket address of SpamAssassin (spamd).

socket **(type: `sock`)**

`sock-opt` {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection to spamd options.

The `sock-opt` section is derived from `sock-opt` section prototype.

For detail description of it, see netio(5).

Changes to the `sock-opt` section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`altq` *altq* [**`paltq`** *paltq*];

ALTQ queues for data sent to antispam.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**

queue name

`paltq` *paltq* **(type: `name` of `pf-queue`, see pf-queue(5), optional, default: NULL)**

priority queue name (if set, used for TCP ACK without data)

[End of section `antispam` description.]

`use-antispam disable`;

`use-antispam enable` *channel* [*limit*];

Antispam usage.

This section defines type of antispam daemon used and mode of antispam checking operation.

<branching element> (type: `enabling`)

channel **(type: `name` of `antispam`, see above)**

Name of antispam global section used.

Referred section defines the way how to communicate with the antispam daemon (see above).

limit **(type: `uint64`, optional, default: 0)**

Size limit (in bytes) for antispam check.

Antispam checking used to be very exhausting operation, and typical spam mails used to be not very large (both for passing by size limit filters and for being able to send a lot of copies). That's why it can be desired to avoid checking of very large mails.

Setting of this limit says antispam module not to check mails larger than given limit and declare their spam score to zero.

Setting this limit to zero disables this feature and enables using of antispam to all mails. Be prepared for high machine load and noticeable delay in delivery if used so.

SEE ALSO

configuration(7), common(5), netio(5), pf-queue(5)

NAME

mod-html-filter — format of mod-html-filter component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **mod-html-filter** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **mod-html-filter** configuration directives:

`accept-deny` **(name-usage obligatory)**

Specifies whether a particular part of a document should be accepted (kept in the document) or denied (deleted from the document).

`accept`

`deny`

ITEMS AND SECTIONS

Configuration of **mod-html-filter** library component consists of following prototypes:
* filter-spec ... ;
* html-filter *name* { ... }

Description:

`filter-spec` ***action val*;**

Various instances of this prototype item (e.g., script-tag-language) control deleting some parts of the document. If no item is present, nothing is deleted. If at least one item is present, the first one with matching VAL determines the result according to its ACTION. If no item matches, the respective part of the document is deleted.

action **(type: `accept-deny`)**

val **(type: `str-set`)**

html-filter *name* {

* script-tag-language ... ;
  replace-head-script-tags ... ;
  replace-body-script-tags ... ;
* style-tag-type ... ;
  replace-style-tags ... ;
* iframe-tag-src ... ;
  replace-iframe-tags ... ;
* intrinsic-language ... ;
* intrinsic-hack ... ;
  replace-intrinsic ... ;
* macro-language ... ;
* macro-hack ... ;
  replace-macros ... ;
* uri ... ;
  replace-uri ... ;
* embed-tag-type ... ;
* embed-src-hack ... ;
* embed-plugin-hack ... ;
  replace-head-embed-tags ... ;
  replace-body-embed-tags ... ;
* applet ... ;
  replace-applets ... ;
* object ... ;
* object-classid-hack ... ;
* object-data-hack ... ;
  replace-head-object-tags ... ;
  replace-body-object-tags ... ;
* param-tags ... ;
  replace-param ... ;
  script-end-hack ... ;

}

Settings of HTML filtration.

Items & subsections:

script-tag-language *action val*;

Controls deleting <SCRIPT> elements. If this item is not present, all scripts are preserved. If at least one item is present, the first one with matching VAL determines the result according to its ACTION. If no item matches, the script is deleted. A script with unspecified language is matched by the empty string. According to HTML definition, each <SCRIPT> tag should contain attribute TYPE which defines the scripting language. Instead of TYPE, (deprecated) attribute LANGUAGE can be used. If both TYPE and LANGUAGE are present, http-proxy makes its decisions according to TYPE (as browsers usually do). A script without a specification of scripting language is a HTML error, but browsers often treat such scripts as JavaScript.

action (type: **accept-deny**)

val (type: **str-set**)

replace-head-script-tags *val*;

Replacement text for deleted scripts in <HEAD>.

val (type: **str**)

replace-body-script-tags *val*;

Replacement text for deleted scripts. in <BODY>.

val (type: **str**)

style-tag-type *action val*;

Controls deleting <STYLE> elements. Matching is done like in SCRIPT-TAG-LANGUAGE. The empty string matches an unknown type (missing TYPE attribute).

action (type: **accept-deny**)

val (type: **str-set**)

replace-style-tags *val*;

Replacement text for deleted <STYLE> elements.

val (type: **str**)

iframe-tag-src *action val*;

Controls deleting <IFRAME> elements.

action (type: **accept-deny**)

val (type: **str-set**)

replace-iframe-tags *val*;

Replacement text for deleted <IFRAME> elements.

val (type: **str**)

intrinsic-language *action val*;

Controls deleting intrinsic events according to scripting language. Like SCRIPT-TAG-LANGUAGE, but controls deleting attributes defining intrinsic event scripts (e.g.,

ONCLICK attribute in HTML). Scripting language is defined by Content-Script-Type HTTP header (or an equivalent <META> HTML tag). Occurrence of an intrinsic event in a document without scripting language definition is a HTML error, but browsers treat such scripts as JavaScript. Such an intrinsic event with undefined scripting language is matched by the empty string.

action (type: **accept-deny**)

val (type: **str-set**)

intrinsic-hack *action val*;

Controls deleting intrinsic events according to attribute value. If both INTRINSIC-LANGUAGE and INTRINSIC-HACK are used then being denied by one of the directives suffices for an intrinsic attribute to be deleted.

action (type: **accept-deny**)

val (type: **str-set**)

replace-intrinsic *val*;

Replacement of deleted intrinsic events.

val (type: **str**)

macro-language *action val*;

Controls deleting script macros according to scripting language. Like SCRIPT-TAG-LANGUAGE, but controls deleting attributes with script macros in HTML attribute values (ATTR NAME="...&{SCRIPT};..."). Scripting language is defined by Content-Script-Type HTTP header (or an equivalent <META> HTML tag). Occurrence of an intrinsic event in a document without scripting language definition (matched by the empty string) is a HTML error, but browsers treat such scripts as JavaScript.

action (type: **accept-deny**)

val (type: **str-set**)

macro-hack *action val*;

Controls deleting script macros according to attribute value. Like INTRINSIC-HACK, but controls deleting attributes with script macros in attribute values. If both MACRO-LANGUAGE and MACRO-HACK are used then being denied by one of the directives suffices for an attribute with a script macro to be deleted.

action (type: **accept-deny**)

val (type: **str-set**)

replace-macros *val*;

Replacement of script macros.

val (type: **str**)

uri *action val*;

Controls deleting URI-valued attributes. Matching is done like in SCRIPT-TAG-LANGUAGE. The following attributes are considered as URI-valued:

ACTION, ARCHIVE, BACKGROUND, CITE, CLASSID, CODE, CODEBASE, DATA, HREF, LONGDESC, PROFILE, SRC, USEMAP. Before matching an URI, it is always combined with the base URI of the document. Therefore, the HREF attribute in <BASE> is never deleted, because it defines the base URI for other relative URIs. Note that if something looks like an URI with unknown scheme (e.g., foo://foo srv/), browsers often assume that it is server name and translate it to "http://foo:80//foo srv/".

action (type: **accept-deny**)

val (type: **str-set**)

replace-uri *val*;

Replacement of deleted URI-valued attributes.

val (type: **str**)

embed-tag-type *action val*;

Controls deleting <EMBED> elements according to type. Matching is done like in SCRIPT-TAG-LANGUAGE. Arguments of this directive are matched with attributes TYPE, LANG, and LANGUAGE of an <EMBED>. If more than one of these attributes are present, the element is not deleted only if all match. Unknown type is matched by the empty string.

action (type: **accept-deny**)

val (type: **str-set**)

embed-src-hack *action val*;

Controls deleting <EMBED> elements according to SRC attribute. Matching is done like in SCRIPT-TAG-LANGUAGE. The value of the attribute is compared to the set of allowed sources from this directive. Comparison is performed as for ALLOWED-URI, i.e., the value of SRC is combined with the base URI of the document.

action (type: **accept-deny**)

val (type: **str-set**)

embed-plugin-hack *action val*;

Controls deleting <EMBED> elements according to PLUGINSPAGE attribute. Matching is done like in SCRIPT-TAG-LANGUAGE.

action (type: **accept-deny**)

val (type: **str-set**)

replace-head-embed-tags *val*;

Replacement of deleted embeds in <HEAD>.

val (type: **str**)

replace-body-embed-tags *val*;

Replacement of deleted embeds in <BODY>.

val (type: **str**)

applet *action val*;

Controls deleting Java applets (elements <APPLET>) according to code location - attributes CODE, OBJECT. Both attributes CODE and OBJECT must pass the test in order to ACCEPT <APPLET> in the document, but matching one of them suffices to DENY. Matching is done like in ALLOWED-URI. If a base URI is specified for a particular <APPLET> element by its CODEBASE attribute, it takes precedence over a document-wide base URI. If CODEBASE is relative, it is combined with the document-wide base URI first.

action (type: **accept-deny**)

val (type: **str-set**)

replace-applets *val*;

Replacement of deleted applets.

val (type: **str**)

object *action val*;

Controls deleting <OBJECT> elements according to their types given by attributes TYPE and CODETYPE. Matching is done like in SCRIPT-TAG-LANGUAGE. Arguments of this directive are matched with attributes TYPE and CODETYPE of an <OBJECT>. If both these attributes are present, matching both is required to ACCEPT the object, but matching one of them suffices for DENY. If none of them is present, the empty string matches.

action (type: **accept-deny**)

val (type: **str-set**)

object-classid-hack *action val*;

Controls deleting <OBJECT> elements according to class-id - attribute CLASSID. The value of the attribute is compared to the list of allowed class-id's from this directive. Comparison is performed as for ALLOWED-URI, i.e., the value of CLASSID is combined with the base URI. If a base URI is specified for a particular <OBJECT> element by its CODEBASE attribute, it takes precedence over a document-wide base URI. If CODEBASE is relative, it is combined with the document-wide base URI first.

action (type: **accept-deny**)

val (type: **str-set**)

object-data-hack *action val*;

Controls deleting <OBJECT> elements according to data location - attribute DATA. The value of the attribute is compared to the list of allowed locations from this directive. Comparison is performed as for ALLOWED-URI, i.e., the value of DATA is combined with the base URI. If a base URI is specified for a particular <OBJECT> element by its CODEBASE attribute, it takes precedence over a document-wide base URI.

action (type: **accept-deny**)

val (type: **str-set**)

replace-head-object-tags *val*;

Replacement of deleted objects in <HEAD>.

val (type: **str**)

replace-body-object-tags *val*;

Replacement of deleted objects in <BODY>.

val (type: **str**)

param-tags *action name value*;

Controls deleting <PARAM> tags according to their attributes NAME and VALUE. If this item is not present, all <PARAM> tags are preserved. If at least one item is present, the first one with matching both NAME and VALUE determines the result according to its ACTION. If no item matches, the tag is deleted.

action (type: **accept-deny**)

name (type: **str-set**)

value (type: **str-set**)

replace-param *val*;

Replacement of deleted <PARAM> tags.

val (type: **str**)

script-end-hack;

How <SCRIPT> and <STYLE> elements are terminated. If Off, scripts in SCRIPT elements and content of STYLE elements end at the first occurrence of "</" followed by letter (according to HTML 4.0 Specification). If set, scripts end only at </SCRIPT> or </STYLE> tag, respectively, which is how scripts are usually treated by browsers.

[End of section `html-filter` description.]

SEE ALSO

configuration(7)

NAME

mod-mail-doc — format of mod-mail-doc component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **mod-mail-doc** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **mod-mail-doc** configuration directives:

direction (see **common**(5))

range-op (see **common**(5))

week-day (see **time**(5))

month (see **time**(5))

virus-status (see **antivirus**(5))

transparency (see **acl**(5))

user-auth-spec (see **acl**(5))

doctype-ident-method (see **acl**(5))

header-op (see **acl**(5))

mail-hdr (name-usage obligatory)

STMP/MIME Headers

Unknown

Date

From

Sender

Reply-To

To

Cc

Bcc

Message-ID

In-Reply-To

References

Subject

Comments

Keywords

Resent-Date

Resent-From

Resent-Sender

Resent-Reply-To

Resent-To

Resent-Cc

Resent-Bcc

Resent-Message-ID

Return-Path

Received

MIME-Version

Content-Type

Content-Transfer-Encoding

Content-ID

Content-Description

Content-Disposition

DKIM-Signature

X-Kernun-Loop-Info

X-Kernun-Spam-Score

X-Kernun-Virus-Status

X-Kernun-Virus-Name

X-Kernun-Quarantine-Tag

X-Orig-Content-Type

X-Orig-Content-Disposition

content-transfer-encoding (name-usage obligatory)

MIME Content-Transfer-Encoding types.

eight-bit

quoted-printable

base64

smtp-error (name-usage obligatory)

Types of SMTP command errors.

cmd-missing

SMTP command missing.

arg-invalid

SMTP cmd argument has invalid syntax.

too-long

SMTP cmd argument exceeds RFC size limitation.

no-domain

SMTP cmd argument has no domain part.

ip-address

SMTP cmd argument with IP address (deprecated).

unknown-perm

SMTP cmd argument DNS resolution failed permanently.

unknown-temp

SMTP cmd argument DNS resolution failed temporarily.

hdr-handling (name-usage obligatory)

Types of header line handling.

fail

Mail processing fails.

keep

Line is not changed.

remove

Line is removed.

mail-reaction (name-usage obligatory)

Types of reaction to mail RFCs violations.

reject

The mail is rejected.

pass

The text is passed without changes, still violating the RFC.

drop

The element (e.g. header line) is dropped from the mail.

fix

The text is corrected according to the RFC.

mail-fallback (name-usage obligatory)

Types of fallback reaction to mail RFCs violations.

reject

The mail is rejected.

pass

The text is passed without changes, still violating the RFC.

drop

The element (e.g. header line) is dropped from the mail.

mime-header-check-type (name-usage obligatory)

MIME headers matching types.

txt

Header is matched after decoding to actual value and converting to ISO-8859-2 (ignoring inconvertible chars).

enc

Header is matched after decoding and converting to UTF-8, pattern is given in MIME format encoding.

raw

Header is matched without decoding, in raw format.

err

Fallback choice for the case of header decoding error.

ITEMS AND SECTIONS

Configuration of **mod-mail-doc** library component consists of following prototypes:
* smtp-adr-check ... ;
* mail-correction ... ;
* mail-hdr-correction ... ;
* mail-len-correction ... ;
* mail-hdr-len-correction ... ;
* mail-filter *name* { ... }
* mime-header-check ... ;
* mail-acl *name* { ... }
* mail-doc-acl *name* { ... }

Description:

smtp-adr-check [*addrs*];

This item defines SMTP addresses matching conditions.

addrs (type: **str-set**, optional, default: *)

Set of addresses matching this configuration item.

The matching rules are slightly modified for the purpose of email addresses matching:

- Regular expressions are matched as usual.
- Addresses defined by string ended with the '@' character match given local-part in any domain.
- Addresses defined by string beginning by the '@' character match any address within given domain.
- Other addresses containing the '@' character match exactly given address.
- Addresses defined by string containing no '@' character match any address within given domain and any its subdomain.

mail-correction [*clear* [*signed* [**fallback** *fallback*]]];

Items of this type define corrections of various mail RFCs violations.

clear (type: **mail-reaction**, optional, default: fix)

reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)

reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)

reaction in case of fixing failure

mail-hdr-correction [*clear* [*signed* [**fallback** *fallback*] [*header*]]];

Items of this type define corrections of various mail RFCs violations, where the decision is based on the header name.

clear (type: **mail-reaction**, optional, default: fix)

reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)

reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)

reaction in case of fixing failure

header (type: **str-set**, optional, default: *)

mail-len-correction [*clear* [*signed* [**fallback** *fallback*] [*limit*]]];

Items of this type define corrections of various mail RFCs violations, where the decision is based on the line length.

clear (type: **mail-reaction**, optional, default: fix)

reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)

reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)

reaction in case of fixing failure

limit (type: **uint16**, optional, default: 0)

Lower bound for line length; lines shorter than this limit are not affected by this item.

mail-hdr-len-correction [*clear* [*signed* [**fallback** *fallback*] [*header* [*limit*]]]];

Items of this type define corrections of various mail RFCs violations, where the decision is based on the header name and line length.

clear (type: **mail-reaction**, optional, default: fix)

reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)

reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)

reaction in case of fixing failure

header (type: **str-set**, optional, default: *)

limit (type: **uint16**, optional, default: 0)

Lower bound for line length; lines shorter than this limit are not affected by this item.

mail-filter *name* {

stamp-limit ... ;

stamp-filter ... ;

* unflagged-8bit ... ;

* bad-end-of-line ... ;

* invalid-header ... ;

* long-header-lines ... ;

* invalid-chars ... ;

* header-8bit-chars ... ;

* bad-boundary-chars ... ;

* bad-boundary-length ... ;

* long-body-lines ... ;

* long-encoded-lines ... ;

enc-line-len ... ;

* bad-mime-struct ... ;

* invalid-encoding ... ;

  treat-rfc822-as-text ... ;

}

This section defines SMTP/MIME document handling attributes.

Items & subsections:

`stamp-limit` [*number* [*text*]];

Maximum number of Received-headers in mail.

This check prevents from mail loops of three categories:

- loops caused by wrong forwarding policy of a user
- loops caused by DNS resolution errors or attack
- loops caused by proxy of forwarder misconfiguration.

number **(type: `uint16`, optional, default: 30)**

text **(type: `str`, optional, default: <NULL>)**

If omitted, default text is used.

`stamp-filter`;

SMTP stamp headers removal.

Enabling this feature causes removing 'Received:' headers (containing local-dependent information) from mails. Number of removed lines is counted and added to mail as a special header X-Kernun-Loop-Info.

`unflagged-8bit` [*clear* [*signed* [**`fallback`** *fallback*]]];

Handling of unannounced 8bit transport.

Some MUAs ignore RFC and use non-US-ASCII characters in mail bodies or MIME documents without announcing (in MAIL commands and MIME headers) the 8bit transport.

By default, the proxy fixes unsigned and passes signed mails with this violation; this item allows to change this behavior.

clear **(type: `mail-reaction`, optional, default: fix)**

reaction to errors in non-signed parts of mail

signed **(type: `mail-reaction`, optional, default: pass)**

reaction to errors in signed parts of mail

`fallback` *fallback* **(type: `mail-fallback`, optional, default: pass)**

reaction in case of fixing failure

Constraints:

DROP reaction is not allowed here.

`bad-end-of-line` [*clear* [*signed* [**`fallback`** *fallback*]]];

Handling of incorrect EOL sequences.

Some buggy MUAs use no or two CR chars before LF.

By default, the proxy fixes unsigned and passes signed mails with this violation; this item allows to change this behavior.

clear **(type: `mail-reaction`, optional, default: fix)**
reaction to errors in non-signed parts of mail

signed **(type: `mail-reaction`, optional, default: pass)**
reaction to errors in signed parts of mail

`fallback` *fallback* **(type: `mail-fallback`, optional, default: pass)**
reaction in case of fixing failure

Constraints:
DROP reaction is not allowed here.

`invalid-header` **[***clear* **[***signed* **[`fallback`** *fallback***]** **[***header***]]];**

Handling of headers with invalid format.

Some MUAs create mail/MIME headers with various format errors (missing semicolon, unquoted chars like spaces, missing terminating double quote etc.).

Such headers cannot be fixed, so if you configure FIX reaction, the FALLBACK elem will be used, instead.

By default, the proxy passes all mails with bad headers, this item allows to change this behavior.

clear **(type: `mail-reaction`, optional, default: fix)**
reaction to errors in non-signed parts of mail

signed **(type: `mail-reaction`, optional, default: pass)**
reaction to errors in signed parts of mail

`fallback` *fallback* **(type: `mail-fallback`, optional, default: pass)**
reaction in case of fixing failure

header **(type: `str-set`, optional, default: *)**

`long-header-lines` **[***clear* **[***signed* **[`fallback`** *fallback***]** **[***header* **[***limit***]]]];**

Handling of too long headers.

Some MUAs create mail/MIME headers exceeding the allowed length limit (1000 chars).

By default, the proxy tries to pass signed mails with this violation and tries to fix it for unsigned mails. In case of unsuccessful correction it passes such a mail. This item allows to change this behavior.

clear **(type: `mail-reaction`, optional, default: fix)**
reaction to errors in non-signed parts of mail

signed **(type: `mail-reaction`, optional, default: pass)**
reaction to errors in signed parts of mail

`fallback` *fallback* **(type: `mail-fallback`, optional, default: pass)**
reaction in case of fixing failure

header **(type: `str-set`, optional, default: *)**

limit (type: **uint16**, optional, default: 0)

Lower bound for line length; lines shorter than this limit are not affected by this item.

invalid-chars [*clear* [*signed* [**fallback** *fallback*]]];

Handling of bare NUL and CR characters.

Some MUAs ignore RFC and use NUL char (0x00) and bare CR (not followed by LF) in mail bodies or document headers.

By default, the proxy fixes unsigned and passes signed mails with this violation, this item allows to change this behavior. The only exception is the case of NUL in header which cannot be handled at all.

clear (type: **mail-reaction**, optional, default: fix)

reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)

reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)

reaction in case of fixing failure

Constraints:

DROP reaction is not allowed here.

header-8bit-chars [*clear* [*signed* [**fallback** *fallback*] [*header*]]];

Handling of non-US-ASCII characters in headers.

Some MUAs ignore RFC and use non-US-ASCII characters in mail or MIME documents headers.

Such headers cannot be fixed, so if you configure FIX reaction, the FALLBACK elem will be used, instead.

By default, the proxy passes all mails with non ASCII headers, this item allows to change this behavior.

clear (type: **mail-reaction**, optional, default: fix)

reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)

reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)

reaction in case of fixing failure

header (type: **str-set**, optional, default: *)

bad-boundary-chars [*clear* [*signed* [**fallback** *fallback*]]];

Handling of BOUNDARY values with invalid characters.

Some MUAs ignore RFC and use invalid characters in boundaries.By default, the proxy fixes unsigned and passes signed mails with incorrect boundary content, this item allows to change this behavior.

clear (type: **mail-reaction**, optional, default: fix)

reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)
reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)
reaction in case of fixing failure

Constraints:
DROP reaction is not allowed here.

bad-boundary-length [*clear* [*signed* [**fallback** *fallback*]]];

Handling of BOUNDARY strings longer than 70 characters.

Some MUAs ignore RFC and use too long boundaries.By default, the proxy fixes all mails with too long boundary string, this item allows to change this behavior.

WARNING! This violation is very dangerous and not very often hence we strongly recommend not to pass such mails!

clear (type: **mail-reaction**, optional, default: fix)
reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)
reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)
reaction in case of fixing failure

Constraints:
DROP reaction is not allowed here.

long-body-lines [*clear* [*signed* [**fallback** *fallback*] [*limit*]]];

Handling of mail body lines longer than 998 character.

Some MUAs ignore RFC and use too long lines.By default, the proxy fixes unsigned and passes signed mails with this violation; this item allows to change this behavior.

WARNING! Allowing to pass very long lines may be dangerous!

clear (type: **mail-reaction**, optional, default: fix)
reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)
reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)
reaction in case of fixing failure

limit (type: **uint16**, optional, default: 0)
Lower bound for line length; lines shorter than this limit are not affected by this item.

Constraints:
DROP reaction is not allowed here.

long-encoded-lines [*clear* [*signed* [**fallback** *fallback*] [*limit*]]];

Handling of mail body encoded lines longer than 76 character.

Some MUAs ignore RFC and use too long lines.By default, the proxy fixes unsigned and passes signed mails with this violation; this item allows to change this behavior.

WARNING! Allowing to pass very long lines may be dangerous!

clear (type: **mail-reaction**, optional, default: fix)
reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)
reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)
reaction in case of fixing failure

limit (type: **uint16**, optional, default: 0)
Lower bound for line length; lines shorter than this limit are not affected by this item.

Constraints:
DROP reaction is not allowed here.

enc-line-len [*len*];

Encoded lines line length limit.

len (type: **uint16**, optional, default: 80)

bad-mime-struct [*clear* [*signed* [**fallback** *fallback*]]];

Handling of invalid structure of MIME documents.

Some MUAs ignore RFC and send MIME documents with invlaid format (e.g. missing empty lines as separatoes etc.).

By default, the proxy fixes unsigned and passes signed mails with this violation; this item allows to change this behavior.

clear (type: **mail-reaction**, optional, default: fix)
reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)
reaction to errors in signed parts of mail

fallback *fallback* (type: **mail-fallback**, optional, default: pass)
reaction in case of fixing failure

Constraints:
DROP reaction is not allowed here.

invalid-encoding [*clear* [*signed* [**fallback** *fallback*] [*header*]]];

Handling of invalid Content-Transfer-Encoding modes.

Some MUAs ignore RFC and use invalid Content-Transfer-Encoding (e.g. BINARY despite no meaning of it in current SMTP).By default, the proxy fixes unsigned and passes signed mails with this violation; fixing means converting to 8bit mode. This item allows to change this behavior, various encoding modes can be handled separately according their names.

clear (type: **mail-reaction**, optional, default: fix)
reaction to errors in non-signed parts of mail

signed (type: **mail-reaction**, optional, default: pass)
reaction to errors in signed parts of mail

fallback *fallback* **(type: mail-fallback, optional, default: pass)**

reaction in case of fixing failure

header **(type: str-set, optional, default: *)**

Constraints:

DROP reaction is not allowed here.

treat-rfc822-as-text;

Handle included documents as text.

If a mailserver reports mail delivery failure it can include original mail into the report. If such a mail is incorrect the proxy will reject whole mail. This item tells proxy to read included documents as plain text and thus to avoid rejection risc.

[End of section `mail-filter` description.]

mime-header-check *name* **[txt]** *value*;

mime-header-check *name* **enc** *pattern*;

mime-header-check *name* **raw** *value*;

mime-header-check *name* **err;**

Entry condition - MIME header content.

name **(type: str-set)**

Header name set (pure strings are case insensitive)

<branching element> (type: mime-header-check-type, optional, default: txt)

value **(type: str-set)**

Set of header values (in ISO-8859-2 charset).

pattern **(type: str-list)**

List of header patterns (case sensitive substrings in any charset) encoded in MIME format.

Constraints:

MIME encoded strings must comply with MIME definition.

mail-acl *name* **{**

* from ... ;

* time ... ;

time-period-set { ... }

* parent-acl ... ;

deny ... ;

accept ... ;

rule ... ;

direction ... ;

* size ... ;

* content-type ... ;

virus-status ... ;

* modify-header ... ;

replace ... ;

* sender ... ;

* recipient ... ;

* recipients ... ;

* spam-score ... ;

* header ... ;

from-quarantine ... ;

prefix-subject ... ;

}

The first ACL on the third level decides how to treat the whole mail after finishing all checks.

The `mail-acl` section is derived from `acl-3` section prototype. For detail description of it, see acl(5).
Changes to the `mail-acl` section:

Item `server` is not valid.

Item `user` is not valid.

Item `mime-type` is not valid.

Item `force-doctype-ident` is not valid.

Item `html-filter` is not valid.

Added items & subsections:

`sender` [*addrs*];

Entry condition (SMTP only) - MAIL FROM address.

addrs **(type: `str-set`, optional, default: *)**

Set of addresses matching this configuration item.

The matching rules are slightly modified for the purpose of email addresses matching:

- Regular expressions are matched as usual.
- Addresses defined by string ended with the '@' character match given local-part in any domain.
- Addresses defined by string beginning by the '@' character match any address within given domain.
- Other addresses containing the '@' character match exactly given address.

- Addresses defined by string containing no '@' character match any address within given domain and any its subdomain.

recipient [*addrs***];**

Entry condition (SMTP only) - RCPT TO address.

addrs **(type: `str-set`, optional, default: *)**

Set of addresses matching this configuration item.

The matching rules are slightly modified for the purpose of email addresses matching:

- Regular expressions are matched as usual.
- Addresses defined by string ended with the '@' character match given local-part in any domain.
- Addresses defined by string beginning by the '@' character match any address within given domain.
- Other addresses containing the '@' character match exactly given address.
- Addresses defined by string containing no '@' character match any address within given domain and any its subdomain.

recipients unknown [*addrs***];**

recipients lt ***limit*** **[***addrs***];**

recipients le ***limit*** **[***addrs***];**

recipients eq ***limit*** **[***addrs***];**

recipients ne ***limit*** **[***addrs***];**

recipients gt ***limit*** **[***addrs***];**

recipients ge ***limit*** **[***addrs***];**

recipients in ***lower upper*** **[***addrs***];**

recipients ni ***lower upper*** **[***addrs***];**

Entry condition - number of recipients.

<branching element> (type: `range-op`)

limit **(type: `uint64`)**

Tested value limitation.

lower **(type: `uint64`)**

Tested value lower bound.

upper **(type: `uint64`)**

Tested value upper bound.

addrs **(type: `str-set`, optional, default: *)**

Restriction to addresses - only ones matching this condition are counted.

Exact address matching conditions are described at SENDER or RECIPENT items.

spam-score unknown;

spam-score lt ***limit*****;**

spam-score le *limit*;

spam-score eq *limit*;

spam-score ne *limit*;

spam-score gt *limit*;

spam-score ge *limit*;

spam-score in *lower upper*;

spam-score ni *lower upper*;

Entry condition - antispam check score.

For score interpretation, see mod-antispam(5).

<**branching element**> (**type: range-op**)

limit (**type: uint64**)

Tested value limitation.

lower (**type: uint64**)

Tested value lower bound.

upper (**type: uint64**)

Tested value upper bound.

header *name* [**txt**] *value*;

header *name* **enc** *pattern*;

header *name* **raw** *value*;

header *name* **err**;

Entry condition - MIME header content.

name (**type: str-set**)

Header name set (pure strings are case insensitive)

<**branching element**> (**type: mime-header-check-type, optional, default: txt**)

value (**type: str-set**)

Set of header values (in ISO-8859-2 charset).

pattern (**type: str-list**)

List of header patterns (case sensitive substrings in any charset) encoded in MIME format.

Constraints:

MIME encoded strings must comply with MIME definition.

from-quarantine [*tags*];

Entry condition (SMTP only) - mail sent by admin from quarantine.

This item is valid in smtp-proxy only. If omitted, any mail can match this ACL.

tags (**type: str-set, optional, default: ***)

Allowed X-Kernun-Quarantine-Tag header values

prefix-subject *prefix*;

Change Subject: mail header.

***prefix* (type: `str`)**

String to be written before the original Subject.

Constraints:

Subject prefix must comply with RFC.

[End of section `mail-acl` description.]

`mail-doc-acl` *name* {

* from ... ;

* time ... ;

time-period-set { ... }

* parent-acl ... ;

deny ... ;

accept ... ;

rule ... ;

direction ... ;

* size ... ;

* content-type ... ;

* mime-type ... ;

virus-status ... ;

* modify-header ... ;

force-doctype-ident ... ;

replace ... ;

html-filter ... ;

* sender ... ;

* recipient ... ;

* spam-score ... ;

* header ... ;

* filename ... ;

from-quarantine ... ;

add-virus-names ... ;

}

The second ACL on the third level decides how to process particular document(s) contained in the mail.

The `mail-doc-acl` section is derived from `acl-3` section prototype.
For detail description of it, see acl(5).

Changes to the `mail-doc-acl` section:

Item `server` is not valid.

Item `user` is not valid.

Item ADD-VIRUS-NAMES is not allowed if DENY is on.

Added items & subsections:

`sender` [*addrs*];

Entry condition (SMTP only) - MAIL FROM address.

addrs **(type: `str-set`, optional, default: *)**

Set of addresses matching this configuration item.

The matching rules are slightly modified for the purpose of email addresses matching:

- Regular expressions are matched as usual.
- Addresses defined by string ended with the '@' character match given local-part in any domain.
- Addresses defined by string beginning by the '@' character match any address within given domain.
- Other addresses containing the '@' character match exactly given address.
- Addresses defined by string containing no '@' character match any address within given domain and any its subdomain.

`recipient` [*addrs*];

Entry condition (SMTP only) - RCPT TO address.

addrs **(type: `str-set`, optional, default: *)**

Set of addresses matching this configuration item.

The matching rules are slightly modified for the purpose of email addresses matching:

- Regular expressions are matched as usual.
- Addresses defined by string ended with the '@' character match given local-part in any domain.
- Addresses defined by string beginning by the '@' character match any address within given domain.
- Other addresses containing the '@' character match exactly given address.
- Addresses defined by string containing no '@' character match any address within given domain and any its subdomain.

`spam-score unknown`;

`spam-score lt` *limit*;

`spam-score le` *limit*;

`spam-score eq` *limit*;

`spam-score ne` *limit*;

spam-score gt *limit*;

spam-score ge *limit*;

spam-score in *lower upper*;

spam-score ni *lower upper*;

Entry condition - antispam check score.

For score interpretation, see mod-antispam(5).

<branching element> (type: **range-op**)

limit (type: **uint64**)

Tested value limitation.

lower (type: **uint64**)

Tested value lower bound.

upper (type: **uint64**)

Tested value upper bound.

header *name* [**txt**] *value*;

header *name* **enc** *pattern*;

header *name* **raw** *value*;

header *name* **err**;

Entry condition - MIME header content.

name (type: **str-set**)

Header name set (pure strings are case insensitive)

<branching element> (type: **mime-header-check-type**, optional, default: txt)

value (type: **str-set**)

Set of header values (in ISO-8859-2 charset).

pattern (type: **str-list**)

List of header patterns (case sensitive substrings in any charset) encoded in MIME format.

Constraints:

MIME encoded strings must comply with MIME definition.

filename *names*;

Entry condition - name of document.

The name is expected either in the FILENAME parameter of Content-Disposition header, or in the NAME parameter of Content-Type header.Only the last part of the file name (without path) is used for matching.

names (type: **str-set**)

from-quarantine [*tags*];

Entry condition (SMTP only) - mail sent by admin from quarantine.

This item is valid in smtp-proxy only. If omitted, any mail can match this ACL.

tags **(type: `str-set`, optional, default: *)**

Allowed X-Kernun-Quarantine-Tag header values

`add-virus-names`;

Add X-Kernun-Virus-Name header for each virus found.

[End of section `mail-doc-acl` description.]

SEE ALSO

configuration(7), acl(5), antivirus(5), common(5), mod-antispam(5), time(5), doctype-identification(7)

NAME

mod-match — format of mod-match component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **mod-match** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **mod-match** configuration directives:

data-match-action (name-usage obligatory)

Types of action done for matching data.

pass

Starts passing the received data. Continues executing other tests.

accept

Accepts the data and stops further scanning, regardless of any tests specified as REQUIRE. No more tests are executed.

deny

Does not pass any data, immediately stops processing and commands the proxy to report an error. No more tests are executed.

require

Requires matching data to appear in the data stream. If scanning reaches the end of data or maximum size of scanned data without a match, the module stops processing, does not pass any data and commands the proxy to report an error. Continues to following tests even after a match.

html-save

Interprets data as HTML form data inapplication/x-www-form-urlencoded encoding and saves the values of controls with matching names in a text file.

html-hash

Interprets data as HTML form data and saves hashes of values of controls with matching names in a database file.

html-alert

Interprets data as HTML form data and checks whether any controls have values corresponding to hashes stored by some HTML-HASH action. If the check succeeds, logs alert and optionally does not pass any data, stops processing and commands the proxy to report an error. No more tests are executed.

html-replace

Interprets data as HTML form data and replaces selected controls values with values computed from it using a lookup table.

html-replace-radius

Extension to HTML-REPLACE: password contains a combination of internal password and radius pasword. Radius authentication is performed before the lookup is done.

ITEMS AND SECTIONS

Configuration of **mod-match** library component consists of following prototypes:
* data-match *name* { ... }

Description:

data-match *name* **{**

max-size ... ;

init-match ... ;

max-match ... ;

step-size ... ;

step-match ... ;

* test ... ;

}

Matching data trasferred by a proxy. Data are passed further by the module only if they pass the specified TESTs.

Items & subsections:

max-size [*bytes***];**

The module performs the last matching attempt after reading this amount of data (or after it reaches end of data). The module then generates the final decision about the scanned data and terminates its operation. It does not process following data.

bytes **(type: uint32, optional, default: 512)**

init-match [*bytes***];**

The module performs PASS tests within this amount of received data. No more PASS tests are executed afterwards.

bytes (type: **uint32**, optional, default: **0**)

max-match [*bytes*];

The maximum length of the matching piece of data. The module keeps this amount of data in the input buffer.

bytes (type: **uint32**, optional, default: **0**)

step-size *bytes*;

Until the module's operation terminates, matching is repeated each time STEP-SIZE additional bytes are read or STEP-MATCH matches. If STEP-MATCH is not specified, the default value of STEP-SIZE is 1 byte. Otherwise, if STEP-SIZE is not set the default is to repeat matching according to STEP-MATCH.

bytes (type: **uint32**)

step-match *cond* [*back*];

Until the module's operation terminates, matching is repeated each time STEP-SIZE additional bytes are read or STEP-MATCH matches.

cond (type: **str-set**)
Matching condition.

back (type: **uint32**, optional, default: **0**)
Match against so many bytes back in data read earlier plus any newly received data.

test pass *match* [**match-data-mime**];

test accept *match* [**match-data-mime**];

test deny *match* [**match-data-mime**];

test require *match* [**match-data-mime**];

test html-save *match file* [**store-orig**];

test html-hash *match file*;

test html-alert *file* [**deny**];
test html-replace *controls file* [**report-controls** *report-controls*] [**keep-not-found**] [**replace-not-found** *replace-not-found*];
test html-replace-radius *controls file* [**report-controls** *report-controls*] [**keep-not-found**] [**replace-not-found** *replace-not-found*] **radius** *radius* **radius-delimiter** *radius-delimiter*;
A single matching test of transferred data.

<branching element> (type: **data-match-action**)
Action done if data match.

match (type: **str-set**)
Strings for matching.

match-data-mime (type: **key**, optional)
Match the detected MIME type instead of the actual data.

controls (type: **str-list**)

Selection of HTML form control names for replacement.

file (type: **str**)

A data file.

report-controls *report-controls* (type: **str-list**, optional, default: <NULL>)

Selection of HTML form control names for logging of values and passing to the executed programs.

store-orig (type: **key**, optional)

Store also the original value before hashing. This may be a security threat if HTML-HASH/HTML-ALERT is used for sensitive data, like passwords.

deny (type: **key**, optional)

Do not pass data if matched.

keep-not-found (type: **key**, optional)

If set and replacement values are not found in the lookup table, pass the selected HTML form controls unchanged. Otherwise, values of the selected controls are deleted.

replace-not-found *replace-not-found* (type: **str**, optional, default: <NULL>)

If set and replacement values are not found in the lookup table, replace values of the selected HTML form controls by this value.

radius *radius* (type: **name** of **radius-client**, see radius(5))

Radius client configuration to be used for authentication. The first form control (see element CONTROLS) is implicitly used as an username. The second form control (see element CONTROLS) is implicitly used as a combination of its internal value and radius password (see element RADIUS-DELIMITER.

radius-delimiter *radius-delimiter* (type: **str**)

A single character used as delimiter of the internal password and the radius password. Last occurence of the delimiter is used. If not present in the particular password, all the password text is interpreted as the radius password.

Constraints:

Only one of elements KEEP-NOT-FOUND and REPLACE-NOT-FOUND may be specified.

CONTROL must contain at least 2 items if HTML-REPLACE-RADIUS action is used..

[End of section `data-match` description.]

SEE ALSO

configuration(7), radius(5)

NAME

monitoring — format of monitoring component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **monitoring** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

ITEMS AND SECTIONS

Configuration of **monitoring** library component consists of following prototypes:
monitoring { ... }

Description:

monitoring {

disabled ... ;

comm-dir ... ;

interval ... ;

user ... ;

aproxy-user ... ;

data ... ;

}

Runtime monitoring of current proxy state.

Items & subsections:

disabled;

Disabling monitoring function.

Use this item to disable monitoring; otherwise it is enabled by default.

comm-dir [*path*];

Directory for monitoring communication files.

***path* (type: `str`, optional, default: "/tmp")**

interval [*seconds*];

Size of data sent during last INTERVAL seconds are used for computing current communication speed.

seconds **(type: uint16, optional, default: 5)**

user [*chars*];

Maximum number of characters from user name stored in the variable-length data area of a monitoring record.

chars **(type: uint16, optional, default: 8)**

aproxy-user [*chars*];

Maximum number of characters from authentication proxy user name stored in the variable-length data area of a monitoring record.

chars **(type: uint16, optional, default: 8)**

data [*chars* [**tail** *tail*]];

Maximum number of characters from data info (filename, URI etc.) stored in the variable-length data area of a monitoring record.

chars **(type: uint16, optional, default: 180)**

tail *tail* **(type: uint16, optional, default: 60)**

long data info is stored as <CHARS-3-TAIL initial characters>...<TAIL trailing characters>

Constraints:

CHARS must be greater than TAIL plus 3.

[End of section `monitoring` description.]

SEE ALSO

configuration(7), monitoring(7)

NAME

nameserver — format of nameserver component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **nameserver** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **nameserver** configuration directives:

yes-no (see **common**(5))

genesis (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

listen-on-sock (see **listen-on**(5))

forward (name-usage obligatory)

only
: Only query the forwarders.

first
: First query the forwarders. If that doesn't answer the question, resolve recursively from root name servers.

disable
: Do not use forwarders. Resolve from root servers only.

ITEMS AND SECTIONS

Configuration of **nameserver** library component consists of following prototypes:
* nameserver *name* { ... }

Description:

nameserver *name* **{**

phase ... ;

* tag ... ;

use-ipv4-only ... ;

listen-on { ... }

forward ... ;

* forwarder ... ;

* from ... ;

dnssec { ... }

send-cookie ... ;

* option ... ;

* raw ... ;

* zone *name* { ... }

}

Domain Name System Server Configuration.

This section allows to define a configuration for a simple nameserver serving local clients as a caching forwarder (typically forwarding to the DNS-PROXY) and an authoritative nameserver for local zones.

For configuration attributes details, see named.conf(5).

Constraints:

Addresses to listen on must be specified.

FORWARDER must be specified for forward mode ONLY or FIRST.

FORWARDER must not be specified for forward mode DISABLE.

Items & subsections:

phase [*number***];**

Application Startup Phase.

***number* (type: `uint8`, optional, default: 30)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

`use-ipv4-only;`

IPv4 only mode of nameserver.

This item affects usage of the -4 option of the named daemon.

If used, the daemon is started with the -4 option.

If omitted, the daemon is started without this option if at least one IPv6 interface is configured in the system.

`listen-on {`

* socket ... ;

}

The `listen-on` section is derived from `listen-on` section prototype.
For detail description of it, see listen-on(5).
Changes to the `listen-on` section:

Item `non-transparent` used as `socket`.

Item `transparent` is not valid.

At least one address to listen on must be specified.

Item `socket` (see listen-on(5))

Element `port` is optional, default: 53.

Element `proto` is optional, default: tcp-udp.

`forward` [*mode*];

mode **(type: `forward`, optional, default: only)**

Whether use the forwarders, resolve from root servers, or both.

`forwarder` [`static`] *addr*;

`forwarder dynamic;`

Next-hop nameserver address.

Usually the forwarder will be a local dns-proxy.

DYNAMIC mode means getting the address via the DHCP.

<branching element> (type: `genesis`, optional, default: static)

addr **(type: `sock`)**

`from` *clients*;

Valid clients definition (ALLOW-QUERY).

clients **(type: `net-set`)**

Constraints:

Regexps and discontiguous masks not allowed in nameserver lists.

dnssec {

managed-keys { ... }

validate ... ;

}

DNSSEC configuration.

DNSSEC support is always switched on.

Items & subsections:

managed-keys {

directory ... ;

* initial-key ... ;

}

DNSSEC validation keys configuration.

Items & subsections:

directory [*path*];

Managed keys directory.

path **(type: str, optional, default: "/var/lib/named/dyn/")**

initial-key *domain flags protocol algorithm key*;

Initial validation key.

domain **(type: str)**

Domain name.

flags **(type: uint32)**

protocol **(type: uint32)**

algorithm **(type: uint32)**

key **(type: str)**

[End of section `nameserver.dnssec.managed-keys` description.]

validate [*val*];

Whether to validate DNSSEC.

val **(type: yes-no, optional, default: no)**

[End of section `nameserver.dnssec` description.]

send-cookie;

DNS cookie sending configuration.

option *line*;

Raw lines into options definition.

line **(type: str)**

raw *line*;

Raw lines into global definition.

line **(type: str)**

zone *name* {

name ... ;

reverse ... ;

* master-server ... ;

* raw ... ;

generate { ... }

}

Nameserver zone definition.

Zone name is defined either by the NAME item (for a regular one) or by the REVERSE item (for a reverse one).

Zone type (master/slave) is defined by using of MASTER-SERVER item(s). If not used, this zone is MASTER, otherwise SLAVE.

Constraints:

Exactly one of NAME and REVERSE must be specified.

MASTER-SERVER and GENERATE are mutually exclusive.

Items & subsections:

name *zone*;

Zone domain name.

zone **(type: str)**

reverse *zone*;

Reverse zone specification.

zone **(type: net)**

master-server *addr*;

Define master server for domain.

addr **(type: sock)**

raw *line*;

Raw lines into zone definition.

line **(type: str)**

generate {

* raw ... ;

}

Zone data generation parameters.

If used, KGB will generate zone database file from HOST-TABLE. Otherwise, zone file is expected to be created outside the CML and will be used without modifications.

Items & subsections:

`raw` *line*;

Raw lines into zone database file.

line **(type: `str`)**

[End of section `nameserver.zone.generate` description.]

[End of section `nameserver.zone` description.]

[End of section `nameserver` description.]

SEE ALSO

configuration(7), common(5), listen-on(5), named.conf(5)

NAME

netio — format of netio component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **netio** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

ITEMS AND SECTIONS

Configuration of **netio** library component consists of following prototypes:

```
  sock-opt { ... }
* ip-tos-from ... ;
  ip-tos-received ... ;
  ip-tos-to { ... }
  ip-tos-to-client { ... }
  ip-tos-to-server { ... }
```

Description:

sock-opt {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Network I/O Module Attributes.

Items & subsections:

conn-timeout [*seconds*];

Maximum connection time.

seconds **(type: uint16, optional, default: 75)**

Constraints:

Timeout must not be set to zero.

recv-timeout [*seconds*];

Maximum socket read time.

seconds **(type: uint16, optional, default: 120)**

Constraints:

Timeout must not be set to zero.

recv-bufsize [*bytes*];

Input buffer size.

bytes **(type: uint32, optional, default: 16384)**

send-timeout [*seconds*];

Maximum socket write wait time.

seconds **(type: uint16, optional, default: 120)**

Constraints:

Timeout must not be set to zero.

close-timeout [*seconds*];

Maximum time for waiting for the peer to close its half of the TCP connection after the proxy has closed its half. If set to zero, the peer gets TCP reset whenever it does not close the connection earlier than the proxy.

seconds **(type: uint16, optional, default: 5)**

send-bufsize [*bytes*];

Output buffer size.

bytes **(type: uint32, optional, default: 16384)**

log-limit *bytes*;

Per block data limit for full log.

bytes **(type: uint32)**

Default for this value is set by the DATA-LIMIT item in the proxy LOG section.

[End of section `sock-opt` description.]

ip-tos-from *val*;

Testing an IP TOS value of received packets.

val **(type: uint8-set)**

ip-tos-received [**mask** *mask*] [**add** *add*];

Send a possibly modified received IP TOS value. The value used for for sending will be (RECEIVED & MASK) | ADD.

mask *mask* **(type: uint8, optional, default: 255)**

A mask of bits used from the received value.

add *add* **(type: uint8, optional, default: 0)**

A mask of bits added to the value to be sent.

ip-tos-to {

fixed ... ;

received ... ;

other ... ;

}

Set an IP TOS value for sent packets.

Items & subsections:

fixed [*val***];**

Use a fixed TOS value.

val **(type: uint8, optional, default: 0)**

received [mask *mask***] [add** *add***];**

Use a value received from this connection.

mask *mask* **(type: uint8, optional, default: 255)**

A mask of bits used from the received value.

add *add* **(type: uint8, optional, default: 0)**

A mask of bits added to the value to be sent.

other [mask *mask***] [add** *add***];**

Use a value received from the other connection.

mask *mask* **(type: uint8, optional, default: 255)**

A mask of bits used from the received value.

add *add* **(type: uint8, optional, default: 0)**

A mask of bits added to the value to be sent.

[End of section `ip-tos-to` description.]

ip-tos-to-client {

fixed ... ;

received ... ;

other ... ;

}

Set an IP TOS value for packets sent to the client. If OTHER is specified, it will be set only after a connection to the server is established. Until then, the value specified by FIXED or RECEIVED (from the client) will be used.

The `ip-tos-to-client` section is derived from `ip-tos-to` section prototype. For detail description of it, see above.
Changes to the `ip-tos-to-client` section:

Only one of FIXED and RECEIVED may be specified.

`ip-tos-to-server` {

fixed ... ;

received ... ;

other ... ;

}

Set an IP TOS value for packets sent to the server.

The `ip-tos-to-server` section is derived from `ip-tos-to` section prototype. For detail description of it, see above.
Changes to the `ip-tos-to-server` section:

Only one of FIXED, RECEIVED, and OTHER may be specified.

SEE ALSO

configuration(7), netio(7)

NAME

nls — format of nls component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **nls** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **nls** configuration directives:

`language` (see **common**(5))

`nls` (see **common**(5))

ITEMS AND SECTIONS

Configuration of **nls** library component consists of following prototypes:

nls ... ;
language ... ;

Description:

`nls` *code*;

Language and charset of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

code (**type: `nls`**)

`language` *lang*;

Language of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

lang (**type: `language`**)

SEE ALSO

configuration(7), common(5)

NAME

ntp — format of ntp component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ntp** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **ntp** configuration directives:

destination (see **common**(5))

ntp-rest-flag (name-usage obligatory)

NTP configuration RESTRICT flags.

ignore

kod

limited

lowpriotrap

nomodify

noquery

nopeer

noserve

notrap

notrust

ntpport

version

ITEMS AND SECTIONS

Configuration of **ntp** library component consists of following prototypes:

ntp { ... }

Description:

ntp {

phase ... ;

* tag ... ;

cfg-resolution ... ;

drift-file ... ;

* peer ... ;

* server ... ;

* clock ... ;

* restrict ... ;

}

NTP daemon definition.

Most configuration directives are synonyms of NTP.CONF ones. See ntp.conf(5) for details.

Items & subsections:

phase [*number*];

Application Startup Phase.

number **(type: uint8, optional, default: 70)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: str)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

cfg-resolution [*max-addrs* [*min-ttl* [*def-ttl* [*max-ttl* [*hosts-ttl* [*pool-dir*]]]]];

Attributes for resolution of domain names in configuration.

max-addrs **(type: uint8, optional, default: 10)**

Maximum of addresses per a single domain name.

min-ttl **(type: uint32, optional, default: 10)**

Minimum TTL accepted, used instead of too small TTL values (e.g. 0).

def-ttl **(type: uint32, optional, default: 1m)**

Default TTL used in case of unsuccessful DNS resolution.

max-ttl (type: **uint32**, optional, default: **1d**)

Maximum TTL accepted, used instead of large TTL values.

hosts-ttl (type: **uint32**, optional, default: **1d**)

TTL used for names in /etc/hosts.

pool-dir (type: **str**, optional, default: **"/tmp"**)

Directory for temporary files used to share results.

drift-file *path*;

NTP daemon drift-file full name.

path (type: **str**)

Constraints:

Path must be absolute and must not contain punctuation chars.

peer *machine*;

Host for peer-to-peer synchronization.

machine (type: **host**)

server *machine*;

Host for client-to-server synchronization.

machine (type: **host**)

clock *type num stratum*;

Device for local synchronization.

type (type: **uint8**)

Clock type.

num (type: **uint8**)

Unit number.

stratum (type: **uint8**)

Stratum number.

Constraints:

Unit number must be at most 3.

Stratum number must be at most 15.

restrict host *host* [*flags*];

restrict net *net* [*flags*];

restrict default [*flags*];

Host-based service restrictions.

<branching element> (type: **destination**)

host (type: **host**)

net (type: **net**)

flags (type: **ntp-rest-flag-list**, optional, default: **{}**)

[End of section `ntp` description.]

SEE ALSO

configuration(7), common(5), ntp.conf(5)

NAME

openvpn — format of openvpn component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **openvpn** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **openvpn** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

week-day (see **time**(5))

month (see **time**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

ovpn-protocols (name-usage obligatory)

udp
: Use UDP protocol. UDP must be specified on both peers.

tcp-client
: Use TCP protocol, be the TCP client (the other peer must use tcp-server). TCP client will attempt to connect, and if that fails, will sleep for a period and try it again.

tcp-server
: Use TCP protocol, be the TCP server (the other peer must use tcp-client). TCP server will wait indefinitely for an incomming connection.

udp6
: Use UDP protocol over IPv6. UDP6 must be specified on both peers.

tcp6-client

Use TCP protocol over IPv6, be the TCP client (the other peer must use tcp6-server). TCP client will attempt to connect, and if that fails, will sleep for a period and try it again.

tcp6-server

Use TCP protocol over IPv6, be the TCP server (the other peer must use tcp6-client). TCP server will wait indefinitely for an incomming connection.

ovpn-remote-proto (name-usage obligatory)

udp

Use UDP protocol

tcp

Use TCP protocol

udp6

Use UDP protocol over IPv6

tcp6

Use TCP protocol over IPv6

implicit

Use the protocol specified by the OPENVPN.PROTO item

ovpn-comp-lzo-mode (name-usage obligatory)

yes

no

adaptive

none

The comp-lzo directive is ommited in the openvpn configuration.

ovpn-cert-types (name-usage obligatory)

client

server

ovpn-cipher-algs (name-usage obligatory)

DES-CBC

64 bit default key (fixed)

RC2-CBC

128 bit default key (variable)

DES-EDE-CBC

128 bit default key (fixed)

DES-EDE3-CBC

192 bit default key (fixed)

DESX-CBC

192 bit default key (fixed)

BF-CBC

128 bit default key (variable)

RC2-40-CBC

40 bit default key (variable)

CAST5-CBC

128 bit default key (variable)

RC5-CBC

128 bit default key (variable)

RC2-64-CBC

64 bit default key (variable)

AES-128-CBC

128 bit default key (fixed)

AES-192-CBC

192 bit default key (fixed)

AES-256-CBC

256 bit default key (fixed)

AES-128-GCM

128 bit key, 128 bit block, TLS client/server mode only

AES-192-GCM

192 bit key, 128 bit block, TLS client/server mode only

AES-256-GCM

256 bit key, 128 bit block, TLS client/server mode only

CAMELLIA-128-CBC

128 bit default key (fixed)

CAMELLIA-192-CBC

192 bit default key (fixed)

CAMELLIA-256-CBC

256 bit default key (fixed)

none

no encryption

ovpn-redirect-gateway-flags (name-usage optional)

local (0)

Add the local flag if both OpenVPN servers are directly connected via a common subnet, such as with wireless. The local flag will cause step 1 above to be omitted.

def1 (1)

Use this flag to override the default gateway by using 0.0.0.0/1 and 128.0.0.0/1 rather than 0.0.0.0/0. This has the benefit of overriding but not wiping out the original default gateway.

bypass-dhcp (2)

Add a direct route to the DHCP server (if it is non-local) which bypasses the tunnel (Available on Windows clients, may not be available on non-Windows clients).

bypass-dns (3)

Add a direct route to the DNS server(s) (if they are non-local) which bypasses the tunnel (Available on Windows clients, may not be available on non-Windows clients).

ovpn-dhcp-option (name-usage obligatory)

DOMAIN

Set connection-specific DNS suffix

DNS

Set domain name server address

WINS

Set WINS server address

NBDD

Set NBDD server address

NTP

Set NTP serveer address

NBT

Set NetBIOS over TCP/IP Node type

NBS

Set NetBIOS over TCP/IP Scope

DISABLE-NBT

Disable Netbios over TCP/IP

ovpn-topology (name-usage obligatory)

net30

subnet

ovpn-local-scope (name-usage obligatory)

any

Accept connections on all interfaces

addr

Accept connections on the particular IP address

tls-mat-variants (name-usage obligatory)

pkcs12

Cryptographic material provided in a single pkcs12 file

ca-cert-key

Cryptographic material provided in three separated files in .pem format

ITEMS AND SECTIONS

Configuration of **openvpn** library component consists of following prototypes:

ovpn-push { ... }
* ovpn-ccd *name* { ... }
ovpn-summary { ... }
* openvpn *name* { ... }

Description:

ovpn-push {

* route ... ;

* route-ipv6 ... ;

redirect-gateway ... ;

redirect-gateway-ipv6 ... ;

* dhcp-option ... ;

block-outside-dns ... ;

* raw ... ;

}

Configuration options to be pushed to the client for remote execution

Items & subsections:

route ***network*** **[*gw*];**

Add route to routing table after connection is established. Multiple routes can be specified. Routes will be automatically torn down in reverse order prior to TUN/TAP device close. Use special value of [0.0.0.0] as gw for specifying the remote VPN endpoint (from the perspective of the client, see vpn gateway in openvpn(8).

network **(type: net)**

Route destination.

gw **(type: host, optional, default: [0.0.0.0])**

Router IP address. Special value [0.0.0.0] can be used for the remote VPN endpoint (from the client's perspective).

route-ipv6 *network* **[***gw***];**

Add IPV6 route to routing table after connection is established. Multiple routes can be specified. Routes will be automatically torn down in reverse order prior to TUN/TAP device close. Use special value of [::] as gw for specifying the remote VPN endpoint (from the perspective of the client, see vpn gateway in openvpn(8).

network **(type: net)**

Route destination.

gw **(type: host, optional, default: [::])**

Router IP address. Special value [::] can be used for the remote VPN endpoint (from the client's perspective).

redirect-gateway *flags*;

Automatically execute routing commands to cause all outgoing IP traffic to be redirected over the VPN. This option performs three steps: (1) Create a static route for the remote address which forwards to the pre-existing default gateway. This is done so that (3) will not create a routing loop. (2) Delete the default gateway route. (3) Set the new default gateway to be the VPN endpoint address

When the tunnel is torn down, all of the above steps are reversed so that the original default route is restored.

Using the def1 flag is highly recommended.

flags **(type: ovpn-redirect-gateway-flags-list)**

redirect-gateway-ipv6;

Automatically execute routing commands to cause all outgoing IPv6 traffic to be redirected over the VPN. The default route is overriden by specifying route for ::/1 and 8000::/1.

dhcp-option 0;

dhcp-option domain *domain*;

dhcp-option dns *dns*;

dhcp-option wins *wins*;

dhcp-option nbdd *nbdd*;

dhcp-option ntp *ntp*;

dhcp-option nbt *nbt*;

dhcp-option nbs *nbs*;

dhcp-option disable-nbt;

Set extended TAP-Win32 TCP/IP properties. This option can be used to set additional TCP/IP properties on the TAP-Win32 adapter, and is particularly useful for configuring an OpenVPN client to access a Samba server across the VPN.

Note that if dhcp-option is pushed via push to a non-windows client, the option will be saved in the client's environment before the up script is called, under the name "foreign option {n}".

<branching element> (type: ovpn-dhcp-option)

***domain* (type: str)**

Set Connection-specific DNS Suffix.

***dns* (type: addr)**

Set primary domain name server address. Repeat this option to set secondary DNS server addresses.

***wins* (type: addr)**

Set primary WINS server address (NetBIOS over TCP/IP Name Server). Repeat this option to set secondary WINS server addresses.

***nbdd* (type: addr)**

Set primary NBDD server address (NetBIOS over TCP/IP Datagram Distribution Server) Repeat this option to set secondary NBDD server addresses.

***ntp* (type: addr)**

Set primary NTP server address (Network Time Protocol). Repeat this option to set secondary NTP server addresses.

***nbt* (type: uint8)**

Set NetBIOS over TCP/IP Node type. Possible options: 1 = b-node (broadcasts), 2 = p-node (point-to-point name queries to a WINS server), 4 = m-node (broadcast then query name server), and 8 = h-node (query name server, then broadcast).

***nbs* (type: str)**

Set NetBIOS over TCP/IP Scope. A NetBIOS Scope ID provides an extended naming service for the NetBIOS over TCP/IP (Known as NBT) module. The primary purpose of a NetBIOS scope ID is to isolate NetBIOS traffic on a single network to only those nodes with the same NetBIOS scope ID. The NetBIOS scope ID is a character string that is appended to the NetBIOS name. The NetBIOS scope ID on two hosts must match, or the two hosts will not be able to communicate. The NetBIOS Scope ID also allows computers to use the same computer name, as they have different scope IDs. The Scope ID becomes a part of the NetBIOS name, making the name unique. (This description of NetBIOS scopes courtesy of NeonSurge@abyss.com)

block-outside-dns;

Block DNS servers on other network adapters to prevent DNS leaks.

raw *row*;

A raw item to be put to the OpenVPN configuration file exactly as given as the "row" element.

***row* (type: str)**

[End of section `ovpn-push` description.]

ovpn-ccd *name* **{**

ifconfig-push ... ;

ifconfig-ipv6-push ... ;

disable ... ;

push { ... }

push-reset ... ;

* iroute ... ;

* iroute-ipv6 ... ;

* route ... ;

* schedule ... ;

* raw ... ;

cn ... ;

}

Client-configuration-directives.

Set of the custom configuration directives to be used for the particular client. After the client has been authenticated, the ccd section with the same name as the client's X509 common name is used.

Constraints:

ROUTE can only be used with IFCONFIG-PUSH.

Items & subsections:

ifconfig-push *local*;

Push virtual local IP endpoint for client tunnel, overriding the ifconfig-pool dynamic allocation.

Note that the parameter local is from the perspective of the client, not the server.

For a tun interface, the remote address is constructed from the local IP address. For a tap interface, the netmask is taken from ^interface.ipv4.net. (Note that the eventual netmask is ignored for the local element).

local **(type: addr)**

ifconfig-ipv6-push *local*;

Push virtual local IPv6 endpoint for client tunnel, overriding the ifconfig-ipv6-pool dynamic allocation.

Note that the parameter local is from the perspective of the client, not the server.

For a tun interface, the remote address is constructed from the local IP address. For a tap interface, the netmask is taken from ^interface.ipv4.net. (Note that the eventual netmask is ignored for the local element).

local **(type: addr)**

disable;

Disable a particular client (based on the common name) from connecting. Don't use this option to disable a client due to key or password compromise. Use a CRL (certificate revocation list) instead (see the crl-verify option).

push {

* route ... ;

* route-ipv6 ... ;

redirect-gateway ... ;

redirect-gateway-ipv6 ... ;

* dhcp-option ... ;

block-outside-dns ... ;

* raw ... ;

}

The push section is derived from ovpn-push section prototype. For detail description of it, see above.

push-reset;

Don't inherit the global push list for a specific client instance.

This option will ignore push options at the global config file level.

iroute ***network*****;**

Generate an internal route to a specific client.

This directive can be used to route a fixed subnet from the server to a particular client, regardless of where the client is connecting from. Remember that you must also add the route to the system routing table as well. The reason why two routes are needed is that the "system route" routes the packet fromthe kernel to OpenVPN. Once in OpenVPN, the iroute directive routes to the specific client.

The iroute directive also has an important interaction with push "route ...". iroute essentially defines a subnet which is owned by a particular client (we will call this client A). If you would like other clients to be able to reach A's subnet, you can use push "route ..." together with client-to-client to effect this. In order for all clients to see A's subnet, OpenVPN must push this route to all clients EXCEPT for A, since the subnet is already owned by A. OpenVPN accomplishes this by not not pushing a route to a client if it matches one of the client's iroutes.

network **(type: net)**

iroute-ipv6 ***ipv6addr*****;**

For ccd per-client static IPv6 route configuration, see IROUTE for more details how to setup and use this, and how IROUTE and ROUTE interact.

ipv6addr **(type: net)**

route ***network*****;**

The given network should be routed through this client as a gateway. The route is added as an iroute to the ccd section, as a route for the openvpn instance and as the system route.

network **(type: net)**
Route destination

schedule *perm* **[day** *day***] [month** *month***] [***wday* **[***hhmm***]];**

Schedule the permissions. The order of this repeatable item is significant. The first matching schedule item is used. Depending on its perm element, the connection is either enabled or disabled. The permissions are checked either when the client is actually connecting as well as it is periodically checked in order to disconnect the clietns whose permission would eventually expire.

perm **(type: enabling)**

day *day* **(type: uint8-set, optional, default: *)**
day of month (1 - 31)

month *month* **(type: month-set, optional, default: *)**
month (Jan - Dec or 1 - 12)

wday **(type: week-day-set, optional, default: *)**
week-day (Sun - Sat or 0 - 6)

hhmm **(type: time-set, optional, default: *)**
time (in form hhmm)

raw *row***;**

An raw item to be put to the OpenVPN configuration file exactly as given as the "row" element.

row **(type: str)**

cn *cn***;**

Entry condition: Common name (CN).

If given, common name of the client is compared to the given string. Else, the name of the ccd section is compared to the CN. In that case, spaces (' ') and dots ('.') in the CN are substituted by underscore ('_').

cn **(type: str)**

[End of section `ovpn-ccd` description.]

ovpn-summary {

top-clients ... ;

top-users ... ;

activity-report { ... }

}

The `ovpn-summary` section is derived from `summary` section prototype. For detail description of it, see application(5).
Changes to the `ovpn-summary` section:

Item `top-groups` is not valid.

Item `top-servers` is not valid.

Item `top-categories` is not valid.

Item `top-senders` is not valid.

Item `top-recipients` is not valid.

Item `top-mime-types` is not valid.

Item `top-qnames` is not valid.

Item `top-qtypes` is not valid.

Item `top-callers` is not valid.

Item `top-receivers` is not valid.

Item `top-sids` is not valid.

Item `top-server-ports` is not valid.

Item `spam-threshold` is not valid.

Item `top-src-ips` is not valid.

Item `top-dst-ips` is not valid.

Item `top-rules` is not valid.

`openvpn` *name* {

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

interface ... ;

topology ... ;

local ... ;

nobind ... ;

user ... ;

group ... ;

persist-tun ... ;

persist-key ... ;

log-debug { ... }

log-stats { ... }

mute ... ;

ping-timer-rem ... ;

keepalive ... ;

proto ... ;

tls-mat ... ;

dh ... ;

secret ... ;

crl-verify ... ;

server ... ;

max-clients ... ;

duplicate-cn ... ;

client-to-client ... ;

ccd-exclusive ... ;

mlock ... ;

float ... ;

push { ... }

ifconfig-pool ... ;

ifconfig-ipv6-pool ... ;

tls-server ... ;

tls-client ... ;

tls-auth ... ;

* remote ... ;

remote-random ... ;

comp-lzo ... ;

verify-x509-name ... ;

remote-cert-ku ... ;

remote-cert-eku ... ;

remote-cert-tls ... ;

cipher ... ;

data-ciphers ... ;

data-ciphers-fallback ... ;

client ... ;

pull ... ;

route-nopull ... ;

no-ifconfig-noexec ... ;

ifconfig-pool-persist ... ;

client-connect ... ;

client-connect-socket ... ;

* ccd *name* { ... }

* raw ... ;

phase ... ;

* tag ... ;

socket-root ... ;

fast-io ... ;

}

OpenVPN configuration.

For configuration attributes details, see openvpn(8).

Constraints:

INTERFACE must be specified.

ROUTE-NOPULL must be specified if CLIENT or PULL is used.

CLIENT, TLS-CLIENT, SERVER, TLS-SERVER and SECRET are mutually exclusive.

Cryptographic material (TLS-MAT) required for any of SERVER, TLS-SERVER, CLIENT, TLS-CLIENT.

tls options (DH, TLS-MAT, PKCS12, VERIFY-X509-NAME, CRL-VERIFY, NS-CERT-TYPE, REMOTE-CERT-KU, REMOTE-CERT-EKU) can only be specified in tls mode (SERVER, TLS-SERVER, CLIENT, TLS-CLIENT).

DH required for SERVER or TLS-SERVER.

Each of IFCONFIG-POOL, CLIENT-CONNECT, CCD, CCD-EXCLUSIVE, CLIENT-TO-CLIENT, DUPLICATE-CN requires SERVER.

CCD-EXCLUSIVE requires some CCD section.

SERVER may be used only with proto UDP or TCP-SERVER.

PULL can only be used with CLIENT, TLS-CLIENT or TLS-SERVER.

REMOTE ad SERVER are mutually exclusive.

REMOTE must be used in proto mode TCP-CLIENT.

proto mode TCP-SERVER allows at most one REMOTE.

NOBIND can only be used with REMOTE and without LOCAL.

Items PROTO, LOCAL and REMOTE must respect each other's address family.

Item SERVER is mutually exclusive with items IFCONFIG-POOL and IFCONFIG-IPV6-POOL.

Cipher AES-GCM is not supported in SECRET mode.

FAST-IO mode is allowed only in UDP.

Items & subsections:

stats-daily {

top-clients ... ;

top-users ... ;

activity-report { ... }

}

The stats-daily section is derived from ovpn-summary section prototype. For detail description of it, see above.

stats-weekly {

top-clients ... ;

top-users ... ;

activity-report { ... }

}

The stats-weekly section is derived from ovpn-summary section prototype. For detail description of it, see above.

stats-monthly {

top-clients ... ;

top-users ... ;

activity-report { ... }

}

The stats-monthly section is derived from ovpn-summary section prototype. For detail description of it, see above.

interface *ifname*;

Interface to be used for the virtual network.

The interface must be of type TUN or TAP. The interface.ipv4.addr specifies the local IP addressin the VPN and the network range of the VPN. For TUN, the ipv4.dest address specifies the address of the peer in the tunnel.

ifname **(type: name of interface, see** interface**(5))**

topology *topo*;

OpenVPN network topology.

If omitted, default topology (net30) is used.

topo **(type: ovpn-topology)**

local any [*port***];**

local [addr] *addr* **[***port***];**

Local IP address and port for bind. If specified, OpenVPN will bind to this address only. If unspecified, OpenVPN will bind to all interfaces, using the default port.

<branching element> (type: ovpn-local-scope, optional, default: addr)

addr **(type: addr)**

IP address to listen on

port **(type: port, optional, default: 1194)**

Port to listen on

nobind;

Do not bind to local address and port. This option is only suitable for peers which will be initiating connections by using remote intem.

user [*user***];**

Change the user ID of the OpenVPN process to user after initialization, dropping privileges in the process. This option is useful to protect the system in the event that some hostile party was able to gain control of an OpenVPN session. Though OpenVPN's security features make this unlikely, it is provided as a second line of defense.

user **(type: str, optional, default: "kernun")**

group [*group***];**

Change the group ID of the OpenVPN process to group after initialization, dropping privileges in the process. This option is useful to protect the system in the event that some hostile party was able to gain control of an OpenVPN session. Though OpenVPN's security features make this unlikely, it is provided as a second line of defense.

group **(type: str, optional, default: "kernun")**

persist-tun [*persist-tun***];**

Don't close and reopen TUN/TAP device or run up/down scripts across SIGUSR1 or ping-restart restarts.

SIGUSR1 is a restart signal similar to SIGHUP, but which offers finer-grained control over reset options.

persist-tun **(type: yes-no, optional, default: yes)**

persist-key [*persist-key***];**

Don't re-read key files across SIGUSR1 or ping-restart.

This option can be combined with user item to allow restarts triggered by the SIGUSR1 signal. Normally if you drop root privileges in OpenVPN, the daemon cannot be restarted since it will now be unable to reread protected key files.

This option solves the problem by persisting keys across SIGUSR1 resets, so they don't need to be re-read.

persist-key **(type: yes-no, optional, default: yes)**

log-debug {

level ... ;

mem-level ... ;

facility ... ;

file ... ;

rotate ... ;
mem-file ... ;
syslog-failure ... ;
data-limit ... ;
dump-hold-time ... ;
}

The `log-debug` section is derived from `log` section prototype. For detail description of it, see log(5).

`log-stats` {
level ... ;
mem-level ... ;
facility ... ;
file ... ;
rotate ... ;
mem-file ... ;
syslog-failure ... ;
data-limit ... ;
dump-hold-time ... ;
}

The `log-stats` section is derived from `log` section prototype. For detail description of it, see log(5).

`mute` [*n*];

Log at most n consecutive messages in the same category. This is useful to limit repetitive logging of similar message types.

n **(type: `uint16`, optional, default: 10)**

`ping-timer-rem` [*ping-timer-rem*];

Run the ping-exit /ping-restart timer only if we have a remote address. Use this option if you are starting the daemon in listen mode (i.e. without an explicit remote peer), and you don't want to start clocking timeouts until a remote peer connects.

ping-timer-rem **(type: `yes-no`, optional, default: yes)**

`keepalive` [*ping* [*ping-restart*]];

A helper directive designed to simplify the expression of ping and ping-restart. See openvpn(8) for details on ping, ping-restart and keepalive directives.

ping **(type: `uint16`, optional, default: 10)**

ping-restart **(type: `uint16`, optional, default: 60)**

`proto` [*proto*];

Specify the protocol to be used for communicating with remote host.

OpenVPN is designed to operate optimally over UDP, but TCP capability is provided for situations where UDP cannot be used. In comparison with UDP, TCP will usually be somewhat less efficient and less robust when used over unreliable or congested networks.

There are certain cases, however, where using TCP may be advantageous from a security and robustness perspective, such as tunneling non-IP or application-level UDP protocols, or tunneling protocols which don't possess a built-in reliability layer.

proto **(type: ovpn-protocols, optional, default: udp)**

tls-mat pkcs12 *pkcs12*;

tls-mat [ca-cert-key] *ca cert key*;

Specify the cryptographic material: root CA certificate, the local peer's certificate (signed by the CA), and the local peer's private key. It can be provided either as a single PKCS#12 file or as 3 files in .pem format.

<branching element> (type: tls-mat-variants, optional, default: ca-cert-key)

pkcs12 **(type: name of shared-file,** see common**(5))**

PKCS #12 file

ca **(type: name of shared-file,** see common**(5))**

Certificate authority

cert **(type: name of shared-file,** see common**(5))**

Local peer's certificate

key **(type: name of shared-file,** see common**(5))**

Local peer's private key

dh *dh*;

File containing Diffie Hellman parameters in .pem format Diffie Hellman parameters may be considered public.

dh **(type: name of shared-file,** see common**(5))**

secret *secret*;

Static Key encryption mode (non-TLS). The same pre-shared secret file is used by both peers.

secret **(type: name of shared-file,** see common**(5))**

crl-verify *crl*;

Check peer certificate against the file CRL (certificate revocation list) in PEM format. A CRL is used when a particular key is compromised but when the overall PKI is still intact.

crl **(type: name of shared-file,** see common**(5))**

server;

This directive will set up an OpenVPN server. It will allocate addresses to clients out of network/netmask specified in the referenced INTERFACE section. The server itself will take the first host address of the given network (which should be specified as the

interface.ipv4.addr) for use as the server-side endpoint of the local TUN/TAP interface. For TUN, the next address (second host address of the given network) should be used as interface.ipv4.dest.

max-clients *n*;

Limit server to a maximum of n concurrent clients.

n (type: **uint16**)

duplicate-cn;

Allow multiple clients with the same common name to concurrently connect. In the absence of this option, OpenVPN will disconnect a client instance upon connection of a new client having the same common name.

client-to-client;

Because the OpenVPN server mode handles multiple clients through a single tun or tap interface, it is effectively a router. The client-to-client flag tells OpenVPN to internally route client-to-client traffic rather than pushing all client-originating traffic to the TUN/TAP interface.

When this option is used, each client will "see" the other clients which are currently connected. Otherwise, each client will only see the server. Don't use this option if you want to firewall tunnel traffic using custom, per-client rules.

ccd-exclusive;

Require, as a condition of authentication, that a connecting client has an explicit ccd section.

mlock;

Disable paging by calling the POSIX mlockall function.

Using this option ensures that key material and tunnel data are never written to disk due to virtual memory paging operations which occur under most modern operating systems. It ensures that even if an attacker was able to crack the box running OpenVPN, he would not be able to scan the system swap file to recover previously used ephemeral keys, which are used for a period of time, and then are discarded.

The downside of using mlock is that it will reduce the amount of physical memory available to other applications.

float;

Allow remote peer to change its IP address and/or port number, such as due to DHCP (this is the default if remote is not used). float when specified with remote allows an OpenVPN session to initially connect to a peer at a known address, however if packets arrive from a new address and pass all authentication tests, the new address will take control of the session. This is useful when you are connecting to a peer which holds a dynamic address such as a dial-in user or DHCP client.

Essentially, float tells OpenVPN to accept authenticated packets from any address, not only the address which was specified in the remote option.

push {

* route ... ;

* route-ipv6 ... ;

redirect-gateway ... ;

redirect-gateway-ipv6 ... ;

* dhcp-option ... ;

block-outside-dns ... ;

* raw ... ;

}

The push section is derived from ovpn-push section prototype. For detail description of it, see above.

ifconfig-pool *start-ip end-ip* **[warn** *warn***];**

Set aside a pool of subnets to be dynamically allocated to connecting clients, similar to a DHCP server. For tun-style tunnels, each client will be given a /30 subnet (for interoperability with Windows clients). For tap-style tunnels, individual addresses will be allocated, and the netmask parameter will also be pushed to clients. The netmask value is taken from ^interface.ipv4.addr. (Note that the eventual netmask is ignored for both start-IP and end ip elements).

start-ip **(type: addr)**

end-ip **(type: addr)**

warn *warn* **(type: yes-no, optional, default: yes)**

Warn on conflicts between ifconfig-pool and ccd.ifconfig-push items.

ifconfig-ipv6-pool *ipv6addr* **[warn** *warn***];**

Specify an IPv6 address pool for dynamic assignment to clients. The pool starts at ipv6addr and increments by +1 for every new client (linear mode). The /bits setting controls the size of the pool.

ipv6addr **(type: addr)**

warn *warn* **(type: yes-no, optional, default: yes)**

Warn on conflicts between ifconfig-pool and ccd.ifconfig-push items.

tls-server;

Enable TLS and assume server role during TLS handshake. Note that OpenVPN is designed as a peer-to-peer application. The designation of client or server is only for the purpose of negotiating the TLS control channel.

tls-client;

Enable TLS and assume client role during TLS handshake.

tls-auth *file*;

Add an additional layer of HMAC authentication on top of the TLS control channel to mitigate DoS attacks and attacks on the TLS stack.

file (type: **name** of **shared-file**, see common(5))

remote *host* [*port* [*proto*]];

Remote host name or IP address.

On the client, multiple remote items may be specified for redundancy, each referring to a different OpenVPN server.

If host is a DNS name which resolves to multiple IP addresses, one will be randomly chosen, providing a sort of basic load-balancing and failover capability.

host (type: **host**)

Host to connect to

port (type: **uint16**, optional, default: **1194**)

Port to connect to

proto (type: **ovpn-remote-proto**, optional, default: **implicit**)

protocol to use when connecting with the remote

remote-random;

When multiple remote items are specified, initially randomize the order of the list as a kind of basic load-balancing measure.

comp-lzo [*mode*];

Use fast LZO compression. May add up to 1 byte per packet for incompressible data.

mode (type: **ovpn-comp-lzo-mode**, optional, default: **none**)

verify-x509-name *name*;

Accept connections only from a host with X509 common name equal to name. The remote host must also pass all other tests of verification.

name (type: **str**)

remote-cert-ku *ku*;

Require that peer certificate was signed with an explicit key usage.

This is a useful security option for clients, to ensure that the host they connect to is a designated server.

The key usage should be encoded in hex, more than one key usage can be specified.

ku (type: **str**)

remote-cert-eku *oid*;

Require that peer certificate was signed with an explicit extended key usage.

This is a useful security option for clients, to ensure that the host they connect to is a designated server.

The extended key usage should be encoded in oid notation, or OpenSSL symbolic representation.

oid (type: **str**)

remote-cert-tls *tls*;

Require that peer certificate was signed with an explicit key usage and extended key usage based on RFC3280 TLS rules.

This is a useful security option for clients, to ensure that the host they connect to is a designated server.

The remote-cert-tls client option is equivalent to remote-cert-ku 80 08 88 remote-cert-eku "TLS Web Client Authentication"

The key usage is digitalSignature and/or keyAgreement.

The remote-cert-tls server option is equivalent to remotecert-ku a0 88 remote-cert-eku "TLS Web Server Authentication"

The key usage is digitalSignature and (keyEncipherment or keyAgreement).

This is an important security precaution to protect against a man-in-the-middle attack where an authorized client attempts to connect to another client by impersonating the server. The attack is easily prevented by having clients verify the server certificate using any one of remote-cert-tls, verify-x509-name, or tls-verify.

tls **(type: ovpn-cert-types)**

cipher [*alg***];**

Encrypt packets with cipher algorithm alg.

alg **(type: ovpn-cipher-algs, optional, default: AES-256-CBC)**

data-ciphers *list*;

Allowed ciphers to be negotiated.

If omitted, it defaults to the current default in openvpn. The last known default is { AES-256-GCM, AES-128-GCM }.

list **(type: ovpn-cipher-algs-list)**

data-ciphers-fallback *alg*;

Fallback cipher if we could not determine which cipher the peer is willing to use.

alg **(type: ovpn-cipher-algs)**

client;

A helper directive designed to simplify the configuration of OpenVPN's client mode. This directive is equivalent to using pull and tls-client.

pull;

This option must be used on a client which is connecting to a multi-client server. It indicates to OpenVPN that it should accept options pushed by the server, provided they are part of the legal set of pushable options (note that the pull option is implied by client).

route-nopull;

When used with client or pull, accept options pushed by server EXCEPT for routes.

When used on the client, this option effectively bars the server from adding routes to the client's routing table, however note that this option still allows the server to set the TCP/IP properties of the client's TUN/TAP interface.

no-ifconfig-noexec;

The interface configuration and management is independent on the OpenVPN in Kernun by default. This way, the TUN/TAP interface is configured constantly, as well as the routes specified in the routes section. Therefore, OpenVPN is not expected to configure the interface. In order to override this default (not to generate ifconfig-noexec into openvpn configuration), use this item

ifconfig-pool-persist *file*;

Persist ifconfig-pool data to file.

file (type: **name** of **shared-file**, see common(5))

client-connect *client-connect-script*;

A script that is run upon each client's connection. The common name (cn) of the client being connected is passed to the script as the parameter. If the script exits with the exit code 0, the client connection is enabled (the client still can be denied by other items in the configuration, i.e. ccd.disable, etc.). If the script exits with the exit code not 0, the client connection is denied immediately. Be sure to re-generate the configuration after eventual change made to the script.

client-connect-script (type: **name** of **shared-file**, see common(5))

client-connect-socket *filename*;

The socket for determining whether the particular client is permitted to connect at the moment. Kernun opens the socket, writes a command in form 'cc instance common-name' to it. If Kernun reads back word 'accept' from the socket, the client is considered permitted by the client-connect-socket. The client is blocked otherwise. Notice that even if the client is permitted by client-connect-socket, it maight still be blocked by some other part of the configuration.

filename (type: **str**)

ccd *name* {

ifconfig-push ... ;

ifconfig-ipv6-push ... ;

disable ... ;

push { ... }

push-reset ... ;

* iroute ... ;

* iroute-ipv6 ... ;

* route ... ;

* schedule ... ;

* raw ... ;

cn ... ;

}

The `ccd` section is derived from `ovpn-ccd` section prototype. For detail description of it, see above.

`raw` *row*;

An raw item to be put to the OpenVPN configuration file exactly as given as the "row" element.

row **(type: `str`)**

`phase` [*number*];

Application Startup Phase.

number **(type: `uint8`, optional, default: 40)**

Phase number; the lower one, the earlier start.

`tag` *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

`socket-root` [*path*];

Prefix of the path of the sockets used by openvpn. The sockets (ccd-provider, manage) are created in subdirectory openvpn.NAME within the directory given in the path element. The default is usually the desired value.

path **(type: `str`, optional, default: "/usr/local/etc")**

`fast-io`;

Optimize I/O writes to avoid polling.

Experimental OpenVPN feature, see openvpn(8).

[End of section `openvpn` description.]

