The Samhain Host Integrity Monitoring System

The Samhain Host Integrity Monitoring System

This is version 2.4.5 of the Samhain manual.
Copyright © 2002-2023 Rainer Wichmann

This manual refers to version 4.5.0 of Samhain.

Table of Contents

Chapter 1. Introduction

samhain is a file and host integrity and intrusion alert system suitable for single hosts as well as for large, UNIX-based networks. samhain offers advanced features to support and facilitate centralized monitoring.

In particular, samhain can optionally be used as a client/server system with monitoring clients on individual hosts, and a central log server that collects the messages of all clients.

The configuration and database files for each client can be stored centrally and downloaded by clients from the log server. Using conditionals (based on hostname, machine type, OS, and OS release, all with regular expresions) a single configuration file for all hosts on the network can be constructed.

The client (or standalone) part is called samhain, while the server is referred to as yule. Both can run as daemon processes.

1. Backward compatibility

Samhain version 4.0 introduces a change in the baseline database format. This does not affect the client/server communication, i.e. it is possible to mix pre-4.0 and 4.0+ clients and server.

If you are using the Beltane II WebGUI, you need version 2.5+ to handle the new database format.

Finally, if you are planning to make use of the new features for integrating samhain into your change control process, you will need Samhain version 4.0+ for clients and sever, and Beltane II version 2.5+.

Chapter 2. Compiling and installing

Samhain as a client/server system

This chapter focuses on building a standalone samhain executable. For a client/server system, client and server executable are built from the same source, but with different options for the 'configure' script (see Section 4 (p. 4)).

Please refer to the chapter Chapter 6 (p. 72) for an explanation of the client/server setup.

1. Overview

Download:

```
sh$ wget http://la-samhna.de/samhain/samhain-current.tar.gz
```

Extract (and verify PGP signature):

```
sh$ gunzip -c samhain-current.tar.gz | tar xvf -
sh$ gpg --verify samhain-N.N.N.tar.gz.asc samhain-N.N.N.tar
sh$ gunzip -c samhain-N.N.N.tar.gz | tar xvf -
sh$ cd samhain-N.N.N
```

Configure:

```
sh$ ./configure
```

Compile:

```
sh$ make
```

Install:

```
sh$ make install
```

Customize:

```
sh$ vi /etc/samhainrc
```

Initialize the baseline database:

```
sh$ samhain -t init
```

Start the samhain daemon:

```
sh$ samhain -t check -D
```

2. Requirements

POSIX environment	Samhain will only compile and run in a *POSIX* operating system, or an emulation thereof (e.g. the free Cygwin POSIX emulation for Windows XP/2000).
ANSI C compiler and build system	You need an *ANSI C compiler* to compile samhain. The GNU C compiler (GCC) from the Free Software Foundation (FSF) is fine. If your vendor's compiler is ANSI compliant, you should give it a try, since it might produce faster code. Also you will need to have standard tools like make in your PATH (the make tool is part of the POSIX standard).
[OPTIONAL] GnuPG / signify-openbsd	If you want to use signed configuration and database files (this is an optional feature), either GnuPG (gpg) or the OpenBSD signify application (signify-openbsd on Debian-based Linux systems) must be installed.
[OPTIONAL] libacl/libattr	Samhain can check and verify POSIX ACLs (access control lists, on operating systems supporting them) and SELinux attributes (Linux). This feature is only compiled in if the required libraries and header files are present (e.g. on Linux the libacl/libattr development packages; in Debian these are named libacl1-dev, libattr1-dev).
[OPTIONAL] libz	Samhain can store the content of files in the baseline database (for files smaller than 9200 bytes after zlib compression). This feature is only available if the zlib library and header files are present (e.g. on Linux the libz development package; in Debian this is named zlib1g-dev).
[OPTIONAL] PCRE	Samhain can monitor logfiles of other applications, e.g. Syslog, Apache (or other webservers with similar log formats), Samba, or pacct (BSD-style process accounting). This extension requires the PCRE (Perl Compatible Regular Expressions) library, e.g. on linux the libpcre package (and for compiling, also the libpcre development package). In Debian, this would be libpcre3 and libpcre3-dev.