SEE ALSO

configuration(7), application(5), common(5), interface(5), log(5), time(5), openvpn(8)

NAME

packet-filter — format of packet-filter component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **packet-filter** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **packet-filter** configuration directives:

yes-no (see **common**(5))

on-off (see **common**(5))

name-selection (see **common**(5))

ip-version (see **common**(5))

in-out (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

pf-osi4-proto (name-usage obligatory)

OSI layer 4 protocols.

any

icmp

ipv6-icmp

ipencap

tcp

udp

tcp-udp

gre

ipv6

ipv6-frag

ipv6-nonxt

ipv6-opts

ipv6-route

esp

ah

esp-ah

carp

pfsync

l2tp

ospf

egp

igp

eigrp

icmp-type (name-usage optional)

ICMP types.

echorep (0)

unreach (3)

squench (4)

redir (5)

althost (6)

echoreq (8)

routeradv (9)

routersol (10)

timex (11)

paramprob (12)

timereq (13)

timerep (14)

inforeq (15)

inforep (16)

maskreq (17)

maskrep (18)

pf-scheduler (name-usage obligatory)

PF schedulers.

cbq

priq

hfsc

pf-proc-mode (name-usage obligatory)

PF packet processing modes.

direct

Forward packet directly without any further processing.

Technically, the packet is tagged by the NONTRANSP tag.

application

Let the packet normally process by Kernun applications.

Technically, the packet is tagged by the APPLICATION tag.

tag

Process packet normally by Kernun and tag it by a new tag.

notag

Process packet normally by Kernun and do not tag it.

proxy-ng

Process packet with proxy-ng transparent listen socket.

ITEMS AND SECTIONS

Configuration of **packet-filter** library component consists of following prototypes:

* peer-list ... ;
  pf-processing ... ;
* pf-raw-acl *name* { ... }
* pf-acl *name* { ... }
  packet-filter { ... }

Description:

peer-list [*addr* [**port** *port*]];

Packet Filter peer list definition.

addr (type: **host-set**, optional, default: *)

Set of peer addresses/names.

port *port* (type: **port-set**, optional, default: *)

Set of ports (valid with TCP/UDP only).

Constraints:

Host list must not be empty.

Regexps and discontiguous masks not allowed in PF lists.

pf-processing [direct];

pf-processing application;

pf-processing tag *tag*;

pf-processing notag;

pf-processing proxy-ng listen-socket-id *listen-socket-id*;

Packet processing mode definition.

<branching element> (type: **pf-proc-mode**, optional, default: direct)

tag (type: **str**)

listen-socket-id *listen-socket-id* (type: **str**)

ID of the PROXY-NG transparent listening socket.

pf-raw-acl *name* {

* descr ... ;

* raw ... ;

}

Packet Filter raw rule set definition.

Items & subsections:

descr *text*;

Rule set comment.

text (type: **str**)

raw *line*;

Raw line to be put into pf.conf.

line (type: **str**)

[End of section `pf-raw-acl` description.]

pf-acl *name* **{**

* descr ... ;

* raw ... ;

* from ... ;

* to ... ;

* iface ... ;

ip ... ;

* protocol ... ;

tagged ... ;

time-period-set { ... }

deny ... ;

accept ... ;

anchor ... ;

symmetric ... ;

}

Packet Filter general rule set definition.

The pf-acl section is derived from pf-raw-acl section prototype. For detail description of it, see above.
Changes to the pf-acl section:

Exactly one of DENY, ACCEPT, ANCHOR and RAW must be specified.

Cannot specify entry condition if RAW used.

Valid transport protocol required if PORT used.

Added items & subsections:

from [*addr* **[port** *port***]];**

Entry condition - source addresses.

addr **(type: host-set, optional, default: *)**

Set of peer addresses/names.

port *port* **(type: port-set, optional, default: *)**

Set of ports (valid with TCP/UDP only).

Constraints:

Host list must not be empty.

Regexps and discontiguous masks not allowed in PF lists.

to [*addr* **[port** *port***]];**

Entry condition - destination addresses.

addr **(type: host-set, optional, default: *)**

Set of peer addresses/names.

port *port* (type: **port-set**, optional, default: *)

Set of ports (valid with TCP/UDP only).

Constraints:

Host list must not be empty.

Regexps and discontiguous masks not allowed in PF lists.

iface any [*dir*];

iface [**name**] *name* [*dir*];

Entry condition - incoming interface.

<branching element> (type: **name-selection**, optional, default: name)

name (type: **name** of **interface**, see interface(5))

dir (type: **in-out**, optional, default: both)

ip *version*;

IP protocol version (IPv4 and IPv6 if not specified).

version (type: **ip-version**)

protocol any;

protocol icmp [**icmp-type** *icmp-type*];

protocol ipv6-icmp;

protocol ipencap;

protocol tcp [**flags** *flags*];

protocol udp;

protocol tcp-udp [**flags** *flags*];

protocol gre;

protocol ipv6;

protocol ipv6-frag;

protocol ipv6-nonxt;

protocol ipv6-opts;

protocol ipv6-route;

protocol esp;

protocol ah;

protocol esp-ah;

protocol carp;

protocol pfsync;

protocol l2tp;

protocol ospf;

protocol egp;

protocol igp;

protocol eigrp;

Entry condition - OSI layer 4 protocol.

<branching element> (type: **pf-osi4-proto**)

flags *flags* (type: **str**, optional, default: <NULL>)
icmp-type *icmp-type* (type: **icmp-type-list**, optional, default: <NULL>)

tagged *tag*;

Entry condition - packet tag.

tag (type: **str**)

time-period-set {

exclude ... ;

* time-spec *name* { ... }

}

The time-period-set section is derived from time-period-set section prototype. For detail description of it, see time(5).

deny;

Global decision mode: operation will not be served.

accept;

Global decision mode: operation will be served.

anchor *path*;

Apply anchor rules.

path (type: **str**)

symmetric;

Use rules for symmetric routing, too.

If used, rules for opposite direction are generated, too, e.g. besides "pass in on ep0 from <A> to <B>", also "pass out on ep0 from <B> to <A>" is generated.

[End of section `pf-acl` description.]

packet-filter {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

pflog ... ;

pfsync ... ;

comm-dir ... ;

ignore-iface ... ;

pcap-timeout ... ;

buffer-size ... ;

* set-option ... ;

timeouts { ... }

limits { ... }

logging-frequence ... ;

* altq *name* { ... }

* scrub-acl *name* { ... }

* rdr-acl *name* { ... }

* nat-acl *name* { ... }

* binat-acl *name* { ... }

* filter-acl *name* { ... }

* load-anchor ... ;

}

Packet filter configuration.

This section allows to define almost all common features for the /etc/pf.conf configuration file with one important exception. The PF tables are used by the KGB for internal purposes to achieve maximal effectiveness and user cannot define them by own.

If this section is not used, the /etc/pf.conf file will be left untought.

For configuration attributes details, see pf.conf(5).

The `packet-filter` section is derived from `alone-application` section prototype. For detail description of it, see application(5).
Changes to the `packet-filter` section:

Section `monitoring` is not valid.

Item `phase` (see common(5))

Element `number` is optional, default: 30.

Added items & subsections:

`pflog` [*dev*];

Device for pflog monitoring.

dev **(type: `str`, optional, default: "pflog0")**

`pfsync` [*dev*];

Device for pfsync monitoring.

dev **(type: `str`, optional, default: "pfsync0")**

`comm-dir` [*path*];

Directory for communication with pfctl.

path **(type: `str`, optional, default: "/tmp")**

`ignore-iface` *ifaces*;

Set of interfaces not monitored by pflogger.

ifaces **(type: `str-set`)**

`pcap-timeout` [*sec*];

Timeout for pcap reader.

sec **(type: `uint32`, optional, default: 500)**

`buffer-size` [*bytes*];

Buffer size for pcap reader.

bytes **(type: `uint32`, optional, default: 20Mi)**

`set-option` *line*;

Setting PF options by SET directive.

line **(type: `str`)**

Option setting (w/o SET keyword).

`timeouts` {

tcp-closing ... ;

tcp-finwait ... ;

tcp-closed ... ;

udp-first ... ;

udp-single ... ;

udp-multiple ... ;

}

Setting various PF timeouts.

Items & subsections:

tcp-closing *seconds*;

Time limit for the state after the first FIN has been sent.

seconds **(type: uint32)**

tcp-finwait *seconds*;

Time limit for the state after both FINs have been exchanged and the connection is closed.

seconds **(type: uint32)**

tcp-closed *seconds*;

Time limit for the state after one endpoint sends an RST.

seconds **(type: uint32)**

udp-first *seconds*;

Time limit for the state after the first packet.

seconds **(type: uint32)**

udp-single *seconds*;

Time limit for the state if the source host sends more than one packet but the destination host has never sent one back.

seconds **(type: uint32)**

udp-multiple *seconds*;

Time limit for the state if both hosts have sent packets.

seconds **(type: uint32)**

[End of section `packet-filter.timeouts` description.]

limits {

states ... ;

frags ... ;

table-entries ... ;

}

Setting various PF limits.

Items & subsections:

states [*size*];

Maximum number of entries in the memory pool used for state table entries

size **(type: uint32, optional, default: 100000)**

frags [*size*];

Maximum number of entries in the memory pool used for packet reassembly (scrub rules).

size **(type: uint32, optional, default: 50000)**

table-entries [*size*];

Maximum number of addresses stored in the packet filter tables.
The adaptive-firewall tables have a separate limit.

size **(type: uint32, optional, default: 200000)**

[End of section `packet-filter.limits` description.]

`logging-frequence` [*sec*];

Frequence of logging stateless events counter.

sec (type: **`uint32`**, optional, default: 60)

`altq` *name* {

on ... ;

scheduler ... ;

bandwidth ... ;

qlimit ... ;

tbrsize ... ;

* queue ... ;

}

ALTQ per interface definition.

Constraints:

Interface name must be defined.

Bandwidth must be defined.

At least one queue must be defined.

Items & subsections:

`on` *name*;

name (type: **`name`** of **`interface`**, see interface(5))

`scheduler` [*name*];

name (type: **`pf-scheduler`**, optional, default: cbq)

`bandwidth` *bits*;

bits (type: **`uint64`**)

`qlimit` *packets*;

packets (type: **`uint32`**)

`tbrsize` *bytes*;

bytes (type: **`uint64`**)

`queue` *name*;

name (type: **`name`** of **`pf-queue`**, see pf-queue(5))

[End of section `packet-filter.altq` description.]

`scrub-acl` *name* {

* descr ... ;

* raw ... ;

* from ... ;

* to ... ;

* iface ... ;

ip ... ;
* protocol ... ;
time-period-set { ... }
deny ... ;
accept ... ;
symmetric ... ;
no-df ... ;
max-mss ... ;
log ... ;
}

Traffic normalization definition.

If not used, the SCRUB IN ALL directive will be generated.

The `scrub-acl` section is derived from `pf-acl` section prototype. For detail description of it, see above.
Changes to the `scrub-acl` section:

Item `tagged` is not valid.

Item `anchor` is not valid.

Added items & subsections:

`no-df`;

Clear the dont-fragment bit from IP packets.

`max-mss` ***number*;**

Enforce a maximum MSS for matching TCP packets.

number **(type: `uint32`)**

`log` **[*mode*];**

Log packets.

mode **(type: `on-off`, optional, default: on=1)**

[End of section `packet-filter.scrub-acl` description.]

`rdr-acl` ***name*** **{**

* descr ... ;
* raw ... ;
* from ... ;
* to ... ;
* iface ... ;
ip ... ;
* protocol ... ;
tagged ... ;
time-period-set { ... }
deny ... ;
accept ... ;

anchor ... ;

rdr-to ... ;

process ... ;

}

NAT redirection definition.

The `rdr-acl` section is derived from `pf-acl` section prototype. For detail description of it, see above.

Changes to the `rdr-acl` section:

Item `symmetric` is not valid.

RDR-TO must be specified if ACCEPT used.

Valid transport protocol required if PORT used.

Item `iface` (see above)

Interface direction not allowed.

Item `protocol` (see above)

ICMP type constraint not allowed.

Added items & subsections:

`rdr-to` *addr* [`port` *port*];

addr **(type: `host`)**

New target address

`port` *port* (type: `port`, optional, default: 0)

New target port (valid with TCP/UDP only)

`process [direct];`

`process application;`

`process tag` *tag*;

`process notag;`

`process proxy-ng listen-socket-id` *listen-socket-id*;

Packet processing mode definition.

<branching element> (type: `pf-proc-mode`, optional, default: direct)

tag **(type: `str`)**

`listen-socket-id` *listen-socket-id* (type: `str`)

ID of the PROXY-NG transparent listening socket.

[End of section `packet-filter.rdr-acl` description.]

`nat-acl` *name* {

* descr ... ;

* raw ... ;

* from ... ;

* to ... ;

* iface ... ;

ip ... ;

* protocol ... ;
  tagged ... ;
  time-period-set { ... }
  deny ... ;
  accept ... ;
  anchor ... ;
  map-to ... ;
  process ... ;
  static-port ... ;
}

NAT mapping definition.

The `nat-acl` section is derived from `pf-acl` section prototype. For detail description of it, see above.

Changes to the `nat-acl` section:

Item `symmetric` is not valid.

MAP-TO must be specified if ACCEPT used.

Valid transport protocol required if PORT used.

Item `iface` (see above)

Interface direction not allowed.

Item `protocol` (see above)

ICMP type constraint not allowed.

Added items & subsections:

`map-to` *addr* [`port` *port*];

addr **(type: `host-list`)**

New source addresses list

`port` *port* (type: `port`, optional, default: 0)

New source port (valid with TCP/UDP only)

`process [direct];`

`process application;`

`process tag` *tag*;

`process notag;`

`process proxy-ng listen-socket-id` *listen-socket-id*;

Packet processing mode definition.

<branching element> (type: `pf-proc-mode`, optional, default: direct)

tag **(type: `str`)**

`listen-socket-id` *listen-socket-id* (type: `str`)

ID of the PROXY-NG transparent listening socket.

`static-port;`

STATIC-PORT option of NAT rule.

[End of section `packet-filter.nat-acl` description.]

binat-acl *name* {

* descr ... ;

* raw ... ;

}

BINAT mapping definition.

The binat-acl section is derived from pf-raw-acl section prototype. For detail description of it, see above.

filter-acl *name* {

* descr ... ;

* raw ... ;

* from ... ;

* to ... ;

* iface ... ;

ip ... ;

* protocol ... ;

tagged ... ;

time-period-set { ... }

deny ... ;

accept ... ;

anchor ... ;

symmetric ... ;

antispoof ... ;

log ... ;

continue ... ;

return ... ;

fastroute ... ;

route-to ... ;

queue ... ;

process ... ;

no-state ... ;

* option ... ;

}

Filter rule set definition.

The filter-acl section is derived from pf-acl section prototype. For detail description of it, see above.

Changes to the filter-acl section:

RETURN/ANTISPOOF can be used only with DENY.

QUEUE, PROCESS and OPTION can be used only with ACCEPT.

FASTROUTE and ROUTE-TO are mutually exclusive.

ANCHOR and LOG are mutually exclusive.

Cannot specify other entry conditions if ANTISPOOF used.
FASTROUTE should be used together with an IFACE.NAME.

Added items & subsections:

antispoof [loop] [routes];

Special entry condition - blocking of faked source addresses on particular interfaces.
If this ACL has a particular interface defined by the IFACE item, antispoof rules are valid for this interface. Otherwise, the rules are applied to all interfaces.
By default, an ACL with ANTISPOOF guarantees filtering of all packets with source addresses from given interface address set coming from other interfaces. Packets sent either locally, or to the local end of a CARP interface are allowed.
This ACL must have DENY policy.

loop (type: key, optional)

Include blocking for loopback interface, too.
With this keyword option, packets with source address from given interface address set, sent either locally, or to the local end of a CARP interface are denied, too.

routes (type: key, optional)

Include blocking for routes, too.
With this keyword option, addresses of all networks routes via given interface are added to the list of denied source addresses.

log [*mode*];

Log packets.

mode **(type: on-off, optional, default: on=1)**

continue;

Last-match applied for all rules in this ACL.
By default, first-match (i.e. QUICK) mode is used.

return [icmp] [*code*];

Return mode definition.
If not used, denial will be done by DROPping packets.

icmp (type: key, optional)

code **(type: uint8, optional, default: 0)**

ICMP message code, ICMP UNREACHABLE by default.

fastroute;

route-to *iface* [*addr*];

iface **(type: name of interface,** see interface(5))

addr **(type: host, optional, default: [0.0.0.0])**

queue *name*;

name **(type: name of pf-queue,** see pf-queue(5))

process [direct];

process application;

process tag *tag*;

process notag;

process proxy-ng listen-socket-id *listen-socket-id*;

Packet processing mode definition.

<**branching element**> (**type: pf-proc-mode**, **optional**, **default: direct**)

tag (**type: str**)

listen-socket-id *listen-socket-id* (**type: str**)

ID of the PROXY-NG transparent listening socket.

no-state;

Disable PF state establishment.

For denial ACLs this item has no meaning since this behavior is default one in this case.

option *text*;

Free-form rule option.

text (**type: str**)

[End of section `packet-filter.filter-acl` description.]

load-anchor *path* **from** *from*;

Loading rules from file into anchor.

path (**type: str**)

Anchor name.

from *from* (**type: str**)

File name.

[End of section `packet-filter` description.]

SEE ALSO

configuration(7), altq(4), application(5), common(5), interface(5), log(5), pf-queue(5), pf.conf(5), time(5)

NAME

pf-control.cfg — format of pf-control program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **pf-control.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **pf-control.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

on-off (see **common**(5))

name-selection (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

in-out (see **common**(5))

report-mode (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

inline-file-format (see **common**(5))

task-frequency (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

source-address-mode (see source-address(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

ids-agent-log-level (see adaptive-firewall(5))

ids-agent-detection-direction (see adaptive-firewall(5))

ids-agent-protocol (see adaptive-firewall(5))

ids-agent-rule-action (see adaptive-firewall(5))

ids-agent-threshold-type (see adaptive-firewall(5))

ids-agent-threshold-track-by (see adaptive-firewall(5))

ids-agent-rate-filter-track-by (see adaptive-firewall(5))

ids-agent-suppress-direction (see adaptive-firewall(5))

policy-level (see adaptive-firewall(5))

pf-osi4-proto (see packet-filter(5))

icmp-type (see packet-filter(5))

pf-scheduler (see packet-filter(5))

pf-proc-mode (see packet-filter(5))

membertype (name-usage obligatory)

    host
    any

ITEMS AND SECTIONS

Program **pf-control** recognizes following items and sections:
  adaptive-firewall { ... }
* shared-file *name* { ... }
* interface *name* { ... }
* pf-queue *name* { ... }
* resolver *name* { ... }

sysctl { ... }
use-resolver ... ;
pf { ... }
ipv6-mode ... ;

Description:

adaptive-firewall {

ids-agent { ... }
* watchdog *name* { ... }
honeypot { ... }
auto-blocking { ... }
adaptive-database { ... }
address-groups { ... }
port-groups { ... }
whitelist ... ;
blacklist ... ;
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
}

The adaptive-firewall section is derived from adaptive-firewall section prototype. For detail description of it, see adaptive-firewall(5).

shared-file *name* **{**

path ... ;
format ... ;
}

The shared-file section is derived from shared-file section prototype. For detail description of it, see common(5).

interface *name* **{**

dev ... ;
ipv4 ... ;
ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`pf-queue` *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

`resolver` *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`pf` {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

pflog ... ;

pfsync ... ;

comm-dir ... ;

ignore-iface ... ;

pcap-timeout ... ;

buffer-size ... ;

* set-option ... ;

timeouts { ... }

limits { ... }

logging-frequence ... ;

* altq *name* { ... }

* scrub-acl *name* { ... }

* rdr-acl *name* { ... }

* nat-acl *name* { ... }

* binat-acl *name* { ... }

* filter-acl *name* { ... }

* load-anchor ... ;

pf-conf ... ;

* table *name* { ... }

}

The `pf` section is derived from `pf` section prototype. For detail description of it, see above.

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), pf-control(8), adaptive-firewall(5), common(5), interface(5), listen-on(5), log(5), packet-filter(5), pf-queue(5), resolver(5), source-address(5), sysctl(5), time(5)

NAME

pf-queue — format of pf-queue component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **pf-queue** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **pf-queue** configuration directives:

bandwidth-mode (name-usage obligatory)

PF bandwidth config modes.

abs

Absolute bandwidth

ratio

Bandwidth relative to parent

pf-sc-setting (name-usage obligatory)

PF Service Curve setting.

total

initial

ITEMS AND SECTIONS

Configuration of **pf-queue** library component consists of following prototypes:

pf-sc ... ;
pf-sched-options { ... }
* pf-queue *name* { ... }

Description:

pf-sc [total] *total-bw*;

pf-sc initial *init-bw milliseconds total-bw*;

PF - HFSC Service Curve.

<branching element> (type: **pf-sc-setting**, optional, default: total)

init-bw (type: **uint64**)

milliseconds (type: **uint32**)

total-bw (type: **uint64**)

pf-sched-options {

default ... ;

red ... ;

rio ... ;

ecn ... ;

borrow ... ;

realtime ... ;

upperlimit ... ;

linkshare ... ;

}

Items & subsections:

default;

red;

rio;

ecn;

borrow;

realtime [**total**] *total-bw*;

realtime initial *init-bw milliseconds total-bw*;

PF - HFSC Service Curve.

<branching element> (type: **pf-sc-setting**, optional, default: total)

init-bw (type: **uint64**)

milliseconds (type: **uint32**)

total-bw (type: **uint64**)

upperlimit [**total**] *total-bw*;

upperlimit initial *init-bw milliseconds total-bw*;

PF - HFSC Service Curve.

<branching element> (type: **pf-sc-setting**, optional, default: total)
init-bw (type: **uint64**)
milliseconds (type: **uint32**)
total-bw (type: **uint64**)

linkshare [total] *total-bw*;

linkshare initial *init-bw milliseconds total-bw*;

PF - HFSC Service Curve.
<branching element> (type: **pf-sc-setting**, optional, default: total)
init-bw (type: **uint64**)
milliseconds (type: **uint32**)
total-bw (type: **uint64**)

[End of section `pf-sched-options` description.]

pf-queue *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

Packet filter QUEUE definition.

For configuration attributes details, see pf.conf(5).

Constraints:

CBQ, PRIQ and HFSC are mutually exclusive.

Items & subsections:

parent *name*;

Parent queue definition (child queue only).

name (type: **name** of **pf-queue**, see above)

bandwidth [abs] *bits*;

bandwidth ratio *percent*;

Queue bandwidth limit.

If omitted, 100% of parent bandwidth assumed.

<branching element> (type: **bandwidth-mode**, optional, default: abs)

bits (type: **uint64**)

Required bandwidth in bps.

percent (type: **uint8**)

Required part of parent bandwidth.

Constraints:

Percent value must be less 100.

priority [*prty*];

prty (type: **uint8**, optional, default: 1)

qlimit *packets*;

packets (type: **uint32**)

cbq {

default ... ;

red ... ;

rio ... ;

ecn ... ;

borrow ... ;

}

The cbq section is derived from pf-sched-options section prototype. For detail description of it, see above.
Changes to the cbq section:

Item `realtime` is not valid.

Item `upperlimit` is not valid.

Item `linkshare` is not valid.

priq {

default ... ;

red ... ;

rio ... ;

ecn ... ;

}

The priq section is derived from pf-sched-options section prototype. For detail description of it, see above.
Changes to the priq section:

Item `borrow` is not valid.

Item `realtime` is not valid.

Item `upperlimit` is not valid.

Item `linkshare` is not valid.

hfsc {

default ... ;

red ... ;
rio ... ;
ecn ... ;
realtime ... ;
upperlimit ... ;
linkshare ... ;
}

The `hfsc` section is derived from `pf-sched-options` section prototype. For detail description of it, see above.
Changes to the `hfsc` section:

Item `borrow` is not valid.

[End of section `pf-queue` description.]

SEE ALSO

configuration(7), altq(4), pf.conf(5)

NAME

pike — format of pike component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **pike** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **pike** configuration directives:

yes-no (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

listen-on-sock (see **listen-on**(5))

proc-priority (see **application**(5))

pike-control-type (name-usage obligatory)

tag

cmd

ITEMS AND SECTIONS

Configuration of **pike** library component consists of following prototypes:

pikemon { ... }

Description:

pikemon {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

udpserver { ... }

priority ... ;

status-file ... ;

hmac ... ;

devd-socket ... ;

garp-keepalive ... ;

* virtual-cluster *name* { ... }

}

PIKE Monitoring Daemon configuration.

The `pikemon` section is derived from `alone-application` section prototype. For detail description of it, see application(5).

Changes to the `pikemon` section:

At least one VIRTUAL-CLUSTER must be defined.

HMAC has to be configured.

Added items & subsections:

listen-on {

* socket ... ;

}

The listen-on section is derived from listen-on section prototype.
For detail description of it, see listen-on(5).
Changes to the listen-on section:

Item `non-transparent` used as `socket`.

Item `transparent` is not valid.

Exactly one address to listen on must be specified.

Item socket (see listen-on(5))

Element `proto` is optional, default: tcp.

udpserver {

max-sessions ... ;

}

The udpserver section is derived from udpserver section prototype.
For detail description of it, see udpserver(5).

priority normal;

priority [realtime] [*realtime***];**

Process priority setting.

<branching element> (type: proc-priority, optional, default: realtime)

***realtime* (type: uint8, optional, default: 16)**

Realtime priority (parameter of rtprio() call).

Accepted values between 0 and 31; 0 is the highest priority.

Constraints:

Priority value must be between 0 and 31.

status-file [*name***];**

File with current cluster status.

***name* (type: str, optional, default: "/var/run/pikemon.status")**

hmac [optional] ***shared-secret*;**

Protocol Verification by HMAC.

The purpose of the OPTIONAL flag is just to eliminate potential problems when changing the shared secret. The first step is to set OPTIONAL on both systems, so that they will temporarily omit the HMAC checking. Then, the shared secret can be changed without the risc of ignoring PIKE protocol packets on either system. When the new configuration is propagated onto both cluster members, the OPTIONAL flag should be cleared.

optional (type: key, optional)

Flag to eliminate HMAC checking.

shared-secret **(type: `str`)**
Shared secret for HMAC SHA256.

`devd-socket` **[***path***];**
Socket of devd(8).

path **(type: `str`, optional, default: "/var/run/devd.pipe")**

`garp-keepalive` **[***period***];**
Periodic sending of GARP by master.

period **(type: `uint32`, optional, default: 60)**
Setting to zero disables the feature.

`virtual-cluster` *name* **{**
id ... ;
* interface ... ;
hello-period ... ;
hello-timeout ... ;
* ping-group *name* { ... }
* iface-monitor ... ;
down-timeout ... ;
up-timeout ... ;
preemptive ... ;
primary ... ;
* control ... ;
}
Single set of virtual addresses with redundancy monitoring.

Items & subsections:

`id` **[***id***];**
Virtual cluster ID.

id **(type: `uint8`, optional, default: 0)**

`interface` *virt***;**
Interface belonging to virtual cluster.

virt **(type: `name` of `interface`, see interface(5))**
Virtual interface name.

`hello-period` **[***sec***];**
Period of PIKE HELLO subprotocol.

sec **(type: `uint8`, optional, default: 1)**

`hello-timeout` **[***sec***];**
Timeout of PIKE HELLO subprotocol.
When a node does not get a PIKE HELLO packet within this period, it assumes the partner to be dead.

When a node does not get a PIKE HELLO packet with UP state from the partner within this period, it assumes the partner to be down.

This timeout should be longer than the longest PING timeout.

sec **(type: uint8, optional, default: 10)**

ping-group *name* **{**

timeout ... ;

* host ... ;

}

Group of hosts being pinged.

Every defined group within a VIRTUAL-CLUSTER section must be alive to bring monitored interfaces "up".

The ping-group section is derived from ping-group section prototype. For detail description of it, see ping(5).

iface-monitor *name*;

Interface being monitored.

name **(type: name of interface, see interface(5))**

down-timeout [*sec*];

Cluster down timeout.

At least one tested IP group must be inaccessible for this time in order to switch the cluster interfaces "down".

sec **(type: uint32, optional, default: 0)**

Timeout in seconds, zero means immediate action.

up-timeout [*sec*];

Cluster up timeout.

All tested IP groups must be accessible for this time in order to switch the cluster interfaces "up".

sec **(type: uint32, optional, default: 0)**

Timeout in seconds, zero means immediate action.

preemptive [*status*];

Preemptive mode.

In this mode, the primary firewall takes the master role whenever is ready.

status **(type: yes-no, optional, default: yes)**

primary [*status*];

Primary router flag.

In preemptive mode, the node marked as primary acts as the primary, dedicated node.

status **(type: yes-no, optional, default: no)**

control [**tag**] *tag*;

control cmd *up down*;

Cluster control.

This item allows to specify which components depend on master/backup state of a virtual cluster, or what commands should be run when the state change occurs.

<branching element> (type: **pike-control-type**, optional, default: tag)

tag **(type: str)**

Control tag - components with this tag run only in master state.

up **(type: str)**

Control command - executed when taking the master role.

down **(type: str)**

Control command - executed when taking the backup role.

[End of section `pikemon.virtual-cluster` description.]

[End of section `pikemon` description.]

SEE ALSO

configuration(7), application(5), common(5), interface(5), listen-on(5), log(5), ping(5), udpserver(5), devd(8)

NAME

pikemon.cfg — format of pikemon program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **pikemon.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **pikemon.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

lagg-protocol (see **interface**(5))

listen-on-sock (see **listen-on**(5))

log-in-vain-proto (see **sysctl**(5))

blackhole-proto (see **sysctl**(5))

proc-priority (see **application**(5))

pike-control-type (see **pike**(5))

ITEMS AND SECTIONS

Program **pikemon** recognizes following items and sections:
* interface *name* { ... }
* resolver *name* { ... }
  sysctl { ... }
  use-resolver ... ;
  pikemon { ... }
  ipv6-mode ... ;

Description:

`interface` ***name*** **{**

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`resolver` ***name*** **{**

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

***name* (type: `name` of `resolver`, see resolver(5))**

`pikemon` {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

udpserver { ... }

priority ... ;

status-file ... ;

hmac ... ;

devd-socket ... ;

garp-keepalive ... ;

* virtual-cluster *name* { ... }

}

The `pikemon` section is derived from `pikemon` section prototype. For detail description of it, see pike(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), pikemon(8), application(5), common(5), interface(5), listen-on(5), log(5), pike(5), resolver(5), sysctl(5)

NAME

ping — format of ping component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ping** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

ITEMS AND SECTIONS

Configuration of **ping** library component consists of following prototypes:
* ping-group *name* { ... }

Description:

ping-group *name* **{**

timeout ... ;

* host ... ;

}

Group of hosts being pinged.

Constraints:

At least one HOST must be specified.

Ping timeout must be specified.

Items & subsections:

timeout *seconds*;

Timeout for ping responses.

seconds (**type: uint16**)

host *target* [**source-address** *source-address*];

Ping group member description.

At least one of hosts within a group must respond within given timeout, this group to be recognized as "alive".

target **(type: `host`)**
: Host IP address

`source-address` ***source-address*** **(type: `host`, optional, default: [0.0.0.0])**
: Source IP address

[End of section `ping-group` description.]

SEE ALSO

configuration(7)

NAME

pop3-proxy — format of pop3-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **pop3-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **pop3-proxy** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

nls (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

auth-method (see **auth**(5))

virus-status (see antivirus(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

listen-on-sock (see listen-on(5))

mime-header-check-type (see mod-mail-doc(5))

pop3-cmd (name-usage obligatory)

POP3 commands

UNKNOWN

command unknown to the proxy

APOP

AUTH

CAPA

DELE

LIST

NOOP

PASS

QUIT

RETR

RSET

STAT

STLS

TOP

UIDL

USER

pop3-capa (name-usage obligatory)

POP3 capabilities

UNKNOWN

capability unknown to the proxy

EXPIRE

IMPLEMENTATION

LOGIN-DELAY

PIPELINING

RESP-CODES

SASL

STLS

TOP

UIDL

USER

ITEMS AND SECTIONS

Configuration of **pop3-proxy** library component consists of following prototypes:
* pop3-proxy *name* { ... }

Description:

pop3-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

ssl-session-cache { ... }

mail-pool ... ;

* session-acl *name* { ... }

* command-acl *name* { ... }

* mail-acl *name* { ... }

* doc-acl *name* { ... }

}

POP3 proxy configuration.

The `pop3-proxy` section is derived from `proxy` section prototype.
For detail description of it, see application(5).
Changes to the `pop3-proxy` section:

Section `udpserver` is not valid.

MAIL-POOL must be specified.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one COMMAND-ACL must be specified.

Section `monitoring` (see monitoring(5))

Item `aproxy-user` is not valid.

Item `data` used as `uri`.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 110.

Element `proto` is optional, default: `tcp`.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 110.

Element `proto` is optional, default: `tcp`.

Item `doctype-identification.order` (see acl(5))

Only DOWNLOAD direction can be used.

Added items & subsections:

`client-conn` {

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection to client options.

The `client-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `client-conn` section:

Item `conn-timeout` is not valid.

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`server-conn` {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection to server options.

The `server-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `server-conn` section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`ssl-session-cache` {

capacity ... ;

dir ... ;

lock ... ;

}

The `ssl-session-cache` section is derived from `ssl-session-cache` section prototype. For detail description of it, see ssl(5).

`mail-pool` *name*;

Mail pool directory.

name **(type: `str`)**

`session-acl` *name* {

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

auth ... ;

idle-timeout ... ;

source-address ... ;

plug-to ... ;

client-ssl ... ;

* client-cert-match ... ;

language ... ;

}

The first level ACL decides how to handle incoming connections.

The `session-acl` section is derived from `acl-1` section prototype.
For detail description of it, see acl(5).
Changes to the `session-acl` section:

Item `user` is not valid.

Item `idle-timeout-peer` is not valid.

SSL/TLS required on in order to match client certificates.

IDLE-TIMEOUT has no use without SSL/TLS.

Item `doctype-ident-order` (see acl(5))

Only DOWNLOAD direction can be used.

Item `auth` (see auth(5))

Element `mode` is optional, default: allowed.

Only out-of-band authentication is supported in this proxy.

Added items & subsections:

`client-ssl` *params*;

Use SSL/TLS on the connection from a client.

params **(type: `name` of `ssl-params`, see ssl(5))**

`client-cert-match` [`subject` *subject*] [`issuer` *issuer*];

Requirements for client certificate.

`subject` *subject* (type: `str-set`, optional, default: *)

acceptable certificate subjects

`issuer` *issuer* (type: `str-set`, optional, default: *)

acceptable certificate issuers

`language` [*code*];

Language and charset of responses generated by Kernun.

If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

code **(type: `nls`, optional, default: EN)**

[End of section `pop3-proxy.session-acl` description.]

`command-acl` *name* {

* from ... ;

* server ... ;

* user ... ;
* time ... ;
  time-period-set { ... }
* session-acl ... ;
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
  source-address ... ;
  plug-to ... ;
* client-cert-match ... ;
  server-ssl ... ;
* server-cert-match ... ;
  language ... ;
  max-bytes-in ... ;
  max-bytes-out ... ;
  max-mail-in ... ;
  max-time ... ;
  idle-timeout ... ;
  commands ... ;
  capabilities ... ;
  cmd-line-len ... ;
  resp-line-len ... ;
  mail-filter ... ;
  use-antispam ... ;
  use-antivirus ... ;
  no-mail-scanning ... ;
  client-altq ... ;
  server-altq ... ;
}

The second level ACL sets parameters of the connection to the server and decides about handling individual commands.

The `command-acl` section is derived from `acl-2` section prototype.
For detail description of it, see acl(5).
Changes to the `command-acl` section:

Item `parent-acl` used as `session-acl`.

SSL/TLS required on in order to match server certificates.

MAIL-FILTER, USE-ANTISPAM, and USE-ANTIVIRUS cannot be used together with NO-MAIL-SCANNING.

Item `doctype-ident-order` (see acl(5))

Only DOWNLOAD direction can be used.

Added items & subsections:
`source-address [client] [addr4` *addr4*`] [addr6` *addr6*`] cluster [`*cluster*`];`
`source-address [client] [addr4` *addr4*`] [addr6` *addr6*`] [physical];`
`source-address [client] [addr4` *addr4*`] [addr6` *addr6*`] no-fallback;`

Source address for outgoing connections to servers.

If omitted, the proper address of the proxy will be used, i.e. in the case of a cluster, the cluster address will be used.

If not specified by the SOURCE-PORT item, a generic port will be used.

The elements entered within this item will be used by the proxy until the first of them is applicable:

- The CLIENT keyword means the original client IP address is used. This mode will be succesful in all cases except mismatch of IP address families.

- The ADDR4/ADDR6 keyword-value pairs mean that the specified address is used for a connection of corresponding address family.

- The CLUSTER keyword means that one of cluster addresses will be used. By default, the main address of the bridge is used, however, any preferred alias address can be listed in the cluster list.- The PHYSICAL option means that the address of the physical interface is used instead of the cluster one.

- The DEFAULT option means the default behavior - i.e. using of the physical address.

- The NO-FALLBACK option means that if no other way of setting the address is acceptable, the session is rejected. Without this option, the system tries to find a suitable source IP address automatically.

`client` (type: `key`, optional)

`addr4` *addr4* (type: `host`, optional, default: [0.0.0.0])

`addr6` *addr6* (type: `host`, optional, default: [::])
<branching element> (type: `source-address-mode`, optional, default: physical)
***cluster* (type: `host-list`, optional, default: {})**

Constraints:

Address family must respect the element's address family..

`plug-to` *addr*`;`

Final destination server.

***addr* (type: `sock`)**

Address/port of final destination server.

If port is zero, then original port is used.

`client-cert-match [subject` *subject*`] [issuer` *issuer*`];`

Entry condition - select an ACL according to a client certificate.

`subject` *subject* (type: `str-set`, optional, default: *)

acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)
acceptable certificate issuers

server-ssl *params*;
Use SSL/TLS on the connection to a server.

params (type: **name of ssl-params**, see ssl(5))

server-cert-match [**subject** *subject*] [**issuer** *issuer*];
Requirements for server certificate.

subject *subject* (type: **str-set**, optional, default: *)
acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)
acceptable certificate issuers

language *code*;
Language and charset of responses generated by Kernun.
If omitted in SESSION-ACL, English is used.If omitted in higer layer ACLs, settings from lower layer is used.

code (type: **nls**)

max-bytes-in *bytes*;
Maximum number of bytes from server to client in a session.

bytes (type: **uint64**)

max-bytes-out *bytes*;
Maximum number of bytes from client to server in a session.

bytes (type: **uint64**)

max-mail-in *bytes*;
Maximum size of any single mail transferred from client to server.

bytes (type: **uint64**)

max-time *seconds*;
Maximum time of session

seconds (type: **uint32**)

idle-timeout [*seconds*];
If no data transmitted for this session in the period of idle-timeout seconds, connection is closed.
If omitted, value of proxy session-acl.idle-timeout is used.

seconds (type: **uint32**, optional, default: **0**)

commands [*cmd*];
Set of allowed POP3 commands.

cmd (type: **pop3-cmd-set**, optional, default: *)

capabilities [*cap*];
Set of allowed POP3 capabilities (sent in response to command.

cap (type: **pop3-capa-set**, optional, default: *)

cmd-line-len [*bytes*];

Maximum length of a command line (including CRLF).

bytes (type: **uint32**, optional, default: **255**)

resp-line-len [*bytes*];

Maximum length of a response line (including CRLF).

bytes (type: **uint32**, optional, default: **512**)

mail-filter *name*;

Filter for mails

name (type: **name** of **mail-filter**, see mod-mail-doc(5))

use-antispam disable;

use-antispam enable *channel* [*limit*];

Antispam usage.

This section defines type of antispam daemon used and mode of antispam checking operation.

<branching element> (type: **enabling**)

channel (type: **name** of **antispam**, see mod-antispam(5))

Name of antispam global section used.

Referred section defines the way how to communicate with the antispam daemon (see above).

limit (type: **uint64**, optional, default: **0**)

Size limit (in bytes) for antispam check.

Antispam checking used to be very exhausting operation, and typical spam mails used to be not very large (both for passing by size limit filters and for being able to send a lot of copies). That's why it can be desired to avoid checking of very large mails.

Setting of this limit says antispam module not to check mails larger than given limit and declare their spam score to zero.

Setting this limit to zero disables this feature and enables using of antispam to all mails. Be prepared for high machine load and noticeable delay in delivery if used so.

use-antivirus disable;

use-antivirus enable *channel*;

Antivirus usage mode.

If omitted, or disabled, no antivirus is enabled. In this case, neither any ANTIVIRUS global section can be present nor any MAIL-ACL and DOC-ACL can have VIRUS item specified.

<branching element> (type: **enabling**)

channel (type: **name-list** of **antivirus**, see antivirus(5))

no-mail-scanning;

Pass mail to the client without checking.

client-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to client.

altq (type: **name** of **pf-queue**, see pf-queue(5))

queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: NULL)

priority queue name (if set, used for TCP ACK without data)

server-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to server.

altq (type: **name** of **pf-queue**, see pf-queue(5))

queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: NULL)

priority queue name (if set, used for TCP ACK without data)

[End of section `pop3-proxy.command-acl` description.]

mail-acl *name* {

* from ... ;

* time ... ;

time-period-set { ... }

* command-acl ... ;

deny ... ;

accept ... ;

rule ... ;

* content-type ... ;

virus-status ... ;

* modify-header ... ;

replace ... ;

* spam-score ... ;

* header ... ;

prefix-subject ... ;

}

The first ACL on the third level decides how to handle the whole mail.

The `mail-acl` section is derived from `mail-acl` section prototype.

For detail description of it, see mod-mail-doc(5).

Changes to the `mail-acl` section:

Item `parent-acl` used as `command-acl`.

Item `direction` is not valid.

Item `size` is not valid.

Item `sender` is not valid.

Item `recipient` is not valid.

Item `recipients` is not valid.

Item `from-quarantine` is not valid.

doc-acl *name* {

* from ... ;
* time ... ;
  time-period-set { ... }
* command-acl ... ;
  deny ... ;
  accept ... ;
  rule ... ;
* size ... ;
* content-type ... ;
* mime-type ... ;
  virus-status ... ;
* modify-header ... ;
  force-doctype-ident ... ;
  replace ... ;
  html-filter ... ;
* spam-score ... ;
* header ... ;
* filename ... ;
  add-virus-names ... ;

}

The `doc-acl` section is derived from `mail-doc-acl` section prototype. For detail description of it, see mod-mail-doc(5).
Changes to the `doc-acl` section:

Item `parent-acl` used as `command-acl`.
Item `direction` is not valid.
Item `sender` is not valid.
Item `recipient` is not valid.
Item `from-quarantine` is not valid.

[End of section `pop3-proxy` description.]

SEE ALSO

configuration(7), acl(5), antivirus(5), application(5), auth(5), common(5), ipc(5), listen-on(5), log(5), mod-antispam(5), mod-mail-doc(5), monitoring(5), netio(5), pf-queue(5), source-address(5), ssl(5), time(5)

NAME

pop3-proxy.cfg — format of pop3-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **pop3-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **pop3-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

nls (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see radius(5))

ldap-tls-reqcert-mode (see ldap(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

antivirus-protocol (see antivirus(5))

virus-status (see antivirus(5))

database-source (see antivirus(5))

source-address-mode (see source-address(5))

accept-deny (see mod-html-filter(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

ssl-ver (see ssl(5))

extension-op (see ssl(5))

veri-fail-action (see ssl(5))

auth-cert-type (see ssl(5))

distrusted-cert-type (see ssl(5))

mail-reaction (see mod-mail-doc(5))

mail-fallback (see mod-mail-doc(5))

mime-header-check-type (see mod-mail-doc(5))

pop3-cmd (see pop3-proxy(5))

pop3-capa (see pop3-proxy(5))

ITEMS AND SECTIONS

Program **pop3-proxy** recognizes following items and sections:
* antispam *name* { ... }
* antivirus *name* { ... }
* fake-cert *name* { ... }
* html-filter *name* { ... }
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* mail-filter *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
* ssl-params *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* pop3-proxy *name* { ... }
  ipv6-mode ... ;

Description:

antispam *name* **{**

connection ... ;

sock-opt { ... }

altq ... ;

}

The antispam section is derived from antispam section prototype. For detail description of it, see mod-antispam(5).

antivirus *name* **{**

connection ... ;

sock-opt { ... }

timeout ... ;

comm-dir ... ;

altq ... ;

max-checked-size ... ;

icap-pass-200-with-pure-body ... ;

persistent-stream ... ;

clamav-agent { ... }

}

The `antivirus` section is derived from `antivirus` section prototype. For detail description of it, see antivirus(5).

`fake-cert` *name* {

key ... ;

auth-ca ... ;

fail-ca ... ;

* extension ... ;

purge ... ;

}

The `fake-cert` section is derived from `fake-cert` section prototype. For detail description of it, see ssl(5).

`html-filter` *name* {

* script-tag-language ... ;

replace-head-script-tags ... ;

replace-body-script-tags ... ;

* style-tag-type ... ;

replace-style-tags ... ;

* iframe-tag-src ... ;

replace-iframe-tags ... ;

* intrinsic-language ... ;

* intrinsic-hack ... ;

replace-intrinsic ... ;

* macro-language ... ;

* macro-hack ... ;

replace-macros ... ;

* uri ... ;

replace-uri ... ;

* embed-tag-type ... ;

* embed-src-hack ... ;

* embed-plugin-hack ... ;

replace-head-embed-tags ... ;

replace-body-embed-tags ... ;

* applet ... ;

replace-applets ... ;

* object ... ;

* object-classid-hack ... ;

* object-data-hack ... ;

replace-head-object-tags ... ;

replace-body-object-tags ... ;

* param-tags ... ;

replace-param ... ;

script-end-hack ... ;

}

The `html-filter` section is derived from `html-filter` section prototype. For detail description of it, see mod-html-filter(5).

`interface` *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

ldap-client-auth *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The ldap-client-auth section is derived from ldap-client-auth section prototype. For detail description of it, see ldap(5).

mail-filter *name* {

stamp-limit ... ;

stamp-filter ... ;

* unflagged-8bit ... ;

* bad-end-of-line ... ;

* invalid-header ... ;

* long-header-lines ... ;

* invalid-chars ... ;

* header-8bit-chars ... ;

* bad-boundary-chars ... ;

* bad-boundary-length ... ;

* long-body-lines ... ;

* long-encoded-lines ... ;

enc-line-len ... ;

* bad-mime-struct ... ;

* invalid-encoding ... ;

treat-rfc822-as-text ... ;

}

The mail-filter section is derived from mail-filter section prototype. For detail description of it, see mod-mail-doc(5).

oob-auth *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The oob-auth section is derived from oob-auth section prototype. For detail description of it, see auth(5).

pf-queue *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The pf-queue section is derived from pf-queue section prototype. For detail description of it, see pf-queue(5).

radius-client *name* {

nas ... ;

groups ... ;

* server ... ;

}

The radius-client section is derived from radius-client section prototype. For detail description of it, see radius(5).

resolver *name* {

* server ... ;

search ... ;

```
    preference ... ;
    edns ... ;
    conf-timeout ... ;
    initial-timeout ... ;
    final-timeout ... ;
    conn-timeout ... ;
    disable-deresolution ... ;
  }
```

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`shared-dir` *name* {

```
    path ... ;
  }
```

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

`shared-file` *name* {

```
    path ... ;
    format ... ;
  }
```

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`ssl-params` *name* {

```
    versions ... ;
    ciphers ... ;
    tcp-eof ... ;
    id ... ;
  * auth-cert ... ;
    distrusted-certs ... ;
    dont-check-crl ... ;
  * crl ... ;
    verify-peer ... ;
    cache-timeout ... ;
```

use-ticket ... ;

enable-renegotiation ... ;

fake-cert ... ;

prefer server ciphers ... ;

enable-ecdh ... ;

}

The `ssl-params` section is derived from `ssl-params` section prototype. For detail description of it, see ssl(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

***name* (type: `name` of `resolver`, see resolver(5))**

`pop3-proxy` *name* {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

ssl-session-cache { ... }

mail-pool ... ;

* session-acl *name* { ... }

* command-acl *name* { ... }

* mail-acl *name* { ... }

* doc-acl *name* { ... }

}

The `pop3-proxy` section is derived from `pop3-proxy` section prototype. For detail description of it, see pop3-proxy(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), pop3-proxy(8), acl(5), antivirus(5), auth(5), common(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), mod-antispam(5), mod-html-filter(5), mod-mail-doc(5), pf-queue(5), pop3-proxy(5), radius(5), resolver(5), source-address(5), ssl(5), sysctl(5), time(5), host-matching(7)

NAME

proxy-ng — format of proxy-ng component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **proxy-ng** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **proxy-ng** configuration directives:

yes-no (see **common**(5))

ip-version (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

source-address-mode (see **source-address**(5))

session-protocol (name-usage optional)

Protocol for handling a TCP session.

tcp-proxy (0)

TCP proxy mode, no application protocol handling.

tcp-proxy-transparent (1)

TCP proxy mode, no application protocol handling, connects to the destination address of the connection from the client unless overriden by PLUG-TO.

http-proxy (2)

HTTP proxy mode.

json-type (name-usage obligatory)

Types of JSON values

raw

Raw JSON value of any type (scalar, array, or object)

null

Null value

false

Boolean false value

true

Boolean true value

uint

Unsigned integer

fract

Fractional value with 3 decimal places

str

String value

http-version (name-usage obligatory)

Version of HTTP.

HTTP-1-0

HTTP-1-1

ITEMS AND SECTIONS

Configuration of **proxy-ng** library component consists of following prototypes:
* json-value ... ;
  log-ng { ... }
* acl-ng *name* { ... }
* session-acl-ng *name* { ... }
* http-request-acl-ng *name* { ... }
* http-doc-acl-ng *name* { ... }
* proxy-ng *name* { ... }

Description:

json-value raw *raw*;

json-value null *path*;

json-value false *path*;

json-value true *path*;

json-value uint *path num ui*;

json-value fract *path num f*;

json-value str *path string*;

A generic JSON value.

<**branching element**> (**type: json-type**)

Type of the value.

raw (**type: str**)

A raw string that will be parsed as an arbitrarily complex JSON.

path (**type: str-list**)

A path to an object element. It is the list of names of nested active level configuration JSON objects containing the value. The last is the name of the element name in the innermost object.

num ui (**type: uint64**)

A value of the object element.

num f (**type: fract**)

A value of the object element.

string (**type: str**)

A value of the object element.

Constraints:

Invalid JSON in RAW..

PATH must not be empty..

log-ng {

level ... ;

facility ... ;

file ... ;

rotate ... ;

enabled ... ;

}

The log-ng section is derived from log section prototype. For detail description of it, see log(5).

Changes to the log-ng section:

Item `mem-level` is not valid.

Item `mem-file` is not valid.

Item `syslog-failure` is not valid.

Item `data-limit` is not valid.

Item `dump-hold-time` is not valid.

Item `file` (see log(5))

Section `usec` is not valid.

Added items & subsections:

`enabled` [*val*];

Whether the log is enabled.

val **(type: `yes-no`, optional, default: yes)**

[End of section `log-ng` description.]

`acl-ng` *name* {

* is-flagged ... ;

* not-flagged ... ;

* jval ... ;

deny ... ;

accept ... ;

continue ... ;

set-flag ... ;

unset-flag ... ;

set-rule ... ;

}

Access Control List.

Constraints:

Only one of ACCEPT, DENY is allowed..

Items & subsections:

`is-flagged` *names*;

Entry condition: Tests that all flags (or rule names) in the list are set.

names **(type: `str-list`)**

`not-flagged` *names*;

Entry condition: Tests that no flags (or rule names) from the list are set.

names **(type: `str-list`)**

`jval raw` *raw*;

`jval null` *path*;

`jval false` *path*;

`jval true` *path*;

`jval uint` *path num ui*;

`jval fract` *path num f*;

jval str *path string*;

An arbitrary JSON value set if this ACL matches.

<**branching element**> (type: **json-type**)

Type of the value.

raw (type: **str**)

A raw string that will be parsed as an arbitrarily complex JSON.

path (type: **str-list**)

A path to an object element. It is the list of names of nested active level configuration JSON objects containing the value. The last is the name of the element name in the innermost object.

num ui (type: **uint64**)

A value of the object element.

num f (type: **fract**)

A value of the object element.

string (type: **str**)

A value of the object element.

Constraints:

Invalid JSON in RAW..

PATH must not be empty..

deny;

Deny communication. This is the default if no ACL containing ACCEPT is applied.

accept;

Permit communication.

continue;

Do not stop checking ACLs if this ACL matches.

set-flag *names*;

Sets flags or rule names. See also SET-RULE.

names (type: **str-list**)

unset-flag *names*;

Unset flags or rule names. See also SET-RULE.

names (type: **str-list**)

set-rule *names*;

Sets rule names. A name used in a SET-RULE will be included in log messages if it is activated by a SET-FLAG or SET-RULE. All other names used in SET-FLAG, UNSET-FLAG, IS-FLAGGED, or NOT-FLAGGED, can be set, unset, or tested, but are not logged.

names (type: **str-list**)

[End of section `acl-ng` description.]

session-acl-ng *name* {

* is-flagged ... ;

* not-flagged ... ;

* jval ... ;

deny ... ;

accept ... ;

continue ... ;

set-flag ... ;

unset-flag ... ;

set-rule ... ;

* listen-socket-id ... ;

* from ... ;

* to ... ;

protocol ... ;

source-address ... ;

plug-to ... ;

http-error { ... }

hand-off ... ;

}

Access Control List evaluated when a new TCP connection is accepted.

The session-acl-ng section is derived from acl-ng section prototype. For detail description of it, see above.
Added items & subsections:

listen-socket-id *val*;

Entry condition: LISTEN-SOCKET-ID of the listening socket.

val **(type: str)**

from *addr*;

Entry condition: Matches source IP address of a TCP connection from a client.

addr **(type: host-set)**

Set of client IP addresses or host names.

Constraints:

Regular expressions are not allowed in host set.

to *addr* [**port** *port*];

Entry condition: Matches destination IP address of a TCP connection from a client

addr **(type: host-set)**

Set of IP addresses or host names.

port *port* (type: **port-set**, optional, default: *)

Set of destination service names/port numbers.

Constraints:

Regular expressions are not allowed in host set.

protocol *val*;

Selects a protocol for handling the session.

val (type: **session-protocol**)

source-address [**client**] [**addr4** *addr4*] [**addr6** *addr6*] **cluster** [*cluster*];
source-address [**client**] [**addr4** *addr4*] [**addr6** *addr6*] [**physical**];
source-address [**client**] [**addr4** *addr4*] [**addr6** *addr6*] **no-fallback**;

Source address for outgoing connections to servers.

If omitted, the proper address of the proxy will be used, i.e. in the case of a cluster, the cluster address will be used.

If not specified by the SOURCE-PORT item, a generic port will be used.

The elements entered within this item will be used by the proxy until the first of them is applicable:

- The CLIENT keyword means the original client IP address is used. This mode will be succesful in all cases except mismatch of IP address families.

- The ADDR4/ADDR6 keyword-value pairs mean that the specified address is used for a connection of corresponding address family.

- The CLUSTER keyword means that one of cluster addresses will be used. By default, the main address of the bridge is used, however, any preferred alias address can be listed in the cluster list.- The PHYSICAL option means that the address of the physical interface is used instead of the cluster one.

- The DEFAULT option means the default behavior - i.e. using of the physical address.

- The NO-FALLBACK option means that if no other way of setting the address is acceptable, the session is rejected. Without this option, the system tries to find a suitable source IP address automatically.

client (type: **key**, optional)

addr4 *addr4* (type: **host**, optional, default: [0.0.0.0])

addr6 *addr6* (type: **host**, optional, default: [::])
<branching element> (type: **source-address-mode**, optional, default: physical)
cluster (type: **host-list**, optional, default: {})

Constraints:

Address family must respect the element's address family..

plug-to *addr*;

Final destination server.

addr (type: **sock**)

Address/port of final destination server.

If port is zero, then original port is used.

http-error {

error-document ... ;

* subst ... ;

}

Controls error responses to HTTP requests.

Items & subsections:

error-document *filename*;

Template of the HTTP error document.

filename (type: **name** of **shared-file**, see common(5))

subst *name value*;

Name/value pairs substituted to ERROR-DOCUMENT.

name (type: **str**)

value (type: **str**)

[End of section `session-acl-ng.http-error` description.]

hand-off *val*;

If YES, the proxy assumes that it connects to another proxy, not to the destination server.

val (type: **yes-no**)

[End of section `session-acl-ng` description.]

http-request-acl-ng *name* {

* is-flagged ... ;

* not-flagged ... ;

* jval ... ;

deny ... ;

accept ... ;

continue ... ;

set-flag ... ;

unset-flag ... ;

set-rule ... ;

* listen-socket-id ... ;

* from ... ;

* to ... ;

source-address ... ;

plug-to ... ;

http-error { ... }

hand-off ... ;

* req-method ... ;

* req-uri ... ;

* req-host ... ;

* req-path ... ;

* req-query ... ;

* req-version ... ;

}

Access Control List evaluated when request headers of an HTTP request are received from a client.

The `http-request-acl-ng` section is derived from `session-acl-ng` section prototype. For detail description of it, see above.
Changes to the `http-request-acl-ng` section:

Item `protocol` is not valid.

Added items & subsections:

`req-method` *val*;

The method of an HTTP request.

val **(type: `str-set`)**

`req-uri` *val*;

The request URI of an HTTP request.

val **(type: `str-set`)**

`req-host` *addr* [`port` *port*];

The host address from an HTTP request.

addr **(type: `host-set`)**
Set of IP addresses or host names.

`port` *port* (type: `port-set`, optional, default: *)
Set of destination service names/port numbers.

`req-path` *val*;

The path from an HTTP request URI.

val **(type: `str-set`)**

`req-query` *val*;

The query from an HTTP request URI.

val **(type: `str-set`)**

req-version *val*;

The HTTP version of an HTTP request.

val (type: **http-version-set**)

[End of section `http-request-acl-ng` description.]

http-doc-acl-ng *name* {

* is-flagged ... ;
* not-flagged ... ;
* jval ... ;
  deny ... ;
  accept ... ;
  continue ... ;
  set-flag ... ;
  unset-flag ... ;
  set-rule ... ;
* listen-socket-id ... ;
* from ... ;
* to ... ;
  http-error { ... }
* req-method ... ;
* req-uri ... ;
* req-host ... ;
* req-path ... ;
* req-query ... ;
* req-version ... ;
* resp-status ... ;
* resp-version ... ;

}

Access Control List evaluated when response headers of an HTTP response are received from a server.

The `http-doc-acl-ng` section is derived from `http-request-acl-ng` section prototype. For detail description of it, see above.

Changes to the `http-doc-acl-ng` section:

Item `source-address` is not valid.

Item `plug-to` is not valid.

Item `hand-off` is not valid.

Added items & subsections:

resp-status *val*;

The status code of an HTTP response.

val (type: **uint16-set**)

resp-version *val*;

The HTTP version of an HTTP response.

val (type: **http-version-set**)

[End of section `http-doc-acl-ng` description.]

proxy-ng *name* {

phase ... ;

* tag ... ;

use-resolver ... ;

nodaemon ... ;

app-user ... ;

log-debug { ... }

log-stats { ... }

resolver-ng { ... }

listen-on { ... }

tcpserver { ... }

* cfg-begin ... ;

* cfg-end ... ;

* jval ... ;

log-audit { ... }

* session-acl *name* { ... }

http-proxy { ... }

}

New unified multi-protocol proxy

The proxy-ng section is derived from proxy section prototype. For detail description of it, see application(5).
Changes to the proxy-ng section:

Section `exclude-log-debug` is not valid.

Section `log-debug` used as `exclude-log-debug`.

Section `exclude-log-stats` is not valid.

Section `log-stats` used as `exclude-log-stats`.

Item `cfg-resolution` is not valid.

Section `monitoring` is not valid.

Section `stats-daily` is not valid.

Section `stats-weekly` is not valid.

Section `stats-monthly` is not valid.

Item `singleproc` is not valid.

Item `idle-timeout` is not valid.

Item `run-block-sigalrm` is not valid.

Section `exclude-listen-on` is not valid.

Section `listen-on` used as `exclude-listen-on`.

Section `exclude-tcpserver` is not valid.

Section `tcpserver` used as `exclude-tcpserver`.

Section `udpserver` is not valid.

Item `source-address` is not valid.

Section `doctype-identification` is not valid.

Added items & subsections:

`log-debug` {

level ... ;

facility ... ;

file ... ;

rotate ... ;

enabled ... ;

}

The `log-debug` section is derived from `log-ng` section prototype.
For detail description of it, see above.
Item `facility` (see log(5))

Element `value` is optional, default: 5.

`log-stats` {

level ... ;

facility ... ;

file ... ;

rotate ... ;

enabled ... ;

}

The `log-stats` section is derived from `log-ng` section prototype.
For detail description of it, see above.
Changes to the `log-stats` section:

Only values NORMAL (log top-level protocol only) and DEBUG (log all protocols) are allowed..

Item `facility` (see log(5))

Element `value` is optional, default: 6.

`resolver-ng` {

cache-size ... ;

refresh-time ... ;

threads ... ;

}

Attributes for configuration of domain names resolution.

Items & subsections:

`cache-size` [*val*];

Number of cached resolved host names or IP addresses

val **(type: `uint32`, optional, default: 10000)**

`refresh-time` [*val*];

Time (seconds) after which cached DNS results are refreshed if they are used or deleted if they are unused.

val **(type: `fract`, optional, default: 300)**

`threads` [*val*];

Number of threads (parallel queries) used by DNS resolver.

val **(type: `uint8`, optional, default: 20)**

[End of section `proxy-ng.resolver-ng` description.]

`listen-on` {

* non-transparent ... ;

* transparent ... ;

}

Items & subsections:

`non-transparent` *addr* **[`version`** *version***]** **`port`** *port*

`listen-socket-id` *listen-socket-id*;

Sockets to bind for non-transparent connections.

addr **(type: `host`)**

Address to be bound

`version` *version* **(type: `ip-version`, optional, default: undefined)**

IP version selection

`port` *port* **(type: `port`)**

Port to be bound (lowest)

`listen-socket-id` *listen-socket-id* **(type: `str`)**

ID of the listening socket.

`transparent listen-socket-id` *listen-socket-id*;

Sockets to handle transparent connections.

listen-socket-id *listen-socket-id* (type: **str**)

ID of the listening socket.

[End of section `proxy-ng.listen-on` description.]

tcpserver {

max-sessions ... ;

queue-size ... ;

worker-threads ... ;

}

General TCP server parameters.

Items & subsections:

max-sessions [*value*];

Maximum number of concurrent sessions from clients.

value (type: **uint16**, optional, default: **1500**)

queue-size [*value*];

Queue length for listen(2) syscall, 0 for system default.

value (type: **uint16**, optional, default: **0**)

worker-threads *val*;

Number of threads for handling network communication. If not set, a default value is determined according to the number of CPUs.

val (type: **uint8**)

[End of section `proxy-ng.tcpserver` description.]

cfg-begin *filename*;

Configuration files that will be read before the file generated from the CML configuration.

filename (type: **name** of **shared-file**, see common(5))

cfg-end *filename*;

Configuration files that will be read after the file generated from the CML configuration.

filename (type: **name** of **shared-file**, see common(5))

jval raw *raw*;

jval null *path*;

jval false *path*;

jval true *path*;

jval uint *path num ui*;

jval fract *path num f*;

jval str *path string*;

An arbitrary JSON value which is set unconditionally upon proxy startup.

<branching element> (type: `json-type`)

Type of the value.

***raw* (type: `str`)**

A raw string that will be parsed as an arbitrarily complex JSON.

***path* (type: `str-list`)**

A path to an object element. It is the list of names of nested active level configuration JSON objects containing the value. The last is the name of the element name in the innermost object.

***num ui* (type: `uint64`)**

A value of the object element.

***num f* (type: `fract`)**

A value of the object element.

***string* (type: `str`)**

A value of the object element.

Constraints:

Invalid JSON in RAW..

PATH must not be empty..

`log-audit` {

level ... ;

facility ... ;

file ... ;

rotate ... ;

enabled ... ;

}

The `log-audit` section is derived from `log-ng` section prototype.
For detail description of it, see above.
Changes to the `log-audit` section:

Only values NORMAL (all audit messages) and ERROR (security violations attempts only) are allowed..

Item `facility` (see log(5))

Element `value` is optional, default: 5.

`session-acl` *name* {

* is-flagged ... ;

* not-flagged ... ;

* jval ... ;

deny ... ;

accept ... ;

continue ... ;

set-flag ... ;

unset-flag ... ;
set-rule ... ;
* listen-socket-id ... ;
* from ... ;
* to ... ;
protocol ... ;
source-address ... ;
plug-to ... ;
http-error { ... }
hand-off ... ;
}

The `session-acl` section is derived from `session-acl-ng` section prototype. For detail description of it, see above.

`http-proxy` {

* request-acl *name* { ... }
* doc-acl *name* { ... }

}

Control of sessions handled as HTTP.

Items & subsections:

`request-acl` *name* {

* is-flagged ... ;
* not-flagged ... ;
* jval ... ;
deny ... ;
accept ... ;
continue ... ;
set-flag ... ;
unset-flag ... ;
set-rule ... ;
* listen-socket-id ... ;
* from ... ;
* to ... ;
source-address ... ;
plug-to ... ;
http-error { ... }
hand-off ... ;
* req-method ... ;
* req-uri ... ;
* req-host ... ;
* req-path ... ;

* req-query ... ;
* req-version ... ;
}

The `request-acl` section is derived from `http-request-acl-ng` section prototype. For detail description of it, see above.

`doc-acl` *name* {
* is-flagged ... ;
* not-flagged ... ;
* jval ... ;
  deny ... ;
  accept ... ;
  continue ... ;
  set-flag ... ;
  unset-flag ... ;
  set-rule ... ;
* listen-socket-id ... ;
* from ... ;
* to ... ;
  http-error { ... }
* req-method ... ;
* req-uri ... ;
* req-host ... ;
* req-path ... ;
* req-query ... ;
* req-version ... ;
* resp-status ... ;
* resp-version ... ;
}

The `doc-acl` section is derived from `http-doc-acl-ng` section prototype. For detail description of it, see above.

[End of section `proxy-ng.http-proxy` description.]

[End of section `proxy-ng` description.]

SEE ALSO

configuration(7), listen(2), application(5), common(5), log(5), source-address(5)

NAME

radius — format of radius component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **radius** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **radius** configuration directives:

radius-attr (name-usage optional)

RADIUS attribute names

reply-message (18)

state (24)

class (25)

vendor-specific (26)

ITEMS AND SECTIONS

Configuration of **radius** library component consists of following prototypes:
* radius-client *name* { ... }

Description:

radius-client *name* **{**

nas ... ;

groups ... ;

* server ... ;

}

RADIUS Client Attributes.

Client identification and list of servers used for authentication.

Constraints:

Item NAS required.

Item SERVER required.

Items & subsections:

nas *id*;

RADIUS NAS identification (identification of the RADIUS client)

id **(type: str)**

groups [*attr*];

Attribute containing list of groups in RAD ACCESS ACCEPT (default is Reply-Message)

attr **(type: radius-attr, optional, default: reply-message=18)**

server *host* [**port** *port*] *secret* [*timeout* [*tries*]];

Definition of RADIUS server

host **(type: host)**

Server host name

port *port* **(type: port, optional, default: 0)**

Server port (0 means take from /etc/services)

secret **(type: str)**

Shared secret

timeout **(type: uint16, optional, default: 5)**

Timeout for receiving replies (seconds)

tries **(type: uint16, optional, default: 5)**

Maximum number of repeated requests before giving up

[End of section `radius-client` description.]

SEE ALSO

configuration(7)

NAME

resolver — format of resolver component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **resolver** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **resolver** configuration directives:

enabling (see **common**(5))

ip-version (see **common**(5))

dns-type (name-usage optional)

- **none** (0)
- **A** (1)
- **NS** (2)
- **MD** (3)
- **MF** (4)
- **CNAME** (5)
- **SOA** (6)
- **MB** (7)
- **MG** (8)
- **MR** (9)
- **NULL** (10)
- **WKS** (11)
- **PTR** (12)
- **HINFO** (13)
- **MINFO** (14)
- **MX** (15)

TXT (16)

RP (17)

AFSDB (18)

X25 (19)

ISDN (20)

RT (21)

NSAP (22)

NSAP-PTR (23)

SIG (24)

KEY (25)

PX (26)

GPOS (27)

AAAA (28)

LOC (29)

NXT (30)

EID (31)

NIMLOC (32)

SRV (33)

ATMA (34)

NAPTR (35)

KX (36)

CERT (37)

A6 (38)

DNAME (39)

SINK (40)

OPT (41)

APL (42)

DS (43)

SSHFP (44)

IPSECKEY (45)

RRSIG (46)

NSEC (47)

DNSKEY (48)

NSEC3 (50)

NSEC3PARAM (51)

TLSA (52)

SPF (99)

TKEY (249)

TSIG (250)

IXFR (251)

AXFR (252)

MAILB (253)

MAILA (254)

ANY (255)

CAA (257)

dns-class (name-usage optional)

NONE (0)

IN (1)

CH (3)

HS (4)

ANY (255)

dns-opcode (name-usage optional)

QUERY (0)

IQUERY (1)

STATUS (2)

NOTIFY (4)

UPDATE (5)

dns-response (name-usage optional)

NoError (0)

FormErr (1)

ServFail (2)

NXDomain (3)

NotImp (4)

Refused (5)

YXDomain (6)

YXRRSet (7)

NXRRSet (8)

NotAuth (9)

NotZone (10)

BADVERS (16)

BADSIG (17)

BADKEY (18)

BADTIME (19)

BADMODE (20)

BADNAME (21)

BADALG (22)

dns-qaction (name-usage obligatory)

Action used for particular query received.

abort

Query is aborted, no answer.

deny

Query is denied, reply by given code.

resolve

Query is resolved from root accepting trusted answers only.

forward

Query is forwarded to DNS server.

fake

Query is replied according to configuration setting.

dns-raction (name-usage obligatory)

Action used for particular resource record received in reply.

abort

Query is aborted, no answer.

deny

Query is denied, reply by given code.

permit

Record is added to reply.

remove

Record is removed from reply.

dns-fake (name-usage obligatory)

RR types with faking implemented.

A

NS

CNAME

PTR

MX

AAAA

xfr-mode (name-usage obligatory)

Zone transfer modes.

keep

Use the same format as originator.

separated

Use more messages, one RR per message.

aggregated

Use one message with all RRs.

ITEMS AND SECTIONS

Configuration of **resolver** library component consists of following prototypes:
* ns-list *name* { ... }
* resolver *name* { ... }
use-resolver ... ;
cfg-resolution ... ;

Description:

ns-list *name* **{**

* server ... ;

}

This section defines set of nameservers used by dns-proxy for forwarding.

No "default server set" exist. Typical set of public internet root servers can be found in file samples/root-servers.cml that you can include into your configuration and use here. See instructions in the file.

Constraints:

At least one server must be specified.

Items & subsections:

server *name addr* [**port** *port*];

Single server description.

name **(type: str)**

Domain name of server

addr (type: **host-list**)

List of server IP addresses

port *port* (type: **port**, optional, default: **53**)

[End of section `ns-list` description.]

resolver *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

Domain Names Resolver Configuration.

This prototype defines Kernun resolver parameters. It can be used generally, for any Kernun application, or redefined for particular proxy.

The `resolver` section is derived from `ns-list` section prototype.
For detail description of it, see above.
Added items & subsections:

search *names*;

Domain search list.

If omitted, system domain name is used.

names (type: **str-list**)

Constraints:

Search list must not be empty.

preference *versions*;

IP address versions preference.

This item controls selection of IPv4 and IPv6 addresses obtained by resolving a name. If not set, the default value depends on global Kernun configuration: if no interface has IPv6 address, only IPv4 addresses are used, otherwise the default behavior according to the RFC 3484 is used.

versions (type: **ip-version-list**)

Ordered list of versions.

Constraints:

Version list must contain one or two items of ipv4/ipv6.

edns [*support*];

EDNS support.

support (type: **enabling**, optional, default: enable)

conf-timeout [*seconds*];

Timeout for resolution of each domain name in configuration.

seconds (type: **fract**, optional, default: 15)

initial-timeout [*seconds*];

Timeout for initial attempt to deresolve client address.

If this deresolution fails, client address will be logged without name till the SESSION-END message.

seconds (type: **fract**, optional, default: 0.200)

final-timeout [*seconds*];

Timeout used for deresolving client address immediately before logging the SESSION-END message (if the first attempt of client deresolution failed due to INITIAL-TIMEOUT).

seconds (type: **fract**, optional, default: 5)

conn-timeout [*seconds*];

Timeout to resolve connection critical addresses.

This timeout will be used for any resolution necessary for successful progress of connection, e.g. server address.

seconds (type: **fract**, optional, default: 30)

disable-deresolution;

Flag to switch off IP addresses deresolution.

[End of section `resolver` description.]

use-resolver *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name (type: **name** of **resolver**, see above)

cfg-resolution [*max-addrs* [*min-ttl* [*def-ttl* [*max-ttl* [*hosts-ttl* [*pool-dir*]]]]]];

Attributes for resolution of domain names in configuration.

max-addrs (type: **uint8**, optional, default: 10)

Maximum of addresses per a single domain name.

min-ttl **(type: `uint32`, optional, default: 10)**

Minimum TTL accepted, used instead of too small TTL values (e.g. 0).

def-ttl **(type: `uint32`, optional, default: 1m)**

Default TTL used in case of unsuccessful DNS resolution.

max-ttl **(type: `uint32`, optional, default: 1d)**

Maximum TTL accepted, used instead of large TTL values.

hosts-ttl **(type: `uint32`, optional, default: 1d)**

TTL used for names in /etc/hosts.

pool-dir **(type: `str`, optional, default: "/tmp")**

Directory for temporary files used to share results.

SEE ALSO

configuration(7), common(5)

NAME

router — format of router component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **router** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **router** configuration directives:

yes-no (see **common**(5))

name-selection (see **common**(5))

export-import-mode (name-usage obligatory)

- **none**
- **filter**
- **all**

ospf-authentication (name-usage obligatory)

- **none**
- **simple**
- **cryptographic**

ospf-area-id-mode (name-usage obligatory)

- **number**
- **dotted**

ITEMS AND SECTIONS

Configuration of **router** library component consists of following prototypes:

export-import-routes ... ;
router-protocol { ... }
router { ... }
bird4 { ... }
bird6 { ... }

Description:

export-import-routes none;

export-import-routes filter *filter*;

export-import-routes [all];

Interchange of routes between protocol and engine.

<**branching element**> (**type: export-import-mode**, **optional**, **default: all**)
Interchange mode.

filter (**type: str**)
Filter rules.

router-protocol {

import ... ;
export ... ;
scan ... ;
* area *name* { ... }
* raw ... ;
}

Protocol configuration.

Items & subsections:

import none;

import filter *filter*;

import [all];

Import routes from protocol module to routing engine.

<**branching element**> (**type: export-import-mode**, **optional**, **default: all**)
Interchange mode.

filter (**type: str**)
Filter rules.

export none;

export filter *filter*;

export [all];

Export routes from routing engine to protocol module.

<**branching element**> **(type: export-import-mode, optional, default: all)**

Interchange mode.

filter **(type: str)**

Filter rules.

scan [time *time***];**

Period of scanning protocol route sources.

time *time* **(type: uint16, optional, default: 10)**

area *name* **{**

id ... ;

* interface *name* { ... }

stub ... ;

* raw ... ;

}

OSPF area definition.

Constraints:

Backbone area cannot be stub.

Items & subsections:

id [number] [*id***];**

id dotted [*addr***];**

Area identification

<**branching element**> **(type: ospf-area-id-mode, optional, default: number)**

id **(type: uint32, optional, default: 0)**

addr **(type: str, optional, default: "")**

Constraints:

Dotted area ID must comply with IPv4 address format.

interface *name* **{**

* iface ... ;

cost ... ;

hello ... ;

retransmit ... ;

priority ... ;

wait ... ;

dead ... ;

authentication ... ;

stub ... ;
* raw ... ;
}

Interface parameters definition.

Constraints:

At least one IFACE item required..

Items & subsections:

`iface any;`

`iface [name]` *name*;

Interface name.
<branching element> (type: **`name-selection`**, optional, default: name)

name (type: **`name`** of **`interface`**, see interface(5))

`cost [`*metric*`];`

Interface metric.

metric (type: **`uint32`**, optional, default: 10)

`hello [`*seconds*`];`

Hello interval.

Routers on the same network need to have the same value.

seconds (type: **`uint16`**, optional, default: 10)

`retransmit [`*seconds*`];`

Retransmition of unacknowledged updates interval.

seconds (type: **`uint16`**, optional, default: 5)

`priority [`*prty*`];`

Designated router selection priority.

prty (type: **`uint16`**, optional, default: 1)

`wait [`*seconds*`];`

Startup wait time.

seconds (type: **`uint16`**, optional, default: 40)

`dead [`*seconds*`];`

Neighbor death timeout.

seconds (type: **`uint16`**, optional, default: 40)

`authentication [none];`

authentication simple *password*;

authentication cryptographic *password*;

Authentication mode.
<**branching element**> (**type: ospf-authentication**, **optional**, **default: none**)

password (**type: str**)

Constraints:

Password can be max. 8 (simple) or 16 (cryptographic) charaters long.

stub;

Stub interface mode.

raw *line*;

Raw interface line.

line (**type: str**)

[End of section `router-protocol.area.interface` description.]

stub;

Stub area mode.

raw *line*;

Raw area line.

line (**type: str**)

[End of section `router-protocol.area` description.]

raw *line*;

Raw protocol line.

line (**type: str**)

[End of section `router-protocol` description.]

router {

phase ... ;

* tag ... ;

use-id ... ;

direct { ... }

kernel { ... }

device { ... }

static { ... }

ospf { ... }

* raw ... ;

}

Routing daemon configuration.

Items & subsections:

phase [*number***];**

Application Startup Phase.

number **(type: uint8, optional, default: 40)**

Phase number; the lower one, the earlier start.

tag ***value*****;**

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: str)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

use-id ***iface*****;**

Router identification.

iface **(type: name of interface, see interface(5))**

Interface name, its IPv4 address will be used as router ID.

direct {

* interface ... ;

}

Interfaces under direct control of BIRD.

Items & subsections:

interface ***name*****;**

Interface definition.

name **(type: name of interface, see interface(5))**

[End of section `router.direct` description.]

kernel {

import ... ;

export ... ;

scan ... ;

* raw ... ;

persist ... ;

}

Kernel routing table interaction description.

The `kernel` section is derived from `router-protocol` section prototype. For detail description of it, see above.
Changes to the `kernel` section:

Section `area` is not valid.

Added items & subsections:

`persist` *enabled*;

Keeping BIRD routes at exit.

enabled **(type: `yes-no`)**

[End of section `router.kernel` description.]

`device` {

import ... ;

scan ... ;

* raw ... ;

}

Network interfaces supervising description.

The `device` section is derived from `router-protocol` section prototype. For detail description of it, see above.
Changes to the `device` section:

Item `export` is not valid.

Section `area` is not valid.

`static` {

import ... ;

export ... ;

* raw ... ;

}

The `static` section is derived from `router-protocol` section prototype. For detail description of it, see above.
Changes to the `static` section:

Item `scan` is not valid.

Section `area` is not valid.

`ospf` {

import ... ;

export ... ;

* area *name* { ... }

* raw ... ;

rfc1583compat ... ;

}

The `ospf` section is derived from `router-protocol` section prototype. For detail description of it, see above.

Changes to the `ospf` section:

Item `scan` is not valid.

Added items & subsections:

`rfc1583compat`;

Routing table calculation according RFC 1583.

[End of section `router.ospf` description.]

`raw` *line*;

Raw router line.

line **(type: `str`)**

[End of section `router` description.]

`bird4 {`

phase ... ;

* tag ... ;

use-id ... ;

direct { ... }

kernel { ... }

device { ... }

static { ... }

ospf { ... }

* raw ... ;

}

The `bird4` section is derived from `router` section prototype. For detail description of it, see above.

`bird6 {`

phase ... ;

* tag ... ;

use-id ... ;

direct { ... }

kernel { ... }

device { ... }

static { ... }

ospf { ... }

* raw ... ;

}

The `bird6` section is derived from `router` section prototype. For detail description of it, see above.
Changes to the `bird6` section:

USE-ID is mandatory in IPv6.

Section `ospf.area.interface` (see above)

Item `authentication` is not valid.

SEE ALSO

configuration(7), common(5), interface(5)

NAME

rtadvd — format of rtadvd component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **rtadvd** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **rtadvd** configuration directives:

yes-no (see **common**(5))

ITEMS AND SECTIONS

Configuration of **rtadvd** library component consists of following prototypes:

rtadvd { ... }

Description:

rtadvd {

phase ... ;

* tag ... ;

default-params { ... }

}

IPv6 router advertisements daemon parameters.

The /etc/rtadvd.conf file consists of parameters specified here and in particular INTERFACE.IPV6-RTADV sections.

Items & subsections:

phase [*number*];

Application Startup Phase.

number **(type: uint8, optional, default: 30)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: `str`)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

`default-params` {

managed-address ... ;

other-stateful ... ;

* raw ... ;

}

Common settings for all interfaces.

The `default-params` section is derived from `ipv6-rtadv` section prototype. For detail description of it, see interface(5).

Changes to the `default-params` section:

Item `enable` is not valid.

[End of section `rtadvd` description.]

SEE ALSO

configuration(7), common(5), interface(5), rtadvd.conf(5), rtadvd(8)

NAME

sip-proxy — format of sip-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **sip-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **sip-proxy** configuration directives:

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

auth-method (see **auth**(5))

source-address-mode (see **source-address**(5))

transparency (see **acl**(5))

user-auth-spec (see **acl**(5))

doctype-ident-method (see **acl**(5))

listen-on-sock (see listen-on(5))

sip-cmd (name-usage obligatory)

- **none**
- **ACK**
- **BYE**
- **CANCEL**
- **INFO**
- **INVITE**
- **MESSAGE**
- **NOTIFY**
- **OPTIONS**
- **PRACK**
- **PUBLISH**
- **REFER**
- **REGISTER**
- **SUBSCRIBE**
- **UPDATE**

peer (name-usage obligatory)

- **none**
- **client**
- **server**
- **both**

message (name-usage obligatory)

- **request**
- **response**

ITEMS AND SECTIONS

Configuration of **sip-proxy** library component consists of following prototypes:
* sip-proxy *name* { ... }

Description:

sip-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

source-address ... ;

doctype-identification { ... }

queue-size ... ;

hash-salt ... ;

ctrl-conn { ... }

data-conn { ... }

map-file ... ;

timeouts { ... }

sessions-table-size ... ;

sockets-table-size ... ;

* keepalive ... ;

* session-acl *name* { ... }

* request-acl *name* { ... }

}

This section defines SIP-proxy attributes.

The `sip-proxy` section is derived from `proxy` section prototype. For detail description of it, see application(5).

Changes to the `sip-proxy` section:

Section `tcpserver` is not valid.

Section `udpserver` is not valid.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one REQUEST-ACL must be specified.

SIP Registration Yellow Pages File name must be specified.

Sessions table size must be specified.

Sockets table size must be specified.

Section `monitoring` (see monitoring(5))

Item `aproxy-user` is not valid.

Item `data` used as `uri`.

Item `idle-timeout` (see application(5))

Element `seconds` is optional, default: 60.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 5060.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 5060.

Added items & subsections:

`queue-size` [*value*];

Queue length for listen(2) syscall.

value **(type: `uint8`, optional, default: 4)**

`hash-salt` [*text*];

Private URI hashing salt.

When publishing data with private addresses (like our clients' Connect URI), the SIP proxy hashes it for security reasons. This hashing can be easily broken by trying all possible private addresses. Therefore, the admin can define a site-specific string that will be added to hashed address to disable this attack.

text **(type: `str`, optional, default: `""`)**

`ctrl-conn` {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Control connection options.

The `ctrl-conn` section is derived from `sock-opt` section prototype.
For detail description of it, see netio(5).

`data-conn` {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Data connection options.

The `data-conn` section is derived from `sock-opt` section prototype.
For detail description of it, see netio(5).

`map-file` *name*;

SIP Registration (Yellow Pages) File.

name **(type: `str`)**

`timeouts` {

timer-c ... ;

timer-dj ... ;

}

Timeout set.

Items & subsections:

`timer-c` [*seconds*];

Proxy transaction timeout (RFC 3261 Timer C).

This timer is used to prevent situations when a request never generates a final response. When this timer fires, the session is cancelled.

seconds **(type: `uint32`, optional, default: 3m)**

`timer-dj` [*seconds*];

Wait time for message retransmits (RFC 3261 Timer D,J).

This timer is used to control removing of sessions from the table after carrying the last message (ACK or final response to non INVITE request).

seconds **(type: `uint32`, optional, default: 32)**

[End of section `sip-proxy.timeouts` description.]

`sessions-table-size` *number*;

Maximal number of active SIP sessions.