3. Download and extract

The current version of samhain can be downloaded from http://www.la-samhna.de/samhain/samhain-current.tar.gz. Older versions of samhain are available from the online archive. You should always make sure that you have a complete and unmodified version of samhain. This can be done by verifying the PGP signature (see below).

The downloaded tarball will contain exactly two files:

1. A tarball named samhain-N.N.N.tar.gz (N.N.N is the version number) containing the source tree, and

2. the PGP signature for this tarball, i.e. a file named samhain-N.N.N.tar.gz.asc. use of the dnmalloc allocator that is the default since samhain 2.4.5, and reverts to using the standard allocator provided by your system.

```
sh$ wget http://la-samhna.de/samhain/samhain-current.tar.gz
sh$ gunzip -c samhain-current.tar.gz | tar tvf -
-rw-r--r-- 500/100 920753 2004-05-24
19:57:55 samhain-1.8.8.tar.gz -rw-r--r-- 500/100 189
2004-05-24 19:58:29 samhain-1.8.8.tar.gz.asc
```

You might wish to verify the PGP signature now, in order to make sure that you have received a complete and unmodified version of samhain. All samhain releases are signed with the key 0F571F6C (Rainer Wichmann).

Key fingerprint = EF6C EF54 701A 0AFD B86A F4C3 1AAD 26C8 0F57 1F6C

```
sh$ gpg --keyserver pgp.mit.edu --recv-keys 0F571F6C
sh$ gpg --verify samhain-N.N.N.tar.gz.asc samhain-N.N.N.tar.gz
```

Now you can proceed to extract the source tarball:

```
sh$ gunzip -c samhain-N.N.N.tar.gz | tar tvf -
```

This will create a new subdirectory `samhain-N.N.N` under your current directory. You should **cd** into this subdirectory to proceed with configuring the source:

```
sh$ cd samhain-N.N.N
```

4. Configuring the source

Before you can start to compile, it is neccessary to configure the source for your particular platform and your personal requirements. This is done by running the **./configure** command in the source directory. If you type **./configure** with no options, the source will get configured with the default options. In particular, a standalone version of samhain will get built which uses the Filesystem Hierarchy Standard (FHS) for file/directory layout. This is *not* the standard GNU layout of 'everything under /usr/local'.

Paths

(A) samhain is a Filesystem Hierarchy Standard (FHS) compliant application. Thus the default directory layout is *not* the standard GNU layout (see Section 10 (p. 9)).

(B) samhain has a concept of *trusted users*, and will refuse to run if the path to critical files is writeable by users not in its list of trusted users (default: root, and the user who has started samhain). Please read Section 10.1 (p. 9) for details.

To change the defaults, **./configure** accepts a variety of command-line options and environment variables (use **./configure --help** for a complete list). The available command line options are listed and explained in Appendix A (p. 109) .

To configure a standalone version of samhain :

```
sh$ ./configure [more options]
```

Important remark on client/server use

Please read Chapter 6 (p. 72) if you intend to use samhain as a client/server system. Things will not work automagically just because you compiled a client and a server

version of samhain. In particular, clients need to *authenticate* themselves to the server, and special configure options are required if you want to keep the configuration file(s) and the baseline database(s) on the central server.

To configure a client version of samhain that can connect to a central server:

```
sh$ ./configure --enable-network=client [more options]
```

To configure a server version of samhain that will act as a central log server:

```
sh$ ./configure --enable-network=server [more options]
```

4.1. Some more configuration options

If you want to use any options/modules that are not enabled by default (e.g. because the majority of users do not require them, or because they require additional programs and/or libraries), at this point you need to specify such options:

- To compile in the module to check for SUID files (see Section 9 (p. 51)) use **./configure --enable-suidcheck**

- To compile in the module to monitor login/logout events (see Section 11 (p. 53)) use **./configure --enable-login-watch**

- To compile in the module to check mount options for mounted filesystems (see Section 12 (p. 55)) use **./configure --enable-mounts-check**

- To compile in the module to specify files relative to user home directories (see Section 13 (p. 55)) use **./configure --enable-userfiles**

- To compile in code for logging to an RDMS, (see Section 12 (p. 33)) use **./configure --enable-xml-log --with-database=** *oracle/mysql/postgresql*

- To compile in code for logging to the Prelude IDS, (see Section 9 (p. 30)) use **./configure --with-prelude**

- To use PGP-signed configuration files, (see Chapter 9 (p. 90)) use **./configure --with-gpg=** */path/to/gpg* . Please review Chapter 9 (p. 90) for further information and additional options to compile in the key fingerprint and/or the checksum of the **gpg** executable.

- To use OpenBSD signify-signed configuration files, (see Chapter 9 (p. 90)) use **./configure --with-signify=** */path/to/signify* . Please review Chapter 9 (p. 90) for further information and additional options to compile in the key checksum and/or the checksum of the **signify** executable.

- To compile samhain for use of the 'stealth' options to hide its presence, please review Chapter 10 (p. 94) for the available options.

- To configure a server version of samhain that will act as a central log server, use **./configure --enable-network=server**

- To configure a client version of samhain that can connect to a central server, use **./configure --enable-network=client** . Please refer to the chapter Chapter 6 (p. 72) for an explanation of the client/server setup, *in particular further options that you need if you want to store configuration files and baseline databases on the server*(see Section 5 (p. 75)).

5. Build

After configuring the source, to build samhain you just have to type the command:

```
sh$ make
```

The standalone/client executable (samhain) and the log server (yule) cannnot be compiled simultaneously. You need to run **./configure && make** separately for both.

If you want to use your native package manager for installation, you might rather want to build a binary package. samhain has support for RPM (rpm), Debian (deb), Gentoo (tbz2), HP-UX (depot), and Solaris packages. Instead of simply typing **make** , you need to type:

```
sh$ make rpm|deb|tbz2|depot|solaris-pkg
```

This will create a custom binary package according to the options that you used when configuring the source (see previous section). For more details, see Section 2 (p. 103) .

If you don't want to include documentation, you can instead use:

```
sh$ make rpm-light|deb-light|depot-light|tbz2-light|solaris-pkg-light
```

Finally, the Makefile supports building a portable (Unix) binary installer package based on the makeself installer ((c) 1998-2004 Stephane Peter). There will be no documentation included. Just type:

```
sh$ make run
```

6. Install

After successful compilation, you can install samhain by typing:

```
sh$ make install
```

The installation routine will not overwrite your configuration file from a previous installation.

Executables will be stripped upon installation. On Linux i386 and FreeBSD i386, the **sstrip** utility (copyright 1999 by Brian Raiter, under the GNU GPL) will be used to strip the executable even more, to prevent debugging with the GNU **gdb** debugger.

After installation, you will be offered to run **make install-boot** in order to install the init scripts that are required to start samhain automatically when your system (re-)boots. For many operating systems (Linux, *BSD, Solaris, HP-UX, IRIX), **configure** will generate init scripts, and **make install-boot** will figure out which of them to install, and where (if the correct distribution cannot be determined, none of them will be installed).

```
sh$ make install-boot
```

6.1. Important make targets

```
sh$ make install
```

Create the required directories (if not existing already), and install the compiled executable and the configuration file.

```
bash$ make DESTDIR=/somedir install
```

Install as if `/somedir` is the root directory. Useful for creating packages or installing for chroot (server).

```
sh$ make install-boot
```

Install runlevel start/stop scripts or create inittab entry (AIX) in order to start the daemon upon system boot. Supported on Linux, *BSD, Solaris, HP-UX, AIX(*), IRIX(*) [(*) untested].

```
sh$ make uninstall
```

Uninstall the executable and remove directories if empty. Does not uninstall the configuration file.

```
sh$ make purge
```

As **make uninstall** , but also remove the the configuration file.

```
sh$ make uninstall-boot
```

Uninstall the runlevel start/stop scripts.