The necessary number can be estimated as number of phones times 3 (client's REGISTER, registrar's OPTIONS and call).

number **(type: `uint16`)**

Constraints:

Number of sessions must not be zero.

sockets-table-size *number*;

Maximal number of active SIP and SDP sockets.

This number must cover two sockets for every simultaneous TCP session plus two sockets for every active media channel of every simultaneous call.

number **(type: uint16)**

Constraints:

Number of sockets must not be zero.

keepalive *peer* [*period* [*content*]];

Sending keepalive packets to peer.

These items enable sending of short packets used for keeping various state tables along the path to the server alive.

peer **(type: host-set)**

Set of hosts interested in receiving such packets.

period **(type: uint16, optional, default: 20)**

content **(type: str, optional, default: <NULL>)**

Packet content, four zero bytes is used by default.

session-acl *name* **{**

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

idle-timeout ... ;

source-address ... ;

plug-to ... ;

hide ... ;

reject-gracefully ... ;

}

The first level ACL decides only between acceptation and denial of the incoming datagram/connection.

The session-acl section is derived from acl-1 section prototype.

For detail description of it, see acl(5).

Changes to the session-acl section:

Item `user` is not valid.

Item `auth` is not valid.

Item `idle-timeout-peer` is not valid.

Added items & subsections:

`hide` [*peer*];

Peer real address replacing by proxy one.

peer **(type: `peer`, optional, default: none)**

`reject-gracefully`;

Graceful rejecting session according to the RFC.

By default, the proxy ignores packets that are not correct session-initiating ones, i.e. invalid requests, correct requests for unknown servers, etc. This behavior can prevent against DoS attacks.

Sometimes, it may be useful to handle such requests gracefully, i.e. to send an answer and wait for the time specified in the RFC. This item will switch this function on. However, it is highly recommended to enable this feature solely for clients from secure (local) network.

[End of section `sip-proxy.session-acl` description.]

`request-acl` *name* {

* from ... ;

* user ... ;

* time ... ;

time-period-set { ... }

* session-acl ... ;

deny ... ;

accept ... ;

rule ... ;

plug-to ... ;

* request-method ... ;

* request-uri ... ;

}

The second level ACL decides about details or processing based on request URI.

The `request-acl` section is derived from `acl-2` section prototype.

For detail description of it, see acl(5).

Changes to the `request-acl` section:

Item `server` is not valid.

Item `parent-acl` used as `session-acl`.

Item `doctype-ident-order` is not valid.

Added items & subsections:

`plug-to` *addr*;

Final destination server.

addr **(type: `sock`)**

Address/port of final destination server.

If port is zero, then original port is used.

request-method *val*;

Entry condition - matching request methods.

val (**type: str-set**)

request-uri *val*;

Entry condition - matching the whole request URI.

Proxy URIs have form sip:[<USER>@]<HOST>[:PORT], e.g., sip:sip.tns.cz:5061.

val (**type: str-set**)

[End of section `sip-proxy.request-acl` description.]

[End of section `sip-proxy` description.]

SEE ALSO

configuration(7), listen(2), acl(5), application(5), auth(5), common(5), listen-on(5), log(5), monitoring(5), netio(5), source-address(5), time(5)

NAME

sip-proxy.cfg — format of sip-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **sip-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **sip-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see **radius**(5))

ldap-tls-reqcert-mode (see **ldap**(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

peer (see sip-proxy(5))

ITEMS AND SECTIONS

Program **sip-proxy** recognizes following items and sections:
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* sip-proxy *name* { ... }
  ipv6-mode ... ;

Description:

interface *name* **{**

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`ldap-client-auth` *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

`oob-auth` *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The **oob-auth** section is derived from **oob-auth** section prototype. For detail description of it, see auth(5).

radius-client *name* {

nas ... ;

groups ... ;

* server ... ;

}

The **radius-client** section is derived from **radius-client** section prototype. For detail description of it, see radius(5).

resolver *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The **resolver** section is derived from **resolver** section prototype. For detail description of it, see resolver(5).

shared-dir *name* {

path ... ;

}

The **shared-dir** section is derived from **shared-dir** section prototype. For detail description of it, see common(5).

shared-file *name* {

path ... ;

format ... ;

}

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`sip-proxy` *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

source-address ... ;

doctype-identification { ... }

queue-size ... ;

hash-salt ... ;

ctrl-conn { ... }

data-conn { ... }

map-file ... ;

timeouts { ... }

sessions-table-size ... ;

sockets-table-size ... ;

* keepalive ... ;

* session-acl *name* { ... }

* request-acl *name* { ... }

}

The `sip-proxy` section is derived from `sip-proxy` section prototype. For detail description of it, see sip-proxy(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), sip-proxy(8), acl(5), auth(5), common(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), radius(5), resolver(5), sip-proxy(5), source-address(5), sysctl(5), time(5)

NAME

smtp-proxy — format of smtp-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **smtp-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **smtp-proxy** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

auth-method (see **auth**(5))

virus-status (see **antivirus**(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

listen-on-sock (see listen-on(5))

smtp-error (see mod-mail-doc(5))

mime-header-check-type (see mod-mail-doc(5))

smtp-cmd (name-usage obligatory)

SMTP commands

NONE

HELO

EHLO

MAIL

RCPT

DATA

RSET

ETRN

TURN

VRFY

EXPN

HELP

NOOP

SEND

SAML

SOML

VERB

AUTH

STARTTLS

QUIT

smtp-size-usage (name-usage obligatory)

disable

input

ssl-startup-mode (name-usage obligatory)

Modes of SSL startup

session

SSL started immediately

command

SSL started after STARTTLS command

postfix-security-level (name-usage obligatory)

Postfix SMTP agent security levels.

none

No TLS.

may

Opportunistic TLS.

encrypt

Mandatory TLS encryption.

dane

Opportunistic DANE TLS.

dane-only

Mandatory DANE TLS.

fingerprint

Certificate fingerprint verification.

verify

Mandatory server certificate verification.

secure

Secure-channel TLS.

postfix-transport-map-mode (name-usage obligatory)

Postfix SMTP agent TRANSPORT-MAP modes.

none

No transport maps used.

relay

All forwarder domains mapped to the relay host.

fallback

All forwarder domains mapped to the relay host, rest of the world to the fallback socket.

extern

External (unhashed) map.

smtp-err-switch (name-usage obligatory)

ok

error

ignore-err

spf-result (name-usage obligatory)

White-listing (Sender Policy Framework) results.

None

Not enough information for checking available.

Neutral

Domain owner stated no decision about this client.

Pass

Client is authorized.

Fail

Client is not authorized.

SoftFail

Client is neither authorized nor strongly rejected.

TempError

Temporary error occured while checking client.

PermError

SPF record cannot be correctly interpreted.

spf-modes (name-usage obligatory)

SPF checking modes.

sender-only

Check only sender domain.

highest-mx

Try to check sender domain and in case of no SPF, check domain of the highest priority sender MX.

check-all

Try to check sender domain and in case of no SPF, check all alternate domains from the list.

matching-mx

Try to check sender domain and in case of no SPF, check domain of the highest priority sender MX that matches any of the domains in the list.

ITEMS AND SECTIONS

Configuration of **smtp-proxy** library component consists of following prototypes:

smtp-limit { ... }
smtp-agent { ... }
* smtp-forwarder *name* { ... }
* smtp-arg-check ... ;
smtp-reply ... ;
* mailbox ... ;
grey-listing { ... }
* smtp-proxy *name* { ... }

Description:

`smtp-limit {`

soft ... ;

hard ... ;

}

This section defines two-level (soft/hard) SMTP limitation.

Items & subsections:

`soft` **[***val* **[***text***]];**

Soft-limit.

Reaching this limit causes error state (response to client).

val **(type: `uint64`, optional, default: 0)**

Limitation value.

Value of zero disables soft-checking of particular limitation.

text **(type: `str`, optional, default: <NULL>)**

Error message.

If omitted, default reply is used.

`hard` **[***val* **[***text***]];**

Hard-limit.

Reaching this limit causes immediate session termination.

val **(type: `uint64`, optional, default: 0)**

Limitation value.

Value of zero disables hard-checking of particular limitation.

text **(type: `str`, optional, default: <NULL>)**

Error message.

If omitted, default reply is used.

[End of section `smtp-limit` description.]

smtp-agent {

phase ... ;

* tag ... ;

relayhost ... ;

source-address ... ;

myhostname ... ;

smtp-helo-name ... ;

myorigin ... ;

inet-protocol ... ;

relay-domains ... ;

mydestinations ... ;

mynetworks ... ;

message-size-limit ... ;

bounce-size-limit ... ;

bounce-queue-lifetime ... ;

delay-warning-time ... ;

tls { ... }

* set-var ... ;

master-cf ... ;

smtpd-option ... ;

transport-map ... ;

}

Local MTA definition.

Constraints:

Relayhost must be defined for automatic transport maps.

MASTER-CF must be specified.

Items & subsections:

phase [*number*];

Application Startup Phase.

***number* (type: `uint8`, optional, default: 40)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value (type: **str**)

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

relayhost *next-hop*;

MTA next-hop relay.

If omitted, MTA will deliver mail according to RCPTs DNS resolution.

next-hop (type: **sock**)

source-address [**client**] [**addr4** *addr4*] [**addr6** *addr6*] **cluster** [*cluster*];
source-address [**client**] [**addr4** *addr4*] [**addr6** *addr6*] [**physical**];
source-address [**client**] [**addr4** *addr4*] [**addr6** *addr6*] **no-fallback**;

Source address of connections outgoing from forwarder to next MTA.

client (type: **key**, optional)

addr4 *addr4* (type: **host**, optional, default: [0.0.0.0])

addr6 *addr6* (type: **host**, optional, default: [::])
<branching element> (type: **source-address-mode**, optional, default: physical)
cluster (type: **host-list**, optional, default: {})

Constraints:

Address family must respect the element's address family..

Using of client source address is not possible.

Using of cluster source address is not possible.

myhostname *hostname*;

Official hostname.

This value is copied to the main.cf 'myhostname' variable. If omitted, the regular hostname is used.

hostname (type: **str**)

smtp-helo-name *hostname*;

Name used in HELO/EHLO command.

This value is copied to the main.cf 'smtp helo name' variable. If omitted, the regular hostname is used instead.

hostname (type: **str**)

myorigin *hostname*;

Host name used for locally posted mails.

This value is copied to the main.cf 'myorigin' variable. If omitted, the regular hostname is used.

hostname (type: **str**)

inet-protocol *version*;

Internet protocol version support.

If omitted, IPv4 is enabled and IPv6 too, if supported.

version (type: **ip-version**)

relay-domains *domains*;

Destinations for which this agent relays mails to.

domains (type: **str**)

mydestinations *domains*;

Destinations for which this agent accepts mails.

domains (type: **str**)

mynetworks *networks*;

List of trusted remote SMTP clients.

If omitted, the 127.0.0.0/8 network is used.

networks (type: **net-list**)

message-size-limit [*bytes*];

MTA mail size limit.

bytes (type: **uint64**, optional, default: 10000000)

bounce-size-limit [*bytes*];

MTA DSN message size limit.

bytes (type: **uint64**, optional, default: 50000)

bounce-queue-lifetime [*seconds*];

MTA queue lifetime.

seconds (type: **uint32**, optional, default: 5d)

delay-warning-time [*seconds*];

MTA delay warning time.

seconds (type: **uint32**, optional, default: 3h)

tls {

security-level ... ;

log-level ... ;

}

Client TLS parameters.

Items & subsections:

security-level [*level*];

Security level.

level (type: **postfix-security-level**, optional, default: may)

log-level [*level***];**

Log level (0..4).

level **(type: uint8, optional, default: 1)**

[End of section `smtp-agent.tls` description.]

set-var *name value*;

Postfix main.cf variables setting.

name **(type: str)**

Variable name.

value **(type: str)**

Variable value.

Constraints:

Variable name must contain alphanumeric chars only.

master-cf *file*;

Postfix master.cf configuration file.

This file serves as the source for the master.cf files copied into etc/postfix.NAME directories. The referenced file can be the master.cf file from the Postfix distribution, because the CML modifies this file according to its purpose, i.e.

- for the LOCAL-MAILER, CML comments off all listening modules

- for SMTP-FORWARDERs, CML comments off local delivery module (if LOCAL-MAILER is used).

file **(type: name of shared-file,** see common**(5))**

smtpd-option *option*;

Additional option to smtpd call from master.cf file.

option **(type: str)**

transport-map [none];

transport-map relay;

transport-map fallback *fallback port*;

transport-map extern *file*;

Transport map definition.

<branching element> **(type: postfix-transport-map-mode, optional, default: none)**

fallback **(type: host)**

Fallback address.

port **(type: port)**

Fallback port.

file **(type: name of shared-file,** see common**(5))**

Constraints:

Fallback port must differ from smtp port.

[End of section `smtp-agent` description.]

`smtp-forwarder` *name* **{**

* server ... ;

agent { ... }

timeouts { ... }

hostname ... ;

size ... ;

source-address ... ;

* domain ... ;

server-ssl ... ;

* server-cert-match ... ;

altq ... ;

}

This section defines SMTP forwarding channel.

Constraints:

At least one server must be specified.

At most 31 servers can be used.

SSL/TLS required on connection in order to match server certificates.

Items & subsections:

`server` ***addr*;**

Forwarding MTA description.

addr **(type: `sock`)**

Server IP address/port

`agent` **{**

phase ... ;

* tag ... ;

relayhost ... ;

source-address ... ;

myhostname ... ;

smtp-helo-name ... ;

myorigin ... ;

inet-protocol ... ;

relay-domains ... ;

mydestinations ... ;

mynetworks ... ;

message-size-limit ... ;

bounce-size-limit ... ;
bounce-queue-lifetime ... ;
delay-warning-time ... ;
tls { ... }
* set-var ... ;
master-cf ... ;
smtpd-option ... ;
transport-map ... ;
}

Relaying by local MTA.

If used, this section defines parameters of a local agent listening on 'server' addresses and delivering mails sent by smtp-proxy.

The `agent` section is derived from `smtp-agent` section prototype.
For detail description of it, see above.

`timeouts {`

welcome ... ;
mail-cmd ... ;
rcpt-cmd ... ;
data-cmd ... ;
data-blk ... ;
data-end ... ;
terminate ... ;
default ... ;
}

This section defines several timeouts for server reply.

Items & subsections:

`welcome` [*seconds*];

Timeout for intial 220 reply.

seconds **(type: `uint16`, optional, default: 300)**

Default value set by RFC2821.

`mail-cmd` [*seconds*];

Timeout for reply to MAIL command.

seconds **(type: `uint16`, optional, default: 300)**

Default value set by RFC2821.

`rcpt-cmd` [*seconds*];

Timeout for reply to RCPT command.

seconds **(type: `uint16`, optional, default: 300)**

Default value set by RFC2821.

`data-cmd` [*seconds*];

Timeout for reply to DATA command.

seconds **(type: uint16, optional, default: 120)**

Default value set by RFC2821.

data-blk [*seconds***];**

Timeout for reply to DATA block send completion.

seconds **(type: uint16, optional, default: 180)**

Default value set by RFC2821.

In fact, RFC says client should have timeout for "awaiting the completion of each TCP SEND call". Instead, smtp-proxy does not start this timeout unless output buffer is full.

data-end [*seconds***];**

Timeout for reply to CRLF.CRLF marker.

seconds **(type: uint16, optional, default: 600)**

Default value set by RFC2821.

terminate [*seconds***];**

Timeout for enforced session termination.

seconds **(type: uint16, optional, default: 10)**

default [*seconds***];**

Timeout for all other situations.

seconds **(type: uint16, optional, default: 300)**

[End of section `smtp-forwarder.timeouts` description.]

hostname *name*;

Hostname to introduce myself to server.

If omitted, global smtp-proxy.hostname is used.

name **(type: str)**

size [*usage***];**

Usage of SIZE ESMTP extension to the server.

This item defines, whether smtp-proxy uses SIZE extension to MAIL command. Possible values are:

- DISABLE ... do not use SIZE.
- INPUT ... use SIZE (if supported by server) with received mail size instead of computing correct one.

usage **(type: smtp-size-usage, optional, default: disable)**

source-address [client] [addr4 *addr4***] [addr6** *addr6***] cluster [***cluster***];**
source-address [client] [addr4 *addr4***] [addr6** *addr6***] [physical];**
source-address [client] [addr4 *addr4***] [addr6** *addr6***] no-fallback;**

Source address for outgoing connections to servers.

If omitted, the proper address of the proxy will be used, i.e. in the case of a cluster, the cluster address will be used.

If not specified by the SOURCE-PORT item, a generic port will be used.

The elements entered within this item will be used by the proxy until the first of them is applicable:

- The CLIENT keyword means the original client IP address is used. This mode will be succesful in all cases except mismatch of IP address families.

- The ADDR4/ADDR6 keyword-value pairs mean that the specified address is used for a connection of corresponding address family.

- The CLUSTER keyword means that one of cluster addresses will be used. By default, the main address of the bridge is used, however, any preferred alias address can be listed in the cluster list.- The PHYSICAL option means that the address of the physical interface is used instead of the cluster one.

- The DEFAULT option means the default behavior - i.e. using of the physical address.

- The NO-FALLBACK option means that if no other way of setting the address is acceptable, the session is rejected. Without this option, the system tries to find a suitable source IP address automatically.

client (type: **key**, optional)

addr4 *addr4* (type: **host**, optional, default: **[0.0.0.0]**)

addr6 *addr6* (type: **host**, optional, default: **[::]**)
<branching element> (type: **source-address-mode**, optional, default: **physical**)
cluster (type: **host-list**, optional, default: **{}**)

Constraints:

Address family must respect the element's address family..

domain *names*;

List of mail server domain names handled by this forwarder.

These items are used when smtp-proxy needs to send either a copy of a mail, or a DSN message, for choosing the right forwarder. The first forwarder section with matching DOMAIN item(s) is used. More occurences of item are treated as a disjunction (OR-mode).

names (type: **str-list**)

server-ssl *params* **[session]**;

server-ssl *params* **command** [*obligation*];

Use SSL/TLS on the connection to a server.

params (type: **name** of **ssl-params**, see ssl(5))
<branching element> (type: **ssl-startup-mode**, optional, default: **session**)
obligation (type: **obligation**, optional, default: **allowed**)

server-cert-match [subject *subject*] **[issuer** *issuer*];

Requirements for server certificate.

subject *subject* (type: **str-set**, optional, default: *)
acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)
acceptable certificate issuers

altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to forwarder.

altq (type: **name of pf-queue**, see pf-queue(5))
queue name

paltq *paltq* (type: **name of pf-queue**, see pf-queue(5), optional, default: **NULL**)
priority queue name (if set, used for TCP ACK without data)

[End of section `smtp-forwarder` description.]

smtp-arg-check [**local-part-len** *local-part-len*] **ok** [*addrs*];

smtp-arg-check [**local-part-len** *local-part-len*] **error** [*errors* [*addrs*]];

smtp-arg-check [**local-part-len** *local-part-len*] **ignore-err** [*addrs*];

This item defines SMTP envelope arguments check conditions.

local-part-len *local-part-len* (type: **uint8**, optional, default: **64**)
Maximum length of local part (default set by RFC2821).

<**branching element**> (type: **smtp-err-switch**)
This element controls which parsing results match this item:

- OK: only correct arguments match
- ERROR: arguments with further specified error match
- IGNORE-ERR: all arguments (correct and errorneous) match

errors (type: **smtp-error-set**, optional, default: *)
Set of errors matching this configuration item.

addrs (type: **str-set**, optional, default: *)
Set of addresses matching this configuration item.

The matching rules are slightly modified for the purpose of email addresses matching:

- Regular expressions are matched as usual.
- Addresses defined by string ended with the '@' character match given local-part in any domain.
- Addresses defined by string beginning by the '@' character match any address within given domain.
- Other addresses containing the '@' character match exactly given address.
- Addresses defined by string containing no '@' character match any address within given domain and any its subdomain.

Warning: in case of ARG-INVALID error, an empty string (instead of the errorneous address) is being matched against the list.

smtp-reply [*code* [*subject* [*detail* [*text*]]]];

This item defines SMTP reply code and text.

The default code & text is used if values are omitted.

code **(type: uint16, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

subject **(type: uint8, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

detail **(type: uint8, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

text **(type: str, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

mailbox *email*;

Mail recipient address.

email **(type: str)**

Deliver mail to this address, using SMTP-FORWARDER chosen by comparing the address domain name with forwarder DOMAIN(s).

Constraints:

E-mail must comply with RFC.

grey-listing {

reply ... ;

block-time ... ;

retry-time ... ;

guard-time ... ;

client-mask ... ;

file ... ;

}

Grey-listing verification parameters.

If the grey-listing method is used, mails for a newly seen triplet <client, sender, recipient> are temporarily rejected for the BLOCK-TIME period and it is expected their delivery to be successfully retried within the RETRY-TIME period. The triplets are stored into a database file.

For more details, see triplicator(1) manual page.

The particular attributes are normally set at smtp-proxy level (for the default values, see the description at that point) and redefined at delivery-acl level, if needed.

Items & subsections:

reply [*code* **[***subject* **[***detail* **[***text***]]]];**

Default refusal response code and text.

***code* (type: uint16, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

***subject* (type: uint8, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***detail* (type: uint8, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***text* (type: str, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

Refusing must be temporary (4xx).

block-time *seconds*;

Timeout for blocking of newly seen triplets.

***seconds* (type: uint32)**

retry-time *seconds*;

Total time of waiting for triplet delivery retrying.

Within this time (after the initial BLOCK-TIME period), the triplet is normally processed.

After this time, the triplet is set into initial state.

***seconds* (type: uint32)**

guard-time *seconds*;

Time of guarded delivery for a triplet.

If the triplet was retried to deliver by the client (and the triplet is thereby enabled), all mails for the triplet are normally processed for the GUARD-TIME period. Every new mail resets this period.

seconds **(type: `uint32`)**

`client-mask` [*bits*];

Database wide client netmask.

Clients are stored into the database using this mask. This feature allows correct function of grey-listing even for MTAs using a cluster of several machines with several IP addresses.

bits **(type: `uint8`, optional, default: 24)**

`file` *name*;

Triplet database file name.

This file must be set at smtp-proxy level because it is common for the whole proxy.

name **(type: `str`)**

[End of section `grey-listing` description.]

`smtp-proxy` *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

mail-pool ... ;

quarantine ... ;

postmaster ... ;

hostname ... ;

init-timeout ... ;

bad-commands ... ;

bad-recipients ... ;

dsn-mail-copy ... ;

use-antivirus ... ;

use-antispam ... ;

ssl-session-cache { ... }

grey-listing { ... }

* session-acl *name* { ... }

* delivery-acl *name* { ... }

* mail-acl *name* { ... }

* doc-acl *name* { ... }

}

This section defines SMTP-proxy attributes.

The `smtp-proxy` section is derived from `proxy` section prototype.
For detail description of it, see application(5).
Changes to the `smtp-proxy` section:

Section `udpserver` is not valid.

Item `source-address` is not valid.

MAIL-POOL must be specified.

POSTMASTER must be specified.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one DELIVERY-ACL must be specified.

At least one MAIL-ACL must be specified.

At least one DOC-ACL must be specified.

USE-ANTIVIRUS must be configured if VIRUS-STATUS used.

QUARANTINE must be configured at proxy level if used in ACLs.

Proxy must listen on QUARANTINE.PORT if used.

GREY-LISTING must be configured if used in DELIVERY-ACL.

WHITE-LISTING must be configured if SPF used in DELIVERY-ACL.

Section `monitoring` (see monitoring(5))

Item `aproxy-user` is not valid.

Item `data` used as `rcpts`.

Item `idle-timeout` (see application(5))

Element `seconds` is optional, default: 300.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 25.

Element `proto` is optional, default: tcp.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 25.

Element `proto` is optional, default: tcp.

Item `doctype-identification.order` (see acl(5))

Only UPLOAD direction can be used.

Added items & subsections:

`client-conn` {

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Client connection options.

The `client-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

Changes to the `client-conn` section:

Item `conn-timeout` is not valid.

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

Item `recv-bufsize` (see netio(5))

Buffer must be at least 1002 bytes long.

Item `send-bufsize` (see netio(5))

Buffer must be at least 1002 bytes long.

`server-conn` {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Server connection options.

In particular, output buffer size must be large enough to hold all lines of maximal allowed SMTP document header.

The `server-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

Changes to the `server-conn` section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

Item `recv-bufsize` (see netio(5))

Buffer must be at least 1002 bytes long.

Item `send-bufsize` (see netio(5))

Buffer must be at least 1002 bytes long.

`mail-pool` *name*;

Mail pool directory.

The directory is used for temporary storing of incoming mails.

***name* (type: `str`)**

Directory path name (relative to the CHROOT-DIR if any).

`quarantine` *dir* [*port*];

Quarantine directory and resend-port.

***dir* (type: `str`)**

Directory path name (relative to the CHROOT-DIR if any).

***port* (type: `port`, optional, default: 0)**

Port for receiving mails sent by admin from quarantine.

This must be one of LISTEN-ON.NON-TRANSPARENT addresses with IP address [127.0.0.1]. Can be omitted if this proxy won't resend quarantine mails.

`postmaster` *email*;

Postmaster address.

Mails for <postmaster> are forwarded to this address.

***email* (type: `str`)**

Postmaster email; local-part with quotting is not valid.

Constraints:

E-mail must comply with RFC.

`hostname` *name*;

Hostname used instead of regular host name.

***name* (type: `str`)**

`init-timeout` [*seconds*];

Timeout for initial command reception.

***seconds* (type: `uint16`, optional, default: 30)**

`bad-commands` [*limit*];

Maximum of rejected SMTP commands per session.

***limit* (type: `uint16`, optional, default: 100)**

`bad-recipients` [*limit*];

Maximum of rejected RCPT commands per mail.

limit (type: **uint16**, optional, default: **100**)

dsn-mail-copy disable;

dsn-mail-copy [enable] [*bytes***];**

Sending of original mail copy in DSN messages.

Delivery Status Notification (DSN) messages (RFC1894, RFC1982) optionally contain portion of the original mail, delivery status of which is being reported. By default, smtp-proxy sends original mail headers and portion of mail body.

This item allows to disable this behavior, or declare the size limit of mail body portion being sent. When enabled, mail headers are sent always complete.

<**branching element**> (type: **enabling**, optional, default: **enable**)

bytes (type: **uint32**, optional, default: **16K**)

use-antivirus disable;

use-antivirus enable *channel*;

Antivirus usage mode.

If omitted, or disabled, no antivirus is enabled. In this case, neither any ANTIVIRUS global section can be present nor any MAIL-ACL and DOC-ACL can have VIRUS item specified.

<**branching element**> (type: **enabling**)

channel (type: **name-list** of **antivirus**, see antivirus(5))

use-antispam disable;

use-antispam enable *channel* [*limit*];

Antispam usage.

This section defines type of antispam daemon used and mode of antispam checking operation.

<**branching element**> (type: **enabling**)

channel (type: **name** of **antispam**, see mod-antispam(5))

Name of antispam global section used.

Referred section defines the way how to communicate with the antispam daemon (see above).

limit (type: **uint64**, optional, default: **0**)

Size limit (in bytes) for antispam check.

Antispam checking used to be very exhausting operation, and typical spam mails used to be not very large (both for passing by size limit filters and for being able to send a lot of copies). That's why it can be desired to avoid checking of very large mails.

Setting of this limit says antispam module not to check mails larger than given limit and declare their spam score to zero.

Setting this limit to zero disables this feature and enables using of antispam to all mails. Be prepared for high machine load and noticeable delay in delivery if used so.

ssl-session-cache {

capacity ... ;

dir ... ;

lock ... ;

}

The ssl-session-cache section is derived from ssl-session-cache section prototype. For detail description of it, see ssl(5).

grey-listing {

reply ... ;

block-time ... ;

retry-time ... ;

guard-time ... ;

client-mask ... ;

file ... ;

}

Grey-listing global default parameters.

Most of these parameters can be redefined in DELIVERY-ACL.

The grey-listing section is derived from grey-listing section prototype. For detail description of it, see above.

Changes to the grey-listing section:

Triplet database file name must be defined.

Item block-time (see above)

Element `seconds` is optional, default: 1h.

Item retry-time (see above)

Element `seconds` is optional, default: 4h.

Item guard-time (see above)

Element `seconds` is optional, default: 36d.

session-acl *name* {

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

idle-timeout ... ;

source-address ... ;

hostname ... ;

welcome-msg ... ;

client-ssl ... ;

* client-cert-match ... ;

unknown-client ... ;

unmatching-client ... ;

* blacklisted-client ... ;

white-listing ... ;

strict-rfc-arg ... ;

size-limit { ... }

size-tolerance ... ;

rcpt-limit { ... }

dsn-rcpt-limit { ... }

mail-filter ... ;

client-altq ... ;

}

The first level of ACL decides whether to accept incoming connection.

The `session-acl` section is derived from `acl-1` section prototype.
For detail description of it, see acl(5).
Changes to the `session-acl` section:

Item `user` is not valid.

Item `auth` is not valid.

Item `idle-timeout-peer` is not valid.

Item `plug-to` is not valid.

SSL/TLS required on connection in order to match client certificates.

Item `doctype-ident-order` (see acl(5))

Only UPLOAD direction can be used.

Added items & subsections:

`hostname` *name*;

Hostname to introduce myself to client.

If omitted, global smtp-proxy.hostname is used.

name **(type: `str`)**

`welcome-msg` *text*;

Welcome greeting message text.

Hostname and date/time are added automatically.

If omitted, Kernun proxy name and version is used.

text **(type: `str`)**

`client-ssl` *params* [`session`];

`client-ssl` *params* `command` [*obligation*];

Use SSL/TLS on the connection from a client.

params (type: **name** of **ssl-params**, see ssl(5))
<**branching element**> (type: **ssl-startup-mode**, optional, default: session)
obligation (type: **obligation**, optional, default: allowed)

client-cert-match [**subject** *subject*] [**issuer** *issuer*];

Requirements for client certificate.

subject *subject* (type: **str-set**, optional, default: *)

acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)

acceptable certificate issuers

unknown-client [*text*];

Check client's reverse DNS record.

If set, all clients are checked to have DNS reverse record and service is rejected for such that do not.

text (type: **str**, optional, default: <**NULL**>)

If omitted, default text is used.

unmatching-client [*text*];

Check client's IP against all client's names' addresses.

If set, all clients are checked to have DNS reverse record correct. It means that at least one name got as reverse client name must have IP address equal to the one of client connection.

text (type: **str**, optional, default: <**NULL**>)

If omitted, default text is used.

blacklisted-client *database* [*text*];

Check client's IP against DNS black-list databases.

If set, all clients are checked to have a DNS A record in given domain and if so, the session is rejected.

If used multiple times, a new resolution operation with full CONN-TIMEOUT is started for every domain (until some query will succeed with an A RR response). Thus, too many items can lead to mail delivery timing out.

database (type: **str**)

List of checked domains.

text (type: **str**, optional, default: <**NULL**>)

If omitted, default text is used.

Constraints:

Blacklist domain name too long.

white-listing [**sender-only**];

white-listing highest-mx;

white-listing check-all [*alt-domains*];

white-listing matching-mx [*alt-domains*];

Provide white-listing (Sender Policy Framework) checking.

The result can be matched in DELIVERY-ACL.

<branching element> (type: `spf-modes`, optional, default: sender-only)

alt-domains **(type: `str-list`, optional, default: {})**

Alternate domains for SPF check.

`strict-rfc-arg`;

Check strict RFC2821 MAIL/RCPT argument format.

RFC 2821 defines correct command argument format, but allows for some obsolete formats to be accepted. This item controls (rejects) usage of these old formats.

`size-limit` {

soft ... ;

hard ... ;

}

Maximum mail size.

Soft-limit is also used as SIZE parameter in EHLO response.

The `size-limit` section is derived from `smtp-limit` section prototype. For detail description of it, see above.

`size-tolerance` ***percent*****;**

Tolerance to clients' SIZE declaration.

If used, client can send given number of percent more data than declared in MAIL SIZE parameter.

If omitted, real size is not checked against declared one.

percent **(type: `uint32`)**

Constraints:

Size tolerance must be at most 100%.

`rcpt-limit` {

soft ... ;

hard ... ;

}

Maximum number of recipients per message.

Setting to less than 100 is RFC2821 violation.

The `rcpt-limit` section is derived from `smtp-limit` section prototype. For detail description of it, see above.

Item `soft` (see above)

Element `val` is optional, default: 100.

`dsn-rcpt-limit` {

soft ... ;

hard ... ;

}

Maximum number of recipients per DSN report.

There is no such limit in RFC2821, however, such mails SHOULD have only one recipient (original sender) and null sender (MAIL FROM: <>) is often used by spammers.

This limitation is additional one to the RCPT-LIMIT.

The `dsn-rcpt-limit` section is derived from `smtp-limit` section prototype. For detail description of it, see above.
Item `soft` (see above)
Element `val` is optional, default: 5.

`mail-filter` *name*;
SMTP/MIME document filtering/checking rules.

name **(type: `name` of `mail-filter`, see mod-mail-doc(5))**

`client-altq` *altq* [`paltq` *paltq*];
ALTQ queues for data sent to client.

altq **(type: `name` of `pf-queue`, see pf-queue(5))**
queue name
`paltq` *paltq* (type: `name` of `pf-queue`, see pf-queue(5), optional, default: NULL)
priority queue name (if set, used for TCP ACK without data)

[End of section `smtp-proxy.session-acl` description.]

`delivery-acl` *name* {
* from ... ;
* time ... ;
time-period-set { ... }
* session-acl ... ;
deny ... ;
accept ... ;
* doctype-ident-order ... ;
rule ... ;
* helo ... ;
* sender ... ;
* recipient ... ;
spf ... ;
* client-cert-match ... ;
abort ... ;
reject ... ;
refuse ... ;
discard ... ;
deliver ... ;
grey-listing { ... }
* copy-to ... ;
quarantine ... ;
}

The second level of ACL decides about reply to particular RCPT command and way how to deliver mail.

The `delivery-acl` section is derived from `acl-2` section prototype.
For detail description of it, see acl(5).
Changes to the `delivery-acl` section:

Item `server` is not valid.

Item `user` is not valid.

Item `parent-acl` used as `session-acl`.

ABORT, REJECT or REFUSE must be specified if DENY is on.

ABORT, REJECT and REFUSE cannot be specified if ACCEPT is on.

DISCARD, DELIVER, QUARANTINE and COPY-TO cannot be specified if DENY is on.

ABORT, REJECT and REFUSE are mutually exclusive.

DISCARD and DELIVER are mutually exclusive.

At least one SENDER must be specified.

At least one RECIPIENT must be specified.

Item `doctype-ident-order` (see acl(5))

Only UPLOAD direction can be used.

Added items & subsections:

`helo` [`local-part-len` *local-part-len*] `ok` [*addrs*];

`helo` [`local-part-len` *local-part-len*] `error` [*errors* [*addrs*]];

`helo` [`local-part-len` *local-part-len*] `ignore-err` [*addrs*];

Entry condition - matching HELO/EHLO command parameter.

`local-part-len` *local-part-len* (type: `uint8`, optional, default: 64)

Maximum length of local part (default set by RFC2821).

<branching element> (type: `smtp-err-switch`)

This element controls which parsing results match this item:

- OK: only correct arguments match
- ERROR: arguments with further specified error match
- IGNORE-ERR: all arguments (correct and errorneous) match

***errors* (type: `smtp-error-set`, optional, default: *)**

Set of errors matching this configuration item.

***addrs* (type: `str-set`, optional, default: *)**

Set of addresses matching this configuration item.

The matching rules are slightly modified for the purpose of email addresses matching:

- Regular expressions are matched as usual.
- Addresses defined by string ended with the '@' character match given local-part in any domain.
- Addresses defined by string beginning by the '@' character match any address within given domain.
- Other addresses containing the '@' character match exactly given address.
- Addresses defined by string containing no '@' character match any address within given domain and any its subdomain.

Warning: in case of ARG-INVALID error, an empty string (instead of the errorneous address) is being matched against the list.

sender [local-part-len *local-part-len***] ok [***addrs***];**

sender [local-part-len *local-part-len***] error [***errors* **[***addrs***]];**

sender [local-part-len *local-part-len***] ignore-err [***addrs***];**

Entry condition - matching MAIL command parameter.

local-part-len *local-part-len* **(type: uint8, optional, default: 64)**

Maximum length of local part (default set by RFC2821).

<branching element> (type: smtp-err-switch)

This element controls which parsing results match this item:

- OK: only correct arguments match
- ERROR: arguments with further specified error match
- IGNORE-ERR: all arguments (correct and errorneous) match

errors **(type: smtp-error-set, optional, default: *)**

Set of errors matching this configuration item.

addrs **(type: str-set, optional, default: *)**

Set of addresses matching this configuration item.

The matching rules are slightly modified for the purpose of email addresses matching:

- Regular expressions are matched as usual.
- Addresses defined by string ended with the '@' character match given local-part in any domain.
- Addresses defined by string beginning by the '@' character match any address within given domain.
- Other addresses containing the '@' character match exactly given address.
- Addresses defined by string containing no '@' character match any address within given domain and any its subdomain.

Warning: in case of ARG-INVALID error, an empty string (instead of the errorneous address) is being matched against the list.

recipient [local-part-len *local-part-len***] ok [***addrs***];**

recipient [local-part-len *local-part-len***] error [***errors* **[***addrs***]];**

recipient [local-part-len *local-part-len***] ignore-err [***addrs***];**

Entry condition - matching RCPT command parameter.

local-part-len *local-part-len* **(type: uint8, optional, default: 64)**

Maximum length of local part (default set by RFC2821).

<branching element> (type: smtp-err-switch)

This element controls which parsing results match this item:

- OK: only correct arguments match
- ERROR: arguments with further specified error match
- IGNORE-ERR: all arguments (correct and errorneous) match

errors (type: **smtp-error-set**, optional, default: *)

Set of errors matching this configuration item.

addrs (type: **str-set**, optional, default: *)

Set of addresses matching this configuration item.

The matching rules are slightly modified for the purpose of email addresses matching:

- Regular expressions are matched as usual.
- Addresses defined by string ended with the '@' character match given local-part in any domain.
- Addresses defined by string beginning by the '@' character match any address within given domain.
- Other addresses containing the '@' character match exactly given address.
- Addresses defined by string containing no '@' character match any address within given domain and any its subdomain.

Warning: in case of ARG-INVALID error, an empty string (instead of the errorneous address) is being matched against the list.

spf [*results*];

Entry condition - white-listing (SPF) result.

results (type: **spf-result-set**, optional, default: *)

client-cert-match [**subject** *subject*] [**issuer** *issuer*];

Entry condition - matching SSL client certificate.

subject *subject* (type: **str-set**, optional, default: *)

acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)

acceptable certificate issuers

abort [*code* [*subject* [*detail* [*text*]]]];

SMTP session is to be immediately closed.

In fact, the operation consists of three steps on client channel: specified reply is sent, reply 421 is sent and connection is closed.

code (type: **uint16**, optional, default: 0)

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

subject (type: **uint8**, optional, default: 255)

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

detail (type: **uint8**, optional, default: 255)

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

text (type: **str**, optional, default: <NULL>)

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

reject [*code* **[***subject* **[***detail* **[***text***]]]];**

Mail is rejected.

If such an action is taken for at least one recipient, the mail will not be accepted and specified reply is sent to the client as a response to both RCPT and DATA commands.

***code* (type: `uint16`, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

***subject* (type: `uint8`, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***detail* (type: `uint8`, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***text* (type: `str`, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

refuse [*code* **[***subject* **[***detail* **[***text***]]]];**

Mail sending to particular recipient is refused.

RCPT command is answered by specified reply code with no impact to other recipients.

***code* (type: `uint16`, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

***subject* (type: `uint8`, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***detail* (type: `uint8`, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***text* (type: `str`, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

discard;

Mail is accepted, but discarded.

The mail will not be sent to this recipient but client is informed as if it was (by reply code 250). No impact to sending to other recipients.

deliver [via *via*] [**to** *to*];

Mail is normally delivered.

via *via* **(type: name of smtp-forwarder, see above, optional, default: NULL)**

Name of FORWARDER used for delivery.

If omitted, forwarder is selected according to their DOMAIN attributes.

to *to* **(type: str, optional, default: <NULL>)**

New addressee (for forwarding to alias).

grey-listing {

reply ... ;

block-time ... ;

retry-time ... ;

guard-time ... ;

}

Grey-listing method setting.

Using of this section enables grey-listing mode.

Omitted parameters take value from proxy-global section.

The grey-listing section is derived from grey-listing section prototype. For detail description of it, see above.

Changes to the grey-listing section:

Item `client-mask` is not valid.

Item `file` is not valid.

copy-to *email*;

Blind-copy of mail will be sent to given address.

email **(type: str)**

Deliver mail to this address, using SMTP-FORWARDER chosen by comparing the address domain name with forwarder DOMAIN(s).

Constraints:

E-mail must comply with RFC.

quarantine;

Store mail into quarantine.

[End of section `smtp-proxy.delivery-acl` description.]

mail-acl *name* **{**

* from ... ;

* time ... ;

time-period-set { ... }

* delivery-acl ... ;

deny ... ;

accept ... ;

rule ... ;

* size ... ;
* content-type ... ;
  virus-status ... ;
* modify-header ... ;
  replace ... ;
* sender ... ;
* recipient ... ;
* recipients ... ;
* spam-score ... ;
* header ... ;
  from-quarantine ... ;
  prefix-subject ... ;
  abort ... ;
  reject ... ;
  discard ... ;
  cancel ... ;
* copy-to ... ;
* redirect-to ... ;
  quarantine ... ;
  omit-dsn ... ;

}

The first ACL on the third level decides how to handle the mail as whole. In fact, one more decision for the whole mail can be made - in DOC-ACL corresponding to the root of MIME tree.For particular recipient, all denial actions from the third level ACL are collected, the one with highest severity (abort > reject > discard > cancel) from all DOC-ACLs is chosen. Some actions (discard, cancel) have per-recipient impact and no influence to sending mail to other recipients. However, other actions (abort, reject) have per-mail impact and the one with highest severity from all recipients is executed.

If no ACL is found, ABORT action is preformed.

The `mail-acl` section is derived from `mail-acl` section prototype.
For detail description of it, see mod-mail-doc(5).
Changes to the `mail-acl` section:

Item `parent-acl` used as `delivery-acl`.

Item `direction` is not valid.

ABORT, REJECT, DISCARD or CANCEL must be specified if DENY is on.

ABORT, REJECT, DISCARD and CANCEL cannot be specified if ACCEPT is on.

ABORT, REJECT, DISCARD and CANCEL are mutually exclusive.

Items QUARANTINE/COPY-TO and ABORT are mutually exclusive.

Items REDIRECT-TO/OMIT-DSN and DENY are mutually exclusive.

Item REDIRECT-TO must be used only once.

Added items & subsections:

`abort` [*code* [*subject* [*detail* [*text*]]]];

SMTP session is to be immediately closed.

In fact, the operation consists of three steps on client channel: specified reply is sent, reply 421 is sent and connection is closed.

***code* (type: `uint16`, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

***subject* (type: `uint8`, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***detail* (type: `uint8`, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***text* (type: `str`, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

`reject` [*code* [*subject* [*detail* [*text*]]]];

Mail is rejected.

If such an action is taken for at least one recipient, the mail is not accepted and specified reply is sent to the client as a response to DATA command.

***code* (type: `uint16`, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

***subject* (type: `uint8`, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***detail* (type: `uint8`, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***text* (type: `str`, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

`discard`;

Mail sending to this recipient is denied but the client is informed as if it was sent (recipient is not added to the DSN report).

cancel [*code* [*subject* [*detail* [*text*]]]];

Mail will not be sent to particular recipient.

The mail still can be successfully sent to other recipients, given response code is used in DSN report that will be sent to the original sender address.

code **(type: uint16, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

subject **(type: uint8, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

detail **(type: uint8, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

text **(type: str, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

copy-to *email*;

Blind-copy of mail will be sent to given address.

email **(type: str)**

Deliver mail to this address, using SMTP-FORWARDER chosen by comparing the address domain name with forwarder DOMAIN(s).

Constraints:

E-mail must comply with RFC.

redirect-to *email*;

Change final mail recipient.

email **(type: str)**

Deliver mail to this address, using SMTP-FORWARDER chosen by comparing the address domain name with forwarder DOMAIN(s).

Constraints:

E-mail must comply with RFC.

quarantine;

Store mail into quarantine.

omit-dsn;

Discard in case of delivery failure.

This item defines proxy behavior in case when a delivery process fails for particular recipient. If it is used, the recipient is dealt like if the delivery succeeds. Thus, this delivery failure does not cause sending of the Delivery Status Notification (DSN) message.

[End of section `smtp-proxy.mail-acl` description.]

doc-acl *name* {

* from ... ;
* time ... ;
  time-period-set { ... }
* delivery-acl ... ;
  deny ... ;
  accept ... ;
  rule ... ;
* size ... ;
* content-type ... ;
* mime-type ... ;
  virus-status ... ;
* modify-header ... ;
  force-doctype-ident ... ;
  replace ... ;
  html-filter ... ;
* sender ... ;
* recipient ... ;
* spam-score ... ;
* header ... ;
* filename ... ;
  from-quarantine ... ;
  add-virus-names ... ;
  abort ... ;
  reject ... ;
  discard ... ;
  cancel ... ;
* copy-to ... ;
  quarantine ... ;

}

The second ACL on the third level decides how to process particular documents (or precisely: MIME tree nodes) contained in the mail.

Denial actions (if any) concern whole mail-copy for particular recipient, not only a single document. The one with highest severity (abort > reject > discard > cancel) from all DOC-ACLs is executed.

If no ACL is found, ABORT action is preformed.

The `doc-acl` section is derived from `mail-doc-acl` section prototype. For detail description of it, see mod-mail-doc(5).

Changes to the `doc-acl` section:

Item `parent-acl` used as `delivery-acl`.

Item `direction` is not valid.

ABORT, REJECT, DISCARD or CANCEL must be specified if DENY is on.

ABORT, REJECT, DISCARD and CANCEL cannot be specified if ACCEPT is on.

ABORT, REJECT, DISCARD and CANCEL are mutually exclusive.

Items QUARANTINE/COPY-TO and ABORT are mutually exclusive.

Added items & subsections:

`abort` [*code* [*subject* [*detail* [*text*]]]];

SMTP session is to be immediately closed.

In fact, the operation consists of three steps on client channel: specified reply is sent, reply 421 is sent and connection is closed.

***code* (type: `uint16`, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

***subject* (type: `uint8`, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***detail* (type: `uint8`, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***text* (type: `str`, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

`reject` [*code* [*subject* [*detail* [*text*]]]];

Mail is rejected.

If such an action is taken for at least one recipient, the mail is not accepted and specified reply is sent to the client as a response to final dot.

***code* (type: `uint16`, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

***subject* (type: `uint8`, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***detail* (type: `uint8`, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***text* (type: `str`, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

discard;

Mail sending to this recipient is denied but the client is informed as if it was sent (recipient is not added to the DSN report).

cancel [*code* **[***subject* **[***detail* **[***text***]]]];**

Mail will not be sent to particular recipient.

The mail still can be successfully sent to other recipients, given response code is used in DSN report that will be sent to the original sender address.

***code* (type: uint16, optional, default: 0)**

The 3-digit response code; the first digit is also used as the first sub-code (class) of ESC (Enhanced Status Code).

If omitted, the default response code is used.

***subject* (type: uint8, optional, default: 255)**

The second sub-code (subject) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***detail* (type: uint8, optional, default: 255)**

The third sub-code (detail) of ESC (Enhanced Status Code).

If omitted, the default sub-code is used.

***text* (type: str, optional, default: <NULL>)**

Response message text.

Constraints:

Reply-code must be 4xx or 5xx.

copy-to *email*;

Blind-copy of mail will be sent to given address.

***email* (type: str)**

Deliver mail to this address, using SMTP-FORWARDER chosen by comparing the address domain name with forwarder DOMAIN(s).

Constraints:

E-mail must comply with RFC.

quarantine;

Store mail into quarantine.

[End of section `smtp-proxy.doc-acl` description.]

[End of section `smtp-proxy` description.]

SEE ALSO

configuration(7), triplicator(1), acl(5), antivirus(5), application(5), auth(5), common(5), ipc(5), listen-on(5), log(5), mod-antispam(5), mod-mail-doc(5), monitoring(5), netio(5), pf-queue(5), source-address(5), ssl(5), time(5)

NAME

smtp-proxy.cfg — format of smtp-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **smtp-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **smtp-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

range-op (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see **radius**(5))

ldap-tls-reqcert-mode (see ldap(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

antivirus-protocol (see antivirus(5))

virus-status (see antivirus(5))

database-source (see antivirus(5))

source-address-mode (see source-address(5))

accept-deny (see mod-html-filter(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

ssl-ver (see ssl(5))

extension-op (see ssl(5))

veri-fail-action (see ssl(5))

auth-cert-type (see ssl(5))

distrusted-cert-type (see ssl(5))

smtp-error (see mod-mail-doc(5))

mail-reaction (see mod-mail-doc(5))

mail-fallback (see mod-mail-doc(5))

mime-header-check-type (see mod-mail-doc(5))

smtp-size-usage (see smtp-proxy(5))

ssl-startup-mode (see smtp-proxy(5))

postfix-security-level (see smtp-proxy(5))

postfix-transport-map-mode (see smtp-proxy(5))

smtp-err-switch (see smtp-proxy(5))

spf-result (see smtp-proxy(5))

spf-modes (see smtp-proxy(5))

ITEMS AND SECTIONS

Program **smtp-proxy** recognizes following items and sections:
* antispam *name* { ... }
* antivirus *name* { ... }
* fake-cert *name* { ... }
* html-filter *name* { ... }
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* mail-filter *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
* smtp-forwarder *name* { ... }
* ssl-params *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* smtp-proxy *name* { ... }
  ipv6-mode ... ;

Constraints:

All configured email domains must be handled by some SMTP-FORWARDER.

Description:

antispam *name* **{**

connection ... ;

sock-opt { ... }

altq ... ;

}

The `antispam` section is derived from `antispam` section prototype. For detail description of it, see mod-antispam(5).

`antivirus` *name* {

connection ... ;

sock-opt { ... }

timeout ... ;

comm-dir ... ;

altq ... ;

max-checked-size ... ;

icap-pass-200-with-pure-body ... ;

persistent-stream ... ;

clamav-agent { ... }

}

The `antivirus` section is derived from `antivirus` section prototype. For detail description of it, see antivirus(5).

`fake-cert` *name* {

key ... ;

auth-ca ... ;

fail-ca ... ;

* extension ... ;

purge ... ;

}

The `fake-cert` section is derived from `fake-cert` section prototype. For detail description of it, see ssl(5).

`html-filter` *name* {

* script-tag-language ... ;

replace-head-script-tags ... ;

replace-body-script-tags ... ;

* style-tag-type ... ;

replace-style-tags ... ;

```
    * iframe-tag-src ... ;
      replace-iframe-tags ... ;
    * intrinsic-language ... ;
    * intrinsic-hack ... ;
      replace-intrinsic ... ;
    * macro-language ... ;
    * macro-hack ... ;
      replace-macros ... ;
    * uri ... ;
      replace-uri ... ;
    * embed-tag-type ... ;
    * embed-src-hack ... ;
    * embed-plugin-hack ... ;
      replace-head-embed-tags ... ;
      replace-body-embed-tags ... ;
    * applet ... ;
      replace-applets ... ;
    * object ... ;
    * object-classid-hack ... ;
    * object-data-hack ... ;
      replace-head-object-tags ... ;
      replace-body-object-tags ... ;
    * param-tags ... ;
      replace-param ... ;
      script-end-hack ... ;
    }
```

The `html-filter` section is derived from `html-filter` section prototype. For detail description of it, see mod-html-filter(5).

`interface` *name* {

```
    dev ... ;
    ipv4 ... ;
    ipv6 ... ;
    mac ... ;
    aggregate ... ;
```

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`ldap-client-auth` *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

`mail-filter` *name* {

stamp-limit ... ;

stamp-filter ... ;

* unflagged-8bit ... ;

* bad-end-of-line ... ;

* invalid-header ... ;

* long-header-lines ... ;

* invalid-chars ... ;

* header-8bit-chars ... ;

* bad-boundary-chars ... ;

* bad-boundary-length ... ;

* long-body-lines ... ;

```
    * long-encoded-lines ... ;
      enc-line-len ... ;
    * bad-mime-struct ... ;
    * invalid-encoding ... ;
      treat-rfc822-as-text ... ;
    }
```

The `mail-filter` section is derived from `mail-filter` section prototype. For detail description of it, see mod-mail-doc(5).

`oob-auth` *name* {

```
    method ... ;
    max-sessions ... ;
    max-user ... ;
    max-groups ... ;
    truncate-groups ... ;
    file ... ;
    lock ... ;
    }
```

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

`pf-queue` *name* {

```
    parent ... ;
    bandwidth ... ;
    priority ... ;
    qlimit ... ;
    cbq { ... }
    priq { ... }
    hfsc { ... }
    }
```

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

radius-client *name* {

nas ... ;

groups ... ;

* server ... ;

}

The **radius-client** section is derived from **radius-client** section prototype. For detail description of it, see radius(5).

resolver *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The **resolver** section is derived from **resolver** section prototype. For detail description of it, see resolver(5).

shared-dir *name* {

path ... ;

}

The **shared-dir** section is derived from **shared-dir** section prototype. For detail description of it, see common(5).

shared-file *name* {

path ... ;

format ... ;

}

The **shared-file** section is derived from **shared-file** section prototype. For detail description of it, see common(5).

smtp-forwarder *name* {

* server ... ;
  agent { ... }
  timeouts { ... }
  hostname ... ;
  size ... ;
  source-address ... ;
* domain ... ;
  server-ssl ... ;
* server-cert-match ... ;
  altq ... ;
}

The smtp-forwarder section is derived from smtp-forwarder section prototype. For detail description of it, see smtp-proxy(5).

ssl-params *name* {

  versions ... ;
  ciphers ... ;
  tcp-eof ... ;
  id ... ;
* auth-cert ... ;
  distrusted-certs ... ;
  dont-check-crl ... ;
* crl ... ;
  verify-peer ... ;
  cache-timeout ... ;
  use-ticket ... ;
  enable-renegotiation ... ;
  fake-cert ... ;
  prefer server ciphers ... ;
  enable-ecdh ... ;
}

The ssl-params section is derived from ssl-params section prototype. For detail description of it, see ssl(5).

sysctl {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

use-resolver *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

smtp-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

mail-pool ... ;

quarantine ... ;

postmaster ... ;

hostname ... ;

init-timeout ... ;

bad-commands ... ;

bad-recipients ... ;

dsn-mail-copy ... ;

use-antivirus ... ;

use-antispam ... ;

ssl-session-cache { ... }

grey-listing { ... }

* session-acl *name* { ... }

* delivery-acl *name* { ... }

* mail-acl *name* { ... }

* doc-acl *name* { ... }

}

The `smtp-proxy` section is derived from `smtp-proxy` section prototype. For detail description of it, see smtp-proxy(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), smtp-proxy(8), acl(5), antivirus(5), auth(5), common(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), mod-antispam(5), mod-html-filter(5), mod-mail-doc(5), pf-queue(5), radius(5), resolver(5), smtp-proxy(5), source-address(5), ssl(5), sysctl(5), time(5), host-matching(7)

NAME

snmpd — format of snmpd component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **snmpd** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **snmpd** configuration directives:

ip-version (see **common**(5))

osi4-proto (see **common**(5))

listen-on-sock (see **listen-on**(5))

snmpd-disk-mode (name-usage obligatory)

SNMP disk monitoring setting type.

space

percent

snmpd-source-mode (name-usage obligatory)

SNMP source definition type.

default

host

net

snmpd-view-type (name-usage obligatory)

SNMP view type.

none

all

snmpd-security-level (name-usage obligatory)

SNMP security level.

noauth

auth

priv

snmpd-auth-hash (name-usage obligatory)

SNMP authentication hash function.

md5

sha

snmpd-encr-alg (name-usage obligatory)

SNMP encryption algorithm.

des

aes

ITEMS AND SECTIONS

Configuration of **snmpd** library component consists of following prototypes:
snmpd { ... }

Description:

snmpd {

phase ... ;

* tag ... ;

listen-on { ... }

* user ... ;

location ... ;

* group *name* { ... }

* proc ... ;

* exec ... ;

* disk ... ;

load ... ;

swap ... ;

* raw ... ;

}

SNMP Daemon configuration.

Constraints:

Addresses to listen on must be specified.

SNMP group name must be at most 32 characters long.

Items & subsections:

phase [*number*];

Application Startup Phase.

number (type: **uint8**, optional, default: 40)

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value (type: **str**)

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

listen-on {

* socket ... ;

}

The listen-on section is derived from listen-on section prototype.
For detail description of it, see listen-on(5).
Changes to the listen-on section:

Item `non-transparent` used as `socket`.

Item `transparent` is not valid.

At least one address to listen on must be specified.

Item socket (see listen-on(5))

Element `port` is optional, default: 161.

Element `proto` is optional, default: tcp-udp.

user *name* [**hash** *hash*] *passwd* [**alg** *alg*] [**privacy-passwd** *privacy-passwd*] [**readonly**];

SNMPv3 user.

name (type: **str**)

hash *hash* (type: **snmpd-auth-hash**, optional, default: sha)

passwd (type: **str**)

alg *alg* (type: **snmpd-encr-alg**, optional, default: aes)

privacy-passwd *privacy-passwd* (type: **str**, optional, default: <NULL>)

Data encryption password; if omitted, data is sent in cleartext.

readonly (type: **key**, optional)

Constraints:

SNMP user password must be at least 8 characters long.

SNMP privacy password must be at least 8 characters long.

location [*text*];

System location.

text (type: **str**, optional, default: "MyLocation")

group *name* {

community ... ;

access ... ;

}

SNMP group definition.

Items & subsections:

community *name* [**default**];

community *name* **host** *host*;

community *name* **net** *net*;

Group members by source address and community.

name (type: **str**)

Community name

<branching element> (type: **snmpd-source-mode**, optional, default: default)

host (type: **host**)

Client address or hostname

net (type: **net**)

Client address with mask/prefix

access *level* [**prefix**] [*context* [*read* [*write* [*notify*]]]];

SNMP views accessibility definition.

level (type: **snmpd-security-level**)

prefix (type: **key**, optional)

context (type: **str**, optional, default: "")

read (type: **snmpd-view-type**, optional, default: none)

write (type: **snmpd-view-type**, optional, default: none)

notify (type: **snmpd-view-type**, optional, default: none)

[End of section `snmpd.group` description.]

proc *name* **min** *min* [**max** *max*];

Process monitoring.

name (type: **str**)

Process name

min *min* (type: **uint16**)

Minimum number of processes

max *max* (type: **uint16**, optional, default: 0)

Maximum number of processes; 0 means infinity

Constraints:

Either min or max must be nonzero.

exec *name text*;

Arbitrary command extension.

name (type: **str**)

Command name

text (type: **str**)

Command text

disk *path* **space** [*space*];

disk *path* **percent** *percent*;

Disk usage monitoring.

path (type: **str**)

Disk pathname

<**branching element**> (type: **snmpd-disk-mode**)

percent (type: **uint8**)

Minimum percentage value

space (type: **uint64**, optional, default: 102400)

Minimum space value

load *max1 max5 max15*;

System load monitoring.

max1 (type: **uint8**)

max5 (type: **uint8**)

max15 (type: **uint8**)

swap *min*;

Swap space monitoring.

min (type: **uint64**)

raw *line*;

Raw line to configuration file.

line (type: **str**)

[End of section `snmpd` description.]

SEE ALSO

configuration(7), common(5), listen-on(5), snmpd.conf(5)

NAME

source-address — format of source-address component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **source-address** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **source-address** configuration directives:

source-address-mode (name-usage obligatory)

Source address (for server connection) mode specification.

cluster

Cluster virtual address is used.

physical

Physical interface address is used.

no-fallback

If no source address is acceptable, reject connection.

default

Default address selection.

source-port-mode (name-usage obligatory)

Source port (for server connection) mode specification.

client

Source port of the client is used.

force

Source port is forced by configuration.

ITEMS AND SECTIONS

Configuration of **source-address** library component consists of following prototypes:

source-address ... ;

source-port ... ;

Description:

source-address [client] [addr4 *addr4*] **[addr6** *addr6*] **cluster [***cluster***];**

source-address [client] [addr4 *addr4*] **[addr6** *addr6*] **[physical];**

source-address [client] [addr4 *addr4*] **[addr6** *addr6*] **no-fallback;**

Source address for outgoing connections to servers.

If omitted, the proper address of the proxy will be used, i.e. in the case of a cluster, the cluster address will be used.

If not specified by the SOURCE-PORT item, a generic port will be used.

The elements entered within this item will be used by the proxy until the first of them is applicable:

- The CLIENT keyword means the original client IP address is used. This mode will be succesful in all cases except mismatch of IP address families.

- The ADDR4/ADDR6 keyword-value pairs mean that the specified address is used for a connection of corresponding address family.

- The CLUSTER keyword means that one of cluster addresses will be used. By default, the main address of the bridge is used, however, any preferred alias address can be listed in the cluster list.- The PHYSICAL option means that the address of the physical interface is used instead of the cluster one.

- The DEFAULT option means the default behavior - i.e. using of the physical address.

- The NO-FALLBACK option means that if no other way of setting the address is acceptable, the session is rejected. Without this option, the system tries to find a suitable source IP address automatically.

client (type: key, optional)

addr4 *addr4* **(type: host, optional, default: [0.0.0.0])**

addr6 *addr6* **(type: host, optional, default: [::])**
<branching element> (type: source-address-mode, optional, default: physical)
cluster **(type: host-list, optional, default: {})**

Constraints:

Address family must respect the element's address family..

source-port client;

source-port [force] *port*;

Source port for outgoing connections to server.

Can be used only with SOURCE-ADDRESS CLIENT.

If omitted, generic port will be used.

<branching element> (type: source-port-mode, optional, default: force)

port **(type: port)**

Use specified port.

SEE ALSO

configuration(7)

NAME

sqlnet-proxy — format of sqlnet-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **sqlnet-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **sqlnet-proxy** configuration directives:

yes-no (see **common**(5))

on-off (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

auth-method (see **auth**(5))

source-address-mode (see **source-address**(5))

transparency (see **acl**(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

listen-on-sock (see listen-on(5))

redirection-mode (name-usage obligatory)

follow

ignore

ITEMS AND SECTIONS

Configuration of **sqlnet-proxy** library component consists of following prototypes:
* sqlnet-proxy *name* { ... }

Description:

sqlnet-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

init-timeout ... ;

protocol-version ... ;

max-service-name-len ... ;

check-reserved-bits ... ;

connect-string-charset ... ;

connect-packet-sizelimit ... ;

* session-acl *name* { ... }

* service-acl *name* { ... }

}

This section defines SQL*Net-proxy attributes.

The `sqlnet-proxy` section is derived from `proxy` section prototype.
For detail description of it, see application(5).
Changes to the `sqlnet-proxy` section:

Section `udpserver` is not valid.

Item `source-address` is not valid.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

At least one SERVICE-ACL must be specified.

Cannot use DB-USER for unknown protocol versions.

Section `monitoring` (see monitoring(5))

Item `aproxy-user` is not valid.

Item `data` used as `uri`.

Item `listen-on.non-transparent` (see listen-on(5))

Element `port` is optional, default: 1521.

Element `proto` is optional, default: tcp.

Item `listen-on.transparent` (see listen-on(5))

Element `port` is optional, default: 1521.

Element `proto` is optional, default: tcp.

Added items & subsections:

`client-conn` {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Client connection options.

The `client-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

`server-conn` {

conn-timeout ... ;

recv-timeout ... ;

recv-bufsize ... ;

send-timeout ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Server connection options.

The `server-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).

`init-timeout` [*seconds*];

Initialization timeout.

seconds **(type: `uint16`, optional, default: 10)**

`protocol-version` [*list*];

Permitted versions of TNS protocol.

By default, SQL*Net proxy permits communication only for known protocol versions (3.07 and 3.10 through 3.13). This item defines permission for other versions. However, unknown versions have some restrictions, e.g. disabled user-checking. If a version different from above and not specified here is detected, proxy tries to continue operation, but warn admin by alert level log message

list **(type: `uint16-set`, optional, default: {})**

Versions (307 for 3.07 etc.)

`max-service-name-len` [*chars*];

Limit to SERVICE NAME length.

Setting this parameter to nonzero value can avoid buffer overrun errors in many SQL*Net listenners. Setting it to zero switches the check off.

chars **(type: `uint16`, optional, default: 40)**

`check-reserved-bits` [*val*];

Enforce checking of reserved bits.

Some TNS listenners crash when receive packet with non-zero reserved bits.

val **(type: `on-off`, optional, default: on=1)**

`connect-string-charset` [*chars*];

Character set for CN string values.

Some clients use nonstandard characters in parameter values of CONNECT string. This item allows administrators to pass character set checks. The default value is reasonable for typical clients.

chars **(type: `str`, optional, default: ".@:-/\\")**

Allowed character set (will be completed by adding of all alphanumeric characters).

Constraints:

CN string charset must be at most 256 chars long.

`connect-packet-sizelimit` [*bytes*];

Maximal length of CN packet.

Some servers have limitation to size of CN packet. This item allows to control which CN packets will be sent to server splitted.

bytes **(type: `uint16`, optional, default: 288)**

`session-acl` *name* {

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;

auth ... ;

idle-timeout ... ;

source-address ... ;

plug-to ... ;

redirections ... ;

}

The first level ACL decides only between acceptation and denial of the incoming connection.

The `session-acl` section is derived from `acl-1` section prototype.

For detail description of it, see acl(5).

Changes to the `session-acl` section:

Item `user` is not valid.

Item `idle-timeout-peer` is not valid.

Only out-of-band authentication is supported in this proxy.

Added items & subsections:

`redirections` [`follow`] [*hops*];

`redirections ignore` [*hops*];

Redirection (RD) packets handling.

The current version of SQL*Net proxy handles RD packets by itself. It means that it checks the packet and tries to connect to the new server. For each client session, the maximal number of RD answers from servers is defined here. If more (than

maximum) servers send RD packet, this is assumed to be an infinite loop and the session is terminated.

By default, the proxy follows the RD string information. Sometimes, another mode may be desired when proxy ignores RD and respects its own configuration. Specially, this is important for the SESSION-ACL.PLUG-TO directive. However, use this IGNORE mode with care because it can simply lead to the infinite redirection loop. The SERVICE-ACL.PLUG-TO directive (if any) is respected in either mode.

<branching element> (type: `redirection-mode`, optional, default: follow)

***hops* (type: `uint16`, optional, default: 10)**

Maximum of redirections allowed.

[End of section `sqlnet-proxy.session-acl` description.]

`service-acl` *name* {

* from ... ;
* server ... ;
* user ... ;
* time ... ;
  time-period-set { ... }
* session-acl ... ;
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
  plug-to ... ;
  source-address ... ;
  service-name ... ;
  default-port ... ;
  db-user ... ;
  client-altq ... ;
  server-altq ... ;

}

The second level ACL decides how to handle particular connection according to data contained in the connect (CN) string.

The `service-acl` section is derived from `acl-2` section prototype.

For detail description of it, see acl(5).

Changes to the `service-acl` section:

Item `parent-acl` used as `session-acl`.

Added items & subsections:

`plug-to` *addr*;

Final destination server.

addr (type: **sock**)

Address/port of final destination server.

If port is zero, then original port is used.

source-address [**client**] [**addr4** *addr4*] [**addr6** *addr6*] **cluster** [*cluster*];

source-address [**client**] [**addr4** *addr4*] [**addr6** *addr6*] [**physical**];

source-address [**client**] [**addr4** *addr4*] [**addr6** *addr6*] **no-fallback**;

Source address for outgoing connections to servers.

If omitted, the proper address of the proxy will be used, i.e. in the case of a cluster, the cluster address will be used.

If not specified by the SOURCE-PORT item, a generic port will be used.

The elements entered within this item will be used by the proxy until the first of them is applicable:

- The CLIENT keyword means the original client IP address is used. This mode will be succesful in all cases except mismatch of IP address families.

- The ADDR4/ADDR6 keyword-value pairs mean that the specified address is used for a connection of corresponding address family.

- The CLUSTER keyword means that one of cluster addresses will be used. By default, the main address of the bridge is used, however, any preferred alias address can be listed in the cluster list.- The PHYSICAL option means that the address of the physical interface is used instead of the cluster one.

- The DEFAULT option means the default behavior - i.e. using of the physical address.

- The NO-FALLBACK option means that if no other way of setting the address is acceptable, the session is rejected. Without this option, the system tries to find a suitable source IP address automatically.

client (type: **key**, optional)

addr4 *addr4* (type: **host**, optional, default: [0.0.0.0])

addr6 *addr6* (type: **host**, optional, default: [::])

<branching element> (type: **source-address-mode**, optional, default: physical)

cluster (type: **host-list**, optional, default: {})

Constraints:

Address family must respect the element's address family..

service-name [*set*];

Additional criteria for session-acl: SID/SERVICE NAME value.

set (type: **str-set**, optional, default: *)

default-port [*value*];

Default port when (PORT=?) attribute is missing in CN string or servername is present in SID w/o port specification.

value (type: **port**, optional, default: 1521)

db-user *names*;

This item switches database-user checking on and defines set of allowed user names. Checking is allowed only for known TNS protocol versions.

names **(type: str-set)**

client-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to client.

altq **(type: name of pf-queue, see** pf-queue**(5))**

queue name

paltq *paltq* **(type: name of pf-queue, see** pf-queue**(5), optional, default: NULL)**

priority queue name (if set, used for TCP ACK without data)

server-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to server.

altq **(type: name of pf-queue, see** pf-queue**(5))**

queue name

paltq *paltq* **(type: name of pf-queue, see** pf-queue**(5), optional, default: NULL)**

priority queue name (if set, used for TCP ACK without data)

[End of section `sqlnet-proxy.service-acl` description.]

[End of section `sqlnet-proxy` description.]

SEE ALSO

configuration(7), acl(5), application(5), auth(5), common(5), ipc(5), listen-on(5), log(5), monitoring(5), netio(5), pf-queue(5), source-address(5), time(5)

NAME

sqlnet-proxy.cfg — format of sqlnet-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **sqlnet-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **sqlnet-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

on-off (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see **radius**(5))

ldap-tls-reqcert-mode (see ldap(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

redirection-mode (see sqlnet-proxy(5))

ITEMS AND SECTIONS

Program **sqlnet-proxy** recognizes following items and sections:
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* sqlnet-proxy *name* { ... }
  ipv6-mode ... ;

Description:

interface *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

ldap-client-auth *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

oob-auth *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

`pf-queue` *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

`radius-client` *name* {

nas ... ;

groups ... ;

* server ... ;

}

The `radius-client` section is derived from `radius-client` section prototype. For detail description of it, see radius(5).

`resolver` *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`shared-dir` *name* {

path ... ;

}

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

`shared-file` *name* {

path ... ;

format ... ;

}

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name (type: **name** of **resolver**, see resolver(5))

sqlnet-proxy *name* {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

init-timeout ... ;

protocol-version ... ;

max-service-name-len ... ;

check-reserved-bits ... ;

connect-string-charset ... ;

connect-packet-sizelimit ... ;

* session-acl *name* { ... }

* service-acl *name* { ... }

}

The sqlnet-proxy section is derived from sqlnet-proxy section prototype. For detail description of it, see sqlnet-proxy(5).

ipv6-mode [*status*];

Enabling/Disabling IPv6 Mode.

status (type: **enabling**, optional, default: enable)

SEE ALSO

configuration(7), sqlnet-proxy(8), acl(5), auth(5), common(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), pf-queue(5), radius(5), resolver(5), source-address(5), sqlnet-proxy(5), sysctl(5), time(5), host-matching(7)

NAME

ssh — format of ssh component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ssh** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **ssh** configuration directives:

ip-version (see **common**(5))

osi4-proto (see **common**(5))

listen-on-sock (see **listen-on**(5))

ssh-key-type (name-usage obligatory)

SSH key types.

ssh-rsa

ssh-ed25519

ssh-proto (name-usage optional)

SSH protocol numbers.

ssh-2 (2)

ITEMS AND SECTIONS

Configuration of **ssh** library component consists of following prototypes:
* ssh-key2 ... ;
* ssh-server *name* { ... }

Description:

ssh-key2 *email type key* [*ignored*];

SSH Version 2 key.

email **(type: `str`)**

Owner email address.

type **(type: `ssh-key-type`)**

key **(type: `str`)**

ignored **(type: `str`, optional, default: <NULL>)**

Elem ignored, retained due to backward compatibility.

`ssh-server` *name* **{**

phase ... ;

* tag ... ;

listen-on { ... }

protocol ... ;

passwd-auth ... ;

ciphers ... ;

kex-algorithms ... ;

macs ... ;

* option ... ;

* subsystem ... ;

}

SSH server definition.

Each configured ssh server is started via standard Kernun startup mechanism (e.g. has its own rc-script) and as such will be handled by KAT program like regular proxy.

The ssh server configuration created by CML is based on values of this section configuration items. Additionally, following options are hardcoded as changes of default values:

- PermitRootLogin without-password
- ChallengeResponseAuthentication no

Constraints:

Addresses to listen on must be specified.

Items & subsections:

`phase` **[***number***];**

Application Startup Phase.

number **(type: `uint8`, optional, default: 30)**

Phase number; the lower one, the earlier start.

tag *value*;

Configuration factorization tag.

This feature allows admin to create groups of Kernun applications (specially proxies and servers) according to various aspects (belonging to one customer, applications of particular network traffic etc.).

Each application can have several tag attributes and the KAT tool can run some commands (like 'ps', 'start' atc.) for applications with or without given tag.

value **(type: str)**

Constraints:

Tag must contain letters, digits, hyphens and dots, only.

listen-on {

* socket ... ;

}

The listen-on section is derived from listen-on section prototype.
For detail description of it, see listen-on(5).
Changes to the listen-on section:

Item `non-transparent` used as `socket`.

Item `transparent` is not valid.

At least one address to listen on must be specified.

Item socket (see listen-on(5))

Element `port` is optional, default: 22.

Element `proto` is optional, default: tcp.

protocol *list*;

Protocol ordering.

If omitted, only SSHv2 is accepted.

list **(type: ssh-proto-list)**

List of protocol numbers.

Constraints:

Protocol list must contain just one item (ssh-2).

passwd-auth;

Enable password authentication for non-root users.

This item affects setting of PasswordAuthentication option to YES.

ciphers [*list*];

List of allowed ciphers.

list **(type: str, optional, default: "chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr,aes256-cbc,aes192-cbc,aes128-cbc")**

kex-algorithms [*list*];

List of allowed key exchange algorithms.

list (type: **str**, optional, default: "diffie-hellman-group14-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group-exchange-sha256,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521")

macs [*list*];

List of allowed MAC (message authentication code) algorithms.

list (type: **str**, optional, default: "umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,umac-128@openssh.com,hmac-sha2-256")

option *name value*;

Additional server configuration options.

name (type: **str**)

Option name.

value (type: **str**)

Option value.

subsystem *name cmd*;

External subsystem definition.

name (type: **str**)

Subsystem name.

cmd (type: **str**)

Command to execute.

[End of section `ssh-server` description.]

SEE ALSO

configuration(7), common(5), listen-on(5), sshd config(5)

NAME

ssl — format of ssl component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **ssl** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **ssl** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

ssl-ver (name-usage obligatory)

SSL/TLS protocol versions.

SSLv3

SSL version 3

TLSv1

TLS version 1

TLSv1-1

TLS version 1.1

TLSv1-2

TLS version 1.2

extension-op (name-usage obligatory)

Certificate Extensions Operations.

keep

remove

veri-fail-action (name-usage obligatory)

Certificate verification failure actions.

ignore

Verification status is absolutely ignored. UNRECOMMENDED OPTION.

pass

Verification status is ignored, the certificate used on the client side is signed by a special, untrusted CA, however. Thus, the final decision is passed to the client.

error

The client side connection is established with fully trusted certificate and then the proxy sends a user readable error message.

fail

The connection establishing fails.

auth-cert-type (name-usage obligatory)

Authorized certificate store type.

kernun-dist

Certificate list from Kernun distribution

file

Own certificate list in PEM file

dir

Own directory with PEM certificate files

distrusted-cert-type (name-usage obligatory)

Distrusted (blacklist) certificate store type.

kernun-dist

Certificate list from Kernun distribution

file

Own certificate list in PEM file

none

No distrusted certificates are used

ITEMS AND SECTIONS

Configuration of **ssl** library component consists of following prototypes:
ssl-session-cache { ... }
* fake-cert *name* { ... }
* ssl-params *name* { ... }
* ssl-cert-match ... ;

Description:

ssl-session-cache {

capacity ... ;

dir ... ;

lock ... ;

}

Cache of active SSL/TLS sessions usable for session resumption.

Constraints:

Item DIR (session cache directory) must be set.

Items & subsections:

capacity [*server* [*client*]];

Maximum number of sessions in the cache.

server **(type: uint16, optional, default: 0)**

sessions in which proxy on server side

client **(type: uint16, optional, default: 0)**

sessions in which proxy on client side

dir *val*;

Directory used to store files with session information.

val **(type: str)**

lock [*path*];

Lock for exclusive access to the cache.

path **(type: str, optional, default: <NULL>)**

If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

[End of section `ssl-session-cache` description.]

fake-cert *name* {

key ... ;

auth-ca ... ;

fail-ca ... ;

* extension ... ;

purge ... ;

}

Server certificate faking parameters.

This section defines parameters of certificates generated by proxies when SSL inspection is turned on. The new certificate contains a copy of important attributes from the original server certificate, it is signed by a trusted Kernun CA and stored into a certificate cache (file with name /data/fake-cert/<SECTION>/<VER><FINGERPRINT>.<NUM> where:

- <SECTION> is the FAKE-CERT section name

- <FINGERPRINT> is the certificate fingerprint

- <VER> is certificate verification status

- <NUM> is certificate distinguishing number.

The verification status reflects result of the original certificate verification - for trusted ones, the 'C' is used, while otherwise the 'F' is used.

Constraints:

Private key used in faked certificates (KEY) must be set.

Trusted certification authority (AUTH-CA) must be set.

Items & subsections:

`key` ***private-key*****;**

Private key used in faked certificates.

private-key **(type: `name` of `shared-file`,** see **common(5))**

`auth-ca` ***private-key certificate*****;**

Private key and certificate of trusted CA.

This certificate and key are used as issuer identity for new certificates made from properly verified server ones.

private-key **(type: `name` of `shared-file`,** see **common(5))**

certificate **(type: `name` of `shared-file`,** see **common(5))**

`fail-ca` ***private-key certificate*****;**

Private key and certificate of untrusted CA.

This certificate and key are used as issuer identity for new certificates made from untrusted server ones.

If this item is not used, faking attempts for untrusted certificates leads to the fatal error.

private-key **(type: `name` of `shared-file`,** see **common(5))**

certificate **(type: `name` of `shared-file`,** see **common(5))**

`extension` ***op*** **[*****names*****];**

Faked certificate Extensions handling.

The new certificate contains all attributes of the original certificate, by default. Copying of some X.509 Extensions could cause problem, so handling of them can be managed by this item. Namely, following Extensions are removed by default, if not stated otherwise in the configuration:

- X509v3 Authority Key Identifier

- Authority Information Access.

op **(type: `extension-op`)**

names **(type: `str-set`, optional, default: *)**

`purge` **[*days*];**

days **(type: `uint16`, optional, default: 7)**

Day limit - certificates unaccessed for this time are purged.

[End of section `fake-cert` description.]

`ssl-params` *name* **{**

versions ... ;

ciphers ... ;

tcp-eof ... ;

id ... ;

* auth-cert ... ;

distrusted-certs ... ;

dont-check-crl ... ;

* crl ... ;

verify-peer ... ;

cache-timeout ... ;

use-ticket ... ;

enable-renegotiation ... ;

fake-cert ... ;

prefer server ciphers ... ;

enable-ecdh ... ;

}

SSL parameters.

Constraints:

ID and FAKE-CERT are mutually exclusive.

Items & subsections:

`versions` *ver*;

SSL/TLS protocol versions supported.

If omitted, only TLSv1.2 protocol is supported.

ver **(type: `ssl-ver-set`)**

`ciphers` **[*val*];**

List of permitted ciphers, see ciphers(1).

val **(type: `str`, optional, default: "ALL:!MEDIUM:!LOW:!RC4:!DES:!3DES:!SEED:!IDEA:!NULL:!eNULL:!aNULL:!ADH")**

tcp-eof [*val*];

Treat closing TCP connection without previous close notify as correct session termination and not a protocol violation.

val (type: **yes-no**, optional, default: yes)

id *private-key certificate*;

Private key and certificate.

The certificate should also contain certificates of intermediate certification authorities, if there are any. It should not contain the certificate of the root certification authority however.

private-key (type: **name** of **shared-file**, see common(5))

certificate (type: **name** of **shared-file**, see common(5))

auth-cert kernun-dist;

auth-cert file *file*;

auth-cert dir *dir*;

Certificates of trusted certification authorities.

This item should usually contain only certificates of root certification authorities.

<branching element> (type: **auth-cert-type**)

Certificate source type.

file (type: **name** of **shared-file**, see common(5))

file with certificates

dir (type: **name** of **shared-dir**, see common(5))

directory with hashed certificate files

distrusted-certs kernun-dist;

distrusted-certs file *file*;

distrusted-certs none;

Distrusted certificates. Certificates listed in this item are considered as a "blacklist" when verifying the certificate. If any of the certificates that build up the certificate chain is listed in DISTRUSTED-CERTIFICATES, the verification is evaluated false.

<branching element> (type: **distrusted-cert-type**)

Certificate source type.

file (type: **name** of **shared-file**, see common(5))

file with distrusted certificates

dont-check-crl;

Do not check CRL validity when verifying certificates.

crl [**missing**] *file*;

Certification revocation list (loaded once during proxy startup).

missing (type: **key**, optional)

CRL may be missing

file (type: **name** of **shared-file**, see common(5))

verify-peer disable;

verify-peer [enable] [allow-no-cert];

Peer verification request.

SSL/TLS handshake fails if peer does not send a certificate or it sends a certificate that cannot be verified. Client does not send a certificate until it is requested by VERIFY-PEER. Omitting this item has the same meaning as using with the DISABLE keyword.

<**branching element**> (type: **enabling**, optional, default: **enable**)

allow-no-cert (type: **key**, optional)

Allow clients which do not present a certificate.

cache-timeout [*sec*];

Maximum time since a session creation when the session can be resumed. After this time, no new connections may be established in the same session, although the existing connection can continue. Setting this to zero disables session resumption.

sec (type: **uint32**, optional, default: **0**)

use-ticket [*val*];

Normally clients and servers will, where possible, transparently make use of RFC4507bis tickets for stateless session resumption. If this option is set to 0, this functionality is disabled and tickets will not be used by client or server.

val (type: **yes-no**, optional, default: **yes**)

enable-renegotiation;

Permits session renegotiation. Do not enable this unless you understand all related negative security consequences!

fake-cert *faking action*;

Server certificate faking parameters.

If this section is used, the proxy generates a faked certificate of each server for the client connection, instead of using a fixed one.

faking (type: **name** of **fake-cert**, see above)

Faking parameters

action (type: **veri-fail-action**)

Reaction when server certificate verification fails.

prefer_server_ciphers [*val*];

Set on servers to choose the cipher according to the server's preferences

val (type: **yes-no**, optional, default: **yes**)

enable-ecdh [*val*];

Enable or disable ephemeral ECDH

val (type: **yes-no**, optional, default: **yes**)

[End of section `ssl-params` description.]

`ssl-cert-match` [**`subject`** *subject*] [**`issuer`** *issuer*];

Matching values from SSL certificate. If not used, peer need not present a cerficate or the client certificate need not be verified correctly. If used, peer must send a valid certificate and its content must match.

`subject` *subject* (type: **`str-set`**, optional, default: *)

acceptable certificate subjects

`issuer` *issuer* (type: **`str-set`**, optional, default: *)

acceptable certificate issuers

SEE ALSO

configuration(7), ciphers(1), common(5)

NAME

sysctl — format of sysctl component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **sysctl** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **sysctl** configuration directives:

log-in-vain-proto (name-usage obligatory)

disable
: Log in vain is disabled

tcp-udp
: Log in vain is enabled for TCP and UDP

tcp
: Log in vain is enabled for TCP

udp
: Log in vain is enabled for UDP

blackhole-proto (name-usage obligatory)

disable
: Blackhole is disabled

tcp-udp
: Blackhole is enabled for TCP and UDP

tcp
: Blackhole is enabled for TCP

udp
: Blackhole is enabled for UDP

ITEMS AND SECTIONS

Configuration of **sysctl** library component consists of following prototypes:

portrange ... ;
sysctl { ... }

Description:

portrange *lo hi*;

Port range specification.

lo (type: **port**)

hi (type: **port**)

sysctl {

* variable ... ;
portrange-default ... ;
portrange-high ... ;
portrange-low ... ;
portrange-reserved ... ;
somaxconn ... ;
log-in-vain ... ;
blackhole ... ;
}

System kernel variables definition.