> **Tip**
>
> You can save the script samhain-install.sh and use it for uninstalling if you ever want to remove samhain:
>
> ```
> sh$ samhain-install.sh purge
> sh$ samhain-install.sh uninstall-boot
> ```

7. Customize

samhain comes with default configuration files for several operating systems: `samhainrc.linux`, `samhainrc.solaris`, `samhainrc.freebsd`, `samhainrc.aix5.2.0` (and `yulerc` for the server). The installation routine will choose the one matching closest your system, or fall back to `samhainrc.linux`, if no good match could be found. However, all these configuration files are kept very general, and most probably you want to adjust settings like:

- which files/directories should be checked

- which logging facilities should be used

The default location of the configuration file is `/etc/samhainrc`(see Section 10 (p. 9)). To customize, type:

```
sh$ vi /etc/samhainrc
```

The default configuration file is heavily commented to help you. For a list of all runtime configuration directives, please have a look at Appendix C (p. 117) .

If you have any typos or other errors in your configuration file, samhain will log warning messages upon startup including the corresponding line number of the configuration file.

8. Initialize the baseline database

samhain works by comparing the present state of the filesystem agains a baseline database. Of course, this baseline database must be initialized first (and preferably from a known good state !). To perform the initialization (i.e. create the baseline database), type:

```
sh$ samhain -t init -p
info
```

(with *-p info*, messages of severity 'info' or higher will be printed to your terminal/console).

If the database file already exists, **samhain -t init** will *append* to it. This is a feature that is intended to help you operating samhain in a slightly more stealthy way: you can append the database e.g. to a JPEG picture (and the picture will still display normally - JPEG ignores appended 'garbage').

Note:

It is usually an error to run **samhain -t init** twice, because (a) it will *append* a second baseline database to the existing one, and (b) only the first baseline database will be used. Use **samhain -t update** for updating the baseline database. Delete or rename the baseline database file if you really want to run **samhain -t init** a second time.

9. Run samhain

After successful initialization of the baseline database, you can run samhain in 'check' mode by typing:

```
sh$ samhain -t check
```

To run samhain as a daemon, you can either use the command line option '-D', or set daemon mode in the configuration file with the option 'Daemon=yes'.

Tip

When testing samhain for the first time, you may want to use the command line option *--foreground* to run samhain in the foreground rather than as daemon. This allows to spot the reason for eventual problems much easier.

10. Files and directory layout

Tip

samhain has its own set of *trusted users*. Paths to critical files (e.g. the configuration file) must be writeable by trusted users only. Failure to ensure this (e.g. by compiling in an appropriate set of trusted users) is *one of the most frequent reasons for problems*. See below for details.

10.1. Trusted users and trusted paths

- *Trusted users* are *root* and the *effective user* of the process (usually, the effective user will be root herself). Additional trusted users can be defined in the configuration file (see Sect. Section 5 (p. 27) for an example), or at compile time, with the option

<div align="center">

bash$ **./configure --with-trusted=0,...**

</div>

- A *trusted path* is a path with all elements writeable only by trusted users. samhain requires the paths to the configuration and log file to be trusted paths, as well as the path to the pid file.

If a path element is group writeable, all group members must be trusted. If the path to the configuration file itself is writeable by other users than *root* and the *effective user* these must be defined as trusted already at compile time.

Note

The list of group members in /etc/group may be incomplete or even empty. samhain will check /etc/passwd(where each user has a GID field) in addition to /etc/group to find all members of a group.

10.2. Directory layout

samhain conforms to the FHS, which mandates a directory layout that is different from the default GNU layout (everything in subdirectories under /etc/local).

Tip

There is an option **./configure --enable-install-name=** *NAME* . When this option is used, not only the executable is installed as *NAME*, but also in all the paths, *samhain* is replaced with *NAME*.

Note

For the yule server, replace *samhain* with *yule* in the paths explained below.