Source for /etc/sysctl.conf file.

Items & subsections:

variable *name value*;

Kernel variable definition.

name (type: **str**)

Variable name.

value (type: **str**)

Variable value.

Constraints:

Variable name must contain alphanumeric chars and dots only.

portrange-default [*lo* [*hi*]];

Port range reserved by system.

lo (type: **port**, optional, default: **49152**)

hi **(type: port, optional, default: 65535)**

portrange-high [*lo* [*hi*]];

Port range reserved by system.

lo **(type: port, optional, default: 49152)**

hi **(type: port, optional, default: 65535)**

portrange-low [*lo* [*hi*]];

Port range reserved by system.

lo **(type: port, optional, default: 1023)**

hi **(type: port, optional, default: 600)**

portrange-reserved [*lo* [*hi*]];

Port range that can only be bind-ed by user with UID 0 (root).

lo **(type: port, optional, default: 0)**

hi **(type: port, optional, default: 1)**

somaxconn [*number*];

Listen queue size.

System default for listen queue size for accepting new TCP connections. If the maximum of MAX-CHILDREN values among all TCP based proxies is higher, the maximum is used instead of this value.

number **(type: uint32, optional, default: 16384)**

log-in-vain [*proto*];

Log incomming packets to closed ports

proto **(type: log-in-vain-proto, optional, default: disable)**

blackhole [*proto*];

Do not send RST on segments to closed ports

proto **(type: blackhole-proto, optional, default: disable)**

[End of section `sysctl` description.]

SEE ALSO

configuration(7)

NAME

system — format of system component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **system** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **system** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

language (see **common**(5))

nls (see **common**(5))

on-off (see **common**(5))

genesis (see **common**(5))

permission (see **common**(5))

direction (see **common**(5))

name-selection (see **common**(5))

destination (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

in-out (see **common**(5))

report-mode (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see common(5))

range-op (see common(5))

yes-no-always (see common(5))

task-frequency (see common(5))

week-day (see time(5))

month (see time(5))

lock-type (see ipc(5))

radius-attr (see radius(5))

ldap-tls-reqcert-mode (see ldap(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

user-match-mode (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

antivirus-protocol (see antivirus(5))

virus-status (see antivirus(5))

database-source (see antivirus(5))

source-address-mode (see source-address(5))

source-port-mode (see source-address(5))

accept-deny (see mod-html-filter(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

header-op (see acl(5))

product-type (see license(5))

component-group (see license(5))

component-type (see license(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

user-type (name-usage obligatory)

Kernun user type.

admin

audit

route-flag (name-usage obligatory)

Route flags.

cloning

xresolve

iface

static

nostatic

usb-auto-setup-policy (name-usage obligatory)

Automatically apply configuration from attached USB devices.

auto_decide

enable

disable

dbglev (see log(5))

logfail-mode (see log(5))

dns-type (see resolver(5))

dns-opcode (see resolver(5))

dns-response (see resolver(5))

dns-qaction (see resolver(5))

dns-raction (see resolver(5))

dns-fake (see resolver(5))

xfr-mode (see resolver(5))

udp-session-type (see udpserver(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

proc-priority (see application(5))

pf-osi4-proto (see packet-filter(5))

icmp-type (see packet-filter(5))

pf-scheduler (see packet-filter(5))

pf-proc-mode (see packet-filter(5))

ids-agent-log-level (see adaptive-firewall(5))

ids-agent-detection-direction (see adaptive-firewall(5))

ids-agent-protocol (see adaptive-firewall(5))

ids-agent-rule-action (see adaptive-firewall(5))

ids-agent-threshold-type (see adaptive-firewall(5))

ids-agent-threshold-track-by (see adaptive-firewall(5))

ids-agent-rate-filter-track-by (see adaptive-firewall(5))

ids-agent-suppress-direction (see adaptive-firewall(5))

policy-level (see adaptive-firewall(5))

ids-agent-rules-download-type (see update(5))

forward (see nameserver(5))

atr-strategy (see atr(5))

atr-fallback (see atr(5))

pike-control-type (see pike(5))

ntp-rest-flag (see ntp(5))

ovpn-protocols (see openvpn(5))

ovpn-remote-proto (see openvpn(5))

ovpn-comp-lzo-mode (see openvpn(5))

ovpn-cert-types (see openvpn(5))

ovpn-cipher-algs (see openvpn(5))

ovpn-redirect-gateway-flags (see openvpn(5))

ovpn-dhcp-option (see openvpn(5))

ovpn-topology (see openvpn(5))

ovpn-local-scope (see openvpn(5))

tls-mat-variants (see openvpn(5))

ipsec-encryption1 (see ipsec(5))

ipsec-encryption2 (see ipsec(5))

ipsec-hash1 (see ipsec(5))

ipsec-auth2 (see ipsec(5))

ipsec-dh-group (see ipsec(5))

ipsec-tunnel-sa-mode (see ipsec(5))

ipsec-auth-method (see ipsec(5))

ipsec-protocol (see ipsec(5))

ipsec-remote-mode (see ipsec(5))

ipsec-rekey-mode (see ipsec(5))

snmpd-disk-mode (see snmpd(5))

snmpd-source-mode (see snmpd(5))

snmpd-view-type (see snmpd(5))

snmpd-security-level (see snmpd(5))

snmpd-auth-hash (see snmpd(5))

snmpd-encr-alg (see snmpd(5))

ssh-key-type (see ssh(5))

ssh-proto (see ssh(5))

export-import-mode (see router(5))

ospf-authentication (see router(5))

ospf-area-id-mode (see router(5))

ssl-ver (see ssl(5))

extension-op (see ssl(5))

veri-fail-action (see ssl(5))

auth-cert-type (see ssl(5))

distrusted-cert-type (see ssl(5))

data-match-action (see mod-match(5))

dns-name-type (see dns-proxy(5))

pass-remove (see ftp-proxy(5))

data-type (see ftp-proxy(5))

ftp-cmd (see ftp-proxy(5))

clear-web-db-category (see clear-web-db(5))

clear-web-db-match-mode (see clear-web-db(5))

replace-authorization-mode (see http-proxy(5))

proxy-via (see http-proxy(5))

http-protocol (see http-proxy(5))

http-scheme (see http-proxy(5))

cookie-table-clean (see http-proxy(5))

accept-gzip (see http-proxy(5))

content-gzip (see http-proxy(5))

http-redirect (see http-proxy(5))

kerberos-user-match (see http-proxy(5))

ldap-select (see http-proxy(5))

auth-headers (see http-proxy(5))

sni-result (see http-proxy(5))

smtp-error (see mod-mail-doc(5))

mail-reaction (see mod-mail-doc(5))

mail-fallback (see mod-mail-doc(5))

mime-header-check-type (see mod-mail-doc(5))

imap4-cmd (see imap4-proxy(5))

imap4-capa (see imap4-proxy(5))

pop3-cmd (see pop3-proxy(5))

pop3-capa (see pop3-proxy(5))

peer (see sip-proxy(5))

smtp-size-usage (see smtp-proxy(5))

ssl-startup-mode (see smtp-proxy(5))

postfix-security-level (see smtp-proxy(5))

postfix-transport-map-mode (see smtp-proxy(5))

smtp-err-switch (see smtp-proxy(5))

spf-result (see smtp-proxy(5))

spf-modes (see smtp-proxy(5))

redirection-mode (see sqlnet-proxy(5))

session-protocol (see proxy-ng(5))

json-type (see proxy-ng(5))

http-version (see proxy-ng(5))

ITEMS AND SECTIONS

Configuration of **system** library component consists of following prototypes:
* system *name* { ... }

Description:

system *name* **{**

product ... ;

admin ... ;

hostname ... ;

domain ... ;

kernun-root ... ;

usb-auto-setup ... ;

apply-host ... ;

config-sync ... ;

users { ... }

sysctl { ... }

* interface *name* { ... }

ipv6-router ... ;

ipv6-addrctl { ... }

pikemon { ... }

routes { ... }

rc-conf { ... }

hosts-table { ... }

* rotate-log *name* { ... }
  ntp { ... }
  dhcp-server { ... }
  dhcp6-server { ... }
  crontab { ... }
  periodic-conf { ... }
  local-mailer { ... }
* ssh-server *name* { ... }
  ssh-keys { ... }
  ica-auto ... ;
  icamd { ... }
  icasd { ... }
  watch { ... }
* acl *name* { ... }
  use-services ... ;
  use-resolver ... ;
* resolver *name* { ... }
* nameserver *name* { ... }
* ns-list *name* { ... }
* atrmon *name* { ... }
* pf-queue *name* { ... }
  packet-filter { ... }
  adaptive-firewall { ... }
  alertd { ... }
  bird4 { ... }
  bird6 { ... }
  rtadvd { ... }
* ssl-params *name* { ... }
* fake-cert *name* { ... }
* html-filter *name* { ... }
* mail-filter *name* { ... }
* aproxy *name* { ... }
* radius-client *name* { ... }
* ldap-client-auth *name* { ... }

* oob-auth *name* { ... }
* antivirus *name* { ... }
* antispam *name* { ... }
* smtp-forwarder *name* { ... }
* web-filter *name* { ... }
  clear-web-db { ... }
* openvpn *name* { ... }
  ipsec-global { ... }
* ipsec-remote *name* { ... }
* ipsec *name* { ... }
* data-match *name* { ... }
* ntlm-auth *name* { ... }
* kerberos-auth *name* { ... }
  cwcatd { ... }
  snmpd { ... }
  http-cache { ... }
  update { ... }
  feedback { ... }
  stats { ... }
  stats-daily { ... }
  stats-weekly { ... }
  stats-monthly { ... }
* tcp-proxy *name* { ... }
* udp-proxy *name* { ... }
* dns-proxy *name* { ... }
* ftp-proxy *name* { ... }
* gk-proxy *name* { ... }
* h323-proxy *name* { ... }
* http-proxy *name* { ... }
* icap-server *name* { ... }
* imap4-proxy *name* { ... }
* pop3-proxy *name* { ... }
* sip-proxy *name* { ... }
* smtp-proxy *name* { ... }

* sqlnet-proxy *name* { ... }

* proxy-ng *name* { ... }

  proxy-ng-transp-ports ... ;

}

Description of one firewall system.

Constraints:

PRODUCT should be specified.

Some configured components are not licensed.

Hostname must be specified.

Domainname must be specified.

Interfaces must be specified.

Source for /etc/services must be specified.

Name resolver configuration must be specified.

System name resolvers must use standard port.

DEFAULT router allowed only if DHCP-CLIENT not used.

Crontab content must be specified.

All interfaces must use unique device names.

All configured email domains must be handled by some SMTP-FORWARDER.

At most one interface with DHCP-CLIENT allowed.

Cannot non-transparently listen on dynamic interfaces.

Openvpn sections must refer interface of type TUN or TAP.

IPSEC sections can refer interface of type GIF or GRE only.

Addresses used in OPENVPN section must respect INTERFACE network range.

Address pushing in OPENVPN section must respect INTERFACE type.

Addresses pushing in OPENVPN section must not collide.

For every IPSEC section must exist IPSEC-REMOTE section with proper remote address.

NTLM-AUTH and KERBEROS-AUTH are mutually exclusive.

At most one NTLM-AUTH section allowed.

At most one KERBEROS-AUTH section allowed.

Clear Web database updates should be configured if Clear Web category matching is used.

Data MIME database required by DATA-MATCH with MATCH-DATA-MIME set.

Item ICA-AUTO is mutually exclusive with sections ICASD or ICAMD.

Item ICA-AUTO can be used with PIKEMON only.

LISTEN-SOCKET-ID must be consistent within PF and PROXY-NG.

Item ADAPTIVE-FIREWALL.IDS-AGENT.RULES.MODIFY-RULES requires item UPDATE.ADAPTIVE-FIREWALL to be enabled.

Item ADAPTIVE-FIREWALL.IDS-AGENT.RULES.ENABLE-RULES requires item UPDATE.ADAPTIVE-FIREWALL to be enabled.

Item ADAPTIVE-FIREWALL.IDS-AGENT.RULES.DISABLE-RULES requires item UPDATE.ADAPTIVE-FIREWALL to be enabled.

Item ADAPTIVE-FIREWALL.IDS-AGENT.RULES.CHANGE-RULES-TO-BLOCK requires item UPDATE.ADAPTIVE-FIREWALL to be enabled.

Section SYSTEM.ADAPTIVE-FIREWALL requires section SYSTEM.PACKET-FILTER.

Section ADAPTIVE-FIREWALL.STATS-DAILY requires section PACKET-FILTER.STATS-DAILY to be enabled because AF is technically part of Packet Filter.

Section ADAPTIVE-FIREWALL.STATS-WEEKLY requires section PACKET-FILTER.STATS-WEEKLY to be enabled because AF is technically part of Packet Filter.

Section ADAPTIVE-FIREWALL.STATS-MONTHLY requires section PACKET-FILTER.STATS-MONTHLY to be enabled because AF is technically part of Packet Filter.

Items & subsections:

`product` *product components* [**`groups`** *groups*] [**`upgrade`** *upgrade*];

Specification of the product installed on this system.

product (type: **`product-type`**)

Type of the product.

components (type: **`component-type-list`**)

List of licensed components.

`groups` *groups* (type: **`component-group-list`**, optional, default: {})

List of licensed component groups.

`upgrade` *upgrade* (type: **`str`**, optional, default: "unlimited")

Upgrade date from a license.

Constraints:

Upgrade must be "unlimited" or a date in format YYYY-MM-DD.

`admin` *system* [*contact*];

Firewall administrator and contact e-mail addresses.

system (type: **`str`**)

The technical administrator(s) of the system; an address or set of comma separated adresses of persons responsible for system maintenance.

contact (type: **`str`**, optional, default: <**NULL**>)

The policy administator; an address of person responsible for system configuration. If not defined, the technical administration is used instead.

Constraints:

Administrator contact must comply with RFC.

hostname *name*;

System name.

name **(type: str)**

Constraints:

Hostname should not contain domain part.

domain *name*;

Domain name.

name **(type: str)**

kernun-root [*path*];

Path to Kernun installation root directory.

path **(type: str, optional, default: "/usr/local/kernun")**

Constraints:

Path must be absolute and must not contain punctuation chars.

usb-auto-setup [*value*];

Policy for automatic configuration application from attached USB devices

value **(type: usb-auto-setup-policy, optional, default: auto decide)**

apply-host *addr*;

Address to connect to by ssh when applying remotely.

If omitted, KAT /APPLY command will force local application.

If used, KAT /APPLY command will use local application only if the machine hostname is exactly HOSTNAME.DOMAIN.

addr **(type: sock)**

config-sync *systems*;

Keep configuration synchronized among the listed systems

systems **(type: str-list)**

users {

* user *name* { ... }

}

Kernun users.

Items & subsections:

user *name* **{**

role ... ;

full-name ... ;

* ssh-key ... ;

}

Constraints:

User role must be specified.

Items & subsections:

`role` *type*;

User role.

There are two kinds of Kernun users:

- ADMINistrators are root-equivalent users
- AUDITors can only view system configuration and logs.

type (type: **`user-type`**)

`full-name` [*gecos*];

Full name of user.

gecos (type: **`str`**, optional, default: "&")

Constraints:

Full name must not contain colon (':').

`ssh-key` *email type key* [*ignored*];

SSH Version 2 key.

email (type: **`str`**)

Owner email address.

type (type: **`ssh-key-type`**)

key (type: **`str`**)

ignored (type: **`str`**, optional, default: <**NULL**>)

Elem ignored, retained due to backward compatibility.

[End of section `system.users.user` description.]

[End of section `system.users` description.]

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`interface` *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`ipv6-router` [*enable*];

Operate as an IPv6 router.

enable **(type: `yes-no`, optional, default: yes)**

`ipv6-addrctl` {

* rule ... ;

}

Defines the configuration table for the IPv4/6 address selection algorithm from RFC 3484. The generated address selection table is stored in /etc/ip6addrctl.conf and managed by command ip6addrctl. If this section does not exist, a default table will be generated. Preference of IPv4 or IPv6 addresses in the default table is controlled by item PROTO in the section RESOLVER referenced by SYSTEM.USE-RESOLVER.

Items & subsections:

`rule` *prefix precedence label*;

A single policy table entry.

prefix **(type: `net`)**

precedence **(type: `uint16`)**

label **(type: `uint16`)**

[End of section `system.ipv6-addrctl` description.]

pikemon {

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

udpserver { ... }

priority ... ;

status-file ... ;

hmac ... ;

devd-socket ... ;

garp-keepalive ... ;

* virtual-cluster *name* { ... }

}

The `pikemon` section is derived from `pikemon` section prototype. For detail description of it, see pike(5).

`routes` {

default ... ;

default6 ... ;

* static *name* { ... }

}

Routing table definition.

Items & subsections:

`default` *gw*;

Default route.

***gw* (type: `host`)**

Router IP address.

default6 *gw*;

Default IPv6 route.

gw **(type: host)**

Router IP address.

static *name* {

dest ... ;

gw ... ;

flags ... ;

}

Static route.

Constraints:

Route destination must be specified.

Router address must be specified.

Dest and gateway must be of the same internet family.

Items & subsections:

dest *dst*;

Route destination.

dst **(type: net)**

gw *gw*;

Router (gateway).

gw **(type: host)**

Router IP address.

flags *set*;

Route flags.

set **(type: route-flag-list)**

[End of section `system.routes.static` description.]

[End of section `system.routes` description.]

rc-conf {

no-kld-list ... ;

* set-env ... ;

* append-env ... ;

}

Additional settings to /etc/rc.conf.

By default, CML generates to rc.conf file following variables:

- kld list (for network transparency modules used by proxy-ng)
- hostname (from HOSTNAME and DOMAIN items)
- network interfaces (from INTERFACE sections)

- default router (from ROUTES section)
- static routes (from ROUTES.STATIC sections)
- syslogd flags ("-ss" and sockets for CHROOT-DIRs)
- devfs set rulesets and devfs system ruleset
- local startup (adds Kernun rc.d directory)
- pf enable (YES)
- sendmail enable (NONE)
- sendmail msp queue enable (NO)
- postfix enable (YES)
- fsck y enable (YES)

Additional variables can be specified in this section.

Even the predefined variables can be modified by adding variable redefinition like SET-ENV var "$var ...";.

Items & subsections:

`no-kld-list;`

Do not generate kld list variable.

After changing this, it is necessary to manually load or unload kernel modules mac bindany and pf transp

`set-env` ***name value*****;**

Set rc-conf variable.

name **(type: `str`)**

Variable name.

value **(type: `str`)**

Variable value.

Constraints:

Variable name must contain alphanumeric chars only.

`append-env` ***name value*****;**

Modify rc-conf variable.

Variable value is just extended (appending the new value), not replaced.

name **(type: `str`)**

Variable name.

value **(type: `str`)**

Variable value.

Constraints:

Variable name must contain alphanumeric chars only.

[End of section `system.rc-conf` description.]

`hosts-table {`

* host ... ;

}

Host table.

This section defines known machines and their addresses. It servers primarily as a source for the /etc/hosts file. If the DHCP-SERVER is enabled in particular SYSTEM, all hosts with an IPv4 address and a MAC address in this table are included into dhcpd.conf. If the DHCP6-SERVER is enabled in particular SYSTEM, all hosts with an IPv6 address and a DUID in this table are included into dhcpd6.conf. If a NAME-SERVER with a ZONE is enabled in particular SYSTEM, all hosts with a proper name are included into proper files.

Items & subsections:

`host` ***address names*** **[***mac* **[***dhcp-opt***]];**

address **(type: `addr`)**

Host IP address.

names **(type: `str-list`)**

Host name and aliases.

mac **(type: `str`, optional, default: <NULL>)**

MAC address (for IPv4) or client's DUID (for IPv6). The acceptable formats are "xx:xx:xx:xx:xx:xx", "xx-xx-xx-xx-xx-xx" and "xxxx.xxxx.xxxx".

dhcp-opt **(type: `str`, optional, default: <NULL>)**

DHCP options.

Constraints:

Name list must not be empty.

Hostnames must comply RFC1034.

MAC address must be in colon, dash or dot separated format.

[End of section `system.hosts-table` description.]

`rotate-log` ***name*** **{**

rotate ... ;

* file ... ;

}

Standard system log files rotation description.

All files referenced in one ROTATE-LOG section use the same rotation policy defined by the ROTATE item. The default policy (if ROTATE item omitted) is daily rotation.

Files not referenced in any ROTATE-LOG section (neither elsewhere in the CML) are rotated according to the /etc/newsyslog.conf file.

Items & subsections:

`rotate` **[`user`** ***user***] **[`group`** ***group***] **[`mode`** ***mode***] **[`count`** ***count***] **[`size`** ***size***] **[***when* **[***zip***]];**

Log file rotation description.

Use the SIZE elem if log file size criterion required. Use the WHEN elem if periodical rotation required. If used both SIZE and WHEN elems, the log file is rotated at a proper time only if size limit is reached.

user *user* **(type: str, optional, default: "root")**

Log file owner - user.

group *group* **(type: str, optional, default: "wheel")**

Log file owner - group.

mode *mode* **(type: uint16, optional, default: 640)**

Log file permissions.

count *count* **(type: uint16, optional, default: 31)**

Number of days being archived.

size *size* **(type: uint16, optional, default: 0)**

Size limit for rotation in KB (ignore log file size if omitted).

when **(type: time-cond, optional, default: anytime)**

Rotation periodicity (use SIZE condition if omitted).

zip **(type: zip-mode, optional, default: bzip2)**

Zipping mode.

Constraints:

Use either size criterion or defined periodicity.

file *name* **[***pidfile* **[***signo***]];**

Particular log file description.

For the PIDFILE and SIGNO elems description, see the newsyslog.conf(5) manual page.

name **(type: str)**

pidfile **(type: str, optional, default: <NULL>)**

signo **(type: uint8, optional, default: 0)**

Constraints:

Log file name must be absolute and must not contain punctuation chars.

PID file name must be absolute and must not contain punctuation chars.

[End of section `system.rotate-log` description.]

ntp {

phase ... ;

* tag ... ;

cfg-resolution ... ;

drift-file ... ;

* peer ... ;

* server ... ;

* clock ... ;

* restrict ... ;

}

The ntp section is derived from ntp section prototype. For detail description of it, see ntp(5).

dhcp-server {

phase ... ;

* tag ... ;

lease-file ... ;

default-lease-time ... ;

max-lease-time ... ;

* domain ... ;

* name-server ... ;

* time-server ... ;

* router ... ;

* raw ... ;

* subnet *name* { ... }

failover { ... }

}

The dhcp-server section is derived from dhcp-server section prototype. For detail description of it, see dhcp-server(5).

dhcp6-server {

phase ... ;

* tag ... ;

lease-file ... ;

default-lease-time ... ;

max-lease-time ... ;

* domain ... ;

* name-server ... ;

* raw ... ;

* subnet *name* { ... }

}

The dhcp6-server section is derived from dhcp6-server section prototype. For detail description of it, see dhcp-server(5).

crontab {

mailto ... ;

* set-env ... ;

* plan ... ;

* monthly ... ;

* weekly ... ;

* daily ... ;

* hourly ... ;

* every ... ;

}

Cron table definition.

No "default content" of crontab is preserved, all table items must be specified here. Typical content of crontab can be found in file samples/crontab.cml that you can include into your configuration and use here. See instructions in the file.

Items & subsections:

`mailto` *addr*;

Set MAILTO crontab variable.

This address is used by cron to send skripts output. Setting via SET-ENV is allowed, however, setting by this item should be prefererred. If undefined, the SYSTEM.ADMIN value is used.

addr **(type: `str`)**

Email address(es).

`set-env` *name value*;

Set crontab variable.

name **(type: `str`)**

Variable name.

value **(type: `str`)**

Variable value.

Constraints:

Variable name must contain alphanumeric chars only.

`plan` *line*;

Crontab (raw) line.

line **(type: `str`)**

`monthly at` *at* **[`by`** *by***]** *cmd* **[`report`** *report***];**

Run task every month.

`at` *at* **(type: `time`)**

Starting time of task (hhmm).

`by` *by* **(type: `str`, optional, default: "root")**

cmd **(type: `str`)**

`report` *report* **(type: `report-mode`, optional, default: nothing=0)**

Task output (stdout and stderr) delivery.

`weekly on` *on* **`at`** *at* **[`by`** *by***]** *cmd* **[`report`** *report***];**

Run task every week.

`on` *on* **(type: `week-day`)**

Weekday of execution.

`at` *at* **(type: `time`)**

Starting time of task (hhmm).

`by` *by* **(type: `str`, optional, default: "root")**

cmd **(type: str)**

report *report* **(type: report-mode, optional, default: nothing=0)**

Task output (stdout and stderr) delivery.

daily at *at* **[by** *by***]** *cmd* **[report** *report***];**

Run task every day.

at *at* **(type: time)**

Starting time of task (hhmm).

by *by* **(type: str, optional, default: "root")**

cmd **(type: str)**

report *report* **(type: report-mode, optional, default: nothing=0)**

Task output (stdout and stderr) delivery.

hourly at *at* **[by** *by***]** *cmd* **[report** *report***];**

Run task every hour.

at *at* **(type: time)**

Starting time of task (mm, hours ignored).

by *by* **(type: str, optional, default: "root")**

cmd **(type: str)**

report *report* **(type: report-mode, optional, default: nothing=0)**

Task output (stdout and stderr) delivery.

every *min* **at** *at* **[by** *by***]** *cmd* **[report** *report***];**

Run task every time range given in minutes.

min **(type: time)**

Period (mm, hours ignored).

at *at* **(type: time)**

Starting time of task (mm, hours ignored).

by *by* **(type: str, optional, default: "root")**

cmd **(type: str)**

report *report* **(type: report-mode, optional, default: nothing=0)**

Task output (stdout and stderr) delivery.

[End of section `system.crontab` description.]

periodic-conf {

mailto ... ;

* set-env ... ;

}

Periodic job configuration information.

The /etc/periodic.conf file content (see periodic.conf(5)) is defined here. Typical content of the file can be found in file samples/crontab.cml that you can include into your configuration and use here. See instructions in the file.

If undefined, the file remains untouched.

Items & subsections:

`mailto` *addr*;

Set MAILTO crontab variable.

This address will be used as value of several variables 'daily output', 'weekly output', 'monthly output', 'daily status security output', 'weekly status security output' and 'monthly status security output'.

If undefined, the SYSTEM.ADMIN value is used.

***addr* (type: `str`)**

Email address(es).

`set-env` *name value*;

Set periodic.conf variable.

***name* (type: `str`)**

Variable name.

***value* (type: `str`)**

Variable value.

Constraints:

Variable name must contain alphanumeric chars only.

[End of section `system.periodic-conf` description.]

`local-mailer` {

phase ... ;

* tag ... ;

relayhost ... ;

source-address ... ;

myhostname ... ;

smtp-helo-name ... ;

myorigin ... ;

inet-protocol ... ;

relay-domains ... ;

mydestinations ... ;

mynetworks ... ;

message-size-limit ... ;

bounce-size-limit ... ;

bounce-queue-lifetime ... ;

delay-warning-time ... ;

tls { ... }

* set-var ... ;

master-cf ... ;

smtpd-option ... ;

transport-map ... ;

}

MTA used for sending mails originated at firewall.

The `local-mailer` section is derived from `smtp-agent` section prototype. For detail description of it, see smtp-proxy(5).

Changes to the `local-mailer` section:

Cannost use automatic transport map for local-mailer.

`ssh-server` *name* {

phase ... ;

* tag ... ;

listen-on { ... }

protocol ... ;

passwd-auth ... ;

ciphers ... ;

kex-algorithms ... ;

macs ... ;

* option ... ;

* subsystem ... ;

}

The `ssh-server` section is derived from `ssh-server` section prototype. For detail description of it, see ssh(5).

`ssh-keys` {

* key2 ... ;

}

SSH keys definition.

Items & subsections:

`key2` *email type key* [*ignored*];

SSH Version 2 key.

email **(type: `str`)**

Owner email address.

type **(type: `ssh-key-type`)**

key **(type: `str`)**

ignored **(type: `str`, optional, default: <NULL>)**

Elem ignored, retained due to backward compatibility.

[End of section `system.ssh-keys` description.]

`ica-auto` *port priv-key pub-key*;

Configure the icamd/icasd automatically. Uses the addresses defined for pikemon

port **(type: `port`)**

The port that icamd listens on and icasd connects to.

priv-key (type: **name** of **shared-file**, see common(5))

The private ssh key used for authentication

pub-key (type: **name** of **shared-file**, see common(5))

The public ssh key used for authentication

icamd {

phase ... ;

* tag ... ;

listen-on { ... }

priv-key ... ;

* slave *name* { ... }

}

The icamd section is derived from icamd section prototype. For detail description of it, see ica(5).

icasd {

phase ... ;

* tag ... ;

priv-key ... ;

* master *name* { ... }

}

The icasd section is derived from icasd section prototype. For detail description of it, see ica(5).

watch {

disable ... ;

}

Watching system parameters by RRD.

Items & subsections:

disable;

Disable watching.

[End of section `system.watch` description.]

acl *name* {

* from ... ;

* to ... ;

* time ... ;

time-period-set { ... }

deny ... ;

accept ... ;

* doctype-ident-order ... ;

rule ... ;
auth ... ;
idle-timeout ... ;
source-address ... ;
plug-to ... ;
service ... ;
}

General ACL definition.

The `acl` section is derived from `acl-1` section prototype. For detail description of it, see acl(5).

Changes to the `acl` section:

Item `user` is not valid.

Item `idle-timeout-peer` is not valid.

Item SERVICE must be specified.

Added items & subsections:

`service` *list*;

List of proxies where this ACL is applicable.

list **(type: `str-set`)**

[End of section `system.acl` description.]

`use-services` *file*;

Source for /etc/services file.

file **(type: `name` of `shared-file`, see common(5))**

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

`resolver` *name* {

* server ... ;
search ... ;
preference ... ;
edns ... ;
conf-timeout ... ;
initial-timeout ... ;
final-timeout ... ;
conn-timeout ... ;
disable-deresolution ... ;
}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`nameserver` *name* {

phase ... ;
* tag ... ;
use-ipv4-only ... ;
listen-on { ... }
forward ... ;
* forwarder ... ;
* from ... ;
dnssec { ... }
send-cookie ... ;
* option ... ;
* raw ... ;
* zone *name* { ... }
}

The `nameserver` section is derived from `nameserver` section prototype. For detail description of it, see nameserver(5).

`ns-list` *name* {

* server ... ;
}

The `ns-list` section is derived from `ns-list` section prototype. For detail description of it, see resolver(5).

`atrmon` *name* {

phase ... ;
* tag ... ;
log-debug { ... }
log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
monitoring { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
run-block-sigalrm ... ;
listen-on { ... }
client-conn { ... }
* session-acl *name* { ... }

* request-acl *name* { ... }
}

The `atrmon` section is derived from `atrmon` section prototype. For detail description of it, see atr(5).

`pf-queue` *name* {
parent ... ;
bandwidth ... ;
priority ... ;
qlimit ... ;
cbq { ... }
priq { ... }
hfsc { ... }
}

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

`packet-filter` {
phase ... ;
* tag ... ;
log-debug { ... }
log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
pflog ... ;
pfsync ... ;
comm-dir ... ;
ignore-iface ... ;
pcap-timeout ... ;
buffer-size ... ;
* set-option ... ;

timeouts { ... }
limits { ... }
logging-frequence ... ;
* altq *name* { ... }
* scrub-acl *name* { ... }
* rdr-acl *name* { ... }
* nat-acl *name* { ... }
* binat-acl *name* { ... }
* filter-acl *name* { ... }
* load-anchor ... ;
}

The `packet-filter` section is derived from `packet-filter` section prototype. For detail description of it, see packet-filter(5).

`adaptive-firewall {`

ids-agent { ... }
* watchdog *name* { ... }
honeypot { ... }
auto-blocking { ... }
adaptive-database { ... }
address-groups { ... }
port-groups { ... }
whitelist ... ;
blacklist ... ;
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
}

The `adaptive-firewall` section is derived from `adaptive-firewall` section prototype. For detail description of it, see adaptive-firewall(5).

`alertd {`

phase ... ;
* tag ... ;
log-debug { ... }
log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
monitoring { ... }

```
    stats-daily { ... }
    stats-weekly { ... }
    stats-monthly { ... }
    nodaemon ... ;
    singleproc ... ;
    app-user ... ;
    idle-timeout ... ;
    run-block-sigalrm ... ;
  * snmp-manager name { ... }
  }
```

The `alertd` section is derived from `alertd` section prototype. For detail description of it, see alertd(5).

```
bird4 {
    phase ... ;
  * tag ... ;
    use-id ... ;
    direct { ... }
    kernel { ... }
    device { ... }
    static { ... }
    ospf { ... }
  * raw ... ;
  }
```

The `bird4` section is derived from `bird4` section prototype. For detail description of it, see router(5).

```
bird6 {
    phase ... ;
  * tag ... ;
    use-id ... ;
    direct { ... }
    kernel { ... }
    device { ... }
    static { ... }
    ospf { ... }
  * raw ... ;
  }
```

The `bird6` section is derived from `bird6` section prototype. For detail description of it, see router(5).

`rtadvd` {

phase ... ;

* tag ... ;

default-params { ... }

}

The `rtadvd` section is derived from `rtadvd` section prototype. For detail description of it, see rtadvd(5).

`ssl-params` *name* {

versions ... ;

ciphers ... ;

tcp-eof ... ;

id ... ;

* auth-cert ... ;

distrusted-certs ... ;

dont-check-crl ... ;

* crl ... ;

verify-peer ... ;

cache-timeout ... ;

use-ticket ... ;

enable-renegotiation ... ;

fake-cert ... ;

prefer server ciphers ... ;

enable-ecdh ... ;

}

The `ssl-params` section is derived from `ssl-params` section prototype. For detail description of it, see ssl(5).

`fake-cert` *name* {

key ... ;

auth-ca ... ;

fail-ca ... ;

* extension ... ;

purge ... ;

}

The `fake-cert` section is derived from `fake-cert` section prototype. For detail description of it, see ssl(5).

html-filter *name* **{**

* script-tag-language ... ;
  replace-head-script-tags ... ;
  replace-body-script-tags ... ;
* style-tag-type ... ;
  replace-style-tags ... ;
* iframe-tag-src ... ;
  replace-iframe-tags ... ;
* intrinsic-language ... ;
* intrinsic-hack ... ;
  replace-intrinsic ... ;
* macro-language ... ;
* macro-hack ... ;
  replace-macros ... ;
* uri ... ;
  replace-uri ... ;
* embed-tag-type ... ;
* embed-src-hack ... ;
* embed-plugin-hack ... ;
  replace-head-embed-tags ... ;
  replace-body-embed-tags ... ;
* applet ... ;
  replace-applets ... ;
* object ... ;
* object-classid-hack ... ;
* object-data-hack ... ;
  replace-head-object-tags ... ;
  replace-body-object-tags ... ;
* param-tags ... ;
  replace-param ... ;
  script-end-hack ... ;

}

The `html-filter` section is derived from `html-filter` section prototype. For detail description of it, see mod-html-filter(5).

mail-filter *name* **{**

  stamp-limit ... ;
  stamp-filter ... ;
* unflagged-8bit ... ;

```
    * bad-end-of-line ... ;
    * invalid-header ... ;
    * long-header-lines ... ;
    * invalid-chars ... ;
    * header-8bit-chars ... ;
    * bad-boundary-chars ... ;
    * bad-boundary-length ... ;
    * long-body-lines ... ;
    * long-encoded-lines ... ;
      enc-line-len ... ;
    * bad-mime-struct ... ;
    * invalid-encoding ... ;
      treat-rfc822-as-text ... ;
    }
```

The `mail-filter` section is derived from `mail-filter` section prototype. For detail description of it, see mod-mail-doc(5).

```
aproxy name {
    auth ... ;
    insecure-cookies ... ;
    oob-auth ... ;
    cookie-name ... ;
    logout ... ;
    timeout-idle ... ;
    timeout-unauth ... ;
    bufsz ... ;
    }
```

The `aproxy` section is derived from `aproxy` section prototype. For detail description of it, see http-proxy(5).

```
radius-client name {
    nas ... ;
    groups ... ;
    * server ... ;
    }
```

The `radius-client` section is derived from `radius-client` section prototype. For detail description of it, see radius(5).

```
ldap-client-auth name {
    server ... ;
```

```
    ssl { ... }
    bindinfo ... ;
    kerberos ... ;
    users ... ;
    groups ... ;
    active-directory ... ;
  }
```

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

```
oob-auth name {
    method ... ;
    max-sessions ... ;
    max-user ... ;
    max-groups ... ;
    truncate-groups ... ;
    file ... ;
    lock ... ;
  }
```

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

```
antivirus name {
    connection ... ;
    sock-opt { ... }
    timeout ... ;
    comm-dir ... ;
    altq ... ;
    max-checked-size ... ;
    icap-pass-200-with-pure-body ... ;
    persistent-stream ... ;
    clamav-agent { ... }
  }
```

The `antivirus` section is derived from `antivirus` section prototype. For detail description of it, see antivirus(5).

```
antispam name {
    connection ... ;
    sock-opt { ... }
    altq ... ;
  }
```

The **antispam** section is derived from **antispam** section prototype. For detail description of it, see mod-antispam(5).

smtp-forwarder *name* {
  * server ... ;
    agent { ... }
    timeouts { ... }
    hostname ... ;
    size ... ;
    source-address ... ;
  * domain ... ;
    server-ssl ... ;
  * server-cert-match ... ;
    altq ... ;
  }

The **smtp-forwarder** section is derived from **smtp-forwarder** section prototype. For detail description of it, see smtp-proxy(5).

web-filter *name* {
    connection ... ;
    fail-ok ... ;
    sock-opt { ... }
  }

The **web-filter** section is derived from **web-filter** section prototype. For detail description of it, see http-proxy(5).

clear-web-db {
    internal-servers ... ;
    db ... ;
    lock ... ;
    local-db { ... }
  }

The **clear-web-db** section is derived from **clear-web-db** section prototype. For detail description of it, see clear-web-db(5).

openvpn *name* {
    stats-daily { ... }
    stats-weekly { ... }
    stats-monthly { ... }
    interface ... ;
    topology ... ;

local ... ;
nobind ... ;
user ... ;
group ... ;
persist-tun ... ;
persist-key ... ;
log-debug { ... }
log-stats { ... }
mute ... ;
ping-timer-rem ... ;
keepalive ... ;
proto ... ;
tls-mat ... ;
dh ... ;
secret ... ;
crl-verify ... ;
server ... ;
max-clients ... ;
duplicate-cn ... ;
client-to-client ... ;
ccd-exclusive ... ;
mlock ... ;
float ... ;
push { ... }
ifconfig-pool ... ;
ifconfig-ipv6-pool ... ;
tls-server ... ;
tls-client ... ;
tls-auth ... ;
* remote ... ;
remote-random ... ;
comp-lzo ... ;
verify-x509-name ... ;
remote-cert-ku ... ;
remote-cert-eku ... ;
remote-cert-tls ... ;
cipher ... ;
data-ciphers ... ;

data-ciphers-fallback ... ;
client ... ;
pull ... ;
route-nopull ... ;
no-ifconfig-noexec ... ;
ifconfig-pool-persist ... ;
client-connect ... ;
client-connect-socket ... ;
* ccd *name* { ... }
* raw ... ;
phase ... ;
* tag ... ;
socket-root ... ;
fast-io ... ;
}

The `openvpn` section is derived from `openvpn` section prototype. For detail description of it, see openvpn(5).

`ipsec-global` {
phase ... ;
* tag ... ;
}

The `ipsec-global` section is derived from `ipsec-global` section prototype. For detail description of it, see ipsec(5).

`ipsec-remote` *name* {
peer ... ;
lifetime ... ;
encryption ... ;
hash ... ;
dh-group ... ;
authentication ... ;
dpd ... ;
rekey ... ;
ike-frag ... ;
esp-frag ... ;
}

The `ipsec-remote` section is derived from `ipsec-remote` section prototype. For detail description of it, see ipsec(5).

ipsec *name* **{**

phase ... ;

* tag ... ;

transport-mode ... ;

tunnel-mode { ... }

phase2 { ... }

}

The ipsec section is derived from ipsec section prototype. For detail description of it, see ipsec(5).

data-match *name* **{**

max-size ... ;

init-match ... ;

max-match ... ;

step-size ... ;

step-match ... ;

* test ... ;

}

The data-match section is derived from data-match section prototype. For detail description of it, see mod-match(5).

ntlm-auth *name* **{**

domain ... ;

workgroup ... ;

* ad-controller ... ;

interfaces { ... }

ldap ... ;

timeout ... ;

timeout-idle ... ;

timeout-unauth ... ;

}

The ntlm-auth section is derived from ntlm-auth section prototype. For detail description of it, see http-proxy(5).

kerberos-auth *name* **{**

domain ... ;

user-match ... ;

kinit ... ;

keytab ... ;

proxy-host ... ;

```
  * ad-controller ... ;
    ldap ... ;
    timeout-idle ... ;
    timeout-unauth ... ;
    lock ... ;
    lock-ldap ... ;
    one-per-session ... ;
  }
```

The `kerberos-auth` section is derived from `kerberos-auth` section prototype. For detail description of it, see http-proxy(5).

```
cwcatd {
    phase ... ;
  * tag ... ;
    log-debug { ... }
    log-stats { ... }
    use-resolver ... ;
    cfg-resolution ... ;
    monitoring { ... }
    stats-daily { ... }
    stats-weekly { ... }
    stats-monthly { ... }
    nodaemon ... ;
    singleproc ... ;
    app-user ... ;
    idle-timeout ... ;
    run-block-sigalrm ... ;
    wakeup ... ;
    retry ... ;
  }
```

Clear Web automatic categorization daemon.

The `cwcatd` section is derived from `alone-application` section prototype. For detail description of it, see application(5).
Added items & subsections:

`wakeup` [*sec*];

Period (in seconds) of waking up of the categorization daemon and checking the queue of categorization requests. In addition, the daemon is awaken by a signal immediately after a new request is enqueued.

sec **(type: `uint16`, optional, default: 60)**

retry [*sec*];

Time (in seconds) after which a failed automatic categorization will be retried.

sec **(type: uint32, optional, default: 3600)**

[End of section `system.cwcatd` description.]

snmpd {

phase ... ;

* tag ... ;

listen-on { ... }

* user ... ;

location ... ;

* group *name* { ... }

* proc ... ;

* exec ... ;

* disk ... ;

load ... ;

swap ... ;

* raw ... ;

}

The snmpd section is derived from snmpd section prototype. For detail description of it, see snmpd(5).

http-cache {

phase ... ;

* tag ... ;

listen-on { ... }

hand-off ... ;

cache-size ... ;

max-object-size ... ;

* raw ... ;

}

The http-cache section is derived from http-cache section prototype. For detail description of it, see http-cache(5).

update {

adaptive-firewall { ... }

clear-web { ... }

}

The update section is derived from update section prototype. For detail description of it, see update(5).

feedback {

adaptive-firewall { ... }

clear-web { ... }

system-status ... ;

reporter ... ;

errors ... ;

}

The feedback section is derived from feedback section prototype.
For detail description of it, see feedback(5).

stats {

keep-days ... ;

disable ... ;

}

Parameters for generating statistics.

Items & subsections:

keep-days [*val*];

How many days of log data are kept in the Kernun Reporter database. Older data will be deleted automatically. If set to zero, no data will be deleted from the database.

val **(type: uint16, optional, default: 31)**

disable;

Do not generate the REPORTER component. This item is intended mainly for testing purposes.

[End of section `system.stats` description.]

stats-daily {

top-clients ... ;

top-users ... ;

top-servers ... ;

}

The stats-daily section is derived from summary section prototype.
For detail description of it, see application(5).
Changes to the stats-daily section:

Item `top-groups` is not valid.

Item `top-categories` is not valid.

Item `top-senders` is not valid.

Item `top-recipients` is not valid.

Item `top-mime-types` is not valid.

Item `top-qnames` is not valid.

Item `top-qtypes` is not valid.
Item `top-callers` is not valid.
Item `top-receivers` is not valid.
Item `top-sids` is not valid.
Item `top-server-ports` is not valid.
Item `spam-threshold` is not valid.
Section `activity-report` is not valid.
Item `top-src-ips` is not valid.
Item `top-dst-ips` is not valid.
Item `top-rules` is not valid.

`stats-weekly` {

top-clients ... ;
top-users ... ;
top-servers ... ;
}

The `stats-weekly` section is derived from `summary` section prototype. For detail description of it, see application(5).
Changes to the `stats-weekly` section:

Item `top-groups` is not valid.
Item `top-categories` is not valid.
Item `top-senders` is not valid.
Item `top-recipients` is not valid.
Item `top-mime-types` is not valid.
Item `top-qnames` is not valid.
Item `top-qtypes` is not valid.
Item `top-callers` is not valid.
Item `top-receivers` is not valid.
Item `top-sids` is not valid.
Item `top-server-ports` is not valid.
Item `spam-threshold` is not valid.
Section `activity-report` is not valid.
Item `top-src-ips` is not valid.
Item `top-dst-ips` is not valid.
Item `top-rules` is not valid.

`stats-monthly` {

top-clients ... ;
top-users ... ;
top-servers ... ;
}

The `stats-monthly` section is derived from `summary` section prototype. For detail description of it, see application(5).

Changes to the `stats-monthly` section:

Item `top-groups` is not valid.

Item `top-categories` is not valid.

Item `top-senders` is not valid.

Item `top-recipients` is not valid.

Item `top-mime-types` is not valid.

Item `top-qnames` is not valid.

Item `top-qtypes` is not valid.

Item `top-callers` is not valid.

Item `top-receivers` is not valid.

Item `top-sids` is not valid.

Item `top-server-ports` is not valid.

Item `spam-threshold` is not valid.

Section `activity-report` is not valid.

Item `top-src-ips` is not valid.

Item `top-dst-ips` is not valid.

Item `top-rules` is not valid.

`tcp-proxy` *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

err-reset ... ;

```
    ssl-session-cache { ... }
    client-ssl ... ;
    client-ssl-timeout ... ;
    data-mime-db ... ;
    auth ... ;
  * session-acl name { ... }
  }
```

The `tcp-proxy` section is derived from `tcp-proxy` section prototype. For detail description of it, see tcp-proxy(5).

```
udp-proxy name {
    phase ... ;
  * tag ... ;
    log-debug { ... }
    log-stats { ... }
    use-resolver ... ;
    cfg-resolution ... ;
    monitoring { ... }
    stats-daily { ... }
    stats-weekly { ... }
    stats-monthly { ... }
    nodaemon ... ;
    singleproc ... ;
    app-user ... ;
    run-block-sigalrm ... ;
    listen-on { ... }
    udpserver { ... }
    source-address ... ;
    doctype-identification { ... }
    auth ... ;
  * session-acl name { ... }
  }
```

The `udp-proxy` section is derived from `udp-proxy` section prototype. For detail description of it, see udp-proxy(5).

```
dns-proxy name {
    phase ... ;
  * tag ... ;
    log-debug { ... }
```

log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
monitoring { ... }
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
doctype-identification { ... }
queue-size ... ;
edns ... ;
dnssec ... ;
cache { ... }
request-timeout ... ;
response-timeout ... ;
query-timeout ... ;
server-dead ... ;
server-retry ... ;
server-proto ... ;
requests-table-size ... ;
sockets-table-size ... ;
internal-request-depth ... ;
adr-reply-limit ... ;
ptr-reply-limit ... ;
client-conn { ... }
server-conn { ... }
* session-acl *name* { ... }
* request-acl *name* { ... }
}

The `dns-proxy` section is derived from `dns-proxy` section prototype. For detail description of it, see dns-proxy(5).

`ftp-proxy` *name* {

phase ... ;

```
        * tag ... ;
          log-debug { ... }
          log-stats { ... }
          use-resolver ... ;
          cfg-resolution ... ;
          monitoring { ... }
          stats-daily { ... }
          stats-weekly { ... }
          stats-monthly { ... }
          nodaemon ... ;
          singleproc ... ;
          app-user ... ;
          idle-timeout ... ;
          run-block-sigalrm ... ;
          listen-on { ... }
          tcpserver { ... }
          source-address ... ;
          doctype-identification { ... }
          client-ctrl { ... }
          server-ctrl { ... }
          client-data { ... }
          server-data { ... }
          init-timeout ... ;
          init-cmdlimit ... ;
        * data-transfer ... ;
          retry-data ... ;
        * session-acl name { ... }
        * command-acl name { ... }
        * doc-acl name { ... }
        }
```

The `ftp-proxy` section is derived from `ftp-proxy` section prototype. For detail description of it, see ftp-proxy(5).

```
gk-proxy name {
          phase ... ;
        * tag ... ;
          log-debug { ... }
          log-stats { ... }
          use-resolver ... ;
```

cfg-resolution ... ;
monitoring { ... }
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
udpserver { ... }
source-address ... ;
doctype-identification { ... }
map-file ... ;
* session-acl *name* { ... }
}

The `gk-proxy` section is derived from `gk-proxy` section prototype. For detail description of it, see gk-proxy(5).

`h323-proxy` *name* {
phase ... ;
* tag ... ;
log-debug { ... }
log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
monitoring { ... }
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
tcpserver { ... }
doctype-identification { ... }

client-ctrl { ... }
server-ctrl { ... }
data-channel { ... }
map-file ... ;
* session-acl *name* { ... }
max-channel-ports ... ;
}

The `h323-proxy` section is derived from `h323-proxy` section prototype. For detail description of it, see h323-proxy(5).

`http-proxy` *name* {
phase ... ;
* tag ... ;
log-debug { ... }
log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
monitoring { ... }
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
tcpserver { ... }
source-address ... ;
doctype-identification { ... }
client-conn { ... }
server-conn { ... }
document-root ... ;
hdr-line-len ... ;
blacklist-db ... ;
connect-data-mime-db ... ;
ftp-proxy ... ;
max-aproxy-sessions ... ;
max-bypass-sessions ... ;

oob-auth-srv ... ;
ssl-session-cache { ... }
aproxy-lock ... ;
cookie-table { ... }
extended-status ... ;
* session-acl *name* { ... }
* request-acl *name* { ... }
* doc-acl *name* { ... }
}

The `http-proxy` section is derived from `http-proxy` section prototype. For detail description of it, see http-proxy(5).

`icap-server` *name* {
phase ... ;
* tag ... ;
log-debug { ... }
log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
monitoring { ... }
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
tcpserver { ... }
doctype-identification { ... }
client-conn { ... }
document-root ... ;
hdr-line-len ... ;
preview ... ;
blacklist-db ... ;
max-bypass-sessions ... ;
ssl-session-cache { ... }
ldap-cache { ... }

* session-acl *name* { ... }
* service-acl *name* { ... }
* request-acl *name* { ... }
* doc-acl *name* { ... }
}

The `icap-server` section is derived from `icap-server` section prototype. For detail description of it, see icap-server(5).

`imap4-proxy` *name* {
phase ... ;
* tag ... ;
log-debug { ... }
log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
monitoring { ... }
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
tcpserver { ... }
source-address ... ;
doctype-identification { ... }
client-conn { ... }
server-conn { ... }
ssl-session-cache { ... }
mail-pool ... ;
* session-acl *name* { ... }
* command-acl *name* { ... }
* mail-acl *name* { ... }
* doc-acl *name* { ... }
}

The `imap4-proxy` section is derived from `imap4-proxy` section prototype. For detail description of it, see imap4-proxy(5).

pop3-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

ssl-session-cache { ... }

mail-pool ... ;

* session-acl *name* { ... }

* command-acl *name* { ... }

* mail-acl *name* { ... }

* doc-acl *name* { ... }

}

The pop3-proxy section is derived from pop3-proxy section prototype. For detail description of it, see pop3-proxy(5).

sip-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
source-address ... ;
doctype-identification { ... }
queue-size ... ;
hash-salt ... ;
ctrl-conn { ... }
data-conn { ... }
map-file ... ;
timeouts { ... }
sessions-table-size ... ;
sockets-table-size ... ;
* keepalive ... ;
* session-acl *name* { ... }
* request-acl *name* { ... }
}

The `sip-proxy` section is derived from `sip-proxy` section prototype. For detail description of it, see sip-proxy(5).

`smtp-proxy` *name* {
phase ... ;
* tag ... ;
log-debug { ... }
log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
monitoring { ... }
stats-daily { ... }
stats-weekly { ... }
stats-monthly { ... }
nodaemon ... ;

singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
tcpserver { ... }
doctype-identification { ... }
client-conn { ... }
server-conn { ... }
mail-pool ... ;
quarantine ... ;
postmaster ... ;
hostname ... ;
init-timeout ... ;
bad-commands ... ;
bad-recipients ... ;
dsn-mail-copy ... ;
use-antivirus ... ;
use-antispam ... ;
ssl-session-cache { ... }
grey-listing { ... }
* session-acl *name* { ... }
* delivery-acl *name* { ... }
* mail-acl *name* { ... }
* doc-acl *name* { ... }
}

The `smtp-proxy` section is derived from `smtp-proxy` section prototype. For detail description of it, see smtp-proxy(5).

`sqlnet-proxy` *name* {
phase ... ;
* tag ... ;
log-debug { ... }
log-stats { ... }
use-resolver ... ;
cfg-resolution ... ;
monitoring { ... }
stats-daily { ... }
stats-weekly { ... }

stats-monthly { ... }
nodaemon ... ;
singleproc ... ;
app-user ... ;
idle-timeout ... ;
run-block-sigalrm ... ;
listen-on { ... }
tcpserver { ... }
doctype-identification { ... }
client-conn { ... }
server-conn { ... }
init-timeout ... ;
protocol-version ... ;
max-service-name-len ... ;
check-reserved-bits ... ;
connect-string-charset ... ;
connect-packet-sizelimit ... ;
* session-acl *name* { ... }
* service-acl *name* { ... }
}

The `sqlnet-proxy` section is derived from `sqlnet-proxy` section prototype. For detail description of it, see sqlnet-proxy(5).

`proxy-ng` *name* {
phase ... ;
* tag ... ;
use-resolver ... ;
nodaemon ... ;
app-user ... ;
log-debug { ... }
log-stats { ... }
resolver-ng { ... }
listen-on { ... }
tcpserver { ... }
* cfg-begin ... ;
* cfg-end ... ;
* jval ... ;
log-audit { ... }
* session-acl *name* { ... }

http-proxy { ... }
}

The `proxy-ng` section is derived from `proxy-ng` section prototype. For detail description of it, see proxy-ng(5).

`proxy-ng-transp-ports` *ports*;

ports **(type: `uint16-list`)**

TCP ports to be used for transparent listening sockets of the PROXY-NG. Defaults to ports {2, 3, 4, 5, 6, 7}.

[End of section `system` description.]

SEE ALSO

configuration(7), acl(5), adaptive-firewall(5), alertd(5), antivirus(5), application(5), atr(5), auth(5), clear-web-db(5), common(5), dhcp-server(5), dns-proxy(5), feedback(5), ftp-proxy(5), gk-proxy(5), h323-proxy(5), http-cache(5), http-proxy(5), ica(5), icap-server(5), imap4-proxy(5), interface(5), ipc(5), ipsec(5), ldap(5), license(5), listen-on(5), log(5), mod-antispam(5), mod-html-filter(5), mod-mail-doc(5), mod-match(5), nameserver(5), newsyslog.conf(5), ntp(5), openvpn(5), packet-filter(5), periodic.conf(5), pf-queue(5), pike(5), pop3-proxy(5), proxy-ng(5), radius(5), resolver(5), router(5), rtadvd(5), sip-proxy(5), smtp-proxy(5), snmpd(5), source-address(5), sqlnet-proxy(5), ssh(5), ssl(5), sysctl(5), tcp-proxy(5), time(5), udp-proxy(5), udpserver(5), update(5), cml(8), kat(8)

NAME

tcp-proxy — format of tcp-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **tcp-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **tcp-proxy** configuration directives:

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

auth-method (see **auth**(5))

source-address-mode (see **source-address**(5))

transparency (see **acl**(5))

user-auth-spec (see **acl**(5))

doctype-ident-method (see **acl**(5))

listen-on-sock (see **listen-on**(5))

ITEMS AND SECTIONS

Configuration of **tcp-proxy** library component consists of following prototypes:
* tcp-proxy *name* { ... }

Description:

tcp-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

err-reset ... ;

ssl-session-cache { ... }

client-ssl ... ;

client-ssl-timeout ... ;

data-mime-db ... ;

auth ... ;

* session-acl *name* { ... }

}

Generic TCP proxy configuration.

The `tcp-proxy` section is derived from `proxy` section prototype. For detail description of it, see application(5).
Changes to the `tcp-proxy` section:

Section `udpserver` is not valid.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

Section `monitoring` (see monitoring(5))

Item `aproxy-user` is not valid.

Item `data` is not valid.

Item `idle-timeout` (see application(5))

Element `seconds` is optional, default: 2d.

Item `listen-on.non-transparent` (see listen-on(5))

Element `proto` is optional, default: tcp.

Item `listen-on.transparent` (see listen-on(5))

Element `proto` is optional, default: tcp.

Added items & subsections:

`client-conn` {

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection to client options.

The `client-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `client-conn` section:

Item `conn-timeout` is not valid.

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`server-conn` {

conn-timeout ... ;

recv-bufsize ... ;

close-timeout ... ;

send-bufsize ... ;

log-limit ... ;

}

Connection to server options.

The `server-conn` section is derived from `sock-opt` section prototype. For detail description of it, see netio(5).
Changes to the `server-conn` section:

Item `recv-timeout` is not valid.

Item `send-timeout` is not valid.

`err-reset`;

If set, reset connection on error (otherwise use normal TCP close).

`ssl-session-cache` {

capacity ... ;

dir ... ;

lock ... ;

}

The `ssl-session-cache` section is derived from `ssl-session-cache` section prototype. For detail description of it, see ssl(5).

`client-ssl` *params*;

Use SSL/TLS on the connection from a client.

params **(type: `name` of `ssl-params`, see ssl(5))**

`client-ssl-timeout` *seconds*;

Timeout for SSL/TLS handshake with client.

seconds **(type: `uint32`)**

`data-mime-db` *filename*;

Data MIME type mapping file.

filename **(type: `name` of `shared-file`, see common(5))**

`auth none`;

`auth passwd` *file*;

`auth radius` *client*;

`auth ldap` *ldap*;

`auth ext` *file*;

`auth oob` *oob* **[**`mode` **[`loose`]]**;

Authentication method and attributes specification.

For more details, see auth(7).

<branching element> (type: `auth-method`)

file **(type: `str`)**

Password/utility file name.

client **(type: `name` of `radius-client`, see radius(5))**

RADIUS client configuration name.

ldap (type: **name** of **ldap-client-auth**, see ldap(5))
: LDAP client configuration parameters.

oob (type: **name** of **oob-auth**, see auth(5))
: OOB authentication parameters.

mode (type: **obligation**, optional, default: allowed)

loose (type: **key**, optional)

Constraints:
: Only out-of-band authentication is supported in this proxy.

session-acl *name* **{**

```
* from ... ;
* to ... ;
* user ... ;
* time ... ;
  time-period-set { ... }
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
  idle-timeout ... ;
  idle-timeout-peer ... ;
  source-address ... ;
  plug-to ... ;
* client-cert-match ... ;
* ip-tos-from-client ... ;
  max-bytes-in ... ;
  max-bytes-out ... ;
  max-time ... ;
  cl2srv-halfclosed-time ... ;
  srv2cl-halfclosed-time ... ;
  server-ssl ... ;
  data-filter-client ... ;
  data-filter-server ... ;
* server-cert-match ... ;
  client-altq ... ;
  server-altq ... ;
  ip-tos-to-client { ... }
  ip-tos-to-server { ... }
}
```

The `session-acl` section is derived from `acl-1` section prototype.
For detail description of it, see acl(5).
Changes to the `session-acl` section:

Item `auth` is not valid.

SSL/TLS required on connection in order to match server certificates.

Added items & subsections:

`client-cert-match` [**`subject`** *subject*] [**`issuer`** *issuer*];

Select an ACL according to a client certificate.

`subject` *subject* (type: **`str-set`**, optional, default: *)

acceptable certificate subjects

`issuer` *issuer* (type: **`str-set`**, optional, default: *)

acceptable certificate issuers

`ip-tos-from-client` *val*;

Testing an IP TOS value of received packets.

val (type: **`uint8-set`**)

`max-bytes-in` *bytes*;

Maximum number of bytes from server to client.

bytes (type: **`uint64`**)

`max-bytes-out` *bytes*;

Maximum number of bytes from client to server.

bytes (type: **`uint64`**)

`max-time` *seconds*;

Maximum time of session

seconds (type: **`uint32`**)

`cl2srv-halfclosed-time` *seconds*;

Maximum duration of client to server communication after the connection is half-closed in server to client direction.

seconds (type: **`uint32`**)

`srv2cl-halfclosed-time` *seconds*;

Maximum duration of server to client communication after the connection is half-closed in client to server direction.

seconds (type: **`uint32`**)

`server-ssl` *params*;

Use SSL/TLS on the connection to a server.

params (type: **`name`** of **`ssl-params`**, see ssl(5))

`data-filter-client` *rules*;

Client data filtering.

rules (type: **`name`** of **`data-match`**, see mod-match(5))

`data-filter-server` *rules*;

Server data filtering.

rules (type: **name** of **data-match**, see mod-match(5))

server-cert-match [**subject** *subject*] [**issuer** *issuer*];

Requirements for server certificate.

subject *subject* (type: **str-set**, optional, default: *)

acceptable certificate subjects

issuer *issuer* (type: **str-set**, optional, default: *)

acceptable certificate issuers

client-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to client.

altq (type: **name** of **pf-queue**, see pf-queue(5))

queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: NULL)

priority queue name (if set, used for TCP ACK without data)

server-altq *altq* [**paltq** *paltq*];

ALTQ queues for data sent to server.

altq (type: **name** of **pf-queue**, see pf-queue(5))

queue name

paltq *paltq* (type: **name** of **pf-queue**, see pf-queue(5), optional, default: NULL)

priority queue name (if set, used for TCP ACK without data)

ip-tos-to-client {

fixed ... ;

received ... ;

other ... ;

}

The **ip-tos-to-client** section is derived from **ip-tos-to-client** section prototype. For detail description of it, see netio(5).

ip-tos-to-server {

fixed ... ;

received ... ;

other ... ;

}

The **ip-tos-to-server** section is derived from **ip-tos-to-server** section prototype. For detail description of it, see netio(5).

[End of section `tcp-proxy.session-acl` description.]

[End of section `tcp-proxy` description.]

SEE ALSO

configuration(7), acl(5), application(5), auth(5), common(5), ipc(5), ldap(5), listen-on(5), log(5), mod-match(5), monitoring(5), netio(5), pf-queue(5), radius(5), source-address(5), ssl(5), time(5), auth(7)

NAME

tcp-proxy.cfg — format of tcp-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **tcp-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **tcp-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

inline-file-format (see **common**(5))

dbglev (see **log**(5))

logfail-mode (see **log**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see **radius**(5))

ldap-tls-reqcert-mode (see **ldap**(5))

ldap-search-scope (see ldap(5))

ldap-group-match (see ldap(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

source-address-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

ssl-ver (see ssl(5))

extension-op (see ssl(5))

veri-fail-action (see ssl(5))

auth-cert-type (see ssl(5))

distrusted-cert-type (see ssl(5))

data-match-action (see mod-match(5))

ITEMS AND SECTIONS

Program **tcp-proxy** recognizes following items and sections:
* data-match *name* { ... }
* fake-cert *name* { ... }
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }

* shared-file *name* { ... }
* ssl-params *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* tcp-proxy *name* { ... }
  ipv6-mode ... ;

Description:

data-match *name* **{**

max-size ... ;

init-match ... ;

max-match ... ;

step-size ... ;

step-match ... ;

* test ... ;

}

The data-match section is derived from data-match section prototype. For detail description of it, see mod-match(5).

fake-cert *name* **{**

key ... ;

auth-ca ... ;

fail-ca ... ;

* extension ... ;

purge ... ;

}

The fake-cert section is derived from fake-cert section prototype. For detail description of it, see ssl(5).

interface *name* **{**

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

`ldap-client-auth` *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

`oob-auth` *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

pf-queue *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The pf-queue section is derived from pf-queue section prototype. For detail description of it, see pf-queue(5).

radius-client *name* {

nas ... ;

groups ... ;

* server ... ;

}

The radius-client section is derived from radius-client section prototype. For detail description of it, see radius(5).

resolver *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The resolver section is derived from resolver section prototype. For detail description of it, see resolver(5).

shared-dir *name* {

path ... ;

}

The shared-dir section is derived from shared-dir section prototype. For detail description of it, see common(5).

shared-file *name* {

path ... ;

format ... ;

}

The shared-file section is derived from shared-file section prototype. For detail description of it, see common(5).

ssl-params *name* {

versions ... ;

ciphers ... ;

tcp-eof ... ;

id ... ;

* auth-cert ... ;

distrusted-certs ... ;

dont-check-crl ... ;

* crl ... ;

verify-peer ... ;

cache-timeout ... ;

use-ticket ... ;

enable-renegotiation ... ;

fake-cert ... ;

prefer server ciphers ... ;

enable-ecdh ... ;

}

The ssl-params section is derived from ssl-params section prototype. For detail description of it, see ssl(5).

sysctl {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

use-resolver *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name **(type: `name` of `resolver`, see resolver(5))**

tcp-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

idle-timeout ... ;

run-block-sigalrm ... ;

listen-on { ... }

tcpserver { ... }

source-address ... ;

doctype-identification { ... }

client-conn { ... }

server-conn { ... }

err-reset ... ;

ssl-session-cache { ... }

client-ssl ... ;

client-ssl-timeout ... ;

data-mime-db ... ;

auth ... ;

* session-acl *name* { ... }

}

The `tcp-proxy` section is derived from `tcp-proxy` section prototype. For detail description of it, see tcp-proxy(5).

`ipv6-mode` [*status*];

Enabling/Disabling IPv6 Mode.

status **(type: `enabling`, optional, default: enable)**

SEE ALSO

configuration(7), tcp-proxy(8), acl(5), auth(5), common(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), mod-match(5), pf-queue(5), radius(5), resolver(5), source-address(5), ssl(5), sysctl(5), tcp-proxy(5), time(5), host-matching(7)

NAME

tcpserver — format of tcpserver component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **tcpserver** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **tcpserver** configuration directives:

yes-no (see **common**(5))

lock-type (see **ipc**(5))

ITEMS AND SECTIONS

Configuration of **tcpserver** library component consists of following prototypes:

tcpserver { ... }

Description:

tcpserver {

queue-size ... ;

init-children ... ;

max-children ... ;

max-children-per-ip ... ;

min-idle ... ;

max-idle ... ;

parent-cycle ... ;

info-cycle ... ;

min-start-rate ... ;

max-start-rate ... ;

kill-rate ... ;

fork-wait ... ;

fork-retries ... ;

lock ... ;

alt-lock ... ;

listener ... ;

conn-rate ... ;

conn-rate-per-ip ... ;

conn-rate-table ... ;

terminate-wait ... ;

}

General TCP server parameters.

Constraints:

INIT-CHILDREN must be within 0 < INIT-CHILDREN <= MAX-CHILDREN.

Idle-values must be within 0 < MIN-IDLE < MAX-IDLE <= MAX-CHILDREN.

MIN-START-RATE must be within 0 < MIN-START-RATE <= MAX-START-RATE.

Connection rate limitation is allowed only in LISTENER mode.

CONN-RATE-PER-IP must not be greater than CONN-RATE.

MAX-CHILDREN-PER-IP is allowed only in LISTENER mode.

MAX-CHILDREN-PER-IP must not be greater than MAX-CHILDREN.

Items & subsections:

`queue-size` [*value*];

Queue length for listen(2) syscall.

value **(type: `uint16`, optional, default: 2000)**

`init-children` [*value*];

Initially started number of child processes.

value **(type: `uint16`, optional, default: 5)**

`max-children` [*value*];

Maximum number of running child processes.

value **(type: `uint16`, optional, default: 400)**

`max-children-per-ip` [*value*];

Maximum number of running child processes per client.

When this limit is reached, no more connections from the client are accepted.

Setting to zero switches the check off.

value **(type: `uint16`, optional, default: 150)**

min-idle [*value*];

Minimum number of idle child processes.

value **(type: uint16, optional, default: 5)**

max-idle [*value*];

Maximum number of idle child processes.

value **(type: uint16, optional, default: 10)**

parent-cycle [*value*];

Interval after which parent checks child processes.

value **(type: uint16, optional, default: 1000)**
(milliseconds)

info-cycle [*value*];

Number of parent cycles after which process statistics are reported.

value **(type: uint16, optional, default: 3600)**
(0 = do not report statistics)

min-start-rate [*value*];

Minimum number of child processes forked per parent-cycle.

value **(type: uint16, optional, default: 8)**

max-start-rate [*value*];

Maximum number of child processes forked per parent-cycle.

value **(type: uint16, optional, default: 64)**

kill-rate [*value*];

Number of child processes killed per parent-cycle.

value **(type: uint16, optional, default: 1)**

Constraints:
KILL-RATE must be positive.

fork-wait [*value*];

Pause after unsuccessful fork(2) before next call.

value **(type: uint16, optional, default: 10000)**
(milliseconds)

fork-retries [*value*];

Maximum number of retries after unsuccessfull fork(2).

value **(type: uint8, optional, default: 1)**

lock [*path*];

Lock file for exclusive access to select/accept loop.

path **(type: `str`, optional, default: <NULL>)**

If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

`alt-lock none`;

`alt-lock semaphore`;

`alt-lock lock2` [*path*];

`alt-lock` [**`multilock2`**] [*path*];

An alternative implemetation of locks.

<branching element> (type: `lock-type`, optional, default: multi-lock2)

path **(type: `str`, optional, default: <NULL>)**

If set to directory, file in that directory is created with name PREFIX.PID.XXXXXX, where PREFIX is a string defined by the proxy, PID is the proxy parent process ID and X is a random suffix. If not set, directory /tmp is assumed. Automatic generation of lock file name is strongly recommended, because each lock must have a unique name.

`listener no`;

`listener` [**`yes`**];

Use a listener process for accepting clients. If enabled then sysctl kern.ipc.soacceptqueue value should be at least MAX-CHILDREN.

<branching element> (type: `yes-no`, optional, default: yes)

`conn-rate` *value*;

Maximum number of connections during one second.

When this limit is reached, no more new connections are accepted within the current second.

If omitted, the value is set to roundup(MAX-CHILDREN / 6), setting to zero switches the check off.

value **(type: `uint16`)**

`conn-rate-per-ip` *value*;

Maximum number of connections from one address during one second.

When this limit is reached, no more new connections from the client is accepted within the current nad next second.

If omitted, the value is set to roundup(MAX-CHILDREN-PER-IP / 3), setting to zero switches the check off.

value **(type: `uint16`)**

conn-rate-table [size *size***] [search** *search***];**

Parameters for per-ip incoming connection rate statistics.

size *size* **(type: uint32, optional, default: 65536)**
table size (in addresses)

search *search* **(type: uint32, optional, default: 20)**
maximum table search steps

terminate-wait *value*;

Each child process waits up to this time when terminating at proxy stop, restart, or reload. It limits the number of processes that are terminating at the same time.

value **(type: uint16)**
(milliseconds)

[End of section `tcpserver` description.]

SEE ALSO

configuration(7), fork(2), listen(2), common(5), ipc(5)

NAME

test-expr — format of **test-expr** command-line arguments

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **test-expr** command-line argument.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

ITEMS AND SECTIONS

Command-line argument `test-expr` can contain following items and sections:

test ... ;
show-acl ... ;
show-proxy ... ;
show-hash ... ;
help ... ;

Description:

test [**from** *from*] [**transparent**] [**to** *to*] [**server** *server*] [**user** *user*] [**group** *group*] [**time** *time*];

Test ACL search.

This item uses standard Kernun configuration style syntax and specifies values that will be used for entry conditions matching when searching for proper ACLs. User must enter all data needed for ACLs that will be checked during the search. For instance, if (and only if) you use the TO item in SESSION-ACLs, you must enter it here.

Example:

test-xxx -f xxx.cfg -t "test from [1.1.1.1] to [2.2.2.2] : 53;"

from *from* **(type: host, optional, default: [0.0.0.0])**

Connection/request client address.

transparent (type: key, optional)

Transparent/non-transparent flag.

to *to* **(type: sock, optional, default: [0.0.0.0]:0)**

Connection/request destination address.

server *server* **(type: sock, optional, default: [0.0.0.0]:0)**

Logical destination server address/name.

user *user* **(type: str, optional, default: <NULL>)**

Proxy user name.

group *group* **(type: str-list, optional, default: {})**

List of groups.

time *time* **(type: uint32, optional, default: 0)**

Time (form: [[mm]dd]hhmm).

Constraints:

GROUP can be used only with USER.

show-acl [*phase***];**

Show ACLs from configuration.

phase **(type: uint8, optional, default: 1)**

Phase of ACL required.

Constraints:

Phase must be at least 1.

show-proxy [all];

Show proxy parameters from configuration.

all (type: key, optional)

Show also default values.

show-hash;

Show proxy configuration hash.

help;

Show test-cfg man page

SEE ALSO

configuration(7)

NAME

time — format of time component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **time** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **time** configuration directives:

week-day (name-usage optional)

Weekday names.

sun (0)

mon (1)

tue (2)

wed (3)

thu (4)

fri (5)

sat (6)

month (name-usage optional)

Month names.

jan (1)

feb (2)

mar (3)

apr (4)

may (5)

jun (6)

jul (7)

aug (8)

sep (9)

oct (10)

nov (11)

dec (12)

ITEMS AND SECTIONS

Configuration of **time** library component consists of following prototypes:
* time ... ;
time-period-set { ... }

Description:

time [**day** *day*] [**month** *month*] [*wday* [*hhmm*]];

Time Specification.

Special item used for time-based limitation in the configuration.

Different categories are checked in conjunction (AND).

day *day* (type: **uint8-set**, optional, default: *)
day of month (1 - 31)

month *month* (type: **month-set**, optional, default: *)
month (Jan - Dec or 1 - 12)

wday (type: **week-day-set**, optional, default: *)
week-day (Sun - Sat or 0 - 6)

hhmm (type: **time-set**, optional, default: *)
time (in form hhmm)

time-period-set {

exclude ... ;

* time-spec *name* { ... }

}

Set of Time Periods Sepcification.

Special section used for time-based limitation in the configuration.

Items & subsections:

exclude;

Inverted time specification flag.

If used, the time defined by this section is complementary to the period set listed.

time-spec *name* {

* dates ... ;

* weekdays ... ;

* hours ... ;

}

Particular time range specification.

Items & subsections:

dates *from-day from-mon till-day till-mon*;

Specification of date within a year.

from-day (type: **uint8**)

from-mon (type: **month**)

till-day (type: **uint8**)

till-mon (type: **month**)

weekdays *from till*;

Specification of day within a week.

from (type: **week-day**)

till (type: **week-day**)

hours *from till*;

Specification of time within a day.

from (type: **time**)

till (type: **time**)

Upper bound of time is not included.

Constraints:

Upper bound must nost be 0000 (use 2400).

[End of section `time-period-set.time-spec` description.]

[End of section `time-period-set` description.]

SEE ALSO

configuration(7), time-matching(7)

NAME

udp-proxy — format of udp-proxy component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **udp-proxy** component configuration.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **udp-proxy** configuration directives:

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

week-day (see **time**(5))

month (see **time**(5))

auth-method (see **auth**(5))

source-address-mode (see **source-address**(5))

source-port-mode (see **source-address**(5))

transparency (see **acl**(5))

user-auth-spec (see **acl**(5))

doctype-ident-method (see **acl**(5))

dbglev (see **log**(5))

logfail-mode (see log(5))

udp-session-type (see udpserver(5))

listen-on-sock (see listen-on(5))

ITEMS AND SECTIONS

Configuration of **udp-proxy** library component consists of following prototypes:
* udp-proxy *name* { ... }

Description:

udp-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

run-block-sigalrm ... ;

listen-on { ... }

udpserver { ... }

source-address ... ;

doctype-identification { ... }

auth ... ;

* session-acl *name* { ... }

}

Generic UDP proxy configuration.

The udp-proxy section is derived from proxy section prototype. For detail description of it, see application(5).

Changes to the `udp-proxy` section:

Item `idle-timeout` is not valid.

Section `tcpserver` is not valid.

Section UDPSERVER required.

At least one SESSION-ACL must be specified (proxy must be named in some SYSTEM.ACL.SERVICES).

Section `monitoring` (see monitoring(5))

Item `aproxy-user` is not valid.

Item `data` is not valid.

Item `listen-on.non-transparent` (see listen-on(5))

Element `proto` is optional, default: udp.

Item `listen-on.transparent` (see listen-on(5))

Element `proto` is optional, default: udp.

Added items & subsections:

`auth none`;

`auth passwd` *file*;

`auth radius` *client*;

`auth ldap` *ldap*;

`auth ext` *file*;

`auth oob` *oob* [*mode* **`[loose]`**];

Authentication method and attributes specification.

For more details, see auth(7).

<branching element> (type: **`auth-method`**)

file (type: **`str`**)

Password/utility file name.

client (type: **`name`** of **`radius-client`**, see radius(5))

RADIUS client configuration name.

ldap (type: **`name`** of **`ldap-client-auth`**, see ldap(5))

LDAP client configuration parameters.

oob (type: **`name`** of **`oob-auth`**, see auth(5))

OOB authentication parameters.

mode (type: **`obligation`**, optional, default: allowed)

`loose` (type: **`key`**, optional)

Constraints:

Only out-of-band authentication is supported in this proxy.

`session-acl` *name* {

* from ... ;

* to ... ;

* user ... ;
* time ... ;
  time-period-set { ... }
  deny ... ;
  accept ... ;
* doctype-ident-order ... ;
  rule ... ;
  idle-timeout ... ;
  idle-timeout-peer ... ;
  source-address ... ;
  plug-to ... ;
  source-port ... ;
  max-dgrams-in ... ;
  max-dgrams-out ... ;
  max-dgram-sz-in ... ;
  max-dgram-sz-out ... ;
  max-bytes-in ... ;
  max-bytes-out ... ;
  session-timeout ... ;
  session ... ;
  client-altq ... ;
  server-altq ... ;
}

The `session-acl` section is derived from `acl-1` section prototype.
For detail description of it, see acl(5).
Changes to the `session-acl` section:

Item `auth` is not valid.

SOURCE-PORT can be used with SOURCE-ADDRESS CLIENT only.

Item `idle-timeout` (see acl(5))

Element `seconds` is optional, default: 60.

Added items & subsections:

`source-port client;`

`source-port [force]` *port*;

Source port for outgoing connections to server.

Can be used only with SOURCE-ADDRESS CLIENT.

If omitted, generic port will be used.

<branching element> (type: `source-port-mode`, optional, default: force)

port **(type: `port`)**

Use specified port.

max-dgrams-in *number*;
Maximum number of datagrams from server to client (0 = unlimited).
number (type: **uint64**)

max-dgrams-out *number*;
Maximum number of datagrams from client to server (0 = unlimited).
number (type: **uint64**)

max-dgram-sz-in [*bytes*];
Maximum size of a datagram from server to client.
bytes (type: **uint16**, optional, default: **65535**)

max-dgram-sz-out [*bytes*];
Maximum size of a datagram from client to server.
bytes (type: **uint16**, optional, default: **65535**)

max-bytes-in *bytes*;
Maximum number of bytes from server to client.
bytes (type: **uint64**)

max-bytes-out *bytes*;
Maximum number of bytes from client to server.
bytes (type: **uint64**)

session-timeout [*seconds*];
Maximum duration of session.
seconds (type: **uint31**, optional, default: **0**)
Duration in seconds (0 = unlimited).

session one-way;

session [**normal**];

session any-port;

session any-sock;

session broadcast [*bits*];
Type of session establishment.
<branching element> (type: **udp-session-type**, optional, default: **normal**)
bits (type: **uint8**, optional, default: **24**)
Mask size for correct responder recognition.
Constraints:
Number of bits must be at most 32.

client-altq *altq*;
ALTQ queue for data sent to client.
altq (type: **name** of **pf-queue**, see pf-queue(5))
queue name

server-altq *altq*;
ALTQ queue for data sent to server.

altq (type: **name** of **pf-queue**, see pf-queue(5))
queue name

[End of section `udp-proxy.session-acl` description.]

[End of section `udp-proxy` description.]

SEE ALSO

configuration(7), acl(5), application(5), auth(5), common(5), ldap(5), listen-on(5), log(5), monitoring(5), pf-queue(5), radius(5), source-address(5), time(5), udpserver(5), auth(7)

NAME

udp-proxy.cfg — format of udp-proxy program configuration file

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **udp-proxy.cfg** configuration file.

Repeatable sections/items are marked by the '*' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **udp-proxy.cfg** configuration directives:

enabling (see **common**(5))

yes-no (see **common**(5))

direction (see **common**(5))

ip-version (see **common**(5))

osi4-proto (see **common**(5))

time-cond (see **common**(5))

zip-mode (see **common**(5))

obligation (see **common**(5))

inline-file-format (see **common**(5))

week-day (see **time**(5))

month (see **time**(5))

lock-type (see **ipc**(5))

radius-attr (see **radius**(5))

ldap-tls-reqcert-mode (see **ldap**(5))

ldap-search-scope (see **ldap**(5))

ldap-group-match (see **ldap**(5))

auth-method (see auth(5))

oob-authentication-method (see auth(5))

bandwidth-mode (see pf-queue(5))

pf-sc-setting (see pf-queue(5))

source-address-mode (see source-address(5))

source-port-mode (see source-address(5))

transparency (see acl(5))

user-auth-spec (see acl(5))

doctype-ident-method (see acl(5))

dbglev (see log(5))

logfail-mode (see log(5))

udp-session-type (see udpserver(5))

lagg-protocol (see interface(5))

listen-on-sock (see listen-on(5))

log-in-vain-proto (see sysctl(5))

blackhole-proto (see sysctl(5))

ITEMS AND SECTIONS

Program **udp-proxy** recognizes following items and sections:
* interface *name* { ... }
* ldap-client-auth *name* { ... }
* oob-auth *name* { ... }
* pf-queue *name* { ... }
* radius-client *name* { ... }
* resolver *name* { ... }
* shared-dir *name* { ... }
* shared-file *name* { ... }
  sysctl { ... }
  use-resolver ... ;
* udp-proxy *name* { ... }
  ipv6-mode ... ;

Description:

interface *name* {

dev ... ;

ipv4 ... ;

ipv6 ... ;

mac ... ;

aggregate ... ;

pike ... ;

vlan ... ;

tunnel ... ;

dhcp-client ... ;

ipv6-rtadv { ... }

* alias *name* { ... }

* tag ... ;

}

The `interface` section is derived from `interface` section prototype. For detail description of it, see interface(5).

ldap-client-auth *name* {

server ... ;

ssl { ... }

bindinfo ... ;

kerberos ... ;

users ... ;

groups ... ;

active-directory ... ;

}

The `ldap-client-auth` section is derived from `ldap-client-auth` section prototype. For detail description of it, see ldap(5).

oob-auth *name* {

method ... ;

max-sessions ... ;

max-user ... ;

max-groups ... ;

truncate-groups ... ;

file ... ;

lock ... ;

}

The `oob-auth` section is derived from `oob-auth` section prototype. For detail description of it, see auth(5).

`pf-queue` *name* {

parent ... ;

bandwidth ... ;

priority ... ;

qlimit ... ;

cbq { ... }

priq { ... }

hfsc { ... }

}

The `pf-queue` section is derived from `pf-queue` section prototype. For detail description of it, see pf-queue(5).

`radius-client` *name* {

nas ... ;

groups ... ;

* server ... ;

}

The `radius-client` section is derived from `radius-client` section prototype. For detail description of it, see radius(5).

`resolver` *name* {

* server ... ;

search ... ;

preference ... ;

edns ... ;

conf-timeout ... ;

initial-timeout ... ;

final-timeout ... ;

conn-timeout ... ;

disable-deresolution ... ;

}

The `resolver` section is derived from `resolver` section prototype. For detail description of it, see resolver(5).

`shared-dir` *name* {

path ... ;

}

The `shared-dir` section is derived from `shared-dir` section prototype. For detail description of it, see common(5).

`shared-file` *name* {

path ... ;

format ... ;

}

The `shared-file` section is derived from `shared-file` section prototype. For detail description of it, see common(5).

`sysctl` {

* variable ... ;

portrange-default ... ;

portrange-high ... ;

portrange-low ... ;

portrange-reserved ... ;

somaxconn ... ;

log-in-vain ... ;

blackhole ... ;

}

The `sysctl` section is derived from `sysctl` section prototype. For detail description of it, see sysctl(5).

`use-resolver` *name*;

Resolver Section Specification.

This item defines name of global (system) resolver section used in particular configuration environment. Namely, it is applicable within SYSTEM section and within any section derived from PROXY prototype. The former usage defines system-wide values, the latter one values valid for particular proxy.

name (type: **name** of **resolver**, see resolver(5))

udp-proxy *name* **{**

phase ... ;

* tag ... ;

log-debug { ... }

log-stats { ... }

use-resolver ... ;

cfg-resolution ... ;

monitoring { ... }

stats-daily { ... }

stats-weekly { ... }

stats-monthly { ... }

nodaemon ... ;

singleproc ... ;

app-user ... ;

run-block-sigalrm ... ;

listen-on { ... }

udpserver { ... }

source-address ... ;

doctype-identification { ... }

auth ... ;

* session-acl *name* { ... }

}

The udp-proxy section is derived from udp-proxy section prototype.
For detail description of it, see udp-proxy(5).

ipv6-mode [*status*];

Enabling/Disabling IPv6 Mode.

status (type: **enabling**, optional, default: enable)

SEE ALSO

configuration(7), udp-proxy(8), acl(5), auth(5), common(5), interface(5), ipc(5), ldap(5), listen-on(5), log(5), pf-queue(5), radius(5), resolver(5), source-address(5), sysctl(5), time(5), udp-proxy(5), udpserver(5), host-matching(7)

NAME

udpserver — format of udpserver component configuration

DESCRIPTION

General syntax rules of Kernun Firewall configuration files are described in configuration(7). This man page describes types, sections and items specific for the **udpserver** component configuration.

Repeatable sections/items are marked by the '⋆' before section/item name.

TYPES

Configuration directives have attributes of several value-types. For the basic types description, see configuration(7).

Enumeration is a list of words (names) representing integer values. Some enumerations accept both names and direct integer values; in this case, enumeration description contains values for every name (in parenthesis next to name). For other enumerations, using of names is obligatory.

The following enumerations are used in **udpserver** configuration directives:

udp-session-type (name-usage obligatory)

Style of session establishment.

one-way

Server never replies.

normal

Server replies from the IP address and port where the first datagram from the client has been sent to.

any-port

Server replies from the IP address where the first datagram from the client has been sent to, and from any port. The source port of the reply defines the server-side port used for all other datagrams of the same session.

any-sock

Server replies from any IP address and any port. The source IP address and port of the reply defines the server-side socket used for all other datagrams of the same session.

broadcast

Client sends broadcast messages, every recipient replies from its own address and port. This session type must not be mixed with normal unicast traffic.

ITEMS AND SECTIONS

Configuration of **udpserver** library component consists of following prototypes:

udpserver { ... }

Description:

`udpserver` {

max-sessions ... ;

}

General UDP server parameters.

Constraints:

Maximum number of sessions must be specified.

Items & subsections:

`max-sessions` *number*;

Maximum number of simultaneously active sessions.

number **(type: `uint16`)**

[End of section `udpserver` description.]

SEE ALSO

configuration(7)

Appendix C

NAME

access-control, acl — Kernun proxies access control system

DESCRIPTION

Access control in the Kernun firewall is driven by so-called *Access Control Lists (ACLs)* defined in the configuration. In terms of Kernun configuration(7), an ACL is a *repeatable section* consisting of subsections and items. A set of ACLs for every particular proxy is defined in the firewall configuration (see kernun.cml(5)) and by the cml(8) tool; it is written to the particular proxy configuration file.

Kernun ACLs form a multilayer structure, in order to fit several phases of decisions made by the particular proxy when serving client requests.

For example, **tcp-proxy** has only one layer, because it makes access decisions only once after accepting a TCP connection. On the contrary, **ftp-proxy** implements three layers of ACLs: `session-acl`, `command-acl`, and `doc-acl`. First, `session-acl` is consulted when receiving a connection from a client. Then, `command-acl` is consulted for each command received from the client. Finally, `doc-acl` is consulted for each transferred file.

At each layer, multiple ACLs can be defined. They present a ruleset for proxy decisions. In an ACL there are configuration directives of two basic functions:

entry conditions define whether or not the particular ACL matches the current situation;

actions define proxy operation continuation.

Every ACL definition in each proxy/layer is derived from the general *prototype* called `acl`. Each proxy/layer ACL changes the general model by

- defining a new name of the configuration section (e.g. `doc-acl` instead of just `acl`);
- renaming subsections/items;
- excluding subsections/items (they become unknown in the new ACL);
- adding new subsections/items specific for the proxied protocol;
- changing default values of item elements (attributes);
- changing/adding/removing integrity constraints.

The ACLs used by individual proxies are described in the proxy-specific manual pages (section 5 and section 8). This manual page desribes the general model of ACLs.

Layering model

As we have stated above, if the proxy needs to make an access control decision in several phases, it uses a multilayer ACL model. The current Kernun proxies use up to three layers of ACLs.

1. The first layer (which is present in all proxies) is called `session-acl`. It is derived from a specialized prototype `acl-1` (see acl(5) for details) and represents the crucial proxy decision: whether or not the client is to be served. This decision is made at the very beginning of the communication and therefore uses only data from the lower OSI layers — connection/request source (item `from`) and destination (item `to`) physical address etc. Specifically, this means that it uses no protocol dependent data.

   For these reasons, this layer plays a very important role and is therefore handled specially in the CML. The phase 1 ACLs (named `acl`) are defined at the `system` level (rather than within a particular proxy definition). Thus, the administrator can express the global access control policy at the single point by defining which clients may connect to which servers, which protocols they may use and at which time.

   A proxy can add proxy-specific actions to a particular `acl` by listing them in a `session-acl` section within its configuration section with the name identical to the `system`-level one.

2. The second layer of ACL (derived from the `acl-2` prototype, see acl(5) for details) comes into effect when the connection or request has been accepted, the proxy has read some data from the protocol and makes some protocol-dependent decisions. That is why:

   - different proxies use different names for the ACL (e.g. `request-acl` for HTTP or DNS, but `delivery-acl` for SMTP);
   - ACL entry conditions use protocol-dependent data (e.g. URI for HTTP, SERVICE for SQL*Net etc.); the `to` item (*physical* target) is mostly changed to the `server` item (*logical* target);
   - ACL actions can define protocol-specific responses (e.g. special reply code and text).

3. The third layer of ACL is specific only for proxies that work with *documents* (i.e. HTTP, FTP and mail handling protocols). It therefore includes document-specific entry conditions and actions (e.g. `mime-type` or `html-filter`, see `acl-3` prototype definition in acl(5) for details).

   One of the tasks that can be performed by the third-layer ACL is antivirus check. There is a fundamental difference between proxies using the *store-and-forward* model (such as SMTP) and those using the *pass-thru* model (such as HTTP). The former have the whole document available before making a decision, so they can use the antivirus result as an *entry condition*. The latter must decide on-the-flow: they define antivirus check as an *action* and, together with it, they define also ex-post reactions.

ACL Selection

If multiple ACLs of the same layer are present in the configuration, the proxy must select a single one that will control its further operation. ACL matching is performed according to the entry conditions used in the particular ACL. The individual ACL sections are evaluated in the order in which they occur in the configuration, and the first matching ACL is selected. If no ACL matches, the service is *denied*.

The following data items are defined in the general `acl` prototype and can be used to select the proper ACL (unless the proxy definition excludes them from a proxy-specific ACL):

`from` *`addrs`* This condition matches if the client address is a member of the *`addrs`* `host-set`. The algorithm used for address and host-name matching is described in host-matching(7).

`to` *`mode addrs`* `[port` *`ports`*`]` This condition tests the destination address of the connection or request. In the `non-transparent` *mode* case, the destination address is one of addresses of the firewall itself. In the `transparent` *mode* case, the destination address is the address of the target server (see also transparency(7)). It is also possible to match the target port number.

`server` *`addrs`* This condition matches if the (logical) target server of the connection/request is a member of the *`addrs`* `host-set`. The server may (but need not) be the same as the host matched by the `to` item. In HTTP, for example, the `to` item checks the IP destination address of a connection from a client, whereas the `server` item checks the server name specified in the request URL or `Host` HTTP request header.

`user none` This condition matches if the user authentication data is not present.

`user` *`users`* `[group` *`groups`*`]` This condition is true if the user has been successfully authenticated and her/his username is a member of the *`users`* `str-set`. It is also possible to match groups a user belongs to.

`time` *`time-spec`* The condition matches if the current time matches *`time-spec`*. The matching algorithm is described in the time-matching(7) manual page.

`parent-acl` *`acl-names`* This condition can be used only in a multilayer ACL set. It is true if the name of the ACL chosen in the previous layer is a member of the *`acl-names`* `str-set`.

Within one ACL section, the entry conditions of the same type are combined using logical *OR* (matches if any one is true). Entry conditions of different types are combined using logical *AND* (matches if all of them are true). Unused conditions are not checked.

For example, the ACL

```
command-acl transparent-for-joe-or-mary {
  user joe;
  from [192.168.254.10];
  user mary;
  ...
}
```

matches if the following logical formula is true:

```
(user=joe OR user=mary) AND from=192.168.254.10
```

ACL Actions

A set of actions can be specified for each ACL using configuration items or subsections within the ACL. The specified actions are performed if the ACL is selected (matched). Each proxy declares its own set of proxy-specific actions. For actions available in individual proxies, consult the proxy-specific manual pages (section 5). The following list presents the set of common actions that have the same meaning in all proxies, in which they are supported. Each ACL must contain exactly one action from the pair {`accept`|`deny`}.

`accept` This item specifies that any connection/request satisfying this ACL is accepted. Various parameters from this ACL and other parts of the configuration control further handling of the connection/request.

`deny` This item specifies that any connection/request satisfying this ACL is denied. Typically, no other actions are neither applicable nor performed, although some proxies may, in some cases, allow special actions that fine-tune the kind of negative reaction to be be used.

`auth` ***`method [param]`*** This item specifies that proxy authentication is required and the authentication method to be used. For more information, see auth(7).

`source-address` **{`client`|*`addr`*}** The specified source address is to be used for connection to the server. If `client` is specified, the client's address will be used. The *`addr`* element is of the `host` type.

`plug-to` ***`server`*** The connection or request is to be forwarded to *`server`* regardless of the IP destination address and target server specified by the user. The *`server`* element is of `sock` type.

SEE ALSO

acl(5), auth(7), configuration(7), host-matching(7), time-matching(7), transparency(7)

NAME

adaptive-firewall — Adaptive Firewall

DESCRIPTION

Adaptive Firewall is a module that can protect against both locally identified and globally shared lists of attackers. The blocking is executed by the pf(4) system component and the Adaptive Firewall controls it by changing the content of specific pf tables. The SQL databases and pf tables related to Adaptive Firewall can be managed by the **kat.af** command.

AUTONOMOUS BLOCKING

The autonomous part of the Adaptive Firewall module works with data collected from local traps. The attackers are stored into various tables of a local database called IDS (Intrusion Detection System) implemented as SQLite file `/data/var/db/af/ids.db`. It contains the following tables:

AKBL4 Addresses of clients already blocked that tried to access the firewall again.

HPOT4 Addresses caught by the Honeypot trap (see below).

Records in this table can have an `HPOT_SYN` flag, this means that this address occured only in SYN packet and so it might be faked. Such addresses are not blocked.

SRCT4 Addresses caught by Suricata.

SSHD4 Addresses found in the `/var/log/auth.log` as a client unsuccessfully trying to log in an ssh server.

Records in this table can have an `SSHD_REP` flag, this means that number of attempts reached limit configured in the adaptive-firewall(5) configuration and the address will be reported as an attacker.

This table is defined by a `watchdog` section of the configuration. Similarly, other database tables can exist if defined by other instances of this section.

Data remains in the IDS tables until is cleaned due to expired lifetime. The lifetime and the time of the day when the cleanup is executed are configured in the `auto-blocking` section.

The newly caught data from the IDS database is periodically converted to an IPS (Intrusion Protection System) database by the **af-db.sh** command. The IPS database is implemented as SQLite file `/data/var/db/af/ips.db`. Currently, it contains only one table, `IPV4`. The new content of the IPS SQL database is then loaded to the `auto-blocking` pf table by the **af-reload** command. The period of this refresh is configured in section `auto-blocking`.

Content of the IDS and IPS SQL database tables can be modified by the **af-db.sh** script but due to the nature of the refreshment process, an address removal, for instance, must be done by a special **unblock** subcommand of the **kat.af**.

ADAPTIVE DATABASE

The globally shared list is called `adaptive-database` and it can be periodically downloaded from a central server. The download parameters are configured in global section `update`. The downloaded database is first filtered according to the `adaptive-database.policy-level`, `adaptive-database.max-entries` and `whitelist` options. Then, the list of addresses is stored to pf tables `adaptive-database-any-block`, `adaptive-database-src-block` and `adaptive-database-dst-block`, according to the original database source.

Any activity from or to blocked addresses is registered and stored into the IDS SQL database tables ADAB, ADSB and ADDB that correspond to the pf table holding the particular address. This data are used only for feedback to the central server. The ADFB table is for an internal purpose only.

HONEYPOT

Honeypot is a special Adaptive Firewall function targeted against port scanning. There is a special IP address (or more), unused and unpublished. On a given port range (can be also 1-65535) on this address, there is the pf-control(8) daemon listening. The daemon accepts and closes every connection, and it adds the client to the HPOT (HPOT4) table in the IDS SQL database.

If a client tries only sending a SYN packet, it is also added to the database, but no restriction is used since the source address can be faked.

SEE ALSO

Kernun: pf-control.cfg(5), ak-db.sh(8), kat(8), pf-control(8)

NAME

antivirus — Kernun virus checking support

DESCRIPTION

As a part of the data content inspection, the Kernun proxies can send processed documents for virus checking to various antivirus engines. The engine interfaces are configured as **antivirus** global sections within a **system** section (see the antivirus(5) manual page). The method of using of the interfaces in particular proxy is defined within the proxy configuration by different approaches:

- In some proxies (e.g. SMTP), the general principles are defined globally on the proxy-level by the **use-antivirus** item, all data is handled by the same manner, and the antivirus check results are used as an entry value for the **doc-acl** search.
- In some proxies (e.g. ICAP, IMAP, POP), the general principles depend on particular service, or protocol command used, thus the **use-antivirus** item is moved into proper ACL (**service-acl**, or **command-acl**).
- In some proxies (e.g. FTP, HTTP), the antivirus check execution depends on the particular document. Thus, a proper **doc-acl** is chosen first, and within it both the antivirus engines selection (by the **antivirus** item) and the virus check result application (by the **accept-antivirus-status**) is defined.

The data can be sent to antivirus checking to more engines in parallel. After finishing all the checks, the final results is set by using the folloving rules:

1. If some virus has been found by any engine, the result `FOUND` is used.
2. Otherwise, if some engine has told that the data is clean, the result `FREE` is used.
3. Otherwise, if some engine configuration has skipped the check due to data size, the result `SKIPPED` is used.
4. Otherwise, if some engine return UNKNOWN status, the result `UNKNOWN` is used.
5. Otherwise, the result `ERROR` is used.

STANDARD OPERATION MODE

Standard behavior of the checking module is to store the whole file to the temporary file first and then to send it to the engine(s). With large files, this may cause some problems in on-line proxies (FTP, HTTP), both on the sender and recipient side.

Some of these problems can be solved by configuring the `max-checked-size` parameter and ways what to do with larger files.

- One possibility is to `skip` the files, i.e. pass them without check.

- The alternative way is to check only the initial part of the file and to decide according to it. During the check, the rest of the file is still being received and stored.

KEEPALIVE MODE OPTION

In some proxies, the check should be configured with so called `keepalive` option. It means that data is transferred in small chunks to the destination prior the check is finished. The document behaves like it would be virus FREE. If the final decision by the engine does not match with the ACL selected in advance, the session is reset.

This option is configured in proper item (`antivirus-keepalive` or `antivirus-mode`) by using nonzero `interval` and `chunk` elements.

STREAM MODE OPTION

In some proxies, the check should be configured with so called `stream` option. It means that data is sent to antivirus engines periodically as soon as a multiple of defined chunk size is reached and after a successfull check, the data is forwarded to the destination. However, only three chunks can be processed by the proxy in parallel (one being read, one being checked and one being sent out). When the output channel or the antivirus check is slow, the receipt of data is suspended until a chunk is released.

This option is configured in proper item (`antivirus-keepalive` or `antivirus-mode`) by using zero (or omitted) `interval` element and nonzero `chunk` one.

ENGINES

The current version of antivirus support following engines:

`ClamAV` ClamAV 0.9X.

`NOD32` ESET File Security v3.0.

`ICAP` Generic engine listening on a TCP/IP socket via the ICAP protocol. In the configuration, the socket address and target URI must be defined. In the URI, the scheme (ICAP), server name/address and optional port need not be included, if they can be derived from the connection.

The following ICAP engines was successfully tested:

`Symantec` Symantec Scan Engine 5.2.

`Sophos` Sophos Anti Virus Dynamic Interface (SAVDI) v2.0.

> **Warning**
>
> In the proper `service` of the `savdid.conf`, the `204` answers must be permitted:
>
> ```
> allow204: YES
> ```

McAfee Email and Web Security 5.6

ESET Gateway Security 4

> **Warning**
>
> Configuration/ICAP/Performance Agent must be enabled

SEE ALSO

Kernun: antivirus(5)

NAME

auth — client authentication on proxy

DESCRIPTION

The `auth` library handles the authentication of client users to proxies. The firewall administrator can restrict access to a proxy by requiring authentication of users. If this is the case, a user must not only connect to the proxy from an allowed host, but also prove his or her identity in order to be granted access.

The proxy authentication library currently supports the following authentication methods: none, password file, RADIUS, LDAP, and out of band authentication. Not all the methods are supported by all proxies.

None Authentication

The none authentication mode means in fact "no authentication is required". Any user is granted access without a requirement to provide their identity.

Password File Authentication

In the password authentication mode, each user is required to provide a valid user name and password. These are compared against a password file stored on the firewall. The file has a simple line structure: each line contains information about one user. It includes two fields delimited by a colon: the first one is the user name and the second one the encrypted password. Optionally, there can be the third colon-delimited field containing a comma-separated list of groups the user belongs to.

The **fwpasswd** utility can be used to manipulate the password file. See also fwpasswd(1).

RADIUS Authentication

The RADIUS Authentication mode also requires valid information to be received from the user. The user identity is verified by sending an authentication request to a RADIUS (Remote Authentication Dial In User Service) server. The RADIUS server checks the information from the user and tells the proxy whether the user has been authenticated successfully. The RADIUS authentication mode supports a simple name/password scheme as well as a securer challenge/response scheme.

If the name/password authentication is used, the user presents his/her name along with a password, which remains always the same. This represents a potential security hole, because the password could be sniffed while being transferred over the network.

The challenge/response scheme offers better security. The user enters his/her name, the RADIUS server sends back a challenge, which is changed after each (successful) authentication attempt of the user. The user performs some calculation on their local machine and obtains a response based on their password and the challenge received. He/she then sends the proxy only the response, but not the password itself. The RADIUS server verifies the response and notifies the proxy whether or not the user should be considered authenticated. An example of a widely

available challenge/response scheme is OPIE (One-time Passwords In Everything). However, the Kernun authentication library offers generic support of challenge/response protocols, which does not depend on a particular challenge/response scheme implemented by a RADIUS server.

LDAP Authentication

LDAP Authentication is somewhat similar to the preceding authentication methods; the difference is that the information about users is stored on an LDAP server.

The user presents his/her username and password in order to prove his/her identity. The firewall first checks the username/password with the information stored on the LDAP server. If the provided information is valid, the firewall obtains the list of groups the user is member of.

External Tool Authentication

The authentication can be managed also by your own tool, instead of LDAP or RADIUS servers.

The tool (with a pathname given in the configuration) is called whenever the user submits data. The data is supplied to the tool in the following environment variables:

USER User name.

RESP User password or response.

DATA Any data supplied by the previous call of the tool.

The tool responds by writing one status and zero or more data lines to the standard output. The status line has following format:

status data-length

The *status* must be a 3-digit status code, the first digit defines authentication result (2 – OK, 3 – another response needed, 4+5 – error). The *data-length* is the total length of all subsequent lines (including the LF characters). Every data line starts with the type character follwed by the colon and the text. There are following types of data lines:

M The message text to be displayed to the user.

D The internal data to be supplied in the subsequent tool invocation.

G The (comma separated) list of groups the user belongs to.

S The session timeout (in seconds).

The `M` data has the following meaning, according to the response codes:

2xx response The (HTML) message contains additional info concerning the session.

3xx response The (plain) message that needs to be displayed to the client in order for the authentication to be completed successfuly. It should be the challenge text in challenge response, an error message if the tool gives the user another chance to reenter response etc.

4xx/5xx response The (plain) message containing the reason of a nonrecoverable authentication error.

Out of Band Authentication

The out of band (OOB) authentication mode is special. All other authentication methods obtain authentication data (a user name, a password, or a response to a challenge) from the user via the proxied application protocol. They are thus usable only in proxies for protocols that support passing of authentication data, such as HTTP or FTP. OOB authentication first authenticates a user to the firewall as a whole. The user may enter the authentication data into an HTML form, or some other mechanism can be used to provide a list of authenticated users. Out of the box, Kernun contains tools for obtaining user list from a Samba server. The use of other sources of authentication data requires only modification of a single script. The authenticated user is associated to an IP address. All connections from this IP address are considered to be from this user. As the source IP address is available in all protocols, OOB authentication can be used in (almost) any proxy. Even the generic tcp-proxy and udp-proxy support OOB authentication.

OOB authentication is managed by an OOB authentication server, which is a specially configured http-proxy(8). The ooba-samba(1) script is used to read of the list of users from Samba.

NTLM Authentication

The NTLM authentication mode is specific to **http-proxy**. It allows a Web browser on a Microsoft Windows client operating system to be authenticated without user interaction, provided that the user is authenticated in an Active Directory domain. If the proxy requests NTLM authentication from a browser, several messages are exchanged among the browser, the proxy, the NTLM authentication support part of the Kernun system, and the Active Directory domain controller. In this process, the browser uses the user credentials acquired during the user login to the Active Directory domain. See the manpage for http-proxy(8) details about the NTLM authentication.

SEE ALSO

Kernun: fwpasswd(1), ooba-samba(1), auth(5), radius(5), http-proxy(8)

FreeBSD: opiekey(1), opiepasswd(1)

NAME

cluster — Kernun firewall redundancy cluster support

DESCRIPTION

To reduce the risk of system failure, Kernun firewall allows to build hot stand-by clusters. As the name suggests, apart from the main system, there is another host, usually equipped with the same features and configuration, ready to step up and start handling the communication automatically if the partner fails. The system that is currently in charge of traffic control is called *master node* within the cluster, while its idle peer is *backup node*.

One physical cluster of firewalls can provide several logical clusters (called *virtual clusters*). In such a case, each cluster may have a different master, which handles a certain subset of services. The advantage of such setup is that when all firewalls are functional, the workload is distributed among them.

The partners in a virtual cluster can be two equivalent peers, or one of them can be dedicated for the master role whenever is ready. This mode is called *preemptive* and the node is called *primary*. In non-preemptive mode, the current master node keeps the role all the time it is operational.

The signalling and switching of states between partners is implemented using a special *PIKE* protocol (as a replacement of VRRP and CARP ones). For this purpose, the couple is interconnected by a special "wire", ideally really an extra ethernet cable binding them. This interface is called *heart-beat* interface. The subprotocol of PIKE responsible for keepalive control is called *HELLO* and it can be controlled by a set of timeouts.

The monitoring is executed by a special daemon **pikemon** (see pikemon(8) and pikemon.cfg(5) manual pages) which is run as an ordinary Kernun application.

Technically, the traffic is lead through a bridge interface with assigned both a common shared IP address and MAC address. The master node sends a gratuitous ARP to inform all network nodes about current localisation of the MAC address. This bridge interface has assigned a single ordinary interface handling the real traffic and this interface becomes a member of the bridge interface when the node takes the master role and discontinues the membership when it drops the master role. The backup node can keep the IP address, or it unconfigures it whenever it looses the master role (so called *nomadic mode* configured within the bridge `interface` section). One virtual cluster can consist of more bridge interfaces.

The operation ability is periodically monitored by the **pikemon** daemon using the ICMP ECHO messages. For every virtual cluster, there can be several *ping* groups, i.e. lists of hosts from which at least one has to respond within given timeout, otherwise the group (and whole node) is considered to be "down".

The node role can be also bound to a set of services. Thus, a set of Kernun components should be started only in case of master role and/or a special command should be executed when taking or dropping the role.

SEE ALSO

Kernun: pikemon.cfg(5), system(5), cml(8), pikemon(8)

NAME

configuration — general syntax of configuration files

DESCRIPTION

Configuring a proxy

Each Kernun firewall proxy must be started with a configuration file (the `-f` option). In fact, in the current version of the Kernun Firewall, these configuration files (so called *low-level* configuration files) are generated by the *CML* (Configuration Meta Language) tool (see cml(8)). Nevertheless, the syntax rules for the CML (*high-level*) configuration files are the same as for the low-level ones.

This manual page describes the general syntax rules for the configuration file. The definitions governing the configuration of particular applications are described in the pertinent section 5 manual pages (e.g. ftp-proxy.cfg(5) for **ftp-proxy**). In the CML, the description can be displayed by

```
/info descr topic
```

Configuration versions

In order to make detection of errors stemming from the use of an old configuration files for newer applications easier and to simplify conversion of older configurations to newer formats, every configuration file must contain the version of Kernun firewall it is designed to work with. For example, version 2.3 of Kernun firewall expects the following line at the beginning of every configuration file:

```
version KERNUN-2_3-RELEASE;
```

The version tag is Kernun's distribution version string. It can be obtained by running a Kernun application with the `-v` option. Also, every section 5 manual page refers to the proper version in the *VERSION* section. Configuration files generated by the CML have this line generated automatically.

SYNTAX

Structure of the configuration file

A configuration file consists of sections and items.

An *item* is a configuration "command". It begins with a word (the item's keyword) followed by its *elements*, that is, attributes represented by values of various types. An example of an item command with its elements:

```
command { RETR, STOR } size 33K;
```

An item can be defined as *repeatable* (all of its occurrences are chained) or *non-repeatable* (only one occurrence is allowed).

A *section* is a block of items and/or subsections. It begins with a word (the section's keyword) followed by items and subsections enclosed in a pair of braces. A repeatable section must have its *name* before the first opening brace. Examples:

```
ftp-proxy FTP {           # repeatable section ftp-proxy named FTP
  proxy-user kernun;      # item proxy-user
  msgs {                  # non-repeatable subsection msgs
    welcome "Welcome";    # item welcome
  }
}
```

The allowed sections, items, types etc. for each application configuration are described in the section 5 manual page (e.g. ftp-proxy.cfg(5) for the **ftp-proxy**). In order to facilitate configuring of similar functions in different proxies by the same means, the configuration description consists of so-called *prototypes*, from which particular proxy-dependent constructions are derived. For instance, the `acl` section prototype collects all general sections and items used in ACL declarations (see access-control(7)), the `acl-1` section prototype is derived from the `acl` prototype by modifying some features to fit phase 1 ACL decisions and the **ftp-proxy** `session-acl` section is derived from the `acl-1` prototype to fit session-initialisation phase decisions in FTP. In such cases, the special section 5 manual page describes only the modifications to the prototypes and the prototypes themselves are described in their own section 5 manual page (e.g. acl(5) in this case).

More occurrences of a non-repeatable section or of a repeatable section with the same name lead to an error.

The order of items/subsections with different names is irrelevant. The synonymic items/subsections are searched in the order they are entered and the order may be meaningful (e.g. first-match), if it is stated so in the section 5 manual page. Otherwise, it is simply an (OR-ed) list of information.

Configuration atoms

The basic term of configuration description is an *atom*. There are the following atom types in Kernun configuration:

integer A positive integer value in decimal or hexadecimal (C-style: `0x`*hex*) form; the decimal ones can have 'K', 'M', 'G' and 'T' suffices meaning 1000, 1000000, 1000000000 and 1000000000000 values respectively, or 'Ki', 'Mi', 'Gi' and 'Ti' suffices meaning 1024, 1048576, 1073741824 and 1099511627776 values respectively, or 'd', 'h' and 'm' suffices meaning days (x 86400), hours (x 3600) and minutes (x 60) respectively.

fractional A positive fractional value with up to three decimal digits (e.g. `1.95`, `2`). Internally, this atom is stored as an integer value (multiplied by 1000) so it behaves like an integer.

word A sequence of letters, digits, underscores ('_') and hyphens ('-') (e.g. `ftp-data`, `kerberos_master`); words are case-insensitive.

string A string of characters enclosed in double quotes. C-style escapes ('\t', '\r', '\n', '\"', '\\' and '\x*hex*) must be used to code special characters. If a string is a concatenation of *word*s, dots ('.') and at-signs ('@') then it need not be enclosed in quotes (e.g. `root@tns.cz`).

IP address An IPv4 address in dotted decimal format. It MUST be enclosed in square brackets (e.g. `[127.0.0.1]`).

IP address with mask IPv4 netmask must follow immediately after an IP address (inside the square brackets), starting with a slash; the allowed formats include single number (the number of bits), dotted decimal format and hexadecimal format (e.g. `[127.0.0.0/8]`, `[10.0.21.0/255.0.255.0]`, `[10.0.0.0/0xFF000000]`). Note that exactly four bytes must be specified in the dotted notation, even if they are zero.

regexp A UNIX regular expression enclosed within a pair of slashes. Inside regexp, literal slashes and spaces must be escaped with backslash (e.g. `/ab\ \/\ cd/` matches string `"ab / cd"`). The character 'i' following immediately after the closing slash makes this regexp to be case-insensitive (e.g. `/abc/i` matches all of `"abC"`, `"AbC"`, `"ABC"` etc.).

Atoms can be separated by a sequence of white-space charaters (space, tab, newline). Thus, newlines (except in comments) have the same meaning as a single space.

A comment starts with a hash ('#') character and ends at the end of line. Note that in the CML, comments are allowed on the session level only (i.e. between two items/subsections, not within items, after section names etc.).

Types

Item elements have defined types and only values (i.e. atoms) of proper type can be used in configuration (see Section C). The following table defines the atom types that are allowed for known types:

`uint8` ?? value (size: 8 bits).

`uint16` ?? value (size: 16 bits).

`uint32` ?? value (size: 32 bits).

`uint64` ?? value (size: 64 bits).

`port` ?? value or ?? (recognized by getservbyname(3), usually stored in `/etc/services` file).

`time` ?? value of daytime in form hhmm.

`fract` ?? value (size: 32 bits after multiplying by 1000).

str ?? or ?? value; known keywords used as simple string must be quoted.

regexp Regular expression (see ??).

host Hostname (expressed as a ??) or an ?? without mask.

addr Interface ??. If mask is omitted, default mask according to class is assumed. Local part of address must not consist of all-zeros or all-ones.

net Network ??. If mask is omitted, default mask according to class is assumed. Local part of address must be zero.

sock Hostname or IP address followed by colon (':') and port number or name (e.g. `[127.0.0.1] : 3333`, `ftp.freebsd.cz:ftp`).

key Exactly the keyword specified in configuration description (Section 5 manual page) of particular item can be used.

Enumerations

For some types, mnemonic names may be used instead of direct integer values. For instance, the `dns-type` type recognizes the word `CNAME` instead of the value `5`. Such types are called *enumerations*. For some enumerations, the use of mnemonic names is obligatory since the corresponding integer values have no real meaning (e.g. FTP command numbers in the `ftp-cmd` type). The description of allowed enumerations is a part of every section 5 manual page. In the CML, you can display the description of an enumeration using

```
/info enum enumeration
```

Section-Name Types

Every repeatable section at the global level can be used to create a special type from the section's keyword and the suffix `"-NAME"`. Prospective values of such a type include the names of all sections (of the proper type). Example:

Suppose the following definition:

```
radius-client FIRST {
  server radius.xyz.com "shared secret";
}
```

Then, the following item is correct, if its second element is of the `radius-client-name` type:

```
auth radius FIRST;
```

Lists

A sequence of zero or more instances of a base-type (see above) is called a *list*. List items are enclosed in braces ('{' and '}') and separated by commas (','). If the list contains only one member, the braces are not neccesary. The type name for a list is derived from the base type by suffixing with "-LIST". Example:

The following value is acceptable for the type port-list:

```
{ ftp, ftp-data, 512 }
```

It represents three values (the ftp and ftp-data ports, and port 512).

A list can have another list as a member. In this case, all sublist members become members of the main list. Example:

The above list can be also expressed as:

```
{ { ftp, ftp-data }, 512 }
```

Sets

A set is another way to specify more values acceptable for an element. The type name for a set is derived from the base type by suffixing with "-SET".

The basic difference between lists and sets is that the former can be used by an application for 'going through all list members', and the latter only for testing 'whether a value matches the set or not'. That is why sets can have the following special members:

Ranges of values Two values separated by a hyphen ('-'). Care should be taken if the lower value is represented by a word; in such a case, the word must be separated from the hyphen by at least one white-space character (e.g. 'ftp-http' is treated as one word, while 'ftp - http' is a correct range).

Excluding members Members prefixed by an exclamation mark ('!') are treated as "value that is not a member of current (sub)set". The matching algorithm works as FIRST-MATCH. Therefore, the excluding members must precede non-excluding members containing excluded values in the same (sub)set. Otherwise, the values are treated as being members of the (sub)set.

ANY value The wildcard character * represents a set containing "all acceptable values" (possibly except the ones specified in excluding members of the (sub)set).

Examples:

The following values are acceptable for the type port-set:

```
{ ! { ftp - ftp-data }, * }      # all ports except FTP ones
{ ! { ftp - ftp-data }, 1-1024 } # all generic ports except FTP
```

The following value is syntactically correct, but has no meaning, because FTP ports were excluded *after* including them:

```
{ 1-1024, ! ftp - ftp-data }      # all generic ports
```

The following value is syntactically correct as well, but has no meaning either, because FTP ports were excluded in a subset and no * or range item is present within this subset:

```
{ { ! ftp - ftp-data }, 1-1024 } # all generic ports
```

In the last two examples above, configuration reader logs a warning.

For two types, suffix "-SET" extends the possibilities of specifying values:

str-set member Besides strings and words, *regular expressions* can be specified. Note that matching of strings in str-set is done in ignore-case manner, whereas ignore-case matching of regular expressions must be forced by the 'i' suffix (see above the Configuration atoms paragraph).

host-set member Besides hostnames, *regular expressions* can be specified. IP addresses can have a *netmask*.

Example:

The following host-set value matches any hostname ending with .cz and any address matching 10.0.*.1:

```
{ /^.*\.cz$/, [10.0.0.1/0xFFFF00FF] }
```

Matching of host values against host-sets is slightly more complicated, see host-matching(7) for details.

Elements

The order and types of elements in an item are fixed. Each element of an item has a type, as discussed above. Unless special conditions (stated below) apply, all elements must be included in every occurrence of an item.

An element can be omitted only if it is defined (in the pertinent Section 5 manual page) as an element with a default value and if one of the two following conditions are true:

1. The element is defined to have an arbitrary keyword.

2. No following element of the item is to be used.

Moreover, some items can be used in several different forms; the particular form is selected by the value of a special element (so called *branching* element).

An example of a section 5 manual page: The configuration of the **acl** library component consists of the following prototypes:

* user ... ;

...

Description:

user none;

user [name [*name* [group *group*]];] User and group specification.

<**branching element**> **(type: user-auth-spec, optional, default: name)**

name **(type: str-set, optional, default: *)** user name (authenticated on firewall)

group ***group*** **(type: str-set, optional, default: *)** list of groups; if present, both NAME and GROUP must match

This repeatable item can be used in two forms selected by the first element. The former one consists of the branching element (none) only. The latter one may have the branching element value (name) specified, or omitted, and many variations of the other elements are valid.

The following forms are all equivalent and only explicitely confirm the default values:

```
user name * group *;
user name *;
user name;
user *;
user;
```

Two examples of defining user-list of two members:

```
user { user1, user2 } group *;
user name { user1, user2 };
```

Two examples of defining both lists as single-member ones:

```
user name { user1 } group grp1;
user user1 group { grp1 };
```

An incorrect example (group-list cannot be specified without user-list):

```
user group *;
```

An incorrect example (the word none cannot be used as a username):

```
user none group none;
```

Possible corrections of the previous mistake:

```
user "none" group none;
user { none } group none;
```

SEE ALSO

Kernun: ftp-proxy.cfg(5), log(5), host-matching(7)
FreeBSD: getservbyname(3), services(5)

NAME

data-matching — generic data matching and processing in proxies

DESCRIPTION

In addition to checking compliance with an application protocol specification, a proxy can also scan protocol payload data. This capability comprises features (described elsewhere), such as HTML filtration or MIME processing, and, in some proxies (only **http-proxy**(8) at the time of this writing), generic configurable data processing (described in this manual page).

Generic data processing is performed by the software module `mod-match`. Its parameters are specified in the configuration section `data-match`. The section can be referenced by an ACL of a proxy that should use the data matching feature. The matching module is inserted to the data flow between a client and the server, separately in each direction. The module scans the initial part of the data; its size can be set in the module configuration using the `max-size` item. As blocks of data are received by the proxy, the scanning process is repeated for newly arriving data, in accordance with the parameters `step-size` and `step-match`. A sequence of checks, defined by repeatable items `test`, is executed. Each test can accept or deny the data. A test of the `html-alert` type can either report a match to the proxy log only, or report and deny, depending on the `deny` flag. The decisions are based on regular expression matching. There are also some more complex tests, suitable particularly for processing of submitted HTML form values in **http-proxy**. For detailed description of avaliable test types, see mod-match(5).

Database files used by the tests `html-hash`, `html-alert`, and `html-replace` are managed by the program **html-match-db**(1).

The test type `html-save` saves values of HTML forms to a text file as hexadecimal strings in a format compatible with Snort rule syntax. The test type `html-hash` saves hashes of HTML form values, so that it is later possible to test whether a value is stored in the database, but it is impossible to get the original values from the database. The test type `html-replace` uses a database with encrypted replacement data and decrypts them by a key obtained from the HTML form values being replaced; therefore, the replacement data cannot be obtained from the database without knowing the corresponding data to be replaced.

DATA MATCHING IN PROXIES

HTTP proxy

It is possible to scan HTTP request and response body. Body processing is enabled and configured by the configuration items `request-acl.request-body-match` and `doc-acl.response-body-match`. Actions `html-save`, `html-hash`, `html-alert`, and `html-replace` are most useful when used for processing HTTP request body.

SEE ALSO

html-match-db(1), mod-match(5), http-proxy(8)

NAME

doctype-identification — document type recognition methods and configuration

DESCRIPTION

The third-level ACL, `doc-acl`, has as an entry condition `mime-type` specification (see access-control(7), configuration(7) manual pages). There are several methods that can be used to recognize the type of a document and several ways to control them in the configuration. The recognized type can be used in ACL matching, and also can be sent instead of the original Content-Type (see `force-doctype-ident` in acl(5) manual page).

METHODS

The following methods are used by Kernun applications. The order of their usage is defined in the configuration. They are tried subsequently until some succeeds; in none does, the type remains unknown, matching the string `""` only.

Original Content-Type

The default metod is quite simple. The proxy uses the type declared by the document originator (server for download, client for upload) in the `Content-Type` header (e.g. SMTP or MIME header).

This method is very clear and fast, it needs no special configuration, but it has two disadvantages: some protocols, such as FTP, cannot use it, and the others are quite vulnerable to type faking.

Method name: `content-type`

File Name Extension Mapping

If the name of the document (or URL) is available, the type can be guessed by searching a database that maps filename extensions to MIME types.

This method has characteristics similar to the default one (clarity, speed, but also vulnerability). It needs somewhat more information to be well configured, namely a file with the extension-to-type mapping database.

Method name: `extension`

Magic Number Recognition

The last method is similar to the one used e.g. by the system command **file**. The proxy reads the initial block of the document (the size of the block is configurable) and tries to guess the file type based on this block, with the help of a magic number file (see magic(5) manual page).

This method is the most complicated — it needs to gather some data from the document originator before the control decisions are made. On the other hand, this method yields results very close to the real content of documents, regardless of data originator's instructions.

Method name: `magic`

CONFIGURATION

In order to ensure correct operation, it is necessary to define the order of the methods and some additional information. If no order is given, only the `content-type` method is used.

Global Information

Each proxy can set global parameters and the default order of methods in the `doctype-identification` section (see application(5) manual page).

`mime-types` *`shared-file-name`*`;` This item must be used if the `extension` method is used anywhere in the proxy configuration.

`magic` *`[filename [scan-size]]`*`;` This item should be used if nonstandard filename or different block size is to be used for the `magic` method.

`order [for` *`direction`*`]` *`order`*`;` This item is repeatable, but the only reason for this is to enable different method order for upload and download. In protocols where direction makes sense (e.g. FTP or IMAP4, contrary to POP3 or SMTP), the keyword `for` with a value can be used to distinguish between upload and download definitions. The *`order`* is simply a list of above-mentioned keywords (method names).

Order Redefinition

Each ACL on the first two levels (with some exceptions, such as `delivery-acl` in SMTP) can redefine the default order (see acl(5) manual page).

`doctype-ident-order [for` *`direction`*`]` *`order`*`;` This item has the same syntax and semantics as the `order` item of the proxy global `doctype-identification` section.

For a particular transfer, the order in the second-level ACL is searched for; then (if not found), the one in the first-level ACL is tried and, finally, the order from the proxy global section is used (if any).

EXAMPLE

Suppose the following configuration:

```
proxy ... {
  doctype-identification { ...
      doctype-ident-order for download { extension, magic };
  }
  session-acl sa-1 { ...
```

```
        doctype-ident-order for upload { content-type, magic };
    }
    session-acl sa-2 { ...
        doctype-ident-order for download { };
    }
    session-acl sa-3 { ...
        doctype-ident-order { content-type, magic };
    }
```

In this case, downloads according to sa-1 use the "{ extension, magic }" order, while uploads use "{ content-type, magic }"; sa-2 downloads use no method (type will be "") while uploads use default method (content-type); finally, transfers via sa-3 use "{ content-type, magic }" (in both directions).

SEE ALSO

Kernun: acl(5), application(5), access-control(7), configuration(7)

FreeBSD: file(1), magic(5)

NAME

host-matching — configuration semantics of lists of hosts

DESCRIPTION

The Kernun Firewall configuration files can contain lists of hosts to be matched against client/server hostnames or IP addresses. This man page describes the semantics of such lists.

List Members

According to the Kernun Firewall configuration file syntax (see configuration(7) for details), you can create a host list containing members of several types:

- *hostname*, e.g. www.tns.cz (meaning: www.tns.cz and all its addresses; the name is resolved to IP addresses during proxy startup and then periodically refreshed when needed)
- *regular expression*, e.g. /^www\..*\.cz$/ (meaning: all hosts named 'www' in domains under TLD 'cz')
- *single IP address*, e.g. [192.168.1.1] (square brackets are not metasymbols, they mark IP addresses in configuration)
- *IP address range*, e.g. [192.168.1.12]-[192.168.2.5]
- *IP address with mask*, e.g. [10.0.1.0/255.0.255.0] or [10.0.1.0/0xFF00FF00] (means: all IP addresses with the 1st and the 3rd bytes equal to 10 and 1, respectively)
- *IP address with mask size*, e.g. [192.168.1.0/24] (meaning: all IP addresses with the first 24 bits defined by 192.168.1.0 address)
- *symbol* * (meaning: all hosts)
- *sublist* (meaning: all hosts represented by a sublist)

Members can be prefixed with an exclamation mark ('!'); in such a case, all names/addresses represented by the particular member are excluded from the list (so-called excluding or negative members).

For example,

```
{ ! [192.168.3.3]-[192.168.3.8], * }
```

means: All but the six addresses specified by the range.

Note that if a sublist is negative, it means that all hosts represented by the sublist are excluded. It does NOT mean that all hosts excluded in the sublist are automatically included in the superlist.

For example,

```
{ ! [10.1.1.1], [10.0.0.0/8] }
```

means: The network of 10.* except for 10.1.1.1. However,

```
{ ! { ! [10.1.1.1], [10.0.0.0/8] }, ...
```

means: All 10s except 10.1.1.1 are excluded. It does not imply inclusion of 10.1.1.1!

As a conclusion, if a negative member is not followed by a positive member on any list level, the resulting list has the same effect as an empty list.

For example,

```
{ ! { ! [10.1.1.1], [10.0.0.0/8] } }
```

means: no hosts are allowed (even [10.1.1.1] is not allowed).

The order of appearance of members in the list is important. Each list is searched from the left to the right and the first match, either positive (for non-excluding member) or negative (for excluding member) stops the search and defines the result (success or failure) that is passed to superlist.

Warning

In fact, hostnames are "stronger" than regular expressions. Let us suppose a host 'a.b.c' with IP address 10.1.2.3 and two lists in the configuration: { a.b.c } and { /^a\.b\.c$/ }. If the proxy is testing the hostname 'a.b.c', both lists match. However, if it is testing the IP address, it doesn't know the hostname and the latter list doesn't match. The former one matches because it behaves like the { a.b.c, [10.1.2.3] } list.

Matching Keys

The host being matched is known either by an IP address or by a hostname. This is the so-called *primary key*. However, the matching algorithm may, in some cases, use also a set of so-called secondary keys:

If the primary key is a *hostname* (e.g. when an HTTP request contains a hostname and the proxy is trying to match it against a particular ACL server list), it is first resolved by a DNS and all IP addresses asociated with it create its secondary key-set.

Matching Modes

If a primary key resolves to more than one secondary key, we must define how to match those secondary keys. Specifically: is it sufficient if just one of the secondary keys matches for the whole key to match, or must all of the secondary keys match? We recognize these two modes of matching of a secondary key-set and use them both, each in a different context.

***Permitive Matching* (MATCH-ALL Mode)** This mode is applied if the list is used to permit some feature, i.e. when matching a host against a server-list in a non-denying ACL (a standard ACL without the `deny` option).

In this case, if the primary key or *all* secondary keys are positively matched, the result is *positive* (i.e., the host is accepted). If the primary key or any secondary key is negatively matched, the result is *negative*.

***Restrictive Matching* (MATCH-ANY Mode)** This mode is used if the list is used to deny some feature, i.e. when matching a host against a server-list in a denying ACL (an ACL with the *deny* option set).

In this case, if the primary key or *any* secondary key is positively matched, the result is *positive* (and the host will be denied). If the primary key or all secondary keys are matched negatively, the result is *negative*.

If a list contains a positive (non-excluding) sublist, the matching mode for the sublist is the same as for the superlist. However, if the sublist is negative (excluding), the matching mode changes to the opposite one. For instance, if a hostname resolving to two IPs 10.1.1.1 and 10.2.2.2 is to be rejected by a MATCH-ANY list and we choose to do it by a negative sublist, the sublist must be as follows:

```
..., ! { [10.1.1.1], [10.2.2.2] }
```

because it will be searched in MATCH-ALL mode (all IPs must be excluded for the host to be excluded).

A rule that applies to both modes is that if the top-level list has been searched through without *positive* result, the result is *negative*.

Matching Algorithm

The secondary key-set is created for the host being matched, and every element of the set is flagged as NOT MATCHED.

Next, the list members are taken in the order of appearance in the list and

- if the member is a *regexp* then

  [for NAME primary key]

  the hostname is checked against the member and if it matches, the result is stated (either positive or negative, according to the member)

  [for IP key]

  the IP address never matches against regexp

- if the member is a *hostname* then

  [for NAME primary key]

  the hostname is checked against the member (hostname member against hostname key, then IP addresses that the list member resolves to against IP addresses in the secondary key-set) and if it matches, the result is stated (either positive or negative, according to the member)

[for IP key]

the IP address is checked against the set of IP addresses that the list member resolves to

- if the member is an *IP address* (or network or range of addresses) then

[for NAME primary key]

each of addresses in the secondary key-set that are still NOT MATCHED is checked against the member and if a match is found, the secondary key is flagged as POSITIVE or NEGATIVE (depending on the "sign" of the list)

[for IP key]

the IP address is checked against the member and if it matches, the result is stated (positive or negative)

- if the member is $\star$ (positive), the result is positive
- if the member is $\star$ (negative), the result is negative.

Now,

[for permitive mode]

- if any set-element is flagged NEGATIVE, the result is negative
- if all set-elements are flagged POSITIVE, the result is positive

[for restrictive mode]

- if any set-element is flagged POSITIVE, the result is positive
- if all set-elements are flagged NEGATIVE, the result is negative

If the result is not stated, go on searching the list.
If the result is not stated after passing the end of the list, the result is negative.

EXAMPLES

Suppose the following configuration:

```
ACL crazy {
  SERVER { ! [10.1.2.3], /\\.crazy\\.com$/ };
  DENY;
}
```

We want to deny access to the crazy.com domain, except for the host 10.1.2.3. Suppose that www.crazy.com administrator tries to compromise this restriction and defines another "fictitious" interface to www.crazy.com with the address 10.1.2.3. A user is trying to contact www.crazy.com.

It resolves to two addresses, and one of them matches the first member of the list. However, this is not satisfactory for restrictive matching (used for DENY), the algorithm continues and the second member of the list is matched with a positive result and the access is denied.

Suppose another configuration:

```
ACL friend {
  SERVER { [10.0.0.0/255.0.0.0] };
  COMMAND * PERMIT;
}
```

We want to allow access to the A-class network 10. However, a host www.friend.com has another interface to another network. This is why connections to www.friend.com will not be allowed - not all its IPs will be flagged as POSITIVE during the matching process. (Of course, this rejection is not necessarily fatal - there can still be another ACL suitable for this host.) If we want to allow the host in this ACL, ALL of its IP addresses or its NAME must be present in the list. Connection to any IP address (not a hostname) within the network 10 will be granted.

SEE ALSO

access-control(7), configuration(7)

NAME

ips — intrusion detection/prevention system and the related aspects

DESCRIPTION

suricata(1) IDS/IPS is integrated in Kernun UTM where it known as `ids-agent`.

Configuration

`ids-agent` is configured in the `adaptive-firewall.ids-agent` section on the `system` level of the Kernun UTM configuration. See system(5) and adaptive-firewall(5).

`ids-agent` can run in two modes, selected by the presence of item `system.adaptive-firewall.ips`:

IDS mode The suspicious traffic is logged but no other action is taken.

IPS mode The suspicious traffic is logged and reported to pf-control(8) which blocks it.

The traffic from the interfaces named by items `iface` are analyzed by the ids-agent.

Rule refreshments aspects can be configured using the `rules-download` section.

Providing the rules

Section `rules` can be used to define the rules directly in CML.

Use item `include-rules` to reference an external file with the rules in suricata syntax. The file will be copied, and the copy-filename will be referenced in ids-agent configuration.

Use item `add-rule` to define the rule directly in CML syntax. The rules will be flushed into a text file in suricata syntax, and referenced in ids-agent configuration.

Item `change-rules-to-block` can be used to change the action of rules specified by ID to block.

Item `disable-rules` disables rules that were published as enabled, effectively removing them from existence for ids-agent.

Item `enable-rules` enables rules that were published as enabled.

Items `global-rate-filter` and `rule-rate-filter` can be used to conditionally change the action of either all rules or only of given rules. It specifies how many times a given rule needs to be detected before its action is changed to a different given action. This is useful for example to make the rule alert 10 times within a minute before the communication is blocked.

Items `global-suppress` and `rule-suppress` are similar to `*-rate-filter`, the difference being that the condition is not number of occurences but rather the IP addresses. These items are useful for making certain rules not apply for a given IP address.

Items `global-threshold` and `rule-threshold` make given rules take action only when they are detected certain times within given time interval.

Items `global-suppress` and `rule-suppress` make given rules not apply for given IP address and direction. It is similar to adding IP address to a whitelist, only more specific, because it is possible to specify the rule and the direction of communication.

Item `modify-rules` changes given rules by replacing a regular expression matching with a given string. This is designed to be used only when none of the options above are applicable because there is no validation of the replacement. If ids-agent gets invalid rules, it ignores them, parsing only valid rules.

Downloading the rules from the Internet

The rules can be automatically, periodically downloaded from Kernun download server using program **pulledpork**. Section `system.update.adaptive-firewall` specifies the parameters for it.

Item `source` defines the source of the rules. There are predefined values for downloading the emerging-threats rules. The `custom` URL can be given — see **pulledpork** documentation on the format of the `rule_url` it expects.

Item `schedule` defines the rules update policy. The rules can be manually (re)downloaded by command **kat ids-agent-update-rules**. The corresponding command can also be performed from the GUI. When the rules are downloaded and processed, the signal is sent to ids-agent, so it reloads the new rules automatically.

The rules can be altered by **pulledpork** by items in section `rules`. The following types of modifications are possible:

- A rule that is distributed as disabled can be enabled by item `enable-rules`. A disabled rule is commented out in the downloaded rule file so the IPS engine would otherwise ignore it.
- When disabling a rule, the administrator has more options. A rule can be disabled unconditionally by item `disable-rule` or only for certain IP addresses by item `rule-suppress`. It is also possible to disable all rules for certain IP addresses by item `global-suppress`.
- In IPS mode, it is sometimes desired to change the rule action from `alert` to `drop` or `reject`. This can be done by items `change-rules-to-drop` and `change-rules-to-reject`.
- Items `rule-rate-filter` and `global-rate-filter` can be used to change the rule action after the rule matched a certain number of times within a specified time frame. Similarly, items `rule-threshold` and `global-threshold` alter the rule so it is applied only after it matches a certain number of times within a specified time frame.
- When the above methods are not sufficient, it is also possible to modify a certain rule by providing a regular expression and a replacement string in item `modify-rules`.

The flags used as the command-line arguments when starting **pulledpork** can be redefined by item `downloader-extra-flags`.

The **pulledpork** configuration is generated in `/usr/local/kernun/etc/pulledpork.conf`. The rules are downloaded to `/usr/local/share/suricata/rules/downloaded-pulledpork.rules`. This file is automatically included in the ids-agent configuration.

Fine-tuning the ids-agent configuration

The configuration of ids-agent is stored in `/usr/local/kernun/etc/suricata.yaml`. Its contents is merged from two sources:

- the configuration provided by the system administrator in item `ADAPTIVE-FIREWALL.IDS.AGENT.ENGINE.CFG-FILE`. It is a recommended practise to use a file based on `/usr/local/kernun/conf/samples/shared/ids-agent.yaml` which is the file that is used if the item is not specified. Be aware when using a custom `CFG-FILE`, the configuration can become obsolete and even unusable after a system upgrade, in which case it is necessary to consult `/usr/local/kernun/conf/samples/shared/ids-agent.yaml` for changes.
- configuration generated by Kernun UTM (`/usr/local/kernun/etc/kernun-suricata.yaml`).
  Namely, this file specifies the rules (section `ADAPTIVE-FIREWALL.IDS.AGENT.RULES`) and the logging/output definition.

Network scanning of ids-agent

The traffic from the interface(s) selected by item(s) `iface` is scanned by ids-agent via PCAP.

SEE ALSO

Kernun: system(5), kat(8), pf-control(8)

FreeBSD: suricata(1), **pulledpork** (`/usr/local/share/doc/pulledpork/README.*`, `/usr/local/etc/pulledpork/*`)

NAME

kernun — signpost to Kernun firewall manual pages

DESCRIPTION

Kernun is a flexible toolkit that makes it possible to build secure network firewalls combining application-specific proxy gateways with stateful packet filtering and address translation (NAT), virtual private networks, network IDS and detailed log analysis.

Individual application proxies, important aspects of the configuration, as well as internal interfaces implemented in Kernun support libraries are documented in their respective manual pages.

The best way to start using the Kernun firewall is to read the *Kernun Firewall Handbook*, especially the tutorial. After learning Kernun firewall basics, detailed information can be found in these manual pages, which are available also as the reference part of the Handbook. The most important administrative tasks are covered by the following manual pages: kat(8), cml(8), and kernun.cml(5). It may be also helpful to examine the initial configuration in `/usr/local/kernun/conf/kernun.cml`, which is generated after the installation, and configuration samples that can be found in `/usr/local/kernun/conf/samples`.

Components

The Kernun firewall consists of:

- The underlying FreeBSD operating system, see also intro(1).
- A high-level configuration interface that integrates the configuration of most components of the Kernun firewall host in a single file, see also cml(8), kernun.cml(5) and configuration(7).
- A graphical user interface (GUI) for remote configuring and monitoring of the Kernun firewall. The GUI is available at least for FreeBSD and Microsoft Windows. It is an open source application so it can be ported to other platforms supported by the Qt toolkit (most notably Linux). The GUI is described in the Kernun Firewall Handbook.
- The command line administration tool for easy configuring and monitoring the firewall, see also kat(8).
- A set of protocol-specific and generic proxies for traffic inspection on the application layer, each with its own configuration mechanism, see also dns-proxy(8), ftp-proxy(8), gk-proxy(8), h323-proxy(8), http-proxy(8), imap4-proxy(8), pop3-proxy(8), sip-proxy(8), smtp-proxy(8), sqlnet-proxy(8), tcp-proxy(8), udp-proxy(8), dns-proxy.cfg(5), ftp-proxy.cfg(5), gk-proxy.cfg(5), h323-proxy.cfg(5), http-proxy.cfg(5), imap4-proxy.cfg(5), pop3-proxy.cfg(5), smtp-proxy.cfg(5), tcp-proxy.cfg(5), udp-proxy.cfg(5), and configuration(7).
- A PF (packet filter) package for traffic inspection on the network and transport layers, network address translation (NAT), and traffic shaping. These functions are controlled by a component pf-control, see also pf-control(8), pfctl(8), pf.conf(5).

- Log processing and runtime monitoring tools that provide statistics and online alert messages, see also sum-stats(1), switchlog(1), logsurfer(1), monitor(1), and rrd(1). The GUI also provides a wide range of log processing and monitoring features.
- User authentication based on various methods including password files, RADIUS, LDAP, and out-of-band authentication (with user login via a Web form or via a Samba server) see also auth(7).
- A virtual private network module, see also openvpn(8).
- NTP, DHCP, DNS, ICAP, and SNMP servers, see also ntpd(8), dhcpd(8), and named(8), and icap-server(8), and snmpd(8).
- An intrusion detection and prevention module, see also adaptive-firewall(5).
- The SpamAssassin antispam module, see also spamassassin(1).
- Web filtration functionality based on the interface to an external Proventia Web Filter.
- The Adaptive Traffic Routing for dynamic loadbalancing, see also atrmon(8).

Features

Components of the Kernun firewall have the following common features:

integrated configuration It covers key system components and all proxies. See kat(8), cml(8), kernun.cml(5).

hot-standby backup firewalls See cluster(7).

intrusion detection/prevention system See ips(7).

name resolving See resolving(7).

sophisticated logging See logging(7).

authentication See auth(7).

fine-grain access-control See access-control(7), host-matching(7), data-matching(7), time-matching(7).

data content inspection See antivirus(7).

document type recognition See doctype-identification(7).

runtime monitoring See monitoring(7).

enhanced network I/O with traffic shaping See netio(7), traffic-shaping(7).

efficient process management See application(5), tcpserver(7), udpserver(7).

network transparency See transparency(7), port-range-listen(7), listen-on(5).

administrative accounts with two levels of privileges The administrator accounts have privileges equivalent to the root user. The auditor accounts are allowed to view the configuration and logs, but do not have privileges to manipulate the state of the firewall (change configuration, start or stop proxies, etc.). See system(5).

SEE ALSO

Kernun: monitor(1), rrd(1), sum-stats(1), switchlog(1), dns-proxy.cfg(5), ftp-proxy.cfg(5), gk-proxy.cfg(5), h323-proxy.cfg(5), http-proxy.cfg(5), imap4-proxy.cfg(5), kernun.cml(5), listen-on(5), pop3-proxy.cfg(5), application(5), smtp-proxy.cfg(5), sqlnet-proxy.cfg(5), system(5), tcp-proxy.cfg(5), udp-proxy.cfg(5), access-control(7), antivirus(7), auth(7), cluster(7), configuration(7), data-matching(7), doctype-identification(7), host-matching(7), ips(7), logging(7), monitoring(7), netio(7), port-range-listen(7), resolving(7), tcpserver(7), time-matching(7), traffic-shaping(7), transparency(7), udpserver(7), atrmon(8), cml(8), dns-proxy(8), ftp-proxy(8), gk-proxy(8), h323-proxy(8), http-proxy(8), icap-server(8), imap4-proxy(8), kat(8), pf-control(8), pop3-proxy(8), smtp-proxy(8), sqlnet-proxy(8), tcp-proxy(8), udp-proxy(8)

FreeBSD: intro(1), logsurfer(1), spamassassin(1), suricata(1) pf.conf(5), openvpn(8), dhcpd(8), named(8), ntpd(8), pfctl(8), snmpd(8),

NAME

logging — Kernun firewall logging facility

DESCRIPTION

All native applications of the Kernun firewall use a special interface to contact the syslog daemon, and a common format of messages. Several attributes of the logging interface can be set in each application configuration by the same means. Information about packet filter (PF) events are read from the pflog(4) and pfsync(4) pseudodevices by the **pf-control** daemon and logged in a similar way. The PF logging is configured also by the same means; moreover, you can precisely configure which events of PF will be logged by special `log` option of PF rules. For detailed description of the logging configuration syntax, see the Section 5 application manual page or log(5).

There are two logs created by Kernun: the *stats log* and the *debug log*.

Stats log

The *stats log* logs each event (request, session, email, etc. — depending on the traffic type) as a single record. The statistics is generated out of this log. The log is (by default) stored in `/var/log/kernun-stats`.

Since there can be a limitation to the maximal length of a single row in the logging facility, a long record can be split into more rows. In that case, there is '\' at the end of the record to be continued, and '~' at the beginning of record that is the continuation of some record.

The statistics is generated from the stats log.

The format of the stats log records is described in their particular man pages: AKHP-888(6), DNSP-888(6), FTPP-888(6), HTTP-888(6), ICAP-888(6), IMAP-888(6), MMCG-888(6), MMCP-888(6), PFLG-888(6), POP3-888(6), SIPP-888(6), SMTP-888(6), SQLP-888(6), TCPP-888(6), UDPP-888(6)

Debug log

The debug log contains verbose information, and is intended for thorough investigation the traffic or other troubleshooting. The debug log is (by default) stored in `/var/log/kernun-debug`.

Kernun debug log messages format

Kernun log messages look like the following example:

```
Sep 8 7:40:22 fw ftp-in[2018]:  FTPP-110-E Message text
```

Parts of the message:

Date and time It is a part of the message and is prepared by the application.

Firewall hostname

Application identification It can be either the name of the application, or any string set by the firewall administrator in application configuration.

Process ID If the application needs to distinguish among several "problems" solved by one process (e.g. several requests served by a UDP-based proxy), it uses also a numerical suffix to PID. The ".0" suffix means the "main program", other suffices denote particular request tracks.

Message identification This identification can facilitate post-processing of the log file, as every message has its own identification and all of them have the same form: Kernun component code, Message number, Severity code.

Message text See below.

Kernun component codes

Each library module or application has its own code. This code has either four letters, or three letters as a prefix and fourth one for detailed distinguishing of submodules. A list of the component codes follows:

ADFI - Adaptive Firewall - core
AFHP - Adaptive Firewall - honeypot module
AFLD - Adaptive Firewall - reload tool
AFWD - Adaptive Firewall - watchdog module
ARGS - argument handling
ASN1 - ASN.1 parser utilities
ATRM - ATR monitoring daemon
AUTx - authentication
  H - general authentication library
  R - RADIUS
CASE - configuration integrity check
CFGx - configuration
  L - lists handling
  P - parsing
  R - reader utilities
CHSC - Character set conversion library
CIBR - Configuration definition reader
CKGB - CML Kernun Generation Base
CMLx - CML/KAT tools
  I - command line interface
  K - Kernun Admin Tool
  M - CML main program
  R - reading configuration
  S - showing configuration
  T - tree management
CWBP - Clear Web ByPass
CWCD - Clear Web automatic categorization daemon (cwcatd)
CWDx - Clear Web DataBase

B - database engine
X - configuration checks
DHCP - DHCP server configuration
DHDR - document header handling
DNSx - DNS-proxy
C - nameserver cache management
E - DNS proxy and resolver engine
I - configuration and post-config initialisation
P - proxy itself
R - resource records utilties
X - configuration checks
FTPx - FTP-proxy
P - FTP-proxy, main program
S - FTP-proxy, control connection operations
T - FTP-proxy, data connection operations
H - FTP-proxy, HTTP <-> FTP gateway
H225 - H.323-proxy, H.225 Parser
H245 - H.323-proxy, H.245 Parser
HTCT - HTTP cookie table
HTCW - Clear Web DataBase
HTTx - HTTP-proxy
A - HTTP authentication proxy
F - module mod-ftp-dir
H - HTTP header processing
P - HTTP-proxy main program
X - configuration checks
ICAx - ICAP server
P - ICAP server
B - ICAP server, ICAP BNF parser/printer
R - ICAP server, server request control module
S - ICAP server, main program
IFSC - pikemon, interface status checking
IMAP - IMAP4-proxy
IPCx - IPC facilities
L - locks
M - shared memory
IPSE - IPsec
KERN - Kernun general messages
KEYV - keyword-value handling
LDAP - LDAP authentication
LIBx - general library functions
A - ACL
I - IP utilities

P - process utilities
T - time utilities
U - general utilities
LICC - license checking
LIST - doubly linked lists
LSQL - SQLite library
LSTN - listening sockets management
MAVC - module mod-antivirus
MAVP - module mod-antivirus
MCHU - module mod-chunked
MEMM - memory managemenent
MENC - MIME encoding/decoding utilities
MGZI - module mod-gzip
MHTF - module mod-html-filter
MIME - MIME type utilities
MIMF - module mod-image-filter
MIMX - MIME features configuration checks
MLSN - module mod-listen
MMAT - module mod-match
MMCx - H.323 Multimedia Communication proxy
C - control protocol (H.225, H.245)
D - multimedia data flow
G - gatekeeper proxy
P - proxy itself
R - Registration and Admission Service
Y - RAS Yellow Pages
MMIM - module mod-mime-magic
MNIO - module mod-netio
MNUL - module mod-null
MODM - module management
MONI - runtime monitoring
MPWF - ICAP interface to Proventia Web Filter
MRDF - module mod-read-file
MRWD - module mod-rw-data
MSPA - module mod-antispam
MWRF - module mod-write-file
NATT - NAT utilities
NETx - network library functions
L - network I/O library
S - select handling library
NTIF - network interface and routing utilities
NTLM - NTLM authentication module
NTPC - NTP configuration resolver

OOBA - out-of-band authentication
OSSL - OpenSSL support
OVPN - OpenVPN
PFCD - Packet-filter - configuration daemon
PFLG - Packet-filter - logger
PIKE - PIKE monitoring daemon
PING - PING group monitoring library
POP3 - POP3-proxy
PRXY - proxy configuration support
RCSL - Revision Control System
RDST - get real destination of transparent connection
RGAI - resolver, getting address info
RGHD - resolver gethostbydns() function
RGHT - resolver gethostbyht() function
RSLx - resolver
  C - resolver, name compressing
  I - resolver, initialisation
  M - resolver, DNS making query
  N - resolver, low-level API
  Q - resolver, DNS query formulation
  V - resolver, Kernun top-level routines
RSND - resolver, DNS query sending
SDPB - SIP-proxy, SDP BNF parser
SDPC - SIP-proxy, data channels management
SIPx - SIP-proxy
  B - SIP-proxy, SIP BNF parser
  C - SIP-proxy, control channels management
  M - SIP-proxy, SIP messages management
  P - SIP-proxy, main program
  R - SIP-proxy, SIP requests management
  S - SIP-proxy, SIP sessions management
  Y - SIP-proxy, SIP YP map management
SLOG - system logging itself
SMTx - SMTP-proxy and mail processing proxies
  B - mailing proxies, BNF parser
  C - mailing proxies, configuration
  D - mailing proxies, mail document module
  I - SMTP-proxy, initialization
  N - SMTP-proxy, DSN creation module
  P - SMTP-proxy, main program
  R - SMTP-proxy, client-side (reader)
  S - SMTP-proxy, server-side (sender)
  T - SMTP-proxy, tools

V - SMTP-proxy, client verification
X - mailing proxies, configuration checks
SQLx - SQL*Net proxy
P - proxy itself
S - TNS session layer
T - SQL RPC transport layer
TCPC - TCP client
TCPP - TCP-proxy
TCPS - TCP server
TCPX - TCP-proxy, configuration checks
TEST - configuration tester
UDPP - UDP-proxy
UDPS - UDP server
USBA - script for auto-configuration from an USB device (usb-setup.pl)
URIP - URI parsing and printing

Severity codes and logging levels

Each Kernun message has assigned a severity code, expressed as the last part of identification - a single letter. Every Kernun severity code corresponds to one syslog severity level and has assigned a numeric value:
X - 0,LOG EMERG - system is unusable
A - 1,LOG ALERT - potential security problem detected
C - 2,LOG CRIT - critical error, application fails
E - 3,LOG ERR - error, current connection fails
W - 4,LOG WARNING - potential error
N - 5,LOG NOTICE - normal but noticeable condition
K - 5,LOG KERNUN - Kernun message
(non-maskable LOG NOTICE messages)
I - 6,LOG INFO - statistical message
D - 7,LOG DEBUG - debugging message
T - 8,LOG DEBUG - tracing message
F - 9,LOG DEBUG - full log message

The firewall administrator can decide what level of debugging they desire; the lowest available is 'E' (error messages).

Warning

The full debug level is very exhaustive for the system resources and disc capacity. It is recommended to set this level only when hunting bugs, to direct logging to a `file` (see log(5) for deatils) instead of `syslogd`, and to do so for a single process only and for as short time as possible. The logging level can be increased and decreased by sending the SIGUSR1/SIGUSR2 signals.

Message texts and explanation

Every (non-debugging) log message has its own Section 6 manual page, the name of which is equal to the first and second message identification parts. For example, the above example would correspond to the FTPP-110 manual page. This manual page shows the message text (if the message contains various values, they are substituted by adequate C-printf style directives — %s, %d etc.) and describes the meaning of the message.

There are five types of message texts:

1. Panic messages

   ```
   **PANIC** [ftp-proxy.c:97] ftpadr():  Bad IP type (0)
   ```

   These messages are logged in the case of unexpected internal errors. The program immediately fails in this case. The information in the square brackets (source module name and line) and before the parentheses (function name) locates the error in the source code and is important when reporting such an error to a support technician. The section 6 manual pages contain only the last part of the messages.

2. Errno messages

   ```
   [log.c:97] open():  Permission denied (EACCES=13)
   ```

   These messages are logged when a syscall returns an error state that cannot be reached in a natural way. All messages of this type have KERN-100 log ID and are usually followed by some "high-level" message describing the situation, in which the error occurs and its consequences.

   For instance, when a `write` syscall fails, the application (or library) will log the appropriate errno message and then another message describing what kind of connection has failed. However, if the reason of the failure is "peer has closed connection", no errno message is generated and only the "high-level" message appears. This kinds of error need not necessarily stop the application.

   The information in square brackets (source module name and line) locates the error in the source code. The name before the parentheses is the name of the syscall that has caused the error. The text after colon is the standard "`strerrno`" text, the name in the last parentheses is the proper `errno` constant, the number is the `errno` value.

3. Configuration error messages

   ```
   Line 21, char 1:  Exactly one of DENY and ACCEPT must be
   specified FTP-PROXY.ACL-1:  Exactly one of DENY and ACCEPT must be
   specified
   ```

   These messages are printed when the configuration reading utility or the CML tool finds an error in the syntax or semantics of the configuration. If the configuration is read by a proxy, the printed line and char numbers approximately locate where in the configuration file the error occurs. In the case of verification by the CML tool, the "configuration path" printed points to the place in the configuration where the error occurs. This path can be used as a parameter of the /**SHOW** command.

4. Ordinary log messages

   ```
   closecfg():  Configuration failed, exiting
   ```

   Such a log message is produced in the case of an error or of a normal, but significant condition. The function name (before parentheses) can sometimes be replaced by the '%s' symbol when the same message is produced by several functions.

5. Statistical messages

   ```
   ACL PHASE=2 CLIENT=[127.0.0.1]:2471 SERVER=localhost:21 USER=des
   PARENT=normal NAME=all ACCEPTED
   ```

   The last type of message is statistical information. It is supposed to be automatically post-processed, which is why its format is less human-readable, but stricter. It begins with a keyword describing the message type, followed by couples *keyword=value* and at most one keyword `ACCEPTED` or `REJECTED` at the end of the message.

Log level setting

By default, Kernun applications log messages up to level 5 (LOG NOTICE). The firewall administrator can change the logging level limit to a value between 3 (error messages) and 9 (full debug) by several means. K-level messages are logged in any case and cannot be switched off.

- The configuration-time log level can be set using the `-d` option on the command line. The value of the option can range from 3 to 9, corresponding to the numeric values of severity levels.
- The run-time log level can be set using the `level` item of the `log` section of the particular application configuration file. The logging levels are expressed mnemonically there: `error`, `warning`, `normal`, `debug`, `trace`, and `full`.
- The last way of setting the logging level is to send a running process the SIGUSR1 (to increase level by one) or SIGUSR2 (to decrease it) signal. The logging level cannot get outside the range 3-9.

Variants of log output

By default, Kernun applications log via syslog using the LOCAL4 facility. Alternatively, the log output can be directed to a file. It is possible to specify how to handle the situation when the writing of a log message fails.

Another variant of the logging output is *logging to memory*. It is independent to and can be used simultaneously with syslog/file logging. Even the log level can be set differently for memory logging. The principle of logging to memory is that each process has a fixed circular memory buffer. Log messages are written to the buffer and if the buffer becomes full, the oldest messages are overwritten. The buffer is mapped to a file, hence it can be viewed by any program that displays file contents (e.g., **less**). When a proxy process terminates successfully, its memory log file is deleted. If the process fails, the file is retained. The memory log is not physically written to the disk until the termination of the process and it takes only a fixed amount of space for each

process. It is therefore faster and takes smaller disk space than the normal log. The memory log level can be set to D or T, in order to get detailed records of the last moments of failed processes.

SEE ALSO

log(5), configuration(7)

NAME

monitoring — Kernun firewall runtime monitoring support

DESCRIPTION

In addition to logging, Kernun applications report their status using the runtime monitoring facility.

Monitoring in proxies provides a means for obtaining information about sessions in progress. Such a session has not yet written its final log message (e.g., `SESSION-END`), hence the log cannot be used to get information, such as the amount of data transferred during the session so far or the current communication speed. Proxy monitoring provides information about each active proxy process, i.e., a process serving some client. The output of monitoring includes session duration, client and server IP addresses and ports, size of transferred data received/sent from/to client/server, and the current speed of communication. Some proxies, for example ftp-proxy and http-proxy, provide additional information: user name, file name, or request URI. This additional data has the form of text strings of variable length, which have a fixed (configurable) space reserved in the communication file. Too long strings are truncated. The monitoring utilities provide indication of such truncation.

Applications using remote host monitoring via ICMP ECHO (ping) write data about total and recent ping attempts (i.e. number of sent packets, number of received responses and the round trip time).

The **pikemon** application writes yet another type of monitoring data about its own health status, priority and role and also some data about the cluster peer (priority, role, status and the last HELLO coming from the peer).

Monitoring is currently available in atrmon(8), ftp-proxy(8), http-proxy(8), imap4-proxy(8), pikemon(8), pop3-proxy(8), smtp-proxy(8), sqlnet-proxy(8), and tcp-proxy(8).

An application generates monitoring data into a communication file named `monitor.`*`app-name`*`.`*`pid`* in a directory specified in the configuration. The communication files are processed by the monitor(1) utility, which collects data from several communication files and outputs selected data in textual or HTML formats. Data in communication files are in a binary format that is decoded by an auxiliary program **monitor-dump** called by the **monitor** utility.

The current communication speed is computed from the amount of data processed in last `T` seconds, where `T` can be set in the configuration. The speed is only an approximation, which may differ from the real current bandwidth utilization, especially in the case of a long `T` parameter, short sessions or rapidly changing communication speed. It should be quite accurate during a long, steady data transfer.

SEE ALSO

monitor(1), atrmon(8), ftp-proxy(8), http-proxy(8), imap4-proxy(8), pikemon(8), pop3-proxy(8), smtp-proxy(8), sqlnet-proxy(8), tcp-proxy(8)

NAME

netio — Kernun firewall network I/O layer

DESCRIPTION

All native Kernun Firewall applications use a special mechanism for network operations called *netio*. Its goal is to unify the processing, logging and configuring of network operations. Several attributes of the `netio` library can be set in each application, configured in the same way (see netio(5) for details).

When connecting to remote sites, it is possible to tell *netio* to use other than the default operating system connection timeout (75 seconds on most systems). This is done using the configurable parameter `conn-timeout` (expressed in seconds, default value 75). The `conn-timeout` parameter is relevant only if the socket connection is initiated by the firewall. This is typically the case of server sockets (clients issue their connections to firewall themselves).

For both input and output, *netio* functions may use internal buffers of configurable sizes (`recv-bufsize` and `send-bufsize`, respectively, expressed in bytes). Some applications may use unbuffered input or output. The setting of buffer size is not allowed in these cases. Both buffer sizes default to 16KB.

The *netio* functions use `recv-timeout` when waiting for client or server responses. It is used when the proxy is awating some input from a client or server. Its use is protocol-dependent. On the other hand, `send-timeout` is used when data has been sent to a peer (client or server) and the proxy is waiting for the acknowledgment of that operation. Both values are expressed in seconds and default to 120 seconds.

When closing a TCP connection, *netio* waits for the peer (usually the client or the server) to close its half of the connection. If the peer does not close the connection until `close-timeout` expires, the proxy terminates the connection by TCP reset. This prevents TCP sockets from getting stuck in the `FIN_WAIT_2` state for a long time. The default value is likely to be suitable for most situations. It should be increased only if it is really needed, because waiting for the close blocks a proxy process. If set to zero, the connection will be terminated by reset whenever the proxy closes the connection earlier than the peer.

If full logging is set (see logging(7) for more details), the *netio* functions also log all the data going through. This may lead to very extensive logging and have significant impact on firewall performance and even on its overall behavior. It is therefore possible to limit the number of data octets per block that are logged in the full logging mode. This is done through the configurable parameter `log-limit`, which is set to 80 (expressed in bytes) by default.

All attributes of an application socket are collected in one configuration section, which is called `sock-opt`. Proxies may use several instances of that structure, typically one for each socket.

For example, **tcp-proxy** uses two instances of `sock-opt`:

`client-conn` defines the socket parameters of a connection from a client to the proxy and

`server-conn` defines the socket parameters of a connection from the proxy to a server.

On the other hand, **ftp-proxy** has four such sections:

client-ctrl defines the socket parameters of an FTP control connection from a client to the proxy,

server-ctrl defines the socket parameters of an FTP control connection from the proxy to a server,

client-data defines the socket parameters of an FTP data connection between a client and the proxy (it may be initiated by any of the parties, depending on whether the connection is passive), and

server-data defines the socket parameters of an FTP data connection between the proxy and a server (it may be initiated by any of the parties, depending on whether the connection is passive).

EXAMPLES

The following is a sample excerpt from **tcp-proxy** configuration:

```
client-conn {
    recv-timeout 60;
    recv-bufsize 32768;
    send-timeout 60;
    send-bufsize 32768;
    log-limit 64;
}
server-conn {
    conn-timeout 120;
    recv-timeout 300;
    recv-bufsize 32768;
    send-timeout 600;
    send-bufsize 32768;
    log-limit 64;
}
```

The client connection timeouts are set to lower values, because we know that clients are on our local network. The connection timeout is relevant only for server connections, as client connections are always initiated by clients. We set the same buffer sizes of 32KB.

The following is a sample excerpt from **ftp-proxy** configuration:

```
client-ctrl {
    recv-timeout 30;
    recv-bufsize 2048;
    send-timeout 60;
    send-bufsize 2048;
```

```
    log-limit 2048;
}
server-ctrl {
    conn-timeout 120;
    recv-timeout 300;
    recv-bufsize 2048;
    send-timeout 600;
    send-bufsize 2048;
    log-limit 2048;
}
client-data {
    conn-timeout 15;
    recv-timeout 60;
    recv-bufsize 32768;
    send-timeout 60;
    send-bufsize 32768;
    log-limit 64;
}
server-data {
    conn-timeout 120;
    recv-timeout 300;
    recv-bufsize 32768;
    send-timeout 600;
    send-bufsize 32768;
    log-limit 64;
}
```

Both control and data connections with clients have their timeout values lower. Both client and server control connections have much lower buffer sizes, as there is supposed to be much lower data flow in control connections than within data connections. The client control connection timeout is irrelevant, as these connections are always issued by clients. However, client data connection may be initiated both by the firewall and by clients, depending on the FTP data mode (either PORT, or PASSIVE). Also, we may want to have more data logged in control connections (which are, in fact, commands) and less data in data connections.

SEE ALSO

netio(5), configuration(7), logging(7)

NAME

port-range-listen

— the ability of proxies to listen on a port range and the related aspects

DESCRIPTION

The Kernun firewall proxies are able to listen on a contiguous set of ports, i.e., on a *port range*.

Configuration

Proxies can be configured to listen on a port range in their listen-on(5) section, in both `transparent` and `non-transparent` item, by specifying the optional element `ports`. Both TCP and UDP based proxies may be configured to listen on a port range.

Proxies listening on a port range can be identified in a running system using sockstat(1) as they show the port range instead of a single port number:

```
kernun   sip-proxy  97949 9  tcp4   vr0>>:5060-5062       *:*
kernun   sip-proxy  97949 10 udp4   vr0>>:5060-5062       *:*
```

Limitations

The port range may not intersect the port ranges defined by `sysctl net.inet.ip.portrange`

- `lowfirst-lowlast`
- `first-last`
- `hifirst-hilast`

SEE ALSO

Kernun: listen-on(5), transparency(7)

FreeBSD: sockstat(1), sysctl(8)

NAME

resolving — DNS resolving in Kernun applications

DESCRIPTION

All Kernun components use Kernun own name resolver, which differs from the standard FreeBSD name resolver in some key features. Moreover, the components having only one regular child (e.g. UDP based proxies) use non-blocking model of name/addresses resolving.

BLOCKING RESOLVER

The Kernun library name resolver differs from the standard FreeBSD name resolver in following points:

- It is possible to set the *total* timeout for a query regardless of the number of domain in the search list and the number of servers.
- It is possible to set *different* timeouts for different situations.
- It is possible to set the timeout for the `connect`() call in the case of a TCP query.
- It is possible to set different *port* numbers for different servers.

NON-BLOCKING RESOLVER

In UDP-based proxies there is a problem with online resolving. Since all requests are processed in a single process, the calling of regular blocking resolver routines would increase proxy latency. Thus, UDP-based proxies running in parent/child mode (i.e. not in the `singleproc` mode) start an extra child process ("Asynchro Parallel Resolver", or *APR*) that provides the resolution. This process is, in fact, an instance of the core of the dns-proxy(8) working in the forwarding mode.

The resolver section used by the APR is converted to the `dns-proxy` structures using this schema:

- There is a special "zone" named *`resolver-section-name`*`.APR`. Name of this pseudozone can be found in log messages like DNSE-590-C.
- There is a special "server" named `#`*`server-number`* within the APR pseudozone for every server in resolver section. These server names can be found in log messages like DNSE-740-W.

RESOLVER CONFIGURATION

The key part of resolver configuration is a section named `resolver` (see resolver(5) manual page) that contains following directives:

conf-timeout The timeout for resolution of each domain name used in configuration. The value is given in seconds with decimal values allowed.

For configuration resolution, see Section C below.

conn-timeout Timeout to resolve connection critical addresses. The value is given in seconds with decimal values allowed.

This timeout will be used for any resolution necessary for successful progress of the proxy work, e.g. of a server address.

disable-deresolution The deresolution of (client and server) IP addresses can be suppressed entirely using this item.

final-timeout The timeout used for deresolving a client address immediately before logging the SESSION-END message. The value is given in seconds with decimal values allowed.

When a session closes, the SESSION-END message is to be logged. For this message, another attempt to deresolve the client's address is made (of course, only if the first attempt on client deresolution failed because of reaching the `initial-timeout`).

initial-timeout The timeout for the initial attempt to deresolve a client address. The value is given in seconds with decimal values allowed.