The following table explains which directory layout results from **./configure --prefix=** *PREFIX*

sbindir	mandir	sysconfdir	localstatedir
PREFIX	(none)		
/usr/local/sbin	/usr/local/man	/etc	/var
PREFIX	USR (all capital)		
/usr/sbin	/usr/share/man	/etc	/var
PREFIX	OPT (all capital)		

sbindir	mandir	sysconfdir	localstatedir
/opt/samhain/bin	/opt/samhain/man	/etc/opt	/var/opt/samhain
PREFIX	/other		
/other/sbin	/other/share/man	/other/etc	/other/var

The file signature database will be written to *localstatedir*/lib/samhain/samhain_file, the pid file to *localstatedir*/run/samhain.pid, and the log file to *localstatedir*/log/samhain_log. In addition, yule writes an HTML status file to *localstatedir*/log/yule/yule.html

To get a more fine-grained control on the layout, the following **configure** options are provided

- *--with-config-file=FILE* — The path of the configuration file.

- *--with-log-file=FILE* — The path of the log file.

- *--with-pid-file=FILE* — The path of the pid file.

- *--with-data-file=FILE* — The path of the file signature database file.

- *--with-html-file=FILE* — The path of the HTML status file (server only).

10.3. Runtime files

10.3.1. Standalone or client

Purpose	Directory
Logfiles	*localstatedir*/log/
Data files	*localstatedir*/lib/samhain/
Pid file	*localstatedir*/run/

10.3.2. Server

Note

The server will drop root privileges after startup. I does not need write access to the data files, thus the data file directory is chmod 555 on installation. It does need write access to the log file directory. As the system logfile directory usually is owned by root, the install script will by default create a subdirectory and chown it to the unprivileged yule user. The PID file is written before dropping root.

Purpose	Directory
Logfiles	*localstatedir*/log/yule/
Data files	*localstatedir*/lib/yule/
Pid file	*localstatedir*/run/

10.4. Installed files

10.4.1. Standalone or client

File	Installed to	Mode
samhain	*sbindir*/samhain	700
samhainrc	*sysconfdir*/samhainrc	600

File	Installed to	Mode
samhain.8	*mandir*/man8/samhain.8	644
samhainrc.5	*mandir*/man5/samhainrc.5	644
(samhain_setpwd)	*sbindir*/samhain_setpwd	700
(samhain_stealth)	*sbindir*/samhain_stealth	700

10.4.2. Server

File	Installed to	Mode
yule	*sbindir*/yule	700
yulectl	*sbindir*/yulectl	700
yulerc	*sysconfdir*/yulerc	600
samhain.8	*mandir*/man8/yule.8	644
samhainrc.5	*mandir*/man5/yulerc.5	644
samhain_setpwd	*sbindir*/yule_setpwd	700

11. The testsuite

Samhain comes with a suite of verification/regression tests located in the `test/` subdirectory of the source tree.

The driver script is `test/test.sh`. Calling it without arguments will provide some usage information. The script should be called as:

```
bash$ test/test.sh [options] <test_number>
```

To get the list of possible tests, use:

```
bash$ test/test.sh -h
```

The recognized options are as follows:

1. *-q/--quiet* No output; success/failure is reported vi exit status only.

2. *-v/--verbose* Report additional information.

3. *-s/--stoponerr* Stop when a test fails.

4. *--no-cleanup* Don't clean up generated test data (useful to investigate the reason for a failure).

5. *--srcdir=...* Tell the script the location of the source tree (not necessary if run from the top source directory).

6. *--color=always/never/auto* Whether to use colour for output. Default is 'auto' (no colour if stdout is not a terminal).

7. *--really-all* This option enable additional test that are not run usually (see below).

The --really-all option

This option enables the following additional tests:

1. *clang/cppcheck* As part of the compile test suite (test 1), the cppcheck checker will be used, and the source will be compiled twice, once with the default compiler and second with the clang compiler.

2. *prelude logging* Logging to prelude will be tested as part of test 7. Requires prelude-manager, and requires that samhain is already registered as analyzer. This test is designed such that it should not interfere with an eventually running instance of prelude-manager.