When a client contacts the proxy/server, an attempt to deresolve its address is made. If it fails, the client's address will be logged without a name until the SESSION-END message. In the case of APR (see above) usage, this timeout is ignored.

preference The order of IPv4 and IPv6 addresses in responses can be selected using this item.

search The order of domains added to non qualified domain names for resolving can be selected using this item.

server The list of nameservers being queried can be defined using this item.

There can be more `resolver` sections in the `kernun.cml` and every component can use its own one (being configured by the `use-resolver` item). The same item is used also on the system-level configuration and this resolver section defines the system-wide parameters, i.e. content of the file `/etc/resolv.conf` and parameters for components not using their own resolver section. The behavior of the system name-service switch dispatcher (`nsdispatch`() function) is not changed - Kernun creates the file `/etc/nsswitch.conf` with the content "hosts: files dns" during the installation and does not alter it further.

CONFIGURATION RESOLUTION

All names in the configuration are resolved during the proxy startup. Within this process, each name resolution is tried for `conf-timeout` seconds; if it fails, the name remains unresolved.

If the proxy runs in the parent/child mode (i.e. not the `singleproc` mode), it starts an extra child process ("Asynchro Configuration Resolver", or *ACR*) as soon as new resolution is needed (i.e. some names have expired). This child tries to resolve the expired names again and stores the

result in a memory mapped file shared by all regular children. There are some exceptions to this rule. For instance, the `listen-on` addresses must be resolved immediately at the beginning of the proxy run and they are not refreshed until the end of the execution of the proxy.

Some parameters of the configuration resolution refreshment can be specified by means of the `cfg-resolution` configuration item (see application(5)).

`max-addrs` Every configuration name has a limited number of addresses, to which it can be resolved. Default: 10.

`def-ttl` If the name remains unresolved (either for the negative answer or because of query expiration), this value is used as expiration (and thus also next refresh period) time. Default: 1 min.

`max-ttl` If the name TTL is too high, or the name is resolved using the `/etc/hosts` file (not by DNS), this value is used as expiration (and thus also next refresh period) time. Default: 1 day.

`pool-dir` Parent process, resolving child and regular children use a shared file for exchanging resolution results. The file is named `RESCFG.`*`proxy-name`*`.`*`parent-PID`* and resides in the `pool-dir` directory. Default: `/tmp`.

SEE ALSO

Kernun: application(5), resolver(5), system(5), dns-proxy(8)

FreeBSD: resolv.conf(5), nsswitch.conf(5)

NAME

tcpserver — TCP client connections and process management in proxies

DESCRIPTION

The part of Kernun Firewall called *tcpserver* handles the server side of proxies. It is implemented by the C function `tcpserver()` contained in a library linked to proxies. After a proxy performs the initializion (command line parsing, configuration reading, log opening), it calls `tcpserver()`. Among other parameters, `tcpserver()` gets a callback function for connection handling. The `tcpserver()` function waits for a connection from a client and then calls the callback and passes it the file descriptor of the accepted connection. The callback is supposed to process the connection (it performs the proxy-specific work) and then return to `tcpserver()`. When this happens, `tcpserver()` waits for the next connection.

The `tcpserver()` function also manages multiple processes needed for parallel handling of connections. Moreover, it processes termination and log level change signals.

The management of proxy child processes is performed using pre-forked processes. This concept of process managemenent is used, for example, by the Apache WWW server.

Most TCP process control attributes are contained in the `tcpserver` configuration section (see tcpserver(5) manual page); some, which are common for TCP and UDP proxies, are part of another configuration section, `application` (see application(5) manual page).

Signals

TCP server handles some signals. All signals except `SIGUSR1` and `SIGUSR2` should be always sent to the parent process of a proxy only.

`SIGUSR1` Increase the log level of a child process (or the parent process and all its children, if sent to parent).

`SIGUSR2` Decrease the log level of a child process (or the parent process and all its children if sent to parent).

`SIGHUP` Graceful termination; the proxy does not accept any new connection, waits until all open connections are closed, and terminates.

`SIGTERM`, `SIGINT`, `SIGQUIT` Immediate termination; the proxy closes all connections and terminates immediately.

Single Process Operation

If item `singleproc` is present in the `application` configuration section, the proxy manages all connections using a single process. The algorithm is very simple:

1. Create and bind sockets according to the configuration (see listen-on(5)).
2. Switch credentials according to the configuration (see application(5)).

3. Wait for a connection from a client.
4. Call the proxy-specific connection handling function and pass it the accepted connection.
5. After a successful return from the handling function, go to 3. If the handling function returns an error, exit TCP server.

Parent/Children Operation

If item `singleproc` is not present in the configuration, the parent proxy process forks child processes that handle incoming connections. The parent does not accept any connection; it only monitors the status of child processes, starts new children and/or kills superfluous ones.

Parent algorithm:

1. Create and bind sockets according to the configuration (see listen-on(5)).
2. Switch credentials according to the configuration (see application(5)).
3. Create `init-children` child processes.
4. Count busy children (those processing a connection) and idle ones (those waiting for a connection).
5. If there are less than `min-idle` idle children, try to fork new children to achieve `min-idle`. At most `min-start-rate` children are forked and the total number of child processes never exceeds `max-children`. If there are still not enough idle child processes during the next parent cycle, 2 * `min-start-rate` new children will be forked. Subsequently, the number of forked children is doubled in each following parent cycle, up to the maximum of `max-start-rate` new children per cycle. If `min-idle` is reached, the number of forks per cycle is changed back to `min-start-rate`.
6. If there are more than `max-idle` idle child processes, try to kill some idle children to achieve `max-idle`. At most `kill-rate` children are killed.
7. If `SIGHUP` has been received, wait for all children to terminate and exit.
8. If the parent cycle has been repeated `info-cycle` times, log a statistical message containing the number of forked and killed children.
9. Wait for `parent-cycle` ms and start a new parent cycle (go to 4).

If the creation of a new child process fails because of a lack of system resources, it is repeated up to `fork-retries` times. There is a pause of `fork-wait` ms between every two attempts. If all `fork-retries` are unsuccessful, no new child is started, but the proxy continues its operation (and possibly starts children later, when the system load decreases).

Additionally, the parent process manages a single child process that resolves DNS names from the configuration. This child process is not controlled by the above algorithm and is restarted as required for proper name resolution (see resolving(7)).

Child algorithm:

1. Start listening on all server sockets, as specified by the `listen-on` configuration value.
2. Wait for a connection from a client.
3. Call the proxy-specific connection handling function and pass it the accepted connection.
4. After a successful return from the handling function, go to 2. If the handling function returns an error, terminate the particular child process. The proxy continues running and replaces the terminated child as necessary.

Inter-Process Communication

In order to be able to manage its child processes, the parent process must communicate with them. Two mechanisms are used for this purpose: shared memory and signals. There is a shared memory structure called "scoreboard" containing one slot for each possible child (i.e., `max-children` slots). Each child maintains a flag in its scoreboard slot that indicates whether the child is busy, or idle. The parent reads these flags when counting its children. The parent sends signals to the children in order to kill a superfluous child, perform an immediate or graceful termination, and increase or decrease the log level. As there are not enough signal numbers available, the parent uses `SIGTERM` for immediate termination and `SIGHUP` for all other requests. The type of request is indicated by a value set by the parent in the scoreboard before sending the signal.

Accept Serialization

Doing `select()`/`accept()` by multiple processes in parallel on the same set of sockets causes a problem (see, e.g., Apache WWW server documentation, section "General Performance hints"). If a single connection arrives, all processes are woken up from `select()` and call `accept()`. A single `accept()` succeeds and returns, all the other processes are blocked in `accept()`. However, all processes are waiting for a connection on a single socket now and the remaining sockets are not handled. Therefore, `select()` and `accept()` are placed in a critical section secured by a lock, which ensures that only one process sleeps in `select()` at a time. The lock is implemented using `flock()` on a file specified by a parameter of the `lock` item. For a large number of child processes (many hundreds or thousands), locking via `flock()` may behave incorrectly and block the proxy operation. Therefore, it is possible to use an alternative lock implementation selected by the `alt-lock` item. The following possibilities are available:

`none` No locking is done. Accept is called in the non-blocking mode, in order to solve the above-mentioned problem with processes blocked in an `accept()` function on a single socket.

`semaphore` Locking is done using a System V semaphore.

`lock2` Locking uses a two-level `flock()` locking scheme with locking parts of a single lock file. This is an experimental variant that should not be used, because it exhibits a similar problem with many processes as the standard single `flock()`.

`multilock2` This is the recommended alternative locking mechanism. It uses a two-level `flock()` locking scheme with each lock on a separate file. The set of NxN processes is

divided into N subsets of N processes. Members of each subset share one lock and there is a single global lock. To acquire the lock, a process must first lock the lock file belonging to its subset and then lock the global lock. This algorithm reduces the maximum number of processes waiting on a single lock file.

If either both or none of the `lock` and `alt-lock` items are specified, the standard locking is used if `max-children` is up to 500, and `multilock2` is used for `max-children` of 501 or more.

Note

Experiments indicate that this arrangement is not strictly necessary on FreeBSD, because it seems that if there is a very short time between `select`() and `accept`(), only a single process is woken up from `select`() and calls `accept`(). However, this positive feature is dependent on timing (and thus on such unpredictable conditions as the system load). We have implemented the serialization lock in order to prevent race conditions.

Warning

Be careful when configuring lock-file names for proxies. If two different proxies happen to use the same filename, one of them gets stuck. Such a situation looks rather strange: the TCP handshake takes place, but data exchange does not. As proxy processes are unable to detect this situation, care should be taken.

Process Groups

Caution

If item `nodaemon` without `singleproc` is used in the configuration, i.e., parent/children operation in no-daemon mode, the proxy runs in the same process group as its parent process (if it was not moved to another group before executing the proxy program). The proxy parent process uses `kill(0, sig)` syscall to propagate `SIGTERM` and `SIGHUP` to its children. But the signal is delivered to all processes in the process group of the proxy. Thus, other processes (not belonging to the proxy) in the same group should make appropriate provisions in order not to be disturbed by these signals.

SEE ALSO

listen-on(5), application(5), tcpserver(5), resolving(7)

NAME

time-matching — syntax and semantics of time specification in configuration

DESCRIPTION

Time specifications may appear in various modifications of `acl` sections of the Kernun configuration (see access-control(7), configuration(7) manual pages). There are two independent mutually exclusive ways how to define proper time interval.

REPEATABLE ITEM **`TIME`**

`time [day` *`days`*`] [month` *`months`*`] [wdays [times]]`

The semantics of individual elements is as follows:

`day` ***`days`*** It stands for days of month and its type is `UINT8-SET`.

`month` ***`months`*** This element is a set of months (again `UINT8-SET`).

`wdays` This parameter is a set of special enumeration type for days of the week, where standard English three-letter shortcuts may be given (Sun, Mon, Tue, Wed, Thu, Fri, Sat). Their numerical counterparts can be given as well, beginning with 0 for Sunday through 1 for Monday, 2 for Tuesday etc. ending with 6 for Saturday.

`times` This element is of type `TIME-SET` and specifies hours within a day, such as { 0800-1630 } which means 08:00:00 to 16:29:59.

Time conditions specified in the same `time` item are linked with logical AND. Unused conditions are not checked. Repeated occurrences of the `time` item within one acl are linked with logical OR.

NONREPEATABLE SECTION **`TIME-PERIOD-SET`**

```
time-period-set {
  [exclude;]
  time-spec NAME {
    [dates from-day from-mon till-day till-mon;]
    [weekdays from till;]
    [hours from till;]
  }
  ...
}
```

The semantics of individual items is as follows:

dates *from-day from-mon till-day till-mon*; Specification of date interval within a year. Lower bound can represent higher date then upper one in which case the interval goes across the new year. Upper bound is included in the range.

weekdays *from till*; Specification of day interval within a week. Day abbreviations of numbers can be used. Lower bound can represent later day then upper one in which case the interval goes across Sunday/Monday. Upper bound is included in the range.

hours *from till*; Specification of time within a day. The HHMM time format is used for values. Lower bound can represent higer time then upper one in which case the interval goes across midnight. Upper bound is NOT included in the range.

Time specifications of the same type are linked with logical OR. Time specifications of different types are linked with logical AND. Unused conditions are not checked. The item `exclude` forces complementary specification, if the current time matches all given conditions, it is not accepted, and vice versa.

EXAMPLES

The following time specification is satisfied every Monday in February and October between 8 a.m. and 1 p.m. as well as every Wednesday, Thursday and Friday in July between 10 p.m. and midnight:

```
time month { 2, 10 } { 1 } { 0800-1300 };
time month { 7 } { Wed - Fri } { 2200-2400 };
```

Below, there is another time specification example that stands for the first ten days of each month between 1 p.m. and 7 p.m.:

```
time day { 1-10 } { 1300-1900 }
```

The last example shows the Czech schoolyear:

```
time-period-set {
  time-spec SCHOOLYEAR {
    dates 1 9 30 6;
    weekdays Mon Fri;
    hours 0800 1400;
  }
}
```

SEE ALSO

acl(5), access-control(7), configuration(7)

NAME

traffic-shaping — Kernun firewall traffic shaping support

DESCRIPTION

Kernun firewall supports traffic shaping. The PF/ALTQ package is used for this purpose, see pf(4), altq(4), pfctl(8), pf.conf(5). ALTQ sends outgoing network packets to queues. Each queue has its own bandwidth parameters that define how it sends packets to the network interface. Packets are assigned to queues by pf filter rules. Queues and filter rules are defined in pf configuration pf.conf(5).

All Kernun proxies support an alternative and more flexible way of assigning traffic to queues. Queue names can be set in ACLs. All communication that matches a particular ACL will use the queues specified in the ACL. This way, the queue selection process may utilize all conditions available for ACL matching, including time-based (slow download speed from some servers during working hours, remove limits in the night), or user-based limits (some users have priority). Queues for a session are configured separately for each communication direction, i.e., sending data to a client, server, antivirus, and antispam modules.

Notes

- ALTQ works only for outgoing traffic. Received packets are not subject to traffic shaping.
- ALTQ queue specifications in proxy ACLs require a kernel patch in order to work. Patching system sources and building a patched kernel is done as a part of the Kernun installation process.
- Each UDP queue specification consists of a single queue name. TCP requires one or two queues. If two names are given, the second one is used only for prioritized packets, i.e., TCP acknowledgements without payload. Everything else is sent via the first queue.
- In order to use ALTQ on a network interface, the network interface card driver must support it. For the list of supported drivers, see altq(4).

SEE ALSO

FreeBSD: altq(4), pf(4), pf.conf(5), pfctl(8)

NAME

transparency — network transparency and the related aspects

DESCRIPTION

The Kernun firewall is able to transparently grab clients' connections and hand them over to its proxies, as well as to pretend to servers that connections come from clients' real IP addresses instead of the firewall's IP address. This ability is implemented in the transparency support that was added to the FreeBSD kernel.

Transparency for clients

Transparency for clients is realized using special sockets called *transparent listening sockets*. Unlike a regular listening socket, a transparent listening socket is able to accept transparent connections, i.e., it accepts a connection even if the client is not connecting explicitly to the firewall, but directly to some server's IP address.

A transparent listening connection can be configured either to accept connections that arrive to any interface, or it can be limited to a particular interface. The former case is configured in the Kernun configuration by specifying a special IP address, `0.0.0.0`. The latter one can be configured either by specifying the name of the interface, or by specifying its IP address (which is only used to determine the particular interface). See listen-on(5) for the proper syntax.

Transparent listening sockets can be identified in a running system by sockstat(1). They are distinguished from regular sockets by the special syntax in the `LOCAL ADDRESS` field of the sockstat(1): `{`*`iface`*`|*}>>:`*`port`*. If present, the *iface* value denotes the interface the transparent listening socket is limited to. Otherwise (denoted by *), the socket listens on all network interfaces.

In the TCP protocol, accepting connection by accept(2) returns a new socket that is used for communication with the client. Packets sent via this socket are automatically assigned the real destination (the server's) IP address as the source address, and all packets sent within this connection from the client to the server would come to this socket. These sockets are indicated by syntax `>>`*`addr`*`:`*`port`* in the `LOCAL ADDRESS` field of the sockstat(1). Here, the *addr* denotes the IP address of the server (i.e., the IP address the client thinks to be connected to).

In the typical scenario there would be one transparent listening socket for each interface the proxy listens on (`FD` 5 in the following example). This socket is shared by the proxy's parent process and by all of its child processes. In addition, there would be two sockets for each connection established via the proxy: one for the connection from the client to the proxy (`FD` 11), and another for the connection from the proxy to the server (`FD` 12). The situation is shown in the following example (`10.1.1.1` stands for the client IP address, `10.3.3.3` stands for the firewall external address and `10.4.4.4` stands for the server IP address):

```
USER   COMMAND   PID   FD PROTO LOCAL ADDRESS  FOREIGN ADDRESS
kernun tcp-proxy 26802 5  tcp4  vr0>>:22       *:*
kernun tcp-proxy 26801 5  tcp4  vr0>>:22       *:*
```

```
kernun tcp-proxy 26800 5  tcp4  vr0>>:22       *:*
kernun tcp-proxy 26800 11 tcp4  >>10.4.4.4:22  10.1.1.1:36528
kernun tcp-proxy 26800 12 tcp4  10.3.3.3:62175 10.4.4.4:22
kernun tcp-proxy 26799 5  tcp4  vr0>>:22       *:*
kernun tcp-proxy 26798 5  tcp4  vr0>>:22       *:*
kernun tcp-proxy 26797 5  tcp4  vr0>>:22       *:*
```

Matching transparency

Connections can be considered in ACLs according to their transparency. This is done using a keyword to the `to` configuration item. If the key `transparent` is present in this item, only transparent connections get matched. Similarly, with keyword `non-transparent` present, only non-transparent connections get matched.

Changing source address

This feature allows servers to see real clients' addresses upon receiving connections instead of the firewall's address (which is the standard behavior for proxies). It can be specified in an ACL section using either of the following syntax constructions:

```
source-address client;
source-address [5.5.5.5];
```

This feature can be regarded as transparency for servers. `source-address` might be used either in `transparent` or `non-transparent` mode.

For example, if a client of `10.1.1.1` wants to connect through proxy at `10.2.2.2` (either transparently or non-transparently) to server `4.4.4.4`, the server sees normally the connection as coming from firewall's external address, e.g. `3.3.3.3`. Taking advantage of `source-address client`, the server sees the connection as if it were coming from `10.1.1.1`. In that case, the sockstat(1) gives the following output:

```
USER    COMMAND    PID   FD PROTO LOCAL ADDRESS  FOREIGN ADDRESS
...
kernun tcp-proxy 23008 11 tcp4  >>10.4.4.4:22  10.1.1.1:36528
kernun tcp-proxy 23008 12 tcp4  10.1.1.1:62175 10.4.4.4:22
...
```

Unlike in the first example, the connection from the firewall to the server (`FD`=12) shows the `LOCAL ADDRESS` to be `10.1.1.1` (i.e., the client's IP address).

Socket conflicts

For a given TCP/UDP port there can be more than one type of application listening side-by-side: transparent proxies, non-transparent proxies or system daemons (such as ssh daemon **sshd**). However, they must not be in mutual conflict.

Conflicts are detected by the /**verify** command of cml(8). Two applications that listen on the same port (or with their listen port ranges overlapping) are in conflict, if any of the following cases occurs:

1. Both listen in the non-transparent mode on the same IP address
2. Both listen in the non-transparent mode on the wildcard IP address `0.0.0.0`
3. Both listen transparently on the same interface (either if the interface name was given directly or if it was deduced from the IP address)
4. Both listen transparently without interface restriction

Note that the conflicts check is only performed for the components configured within the cml configuration file. It is not performed for the components that are configured out of it.

Socket precedence

When a packet arrives, the most specific socket is chosen according to the following precedence order (from the most specific to the most generic):

1. Non transparent, single IP address, single port
2. Non transparent, wildcard address (`0.0.0.0`), single port
3. Transparent with interface restriction, single port
4. Transparent without interface restriction, single port
5. Non transparent, single IP address, port range
6. Non transparent, wildcard address (`0.0.0.0`), port range
7. Transparent with interface restriction, port range
8. Transparent without interface restriction, port range

It is therefore possible to provide several services for the same port, as long as they do not collide. For example, the SSH daemon might be available for an administrative connection to the firewall on all interfaces (SSH daemon only makes sense in the non-transparent mode), while `tcp-proxy` might be configured in the transparent mode for proxying the ssh traffic through the server for the internal interface. In that case, packets with the destination address equal to any of the firewall's IP addresses would end up in the SSH daemon, while packets with the destination in the external network would be processed by the tcp-proxy. See the examples section.

Bypassing transparency

When transparency is enabled (`sysctl net.inet.ip.transparency=1`), every packet is considered to be potentially local (i.e., destinated for some firewall's process) and is therefore delivered into the local IP stack. One of the consequences of this fact is that the packet is not eventually ip-forwarded (even if `sysctl net.inet.ip.forwarding=1` and it would have been forwarded, if the standard FreeBSD kernel had been used).

Under certain circumstances it might be desirable to bypass transparency. Kernun firewall uses the packet filter (pf(4), packet-filter(5)) for this purpose. When the packet has the pf tag `NOTRANSP` set, the kernel handles it the same way the regular FreeBSD kernel would. The actual tag that is used for this purpose can be changed using sysctl `net.inet.ip.no_transp_tag`. NOTRANSP is the default value.

The following rule can be used to tag all the traffic from client of `10.1.1.1`:

```
pass in on vr0 from 10.1.1.1 to any tag NOTRANSP
```

Note that all the traffic that is required not to undergo the transparency must be tagged `NOTRANSP`. Especially, for bidirectional communication, packets for both directions must be tagged so (see the latter example in this manual page).

SEE ALSO

Kernun: acl(5), ftp-proxy.cfg(5), listen-on(5), access-control(7), dns-proxy(8), ftp-proxy(8), kat(8), smtp-proxy(8), tcp-proxy(8)

FreeBSD: ioctl(2), pf(4), pf.conf(5), pfctl(8)

NAME

udpserver — UDP session and process management in proxies

DESCRIPTION

UDP-based proxies in Kernun (e.g., **udp-proxy**) use a special library module called *udpserver* for the process and operation control.

Process control

There are two modes of udpserver operation. In the normal operational mode (parent/child mode), the proxy consists of a parent and up to three child processes.

parent process The parent process monitors the children, restarts them if they terminate unexpectedly, and terminates them if proxy termination is requested by the administrator (via the **kat** utility).

regular process (REG) The REG child process handles the "real work" of the proxy, that is, communication with clients and servers.

configuration resolver process (ACR) The ACR child process tries to resolve all domain names from configuration (see resolving(7)). It starts at the proxy beginning and reschedules itself so that expired names can be resolved again.

run-time resolver process (APR) The APR child process is used to process on-line resolving for the REG process so that it needs not to wait for nameservers' responses (see resolving(7)).

In the single-process mode (turned on by item `singleproc` in the configuration), the proxy consists of a single process. The configuration resolution is not refreshed, the proxy waits for online resolutions. This mode is intended for debugging purposes only.

Signals

The udpserver handles the following signals:

`SIGINFO` Log process-specific information. This signal must be sent to a particular process and it is valid for the process only.

`SIGUSR1` Increase the log level. This signal can be sent either to a child process (valid for the child only), or to the parent (to be resent to all its children).

`SIGUSR2` Decrease the log level. This signal can be sent either to a child process (valid for the child only), or to the parent (to be resent to all its children).

`SIGWINCH` Reopen the log file. If the proxy logs to a file, this signal forces it to reopen the log file (e.g. after the log file rotation). This signal can be exceptionally sent to a child process, but under the normal circumstances it should be sent to the parent (to be resent to all its children).

SIGHUP, **SIGTERM**, **SIGINT**, **SIGQUIT** Immediate termination. The proxy immediately terminates all sessions and exits. This signal can be exceptionally sent to a child process, but under the normal circumstances it should be sent to the parent (to be resent to all its children).

Operation control

The udpserver receives incoming UDP datagrams, assigns them to proper active sessions (or creates new ones), controls time and data limitations for session termination etc. In the case of specific proxies (such as **gk-proxy**), udpserver calls proxy-specific functions for further processing.

SEE ALSO

listen-on(5), application(5), udpserver(5), resolving(7)

Appendix D

NAME

af-db.sh — Adaptive Firewall database management tool

SYNOPSIS

af-db.sh [-h] [-f *db-file*] [-v] [*table*|*db*] *command* [*parameters*]

DESCRIPTION

Utility **af-db.sh** provides a user interface to the blacklist databases of the Adaptive Firewall (see adaptive-firewall(7) manual page) module.

For the list of tables, see the adaptive-firewall(5) manual page, or run the script with the -h option.

Options

The script options are as follows:

-f Use given filename. By default, the standard filename is used, according to the table name selected.

-h Print usage and exit.

-v Be more verbose; print also all SQL commands being executed.

COMMANDS

feed [*address...*] Exports data from IDS databases, makes a new IPS database.

- Addresses in the parameter are temporary whitelisted, i.e. they will not be included in the IPS database.

***db* remove [-y]** Removes an Adaptive Firewall database.

- If the -y option is used, the script does not query to confirm the removal.

***db* list** Prints list of tables in given database file.

***db* find { *IP-address* | *regular-expression* }** Finds all occurences of an IP address or an IP address pattern in all tables of given database.

***table* show [-uR** [-tc] [-r] [-n *num*] [-fF *flag*]] Displays content of a database table. By default, it sorts the output by IP addresses.

The output format for all tables begins by four columns (ADDRESS, FLAGS, COUNT, LAST). For the SSHD table, there is another colmun at the end of line showing the difference (in seconds) between the LAST occurence and the occurence number num given by the -n option.

- If the `-c` option is used, the items are sorted by number of occurences.
- If the `-f` option is used, only the items having given flag set are printed.
- If the `-F` option is used, only the items having given flag unset are printed.
- If the `-r` option is used, the items are sorted in reverted (descending) order.
- If the `-R` option is used, the items are printed in raw format (no formatting).
- If the `-t` option is used, the items are sorted by time of last occurence.
- If the `-u` option is used, the times are shown in UTC instead of local time.

table **add [*flag*** *IP-address* { *+time-offset* | *−time-offset* | 0 }...]

> **Warning**
>
> SSHD table version...

Adds given client to the table with any number of recent occurences set as current time plus/minus given offset(s) and flag set accordingly.

table **add [*flag*** { *+time-offset* | *−time-offset* | 0 } *IP-address*...]

> **Warning**
>
> Non-SSHD table version...

Adds given clients to the table with last time set as current time plus/minus given offset and flag set accordingly.

table **del *IP-address...*** Deletes given clients from the database.

table **find { *IP-address* | *regular-expression* }** Finds all occurences of an IP address or an IP address pattern in given DB table.

table **flush** Removes the whole content of given DB table.

SEE ALSO

Kernun: adaptive-firewall(7)

NAME

alertd — The SNMP trap sending daemon

SYNOPSIS

alertd [-hv] [-d *dbglev*] -f *cfgfile*

DESCRIPTION

The alertd daemon is a component waiting for messages sent via special UNIX sockets (`security.alert`, `security.notify`, `system.alert` and `system.notify`) in the `/var/run/alertd` directory and send them as a SNMP trap to a set of SNMP managers.

The daemon runs in fact as three processes, like Kernun proxies do. The main process just controls run of its children. The Asynchronous Configuration Resolver provides for DNS resolution refreshing. The regular child process handles the real operation.

SIGNALS

The **alertd** daemon handles following signals:

SIGUSR1 Log level increasing.

SIGUSR2 Log level decreasing.

SIGINFO Operation status logging; parent process logs info about all children, child process logs current status of all ping groups.

SIGHUP, **SIGINT**, **SIGQUIT**, **SIGTERM** Immediate termination; the daemon immediately closes the service.

OPTIONS

-h Print usage information.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

SEE ALSO

Kernun: application(5), alertd.cfg(5), configuration(7), logging(7), resolving(7), kat(8)

NAME

atrmon — Adaptive Traffic Routing monitoring daemon

SYNOPSIS

atrmon [-hv] [-d *dbglev*] -f *cfgfile*

DESCRIPTION

The Adaptive Traffic Routing is a component allowing to delegate a special zone (or zones) nameserver to itself and respond to client requests according to the current accessibility of particular hosts.

The ATR monitor sends ICMP ECHO messages to all configured targets (`ping` groups). From every group, at least one response must be received within configured timeout to keep the group alive. If all groups are alive, particular address is included into the set of addresses being sent as a response to DNS query. Otherwise it is excluded from the set. Currently, only IPv4 targets can be tested.

If there is more than one live address, the ATR monitor selects the answer according to the `strategy` item setting. It can send `all` live addresses, or select only one according to the loadbalancing strategy. If there is no live address, the ATR monitor behaves according to the `fallback` item setting.

The daemon runs in fact as three processes, like Kernun proxies do. The main process just controls run of its children. The Asynchronous Configuration Resolver provides for DNS resolution refreshing. The regular child process handles the real operation.

The current status of pinging to the target hosts can be watched by the monitor(1) tool avaliable also as a command of the kat(8) tool.

EXAMPLE

We create a special subdomain for loadbalancing, define the nameserver for it and alias a host to a name in this special domain:

```
lb.tns.cz.  3600  IN  NS     atr.tns.cz.
www.tns.cz. 3600  IN  CNAME  www.lb.tns.cz.
```

Then we run the ATR on the host atr.tns.cz with configuration like this:

```
atrmon ATR {
  listen-on {
```

```
      non-transparent atr.tns.cz;
    }
    session-acl ALL {
      accept;
    }
    request-acl DOMAIN {
      name lb.tns.cz;
      accept;
      nameserver 3600 atr.tns.cz;
    }
    request-acl WWW {
      name www.lb.tns.cz;
      accept;
      address BNS {
        data 10 [1.1.1.1];
        ping { [1.1.1.1] } 5;
      }
      address PHA {
        data 10 [2.2.2.2];
        ping { [2.2.2.2] } 5;
      }
    }
  }
```

SIGNALS

The **atrmon** daemon handles following signals:

SIGUSR1 Log level increasing.

SIGUSR2 Log level decreasing.

SIGINFO Operation status logging; parent process logs info about all children, child process logs current status of all ping groups.

SIGHUP, **SIGINT**, **SIGQUIT**, **SIGTERM** Immediate termination; the daemon immediately closes the service.

OPTIONS

-h Print usage information.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

SEE ALSO

Kernun: monitor(1), application(5), atrmon.cfg(5), configuration(7), logging(7), resolving(7), kat(8)

NAME

bootmgr — tool for configuring Kernun boot manager

SYNOPSIS

bootmgr [-x] [-d { 1| 2| 3 }] [-u| -n] [-1 *label*| +| -] [-2 *label*| +| -] [-3 *label*| +| -] [*dev*]

DESCRIPTION

Utility **bootmgr** can be used to view or change configuration of the Kernun boot manager. If invoked without command line parameters, the boot manager configuration is displayed in a text format. If only option -x is specified, the boot manager configuration is displayed in XML. Other command line options set various parameters of the boot manager.

The boot manager displays a boot menu that allows to boot one from up to three installations (in the first three disk slices) of the Kernun firewall. For security reasons, the administrator can disable some menu choices. There is always a default selection, that is booted after a timeout if no key is pressed. It is configurable whether selection of a non-default choice changes the default for next reboots or not.

The Kernun boot manager is a slightly modified FreeBSD boot manager. It is installed by the Kernun installation process into the MBR (Master Boot Record) of the system disk. It can be reinstalled by boot0cfg(8), but this is almost never needed. The main difference from the FreeBSD loader is the boot menu. Instead of slice types, it shows changeable labels of individual Kernun installation. There are two variants of the boot manager `boot0` and `boot0ext`, stored in directory `/usr/local/kernun/lib`. The preferred variant is `boot0ext`, which is two sectors (1024 B) long. It has space for 74 character labels. By default, a label contains a Kernun version identification, including a full build number, and the date and time of the installation. The smaller, one sector (512 B) variant `boot0` has space for ony 9 character labels. By default a label contains date and time of the Kernun installation.

Options

-x Displays boot manager configuration in XML.

-d *{1 | 2 | 3}* the default slice to boot from

-u Selection from the boot menu will update the default choice.

-n Selection from the boot menu will leave the default choice unchanged.

-1 *label*

-2 *label*

-3 *label* Changes a label of a menu item.

-1 +

-2 +

-3 + Enables a menu item and booting from the corresponding slice.

-1 -

-2 -

-3 - Disables a menu item and booting from the corresponding slice. Use with caution, as disabling all working Kernun installations will render the system unable to boot. In such situation, the boot manager may be reconfigured using the installation medium.

dev Operates on the boot manager located on this disk device. If not used, the disk containing the root file system is selected.

FILES

/usr/local/kernun/lib/boot0 The one sector boot manager; deprecated, because it has only 9 characters for each boot menu item.

/usr/local/kernun/lib/boot0ext The two sectors boot manager; preferred, with 74 characters for each boo menu item.

SEE ALSO

boot0cfg(8), boot(8), fdisk(8)

NAME

cml — Configuration Meta-Language

SYNOPSIS

cml [-hv]

cml [-d *dbglev*] [-f *cfgfile*] [-k] [-r *revision*] [-R]

cml [-d *dbglev*] -f *cfgfile* [-k] -g

cml [-d *dbglev*] [-f *cfgfile*] [-lLicsu]

DESCRIPTION

The abbreviation CML denotes both the Kernun Firewall configuration language and the command-line tool for editing and verifying the configuration and for generating proper files used by the system. The tool can be used in three modes:

- When option -g is not given, the CML will start an interactive mode and prompts user for commands (see Section D below).
- When option -g is given, the CML run is intended to generate target configuration tree only (see option explanation below).
- When one of options -l, -L, -i, -c, -s, -u is given, the CML performs just the requested RCS file locking operation (see option explanation below).

Options

The CML options are as follows:

-h Print usage information and exit.

-v Display version information and exit.

-d *dbglev* Set debugging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is needed for debugging until the CML prompt is ready. Then, more flexible /dbg command (see Section D below) can be used.

-f *cfgfile* As the first operation, the CML will load named configuration file. If not used, the CML will start with an empty configuration. The *cfgfile* can be then later loaded by the /**load *cfgfile*** command (see Section D below).

-g The CML does not enter the interactive mode, just loads the configuration file (option -f must be used, too) and generates configuration files tree (see /generate command in Section D below).

-r *revision* If the CML is invoked with the `-f` option, the specified *revision* saved in the RCS file *cfgfile*`,v` is used instead of the current configuration file. For more details see the /**load** command description in Section D below.

-R The CML runs in read-only mode (this is default, when started under AUDITOR user).

-k The CML does not unlock the configuration file when exiting. The file is kept locked for current user.

-l The CML tries to lock particular RCS file. If the file is not free, or the plaintext version differs from the RCS copy, the program will fail.

-L The CML tries to break the lock of particular RCS file.

> **Warning**
>
> Use this operation with care!

-i The CML tries to create particular RCS file. If the file is exists, the program will fail.

-c The CML tries to check whether the current user owns the lock of particular RCS file. If not, the program will fail.

-s The CML tries to save (check-in) current plaintext into particular RCS file. If the file is not locked for current user, the program will fail.

-u The CML tries to unlock particular RCS file.

CML Operation

After start, the CML checks whether running from a terminal. If so, it prepares the `editline` environment (see editline(3)) and Command Completion and Context Help support (see Section D below). Otherwise, the CML will read commands from standard input.

If the `-f` option was used, the CML loads configuration from given file like if it would be ordered by the /**load** command (see Section D below). If not, no configuration is read, the user must either use the /**load** command or create a new configuration from scratch.

Finally, the CML looks whether the `-g` option was used and if so, it generates output files using /**generate** command (see Section D below) and exits. Otherwise the CML starts normal command processing.

The CML tool uses standard Kernun logging library for displaying messages (see logging(7)), the messages are written both to the standard error output (i.e. sent to the terminal, typically) and to the system log (as configured in `/etc/syslog.conf` file). This behavior can be changed by setting the environment variable KERNUN_LOG_FILE to a file name willing to be the log target. As usual, every message (produced by the CML, not by other system programs called by the CML) has a *log-id* prefix (e.g. `CMLR-710-W`) that can be found in Kernun section 6 manual pages (CMLR-710(6) in above example).

If the CML runs in read-only mode (due to current user type, or the `-R` option), it allows only commands they do not alter the configuration content. Other commands are disabled.

CML COMMANDS

The maximal length of the command line depends on current system settings of the editline(3) library. However, for the purpose of configuration editing is sufficient. The Section D may work slightly fuzzy on too long lines.

There are three essential types of commands in the CML:

Low-level configuration directives Most of Kernun configuration is based on directives known for both the CML and the idividual proxies. They have the same syntax and semantics as described in section 5 manual pages, root of them is kernun.cml(5). These directives are written in the Kernun configuration file and the CML copy them into individual proxies configuration files when preparing the output. Example:

```
system BILOVICE {
  hostname bns;
  admin root@tns.cz;
}
```

High-level configuration directives There are a group of directives having the sense only in the CML. They are written in the Kernun configuration file, but they are only used by the CML when generating the output files which they are not a part of. Example:

```
include "samples/root-servers.cml";
set name = "gateway";
for x in { INT, OUT } {
  ...
}
```

The CML control commands These commands are used for configuration manipulation (editing, saving etc.) and therefore they are not part of saved configuration files. Hence, they also have a different syntactical form: they are prefixed by a slash ('/') character or optionally dot-slash ('./') pair. Examples:

```
/load kernun.cml
/cut
./paste NEWNAME
```

All commands, configuration directives and names are case-insensitive.

Low-level configuration directives

Although general syntax of configuration files is free in the sense of whitespaces and newlines using, the CML uses strict form of line breaks, indentation etc. The reasons for it are both version difference checking and easier Command Completion and Context Help support (see Section D below).

For the detail description of low-level syntax, see configuration(7). For clarity, we mention here only a few basic facts. The configuration consists of configuration directives of two types:

Item is a series of elements (or "attributes") of simple values (integers, names, strings, IP addresses and also comma separated lists of previous types enclosed in curly braces ('`{ ... }`') terminated by a semicolon ('`;`') character.

Section is a set of configuration directives (items and subsections) enclosed in a pair of curly braces ('`{ ... }`').

Both items and sections are either repeatable or non-repeatable. Repeatable sections must have a name specified between the keyword and the opening brace. The name must be at most 63 characters long, otherwise it is not recognized by parser and it is treated as string. For brevity we use the term *name* in the natural sense, in the case of repeatable section we distinguish between *type name* and *name*. Example:

```
routes {           # non-repeatable section of type routes
  static DMZ {     # repeatable section of type static,
                   # section name is DMZ
    flags { cloning, iface };  # item with one element (list)
  }                # end of section DMZ
}                  # end of section routes
```

Comment lines start with the hash ('#') character and can be inserted freely between configuration directives. Lines with a single hash character will be copied to the low-level (proxy) configuration files. If you want to make a comment hidden for the target files, simply double the hash character on the beginning of the comment. Example:

```
# This comment will be generated
## ...while this not
```

If you type a configuration directive on the command line, the behavior depends on directive type and current configuration content.

- If the directive is non-existing (i.e. not specified yet) non-repeatable section/item, then a new one is entered.
- If the directive is existing (i.e. already specified before) non-repeatable section/item, then the old one is modified.

- If the directive is repeatable section and its name has been already used, then the old one is modified.
- If the directive is repeatable section and its name has not been used yet, then a new one is entered.
- If the directive is repeatable item, then a new one is entered.

Repeatable section names must be unique within current section. On the contrary to the low-level configuration files, in the CML, this is true even for sections of different types. For instance, you cannot use the same name for `doc-acl` and `command-acl` in the **ftp-proxy** configuration.

When entering a configuration directive, the CML tries to check correctness of it by so called *on-line verification*. In fact, it tries to parse the directive by an ordinary low-level parser, the same one as will parse it later on within low-level configuration files generated for proxies. This can recover some errors in very early stage of configuration creation.

High-level configuration directives

The high-level configuration handles primarily variables and loops.

Variables

The most important value added in high-level configuration is the concept of variables. We recognize two kinds of variables:

data variables can contain one "simple" value, i.e. one element of item (examples see below)

section variables can contain several directives (items, subsections, comments, etc.) of some section.

Variable names are at most 32 characters long, they can consist from alphanumeric characters and the hyphen ('-') characters. the first character must be a letter.

When applied, variable name prefixed by a dollar ('$') character is used.

Variable is visible from the place where defined, on the level where defined and on all lower levels. Within visibility range, no other variable can have the same name. Example:

```
set X = 1;
system FIRST {
  set X = 2;      # false - repeated definition
  set Y = $X;     # correct
  set Z = $W;     # false - variable $W unknown
}
set W = 3;
system SECOND {
  set Z = $W;     # correct
}
```

In-line files

An interesting feature of the high-level configuration is the list outsourcing. If you have an external source for some list (e.g. list of "valid" email addresses), you can define a global `shared-file` section for the file and instead of writing the whole list directly into CML (using the CML syntax), you will use only a special construct "< *shared-file-name*". Example:

```
shared-file ADDRESSES {
  path "/var/adm/emails";
  format text;
}
...
system SYS {
  smtp-proxy SMTP {
    delivery-acl ACL-2 {
      recipient ok < ADDRESSES;
```

The `format` item defines handling modes for lines from the source file:

`text` (default) Whitespaces form the begining and the end of lines are stripped, rest of the line is double-quoted (like regular strings in the CML). Empty lines and lines starting by the # character are ignored.

`raw` No special handling, the lines are just double-quoted.

`native` Lines should contain valid low-level syntax values, they are used as-is, just adding separating commas. Empty lines and lines starting by the # character are ignored.

`ip-addr` Lines should contain IP addresses with or without mask (in slash-notation), but without brackets. Empty lines and lines starting by the # character are ignored.

`regexp` Lines should contain regular expressions without delimiters (slashes). Empty lines and lines starting by the # character are ignored.

High-level directives desription

The CML recognizes following high-level directives:

include "*`filename`*"; Import variable definitions from another file. This feature was added to facilitate some standard settings, like set of root nameservers or general crontab setting. The aim of it is not to split configuration to several files and so the CML does not support for editing the included files.

Including is allowed only on the highest level of configuration and included files can contain only comments and variable definitions. The *filename* is either absolute, or relative to the "configuration root path". This path is set to `/usr/local/kernun/conf`, by default, and can be changed by the /**cfgpath** command (see Section D below).

In some cases, parts of the include file can be "parametrized" by variables intended to be set in the main configuration file (not within the file). Such variables must be declared within the include file using the `param` directive.

set *varname = value;* Define data variable. For the *value*, any type (including LISTs and SETs) can be used. Besides, for string variables, also concatenation is allowed, concatenation operator is a plus ('+') character. Examples:

```
set ADMIN = root;
set DOMAIN = xyz.com;
set EMAIL = $ADMIN + @ + $DOMAIN;
set MY-ADDR = [1.2.3.4];
set DNS-ADDR = $MY-ADDR : 53;
set IFACES = { INTERNAL, EXTERNAL };
```

set *varname* [*section.path*] { *section variable body* }

set *varname prototype.path* { *section variable body* } Define section variable (macro). The "value" of the variable is a set of configuration directives. Such a variable can be primarily applied as a part of regular configuration section. For this reason, the *context* of the variable must be known, i.e.

1. point of the configuration tree, where it can be applied
2. set of subsections and items it can contain.

This context is defined by the "path" between the variable name and the opening brace and one of the following modes can be used:

configuration context This mode means that variable context is derived from current point of configuration. If the current section is to be the context, the path should be omitted. If a sub-node of the current node is to be the context, the (type) name of the node should be written. If a distant descendant is to be the context, a "path" should be constructed by particular names of section along to the path from current point to the node desired, separating them by dots ('.'). Examples:

```
set SYSTEM-DATA system { # applicable within any system
  DOMAIN xyz.com;
}
set IN-OUT system.acl {  # applicable within system.acl
  from [10.0.0.0/8];
  accept;
}
system BILOVICE {
  $SYSTEM-DATA;          # variable application
```

```
  acl IN-OUT {
    $IN-OUT;              # variable application
  }
}
```

This excerpt will be expanded as:

```
system BILOVICE {
  domain xyz.com;
  acl IN-OUT {
    from [10.0.0.0/8];
    accept;
  }
}
```

prototype context This mode means that variable context is derived from a *prototype*. Prototypes are templates used for building of similar configuration directives in different parts of the configuration tree (see more in configuration(7)). The prototypes can be also used as the context and such variables can be used in different evironments. Example:

```
system BILOVICE {
  set IN-OUT acl-1 { ... } # prototype acl-1 used
  ftp-proxy FTP-PROXY {
    session-acl OUTGOING {
      $IN-OUT; ... } }     # used in ftp-proxy.session-acl
  tcp-proxy TCP-PROXY {
    session-acl OUTGOING {
      $IN-OUT; ... } }     # used in tcp-proxy.session-acl
```

Warning

Current version of the CML does not support defining of `system` sections within a macro body.

param *varname;* Section variable (macro) parameter, or forward variable declaration for `switch` and `include` directives.

Section variables can have parameters that behaves like data variables inside the section variable definition. Values of parameters are defined when variable is applied. Example:

```
set CLUSTER system {        # macro definition
  param NAME;               # parameters definition
```

```
  param PROXIES;

  hostname $NAME;
  acl IN-OUT {
    services $PROXIES;
  ...
}
system FW-A {
  $CLUSTER GATE-1 { FTP }; # macro application with 2 parameters
}
```

This excerpt will be expanded as:

```
system FW-A {
  hostname GATE-1;
  acl IN-OUT {
    services { FTP };
```

Number of parameters is fixed for all applications of parametrized variable.

for *varname* in { *iteration list* } { *for loop body* } Repeat set of configuration directives several times. The loop control variable *varname* is set to each element of the *iteration list* in turn, and the *for loop body* is generated each time. The iteration list can consist of simple values, variables and sublists which are considered as "subset" rather than "one element". Example:

```
set ALL-ACLS = { IN-OUT, OUT-IN };
command-acl ALLOW-ALL {
  for X in { $ALL-ACLS } { # list with variable with a sublist
    session-acl $X;        # loop control variable application
  }
}
```

This excerpt will be expanded as:

```
command-acl ALLOW-ALL {
  session-acl IN-OUT; # iteration #1
  session-acl OUT-IN; # iteration #2
}
```

In the for-loop body, besides the loop control variable, two special variables can be used, too.

$_run_ Value of this variable is an iteration number.

$_all_ Value of this variable is the whole iteration list.

These variables can help to construct unique names, addresses etc. within multiple times generated for-loop body.

> **Warning**
>
> Current version of the CML does not support defining of `system`, `system.acl` and `proxy` sections within a for-loop body. Also, for-loop cannot be used for constructing of list values (i.e. for-loop inside the item).

switch $*varname* { case *value* { ... } } Differentiate parts of configuration depending on the value in a variable (typically a parameter). According to the value of the *varname* variable, proper **case** label is selected. Match means that the case label *value* is equal to the variable one, or the label has value `*`. Within **case** branches, normal configuration source code continues.

There is one small exception, if you want to alter the value of another variable. Simple using of the **set** command within a **case** branch, will fail due invisibility of the variable behind the **switch** command. In this case, such a variable must be declared in advance by the **param** command. Example:

```
set CLUSTER {
  param HOST;
  ...
  switch $HOST {
    param ADDR;                # variable declaration
    case host-a {
      set ADDR = [10.0.0.1];   # assigning value for "host-a"
    }
    case host-b {
      set ADDR = [10.0.0.2];   # assigning value for "host-b"
      interface DMZ { ... }    # some more definitions
    }
    case * {                   # branch for other cases
    }
  }
  interface INT {
    ipv4 $ADDR;                # using the value
```

> **Warning**
>
> Current version of the CML does not support defining of `system` sections within a switch-case body.

Variable and reference paths

Wherever a simple value can be used, a data variable (of proper type) can be used, instead. Moreover, even a part of a (*non-parametrized*) section variable can be used, too. Proper part must be specified with "dotted-path" (*variable path*). Every path component is either a type name of a non-repeatable section/item/element or name of a repeatable section. Example:

```
set SYSTEM-DATA system {
  domain xyz.com;
}
system BILOVICE {
  set ADMIN = root@ + $SYSTEM-DATA.domain.name;
                        # variable SYSTEM-DATA
                                    # non-repeatable item domain
                                           # item element name
```

Similar concept can be used for references to existing configuration directives. So called *reference path* starts either with a subnode name (when following path downward), or with (one or more) path item consisting of a single up-arrow ('^') character. The latter way may be more explicitly expressed by a single path item formed from an up-arrow and parent section name. Example:

```
shared-file ERR-DOC {
  path "samples/error-documents";
}
system BILOVICE {
  http-proxy HTTP-PROXY {
    document-root ^.^.ERR-DOC;    # twice up, then to ERR-DOC
  # or better
    document-root ^root.ERR-DOC; # up to root, then to ERR-DOC
```

In the example above we see also an application of a "section" (ERR-DOC) in a place where a simple `string` value is expected (element of `mime-types` item). In this case, *name* of the section is used. So, the item will be expanded as `document-root "ERR-DOC";`.

As for variable ones, all reference paths must lead "towards the beginning" of the file.

Methods

Some items have special "elements" called *method*s that can operate with other item elements and produce some output. All methods of an item are described in particular section 5 manual page

describing the item. Example:

```
system BILOVICE {
  interface I1 {
    ipv4 [1.2.3.4/8];
  }
  acl IN-OUT {
    from ^system.I1.ipv4.addr; # error: [1.2.3.4/8] is invalid here
    from ^system.I1.ipv4.net;  # method returning [1.0.0.0/8]
  }
  ftp-proxy FTP {
    listen-on {
      transparent ^system.I1.ipv4.addr:21; # [1.2.3.4/8] invalid here
      transparent ^system.I1.ipv4.host:21; # method => [1.2.3.4]
    }
```

Some predefined methods are available to simple data variable values, too. Namely, the two methods for IP addresses modifications (`host` and `net`, see above) are valid for them. Example:

```
set ACL-INT system {
  param ADDR;
  acl FROM {
    from $ADDR.net;
    ...
  }
}
  $ACL-INT [1.2.3.4/8]; # will produce ... from [1.0.0.0/8];
```

Special usage of data variables

Data variables can be used also as a *token of path*. This is very useful, namely in for-loops. Example:

```
system BILOVICE {
  interface I1 { ipv4 [1.2.3.4/8]; }
  interface I2 { ipv4 [4.3.2.1/8]; }
  ftp-proxy FTP {
    listen-on {
      for X in { I1, I2 } {
        transparent ^system.$X.ipv4.host:21;
      }
    }
```

This excerpt will be expanded as:

```
system BILOVICE {
  interface I1 { ipv4 [1.2.3.4/8]; }
  interface I2 { ipv4 [4.3.2.1/8]; }
  ftp-proxy FTP {
    listen-on {
      transparent [1.2.3.4]:21;
      transparent [4.3.2.1]:21;
    }
```

Data variables can be used also as a *name of repeatable section* (unfortunately not for proxy and ACL level 1 names in current version of CML). This is, again, useful in for-loops. Example:

```
routes {
  for X in { net1, net2 } {
    set NAME = ROUTE- + $_run_;
    static $NAME { ... };
  }
```

This for-loop will produce two sections named `ROUTE-1` and `ROUTE-2`.

Special usage of section variables

Section variables, in fact, represent a "container" of configuration directives. Sometimes, it can be useful to apply only a part of the variable. In this case, we can use the the *variable path* (as discussed above, in Section D) and CML will insert only part of the variable definition corresponding to the path.

There are two modes of insertion:

container mode If the variable with path is applied in the context *equal* to the context of the node referenced, the *content* of it (i.e. all subnodes) are expanded from the definition. Example:

```
  set FTP-PROXIES system {
    ftp-proxy FTP-IN-OUT {
      listen-on { ... }
      ...
    }
  }

  system BILOVICE {
    ftp-proxy SPECIAL-IN-OUT {
```

```
    $FTP-PROXIES.FTP-IN-OUT;    # expand content of FTP-IN-OUT ...
    command-acl SPECIAL-ACL-2 { ... }   # ... and complete it
    ...
  }
```

This excerpt will be expanded as:

```
system BILOVICE {
  ftp-proxy SPECIAL-IN-OUT {
    listen-on { ... }
    ...
    command-acl SPECIAL-ACL-2 { ... }   # ... and complete it
    ...
  }
```

single-node mode If the variable with path is applied in the context *parental* to the context of the node referenced, the node as a *whole* is expanded from the definition. Example:

```
set FTP-PROXIES system {
  ftp-proxy FTP-IN-OUT {
    listen-on { ... }
    ...
  }
}

system BILOVICE {
  $FTP-PROXIES.FTP-IN-OUT;    # expand unmodified FTP-IN-OUT node
```

This excerpt will be expanded as:

```
system BILOVICE {
  ftp-proxy FTP-IN-OUT {
    listen-on { ... }
    ...
  }
```

The CML Control Commands

There are four groups of the CML control commands:

informational **/help**, **/show**, **/info**, **/man**

editing /**goto**, /**edit**, /**delete**, /**undelete**, /**rename**, /**cut**, /**copy**, /**paste**, /**hide**, /**unhide**

configuration /**load**, /**save**, /**verify**, /**generate**, /**cfgpath**, /**rcs**

control /**quit**, /**dbg**

Editing commands operate at one moment with a single configuration node. This point of configuration is called *cursor* and it is indicated by an arrow symbol ("->") pointing to particular node when displaying content of the currently modified node. The cursor node is e.g. deleted if /**delete** command is issued. If the cursor has an invalid value (e.g. when modifying an empty section), such a command will fail. The cursor represents also the exact point, where new nodes will be placed. More precisely: new nodes will be placed *after* the cursor. When editing or adding a node, this node becomes the cursor. When deleting a node, the cursor moves to the node following the removed one. This is the reason, why sequence of /**cut** + /**paste** commands swops the two nodes, similarly like the sequence of "**xp**" or "**ddp**" commands in the **vi** editor.

The configuration files are kept not only in the plain form, however, the RCS system (see rcs(1)) is used for keeping track of configuration changes. The /**load** command checkes whether plain and RCS versions of configuration files differ; if you change the configuration file in an external editor, you will be prompted for authorizing the changes. The /**save** command stores file to the plain form and then does the check-in operation to save it to the RCS form. When creating a new RCS file or a new RCS file version, user is prompted by **rcs** program to add some comment. The comment is closed by entering a line with the dot ('.') only.

/**cfgpath** ***directory*** Set the "configuration root path" used as reference point for relative paths in

- **include** directives (see Section D above)
- `shared-file` and `shared-dir` sections path items
- all other strings referring the filename.

The *directory* must be absolute and must exist.

/**copy** [{ **+** | ***n*** | **+*n*** | ***** }]

Copy nodes to the *clipboard*. The clipboard content can be inserted back to the configuration by the /**paste** command.

By default, only the single node under cursor is copied. This usage of the command is allowed for any node type. All other possibilities (with parameters) allow to store more nodes from the same context.

By means of + parameter, the node under cursor can be appended to the current clipboard content (instead of replacing it).

Instead of several subsequent calls of appending copy, a group of nodes (starting by the current one) can be stored into the clipboard by using the number of nodes as a parameter.

Similarly, all subnodes at current level can be placed to the clipboard by using the asterisk as a parameter (if the above constrains are fulfilled).

/**cut** Move the cursor node to the *clipboard*. The cursor will then point to the next node. The clipboard content can be then put back to the configuration by the /**paste** command.

All the parameter variants of the /**copy** are allowed for this command, too.

/dbg ***level*** **[** **{** **con** | **log** **[*filename*]** **}** **]**

Change amount of displayed messages to the *level* (see logging(7) for possible values and their meaning).

If you want to change the level only for console or log, you can specify the `con` or `log` keyword respectively. In the case of log, you can also change the target file name.

/**delete** Remove the cursor node. The cursor will then point to the next node. The node can be imediately restored by the /**undelete** command.

/**edit** ***configuration directive ...*** Edit the cursor node. The *configuration directive* must be of the same type as the cursor node. The command will replace the cursor node (in an item case) or will reopen it (in a section case).

This command is used for editing (i.e. changing of the *content*) of configuration directives. For some kind of directives (repeatable items, comments, for-loops, section variable applications) this is the only possibility of editing. Other directives can be edited simply by typing them anew and the /**edit** command is only one of the ways how to modify them.

This command cannot be used for *replacing* of one configuration directive by anoter one (in other words, you cannot change *type* of the directive, e.g. to comment-out a line, to change an item for other one etc.). This can be done only by deleting the old directive and typing the new one.

Similarly, when using this command, the node *name* (i.e. repeatable section name, for-loop control variable name) cannot be *changed*, the /**rename** command must be used for this purpose, instead.

Using of this command is simplified by help of C3H — you can type /**edit**, space and press **<TAB>** — the CML will complete the command line by the content of the cursor node. Similar behavior is implemented for editation of some configuration directives not using the /**edit** command (e.g. the `set` directive — see above).

/**edit** Edit the cursor node by an external editor. The content of all subnodes of the node under cursor are exported to a temporary file, the external editor is called, and the edit result is stored back into the configuration file replacing the original node. The editor can be defined by the environment variable `$EDITOR`.

In the current version of the CML, this command usage is permitted for structured comments only. The slash characters ('`#`') on the beginning of comment lines are removed when saving into the file and returned back before reading.

/**generate** Generate system configuration files. For every `system` section, a set of files is created under the directory named `SYSTEM-`*name* (for more details, see Section D below).

If the configuration was changed after the last loading/saving operation, or the configuration loaded is not equal to the last version stored in the RCS file, the /**save** command must be used before issuing the /**generate** command.

[/**goto** *number*]

[/**goto** { + | − } [*count*]]

[/**goto** { + | − | = } *type*]

[/**goto** = *name*]

Move the cursor within the node currently being processed. You can move cursor

- to the node with given absolute *number* as shown by the /**show** command
- *count* nodes from current position toward the end ('+') or the beginning ('−') of node currently being processed (default: one)
- to the next ('+'), the previous ('−') or the first ('=') node with given *type* (where *type* can be "FOR", "SET", section or item type name)
- to the node (repeatable subsection) with given *name* (you can use C3H to complete the name).

/**help [*topic*]** Show the help (to the given command name).

/**hide** Hide the cursor node. The cursor node will be marked as *hidden* and will not be taken into account while working with the configuration, except saving and loading it (e.g. it is possible to have several hidden versions of definition of the same variable). Thus, this feature can be used instead of commenting-out some parts of the configuration.

The node can be enabled by the /**hide** command later.

/**info enum *enumeration*** Show the *enumeration* description (meaning, values...).

/**info descr [*path*]** Show configuration directive (item or section) description). The configuration node is given by the CDF *path*.

Without any parameter, the current node description is shown. Examples:

```
/info descr
/info descr ftp-proxy.command-acl
/info descr ^root.system.interface
```

/**info find [*pattern*]** Find all configuration directives (items and sections) containing *pattern* as a substring. CDF node paths of found directives (applicable e.g. in the /**info descr** command) are printed.

/**info param** Show internal parameters of the configuration. For the explanation of the parameters, see proper section 6 manual pages.

/**load** **[-r *revision*** `[filename]` **]**

Load the configuration from given *filename*. If the name is omitted, last loaded file name is used. Before loading the file, its content is checked against the one checked-out from the RCS file *filename*,v (see rcs(1) for the details). The following cases can be recognized:

- Neither *filename* nor *filename*,v exist. User is prompted whether to create (both of) them.
- File *filename* exists, but *filename*,v not. User is prompted whether to create the RCS file.
- File *filename* does not exist, while *filename*,v does. User is prompted whether to recreate the plain file from the current RCS version.
- Both *filename* and *filename*,v exist, but the checked-in version differs from the plain file. User is prompted whether to store the plain file as a new version.
- Files *filename* and *filename*,v are consistent. File is loaded.

In most cases, user can reject recommended actions and can continue to work with the configuration.

The -r option allows user to request loading of a particular revision from RCS file *filename*,v. This version can be normally browsed in CML and then stored back into both configuration files (plain and RCS). The acceptable forms for *revision* are *major.minor* or '0' for the last version stored in RCS.

/**man [*section*] *topic*** Show given Kernun manual page. The command behaves similarly to the system man(1)) except that it shows only Kernun manual pages. On the other hand, the advantage is in using of C3H support for the *section* and the *topic* completion.

/**paste [*newname*]** Paste the *clipboard* content after the cursor. If the context is not compatible, the command fails. If some unrepeatable nodes were duplicate, the behavior depends on the length of the clipboard: pasting of single member clipboard fails while in the multimember case, all incorrect nodes are pasted as hidden. In the singlemember case, the pasted node can be renamed, if applicable (see the /**rename** command below).

/**quit [!** **]**

Quit the CML tool. This command ends the CML run, using of it is necessary only in the case when configuration has been changed, but is not to be saved (use with the '!' parameter).

/**rcs diff [-r *rev1* [-r *rev2*** **]]**

Display differences between particular revisions of the current configuration file.

/**rcs lock** Lock the current configuration file.

/**rcs log** Display the RCS log for the current configuration file.

/**rcs unlock** Unlock the current configuration file.

/**rcs remove** ***revision*** Remove the particular revision of the current configuration file from the RCS file.

/**rename** ***newname*** Rename cursor node. If the node has a name (repeatable section, variable definition, for-loop) this command can change the name.

/**save [!** `[filename]` **]**

Save the configuration file to the plain file and store it then into RCS. The *filename* parameter can be omitted only if some configuration file was loaded (see the /**load** command above) and the same filename will be used by this command.

Before saving the file, the CML will try to *verify* the configuration. If this verification fails, configuration *will not be saved*. This feature can be switched off by using the '!' parameter. However, we do not recommend to use it unless it is absolutely necessary because the configuration may not be loadable more.

If you use C3H in the place of command argument, the CML will offer the name of the last processed file.

/**[show** [{ `-a` | `-e` | `-p` | `-P` }] [{ *path* | . } [*filter*]]]

/**[show** [{ `-a` | `-e` }] `-c`]

Display the content of the node being currently edited, or the node referred by the *variable path* or *reference path* (for the explanation of paths, see above, in Section D). The last possible choice is showing the clipboard content which is requested by using a special option `-c`.

When option `-a` ("all") is used, the whole configuration tree is displayed (recursive display) instead of a list of subnodes only.

When option `-e` ("expand") is used, the expanded form of configuration tree is displayed (including variable expansion, for-loop expansion etc.).

When option `-p` or `-P` is used, only nodes relevant to current proxy are displayed. The lower-case form show also nodes the relevancy of which is not known.

When the last parameter is used, only nodes with given CDF type are displayed.

The command accepts a bit more complex CML paths than described above in Section D. So called *extended path* can contain (besides normal tokens) also indexed tokens so that it can point even to a repeatable item, section variable application etc. This paths are also used by CML when referring to an error during on-line verification. Example:

```
CML.BILOVICE.HTTP-PROXY> /show ^system ftp-proxy
  ftp-proxy FTP-PROXY { ... }
  ftp-proxy FTP-VIA-HTTP { ... }
CML.BILOVICE.HTTP-PROXY> /show ^system.ftp-proxy[2]
  ftp-proxy FTP-VIA-HTTP { ... }
```

/**undelete** The last deleted node is restored.

/**unhide** Enable the hidden node. The cursor node marked as *hidden* (by the /**hide** command sometimes in the past) will be enabled. During this operation, validity of the node occurrence is checked (e.g. unhiding a repeatable section with the name equal to another section will fail).

/**verify [{ . | -> }]**

Verify formal correctness of configuration. Without parameters, this command, in fact, does the first phase of the /**generate** command work (see above) — expands configuration into a temporary file and tries to read it by the low-level parser.

> **Warning**
>
> This command re-creates some files and directories for its purpose, so if you want to use the configuration (e.g. by the **apply** in the **kat** tool), you must use the **/generate** as the last command.

With parameters, only a part of configuration (the node currently being edited, od the node pointed to by the cursor, respectively) is verified.

C3H SUPPORT

The Command Completion and Context Help support is one of the basic features of the CML. It helps admin to write correct configuration and even guides him to solve some troubles he can get into. The simple basic rule is: "If you don't know what to do now, press **<TAB>**!"

The basic function is quite clear: C3H completes names of the CML commands of all types. In addition, the CML shows possible structure of the configuration node being currently modified.

If the **<TAB>** is pressed after the first word on the line and the word is `set` or type name of a repeatable section, C3H searches for all acceptable names already entered and offers their names to complete. In the case of proxy phase 1 ACL section, the C3H offers `system.acl` names. Example:

```
CML> <TAB>
    * shared-dir <name> { ... }
    * shared-file <name> { ... }
    * system <name> { ... }
CML> sy<TAB>
CML> system <TAB>
<new name>  BILOVICE
CML> system BILOVICE { <TAB>
      admin ...;
      ...
CML> system BILOVICE { _
```

If the **<TAB>** is pressed somewhere in the middle of the line, C3H tries to advise in following cases:

- If the current value must be a keword.
- If the current value must be an enumeration member.
- If the current value must be a global section name.
- If the current value must be a parent ACL section name.
- If the current word begins with a dollar ('$') character, all possible variables are offered. If there already is a variable name followed by a dot ('.'), C3H tries to complete *variable path* (see above, in Section D).
- If the current word begins with a single up-arrow ('^') character, all possible continuation of *reference path* (see above, in Section D) is offered.
- If the current word begins with a double quote ('"') character, filename completion is provided.

The C3H helps you also in some special circumstances:

- If you type the /**EDIT** command and press the **<TAB>**, C3H will complete command line by the content of the node being currently under the cursor so that you can simply edit it.
- The C3H will help you to complete the (extended) CML path when using the /**SHOW** command.
- The C3H will help you to complete the CDF context path when defining a section variable.
- The C3H try to help you to close an errorneous configuration directive and recover from the error state.

Control Sequences

The End-of-file control sequence (**^D**, Control-D) can be used for quitting the CML. It works silently only if the current configuration was not changed after the last loading/saving operation. Otherwise it rejects the operation and the /**quit !** command must be used.

The **^R** (Control-R) sequence is used for command history searching. You can type part of some previous command (the part is displayed in the prompt) and C3H searches in the history to the last command containing such a string. This command is then displayed on the command line and you can tune the selection by adding more characters to the pattern, removing some characters by the **<Backspace>** key or repeating the search by pressing the **^R** again. If your selection is completed, press **<Enter>**, the selected command is placed into command line buffer and you can edit it. The CML tool saves command history at the end of its work and restores it at the beginning.

The **^U** (Control-U) sequence is used for clearing the command line.

THE CML PROMPT

The CML prompts users with its own prompt. On the beginning, the text "`CML> `" is in the prompt. Whenever user enter some section, item or variable definition, for-loop etc., name of component occurs at the end of the prompt. Example:

```
CML> system BILOVICE {
CML.BILOVICE> routes {
CML.BILOVICE.routes> set VAR {
CML.BILOVICE.routes.set[VAR]> #{
CML.BILOVICE.routes.set[VAR].#> #}
CML.BILOVICE.routes.set[VAR]> }}}
CML>
```

If the configuration is in not a well defined state, a special appendix to the prompt occurs. The appendices are:

[FAIL] Last operation failed, currently edited configuration directive will be later deleted, it is time to immediately end it (you can try to let the C3H to advise you proper break-thru — press the **<TAB>** and **<Enter>** keys).

[OPEN] Grammar parser is awaiting for the opening brace, type "**{**".

[NAME] Variable name is expected (in the for-loop or variable setting).

[TYPE] Variable type name (an equal sign or CDF context) is expected.

[LIST] For-looop iteration list is expected.

If the prompt is too long (the path is too deep), the path displayed in the prompt is stripped from the left.

THE CML SEMANTICS

There are several configuration directives having a special meaning in the CML.

VERSION, ORIGIN and CML-ID Tags

The CML adds a special directive `version` to each saved configuration so that newer versions of the software can indicate potential problems with understanding of an old configuration. This tag cannot be entered by user and is invisible for him/her.

Another hidden directive `cml-id` keeps the current RCS version of the configuration for easier detection of problems caused by some configuration changes done by the admin.

Also, the `origin` directive keeps track for the time and host name of configuration creation.

Shared Files and Directories

On the global configuration level, two kinds of sections concerning shared files (across the whole CML) can be defined. Both of them specify files/directories which primarily resides on the computer where the CML is run. During generation process, all the files are copied to the output tree. Sections `shared-file` and `shared-dir` are then copied to proxy configuration files, however, paths are changed so that they are valid on the target system. Example:

```
shared-file MIME-SAMPLE {
  path "samples/mime-types"; # path relative to the CML cfg file
}
system BILOVICE {
  kernun-root "/usr/local/kernun";
  ftp-proxy FTP-PROXY {
    ...
```

On the target system, the MIME sample file will be placed as `/usr/local/kernun/etc/shared/mime-types` and the configuration file for the **ftp-proxy** (i.e. `/usr/local/kernun/etc/ftp-proxy.cfg`) will contain

```
shared-file MIME-SAMPLE {
  path "/usr/local/kernun/etc/shared/mime-types";
}
  ftp-proxy FTP-PROXY {
    ...
```

Level 1 ACL

Kernun proxies use several levels of ACL to control access to their services (see access-control(7)). Typically, the first level of ACL (called `session-acl`) decides according to the source and destination addresses, transparency, time etc. All these data are common to all proxies. Higher levels of ACL take in the count also some protocol specific information. In the CML, therefore the first level of ACL is picked up from the proxies definition to the system one. This allows the administrator to make general decisions and propagate them to particular proxies. At the proxy level, some protocol specific directives can be appended to all or to concrete ACLs. Higher levels of ACL are left fully to proxy specifications. Examples:

```
system BILOVICE {
  acl IN-OUT {
    from [1.0.0.0/8];
    service { FTP-PROXY, HTTP-PROXY }; # target proxies
  }
  acl OUT-IN {
```

```
    from { ! [1.0.0.0/8]; * };
    service { HTTP-PROXY };            # target proxies
  }
  ftp-proxy FTP-PROXY {
    session-acl * {      # add to all ACLs
      data-port 20;
    }
  }
  http-proxy HTTP-PROXY {
    session-acl OUT-IN { # add only to named ACL
      plug-to www.internal : 80;
    }
  }
```

Proxy configurations will look like:

```
ftp-proxy FTP-PROXY {
  session-acl IN-OUT {
    from [1.0.0.0/8];          # inherited from BILOVICE.IN-OUT
    data-port 20;              # added by FTP-PROXY.*
  }
  # session-acl OUT-IN is not generated
}
http-proxy HTTP-PROXY {
  session-acl IN-OUT {
    from [1.0.0.0/8];          # inherited from BILOVICE.IN-OUT
    # nothing more added
  }
  session-acl OUT-IN {
    from { ! [1.0.0.0/8]; * }; # inherited from BILOVICE.OUT-IN
    plug-to www.internal : 80; # added by HTTP-PROXY.OUT-IN
  }
}
```

GENERATED OUTPUT

Command /**generate** (see above Section D) creates in the configuration directory (i.e. directory where the currently loaded the CML file resides, or the current working directory) a tree of configuration files with root named `SYSTEM-`*`system-name`*. This tree is then copied into the filesystem root onto target machine by the kat(8) **apply** command. Within this tree, following files are generated:

/etc/rc.conf sources: `system.hostname`, `system.domain`, `system.interface(.alias)`, `system.routes.default`,

system.routes.static, system.cluster.interface(.alias), system.rc-conf

/etc/sysctl.conf sources: system.sysctl

/etc/pf.conf sources: system.packet-filter, system.pf-queue

/etc/passwd and other source: system.user and original /etc/passwd

/etc/services source: system.services

/etc/aliases source: system.admin (for root alias), original /etc/aliases

/etc/resolv.conf sources: system.domain, system.resolver, system.use-resolver

/etc/hosts source: system.hosts-table

/etc/crontab source: system.crontab

/etc/ntp.conf source: system.ntp

/etc/sshd/sshd_*NAME*_config for each ssh-server section *NAME*

***KERNUN_ROOT*/etc/*NAME*.cfg** for each proxy section *NAME*

KERNUN_ROOT*/etc/shared/*NAME for each shared-file and shared-dir section *NAME*

***KERNUN_ROOT*/etc/postfix.*NAME*/master.cf and main.cf** for each smtp-forwarder section *NAME* with *agent* (Postfix) and for local-mailer (if used).

***KERNUN_ROOT*/etc/namedb.*NAME*/ directory** for each nameserver section *NAME*.

***KERNUN_ROOT*/etc/newsyslog.conf** source: system.rotate-log and /etc/newsyslog.conf

The *KERNUN ROOT* denotes the value of the kernun-root configuration directive of the particular system section (typically /usr/local/kernun).

ENVIRONMENT

KERNUN_LOG_FILE The file name where log messages will be redirected. If not set, system logging is used.

SEE ALSO

Kernun: access-control(7), configuration(7), kernun(7), logging(7), kat(8)

FreeBSD: man(1), rcs(1), vi(1), editline(3)

BUGS

The current version of the CML cannot process whole systems, level 1 ACLs and whole proxies within for-loops or section variables.

NAME

cwcatd — Clear Web automatic categorization daemon

SYNOPSIS

cwcatd [-hv] [-d *dbglev*] -f *cfgfile*

DESCRIPTION

The **cwcatd** daemon handles automatic categorization of WWW servers. If categories for a requested URL are not found in the Clear Web database, the http-proxy(8) or the icap-server(8) can optionally request automatic categorization of the URL. These requests are appended to a queue that is processed by daemon **cwcatd**.

The daemon reads uncategorized URLs from the queue. For each URL, it tries to download the referenced web page. The downloaded page is passed to an automatic categorizer, which tries to assign categories according to a heuristic applied to the page content. If it succeeds, the result is stored in a local database.

Future requests to a locally categorized server will get categories assigned according to the local database. If categories of the server appear in the downloaded Clear Web database in its periodic update, the result of the automatic local categorization will not be used any more.

OPTIONS

-h Print usage information.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

SEE ALSO

http-proxy(8) or the icap-server(8)

NAME

dns-proxy, test-dns — Domain Name System (DNS) proxy

SYNOPSIS

dns-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-dns [-hv] [-d *dbglev*] -f *cfgfile* [-t *test_expr*]

DESCRIPTION

The **dns-proxy** provides proxying service for the Domain Name System (RFC 1034+1035 and some of their extensions).

The proxy handles incoming requests in four basic modes:

Request is *denied*. Request is not processed at all. Administrator can choose from several ways of request denial.

Request is *faked*. Request is responded without querying any other server. The faked response is set by administrator.

Request is *forwarded*. Request is resent to one from group of servers defined by the administrator. This mode is typical for

- forwarding queries from host not fully connected to the internet
- forwarding queries to server hidden in private network
- forwarding zone transfers across the firewall.

Request is *resolved*. This is fully functional mode when the proxy tries to recursively resolve the request form the scratch. It starts from the root servers and accepts only authoritative answers until it gets the final authoritative answer.

The proxy is not designed as a server, neither for external queries to our domain, nor as a local caching name server. Typical scenario is that all clients ask another server in the internal network and this server queries the proxy (if needed) and caches the answers. That's why the proxy does not cache answers, there is only a small cache of name servers for internal purpose only.

The proxy behaves like a server for the client and vice versa, with full syntax and semantics verification (see below, protection against *cache-poisoning*). When querying other servers, **dns-proxy** uses random source port and ID (protection against *reply-spoofing*).

The proxy completely reconstructs answers from servers and limits the size of reply sent to client (configurable by `ptr-reply-size` and `adr-reply-size` items). This feature protects clients against to attack based on buggy resolver routines.

The proxy usually runs as two processes: the single child process manages all the sessions and the parent process manages the child and restarts it after a failure. You can learn more in

udpserver(7) manual page, although the **dns-proxy** does not use the **udpserver** library, in fact. However, it uses the same operation logic.

Format of the proxy configuration file is described in dns-proxy.cfg(5). For the **dns-proxy**, any host in configuration file must be specified by its address, not by its name.

Program `test-dns` tests syntax and partially semantics of configuration; for test expression syntax, see test-expr(5).

Startup and Configuration

The proxy reads its configuration file and starts listening on specified IP sockets (address/port couples), as specified in the `listen-on` configuration section (see listen-on(5)). Proxy listens for both UDP and TCP protocols.

If support of transparent connections (see transparency(7)) is requested by item `transparent` in section `listen-on`, the corresponding NAT redirections are established during proxy startup and removed upon exit. However, transparent connections are in fact not supported by this proxy, decisions about servers are made according to the proxy configuration, not by original destination.

Warning

In the case of UDP requests redirected to the proxy by more general NAT rules, the real destination is neither being detected nor used in ACL selection.

In the resolve mode, the proxy checks each address being stored into cache whether it matches to one of addresses, the proxy listens on. If it does, the record is not stored, so that this does not lead to infinite loop. For this reason, listening addresses (for port 53) must be specified explicitly, expression [0.0.0.0]:53 is not allowed.

DNS Subset

Current version of the **dns-proxy** implements following subset of protocol:

*OPCODE*s

`QUERY` RFC 1035

`NOTIFY` RFC 1996

Requests with unimplemented OPCODEs are replied with the `NotImp` response code. Requests with unknown OPCODEs are replied with the `FormErr` response code.

*CLASS*es

`IN` RFC 1035

`*` RFC 1035

Requests with query resource record (RR) of unimplemented CLASS are replied with the `NotImp` response code, unknown classes cause the `FormErr` response code. The '*' requests are converted to IN, resolved and then sent to the client with authority flag set to 0.

*TYPE*s

`A` RFC 1035

`AAAA` RFC 3596

AFSDB	RFC 1183
AXFR	RFC 1035
CNAME	RFC 1035
DNSKEY	RFC 4034
DS	RFC 4034
HINFO	RFC 1035
IXFR	RFC 1995
MX	RFC 1035
NS	RFC 1035
NSEC	RFC 4034
NSEC3	RFC 5155
NSEC3PARAM	RFC 5155
OPT	RFC 6891
PTR	RFC 1035
RRSIG	RFC 4034
SOA	RFC 1035
SPF	RFC 4408
SRV	RFC 2782
SSHFP	RFC 4255
TXT	RFC 1035
*	RFC 1035

Requests with query RR of unimplemented TYPE are by default replied with the `NotImp` response code. This behavior can be changed in the configuration (ACL settings). Requests to unknown TYPE are replied with the `FormErr` response code.

Requests

Every request is stored into a table item containing all necessary information. Size of this table must be specified in configuration (`requests-table-size` item) and it is recommended to reserve a little more items then estimated number of parallel requests. Some requests processed in resolve mode can generate so called *internal requests* (see below) that occupy table items, too.

Besides number of requests, number of simultaneously opened sockets is monitored. The maximum of sockets must be specified in the configuration (`sockets-table-size` item).

There are two kinds of internal requests:

CACHE requests If the proxy is to ask some server, address of which is not in the cache, it generates an internal requests with `A` and `AAAA` query for the name of the server. This request is handled in the same way as the original query with the exception that the result is stored in the cache.

Choosing just IPv4 or IPv6 protocol when querying servers can be done by the `server-proto` item. The default is using both of them, or using just the IPv4 when no system interface has an IPv6 address defined.

CNAME requests If the answer got from a server contains `CNAME` RR and no (trusted) RR for the canonical name, proxy generates an internal request with the same query RR type and query name equal to the canonical name received. This request is handled in the same way as the original query with the exception that the result is added to the previously received RRs. If the internal request fails to complete the resolution, the original request is replied by the `ServFail` response code.

Both types of internal requests suspend processing of the original request (originator) until the internal request is completed. If another request is to generate a new internal request of the same subject as another running one, no new internal request is created and the originator is suspended waiting for the first internal request, too. After (successful or unsuccessful) completion of internal request, all originators are waked up.

Similar principle is used also for client requests. If a request with the same parameters (query name, query type, EDNS UDP payload etc.) is already being processed, new request is also suspended, waiting for the result of the previous request.

Both types of internal requests can also generate new internal requests. For instance, in following definition:

```
domain1 IN NS ns.domain2
domain2 IN NS ns.domain3
```

a request to the `domain1` will generate request to the `ns.domain2` name and solution of this will result to a new `CACHE` request to `ns.domain3`. Proxy detects an infinite loop, if occurs. Proxy also limits number of internal requests generated by one internal request in the row (item `internal-request-depth`) to prevent DoS attack by means of non-infinite but very long loop of references.

Both types of internal requests respect the resolving policy according to the query name and type given by the configuration. For every new request, the `request-acl` list (see below) is searched through and the new ACL defines operation to be used.

If no free request table item is available,

- incoming UDP requests are replied with the `ServFail` response code
- incoming TCP requests are rejected by closing the connection
- internal requests of any type fail and these failures are propagated to all originators.

Access Control Lists

The proxy uses two layers of ACL (see access-control(7)) named `session-acl` and `request-acl`.

When a request arrives, configuration is consulted, proper `session-acl` is selected and according to it, request is served or not.

Subsequently, protocol-specific parameters of query is checked against set of `request-acl` entry conditions and proper mode of operation is selected.

Additionally to the general Kernun ACL concept, `request-acl` brings new entry condition items:

`query-name` This item contains a set of regular expressions and/or strings describing names, querying for which is to be dealt by this `request-acl`. When using the regexp form, you have to respect the dot placed to the end of every name before request processing. The queried name matches a member of the set if

- matches the regexp (regexp case) or
- is part of the domain (string case).

Examples:

- Regular expression `/^[^.]*\.tns\.cz\.$/` matches queries to all hosts in the domain `tns.cz`.
- String `kernun.com` matches e.g. queries to `www.cz.kernun.com`, or `kernun.com`, but not `kernun.com.cz`.

`request-type` This item can define subset of DNS operation codes and RR types that is to be dealt by this `request-acl`.

By these two items, a detail selection of `request-acl` can be done to set special handling for different tasks like regular queries, zone transfers, server notifications etc.

If no matching ACL is found, request is replied by the `Refused` response code. If ACL is found, query type and class are checked. Requests with classes other than `IN` and `*` (ANY class) are rejected with response code `NotImp`.

As we stated above, there are several possible proxy operations. The proxy decides among them by matching query RR type against a set of special `request-acl` items `query`/`notify`. The first matching item is used and the operation is executed. If no proper item is found, request is rejected with the `Refused` response code.

In case of `resolve` and `forward` operations, request is resent (with a new, random ID) to a new server. The set of possible responders is defined in a global section of `ns-list` type. Each received reply RR is then checked against a set of special ACL items called `reply` in the same manner as queried RR is checked. The `reply` items can tune handling of particular RR (`permit`, `remove`), so as even predefine reaction to the whole request (`abort`, `deny`). If no proper `reply` item is found, request is rejected with the `Refused` response code. If `permit` action is required for non-implemented RR, record is removed.

After filtering the response from server, other proper RRs (given by special `fake` items) can be added to the answer. The same set of items is used for reply construction in case of `fake` operation. Fake RRs are placed into the answer in the order of appearance in the configuration.

After completing all answer RRs, reply is completely reconstructed and sent to the client. The resolved requests will have the authority (AA) flag cleared while for the forwarded requests, the admin can choose whether to preserve or clear the flag (see the `request-acl.query.clear-aa` definition in dns-proxy(5)).

If the response has the `NXDomain` response code, or the `NoError` response code with no answer (AN) records, and if this status was caused by the proxy (e.g. due to denying query or filtering response), the proxy will add, by default, a SOA record with proper TTL for successful negative caching in clients. This behavior can be configured by the `neg-resp-ttl` item of the particular `request-acl`.

Warning

If the authoritative answer in `resolve` operation is not available, request is replied with the `ServFail` response code. There are some situations where this approach is not applicable. For instance, queries to `mail-abuse.org` domain end by non-authoritative answer. We recommend using a special `request-acl` for this case, forwarding requests of this type directly to proper name servers.

Syntax and Semantics Verification

Besides Security Policy application, the proxy checks both queries and replies to correctness in sense of relevant RFCs.

Names

Names can be at most 254 bytes long, every label can be at most 63 bytes long. Labels can contain only alphanumerical characters and a hyphen ('-'). We allow also underscore ('_') and slash ('/') because they are commonly used.

Consistency

Request ID of the answer is checked to be equal to the ID of query. Query (QD) section of the answer is checked to be equal to the query. Every answer (AN) RR must be relevant to the query or to the previous RR. Every authority (NS) RR must be relevant to the query or to the canonical name of some AN RR. Every additional (AR) RR must be relevant to some AN or NS RR.

Trust

Every server is introduced into the cache as an authoritative name server for some domain. So, all answer RRs received from this server are trusted only if they belong to this domain or some subdomain. This criteria may cause an infinite loop when two or more domains refer each other without having any regular glue record (i.e. nameserver in own domain or a subdomain). If you need to accept such a domain, you must make an ACL for this domain forwarding requests directly to proper nameservers.

Cache

Name server cache is used for increased efficiency, namely for repeated queries to the same domain (TLDs, resending query via TCP, resolution of CNAMEs etc.). "Root servers" for different network

zones are defined in `ns-list` configuration sections, each zone has a separated cache "zone" named by the name of ns-list section. All other name servers and their addresses are introduced to the cache as a result of authoritative answers. "Authoritative" server (in sense of dns-proxy) is either a root server or a server delegated by some "authoritative" server for a parent domain (already being in the cache). All new RRs are used with respect of their time to live (TTL) value. When the minimal of TTLs of name servers (both their NS and address records) for a domain expires, the domain item is unusable and it is removed from the cache at nearest cleanup. Similarly, when the minimal of TTLs of addresses for a host expires, the host item is unusable and it is removed from the cache at nearest cleanup.

Some properties of the cache are configurable in a special `cache` section:

`cleanup-period` sets the period (in seconds) of cache cleaning up. After the period, all items that were not used within the period and all expired items are removed.

`max-domains` sets the maximum number of *domains* (not individual NS RRs) stored in the cache. This value should be at least as large as `requests-table-size`.

`max-hosts` sets the maximum number of *hosts* (not individual address RRs) stored in the cache. This value should be at least approximately five times larger than `requests-table-size`.

If any maximum is reached, a non-periodical cleanup is started. This cleanup removes all items currently not used.

Server Selection Algorithm

When a domain name is to be resolved, the longest match search in the cache is done. After it, the new best server for the domain found is chosen. Server comparison criteria:

1. Resolved, but never contacted servers.
2. Unresolved servers, never tried to be resolved.
3. Responding servers (sorted by response time rounded to entire seconds).
4. Non-resolved or non-responding servers (sorted by time of the last attempt).

This algorithm guarantees a primitive "load balancing" and error recovery of multiple servers.

When querying for an EDNS request, the EDNS servers have priority. If a server responds FormErr to an EDNS query, the non-EDNS query is repeated immediately.

First of all, the selected server is queried with a very short (1 sec.) timeout. When this timeout fires, the query is simply repeated to avoid errors caused by loosing UDP packets.

Then the queried server has a longer timeout for the response. Each server has its own timeout stored in the cache. Starting value of this timeout is set by `query-timeout` configuration directive. Each time the server does not respond within the timeout, the timeout doubles, up to `server-dead` value. When the timeout reaches this value, server is marked as "dead" and a new attempt to contact it cannot be done until `server-retry` seconds period. If the server responds within the timeout, his timeout for the next attempt is set to his response time plus `query-timeout`.

The number of attempts per one query is hardcoded to eight (regardless of which servers were queried). However, typically this number is not reached before firing the `request-timeout` (see below).

The same mechanism is used for selection among forwarders. That's why forwarders lists are also stored in the cache (every list in its separated zone).

The timeout for the whole request processing (including resolving of generated internal requests etc.) is also set in the configuration (`request-timeout`) and if reached, request is replied with the `ServFail` response code.

Zone Transfers

Zone transfers need some more special handling. First of all, the requests are typically addressed by originators *directly* to a conrete server. That's why either the transparent mode (if public addresses are used), or non-transparent mode to a dedicated address/port on firewall (in the case of server on a private address) should be used. Also, besides `QUERY` operation, the `NOTIFY` operation should be permitted.

Moreover, the own transfers (responses to the `AXFR`/`IXFR` queries) should be sent by servers either *separated* (i.e. more DNS messages with one RR in each one) or *aggregated* (i.e. all RRs in one DNS message). The proxy can force one of these methods, or keep the incoming format according to the `xfr-format` configuration directive.

Configuration example:

```
ns-list MASTER {
  server ns.x.y.z [10.1.1.1];
}
ns-list SLAVE {
  server sns.x.y.z [20.2.2.2];
}
...
dns-proxy ZONE-TRANSFERS {
  ...
  request-acl TO-MASTER {
    to non-transparent [20.2.2.1]; # special external fw adr
    query { axfr, ixfr } forward MASTER;
    ...
  }
  request-acl TO-SLAVE {
    to transparent [20.2.2.2];
    notify forward SLAVE;
    ...
  }
```

Common Kernun Features

The proxy uses common Kernun mechanism for listening on its sockets, optionally changing root directory and running with alternative user privileges. For more detailed information, see application(5) and listen-on(5).

The proxy uses common Kernun mechanism for network input/output operations. Configuration allows for specifying several parameters like buffer sizes and timeouts, both for client and server connections. They can be included in `client-conn` and `server-conn` configuration sections, respectively. For more detailed information, see netio(7).

The proxy uses common Kernun mechanism for logging. For more detailed information, see logging(7). For every request, one REQUEST (DNSP-860-I) message is logged (besides one or two ACL messages - DNSP-810-I and DNSP-820-I). Every log message has process ID suffix equal to the index of request being currently processed.

The proxy, in fact, does not use common Kernun mechanism for name resolving (see resolving(7) manual page), because it does not use DNS names at all. In spite of that, the item `use-resolver` remains in the **dns-proxy** configuration for compatibility with other proxies (and thus e.g. ability to use common cml(8) variables).

Signals

The **dns-proxy** handles following signals:

SIGUSR1 Log level increasing.

SIGUSR2 Log level decreasing.

SIGINFO Operation status logging; parent process logs info about all children, child process dumps cache content and requests table content.

SIGHUP, **SIGINT**, **SIGQUIT**, **SIGTERM** Immediate termination; proxy immediately closes all connections and terminates.

Program options

The program options are as follows:

-h Print usage information and exit.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

-t *test-expr* Test configuration according to given expression. Format of the *test-expr* is described in test-expr(5).

BUGS

Currently, the **dns-proxy** doesn't implement following features:

- more queries (QD RRs) in one request
- wildcard ('*') queries.

SEE ALSO

Kernun: dns-proxy.cfg(5), listen-on(5), application(5), test-expr(5), DNSP-810(6), DNSP-820(6), DNSP-860(6), access-control(7), configuration(7), logging(7), netio(7), resolving(7), transparency(7), udpserver(7)

FreeBSD: named(8)

NAME

ftp-proxy, test-ftp — File Transfer Protocol (FTP) proxy

SYNOPSIS

ftp-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-ftp [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

The **ftp-proxy** provides proxying service for the File Transfer Protocol (RFC 959) and its extensions. For crucial commands, it behaves like a server for the client and vice versa, with full syntax and semantics verification, commands not impacting session state are only recognized and simply forwarded to server.

Proxy reads its configuration file and starts listening on specified IP sockets (address/port couples), as specified in the `listen-on` configuration section (see listen-on(5)).

Format of the configuration file is described in ftp-proxy.cfg(5).

Program **test-ftp** tests syntax and partially semantics of the configuration; for test expression syntax, see test-expr(5).

Access Control Lists

FTP-proxy uses three layers ACL (see access-control(7)), namely `session-acls`, `command-acls` and `doc-acls`.

session-acl

When a connection arrives, configuration is consulted, proper `session-acl` is selected and according to it, connection is allowed or not.

When a transparent connection arrives (i.e., a connection destined for a real server, transparently redirected to the proxy by PF, see transparency(7) for details), the proxy will connect to the original destination server.

When a non-transparent connection arrives (i.e., a connection destined directly for one of the sockets the proxy is listening on) and is allowed by policy, either the proxy must be configured to connect to a specific remote server with the `plug-to` configuration directive (see below) or user must specify remote server (see below).

If a `plug-to` directive is used for a transparent connection, it has precedence over the original destination. This means that the proxy will ignore the original destination and connect to the `plug-to` server, instead.

Another possible operation mode is to forward all FTP traffic to another proxy in chain. In this case, name or address of this `hand-off` proxy must be specified in configuration together with a method used to introduce the remote server to the `hand-off` proxy. The current version of Kernun ftp-proxy can use all methods that it understands, i.e.

- appending server name to the first **USER** command
- using **SITE** command (or aliases of it).

Firewall administrator can choose any method described in auth(7) (except for NTLM) for authenticating users on the proxy. If the method requires sending of proxy-username and proxy-password, the client can use several methods of combination of **USER** and **PASS** commands (see below).

command-acl

After successful proxy-authentication, proxy attempts to find a `command-acl` acceptable for this session. When found, access is either denied or granted and ACL restrictions are applied. Then, connection to the remote server (or hand-off proxy) is made and normal FTP operation starts. At this point, the so called "initialization phase" is finished. Firewall administrator can limit, for how long this phase can last at maximum, measured both in seconds and in number of commands. ACL search is then repeated for each incoming user command.

There are three FTP specific restrictions in `command-acl`:

enable-port This item specifies that user can request connection to any port, not only to standard FTP port (21).

command Set of these items specify restrictions for the FTP Protocol commands (permitting/denying or data-transfer size limits). By default, all commands are denied. If you use restrictive policy, be careful when allowing commands — some of them are used in groups (e.g. `PASV+RETR`, `RNFR+RNTO` etc.).

feature Set of these items specify restrictions for the FTP Protocol extensions. When the proxy forwards server reply to the `FEAT` command, all options are compared to the set of these items and proper filtration is applied. The options unimplemented in the proxy are denied and cannot be permited. Other options can be permited and/or denied, the default proxy behavior is passing all known and removing all unknown ones.

doc-acl

When a data transfer (downloading or uploading a file), is initiated the list of `doc-acls` are searched. Among standard layer 3 conditions, the foolowing should be used: `direction` (incoming or outgoing data), `size` and `mime-type`. For the MIME type identification (see doctype-identification(7)), only filename extension and magic number recognition should be used in FTP. In adition, a FTP specific condition, the `filename` — matching the file name (without path) can be applied.

> **Warning**
>
> Be careful when using both `filename` and `mime-type` entry conditions. As they are two different item types, they are checked in conjunction, so the file must match *both of them* to match the ACL.

A `doc-acl` controls how the proxy handles the data. In addition to standard ACL actions `accept` and `deny`, additional processing may be configured:

`antivirus` Specifies parameters and actions for checking data for viruses.

`html-filter` Specifies parameters for filtration of HTML documents. Details about HTML filter configuration can be found in mod-html-filter(5).

Data Transfers

For data connections, proxy uses the same transfer initiation method as the client, by default. If client uses active method, proxy uses **PORT** command, if client uses passive method, proxy tries **EPSV** command and if it is rejected, **PASV** command. Forcing of active/passive method for particular servers can be specified by `data-transfer` configuration directive. When connecting to a server, it is checked against all data-transfer lists (using MATCH-ANY style, see host-matching(7) for details) and if it matches, proper data transfer method is used.

If client uses active data transfer, the proxy binds generic source-port for the data connection, unless another port is forced by `data-port` configuration directive. Under normal circumstances, the proxy runs under non-root user and cannot use reserved data ports (0..1023) this way.

User can choose data transfer method used from proxy to server by Kernun specific FTP command **BNB** (see below).

If the proxy executes the antivirus check, it must receive the whole file content before sending any data to destination. It can cause problems with some timeouts on peers. In such situations, the admin can decide to use the *antivirus keepalive* feature (see antivirus(5)). In this case, proxy sends some data to destination before it completes antivirus checking and (if a virus is found) aborts the file transfer. However, on the destination side, an invalid fragment of transferred file will occur.

Common Kernun Features

The proxy uses common Kernun mechanism for listening on its sockets, forking new processes as needed and killing old redundant processes, optionally changing root directory and running with alternative user privileges. For more detailed information, see application(5) and tcpserver(7).

The proxy uses common Kernun mechanism for network input/output operations. Configuration allows for specifying several parameters like buffer sizes and timeouts, both for client and server connections. They can be included in `client-conn` and `server-conn` configuration sections, respectively. For more detailed information, see netio(7).

The proxy uses common Kernun mechanism for logging (see logging(7)). For every session, a couple of SESSION-START (FTPP-808-I) and SESSION-END (FTPP-809-I) messages is logged (besides one or two ACL messages - FTPS-810-I and FTPS-820-I). For every data transfer, two DATA-INIT (FTPT-880-I) and one summarization DATA-END (FTPT-890-I) messages are logged.

The proxy uses common Kernun mechanism for name resolving (see resolving(7)).

The proxy uses common Kernun mechanism for runtime monitoring. For more detailed information, see monitoring(7).

The proxy uses common Kernun mechanism for traffic shaping. For more detailed information, see traffic-shaping(7).

Program Options

The program options are as follows:

-h Print usage information and exit.

-v Display version information and exit.

-d *dbglev* Set debug level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

-r Resolve names in configuration prior to testing.

-t *test_expr* Test configuration according to given expression. Format of the *test expr* is described in test-expr(5).

KERNUN SPECIFIC COMMANDS

Kernun FTP Proxy recognizes several specific commands during session initialization and one special command within FTP Protocol ('**BNB**') for setting some options during normal operation.

Transparent/plug-to Case, No Authentication

In this case, no initialization commands are needed, because the proxy knows the targer server either from socket destination IP address (see transparency(7)) or from configuration `plug-to` directive (see ftp-proxy.cfg(5)).

Non-transparent/plug-to Case, No Authentication

In this case, the proxy must get to know name or address of the target server. The client can specify it by two ways:

a) `SITE` *`server`* `[`*`port`*`]`

This command must be used as the first one in session. The proxy tries to connect to the given server and port and if succeeds, the initialization phase is completed.

b) `USER` *`remote-user@server`*`[:`*`port`*`]`

or

```
USER remote-user@server [port]
```

or

```
USER server!remote-user [port]
```

This command must be used as the first one in session. Proxy tries to connect to given server and port and to apply a new **USER** command with the given user name to this server. If it succeeds, the initialization phase is completed.

In both cases, the server can be specified by a host name or by an IP address, port can be specified by a number or as a Well-Known Service (WKS, see /etc/services file and services(5) system manual page).

Transparent/Plug-to Case, with Authentication

In this case, the proxy must get to know the name of proxy user. The client can specify it by two ways:

a)
```
USER proxy-user
PASS proxy-pasw
```

These commands must be used as the first ones in session. If the proxy receives the first **USER** command with argument containing no '@'s, it handles this command as proxy-authentication command and expects **PASS** command with proxy-user password. After successful authentication, proxy tries to connect to target server and if succeeds, the initialization phase is completed.

b)
```
USER proxy-user@remote-user
PASS proxy-pasw@remote-pasw
```

These commands must be used as the first ones in session. The proxy remembers both user names and expects the **PASS** command. Then tries to authenticate the proxy-user and after connecting to the target server applies both **USER** and **PASS** commands for the remote user, too. If it succeeds, the initialization phase is completed.

Warning

For this case, the proxy-user password must not contain the '@' character.

Non-transparent, non plug-to Case, with Authentication

In this case, the proxy must get to know both the target server and the proxy-user name/password. The client can specify it by many ways resulting as combination of methods described in the last two sections, e.g.:

```
a)  USER proxy-user@remote-user@server[:port]
    PASS proxy-pasw@remote-pasw

b)  SITE server [port]
    USER proxy-user@remote-user
    PASS proxy-pasw@remote-pasw

c)  USER proxy-user
    PASS proxy-pasw
    USER remote-user@server[:port]

d)  SITE server [port]
    USER proxy-user
    PASS proxy-pasw

e)  USER proxy-user
    PASS proxy-pasw
    SITE server [port]
```

BNB Command

BNB Command is intended to allow communication with the proxy not only during the initialization phase, but within whole session. In current version, the communication language and data transfer mode (active/passive) to server can be set using this command. Quick help on this command is available by using '**BNB HELP**' command. This command should be send from regular ftp client by using the **QUOTE** command (see ftp(1) for details).

HTFTP Mode

The **ftp-proxy** is able to cooperate with the **http-proxy**. If an HTTP request with `ftp:` scheme is received by the **http-proxy**, it is forwarded to the **ftp-proxy** according to the item `ftp-proxy` in **http-proxy** configuration. Then the **ftp-proxy** handles the request using FTP and returns the result back to the **http-proxy**. A Kernun-specific protocol called HTFTP is used for communication between the two proxies. If the **ftp-proxy** should cooperate with the **http-proxy**, it is necessary to set `htftp-mode` in one of its `session-acls`. Any connection matching this `session-acl` is handled in HTFTP mode while other connections use normal FTP. A typical configuration reserves a single non-transparent listening port accessible only from localhost for HTFTP.

Note that it is possible to configure the HTML filter and antivirus checking both in the **ftp-proxy** and in the **http-proxy**. Since double-checking the same data is redundant, it is recommended to configure filtration and antivirus checking in only one proxy for data passing through both proxies via HTFTP.

BUGS

Currently, the **ftp-proxy** doesn't implement RFC 1639 (LPSV and LPRT commands) and RFC 2228 (Authentication).

SEE ALSO

Kernun: antivirus(5), application(5), ftp-proxy.cfg(5), listen-on(5), mod-html-filter(5), test-expr(5), FTPP-808(6), FTPS-810(6), FTPS-820(6), FTPP-809(6), FTPT-880(6), FTPT-890(6), access-control(7), auth(7), configuration(7), doctype-identification(7), host-matching(7), logging(7), monitoring(7), netio(7), resolving(7), tcpserver(7), time-matching(7), traffic-shaping(7), transparency(7)

FreeBSD: ftp(1), services(5)

NAME

gk-proxy, test-gk — H.323 Gatekeeper RAS proxy

SYNOPSIS

gk-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-gk [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

The **gk-proxy** provides H.323 Registration, Admission and Status (RAS) proxying service. It allows clients in local network to register at gatekeepers and to request them to manage H.323 connections.

Program **test-gk** tests syntax and partially semantics of the configuration; for test expression syntax, see test-expr(5).

The **gk-proxy** reads its configuration file and starts receiving datagrams on UDP sockets (address/port couples) specified by the `listen-on` configuration directive (see listen-on(5)). It also maintains a list of active sessions with two connections (one from the client, the other to the server). When a datagram arrives, the proxy checks its source and destination addresses and tries to assign the datagram to an existing session. If a match is found, the datagram is passed to a peer belonging to the session. If no session matches and ACL allow it, a new session is created. Otherwise, the datagram is dropped.

The proxy usually runs as two processes (not counting the configuration resolving process - see resolving(7)): the single child process manages all the sessions and the parent process manages the child and restarts it after a failure, see also udpserver(7). Format of the proxy configuration file is described in gk-proxy.cfg(5). The maximum number of concurrent active sessions is set by the configuration directive `max-sessions`.

The proxy registers all clients in a special memory mapped file. Its name must be specified in `map-file` configuration item. Contents of the file is used by the `h323-proxy` for decision about H.323 connection destinations.

The **gk-proxy** uses single-phase ACLs which are checked at the moment of a session establishment. The ACL is named `session-acl`.

When a non-transparent session is created (i.e., a session initiated by a datagram destined directly to one of the sockets the **gk-proxy** is listening on) and is allowed by policy, the proxy must be configured to communicate to a specific remote server with the `plug-to` configuration directive.

When a transparent session is created (i.e., a session initiated by a datagram destined to a real server and transparently redirected to the **gk-proxy** (see transparency(7) for details), the proxy either communicates with the destination server specified by the client or with the one defined by the `plug-to` directive. If a `plug-to` is applicable for a transparent session, it has precedence

over the original destination. This means that **gk-proxy** will ignore the original destination and communicate with the `plug-to` server.

Common Kernun Features

The proxy uses common Kernun mechanism for name resolving (see resolving(7)).

The proxy uses common Kernun mechanism for logging (see logging(7)). When a RAS request comes to the proxy, the SESSION-START (MMCG-810-I) and the ACL (MMCG-821-I) messages are logged. If the request is accepted, a new ID is assigned to it and the SESSION-INIT (MMCG-811-I) message is logged. After end of processing the set of requests from/for the client, the SESSION-END (MMCG-812-I) message is logged.

The proxy uses common Kernun mechanism for traffic shaping. For more detailed information, see traffic-shaping(7).

The proxy uses common Kernun mechanism for policy decisions about received and sent datagrams. It is described in access-control(7) and host-matching(7). For example, it is possible for **gk-proxy** to use the real client's address or any specified address as source address for datagrams forwarded to a server.

Special Configuration Topics

The **gk-proxy** adds many configuration directives to the `session-acl`:

`register` Type of client registration. For details, see h323-proxy(8).

`h323-address` Address to which announced H.323 sessions are redirected. For details, see h323-proxy(8).

`h323-session-timeout` Timeout to establish the H.323 session. After receiving an ARQ or ACF packet, the **gk-proxy** builds the NAT rule so that the following H.323 session will reach running **h323-proxy**. This rule must be deleted by the originator — **gk-proxy**. However, the **gk-proxy** has no information about success or failure of the H.323 session establishing. That's why there is a timeout to delete the rule. If the client starts the H.323 session within the timeout, deleting the rule will not affect the session. Otherwise, the H.323 session will not succeed. If there are problems with establishing H.323 sessions in your network, increase this timeout.

`timeout.session` The session will be terminated if this number of seconds elapse since the session establishment.

`timeout.in` Timeout for datagrams from the server. If so many seconds elapse without receiving a datagram from the server, the session will be terminated.

`timeout.out` Timeout for datagrams from the client. If so many seconds elapse without receiving a datagram from the client, the session will be terminated.

`timeout.both` Timeout for datagrams regardless their direction. If no datagram belonging to a session is received for so long time period, the session will be terminated.

Program options

-h Print usage information.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

SEE ALSO

gk-proxy.cfg(5), listen-on(5), application(5), test-expr(5), MMCG-810(6), MMCG-811(6), MMCG-812(6), MMCG-821(6), access-control(7), configuration(7), host-matching(7), logging(7), netio(7), resolving(7), time-matching(7), traffic-shaping(7), transparency(7), udpserver(7), h323-proxy(8)

NAME

h323-proxy, test-h323 — H.323 Protocol Family proxy

SYNOPSIS

h323-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-h323 [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

The **h323-proxy** provides proxying service for the set of multimedia communication protocols called H.323 protocols. Current version covers many popular applications like Voice-over-IP (VOIP), Microsoft Netmeeting or GnomeMeeting. Proxy handles protocol extensions like H.245 tunneling or Fast Start processing and can co-operate with the gk-proxy(8) — proxying clients' registration requests to gatekeepers. Proxy controls H.225 and H.245 sessions and manages logical data channels opened according to H.245 commands.

Program **test-h323** tests syntax and partially semantics of the configuration; for test expression syntax, see test-expr(5).

Proxy reads its configuration file and starts listening on specified IP sockets (address/port couples), as specified in the `listen-on` configuration section (see listen-on(5)).

Format of the configuration file is described in h323-proxy.cfg(5).

Access Control Lists

The **h323-proxy** uses only single layer ACL (see access-control(7)) called `session-acl`.

When a connection arrives, configuration is consulted, proper `session-acl` is selected and according to it, connection is allowed or not.

When a transparent connection arrives (i.e., a connection destined for another server is transparently redirected to **h323-proxy** — see transparency(7) for details), the proxy will connect to the original destination server.

When a non-transparent connection arrives (i.e., a connection destined directly for one of the sockets **h323-proxy** is listening on) and is allowed by policy, either the proxy must be configured to connect to a specific remote server by the `plug-to` configuration directive (see below) or the caller must identify destination in H.245 Setup packet.

If a `plug-to` directive is used for a transparent connection, it has precedence over the original destination. This means that **h323-proxy** will ignore the original destination and connect to the `plug-to` server.

Common Kernun Features

The proxy uses common Kernun mechanism for listening on its sockets, forking new processes as needed and killing old redundant processes, optionally changing root directory and running with

alternative user privileges. For more detailed information, see application(5) and tcpserver(7).

The proxy uses common Kernun mechanism for network input/output operations. Configuration allows for specifying several parameters like buffer sizes and timeouts, for client and server connections and multimedia data channels. They can be included in `client-ctrl`, `server-ctrl` and `data-channel` configuration sections, respectively. For more detailed information, see netio(7).

The proxy uses common Kernun mechanism for name resolving (see resolving(7)).

The proxy uses common Kernun mechanism for logging (see logging(7)). When a connection comes to the proxy, the SESSION-START (MMCP-808-I) and the ACL (MMCP-810-I) messages are logged. After successfull decision about the server being connected, the SESSION-INIT (MMCC-808-I) message is logged. After closing the session, the SESSION-END (MMCP-809-I) message is logged.

The proxy uses common Kernun mechanism for traffic shaping. For more detailed information, see traffic-shaping(7).

Program Options

The program options are as follows:

-h Print usage information and exit.

-v Display version information and exit.

-d *dbglev* Set debug level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

GATEKEEPER MODES

Kernun H.323 Proxy supports two modes how clients can use gatekeepers. In both cases, the gk-proxy(8) must be run and two special configuration items must be set:

map-file Specifies the name of the file, where **gk-proxy** stores information about registered clients.

client-block Specifies what address and port block is to be used for incoming connections. The proxy must listen on the port specified and all other ports from consequent block of given size are redirected by the proxy by means of NAT.

Open gatekeeper mode

In this case, clients use a gatekeeper without authentication and the **gk-proxy** can modify content of RAS packets so that instead of clients' private addresses only firewall's public address appears on the net. The **gk-proxy** uses for each client separate port from the dedicated block and the **h323-proxy** must manage NAT rules for these ports. This is the meaning of the `client-block`

configuration item. Address must be set identically in both proxies and size defined for **h323-proxy** must be at least as large as **gk-proxy** connection table size. Additionally, the **h323-proxy** must listen also on one special port, to which **gk-proxy** redirects sessions requested by local clients and destined to outside world.

Example:

```
gk-proxy open-gk {
    listen-on {
        transparent fw-int : 1719;
    }
    udpserver { max-sessions 10; }
    map-file  "/tmp/ras.yp";

    session-acl open-gk {
        to transparent open-gk.abc.com;
        register       fw-ext : 40000;
        h323-address   loopback : 3420;
    }
}

h323-proxy mm-proxy {
    listen-on {
        non-transparent loopback : 3420;
        non-transparent fw-ext : 40000;
    }
    map-file "/tmp/ras.yp";
    client-block fw-ext : 40000 10;

    session-acl outgoing {
        from [192.168.0.0/16];
        to transparent *;
    }

    session-acl gk-served {
        to non-transparent *;
        source-address client;
    }
}
```

Authenticated gatekeeper mode

When clients use a gatekeeper with authentication the **gk-proxy** cannot modify content of RAS packets because authentication would fail. That's why it uses *client* mode of registration (unfortu-

nately, private addresses occur on the net). In this case, an H.323 connection is always initiated by local client, so only the **h323-proxy** special port is needed, to which the **gk-proxy** redirects (by means of NAT) all connections. That's the reason for apparent discrepancy between `listen-on` (non-transparent) and `session-acl` (transparent) directives.

Example:

```
gk-proxy auth-gk {
    listen-on {
        transparent fw-int : 1719;
    }
    udpserver { max-sessions 10; }
    map-file  "/tmp/ras.yp";

    session-acl auth-gk {
        to transparent auth-gk.abc.com;
        register       client;
        h323-address   loopback : 3420;
    }
}

h323-proxy mm-proxy {
    listen-on {
        non-transparent loopback : 3420;
    }

    session-acl outgoing {
        from [192.168.0.0/16];
        to transparent *;
    }
}
```

SEE ALSO

h323-proxy.cfg(5), gk-proxy.cfg(5), listen-on(5), application(5), test-expr(5), MMCP-808(6), MMCP-810(6), MMCC-808(6), MMCP-809(6), access-control(7), configuration(7), host-matching(7), logging(7), netio(7), resolving(7), tcpserver(7), time-matching(7), traffic-shaping(7), transparency(7), gk-proxy(8)

NAME

http-proxy, test-http — HyperText Transfer Protocol (HTTP) proxy

SYNOPSIS

http-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-http [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

Program **http-proxy** is the proxy daemon for HyperText Transfer Protocol (RFCs 1945, 2616). It supports HTTP versions 0.9, 1.0, and 1.1 clients and HTTP 1.0 and 1.1 servers. The proxy supports secure communication via SSL/TLS protocols, see ssl(5).

Startup and Configuration

The proxy reads its configuration and starts listening on TCP sockets (address/port couples) specified by `listen-on` configuration section, see listen-on(5). If support of transparent connections (i.e., connections made directly to a HTTP server and redirected to the proxy by IP Filter as described in transparency(7)) is requested by item `transparent` in section `listen-on`, the corresponding NAT redirections are established during proxy startup and removed upon exit.

Format of the configuration file is described in http-proxy.cfg(5). General syntax of Kernun configuration files is explained in configuration(7). Program **test-http** tests syntax and partially semantics of configuration; for test expression syntax, see test-expr(5).

Access Control

Http-proxy uses three-phase ACLs, see access-control(7). The first phase, `session-acl` is checked once for each client connection. It permits or denies client access and sets some connection parameters. The second phase, `request-acl` is checked for each HTTP request after the request headers are received from the client, but before anything is sent to the server. It decides about permitting/denying the request and it can also set some request parameters. Note that there can be several requests per connection if persistent connections are used. The third phase, `doc-acl` is checked for each HTTP request after the response headers are received from the server, but before the response is sent to the client.

Connection Establishment

When a connection from an HTTP client arrives, the configuration is searched for a matching `session-acl`. If the ACL says that the connection should be denied or there is no matching ACL, the proxy does not communicate with the client and closes the connection immediately. In addition to the generic ACL conditions and actions described in access-control(7), some **Http-proxy**-specific conditions and parameters can be set.

If the proxy sends a response to the client, but the client is still sending some data, it may be necessary not to close the read side of the connection for some time (see RFC2616 sect. 10.4 for details). Configuration item `linger-time` sets the time which the proxy should wait before closing the read side of the client connection.

It is possible to set idle timeouts for request and response by item `idle-timeout`. If no data are received from the client or the server for more than `idle-timeout` seconds, the request fails.

Items `client-keepalive` and `server-keepalive` define whether persistent connections are to be used on the client and server sides of the proxy. It is possible to limit the number of requests per connection or to disable persistent connections entirely. Timeouts for closing an idle connection can be set too.

The language used by the proxy in error messages sent to the client is defined by item `language`.

It is possible to forward all requests to some other proxy instead of sending them directly to origin servers. The next-hop proxy is set by `hand-off`.

Item `ssl` switches on SSL/TLS on the client connection and sets various SSL/TLS parameters. If the connection from the client uses SSL/TLS, item `client-cert-match` defines the acceptable client certificates. If the client certificate does not pass the test, SSL/TLS connection establishment fails and the connection is closed.

When **http-proxy** is used as an authentication proxy for accessing protected HTTP servers, parameters of the client authentication are set by item `aproxy`.

Request Processing

For each request from the client, the proxy reads the request line and headers. Then it finds the appropriate `request-acl`. If the ACL says that the request should be denied, the user should authorize, or there is no matching ACL, the proxy sends back a reply informing the user that the request has been denied. In addition to the generic ACL conditions and actions described in access-control(7), many **Http-proxy**-specific conditions and parameters can be set.

Items `request-uri`, `request-method`, `request-scheme`, `request-path`, and `request-version` match values from the first line of the request. Note that the server hostname and port is matched by the standard ACL condition `server`. In a non-transparent proxy, the server hostname and port for matching both `request-uri` and `server` are taken from the request URI received from the client. If `request-acl.host-hdr-transp` is set, the Host header is used instead. This allows a request URI without a server name in a non-transparent request, which can occur if a request is transparently redirected differently than by the Kernun's own transparency. In a transparent proxy, the server hostname and port are taken from the Host request header. If Host is missing in the request (possible in HTTP/1.0) then the original destination IP address and port of the connection from the client is used instead. The server address and port used for matching is also the address where the proxy connects to. Especially, in a transparent proxy this may differ from the original destination address used by the client. Combination of a server and an initial part of path can be matched against a blacklist using `blacklist` item. The name of the blacklist database file is specified by `blacklist-db`. Utilities for working with blacklists are mkblacklist(1), printblacklist(1), and

resolveblacklist(1). The set of categories assigned to the request URI can be matched by item `clear-web-db`.

Sometimes different settings are needed for some types of clients due to various errors and incompatibilities in web browsers. Therefore, the value of User-Agent HTTP header may be used for ACL matching (item `user-agent`).

If the client connection uses SSL/TLS, the values from the client certificate are compared to item `client-cert-match` during ACL matching. An ACL with `client-cert-match` is never used for plaintext HTTP connections.

If `aproxy` is configured in the `session-acl`, it is possible to use item `aproxy-user` in order to match the user and the group authenticated by AProxy. If `aproxy` is not configured, `aproxy-user none` matches.

It is possible to change the whole request URI (e.g., http://www.tns.cz/index.html). The URI is matched with a regular expression. A matching URI can be rewritten to some other string, as defined by item `rewrite`. The request processing continues with the current ACL, even if the new URI does not satisfy the conditions of this ACL, because `request-acl` matching is done only once and with the original URI. It is possible to specify a `redirect permanent` or `redirect temporary` in a `rewrite`. Then the proxy will not fetch the rewritten URI, but it will return a HTTP redirect response (status code 301 or 302) to the client.

Item `plug-to` changes the server address which the proxy connects to, but it does not change the content of Host header. It is possible to change Host header (and also the server in request URI in the case of `hand-off`) by `http-host`.

Items `hand-off` and `language` allow to overwrite the values from `session-acl` for a single request.

Items `file-response` and `program-response` generate a response locally by the proxy. Unlike the `replace-response` item in `doc-acl` described below, the proxy does not contact the origin server and generates the response immediately. See Section D for details about the proxy-to-program interface.

Item `select-optimization` influences internal handling of network communication. The proxy repeatedly checks its client and server network connections for a possibility to read or write data. When a connection is ready, a piece of data is sent or received. It is more efficient to try several send or receive operations on the connections that have been ready recently before checking all existing connections again. The number of such retries is controlled by `select-optimization`. It may improve the proxy performance if set to a small positive value, for example 10.

Request and response headers may be filtered by items `allow-req-hdr` and `allow-resp-hdr`. These items define names of headers that will be passed by the proxy. All other headers will be deleted from the request or response. It is also possible to reject requests with request, status, and header lines not matching items `req-line-check`, `req-hdr-check`, `status-line-check`, and `resp-hdr-check`.

The proxy may add Via HTTP headers to requests and responses. These headers inform about proxies which the request or response passed through. It may be useful to track problems with a proxy, but sometimes the administrator wants to hide information that the proxy is present. Hence the use of Via headers may be configured by items `request-via` and `response-via`.

Item `request-time` places an upper limit on the request handling time. It eliminates stuck

clients and servers and help against some DoS attacks.

Item `auth-req` causes that the proxy responds with 407 Proxy Authentication Required and sets the authentication realm sent to the client.

Maximum amount of data transferred between the client and the server may be limited by item `max-bytes` separately for each direction (client to server and server to client). Data filtration can change the size of data significantly, therefore the limits are set separately for client and server connections.

In some situations, e.g., when chunked transfer encoding is in use, the proxy buffers incoming data and sends them only when the buffer is full. There are applications which require all data to be forwarded immediately. For such situations, item `flush` switches buffering off.

Item `ssl` switches on SSL/TLS on the server connection and sets various SSL/TLS parameters. If the connection to the server uses SSL/TLS, item `server-cert-match` defines the acceptable server certificates. If the server certificate does not pass the test, SSL/TLS connection establishment fails and the request terminates with an error.

After `request-acl` is processed, the request is forwarded to a server. When the servers answers with a status line and response headers, the proxy finds the appropriate `doc-acl`. In addition to the generic ACL conditions and actions described in access-control(7), some **Http-proxy**-specific conditions and parameters can be set.

Items `request-scheme`, `request-path`, and `blacklist` have the same meaning as in `request-acl`. Item `mime-type` provides matching of response document content type. The proxy provides three methods of detecting the content type: `content-type` (from the Content-Type response header), `extension` (matching request URI suffix with information from *mime-types*), and `magic` (guessing the type from an initial part of response data, using the same algorithm as in the standard utility *file*). Selection and priorities of the methods are defined by `http-proxy.doctype-identification`, `session-acl.doctype-ident-order`, and `request-acl.doctype-ident-order`. The first successful method defines the type. If no method succeeds, the type will be represented by an empty string. Maximum size of data scanned by `magic` method can be changed by `http-proxy.doctype-identification.magic`.

The content type, i.e., the Content-Type header sent to the client, can be forced by `set-mime-type`, or set to the content type discovered for `mime-type` matching by `force-doctype-ident`. Otherwise, the header is left unchanged.

It is possible to discard some responses and to replace them with a local file. Item `replace-response` defines this replacement.

GIF, JPEG, and PNG images may be filtered according to the image dimensions. A local image is returned to the client instead. The image substitution is defined by `filter-images`. This feature can be used, for example, to filter advertisement banners, because thy often have known characteristic dimensions. Dimensions of GIF and PNG images are stored at fixed offset near the beginning of the respective files, but JPG dimensions may be far from the file beginning. Item `jpeg-scan-sz` restrict the size of initial part of JPEG files scanned for dimensions.

Http-proxy can filter data through an antivirus. Antivirus checking is defined by item `antivirus` which selects a top-level `antivirus` section. See antivirus(5) for details about configuration of virus checking.

Http-proxy provides HTML filtering. It is usually used to delete potentially dangerous

parts of HTML data passed to client, e.g., scripts or Java applets. Features of the HTML filter are controlled by item `html-filter`, which selects a top-level `html-filter` section. See mod-html-filter(5) for details about configuration of HTML filtration.

HTTP request and response data can be processed and actions can be taken accordingly. Matching in request data is configured by item `request-acl.request-body-match`, response data matching is controlled by `doc-acl.response-body-match`. See data-matching(7) for detailed description of the data matching and processing feature.

The maximum size of the HTTP request body can be limited by setting `request-acl.request-body-max-size`.

If the request URI is categorized by `clear-web-db`, Bypass function can be enabled by `clear-web-db-bypass`. When accessing a matching page, the user gets an error page. By clicking a link on the page, access is enabled for a limited time to the blocked web server, or all servers belonging to the categories specified by `clear-web-db-match`.

Using CONNECT Method

HTTP method CONNECT is reserved for tunneling other protocols through **http-proxy**. It is usually utilized for SSL/TLS access to HTTPS servers. When a user sets its browser to use the proxy in the nontransparent mode, an HTTPS request causes a CONNECT request to be sent to the proxy. The proxy then creates a tunnel between the client and the server.

Note that data passed through the tunnel are encrypted and thus inaccessible to the proxy. After the tunnel is established, it is not possible to deny any HTTPS requests nor to perform data checking like HTML filtering or antivirus testing. It is therefore appropriate to limit the servers accessible via CONNECT.

In a transparent proxy configuration and HTTPS, the client does not use CONNECT, but it starts SSL/TLS immediately after establishing a TCP connection. Although it is possible to utilize **http-proxy** in this case (see the description of `session-acl.simulate-connect` below), it is usually easier to use tcp-proxy(8). An exception that requires **http-proxy** is when some HTTPS connections should be just passed via a TCP tunnel, but other should be decrypted by the proxy.

An alternative to a simple HTTPS tunneling is to use SSL/TLS decryption/encryption functionality of the proxy. A transparent **http-proxy** — which does not use CONNECT — can be configured to decrypt the connection from a client (by `session-acl.ssl`), process the encapsulated HTTP, and encrypt the connection to a server (by `request-acl.ssl`). A more complicated situation arises in the non-transparent mode. As mentioned above, the browser tries to establish a TCP tunnel through the proxy using the CONNECT method. If the `request-acl` contains item `capture-connect`, the proxy captures the CONNECT request, that is, it responds to the request as if the tunnel was established, but does not open the connection to the server. Instead, it restarts the session in transparent mode. End of the CONNECT request and session is logged, and a new session is started, which behaves as a transparent session to the server specified in the CONNECT. New `session-acl` and `request-acl` are selected that can, among other things, enable SSL/TLS decryption and encryption, in the same way as in a normal transparent **http-proxy** configuration. To be chosen for a new session emerged from a captured CONNECT request, a `session-acl` must contain item `captured-connect`. The `session-acl` selection

can be based on the ACLs used for handling the CONNECT. Those ACLs are specified by items `connect-session-acl` and `connect-request-acl`.

If some connections should be decrypted and re-encrypted, but other ones are to be just passed, it is possible to set `simulate-connect` in a `session-acl` matching connection that will not be decrypted. This option behaves as if the data from the client were preceded by a CONNECT request to the destination address of the TCP connection from the client. That is, the proxy just establishes a tunnel and passes data unmodified between the client and the server. The proxy must learn the server address somehow, hence `simulate-connect` requires either a transparent proxy mode, or a `plug-to` item specifying the server address explicitly in an ACL.

The last option is to perform full inspection of the HTTPS. In this case, the **http-proxy** interrupts the initial phase of establishing the SSL connection from the client, it tries to contact the server and to get its certificate. If it fails, the connection to client will be reset. If the server is connected and its certificate is verified, the proxy generates a new certificate with all attributes (except some unwanted ones) from the original server's one, subscribes it by own certifiate authority and uses this new certificate for completing the connection to the client. If the original server certificate cannot be verified, then several options are available:

`error` The new certificate is signed by proper Kernun CA certificate and after establishing the client connection, an error message is sent as a reply.

`pass` The new certificate is signed by a special Kernun CA certificate which is intended not to be added among client's trusted key ring. Thus, the user gets a warning from the browser and he or she can decide how to continue.

`fail` The connection establishing fails.

`ignore` The verification failure is ignored. *Highly unrecommended option!*

The new certificate is stored in the cache (a file in the `/data/fake-cert` directory) for later re-using. Correct certificates have names starting by the `C` letter followed by the certificate hash and distinguishing number. Wrong certificates (used in the `pass` case) have the `F` letter on the beginning, instead. See the ssl(5) manual page for further details.

Using FTP Scheme

When the client uses the proxy in the nontrasparent mode and the user requests data from an FTP server by entering a URL starting with ftp:, the client sends an HTTP request with that URL to the proxy. The proxy is then expected to fetch the document from an FTP server and return it as an HTTP response to the client. **Http-proxy** does not communicate directly with FTP servers. Instead, it asks ftp-proxy(8) for doing the work. Communication between **http-proxy** and ftp-proxy is done in a private protocol created specially for Kernun firewall. Parameters needed for connecting to ftp-proxy are specified by item `ftp-proxy`.

When the firewall works in the transparent mode, HTTP clients talk directly to FTP servers. Appropriately configured ftp-proxy is needed in such situation.

User Authentication

User authentication on proxies works in HTTP in a similar way as authentication on origin servers. The difference is in status codes (407 instead of 401) and headers (Proxy-Authenticate and Proxy-Authorization instead of WWW-Authenticate and Authorization). When the proxy requires authentication and a request does not contain valid credentials, the proxy replies with 407 and sends an authentication method and a realm to the client in header Proxy-Authenticate. The client then obtains user's credentials and repeats the request with them in header Proxy-Authorization. The credentials are sent automatically in all subsequent requests. Only the Basic, Kerberos (Negotiate), and NTLM authentication schemes are supported by **http-proxy**. The proxy can be configured for one or both of them. If both authentication schemes are enabled, a client can choose which scheme it will use. Typically, Kerberos/NTLM-capable web browsers will use Kerberos/NTLM, other browsers will use Basic.

Basic Authentication

In order to enable user authentication, item `auth` must be present in `session-acl`, see auth(5) and auth(7). It defines authentication database (for example, a file or a RADIUS server) which will verify credentials from users. All of the authentication methods mentioned in the man page auth(7) are supported in **Http-proxy**. The item `user` is used to match user names in `request-acl` and `doc-acl`. A user name is matched if it is present with a valid password and is successfully verified. Otherwise, `user none` is matched.

A typical setting of user authentication involves at least two request ACLs. One is for permitting access to the authenticated users, the other one denies access, sends a realm, and asks for credentials. Example:

```
# Switch checking credentials on and choose user database.
session-acl SET-AUTH {
    auth passwd "/usr/local/kernun/etc/passwd";
}
# Permit any successfully authenticated user.
request-acl OK {
    user *;
}
# Not authenticated, ask for credentials.
request-acl ASK-AUTH {
    user none;
    auth-req "Kernun http-proxy";
}
```

Kerberos Authentication

Kerberos authentication is intended primarily for Active Directory environment, although it can be used with any Kerberos server. When using Kerberos authentication, the proxy obtains the user name from the Kerberos ticket received from the client, but the ticket does not contain information

about group membership. The list of groups, which is usable in `request-acl.user` matching, can be obtained via LDAP.

Kerberos authentication is enabled by item `kerberos-auth` in a `session-acl`. It references a section `kerberos-auth` on the `system` level. The section specifies the Active Directory domain name and the domain controller address. In the case of a generic Kerberos, not being in an Active Directory environment, the Kerberos realm is defined by `domain` and the Kerberos ticket granting server by `ad-controller`. The `kerberos-auth` section can reference an LDAP server by item `ldap`. As the Active Directory controller contains group membership data and provides LDAP access, it is typically used also as the LDAP server.

Kerberos can be also utilized to authenticate LDAP requests by adding `kerberos` instead of `bindinfo` into an `ldap-client-auth` section. Then the proxy authenticates itself (obtains a TGT) upon startup using the machine account of the Kernun system in the Active Directory. Hence the machine account must have enough rights to read user group information from the Active Directory database.

As in Basic authentication, at least two request ACLs are used for Kerberos authentication. One of them permits access to authenticated users, the other one denies access and requests authentication. Example:

```
system ... {
    # Active Directory controller used as an LDAP server
    ldap-client-auth LDAP-AD {
        server "ldap://ad.tns.cz";
        # Authenticate to LDAP using Kerberos
        kerberos;
        active-directory "tns.cz";
    }
        # Kerberos authentication by the Active Directory Controller
    kerberos-auth KERBEROS {
        domain "TNS.CZ";
        ad-controller "ad.tns.cz";
        ldap LDAP-AD;
    }
    http-proxy HTTP {
        ...
        session-acl AUTH {
            accept;
            auth none;
            kerberos-auth KERBEROS;
        }
        ...
        request-acl KERBEROS-OK {
            user *;
            accept;
        }
```

```
        request-acl KERBEROS-ASK {
            user none;
            accept;
            auth-req "Kernun http-proxy";
        }
        ...
    }
}
```

After applying the Kerberos authentication configuration for the first time, the Kernun system must become a member of the Active Directory domain. Its machine account is created by the shell command

kinit *user*
msktutil -c –computer-name ‘hostname -s‘ -s HTTP/‘hostname‘ \
–server *ADC* –no-pac
chown kernun /etc/krb5.keytab

where *user* is a user with `Domain Admins` rights and *ADC* is the address of the Active Directory Controller. If the system is to be removed from the domain later (when Kerberos authentication is no more required or if the system will be moved to another domain), remove file `/etc/krb5.keytab` and delete the machine account on the Active Directory Controller.

A proxy with Kerberos authentication enabled needs access to the Kerberos configuration files `/etc/krb5.conf` and `/etc/krb5.keytab`. Hence, the proxy cannot be run chrooted unless the chroot environment is appropriately extended.

Group membership information of users authenticated by Kerberos can be cached in order to decrease load of the LDAP server. Configuration of caching consists of adding the global section `oob-auth OOB` and referencing it by item `http-proxy.oob-auth-srv`. Cached group membership information for a user name expires after a timeout controlled by items `kerberos-auth.timeout-idle` (expiration after a period of inactivity) and `kerberos-auth.timeout-unauth` (unconditional expiration).

More details about Kerberos authentication can be found in the Kernun Handbook.

NTLM Authentication

The NTLM authentication is enabled by item `ntlm-auth` in a `session-acl`. It references a section `ntlm-auth` on the `system` level. The section specifies the Active Directory domain name and the domain controller address. The proxy obtains the user name from the NTLM authentication process, but it does not get any information about group membership. The list of groups, which is usable in `request-acl.user` matching, can be obtained via LDAP. The `ntlm-auth` section can therefore reference a LDAP server by item `ldap`. As the Active Directory controller contains group membership data and provides LDAP access, it is typically used also as the LDAP server.

As in Basic authentication, at least two request ACLs are used for NTLM authentication. One of them permits access to authenticated users, the other one denies access and requests authentication. Example:

system ... {

```
    # Active Directory controller used as a LDAP server
    ldap-client-auth LDAP-AD {
        server "ldap://ad.tns.cz";
        bindinfo "cn=ADUser,dc=tns,dc=cz" "ldap-password";
        active-directory "tns.cz";
    }
    # NTLM authentication by the Active Directory Controller
    ntlm-auth NTLM {
        domain "tns.cz";
        ad-controller "ad.tns.cz";
        ldap LDAP-AD;
    }
    http-proxy HTTP {
        ...
        session-acl AUTH {
            accept;
            auth none;
            ntlm-auth NTLM;
        }
        ...
        request-acl NTLM-OK {
            user *;
            accept;
        }
        request-acl NTLM-ASK {
            user none;
            accept;
            auth-req "Kernun http-proxy";
        }
        ...
    }
}
```

After applying the NTLM authentication configuration for the first time, the Kernun system must become a member of the Active Directory domain. It is done by issuing the shell command
net ads join -U *user*

where *user* is a user with `Domain Admins` rights, and rebooting the system. If the system is to be removed from the domain later (when NTLM authentication is no more required or if the system will be moved to another domain), it can be done by the command
net ads leave -U *user*

A proxy with NTLM authentication enabled needs access to the utility **ntlm auth**(1), which in turn accesses contents of directory `/var/db/samba/winbindd_privileged`. Hence, the proxy cannot be run chrooted unless the chroot environment is appropriately extended.

Results of NTLM authentication can be cached by out-of-band authentication, in order to decrease load of Active Directory and LDAP servers. Each new client is authenticated by NTLM. The combination of the client IP address, the user name and the list of groups is remembered in the OOB session table. Following requests from the same IP address will be authenticated as the same user and groups, without contacting the AD controller and the LDAP server.

Configuration of NTLM caching consists of adding the global section `oob-auth OOB`, referencing it by item `http-proxy.oob-auth-srv`, and adding `auth oob OOB` to each `session-acl` that contains item `ntlm-auth`. Cached user and group information for a client IP address expires after a timeout controlled by items `ntlm-auth.timeout-idle` (expiration after a period of inactivity) and `ntlm-auth.timeout-unauth` (unconditional expiration).

Combined Authentication Methods

In order to support clients incapable of NTLM authentication, it is possible to enable both authentication schemes by configuring the NTLM authentication and simultaneously using item `auth` in `session-acl` with a method other than `none`. The above NTLM example can be modified by simply changing `auth none` to `auth passwd "..."`.

Cookie Modification

The proxy can be cofigured to perform modification of cookies passed between a client and a server. The value of a cookie received from a server is replaced by a new value and passed to the client. If the client sends the cookie back to the server, the proxy restores its original value before passing it to the server.

This feature reduces exploitability of stolen cookies, especially session-identification cookies in various web applications. A cookie stolen from the client is useless outside the network protected by Kernun, because its value is not that expected by the server. Even inside the protected network, a stolen cookie has only limited potential of misuse, because after the proxy sends a cookie to a client, it accepts it back only from the same client IP address.

The proxy maintains a cookie table that is use for restoring modified values of cookies passed from a client to a server. To increase security, neither the modified cookie value passed to the client, nor the related record in the cookie table suffices for restoring the original cookie value. The two pieces of information must be put together in order to reverse the cookie modification operation.

Properties of the cookie table (file name, size, expiration, and cleaning rule) are set in section `cookie-table`. Rules for cookie modification are defined by items `request-acl.modify-cookies`. It is possible to modify only some cookies, selected by name, disable checking of client IP address by flag `any-client`, and decide whether cookie values sent by a client to a server and not found in the cookie table should be passed unchanged (flag `keep-not-found`) or replaced with an empty value. A request that uses a `request-acl` with item `delete-cookies` causes deleting all cookies related to a single IP address. Either the IP address of the requesting client, or the IP address contained (in standard textual notation) in the query part of the request URI, is used, according to flag `ip-from-query`.

Authentication Proxy (AProxy)

It is possible to configure **http-proxy** for providing access from an external network to some web server in the internal protected network. Often requirements in such configuration are encryption of the communication between the client and the proxy and using challenge-response authentication. Module AProxy of **http-proxy** provides this functionality. If a user is not authenticated, the proxy returns an authentication form instead of a normal response. When the user authenticates, the response for the original request is returned and further requests are processed normally until the user logs out or a timeout expires.

AProxy mode is switched on by item `aproxy` in `session-acl`. It is advisable to turn on SSL/TLS between clients and the proxy by item `ssl` in `session-acl`. Configuration section `aproxy` sets various AProxy parameters. Section `auth` defines AProxy authentication database. Username/password authentication is supported for both *passwd* and *radius*, challenge/response authentication may be used only with *radius*. User and group names obtained during AProxy authentication are matched against `request-acl.aproxy-user` condition.

The proxy identifies sessions belonging to authenticated users by cookies. It is necessary to choose a `cookie-name` so that it does not collide with cookies used by the origin server. The maximum number of simultaneously active user sessions is specified by `max-aproxy-sessions`. If `insecure-cookie` is not set, the client is asked not to send the session cookie across an unencrypted connection. It prevents possible revealing of the cookie when the user inadvertently enters http: instead https: into the browser.

Out of Band Authentication Server

Http-proxy is used also as an OOB authentication server, see auth(7). In this mode, the proxy manages the list of OOB authenticated users and provides the list to other proxies. OOB authentication server is turned on by a section `aproxy` containing item `oob-auth`. Parameters of the OOB authentication are set by a section `oob-auth` referenced by `http-proxy.oob-auth-srv`. OOB authentication uses either the `html-form` method (users authenticate themselves by filling the same form as in AProxy authentication) or the `external` method (the list of users is provided by an external program, e.g., ooba-samba(1), which passes it via HTTP to the authentication server).

Web Filter

The request URI can be processed by an external web filter. Interface to *IBM Proventia Web Filter* is implemented in the proxy. The web filter has a regularly updated database of web servers. It takes a request URI from **http-proxy** and assigns a set of categories to it (for example, pornography, games, lifestyle, criminal activities). Then it processes the categories together with client IP address and user name (if proxy authentication is enabled) and decides according to its ruleset whether the URI should be accepted or rejected. If the web filter accepts the URI, request processing continues in **http-proxy**. Otherwise, the proxy returns an error page to the client.

In the web filter configuration, *ICAP Integration* must be enabled (in *Proxy Integration* dialog of the management console). Also select *User Profile Support* in this dialog. In the Kernun

configuration, section `web-filter` contains parameters of a connection to a web filter. Processing a request URI by the web filter is enabled by item `request-acl.web-filter`.

IBM Proventia Web Filter requires user names in the form *`domain\user`*. The **http-proxy** uses always domain name `kernun`. Therefore, user names in web filter configuration must be entered as `kernun\`*`user`*.

Program-Generated Responses

If item `request-acl.program-response` is set in the configuration, HTTP requests from clients are processed by an external program specified in this configuration item. A new instance of the program is started for each request. The complete HTTP request as received from the client is passed to the standard input of the program. The program must reply with a valid HTTP response written to its standard output and terminate. The proxy then sends the response back to the client. If the program does not terminate until a configured timeout or the request processing is interrupted before the program terminates, the proxy sends the `SIGTERM` signal to the running program.

In addition to the HTTP request on the standard input, the program is also provided with a set of environment variables:

APROXY_USER User name from the AProxy authentication

CONTENT_LENGTH Size in bytes of the request body. Word `chunked` means that request body uses the chunked transfer encoding.

CONTEXT Context which the program is executed in. It can be `program-response` for a program executed via `request-acl.program-response`, or one of `request-end-program-ACCEPTED`, `request-end-program-REJECTED`, `request-end-program-FAILED` for a program executed by `request-acl.request-end-program` (as described in the next section).

DOC_ACL Name of the `doc-acl` used for this request or the empty string if no `doc-acl` has been selected. Note that in the case of a `program-response`, no `doc-acl` is used.

HTML_REPLACE_HASH If request data have been matched by a `request-acl.request-body-match` rule with type `html-replace`, this variable contains a hash value computed fro the matching HTML form values. Otherwise, the variable contains the empty string. In fact, this variable can be also set by response data mtching a `doc-acl.response-body-match` rule with type `html-replace`, but HTML form data are usually not sent and matched in HTTP responses.

LOG_FILE The name of the file used by the proxy for logging, or the empty string if the proxy logs via syslog.

LOG_LEVEL The current numeric log level of the proxy.

PATH_INFO This is the path part of the request URI, without the query part.

PROXY_NAME The name of the proxy as specified in the configuration.

QUERY_STRING Contains the query part of the request URI, without the initial question mark delimiting it from the path.

REMOTE_ADDR IP address of the client

REMOTE_HOST Host name of the client if known, empty otherwise

REMOTE_USER User name if the user was authenticated by the proxy.

REQUEST_ACL Name of the `request-acl` used for this request.

REQUEST_HOST The host part of the request URI.

REQUEST_METHOD The HTTP request method as specified by the client in the request

REQUEST_URI The complete request URI.

SESSION-ACL Name of the `session-acl` used for this request.

Note that although this program interface resembles the CGI commonly used by WWW servers, it does not comply to the CGI definition in RFC 3875.

Running a Program at the End of Request

Item `request-acl.request-end-program` enables running an external program at the end of each request. The proxy does not wait for termination of the program. The program gets information about the request in the same set of environment variables as a program for generating responses described in the previous section. The suffix of the `CONTEXT` variable value corresponds to the request processing result as reported in the `REQUEST-END` log message.

Logging

As all other Kernun proxies, **http-proxy** generates many log messages during its operation. Meaning of the messages may be found in section 6 of the manual pages. Details about Kernun logging can be found in logging(7).

The proxy logs statistical messages about each client connection and each request. When a connection arrives, SESSION-START is logged. Then ACL message informs about the session ACL selected for this connection. Each request generates REQUEST-START (when the request line and headers are received from the client), ACL (selection of a request ACL), and REQUEST-END (at the end of request processing). Finally, SESSION-END is logged when the client connection is closed. If AProxy is enabled, login and logout of each user is reported as an APROXY-AUTH message.

Common Kernun Features

Http-proxy uses common Kernun mechanisms for listening on its sockets, accepting client connections, and managing its processes. It can also run in a chrooted environment and change its user identity upon startup. See also application(5), tcpserver(5), and tcpserver(7).

The proxy uses a common Kernun mechanism for network input/output. The configuration allows to specify several parameters like buffer sizes and timeouts, both for client and server connections. The parameters are set in configuration sections `client-conn` and `server-conn`. See netio(7) for details.

The proxy uses common Kernun mechanism for name resolving (see resolving(7) manual page).

Http-proxy uses common Kernun mechanism for runtime monitoring. For more detailed information, see monitoring(7).

Http-proxy uses common Kernun mechanism for traffic shaping. For more detailed information, see traffic-shaping(7).

OPTIONS

-h Display usage information and exit.

-v Print version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read configuration from *cfgfile*.

-r Resolve names in configuration prior to testing.

-t *test_expr* Test configuration according to given expression. Format of the *test expr* is described in test-expr(5).

DOCUMENT TEMPLATES

Two kinds of errors are generated by the **http-proxy**: hard and soft. A hard error is such a state of the proxy when the only possible reaction is to close (reset) the connection to the client immediately. A soft error means that the state of the protocol and the nature of the error allow to send an error response describing the error to the client. If a soft error occurs, the proxy sends a response (an error document) describing the error. The same mechanism is used for other responses generated locally by the proxy, e.g., FTP response for PUT method, AProxy authentication form and AProxy logout page.

The content type of a response document is text/html. The response document can be in various languages (UTF-8 charset), depending on the proxy configuration and client's preferences. Templates of response documents are stored in files named `document-root/class.html.`*language* where *document-root* is the root directory of error documents set in the proxy configuration file, *class* is a class of a response, and *language* distinguishes documents in various languages. Possible values for *language* are:

EN language English (en)

CZ language Czech (cs)

For each pair of response class and language there a is template file in the document root directory. The template is merely an HTML document possibly containing `$$`, `$-$`, and `$`*n*`$` (where *n* is a non-negative integer) directives. Each `$$` or `$0$` is substituted by a single `$` character. Directives `$-$`, `$1$`, `$2$`, etc. are replaced by substitution strings generated by the proxy. The substitution strings contain variable parts of a response document, which are specific for each response of a given class. Some substitution strings are common to all response classes. Additional ones may be defined for a particular class. The substitution strings are:

common for all classes

1 the HTTP status code of the response

2 the reason phrase corresponding to the status code

3 the request URI of the request (if the URI does not contain host, it is added from the Host header and if there is no Host header and the request is transparent, the real destination address is used)

4 the firewall administrator address taken from configuration (item `admin`)

5 the Kernun product type (UTM / Clear Web / Firewall+)

6 the session id in log format (altname[pid.session])

7 the request start date/time (%Y/%m/%d %H:%M:%S)

acl-deny

Error response when the request is denied by `request-acl` or `doc-acl`.

8 name of the ACL that denied access

9 the message specified by item `request-acl.deny-msg` or `doc-acl.deny-msg`

10 the client IP address and/or domain name

11 the user name (if authenticated)

12 the AProxy user name (if AProxy authentication used)

13 the list of categories assigned to the request URI by the Clear Web DataBase

14 the Clear Web DataBase categories assigned to the request URI matched by the `clear-web-db-match` item, that is, the intersection of 13 and 15

15 the Clear Web DataBase categories specified in the selected `request-acl` by item `clear-web-db-match`

The same set of substitution string is used also by the response classes `clear-web-db-deny` and by responses defined by `request-acl.deny-msg` and `doc-acl.deny-msg`.

clear-web-db-deny

Error response used when a request is denied by the Clear Web DataBase, that is, if a `request-acl` contains both items `clear-web-db-match` and `deny`.

bypass

The Clear Web DataBase Bypass activation page, returned if a `request-acl` contains both items `clear-web-db-match` and `clear-web-db-bypass`, and bypass has not been activated yet.

8 the list of categories assigned to the request URI by the Clear Web DataBase

9 bypass duration as set by `clear-web-db-bypass.duration`

10 the Clear Web DataBase categories specified in the selected `request-acl` by item `clear-web-db-match`

generic-error

Error response used when a soft error occurs. Description of the error is substituted for `$8$`.

cert-error

Error response used when a server presents an invalid certificate.

8 the certificate common name

9 the certificate issuer name

10 the certificate serial number

redirect

The response used when `request-acl.rewrite` contains a `redirect`. The redirection target URI is substituted for `$8$`.

ftp-response-put

Response for a PUT request with ftp: scheme. Result returned by ftp-proxy is substituted for `$8$`.

aproxy-password-form

AProxy form for entering user name and password.

8 error message generated by AProxy

10 AProxy cookie name

11 AProxy cookie value

12 original request method

13 encoded original request headers

$-$ encoded original request body

aproxy-response-form

AProxy form which displays a challenge and asks for a response.

8 error message generated by AProxy

9 AProxy authentication challenge

10 AProxy cookie name

11 AProxy cookie value

12 original request method

13 encoded original request headers

$-$ encoded original request body

aproxy-logout

A page with information that the user has been logged out by AProxy.

FILES

error documents Directory containing templates of error responses, FTP responses, AProxy forms, and local replacement documents; its real name and location is specified by configuration item `document-root`.

BUGS

The Kernun **http-proxy** is a security proxy, not a caching proxy. If caching of HTTP responses is needed, some caching HTTP proxy server can be chained using hand-off configuration directive or using a transparent redirection of requests.

HTTP/1.1 request pipelining is not supported. If the client sends pipelined requests, they are processed sequentially, as in the non-pipelined case.

SEE ALSO

Kernun: mkblacklist(1), ooba-samba(1), printblacklist(1), resolveblacklist(1), antivirus(5), application(5), auth(5), http-proxy.cfg(5), listen-on(5), mod-html-filter(5), ssl(5), tcpserver(5), test-expr(5), access-control(7), auth(7), configuration(7), data-matching(7), logging(7),

monitoring(7), netio(7), tcpserver(7), traffic-shaping(7), transparency(7), resolving(7), doctype-identification(7), ftp-proxy(8), tcp-proxy(8)

FreeBSD: ntlm auth(1)

NAME

icamd — Intra Cluster Accessibility Master Daemon

SYNOPSIS

service icamd { start| stop| restart| reload| status| hash }

DESCRIPTION

This daemon allows one or more Kernun system (slaves) to be controlled from one Kernun system (master). Each slave starts the complementary daemon icasd(8). The relation is asymetric: the master can control the slave(s). If two systems should be able to control each other, each of them must start both master daemon (icamd) and slave daemon (icasd).

When running, the icamd daemon waits for slaves to connect. When a slave connects, a backward SSH connection to the slave is established, which can be used for controlling the slave. The slave remains accessible until the slave icasd daemon is stopped or the connection is interrupted. The icasd tries to reconnect after interruption.

The ssh rsa key pair is used for icamd authentication. The icamd private key is the part of the icamd configuration. The icamd public key is part of the slave(s) configuration. Use ssh-keygen(1) for creating the ssh key pair.

Each slave is given a name in the icamd configuration. The ssh config file is provided which defines a host section for each configured slave. The ssh(1) can be used for connecting to the slave. For instance, for running the kat(8) the following command can be used:

```
ssh -t slave-name kat
```

Kernun GUI takes advantage of the connected slaves. It is possible to controll all connected slaves.

kat(8) takes advantage of the connected slaves. It is possible to apply the configuration remotely through the established icamd/icasd connection. If the name of the system being applied equals to the name of a connected slave, that slave connection is used for applying the configuration.

Commands

service icamd start Starts the daemon. The daemon listens for slave(s) connection from other systems according to the configuration.

service icamd stop Stops the daemon. The connected slave(s) (icasd) are disconnected. The default behavior of the slave is to retry the connection periodically. Therefore, they eventually connect automatically, when the icamd becomes started again.

service icamd restart

service icamd reload Stops and starts the icamd daemon.

service icamd status Prints the status of the icamd daemon. If running, all the configured slaves are listed with the information whether they are currently connected or not.

service icamd hash Prints the configuration hash.

Configuration

The icamd daemon is enabled in `rc.conf` with variable `icamd_enable="YES"`.

The configuration of the icamd daemon is in `/usr/local/kernun/etc/icamd.conf`. The following variables can be set in the configuration file:

MASTER_PORT The port for icamd to listen. This TCP port must be visible for the icasd slave for connection. The SSH protocol is used.

MASTER_ID_RSA

FN_MASTER_ID_RSA The private SSH key of the icamd daemon. Either the contents of the file, or the file name.

WRITE_CFG_HASH The file name where the hash should be written upon start (including restart, reload).

SLAVE_NAMES The list of slaves (space separated). For each slave *SL*, the following variables define each icasd slave:

SLAVE_ID_RSA_PUB_*SL*

FN_SLAVE_ID_RSA_PUB_*SL* The public SSH key of the icasd slave. When more than one isasd slave is configured for the master, they are distinguiseh by the SSH key each of them uses.

SLAVE_CFG_NAME_*SL* Optional. Defines the name for the slave *SL*. Use this, if the icasd name should differ from *SL* (*SL* may not contaion hypens (-)).

See ica(5) for the high level configuration in CML Kernun configuration.

SEE ALSO

Kernun: icasd(8), ica(5), cluster(7), configuration(7),

NAME

icap-server, test-icap — ICAP server for document inspection

SYNOPSIS

icap-server [-hv] [-d *dbglev*] -f *cfgfile*

test-icap [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

The **icap-server** is a server of the Internet Content Adaptation Protocol (RFC 3507) implementing the security policy for the access control and the document inspection based on the Kernun configuration logic.

The **test-icap** program checks the syntax and partially the semantics of the configuration; for test expression syntax, see test-expr(5).

Startup and Configuration

The server reads its configuration and starts listening on TCP sockets (address/port couples) specified by `listen-on` configuration section, see listen-on(5).

Format of the configuration file is described in icap-server.cfg(5). General syntax of Kernun configuration files is explained in configuration(7).

Access Control

The **icap-server** uses standard Kernun access control (see access-control(7)) with four types of ACLs on three levels:

1. `session-acl` (level 1) is checked once for each client connection and defines general protocol behavior, or rejecting the connection. In addition to the generic ACL conditions and actions, some **icap-server**-specific conditions and parameters can be set (see icap-server(5)).

2. `service-acl` (level 2I) is checked once for each ICAP request received and defines the service(s) and attributes used for the request. All entry conditions of this level are related to the ICAP request line and ICAP headers. Accepting ACLs cause 2xx ICAP response codes while denying ACLs cause 4xx and 5xx codes.

3. `request-acl` (level 2H) is checked once for each encapsulated (HTTP) request and/or document inspected and defines the behavior variation according to the HTTP request URI. All entry conditions of this level are related to the encapsulated HTTP request line, or client data sent by special ICAP headers X-Client-IP and X-Client-Username.

4. `doc-acl` (level 3) is checked once for each encapsulated (HTTP) document inspected and defines document processing mode (e.g. document type identification, filtering, replacing etc.).

The only exception from this rule is the case of antivirus checking with the `keepalive` option (see antivirus(7)). In this case the `doc-acl` is checked once more after the antivirus check is finished. If the `doc-acl` contains the `virus-status` item which corresponds with the final result, session continues. If the `doc-acl` contains the `virus-status` item which does not correspond with the final result, or it does not contain any `virus-status` ites, connection is reset.

Firewall administrator can choose any method described in auth(7) (except for NTLM) for authenticating users on the proxy.

Protocol Features

The recognized file type (see doctype-identification(7)) is returned to the client via the ICAP response header ("X-Kernun-Content-Type").

If the request URI is categorized by `clear-web-db`, the set of categories found are sent to the client via the ICAP response header ("X-Kernun-Categories").

In the case of request/document allowed by the policy and processed without any modification, the Kernun **icap-server** can return either the 200 response code (together with the original document) or the 204 response code (without data, if client permits it by the `Allow` header).

In the case of request/document refused by the policy (e.g. due to HTTP request attributes, file extension, recognized file type, virus found etc.), the ICAP response code has value 201 and either a standard or an own error web page is returned.

The "Preview" mode is supported. In this case, the client sends only a part of the document to the server, the server makes a decision and responds by the `100 Continue`, or some error response. After the `100 Continue` response, the client continues sending the rest of the document. The size of the preview block is recommended by the server to the client via the **OPTIONS** request response according to the services (antivirus, doctype recognition etc.) offered by corresponding `service-acl`. The admin can force another recommendation by the `preview` item.

Authentication

On the ICAP layer, the server uses similar authentication methods as the **http-proxy**. However, the HTTP layer of authentication is more important, here. This can be applied if ICAP clients use the X-Client-IP and X-Client-Username headers. The server expects authentication being done by the ICAP client and the resulting username is told to the server. Very often, the username contains also the domain name and the server can use also this piece of information.

The Kernun authentication methods concludes also checking of group membership. For obtaining the list of groups, to which particular user belongs, a LDAP server can be used (see the `service-acl.ldap-groups` item). For increasing of throughput, the **icap-server** can store received group memberships to a cache (the single one for all potential domains). Parameters of the cache are defined in the `ldap-cache` section and they have the same meaning as the ones from the `oob-auth` section. The only exception is the `timeout` item defining the lifetime of a record in the cache. If the section is omitted, no caching takes place.

OPTIONS

-h Print usage information.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

-r Resolve names in configuration prior to testing.

-t *test_expr* Test configuration according to given expression. Format of the *test expr* is described in test-expr(5).

SEE ALSO

antivirus(5), application(5), icap-server(5), icap-server.cfg(5), listen-on(5), mod-html-filter(5), ssl(5), tcpserver(5), test-expr(5), access-control(7), configuration(7), doctype-identification(7), host-matching(7), logging(7), netio(7), resolving(7), tcpserver(7), time-matching(7), traffic-shaping(7)

NAME

icasd — Intra Cluster Accessibility Slave Daemon

SYNOPSIS

service icasd { start| stop| restart| reload| status| hash }

DESCRIPTION

This daemon allows one or more Kernun system (masters) to control this Kernun system (slave). Each master starts the complementary daemon icamd(8). The relation is asymetric: the master can control the slave(s). If two systems should be able to control each other, each of them must start both master daemon (icamd) and slave daemon (icasd).

When started, the icasd daemon tries to connect to all the configured masters. If the connection to some master fails, the daemon retries to connect. The daemon tries to keep the connections to the masters established. When the connection to the master is established, the master can control the slave system.

The ssh rsa key pair is used for icasd authentication. The icasd private key is the part of the icasd configuration. The icasd public key is part of the master(s) configuration. Use ssh-keygen(1) for creating the ssh key pair.

Commands

service icasd start Starts the daemon. The daemon connects to the master(s), and possibly reconnects to them when the connection fails.

service icasd stop Stops the daemon. The connected master(s) are disconnected.

service icasd restart

service icasd reload Stops and starts the icasd daemon.

service icasd status Prints the status of the icasd daemon. If running, all the configured masters are listed with the information whether they are currently connected or not.

service icasd hash Prints the configuration hash.

Configuration

The icasd daemon is enabled in `rc.conf` with variable `icasd_enable="YES"`.

The configuration of the icasd daemon is in `/usr/local/kernun/etc/icasd.conf`. The following variables can be set in the configuration file:

SLAVE_ID_RSA

FN_SLAVE_ID_RSA The private SSH key of the icasd daemon. Either the contents of the file, or the file name.

WRITE_CFG_HASH The file name where the hash should be written upon start (including restart, reload).

MASTER_NAMES The list of masters (space separated). For each master *MA*, the following variables define each icamd master:

MASTER_ADDR_*MA* The address of the master *MA*. Either IP address or hostname. Slave connects to this address using ssh protocol.

MASTER_PORT_*MA* The port of the master *MA*. Slave connects to this port using ssh protocol.

MASTER_ID_RSA_PUB_*MA*

FN_MASTER_ID_RSA_PUB_*MA* The public SSH key of the icamd master. The authenticity of the master is checked against the ssh key.

MASTER_CFG_NAME_*MA* Optional. Defines the name for the master *MA*. Use this, if the icamd name should differ from *MA* (*MA* may not contaion hypens (-)).

See ica(5) for the high level configuration in CML Kernun configuration.

SEE ALSO

Kernun: icamd(8), ica(5), cluster(7), configuration(7),

NAME

imap4-proxy, test-imap4 — Internet Message Action Protocol v. 4 (IMAP4) proxy

SYNOPSIS

imap4-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-imap4 [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

Program **imap4-proxy** is the proxy daemon for the Internet Message Access Protocol version 4rev1 (IMAP4rev1), defined by RFC 3501. The proxy supports secure communication via SSL/TLS protocols, see ssl(5).

STARTUP AND CONFIGURATION

The proxy reads its configuration and starts listening on TCP sockets (address/port couples) specified by `listen-on` configuration section, see listen-on(5). If support of transparent connections (i.e., connections made directly from an IMAP4 client to an IMAP4 server and redirected to the proxy by NAT as described in transparency(7)) is requested by item `transparent` in section `listen-on`, the corresponding NAT redirections are established during proxy startup and removed upon exit.

Format of the configuration file is described in imap4-proxy.cfg(5). General syntax of Kernun configuration files is explained in configuration(7). Program `test-imap4` tests syntax and partially semantics of configuration; for test expression syntax, see test-expr(5).

Access Control

Imap4-proxy uses three-phase ACLs, see access-control(7). The first phase, `session-acl` is checked once for each client connection. It permits or denies client access and sets some connection parameters. The second phase, `command-acl` is also checked once for each connection, but it can be selected according to the client certificate in case of SSL/TLS enabled by `session-acl`. Various parameters can be set in `command-acl`, e.g., permitted sets of IMAP4 commands and capabilities, timeouts, SSL/TLS on the server connection. The third phase ACLs are used only if mail processing is enabled in `command-acl`.

There are two types of them. `Mail-acl` is checked once for each transferred mail. It defines rules for accepting or rejecting the mail according to its content and antivirus/antispam test results. `Doc-acl` is checked once for each document (MIME part) of a mail. It defines document processing, e.g., filtration or replacement by a fixed file. See mod-mail-doc(5) for more details.

Connection Establishment

When a connection from a IMAP4 client arrives, the configuration is searched for a matching `session-acl`. If the ACL says that the connection should be denied or there is no matching ACL, the proxy does not communicate with the client and closes the connection immediately. In addition to the generic ACL conditions and actions described in access-control(7), some **Imap4-proxy**-specific conditions and parameters can be set. It is possible to set `language` of protocol response messages generated by the proxy.

Item `client-ssl-params` switches on SSL/TLS on the client connection and sets various SSL/TLS parameters. If the connection from the client uses SSL/TLS then item `client-cert-match` defines the acceptable client certificates. If the client certificate does not pass the test, SSL/TLS connection establishment fails and the connection is closed. SSL/TLS handshake must complete until `idle-timeout` expires, otherwise the proxy closes the connection.

If the client connection is transparent (arriving to a transparent listening port), the original destination address is detected by the proxy and used as the server address for the server connection. Otherwise, the server must be specified by item `plug-to`. It is also possible to override a transparent destination address by `plug-to`.

Firewall administrator can choose the out-of-band method described in auth(7) for authenticating users on the proxy.

In the next step, the configuration is searched for a matching `command-acl`. It is possible to use values from a client certificate as a search condition. There are many options settable in `command-acl`. Language of protocol response messages generated by the proxy can be changed by `language`. This item overrides language setting from `session-acl`.

It is possible to turn on SSL/TLS on the server connection by `server-ssl-params` and to set requirements for the server certificate by `server-cert-match`. SSL/TLS can be used independently on the client and the server connection, hence the proxy may provide translation between unencrypted and encrypted communication.

Many limits can be set for a session. If any of the limits is exceeded, the proxy terminates the session. Total number of bytes transferred during a session is limited separately for client-to-server (`max-bytes-out`) and server-to-client (`max-bytes-in`) directions. No single mail may be larger than `max-mail-in` (server-to-client) or `max-mail-out` (client-to-server) bytes. Total time of the session is bounded by `max-time`. The session is terminated if it is idle longer than `idle-timeout`.

When a matching `command-acl` is found and it does not deny the session, the proxy connects to the server.

Protocol Processing

The proxy passes IMAP4 communication between the client and the server. It performs basic checks of the protocol. It is possible to permit only a subset of command by `command-acl.commands`. A forbidden command is not sent to the server and the proxy returns an error response. Item `command-acl.capabilities` selects an allowed subset of capabilities (returned by server in response to CAPABILITY command). A forbidden capability

is discarded by the proxy and not sent to the client. IMAP4 command LOGOUT or connection close by either the client or the server terminates the session.

Mail can be transferred to the client or to the server in one of two modes. In the first mode, the mail is first stored by the proxy, processed, and the result is sent to the client/server. In the second mode, turned on by item `no-mail-scanning` in `command-acl`, the mail is not processed by the proxy and data from the server are immediately passed to the client and vice versa. In the second mode without mail processing, antivirus and antispam checking is not performed. No conditions on mail contents and no mail modification options in `mail-acl` and `doc-acl` work, because `mail-acl` and `doc-acl` are not consulted at all (they can be even missing).

Mail Processing

Mail processing is controlled separately for mail transferred from the client to the server (section `command-acl.upload`) and for mail transferred from the server to the client (section `command-acl.download`). Mail processing is performed for each mail if the active `command-acl` does not contain `no-mail-scanning`. Mail processing options can be set by `mail-filter` which contains options specifying corrections of mails violating RFCs. In `command-acl`, there are also settings for antivirus and antispam checks (items `use-antivirus` and `use-antispam`, respectively). After a mail is read and stored by the proxy, it is checked by antivirus and antispam and its structure is analyzed.

`Mail-acl` (only one) and `doc-acl` (one for every MIME part of the mail) are found according to the conditions like direction of mail transfer (download or upload), results returned by the antispam and the antivirus, size, or MIME type. If any of the selected ACLs contains item `deny`, the mail is discarded and an error response is returned to the client. According to `doc-acl`, each document (MIME part) may be left unchanged, passed to the HTML filter, or replaced by a file. Actions defined by `mail-acl` for the whole mail include adding text to the subject and replacing the mail body by content of a file. See mod-mail-doc(5) for more details.

Logging

As all other Kernun proxies, **imap4-proxy** generates many log messages during its operation. Meaning of the messages may be found in section 6 of the manual pages. Details about Kernun logging can be found in logging(7).

The proxy logs statistical messages about each client connection and each request. When a connection arrives, SESSION-START is logged. Then ACL messages inform about the session and command ACLs selected for this connection. If mail processing is enabled, ACL messages are logged for each mail and doc ACL. Finally, SESSION-END is logged when the session is terminated.

Common Kernun Features

Imap4-proxy uses common Kernun mechanisms for listening on its sockets, accepting client connections, and managing its processes. It can also run in a chrooted environment and change its user identity upon startup. See also application(5), tcpserver(5), and tcpserver(7).

The proxy uses a common Kernun mechanism for network input/output. The configuration allows to specify several parameters like buffer sizes and timeouts, both for client and server connections. The parameters are set in configuration sections `client-conn` and `server-conn`. See netio(7) for details.

The proxy uses common Kernun mechanism for name resolving (see resolving(7) manual page).

Imap4-proxy uses common Kernun mechanism for runtime monitoring. For more detailed information, see monitoring(7).

Imap4-proxy uses common Kernun mechanism for traffic shaping. For more detailed information, see traffic-shaping(7).

The proxy uses common Kernun mechanism for document type identification (see doctype-identification(7) manual page).

OPTIONS

-h Display usage information and exit.

-v Print version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read configuration from *cfgfile*.

-r Resolve names in configuration prior to testing.

-t *test_expr* Test configuration according to given expression. Format of the *test expr* is described in test-expr(5).

SEE ALSO

listen-on(5), imap4-proxy.cfg(5), application(5), ssl(5), tcpserver(5), test-expr(5), access-control(7), configuration(7), logging(7), monitoring(7), netio(7), tcpserver(7), traffic-shaping(7), transparency(7), resolving(7), doctype-identification(7)

NAME

kat — Kernun Admin Tool

SYNOPSIS

kat [-hv]

kat [-d *dbglev*] [-f *cfgfile*]

kat [-d *dbglev*] [-f *cfgfile*] *command* [*params* ...]

DESCRIPTION

The **kat** is a complex tool for the Kernun Firewall Administrator to facilitate his/her work (configuration, process management, log inspection etc.). It can be used in two modes:

- When no *command* is used, the KAT will start interactive mode and prompts user for commands (see Section D below).
- When a *command* is used, the KAT executes only given command (see Section D below) and exits.

The KAT tool uses standard Kernun logging library for displaying messages (see logging(7)), the messages are written both to the standard error output (i.e. sent to the terminal, typically) and to the system log (as configured in `/etc/syslog.conf` file). This behavior can be changed by setting the environment variable KERNUN_LOG_FILE to a file name willing to be the log target. As usual, every message (produced by the KAT, not by other system programs called by the KAT) has a *log-id* prefix (e.g. `CMLK-872-E`) that can be found in Kernun section 6 manual pages (CMLK-720(6) in above example).

Options

The KAT options are as follows:

-h Print usage information and exit.

-v Display version information and exit.

-d *dbglev* Set debugging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details.

-f *cfgfile* Define the default configuration filename for the KAT **cml** command (see Section D below) when issued without parameter.

KAT COMMANDS

The command line length must be at most 512 characters and at most 50 arguments are allowed. Command names can be prefixed by a slash ('/') or dot-slash ('./') characters for the cml(8) commands compatibility. Command names are case insensitive.

There are several groups of KAT commands:

configuration **cml**, **apply**, **rlog**, **rcsdiff**

component management **showapp**, **start**, **stop**, **restart**, **reload**

process management **ps**, **kill**

other admin tools interface **af**, **cluster**, **monitor**, **quarc**, **router**, **triplicator**

miscellaneous **help**, **man**, **dbg**, **!**, **quit**

If the KAT runs in read-only mode (due to current user type), it allows only commands they do not change the firewall functions. Other commands are disabled.

Configuration commands

One of the essential functions of the KAT is to support the configuration task. The **cml** command starts the Configuration Meta Language (CML) tool, while the **apply** command distributes the configuration files (generated by the CML) onto proper places of filesystem so that operating system and Kernun applications can run according to the settings made by the Kernun administrator.

cml [-g] [-r *revision*] [*filename*] Start the CML tool with *filename* as the configuration file name. For detail information about the tool, see cml(8).

If the *filename* is omitted, the configuration file is set according to the following rules:

1. the same configuration file as the last `cml` command used
2. value of the `-f` option (if given when **kat** started)
3. value of environment variable `$KERNUN_CONF`
4. value of C preprocessor macro `KERNUN_CONF` (in compile time)
5. `"conf/kernun.cml"`.

In the last three cases, if the filename has a relative pathname, it is relative to the *kernun-root* directory defined during the installation process. If you want to choose another directory, you can set the environment variable `$KERNUN_ROOT` to the directory pathname (w/o the final slash).

The directory where the configuration file resides becomes the *configuration directory* used also in subsequent **apply** command (see below).

Typically, the /**generate** command of the cml(8) tool is used at the end of the configuration process to prepare a full set of configuration files to be applied in the system (see the **apply**

command below). The "generate-only" mode of the CML run can be requested by using the `-g` option — the CML only loads the configuration, generates the output and exits.

The `-r` option allows user to request loading of a particular revision from RCS file *`filename`*`,v`. The acceptable forms for *revision* are *major.minor*, '`0`' for the last version stored in RCS, or *−n* for the *n*-th version preceding the last version stored in RCS.

apply *system-name* [*host* [*port*]] Copy files generated by the CML tool to the target system. The files are expected to be in the *configuration directory* of the last **cml** command. If no such command was executed yet, the directory is determined by the same way as specified above in the **cml** command description.

In the configuration directory, subdirectories named `SYSTEM-`*`name`* are searched for. If some of them matches the argument given to this command, the `$KERNUN_ROOT/bin/apply.sh` shell-script is executed to distribute the file tree.

The command will operate *locally* (i.e. copy the files into filesystem root on local host) unless the `apply-host` directive is specified in the particular `system` section. In this case, the generated output tree is first copied by the ssh(1) to the target machine and then, the **kat apply** is run there. The last two parameters can be used to redefine the address and port where the ssh daemon on the target host listens on.

Besides copying files onto proper place in the filesystem, the command typically does some other work, like modifying operating system user set, creation of chroot-directories including all necessary files, mounting device filesystem etc.

> **Warning**
>
> The names of `system` sections in the CML are case-insensitive, but after generating the file tree, they becomes the regular UNIX file names that are *case sensitive*, of course. If you have changed the system name capitalisation not changing the spelling, you will have to remove the old configuration output tree by hand.

rlog [[-r] *rev* [`[-r]` *rev*] [*file*]]

Display RCS log of given *file* (`kernun.cml` by default).

Without `-r` options, the command shows the complete log. When some *rev* arguments used, only these revision logs are displayed.

For the *rev* specification, the following forms are allowed:

revision number For instance, "`2.13`".

0 The current revision.

-1, ... The previous and older revisions.

If the *rev* starts with a digit, the "`-r`" text should be omitted.

rcsdiff [[-r] *rev* [`[-r]` *rev*] [*file*]]

Display differences (the diff(1) output) between two versions of given *file* (`kernun.cml` by default).

Without `-r` options, the command compares the *file* with the last revision in the RCS file. With a single `-r` option, the command compares the *file* with the given revision. With two `-r` options, the command compares the two given revisions.

For the *rev* specification, the same forms as in the **rlog** command are allowed.

Component management commands

The KAT tool facilitates managing of Kernun firewall components, i.e. Kernun applications (Kernun proxies, SSH servers, nameservers, DHCP server, NTP daemon, postfix SMTP forwarders, PIKE monitor) and networking components (packet filter, network interfaces and routing) configured in the configuration file. All networking components are started by operating system itself. All applications are started after operation system boot by means of the `kernun.sh` rc-script placed into the directory `/usr/local/etc/rc.d` during the installation process.

All components write a hash of their configuration into the file `/var/run/`*name* during the startup process. Commands of this group can then detect components running with outdated configuration and needed to be restarted.

All commands of this group primarily look for component names in the `$KERNUN_ROOT/etc/component.lst` file generated by the **cml.generate** command. They do not respect the running applications (except for displaying their status etc.), they can be managed by Process management commands (see Section D below).

showapp [-n] [-tT *tag* [-o *format*]]

Display all configured components.

Options:

-n Display only "new" components, i.e. such ones with changed configuration.

-o *format* Display columns in order given by the *format* string. The string consists from heading names separated by commas. By default, command behaves like with the `NAME,PROG,TYPE,PARM,TAGS,STAT,PRTY` format.

-tT *tag* Display all components with (`-t`) or without (`-T`) given *tag*. Use the asterisk (`*`) value for "any tag". For the list of the tags, use the C3H support.

start *options params* Start the component(s).

Prior to starting any application, the KAT checks, whether requested applications are not already running, in which case the KAT rejects to start it.

stop *options params* Immediately stop component(s).

This command is not effectual to unconfigured or exiting proxies. Those must be killed by the **kill** command (see below).

restart *options params* Immediately stop component(s).

reload ***options params*** Gracefully stop component(s).

This command should be preferred in the normal circumstances. Use it rather than the **restart** command. All particular proxies are switched to an *exiting* state in which they do not accept new incoming requests awaiting to the end of all established connections.

All the control commands (**start**, **stop**, **restart** and **reload**) have following parameter specification possibilities:

command name Start/stop/restart/reload a component referenced in the configuration by the *name*.

command* `-a` *type Start/stop/restart/reload all components of given type. For the list of the types, use the C3H support. Generally, proxy type names (like `http-proxy`) and component types (like `proxy`, `sshd`, `postfix`, `cluster.monitor`, `interface` and `routing`) can be used.

command* `-tT` *tag Start/stop/restart/reload all components with (`-t`) or without (`-T`) given *tag*. Use the asterisk (`*`) value for "any tag". For the list of the tags, use the C3H support.

command Start/stop/restart/reload all Kernun applications (i.e. all components except network interfaces and routing). Before execution, the KAT tool asks the user to confirm it.

All the control commands recognize following options:

`-n` Start/stop/restart/reload only "new" components, i.e. such ones with changed configuration.

`-y` Answer "yes" to confirmation when starting all Kernun applications.

If there are more components to start, the KAT tool tries to repeat attempts to start them until all components succeed, or two consecutive loop iterations do not improve the state.

Process management commands

In the contrary to previous group of commands, process management commands does not deal with applications (proxies/ssh servers) according to the way how they are configured. Instead, current *running processes* are dealt with. The *name* of the proxy or ssh server is read from the ps(1) system command output.

`ps [-abdS] [ -tT` ***tag*** { *name*[`=`*parent-pid*] | [`-p` *program-name*] }]

Display information (using the system ps(1) command) about

all Kernun applications (proxy and server parents) if used without parameter.

all Kernun applications of given program type if used with the *program-name* parameter.

all processes (including children) of all Kernun applications given by name if used with the *name* parameter.

all processes (including children) of a single Kernun application if used with the *name* and the *parent-pid* parameters.

Option `-a` causes displaying information about *all* processes (including children) in any form of command.

Option `-b` forces *brief* output format.

Option `-d` restricts information to only such applications that are still running although they have been unconfigured (*dead*).

Option `-t` restricts information only to applications with given *tag*. If the asterisk (`*`) value is used, applications with any tag are displayed.

Option `-T` restricts information only to applications without given *tag*. If the asterisk (`*`) value is used, applications without any tag are displayed.

Option `-S` shortens output lines up to the terminal width.

kill [*-signal* `name`[=`parent-pid`] [`child-pid`]]

Send a signal to

a non-proxy component process The component is identified by its *name*.

a single proxy-parent The proxy is identified by its *name* and it must be the only live (not counting exiting processes) parent of given proxy.

a selected proxy-parent The proxy is identified by its *name* and *parent-pid*. This form of command acts to the *exiting* processes, too.

all proxy-parents The proxy is identified by its *name* and an asterisk (`*`) is used instead of the *parent-pid*. This form of command acts to the *exiting* processes, too.

all components The asterisk (`*`) is used both for the *name* and the *parent-pid*. This form of command acts to the *exiting* processes, too.

a selected proxy-child The proxy is identified by its *name* and eventually *parent-pid*, child process is identified by its *child-pid*.

all proxy-children The proxy is identified by its *name* and eventually *parent-pid*, the signal is sent to all children of given parent(s) if the asterisk (`*`) is used instead of the *child-pid* parameter.

If no *signal* is specified, the default signal is used (`15`, or `TERM`). When sending the `TERM` or the `HUP` (`1`) signal to a proxy parent, the KAT repeats sending of it until killing succeeds. If the retry limit (10x) is reached, the `KILL` (`9`) signal is sent (to parent and all children), instead.

kill -d *name*[=*parent-pid* [`child-pid`]]

Send the `TERM` signal to all unconfigured (dead) components.

Log control and management

The KAT tool facilitates log level control, log rotation and log inspection.

log { incr | decr | restart | rotate } [*proxy*]

Control proxy logging. Subcommands `incr` and `decr` increases/decreases proxy logging level. Subcommand `restart` forces proxy to reopen the log file. Subcommand `rotate` rotates particular proxy log file (i.e. renames the file and restarts proxy logging). If used without proxy name, rotates all proxies.

Other admin tools interface

There are some other administrator tools in the Kernun Firewall. Commands in this group facilitate using of them.

cluster { take | drop } [*VCID*]

Take or drop Master role for all virtual clusters, or just the one with number *VCID*.

af blacklist { show | count | flush | add ... | del ... | unblock *IP-address* | upload | refresh }

Adaptive Firewall autonomous blocking module control.

show Show the current blacklist set in the packet filter.

count Print number of addresses in the current blacklist in the packet filter.

flush Flush all addresses of the current blacklist in the packet filter.

add *IP-address* Add an IP address to the current blacklist in the packet filter.

> **Warning**
>
> The table remains changed only until the next table refresh interval. Do not use this command to block an IP address. Use the BLOCK command instead.

del *IP-address* Delete an IP address from the current blacklist in the packet filter.

> **Warning**
>
> The table remains changed only until the next table refresh interval. Do not use this command to unblock an IP address. Use the UNBLOCK command instead.

block *IP-address* Adds given address to the PF blacklist table and the AKBL database table.

unblock *IP-address* Removes given address from the PF blacklist table and all IDS database tables.

upload Upload the Adaptive Firewall IDS database to central server.

refresh Fetch a new Adaptive Firewall IPS database (according to the configuration either feed it from the local IDS database, or download it from the central server) and install it to the packet filter.

af pf { find *IP-address* | show *table-name* } Works with AF packet filter tables.

find *IP-address* Search for an IP address in all Adaptive Firewall packet filter blocking tables.

show *table-name* Show table content.

af download ids-agent-rules Downloads IDS agent rules and install them.

af download adaptive-database Downloads the Adaptive database install the data into PF tables.

af `...` Calls af-db.sh(8) tool.

monitor `...` Monitor proxies activity, see monitor(1).

quarc `...` Control mail quarantine, see quarc.sh(1).

router *name* `...` Provides additional routing protocol daemon commands:

show route [all] Show routing table.

show ospf Show OSPF configuration information.

show ospf state[all] Show OSPF status information.

triplicator *proxy command* **[** *params* **]** Display or change data in grey-list triplet database, see triplicator(1).

Miscellaneous commands

dbg *level* [{ **con** | **log** [*filename*] }]

Change amount of displayed messages to the *level* (see logging(7) for possible values and their meaning).

If you want to change the level only for console or log, you can specify the `con` or `log` keyword respectively. In the case of log, you can also change the target file name.

help *command* Get help about the KAT *command*.

man [*section*] *topic* Interface to the man(1) command with the priority of the Kernun manual pages.

[!] *command...* Execute shell *command*.

If the command name does not collide with any KAT command, the '`!`' prefix should be omitted.

quit Quit the KAT tool.

C3H SUPPORT

The Command Completion and Context Help (C3H) support helps you to write correct commands or proper parameters a bit faster. The simple basic rule is: "If you don't know what to do now, press **<TAB>**!". Of course, it does not work absolutely perfectly in all situations, but it works e.g. when selecting a configuration file (**cml**, **quarc**), a `system` name (**apply**), a proxy name (application management, process management, **showlog**, **monitor**), a manual page name (**man**), a signal name (**kill**) etc.

Control Sequences

The End-of-file control sequence (**^D**, Control-D) can be used for quitting the KAT tool.

The **^R** (Control-R) sequence is used for command history searching. You can type part of some previous command (the part is displayed in the prompt) and C3H searches in the history to the last command containing such a string. This command is then displayed on the command line and you can tune the selection by adding more characters to the pattern, removing some characters by the **Backspace** key or repeating the search by pressing **^R** again. If your selection is completed, press **Enter**, the selected command is placed into the command line buffer and you can edit it. The KAT tool saves command history at the end of its work and restores it at the beginning.

The **^U** (Control-U) sequence is used for clearing the command line.

ENVIRONMENT

KERNUN_LOG_FILE The file name where log messages will be redirected. If not set, system logging is used.

SEE ALSO

Kernun: af-db.sh(1), monitor(1), quarc.sh(1), triplicator(1), kernun.cml(5), adaptive-firewall(7), configuration(7), logging(7), tcpserver(7), transparency(7), cml(8), pf-control(8)

FreeBSD: bzip2(1), diff(1), kill(1), man(1), ps(1), ssh(1), tail(1), pf(4), pfctl(8)

NAME

kavhttpd — Kaspersky AV in HTTP mode integrated in Kernun UTM

SYNOPSIS

service kavhttpd { bases_being_updated| bases_check| bases_date| bases_last_update| bases_redownload| bases_update| bases_update_slp| hash| keyinfo| kill_orphans| licinfo| reload| restart| start| status| stop| test| version }

DESCRIPTION

Kavhttpd is controlled by this RC script. This page describes the special configuration parameters and commands for the RC script.

Configuration

The kavhttpd is configured by rc.conf(5).

The working configuration file for **kavhttpd** is created by this script from the configuration parameters. The working configuration file for **kavhttpd** is located at `/usr/local/kav_httpd/etc/kavhttpd.xml`.

The working configuration file for **keepup2date8** (the bases update program) is created by this script from the configuration parameters. The working configuration file for **keepup2date8** is located at `/usr/local/kav_httpd/etc/keepup2date8.xml`.

The following special variables can be set:

kavhttpd_port The TCP port the kavhttpd should listen for connections.

kavhttpd_addr The IP address or hostname the kavhttpd should listen for connections.

kavhttpd_maxhttpsessionsnum The maximum number of active HTTP sessions that are handled by KAV HTTPD simultaneously. Extra connections are queued by the system. The queue size is defined by `kavhttpd_acceptqueuelen`.

kavhttpd_acceptqueuelen The length of the queue of pending connections to the kavhttpd (the backlog argument to the listen(2) syscall). 0 means the system default.

kavhttpd_maxtcpfilesize The maximum size (in bytes) of the file contents which can be passed to KAV HTTPD.

kavhttpd_sessiontimeout The timeout for processing the request and sending the response, in milliseconds. To use an infinite timeout, set this value to 0.

kavhttpd_scannerscount The number of scanning processes. The maximum permissible number of scanning processes is 256. Note that in out-of-process mode every scanner process uses its own copy of anti-malware database. Thus initializing with a large number of scanner

processes consumes considerable time and memory resources. It is recommended to use kavhttpd scannerscount equal to the number of CPU cores.

kavhttpd_threadscount The maximum number of simultaneously running scanning threads. The scanning threads are distributed among scanning processes. The maximum permissible number of scanning threads is 256.

kavhttpd_queuelen The maximum length of the scanning task queue. The queue is used in asynchronous scanning. All the scanning processes take the scanning tasks from a single queue.

kavhttpd_loglevel 0 disables logging. 1 enables full logging mode. Use this mode for debugging purposes.

RC-script commands

The following extra commands can be used.

bases_being_updated Checks whether the bases update is being performed at the moment. Returns 1 if yes. Returns 0 if not.

bases_check Runs the kav internal program for checking the bases consistence.

bases_date Prins the bases release timestamp.

bases_last_update Prints the timestamp when the last succesful bases update has finished (either the bases were updated or they were already up-to-date).

bases_redownload Deletes the bases and downloads it from a scratch.

bases_update Updates the bases now. The progress is printed to log and to stderr. Eventual errors are printed to stdout. Return 0 if the update was succcessfull.

bases_update_slp Sleeps random time (0..1800 sec) and updates the bases. The progress is printed to log and stderr. Eventual errors are printed to stdout. Return 0 if the update was succesful.

hash Prints the configuration hash.

keyinfo Prints the information on the KAV KEY file.

kill_orphans Kills the eventual 'kavscanner' orphans. They may occur, when the 'kavscanner' is inappropriately killed.

licinfo Prints the information on the KERNUN-KAV license file. Returns 0 if the license is valid.

reload Same as restart.

restart Restarts the kavhttpd.

start Starts the kavhttpd.

status Whethe the kavhttpd is running.

stop Stops the kavhttpd.

test [filename ...] Performs a basic AV test: CLEAN and EICAR files are tested. Returns 0 upon success. If files are given, they are tested instead of CLEAN and EICAR. If files are given, the return code is always 0.

version Prins the version of kavhttpd.

SEE ALSO

Kernun: configuration(7),

System: rc.conf(5),

NAME

pf-control — Packet filter control daemon

SYNOPSIS

pf-control [-hv] [-d *dbglev*] -f *cfgfile*

DESCRIPTION

The packet filter function of Kernun is configured by the **packet-filter** CML section and controlled by a special component **PF** of type **pf-control**.

When started, this application tries to resolve all domain names in the configuration, prepares PF tables and schedules itself to make necessary changes if the configuration contains time-limited rules. Then it enables the PF in the system (see pfctl(8)) and starts logging of PF events and monitoring of changes due to DNS resolution or time contraints. When stopped, the program disables the PF in the system.

The daemon runs in fact as three processes, like Kernun proxies do. The main process just controls run of its children. The Asynchronous Configuration Resolver provides for DNS resolution refreshing. The regular child process handles PF tables and reads pflog(4) and pfsync(4) devices as a source of PF event information to log it to both `log-debug` and `log-stats` logs.

LOGGING

The **pf-control** daemon uses the same principles for configuration of logging like other Kernun components (see logging(7)). However, some aspects of its logging are a bit special. Every ACL from the **packet-filter** configuration controls some events that are logged according their nature and the same is true for raw rules configured manually. The extent of logging can be also changed by using a special ACL item or PF rule attribute **log**.

The ACL item has four possible values (some are not valid for some types):

`default` Default way of logging according to the event type.

`off` Logging is suppressed to the minimum extent. It corresponds to raw rules with no `log` option.

`on` Logging is switched on. It corresponds to raw rules with `log` option.

`all` Logging is switched to the maximum extent (all packets). It corresponds to raw rules with `log(all)` option.

The event types behaves according to this schema:

`Blocking rules` By default, an event is logged to both `log-debug` and `log-stats` logs with the I-level. No modification of this mode is allowed.

Stateful PASS rules By default, an event is logged to the `log-stats` log at the end with the I-level, and to the `log-debug` log also at the start with the D-level. Values `off` and `all` can be used for varying the quantity of `log-debug` messages.

Stateless PASS rules By default, an event is logged only to the `log-debug` log with the D-level. Value `off` can be used for switching off the logging at all.

NAT and RDR rules By default, an event is logged to the `log-stats` log at the end with the I-level. No modification of this mode is allowed.

SIGNALS

The **pf-control** daemon handles following signals:

SIGUSR1 Log level increasing.

SIGUSR2 Log level decreasing.

SIGINFO Operation status logging; parent process logs info about all children, child process logs current rule map.

SIGHUP Service termination; the daemon keeps the PF tables, rules and states.

SIGINT, **SIGQUIT**, **SIGTERM** Immediate termination; the daemon flushes the PF states and disables PF at all.

OPTIONS

-h Print usage information.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

SEE ALSO

Kernun: application(5), pf-control.cfg(5), configuration(7), logging(7), resolving(7)

FreeBSD: pf.conf(5), pfctl(8)

NAME

pikemon — PIKE cluster protocol control daemon

SYNOPSIS

pikemon [-hv]

pikemon [-d *dbglev*] -f *cfgfile*

pikemon [-d *dbglev*] -f *cfgfile* -c *command*

test-pikemon [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

The redundancy cluster feature of Kernun firewall needs a daemon monitoring the status of cluster members and operation ability using the PIKE protocol. This is the task of the **pikemon** application. The second task of the application is to execute special commands for controling the cluster. For this purpose, the application is called in non-daemon mode.

When started as the PIKE protocol monitor, **pikemon** reads the `status-file` and sets the (Master or Backup) role of the node according to the content of the file. Then it starts to check its health status by sending ICMP ECHO messages to all configured targets (`ping` groups) and monitoring status of interfaces defined in the configuration. It also starts to send **HELLO** messages from the *PIKE* protocol over the *heart-beat* interface to the cluster peer. The health status and the result of the dialogue with the partner in the cluster can lead to taking or dropping the Master role of the node.

Taking the Master role means stealing the shared *virtual* IP and MAC addresses of all controlled interfaces i.e. sending proper gratuitous ARP packets. Thus, all bridge interfaces must have the IP address assigned in the configuration. The MAC address is assigned as 02:*IP address*:00 by default, but it can be changed. The Backup node keeps the IP address assigned unless marked as `nomadic` in the `pike` item.

The daemon runs in fact as three processes, like Kernun proxies do. The main process just controls run of its children. The Asynchronous Configuration Resolver provides for DNS resolution refreshing. The regular child process handles the real operation and in its process information (shown by the **ps**), the current status of all virtual clusters is figured out. There is a group of three letters for each virtual cluster with following meaning:

P This node wants to act as the primary node.

S This node wants to act as the secondary node.

M This node currently plays the Master role.

B This node currently plays the Backup role.

U This node has responses from all ping groups (“up” state).

D This node did not get response from at least one ping group ("down" state).

The current status of this host and the cluster peer as well as results of pinging to the target hosts can be watched by the monitor(1) tool avaliable also as a command of the kat(8) tool.

When started with the -c option, **pikemon** reads the status file and the configuration, executes command requested and exits.

SIGNALS

The **pikemon** daemon handles following signals:

SIGUSR1 Log level increasing.

SIGUSR2 Log level decreasing.

SIGINFO Operation status logging; parent process logs info about all children, child process logs current status of all ping groups.

SIGHUP Service termination; the daemon keeps the state until a new instance is started which kills it.

SIGINT, **SIGQUIT**, **SIGTERM** Immediate termination; the daemon immediately closes the service and drops Master role.

OPTIONS

-h Print usage information.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

-c *command* Execute *command* (see below) and exit.

COMMANDS

take [*VCID*]

Takes Master role in all virtual clusters, or just in the virtual cluster with number *VCID*.

drop [*VCID*]

Drops Master role in all virtual clusters, or just in the virtual cluster with number *VCID*.

SEE ALSO

Kernun: monitor(1), application(5), pikemon.cfg(5), cluster(7), configuration(7), logging(7), resolving(7), kat(8)

NAME

pop3-proxy, test-pop3 — Post Office Protocol v. 3 (POP3) proxy

SYNOPSIS

pop3-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-pop3 [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

Program **pop3-proxy** is the proxy daemon for Post Office Protocol version 3 (RFCs 1939, 2449, 1734). The proxy supports secure communication via SSL/TLS protocols, see ssl(5).

STARTUP AND CONFIGURATION

The proxy reads its configuration and starts listening on TCP sockets (address/port couples) specified by `listen-on` configuration section, see listen-on(5). If support of transparent connections (i.e., connections made directly from a POP3 client to a POP3 server and redirected to the proxy by NAT as described in transparency(7)) is requested by item `transparent` in section `listen-on`, the corresponding NAT redirections are established during proxy startup and removed upon exit.

Format of the configuration file is described in pop3-proxy.cfg(5). General syntax of Kernun configuration files is explained in configuration(7). Program **test-pop3** tests syntax and partially semantics of configuration; for test expression syntax, see test-expr(5).

Access Control

Pop3-proxy uses three-phase ACLs, see access-control(7). The first phase, `session-acl` is checked once for each client connection. It permits or denies client access and sets some connection parameters. The second phase, `command-acl` is also checked once for each connection, but it can be selected according to the client certificate in case of SSL/TLS enabled by `session-acl`. Various parameters can be set in `command-acl`, e.g., permitted sets of POP3 commands and capabilities, timeouts, SSL/TLS on the server connection.

The third phase ACLs are used only if mail processing is enabled in `command-acl`. There are two types of them. `Mail-acl` is checked once for each mail transferred from the server to the client. It defines rules for accepting or rejecting the mail according to its content and antivirus/antispam test results. `Doc-acl` is checked once for each document (MIME part) of a mail. It defines document processing, e.g., filtration or replacement by a fixed file. See mod-mail-doc(5) for more details.

Connection Establishment

When a connection from a POP3 client arrives, the configuration is searched for a matching `session-acl`. If the ACL says that the connection should be denied or there is no matching ACL, the proxy does not communicate with the client and closes the connection immediately. In addition to the generic ACL conditions and actions described in access-control(7), some **Pop3-proxy**-specific conditions and parameters can be set. It is possible to set `language` of protocol response messages generated by the proxy.

Item `client-ssl-params` switches on SSL/TLS on the client connection and sets various SSL/TLS parameters. If the connection from the client uses SSL/TLS then item `client-cert-match` defines the acceptable client certificates. If the client certificate does not pass the test, SSL/TLS connection establishment fails and the connection is closed. SSL/TLS handshake must complete until `idle-timeout` expires, otherwise the proxy closes the connection.

If the client connection is transparent (arriving to a transparent listening port), the original destination address is detected by the proxy and used as the server address for the server connection. Otherwise, the server must be specified by item `plug-to`. It is also possible to override a transparent destination address by `plug-to`.

Firewall administrator can choose the out-of-band method described in auth(7) for authenticating users on the proxy.

In the next step, the configuration is searched for a matching `command-acl`. It is possible to use values from a client certificate as a search condition. There are many options settable in `command-acl`. Language of protocol response messages generated by the proxy can be changed by `language`. This item overrides language setting from `session-acl`.

It is possible to turn on SSL/TLS on the server connection by `server-ssl-params` and to set requirements for the server certificate by `server-cert-match`. SSL/TLS can be used independently on the client and the server connection, hence the proxy may provide translation between unencrypted and encrypted communication.

Many limits can be set for a session. If any of the limits is exceeded, the proxy terminates the session. Total number of bytes transferred during a session is limited separately for client-to-server (`max-bytes-out`) and server-to-client (`max-bytes-in`) directions. No single mail may be larger than `max-mail-in` bytes. Total time of the session is bounded by `max-time`. The session is terminated if it is idle longer than `idle-timeout`. POP3 is a line-oriented protocol. The proxy checks length of each line and terminates the session if a line exceeds a limit: `cmd-line-len` for command lines sent by the client, `resp-line-len` for response lines sent by the server, or `mail-line-len` for mails received from the server.

When a matching `command-acl` is found and it does not deny the session, the proxy connects to the server.

Protocol Processing

The proxy passes POP3 communication between the client and the server. It performs basic checks of the protocol. Line lengths are compared to limits from `command-acl`. It is possible to permit only a subset of command by `command-acl.commands`. A forbidden command is not

sent to the server and the proxy returns an error response. Item `command-acl.capabilities` selects an allowed subset of capabilities (returned by server in response to CAPA command). A forbidden capability is discarded by the proxy and not sent to the client. POP3 command QUIT or connection close by either the client or the server terminates the session.

Retrieved mail can be transferred to the client in one of two modes. In the first mode, the mail is first stored by the proxy, processed, and the result is sent to the client. In the second mode, turned on by item `no-mail-scanning` in `command-acl`, the mail is not processed by the proxy and each line from the server is immediately passed to the client. In the second mode without mail processing, antivirus and antispam checking is not performed. No conditions on mail contents and no mail modification options in `mail-acl` and `doc-acl` work, because `mail-acl` and `doc-acl` are not consulted at all (they can be even missing).

Mail Processing

Mail processing is performed for each mail if the active `command-acl` does not contain `no-mail-scanning`. Mail processing options may be set by `command-acl.mail-filter`. In `command-acl`, there are also settings for antivirus and antispam checks (items `use-antivirus` and `use-antispam`, respectively). Section `mail-filter` contains options mainly specifying corrections of mails violating RFCs. After a mail is read from a server and stored by the proxy, it is checked by antivirus and antispam and its structure is analyzed.

`Mail-acl` (only one) and `doc-acl` (one for every MIME part of the mail) are found according to the conditions like results returned by the antispam and the antivirus, size, or MIME type. If any of the selected ACLs contains item `deny`, the mail is discarded and an error response is returned to the client. According to `doc-acl`, each document (MIME part) may be left unchanged, passed to the HTML filter, or replaced by a file. Actions defined by `mail-acl` for the whole mail include adding text to the subject and replacing the mail body by content of a file. See mod-mail-doc(5) for more details.

Logging

As all other Kernun proxies, **pop3-proxy** generates many log messages during its operation. Meaning of the messages may be found in section 6 of the manual pages. Details about Kernun logging can be found in logging(7).

The proxy logs statistical messages about each client connection and each request. When a connection arrives, SESSION-START is logged. Then ACL messages inform about the session and command ACLs selected for this connection. If mail processing is enabled, ACL messages are logged for each mail and doc ACL. Finally, SESSION-END is logged when the session is terminated.

Common Kernun Features

Pop3-proxy uses common Kernun mechanisms for listening on its sockets, accepting client connections, and managing its processes. It can also run in a chrooted environment and change its user identity upon startup. See also application(5), tcpserver(5), and tcpserver(7).

The proxy uses a common Kernun mechanism for network input/output. The configuration allows to specify several parameters like buffer sizes and timeouts, both for client and server connections. The parameters are set in configuration sections `client-conn` and `server-conn`. See netio(7) for details.

The proxy uses common Kernun mechanism for name resolving (see resolving(7) manual page).

Pop3-proxy uses common Kernun mechanism for runtime monitoring. For more detailed information, see monitoring(7).

Pop3-proxy uses common Kernun mechanism for traffic shaping. For more detailed information, see traffic-shaping(7).

The proxy uses common Kernun mechanism for document type identification (see doctype-identification(7) manual page).

OPTIONS

-h Display usage information and exit.

-v Print version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read configuration from *cfgfile*.

-r Resolve names in configuration prior to testing.

-t *test_expr* Test configuration according to given expression. Format of the *test expr* is described in test-expr(5).

SEE ALSO

listen-on(5), pop3-proxy.cfg(5), application(5), ssl(5), tcpserver(5), test-expr(5), mod-mail-doc(5), access-control(7), configuration(7), doctype-identification(7), logging(7), monitoring(7), netio(7), resolving(7), tcpserver(7), traffic-shaping(7), transparency(7)

NAME

sip-proxy, test-sip — Session Initiation Protocol (SIP) proxy

SYNOPSIS

sip-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-sip [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

The **sip-proxy** provides proxying service for the Session Initiation Protocol (RFC 3261 et al.). This is the first version implemented in the Kernun firewall and as such it supports mainly Internet telephone calls and related services.

Program `test-sip` tests syntax and partially semantics of the configuration; for test expression syntax, see test-expr(5).

Format of the proxy configuration file is described in sip-proxy.cfg(5).

RFC describes two modes of proxy operation, stateful and stateless. The Kernun firewall SIP proxy behaves something in between of them. It provides almost full control of security aspects (allowing and denying requests, checking messages format, watching time spent and data transferred etc.). On the other hand, it does not take the control of application-level aspects (like resending requests and responses, acknowledging sessions etc.), these remain completely on the UAs responsibility.

In normal circumstances, clients register themselves via the proxy and after it they can either invoke or accept calls. In the case of using of public addresses or controlling via the configuration, clients can accept calls without registration. Data of all registered clients are stored into a special file (see the `map-file` configuration item desciption).

The proxy works both in transparent (see transparency(7)) and in non-transparent mode. The latter is recommended, however it requires to configure the *outbound* proxy on all clients and the `plug-to` configuration item in the proxy configuration. In the transparent case, the proxy hides its role in the communication process and it can cause problems in some cases.

The proxy remembers transparency mode for every registration and subsequent incoming calls are forwarded according to it. Thus, if a client (phone) has registered transparently, the data sent from the proxy to it in future incoming calls will use the caller address as the IP source address (like if the `source-address client` mode used). The proxy assumes that responses will be sent transparently to caller address and so they will reach the proxy again.

In all other cases, the proxy forwards messages using its own address. It tries to find a proper route, takes the corresponding interface address and checks whether it also listens on this address. Then, it uses this address (and the first bound port) as the source address. If some of these cheks fail, the session is rejected. Specially, for every *outgoing* direction, you must have proper *non-transparent listening* socket opened to be able to receive responses and incoming calls. The

`source-address` is kept in the configuration for special cases, but in normal circumstances, it is not needed.

For the security reason, the proxy allows the administrator to hide some important data (e.g. internal addresses) usually stored into SIP messages headers. This can be done via the `hide` configuration item of the `session-acl` configuration sections. In this case, the proxy hashes (using the `hash-salt`) private data and presents itself to outer world, instead of real client or server. The session identification (carried in the `Call-ID` header) is rewritten always, so the proxy acts as a terminal UAC/UAS of the call, from the Call-ID point of view.

The proxy usually runs as two processes (not counting the configuration resolving process - see resolving(7)): the single child process manages all the sessions and the parent process manages the child and restarts it after a failure. You can learn more in udpserver(7) manual page, although the **sip-proxy** does not use the **udpserver** library, in fact. However, it uses the same operation logic.

Common Kernun Features

The proxy uses common Kernun mechanism for network input/output operations. Configuration allows for specifying several parameters like buffer sizes and timeouts, both for control and data connections. They can be included in the `ctrl-conn` and `data-conn` configuration sections, respectively. For more detailed information, see netio(7).

The proxy uses common Kernun mechanism for logging (see logging(7)). For every session, the SESSION-START (SIPS-800-I), the ACL (SIPS-821-I and SIPS-822-I), the SESSION-INIT (SIPS-801-I) and the SESSION-END (SIPS-809-I) messages are logged. For every request, the SIPR-800-N message is logged, reporting both the request and the response. For every used data channel, a couple of DATA-END (SDPC-800-N) messages are logged, reporting amount of data and termination time of both sides of the channel.

Startup and Configuration

The proxy reads its configuration file and starts listening on specified IP sockets (address/port couples), as specified in the `listen-on` configuration section (see listen-on(5)). Proxy listens for both UDP and TCP protocols.

If support of transparent connections (see transparency(7)) is requested by item `transparent` in section `listen-on`, the corresponding NAT redirections are established during proxy startup and removed upon exit.

Access Control Lists

The proxy uses two layers of ACL (see access-control(7)) named `session-acl` and `request-acl`.

When the first request of a session arrives, configuration is consulted, proper `session-acl` is selected and according to it, the session is served or not. In the latter case, the request is ignored, by default. This feature can prevent against DoS attacks caused by sending lots of unauthorized packets. Regular termination of session according to the RFC (i.e. with replying and waiting

for specified time, client deresolving, etc.) can be forced by the `reject-gracefully` item of particular `session-acl` section.

Subsequently, protocol-specific parameters of the request is checked against set of `request-acl` entry conditions and proper mode of operation is selected.

Additionally to the general Kernun ACL concept, `request-acl` brings a new entry condition item:

`request-uri` This item contains a set of regular expressions and/or strings describing URIs used in request that is to be dealt by this `request-acl`. By this item, the administrator can switch traffic to different `plug-to` servers according to the Request URI format.

Subsequent requests within the same session are not checked again, in this version.

Signals

The **sip-proxy** handles following signals:

`SIGUSR1` Log level increasing.

`SIGUSR2` Log level decreasing.

`SIGINFO` Operation status logging; parent process logs info about all children, child process dumps registration map and all tables content.

`SIGHUP`, **`SIGINT`**, **`SIGQUIT`**, **`SIGTERM`** Immediate termination; proxy immediately closes all connections and terminates.

`SIGWINCH` Reopen logfile; if the proxy logs into a file, sending this signal causes the file will be reopened. This feature helps admin to rotate logfiles.

Program options

The program options are as follows:

`-h` Print usage information and exit.

`-v` Display version information and exit.

`-d` ***`dbglev`*** Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

`-f` ***`cfgfile`*** Read *cfgfile* for configuration information.

`-r` Resolve names in configuration prior to testing.

`-t` ***`test-expr`*** Test configuration according to given expression. Format of the *test-expr* is described in test-expr(5).

BUGS

Currently, the **sip-proxy** doesn't resolve domain names used in SIP messages and UA addresses. The DNS querying should slow down the proxy, too much.

SEE ALSO

Kernun: sip-proxy.cfg(5), listen-on(5), application(5), test-expr(5), SIPS-800(6), SIPS-801(6), SIPS-809(6), SIPS-821(6), SIPS-822(6), SIPR-800(6), SDPC-800(6), access-control(7), configuration(7), logging(7), netio(7), resolving(7), transparency(7), udpserver(7)

NAME

smtp-proxy, test-smtp — Simple Mail Transfer Protocol (SMTP) proxy

SYNOPSIS

smtp-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-smtp [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test-expr*]

DESCRIPTION

Program **smtp-proxy** is the proxy daemon for the Simple Mail Transfer Protocol (RFCs 2821, 2822, 2045 etc.). The proxy is not a final delivery server, so it indeed supports the SMTP commands like **VRFY** and **EXPN** but responds by the SMTP defined negative response.

Startup and Configuration

The proxy reads its configuration and starts listening on TCP sockets (address/port couples) specified by `listen-on` configuration section, see listen-on(5). If support of transparent connections (see transparency(7)) is requested by item `transparent` in section `listen-on`, the corresponding NAT redirections are established during proxy startup and removed upon exit. However, transparent connections are in fact not supported by this proxy, decisions about servers are made according to the proxy configuration, not by original destination.

Format of the configuration file is described in smtp-proxy.cfg(5). General syntax of Kernun configuration files is explained in configuration(7).

Program **test-smtp** tests syntax and partially semantics of configuration; for test expression syntax, see test-expr(5).

Access Control

The **smtp-proxy** uses standard Kernun access control (see access-control(7)) with four types of ACLs on three levels:

1. `session-acl` (level 1) is checked once for each client connection and defines general protocol behavior, or rejecting the connection. In addition to the generic ACL conditions and actions, some **smtp-proxy**-specific conditions and parameters can be set (see smtp-proxy(5)).

2. `delivery-acl` (level 2) is checked once for each mail recipient and defines the response to the **RCPT TO** command, i.e. way of delivery, or rejecting particular addressee. During this phase, the correctness of all the commands (**HELO**, **MAIL FROM** and **RCPT TO**) and its arguments is checked and proper decision is made.

Warning

When resolving domain names, the current `resolver` section setting is applied. It means that the `search` list (if present) is tried when some resolution fails. If this search-list contains some domain having the `*.domain` MX record set, then every resolution will succeed and errors like `unknown-perm` will never occur. This is probably not what you want. To solve this problem you can define a new `resolver` section without the `search` item and then use this resolver in the particular **smtp-proxy** only by the `use-resolver` item in the proxy configuration section.

3. `mail-acl` (level 3M) is checked once for each mail recipient and it defines rules whether to reject/accept forwarding the mail to him/her.

4. `doc-acl` (level 3D) is checked once for each recipient and document (MIME part) and it defines document processing mode (e.g. filtering, replacing etc.).

Warning

The proxy uses the same approach to the complexity of ACL set as any other Kernun proxy. However, there is a special feature of the SMTP protocol that within a single session, data for many users can be transported. An incomplex ACL set leading to the ACL search failure will cause (comparably to other proxies) an abort of the whole session. Thus, it is highly recommended to add a single denial ACL to the end of every layer ACL set (i.e. `delivery-acl`, `mail-acl` and `doc-acl`). These "sentinel" ACLs should cover "the rest of the world", i.e. they will typically have no entry conditions and a proper denial operation included.

For detail description of mail processing, see below Section D.

Client verification

The Kernun **smtp-proxy** provides some configurable client identity checking to protect against unsolicited mails.

- The client address can be deresolved to check whether it is a known station (see `unknown-client` item in `session-acl`). If the client deresolution fails, the session is rejected.

- The client address can be cross-resolved to check whether its name is not faked (see `unmatching-client` item in `session-acl`). Every name got by deresolution is resolved back and the resulting addresses are checked against the client IP address. If no one address matches, client session is rejected.

- The client address can be queried at any *black-list database* that uses the DNS based querying model (like `http://www.spamhaus.org` or `http://www.ordb.org`). The list of databases used can be specified in `session-acl.blacklisted-client` configuration

directives. If the client address is found in any of the configured databases, client session is rejected.

- After receiving the `MAIL FROM` command, the client can be checked according to the Sender Policy Framework (`http://www.openspf.org`) of domain presented by the client as the mail *return path* (see `white-listing` item in `session-acl`). The result of this check can be used for matching during `delivery-acl` selection (see `spf` item).

> **Warning**
>
> This feature has some weaknesses. SPF is unusable in case of forwarding mails from a foreign mailserver outside our domain to a target mailserver in our domain. Such cases should be handled specially in your configuration. In case of server-to-server forwarding (e.g. from your secondary MX) you will probably deny white-listing in particular `session-acl`, in case of individual forwarding (between users' mailboxes) you can handle it in per-user `delivery-acl`.

- The **smtp-proxy** provides possibility of the grey-listing method (`http://projects.puremagic.com/greylisting`) if configured by `grey-listing` section in the `delivery-acl`. For more details, see the triplicator(1) tool description.

Forwarding

The Kernun **smtp-proxy** is a security proxy, not a regular MTA (Mail Transfer Agent). Its task is to apply security policy, check incoming mails for correctness and then use some *mail-forwarder* to queue and distribute the mails. For this purpose, several `smtp-forwarder` global sections can be defined in the configuration. Every section specifies internet *domain* which the forwarder is to be used for and connection parameters of forwarding channels (e.g. addresses, timeouts etc.).

> **Warning**
>
> You can omit the `domain` settings in `smtp-forwarder` sections and postpone forwarders selection to the `delivery-acl` sections using the `via` element of the `deliver` item. For the original recipients of a mail, this solution is sufficient. However, if you will use sending a copy of a mail to another address by the `copy-to` item, you have to use the domain setting because for such recipients there are no other way how to select proper SMTP forwarder channel.

Forwarders can be instances of common UNIX MTA daemons (e.g. **postfix** distributed together with Kernun) running on the firewall machine, or some external MTAs (e.g. the central site mailserver for incoming mails and the ISP mailserver for outgoing ones). In the former case, the configuration of the postfix forwarder can be entered directly in **smtp-proxy** configuration (in the `agent` section within the `smtp-forwarder` one) and the proper configuration files for postfix are generated automatically.

Kernun **smtp-proxy** does not provide any queuing service. If the mail is not deliverable to forwarders for all recipients, mail is rejected and resending is in clients' responsibility. If at least one delivery succeeds, mail is accepted and the DSN (Delivery Status Notification) message is constructed and sent to the original sender (if not null).

SSL support

The proxy supports secure communication via SSL/TLS protocols, see ssl(5).

Item `session-acl.client-ssl` switches on SSL/TLS on the client connection and sets various SSL/TLS parameters. If the connection from the client uses SSL/TLS, item `client-cert-match` defines the acceptable client certificates. If the client certificate does not pass the test, SSL/TLS connection establishment fails and the connection is closed. There are three possible modes of SSL/TLS usage:

immediate The SSL/TLS handshake is started immediately after accepting the client connection. This mode is configured by

`ssl` *name* `connection;` or simply `ssl` *name*`;`

Such a mode of SSL is typically configured on a special port different from the SMTP port 25 (usually 465).

mandatory The session is started in normal way. However, the **STARTTLS** command is expected to be used by the client. Without using it, only a subset of SMTP commands is accepted (e.g. the **MAIL** command is not allowed). This mode is configured by

`ssl` *name* `command required;`

optional The session is started in normal way. The **STARTTLS** command is offered to the client and the decision whether to use SSL/TLS or not is left to it. This mode is configured by

`ssl` *name* `command allowed;` or simply `ssl` *name* `command;`

If the client does not issue the **STARTTLS** command, session continues. If the client uses the command, proxy responds by the `220` response code and starts the SSL/TLS handshake. If it succeeds, the session is reset according to RFC 3207. If it fails, the connection is closed.

Item `smtp-forwarder.server-ssl` switches on SSL/TLS on the server connection and sets various SSL/TLS parameters. If the connection to the server uses SSL/TLS, item `server-cert-match` defines the acceptable server certificates. If the server certificate does not pass the test, SSL/TLS connection establishment fails and the request terminates with an error. There are again the three possible modes of SSL/TLS usage listed above:

immediate The SSL/TLS handshake is started immediately after connecting to the server. This mode is configured by

`ssl` *name* `connection;` or simply `ssl` *name*`;`

mandatory The session is started in normal way. However, if the **STARTTLS** command is not offered by the server, or its issuing fails, session is closed. This mode is configured by

`ssl` *name* `command required;`

optional The session is started in normal way. If the **STARTTLS** command is not offered by the server, session continues without trying to issue it. If the server supports SSL/TLS, the **STARTTLS** command is issued immediately. This mode is configured by

`ssl` *name* `command allowed;` or simply `ssl` *name* `command;`

Common Kernun Features

The proxy uses common Kernun mechanisms for listening on its sockets, accepting client connections, and managing its processes. It can also run in a chrooted environment and change its user identity upon startup. See also application(5), tcpserver(5) and tcpserver(7).

The proxy uses a common Kernun mechanism for network input/output. The configuration allows to specify several parameters like buffer sizes and timeouts, both for client and server connections. The parameters are set in configuration sections `client-conn` and `server-conn`. See netio(7) for details.

The proxy uses common Kernun mechanism for name resolving (see resolving(7)).

The proxy uses common Kernun mechanism for logging (see logging(7)).

- For every *session*, the SESSION-START (SMTP-801-I), the `session-acl` decision (SMTR-801-I) and the SESSION-END (SMTP-809-I) messages are logged.
- For every *mail*, a couple of MAIL-START (SMTR-811-I) and MAIL-END (SMTR-819-I) messages is logged.
- For every mail *recipient*, the `delivery-acl` decision message (SMTR-802-I), the `mail-acl` decision message (SMTR-803-I), the `doc-acl` decision messages (one SMTD-803-I for each MIME document) and the RCPT-RESULT (SMTS-815-I) messages are logged.

The proxy uses common Kernun mechanism for runtime monitoring. For more detailed information, see monitoring(7).

The proxy uses common Kernun mechanism for traffic shaping. For more detailed information, see traffic-shaping(7).

Program options

`-h` Display usage information and exit.

`-v` Print version information and exit.

`-d` ***`dbglev`*** Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

`-f` ***`cfgfile`*** Read configuration from *cfgfile*.

`-r` Resolve names in configuration prior to testing.

-t *test-expr* Test configuration according to given expression. Format of the *test-expr* is described in test-expr(5).

MAIL PROCESSING

Following steps are gone through when processing incoming mails:

1. When a connection from an SMTP client arrives, the configuration is searched for a matching `session-acl`. If the ACL says that the connection should be denied or there is no matching ACL, the proxy closes regularly the session.

2. According to the `session-acl` settings, the client identity is verified (see the Client verification section above). If the verification fails, the proxy closes regularly the session.

3. Proxy reads the **HELO/EHLO** command, checks it according to the RFC 2821, tries to resolve its argument (A DNS RR) in case of domain name and stores all results for further processing. The command is always responded by `250 OK` reply.

4. Proxy reads the **MAIL** command, checks it according to the RFC 2821, tries to resolve domain part of sender's email address, checks client identity according to the SPF (if configured, see above Section D) and stores all results for further processing. The command is always responded by `250 OK` reply.

   The domain name resolution proceeds as follows: The proxy tries to find MX DNS RRs for the domain name. If it succeedes, each mail exchanger is resolved again until at least one A DNS RR is found. If no MX DNS RR exists, the proxy tries to get an A DNS RR for the domain name. If no A DNS RR is found by either way, the sender name is treated as `unknown` for further processing (in `delivery-acl.sender` items). The whole resolution process has a single total timeout equal to the `conn-timeout` value in the sense of resolver(5) settings.

   For every mail, a unique internal message identification is generated. This *MSGID* is then used for identifying log messages, mail documents etc.

5. Proxy repeatedly reads **RCPT** commands, checks them according to the RFC 2821, tries to resolve domain part of recipient's email address (in the same manner as the sender's one) and then finds the appropriate `delivery-acl` taking paricular recipient and also HELO/MAIL check results into account. According to the `delivery-acl` settings, proxy responds to the command.

   > **Warning**
   >
   > If no `delivery-acl` is found, the session is aborted and the mail is not delivered to any addressee. The ACL set should be closed by an ACL with no entry conditions and a proper denial operation.

   There are several ways how to `deny` service for particular recipient:

abort The **RCPT** command is responded by given reply code and text and then the session is ended with `421` response code.

reject The **RCPT** command is responded by given reply, so as all following ones till the `DATA` command (if issued by client).

refuse The **RCPT** command is responded by given reply and processing of the mail regularly continues.

discard The **RCPT** command is responded by `250 OK`, but mail will not be sent to the recipient. This action has no impact to other recipients.

If the recipient is `accept`ed, the default behavior is to deliver the mail to him/her via proper `smtp-forwarder`. The list of forwarders is searched for the first one who serves domain of particular recipient. The item `deliver` can be used to choose another forwarder (`via` elem) or even another recipient (`to` elem).

If the response code was `250` (`accept` or `deny+discard` used), a copy of this mail can be sent to another person. This person's email address is given as a parameter of the `copy-to` item. This address is added to the set of recipients for further processing (searching for proper ACLs of phase 3) with the same `delivery-acl` as the originator has (no new search for it is done). The `smtp-forwarder` for the delivery will be determined by searching through forwarders' `domains` regardless of the `deliver` item of the original `delivery-acl`.

Under the same condition, the mail can be stored into the *quarantine* by the `quarantine` item. For easy controlling the mails in the quarantine, use the quarc.sh(1) tool.

6. When the **DATA** command comes, proxy checkes the number of accepted recipients and, if not zero, responds by `354` response code and starts receiving the mail. Otherwise, the proxy responds by `554` response code and waits for the next **MAIL** command.

7. The mail is read into the `mail-pool` directory under a name derived from the MSGID. It is stored as-is, without any changes, checking only the RFC 2821 rules (line length, leading dot dubbing etc.).

   By default, the proxy checks every line, whether its *normalized* length (line length not counting the leading dot duplicated for transparency and the CR+LF line separators) fits the 998 chars limitation. This check can be changed or switched off by the `session-acl.mail-line-len` configuration item. Even in this case, mail line length (unnormalized) cannot be greater than the `client-conn.recv-bufsize` and the `server-conn.send-bufsize` value.

   The mail `size-limit` is also checked during mail reading. Every line is counted in its normalized length plus two bytes (for the standard CR+LF line separator). This normalized size is then used in "statistical" log messages (SMTS-815-I and SMTR-819-I), too.

8. The mail MIME structure is decoded according to the RFC 2822, 2045 and others. All MIME documents are stored into the mail-pool and several checks are done. For detail configuration description, see mod-mail-doc(5).

The **smtp-proxy** can check mail documents by an *antivirus*. Antivirus checking is defined by item `use-antivirus` which selects a global `antivirus` section. See antivirus(5) for details about configuration of virus checking.

The **smtp-proxy** can check the whole mail by an *antispam*. In the current version of Kernun firewall, **SpamAssassin** is supported. Antispam checking is defined by item `use-antispam` which selects a global `antispam` section. See mod-antispam(5) for details about configuration of spam checking.

The **smtp-proxy** can check particular MIME types of mail body documents according to the rules defined in configuration, see doctype-identification(7) for details.

The **smtp-proxy** checks correctness of the mail headers and the MIME structure. Mails not conforming the RFCs are rejected. However, many clients do not respect RFC and if the security policy allows sending such mails you can enforce proxy to correct or even to pass them.

> **Warning**
>
> Read carefully the `mail-filter` section description in the mod-mail-doc(5) and set only *really necessary* exceptions to the RFC and preferably only in direction inside your organisation where you can check, in some extent, ability of mailservers to accept incorrect mails.

If a document header folded line is longer than the internal buffer size, the header is not processed and the mail is rejected. The buffer size is 16kB, by default, and it can be increased by the `client-conn.recv-bufsize` and `server-conn.send-bufsize` configuration directives.

If **smtp-proxy** forwards delivery error report, it checks (and corrects, if required) also the reported mail (if included). If the correction fails, it rejects whole mail. You can switch off this behavior by setting the `treat-rfc822-as-text` configuration directive that tells proxy to read included mails as regular texts.

9. After receiving the whole mail data, proxy finds the appropriate `mail-acl` taking all checks from previous steps into account. After that, for each MIME document, the appropriate `doc-acl` is searched for. These steps are repeated for every recipient (including the ones added by `copy-to` items), because mail processing can vary for different addressee (level 3 ACLs can use also sender's and recipient's email addresses as entry condition). In the case of changing the final recipient (by `deliver+to` item), the original recipient's address is used for this search.

   > **Warning**
   >
   > If no `mail-acl` or no `doc-acl` is found, the session is aborted and the mail is not delivered to any addressee. The ACL sets should be closed by two ACLs with no entry conditions and a proper denial operation.

In phase 3 ACL, following actions are available to `deny` the service for particular recipient:

`abort` The mail final `CRLF.CRLF` is responded by given reply and then the session is ended with the `421` response code.

`reject` The mail final `CRLF.CRLF` is responded by given reply, the session continues.

`cancel` The mail will not be sent to this recipient, given reply is stored as the *mail forwarding result* for this recipient and processing of the mail regularly continues.

`discard` The mail will not be sent to this recipient, but no failure is registered for this recipient so the client will assume successfull delivery.

Among all denial actions for one recipient ordered by all `mail-acl` and `delivery-acl` sections, the one with the highest severity is chosen to use.

If the mail/document is `accept`ed, there are several modifications that can be done: HTML filtering, document replacement, header modifications etc., see mod-mail-doc(5) for more details.

If the mail is not aborted, it can be stored into the quarantine by the `quarantine` item.

> **Warning**
>
> If you configure mail denial and some further processing (storing to quarantine, sending a copy etc.), be careful because without special handling, a repeated mail processing (when resent from quarantine or when sending to a `copy-to` address) will fall to the same level 3 ACL set. That's why you have to handle these cases by individual `mail-acl`s and/or `doc-acl`s *prepended* before the regular ACLs to avoid the denial.

10. If the mail is accepted to delivery, all forwarder channels are opened and the mail is forwarded to all recipients. While sending copies, the recipients are grouped together according to the set of phase 3 ACL used.

    For every such a group, the mail is reassembled from parts (excluding the case of signed documents) using the filtering operations specified in level 3 ACLs (see mod-mail-doc(5)).

    Moreover, some header modifications are done in following steps:

    (a) The `Return-Path` header (if any) is removed.

    (b) If the `stamp-filter` directive is used, all `Received` headers are removed and replaced by the `X-Kernun-Loop-Info` header to preserve the function of "Too many hops" checking (controlled by the `stamp-limit` directive).

    (c) If the antivirus check was completed, the result of it is reported by adding/replacing the `X-Kernun-Virus-Status` header. Notice that replacement of the `X-Kernun-Spam-*` headers (antispam check result) is done already in the phase of mail reading.

    (d) The `Content-Type` header is changed according to the value recognized by the *doctype-identification* tool (see doctype-identification(7)), if requested by the configuration directive `force-doctype-ident`.

(e) The header modifications ordered by the configuration directives `modify-header` are realized.

> **Warning**
>
> No check for correctness of changes requested is done. The resulting state is in administrator's responsibility.

(f) The header modifications required by the RFCs for given types, character sets etc. are realized.

(g) The `Subject` header is modified according to the configuration directive `prefix-subject`.

During mail reassembly, the RFC 2821 and 2045 rules are respected. Every line longer than 998 bytes (in the sense of normalized length — see above) is splitted into parts (converting the subdocument into MIME quoted-printable transfer encoding, if necessary). Subdocuments being encoded by some MIME encoding (`quoted-printable` or `base64`) are re-encoded using 76 chars long lines.

If **smtp-proxy** forwards signed document, it must not alter its content. That's why the standard way of processing mails (decomposition and reconstruction) cannot be used. Thus, the proxy uses original mail image saved on the beginning of processing, except two cases:

- Some changes were *forced* by the configuration (e.g. filtering or replacing some documents).
- The mail *violates* RFC in such extent that cannot be ignored by itself (e.g. wrong line length, incorrect header format etc.) and passing of such mails is not explicitly allowed by the administrator (see the `mail-filter` description in mod-mail-doc(5)).

In these cases the proxy does reconstruct mail (because site security has priority) and *signature* becomes not *valid*.

11. In the case of transient failure during forwarding, all servers available for particular forwarder channel are tried. In the case of permanent failure, or when all servers have failed, sending the copy is given up. If any forwarding server does not support 8bit transfer, the mail is converted to 7bit stream. If this is not possible, mail forwarding is rejected.

    The result of forwarding is stored for every recipient and in the case of failure of mail coming from a non-null sender, the DSN (Delivery Status Notification) message is constructed and sent to the original sender. This behavior and the content of the message can be partially controlled by the `dsn-mail-copy` and `omit-dsn` configuration items. In the case of forwarding mail to a forwarder without ESMTP DSN extension, the proxy sends also DSN RELAYED messages if requested by the clients.

12. If the mail was ordered to be stored into the quarantine, the image of it, stored at the beginning of processing, is moved or copied to the quarantine directory. Together with this copy, the quarantine control file is stored. It contains all available information about

sender, recipient, MIME structure, viruses etc. and also reasons for storing the mail into the quarantine.

13. The final response code is sent to the client and the proxy is waiting for a new mail, or the **QUIT** command.

BUGS

The Kernun **smtp-proxy** is a security proxy, not a regular MTA (Mail Transfer Agent), it does not provide for queuing facility, so it requires some regular MTA to be used as mail-forwarder.

The proxy does not support some obsoleted features (e.g. source routes).

The proxy does not support caching of SSL/TLS sessions.

If an excerpt of original mail is sent as part of the DSN (Delivery Status Notification) message, the excerpt is sent as-is (e.g. with only `LF` as end-of-line separator). The statistical messages then contain "real" sizes, not the normalized ones.

SEE ALSO

quarc.sh(1), triplicator(1), antivirus(5), application(5), listen-on(5), mod-antispam(5), mod-html-filter(5), mod-mail-doc(5), smtp-proxy(5), smtp-proxy.cfg(5), ssl(5), tcpserver(5), test-expr(5), SMTR-801(6), SMTR-811(6), SMTR-802(6), SMTR-803(6), SMTD-803(6), SMTS-815(6), SMTR-819(6), SMTP-809(6), SMTS-815(6), SMTR-819(6), SMTP-801(6), access-control(7), configuration(7), doctype-identification(7), host-matching(7), logging(7), monitoring(7), netio(7), resolving(7), tcpserver(7), time-matching(7), traffic-shaping(7), transparency(7)

NAME

sqlnet-proxy, test-sqlnet — Oracle SQL*Net Proxy

SYNOPSIS

sqlnet-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-sqlnet [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

The **sqlnet-proxy** provides proxying service for the Oracle SQL*Net Protocol. The proxy recognizes the initial Transparent Network Substrate (TNS — do not mistake this abbreviation for Trusted Network Solutions, a.s.) handshaking, after successful intialisation, it forwards requests and responses between client and server (checking TNS Layer). During this phase, the proxy checks database login attempts so that effective limitation of allowed users can be done.

The proxy reads its configuration file and starts listening on specified IP sockets (address/port couples), as specified in the `listen-on` configuration section (see listen-on(5)).

Format of the configuration file is described in sqlnet-proxy.cfg(5).

Program **test-sqlnet** tests syntax and partially semantics of configuration; for test expression syntax, see test-expr(5).

Access Control Lists

The current version of SQL*Net proxy uses two layers of ACL (see access-control(7)) named `session-acl` and `service-acl`.

When a connection (either transparent or non-transparent) arrives, proxy decides according to the first level of ACL whether to accept it or not. Then proxy waits for the connect (CN) packet. The total time until whole CN packet is received is limited to the `init-timeout`. After completing the CN packet, the configuration is consulted, proper `service-acl` is selected and according to it, connection is allowed or not. In addition to the general Kernun ACL criteria, the set of acceptable service names can be specified (`service-name` item, see sqlnet-proxy.cfg(5)).

Target server/port/service can be coded by the client in the `CONNECT_DATA.SERVICE_NAME` (or `CONNECT_DATA.SID`) aatributes by the following syntax *`service_name`*`@`*`server`*`[:`*`port`*`]`. Another way to do it is to code it directly to `ADDRESS.HOST` and `ADDRESS.PORT` attributes. The default port can be set in configuration (the `default-port` item).

If a `plug-to` directive is used in `session-acl` found, its value has precedence over the server used in the CN string. This means that the **sqlnet-proxy** will ignore the server given in the CN string and connect to the `plug-to` server.

Firewall administrator can choose the out-of-band method described in auth(7) for authenticating users on the proxy.

Redirections

If the remote server replies by the RD (redirect) string, the proxy itself tries to connect the advised server without co-operation with the client. Of course, the new connection is again checked against the `session-acl` set. The client is notified by the final response — accept (AC) or refuse (RF).

The **sqlnet-proxy** protects against the risc of an infinite loop in RD packets. The admin can set the maximal number of RD packets in one session (the `session-acl.redirections` item).

Special care should be taken in case of using the `plug-to` directive. If remote (plug-to) server uses RD string, result of `session-acl` checking can lead to re-connecting this server (and infinite loop, too). If intention of RD answer is only port switching, the port number of the `plug-to` directive value should be zero ("let port untouched").

TNS Listenner Protection

Several versions of TNS Listenner (lower layer of SQL*Net Server) have weak protection against intrusion attacks. The Kernun **sqlnet-proxy** can avoid some of them by configuring additional security attributes in the configuration:

`max-service-name-len` Several versions of TNS Listenner crash by the buffer overrun error when reading too long names. This item specifies the maximal length of the SID/SERVICE NAME attribute of the CN string allowed by the proxy.

`check-reserved-bits` Some versions of TNS Listenner crashes when the initial TNS handshaking packet has reserved bits filled with non-zero values. This item switches on/off checking of the reserved bits.

Database User Check

The **sqlnet-proxy** allows administrators to restrict which database users can use its service for particular ACL. However, the protocol is not freely published and there can be some problems with unknown versions of it. Thus, full support of this feature is provided for TNS (Transparent Network Substrate) versions 3.07 and 3.10 through 3.13. Proxying of other versions must be permitted by the `protocol-version` configuration item, however, database user checking is switched off in this case.

For known protocol versions, set of permitted database user can be specified by the `db-user` configuration item.

Common Kernun Features

The proxy uses common Kernun mechanism for listening on its sockets, forking new processes as needed and killing old redundant processes, optionally changing root directory and running with alternative user privileges. For more detailed information, see application(5) and tcpserver(7).

The proxy uses common Kernun mechanism for network input/output operations. Configuration allows for specifying several parameters like buffer sizes and timeouts, both for client and server connections. They can be included in the `client-conn` and `server-conn` configuration sections, respectively. For more detailed information, see netio(7).

The proxy uses common Kernun mechanism for name resolving (see resolving(7)).

The proxy uses common Kernun mechanism for logging (see logging(7)). For every session, the SESSION-START (SQLP-801-I), the ACL (SQLS-810-I and SQLS-820-I), the SESSION-INIT (SQLS-840-I) and the SESSION-END (SQLP-809-I) messages are logged.

Program Options

The program options are as follows:

-h Print usage information and exit.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

-r Resolve names in configuration prior to testing.

-t *test_expr* Test configuration according to given expression. Format of the *test expr* is described in test-expr(5).

BUGS

Currently, the **sqlnet-proxy** doesn't implement SQL Server user restrictions in its ACL criteria.

SEE ALSO

listen-on(5), application(5), sqlnet-proxy.cfg(5), test-expr(5), SQLP-801(6), SQLS-810(6), SQLS-820(6), SQLS-840(6), SQLP-809(6), access-control(7), configuration(7), host-matching(7), logging(7), netio(7), resolving(7), tcpserver(7), time-matching(7), transparency(7)

NAME

sysmgr — tool for Kernun installation, upgrade, backup, and restore

SYNOPSIS

Products Other than Kernun Branch Access

sysmgr [-b] applycfg *partition*

sysmgr [-b] backup [*partition*]

sysmgr [-b] backups

sysmgr [-b] checkcfg

sysmgr [-b] delimg *buildnum*

sysmgr [-b] images

sysmgr [-b] install [-n] *partition buildnum*

sysmgr [-b] resolve

sysmgr [-b] restore *partition backup*

sysmgr [-b] upgrade *partition backup*

sysmgr [-b] upgradecfg [*partition*]

Kernun Branch Access

sysmgr [-b] cancel

sysmgr [-b] commitcfg

sysmgr [-b] install

sysmgr [-b] loadcfg [*copy*]

sysmgr [-b] savecfg

sysmgr [-b] saveimg

DESCRIPTION

Program **sysmgr** is a command line tool for installing, upgrading, backing up, and restoring Kernun. It must be run from a running Kernun with standard disk partitioning (one disk containing

three system partitions and a data partition on the same or another disk). Operation of the program is controlled by a command specified as the first argument after the optional `-b`.

Commands (Except on Kernun Branch Access)

Applying Configuration (`applycfg`)

Generates and applies the configuration in a selected partition. Detects the system section used in the currently running system in order to select the system to apply in the target partition. This command is usually used during Kernun upgrade process.

Creating a Backup (`backup`)

Creates a backup of a selected partition and stores it in `/data/backup`. The backup file name contains the build number of the backed up installation, the system partition number, and the date and time of backup creation. The backup file contains all modifications (file creations, deletions, and changes) since Kernun was last installed on the selected partition. If a partition is not specified, the partition containing the currently running system is backed up.

A backup contains only selected files from a single system partition. The list of files to back up is in file `/etc/kernun-fsdb-include`. Other partitions, especially `/data`, should be backed up separately according to the local policy, for example, a daily backup of logs executed by **cron**.

Listing Backups (`backups`)

Lists all backup files in `/data/backup`.

Checking and Normalizing Configuration (`checkcfg`)

Checks that the configuration is valid. If the configuration is successfully loaded, it will be saved. Loading and saving normalizes the configuration, that is, it fixes parts of configuration that, although syntactically correct, could confuse the configuration upgrade process during a system upgrade.

Deleting Installation Images (`delimg`)

Deletes all installation images in `/data/dist` that have older build numbers than the build number specified by parameter *buildnum*.

Listing Installation Images (`images`)

Lists all installation images in `/data/dist`. Images that can be installed are marked by '*'.

Installation (`install`)

Installs Kernun. Parameters are number of a target system partition and build number of the image to install. The image must exist in `/data/dist` and must be installable (marked by '*' in the output of command `images`). An installable image is either a full image, or a patch image with an installable base image.

The target partition must be different from the partition that contains the currently running system. Booting from the target partition is enabled. The partition is made the default choice in the boot manager unless option `-n` is set. Label of the partition is changed to a standard label containing Kernun version, date and time of installation, and the build number.

Conflict Resolution (`resolve`)

Resolves restore conflicts according to file `/data/restore/resolve`. When restoring a backup on a Kernun build different from the one used for creating the backup, conflicts may occur. A conflict is a file that was changed in two ways: first between the builds used for creating and restoring the backup, and second after installation and before creating the backup. Conflicting files from the backups are not restored to the main file system, but are stored in respective places in the directory tree rooted at `/data/restore/conflicts`. List of conflicts is saved in `/data/restore/resolve`.

Before running `sysmgr resolve`, the list of conflicts should be edited. The first character on each line defines what to do with the file named on that line. Possibilities are:

- '+' — uses the file from the backup
- '−' — deletes the file
- '`.`' — keeps the existing file
- '`!`' — retains the conflict and postpones it to the next iteration of conflict resolution

Restore from a Backup (`restore`)

Restores data from a backup. Each backup should be restored to a newly installed Kernun with the same build number as was used for creating the backup. This build number is a part of the backup file name. Trying to unpack a backup on a different build fails in batch mode and asks for confirmation in interactive mode.

Upgrade (`upgrade`)

This is similar to `restore`, but but build number identity is not checked. Command `upgrade` is usually used during upgrading to a new version of Kernun. The standard upgrade procedure is backing up the current installation (`backup`), installing the new version to a free partition (`install`), restoring the backup to the new installation (`upgrade`), resolving eventual conflicts (`resolve`), converting configuration for the new version (`upgradecfg`), applying the converted configuration (`applycfg`), and reboot to the new installation.

Upgrade of Configuration (`upgradecfg`)

This command is usually invoked after upgrading to a new version of Kernun. It tries to convert an old Kernun configuration file to a valid configuration of the currently installed version. After conversion, the administrator should review the modified configuration file and then apply the configuration. If a partition is not specified, the configuration file in the partition containing the currently running system is upgraded.

Commands (Only on Kernun Branch Access)

Cancel Installation (`cancel`)

Cancels any previous `install` command. That is, a new system image will not be installed after reboot to the service system.

Commit the Configuration (`commitcfg`)

Reconfiguring a Kernun Branch Access stores configuration changes only to ramdisk. In order to retain the changes after reboot, they must be stored into persistent storage. It is done by this command.

Schedule Installation (`install`)

After the next reboot to the service system, the main system will be reinstalled. There is a factory-preloaded installation image for this purpose. It can be replaced (for system upgrade) by command **sysmgr saveimg**.

Obtain Configuration for Backup (`loadcfg`)

Outputs the configuration archive from persistent storage to the standard output. The archive file is in the `tbz` format. Output of this command can be stored as the backup of the complete system configuration.

There are two copies of configuration. They should be always identical. For the rare cases when they are different (after a system failure during committing the configuration), it is possible to select which copy to use (default is 1).

Save Configuration from Backup (`savecfg`)

Reads a configuration archive from the standard input and stores it to persistent storage. The input archive file should be a result of a previous **sysmgr loadcfg** command.

Prepare Installation Image (`saveing`)

Reads an installation image from the standard input and stores it for later command **sysmgr install**).

Options

`-b` If set, the command runs in the batch mode. No user interaction is done. If the command would asked the user for confirmation in the interactive mode, it terminates instead in the batch mode. This option is intended primarily for calling **sysmgr** from other programs (GUI) that handle user interaction themselves.

partition Number of a system partition, must be 1, 2, or 3.

buildnum A Kernun build number.

backup A name of a backup file. If a name without path is specified, the file is searched in directory `/data/backup`.

FILES

`/1 /2 /3` Directories used to mount system partitions other than the partition containing the currently running system.

`/data` Mount point for the data partition

`/data/backup` Directory for storing backups.

`/data/dist` Directory for storing installation images

`/data/log` Directory for storing log files

`/data/restore` Directory for restoring data from backups and solving conflicts

`/data/restore/conflicts` Directory containing onflicting files unpacked from a backup

`/data/restore/resolve` List of conflicts and instructions how to resolve them

`/etc/kernun-fsdb-include` List of files contained in backups

SEE ALSO

diskdb(1), bootmgr(8)

NAME

tcp-proxy, test-tcp — transparent generic TCP proxy

SYNOPSIS

tcp-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-tcp [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

The **tcp-proxy** is a generic TCP proxy that provides proxying protocols behaving according to standard TCP client/server model. The **tcp-proxy** assumes that servers are listening on a fixed TCP port number, clients connect to them using arbitrary source TCP port numbers and each session takes place within a single TCP connection; no other connections are involved in the session. The **tcp-proxy** is able to mediate either transparent connections to any number of servers according to the original destination address or to a fixed server, given its address in configuration.

The **tcp-proxy** reads its configuration file and starts listening on specified TCP sockets (address/port couples), as specified in the `listen-on` configuration directive. When a connection arrives, configuration is consulted and based on it, a decision is made whether this connection will be permitted. If permitted, several parameters of that connection may be set.

Format of the configuration file is described in tcp-proxy.cfg(5). Program `test-tcp` tests syntax and partially semantics of configuration; for test expression syntax, see test-expr(5).

Tcp-proxy uses single phase ACL (see access-control(7) manual page) named `session-acl`.

When a non-transparent connection arrives (i.e., a connection destined directly for one of the sockets **tcp-proxy** is listening on) and is allowed by policy, the proxy must be configured to connect to a specific remote server with `plug-to` configuration directive (see below).

When a transparent connection arrives (i.e., a connection destined for a real server transparently redirected to **tcp-proxy**, see transparency(7) for details), the proxy may decide to connect to the original destination server or to the `plug-to` given in configuration. If a `plug-to` directive is applicable for a transparent connection, it has precedence over the original destination. This means that **tcp-proxy** will ignore the original destination and connect to the `plug-to` server.

The **tcp-proxy** uses common Kernun mechanism for listening on its sockets, forking new processes as needed and killing old redundant processes, optionally changing root directory and running with alternative user privileges. For more detailed information, see application(5), tcpserver(5), and tcpserver(7).

The **tcp-proxy** uses common Kernun mechanism for network input/output operations. Configuration allows for specifying several parameters like buffer sizes and timeouts, both for client and server connections. They can be included in `client-conn` and `server-conn` configuration sections, respectively. For more detailed information, see netio(7).

The **tcp-proxy** uses common Kernun mechanism for logging. For more detailed information, see logging(7). For each connection, three statistical messages are logged: SESSION-START

(when the connection is established), ACL (informs about ACL selected for the connection), and SESSION-END (when the connection is closed).

The **tcp-proxy** uses common Kernun mechanism for policy decisions on arriving connections. It is described in access-control(7) and host-matching(7). For example, it is possible for **tcp-proxy** to use the real client's address or any specified address as source address for connection to server.

The **tcp-proxy** uses common Kernun mechanism for runtime monitoring. For more detailed information, see monitoring(7).

The **tcp-proxy** uses common Kernun mechanism for traffic shaping. For more detailed information, see traffic-shaping(7).

Firewall administrator can choose the out-of-band method described in auth(7) for authenticating users on the proxy.

The **tcp-proxy** allows to set several parameters in `session-acl`:

`plug-to` ***addr***; Specify server socket address to connect to. This applies both for transparent and non-transparent connections (in the latter case, it is even mandatory).

`max-bytes` ***number number***; Maximum number of octets transferred from server to client (first number) and from client to server (second number). The numbers are optional but they default to zero which has a special meaning "no limit to transfer size". When either of the limits gets exceeded, both client and server connections are closed by the proxy.

`max-time` ***seconds***; Maximum duration of a session. When this time elapses since the connection establishment, both client and server connections are closed by the proxy.

`cl2srv-idle-timeout` ***seconds***; Maximum idle time for client-to-server data. When no data are received from the client for this time interval, both client and server connections are closed by the proxy.

`srv2cl-idle-timeout` ***seconds***; Maximum idle time for server-to-client data. When no data are received from the server for this time interval, both client and server connections are closed by the proxy.

The **tcp-proxy** provides encryption and authentication using SSL/TLS protocols. SSL/TLS can be configured separately for the connection from the client and the connection to the server. Four variants are possible: either no connection uses SSL/TLS, or both connections use SSL/TLS, or only one of them uses SSL/TLS. The SSL/TLS mode can be used for building secure tunnels. A client communicates using plain TCP with **tcp-proxy**. The proxy connects via an SSL/TLS encrypted channel to another proxy across an untrusted network. The second proxy opens a plain TCP connection to a remote server.

SSL/TLS communication with clients can be turned on globally by `client-ssl-params` configuration directive. Parameter `client-ssl-timeout` limits the time interval between TCP connection establishent and finishing SSL/TLS handshake. When SSL/TLS is used, `session-acl` can be selected according to the values from the client certificate (`client-cert-match`). Enabling SSL/TLS is done by `server-ssl-params` in `session-acl`. Item `server-cert-match` defines requirements for the server certificate. If the certificate does not satisfy the requirements, the proxy terminates the session.

Options

-h Print usage information.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

-r Resolve names in configuration prior to testing.

-t *test_expr* Test configuration according to given expression. Format of the *test expr* is described in test-expr(5).

SEE ALSO

listen-on(5), application(5), tcp-proxy.cfg(5), tcpserver(5), test-expr(5), access-control(7), configuration(7), host-matching(7), logging(7), monitoring(7), netio(7), tcpserver(7), time-matching(7), traffic-shaping(7), transparency(7)

NAME

udp-proxy, test-udp — generic UDP proxy

SYNOPSIS

udp-proxy [-hv] [-d *dbglev*] -f *cfgfile*

test-udp [-hv] [-d *dbglev*] -f *cfgfile* [-r] [-t *test_expr*]

DESCRIPTION

The **udp-proxy** is a generic proxy for protocols based on the UDP.

The **udp-proxy** assumes that a server or servers listen on a fixed UDP port number and clients communicate from arbitrary source ports. The **udp-proxy** is able to mediate either transparent communication to any number of servers according to the original destination address or to a fixed address given in the configuration (the original destination address is one of the proxy addresses).

The **test-udp** program checks the syntax and partially the semantics of the configuration; for test expression syntax, see test-expr(5).

Format of the proxy configuration file is described in udp-proxy.cfg(5).

The **udp-proxy** uses common Kernun mechanism for policy decisions about received and sent datagrams. It is described in access-control(7) and host-matching(7). The proxy uses single-phase ACLs (`session-acl`) which are checked at the moment of a session establishment.

The session

The proxy uses an abstraction of *session*. The session is a distinctive period of communication between a single client (i.e. IP address and port) and a server (or server set), e.g. TFTP transfer of a file. The session is just a logical concept as there is nothing similar to the TCP connection in the UDP. There is no means of unambiguous detection of session termination (which TCP provides by closing a connection). Thus a number of conditions can be defined in the configuration to limit the session duration.

Session types

There are several types of sessions:

one-way Only datagrams in the direction from the client to the server are allowed. If a datagram from the server is received, it is not passed to the client and the session is immediately terminated. An example of one-way protocols is SYSLOG.

normal Sessions of this type are used for ordinary UDP protocols with bidirectional flow of datagrams between a client and a server, e.g. NFS. The datagrams from the client are forwarded to a fixed server IP address and port and replies from the server must come from the same IP and port. From the client's point of view, all replies come from the same IP address and port as were the original ones used by the client.

any-port This session type is prepared for situations when the server listens on a port, but it replies form another one. Thus, the first datagram from the server must have a proper source IP address but it can have an arbitrary source port number. The port is then fixed and used for all remaining datagrams of the session (in both directions) on the server side. On the client side, the original address and port remain in use. For example, TFTP servers can be configured to work this way.

any-sock This is similar to the `any-port` session type, extending it to the possibility of fixing not only the port of the server, but even the IP address. The first reply can come from any IP address and port. From the client's point of view, there is an important difference because in the transparent case, proxy keeps the actual server address (not the port) in replies. Be careful and check whether the actual servers are reachable from the client's network.

broadcast This is another communication model. A client sends all datagrams to an IP broadcast address and any number of servers in the target network can reply to the client. There is no address fixing like in the `any-*` cases and all replies are forwarded downstream with their actual addresses (keeping the original ports). Again, you have to check whether the actual servers are reachable from the client's network (in both transparent and non-transparent cases). This mode enforces nonstandard session handling and must not be mixed with normal unicast traffic. Similar mode uses e.g. DHCP, but the DHCP itself cannot be processed by the **udp-proxy** (due to lack of information about the original client when forwarding the reply).

Server connection

When a datagram arrives, the proxy checks its source and destination addresses and tries to assign the datagram to an existing session. If a match is found, the datagram is passed to a peer belonging to the session. If no session matches, the configuration ACL set is searched for, whether particular peers are allowed to communicate through the proxy. If this search fails, the datagram is dropped. Otherwise, a new session is established, the target server is selected and the outgoing socket to it is created or assigned.

When a non-transparent session is created (i.e., a session initiated by a datagram destined directly for one of the sockets **udp-proxy** is listening on), the proxy must be configured to communicate to a specific remote server with the `plug-to` configuration directive. Otherwise, the session initiation fails.

When a transparent session is created (i.e., a session initiated by a datagram destined to a real server "behind" the **udp-proxy**, see transparency(7) for details), the proxy either communicates with the original destination server or with the one defined by the `plug-to` directive. If the `plug-to` configuration directive is used in the ACL selected for a transparent session, it has precedence over the original destination. This means that the **udp-proxy** will ignore the original destination and communicate with the `plug-to` server.

When creating the communication channel to the server(s), the proxy uses a generic IP address and port. It is possible to force the source address of outgoing datagrams, either to a specific IP address, or to the address of the client (by the `source-address` configuration directive).

UDP based protocols sometimes require to use a specific source port, not a generic one. Like the IP address, also the source port number of outgoing datagrams can be forced in the configuration (by the `source-port` configuration directive). Of course, this setting can be done only if the client source address is forced, too. Otherwise the proxy should not distinguish among different server replies. Moreover, the proxy must be in this case configured for listening on the particular outgoing interface and forced source port, because it cannot create a separate server-side socket for every session (server-side sockets of different session from the same client would have the same source socket address in this case).

Firewall administrator can choose the out-of-band method described in auth(7) for authenticating users on the proxy.

Broadcasts

There are some limitations when using the **udp-proxy** for protocols with IP broadcast used as destination address.

If the proxy should process datagrams destined non-transparently to it using its client-side network broadcast address (or "limited" broadcast address 255.255.255.255), the proxy must listen on the address 0.0.0.0. The reason is that the proxy must be able to modify the source address of replies outgoing back to the client (for which, the socket bound to 0.0.0.0 is required).

If the proxy should process datagrams destined transparently to some network "behind" the proxy, it must listen on the address 0.0.0.0, or on the interface address transparently. Note that the proxy processes transparently even broadcast datagrams destined to a network directly connected to the proxy, if it is not the client-side network. Example: if the proxy operates on two interfaces with addresses 10.1.1.1 and 192.168.1.1, then a datagram sent from the address 10.1.1.2 to the address 192.168.1.255 will establish a transparent session.

The transparent broadcast mode is not allowed in the case when the incoming and outgoing interface is the same (e.g. in the case of two networks aliased to the same interface) and source-port forcing is on. The proxy has implemented a check against re-processing datagrams sent to the target network by itself (the datagrams will appear on the proxy's listening interface again). However, in this case, the proxy cannot recognize and distinguish a new datagram from the client and a datagram sent by itself. All attributes (interface name, source address, source port, target address and target port) of both of them match.

Data limitations

In the `session-acl`, several data limitations can be defined for the session; if any of them becomes true, the session is terminated.

`max-dgrams-in` The maximum number of datagrams passed in the direction from the server(s) to the client per session. If more datagrams are sent from the server, the session will be terminated.

`max-dgrams-out` The maximum number of datagrams passed in the direction from the client to the server(s) per session. If more datagrams are sent from the client, the session will be terminated.

max-bytes-in The maximum number of bytes transferred from the server(s) to the client during a session. When exceeded, the session will be terminated.

max-bytes-out The maximum number of bytes transferred from the client to the server(s) during a session. When exceeded, the session will be terminated.

Note that if both the `max-dgrams-in` and the `max-dgrams-out` limits are reached (not exceeded), the session is terminated immediately. This feature can significantly decrease number of "dead" sessions having exchanged sufficient number of datagrams and waiting for a timeout only. However, if some of the datagrams have lost (due to the unreliability of the UDP), such a session fails because a datagram resent by any peer will exceed the limitations.

Time limitations

In the `session-acl`, several time limitations can be defined for the session; if any of them becomes true, the session is terminated.

session duration (the `session` element of the `timeout` item) The maximum time for a session duration. The session will be terminated if `session` seconds elapse since the session establishment time.

server idle timeout (the `in` element of the `timeout` item) Timeout for datagrams from the server(s). If `in` seconds elapse without receiving a datagram from the server(s), the session will be terminated.

client idle timeout (the `out` element of the `timeout` item) Timeout for datagrams from the client. If `out` seconds elapse without receiving a datagram from the client, the session will be terminated.

session idle timeout (the `both` element of the `timeout` item) Timeout for datagrams regardless their direction. If no datagram belonging to a session is received for `both` seconds, the session will be terminated.

Note that unidirectional idle timeouts can be affected by the opposite peer latency. If, for instance, the client consumes almost all the time of the `in` timeout, the server may respond very quickly and in spite of that the timeout will be reached.

Configuration details

The **udp-proxy** uses common Kernun mechanisms for several sofisticated features.

The **udp-proxy** reads its configuration file and starts receiving datagrams on UDP sockets (address/port couples) specified by the `listen-on` configuration directive. It also maintains a list of active sessions (client/server or client/server-set couples), the maximum number of concurrently active sessions must be set by the configuration directive `udpserver.max-sessions`.

The proxy usually runs as one parent and several child processes, see udpserver(7) for details.

The **udp-proxy** uses common Kernun mechanism for listening on its sockets, optionally changing root directory and running with alternative user privileges. For more detailed information, see application(5).

For the details of the **udp-proxy** logging possibilities and configuration, see logging(7).
For the details of the **udp-proxy** traffic shaping, see traffic-shaping(7).

OPTIONS

-h Print usage information.

-v Display version information and exit.

-d *dbglev* Set debuging level to a specific number. Permitted values are 3 through to 9, 3 being the least and 9 the most verbose. See logging(7) for details. This setting is relevant only till configuration reading is finished.

-f *cfgfile* Read *cfgfile* for configuration information.

-r Resolve names in configuration prior to testing.

-t *test_expr* Test configuration according to given expression. Format of the *test expr* is described in test-expr(5).

SEE ALSO

application(5), test-expr(5), udp-proxy.cfg(5), udpserver(5), access-control(7), host-matching(7), logging(7), traffic-shaping(7), transparency(7), udpserver(7)

BUGS

Due to the nature of the UDP protocol, handling of sessions by **udp-proxy** cannot work perfectly under all circumstances.