3. *mysql/postgresql logging* Logging to mysql and/or postgresql will be tested with tests 13/14. Requires a running database with an existing default setup (database/user/password = samhain/samhain/samhain, table = log).

CAVEAT

The database tests (13/14) with --really-all will modify (i.e. log to) the database. These are the only tests that are not confined to the directory where the test is run.

Chapter 3. General usage notes

1. How to invoke

From the command line

- **samhain -t init [more options]** — To initialize the database

- **samhain -t check [more options]** — To check against the database

By default, samhain will *not* become a daemon, but stay in the foreground. Daemon mode must be set in the configuration file or on the command line. Also by default, samhain will *neither* initialize its file system database *nor* check the file system against it. The desired mode must be set in the configuration file or on the command line. A complete list of command line options is given in the appendix.

To start as daemon during the boot sequence

For Linux (Debian, Redhat, Gentoo, and SuSE), *BSD, Solaris, HP-UX, AIX, IRIX **make install-boot** will setup your system for starting the daemon upon system boot (if the correct OS/distribution cannot be determined, nothing will be done).

For any other system, you need to figure out by yourself how to start samhain during the boot sequence.

2. Using daemontool (or similar utilities)

samhain does not auto-background itself (to become a daemon) unless explicitely specified in the config file or on the command line. However, normally it runs in single-shot mode if not used as daemon. To cause samhain to enter the main loop while running in the foreground (as required if you want to use daemontool), you need to start with the option *-f* or *--forever*. Note that yule, the server, will always loop.

3. Controlling the daemon

As part of their boot concept, some systems have individual start/stop scripts for each service (daemon). As a minimum, these scripts take either 'start' or 'stop' as argument, sometimes also e.g. 'reload' (to reload the configuration), 'restart', or 'status' (check whether the daemon is running). While this is convenient, there are also a number of problems:

- Some systems do not have such start/stop scripts.

- There is no standard for the location of these scripts.

- There is no standard for the arguments such a script may take, neither for their interpretation (e.g.: on Linux distribution XYZ, do the start/stop scripts take 'status' as argument, and if, is the status reported by printing a message or by the exit status ?)

To provide a portable interface for controlling the samhain daemon, the executable itself can serve for this purpose (*only if invoked by the superuser*) The supported actions, which must be given as *first argument* on the command line, are:

- *start* Start samhain. Arguments after 'start' are passed to the process. Daemon mode will be enforced, as well as running in 'check' mode, irrespective of command line or config file settings.

- *stop* Stop the daemon. On Linux and Solaris, actually all running instances of samhain are stopped, even if no pid file is available.

- *restart* Stop and start.

- *reload or force-reload* Reload the configuration file.

- *status* Check whether the daemon is running.

Success/failure is reported via the exit status as follows: *0* Success. (On Linux/Solaris, *stop* will always be successful, on other systems only if the pid file is found.) *1* Unspecified error. *4* User had insufficient privilege. *5* Program is not installed. *7* Program is not running.

If the *status* command is given: *0* Program is running. *1* Program is dead and /var/run pid file exists. *3* Program is stopped. *4* Program status is unknown.

I.e., this interface behaves as mandated by the LSB Standard for init scripts.

4. Signals

On startup, all signals will be reset to their default. Then a signal handler will be installed for all signals that (i) can be trapped by a process and (ii) whose default action would be to stop, abort, or terminate the process, to allow for graceful termination.

For SIGSEGV, SIGILL, SIGBUS, and SIGFPE, a 'fast' termination will occur, with only minimal cleanup that may result in a stale pid file being left.

If the operating system supports the *siginfo_t* parameter for the signal handling routine (see **man sigaction**), the origin of the signal will be checked.

The following signals can be sent to the process to control it:

- *SIGUSR1* Switch on/off maximally verbose output to the console.

- *SIGUSR2* Suspend/continue the process, and (on suspend) send a message to the server. This message has the same priority as timestamps. This signal allows to run **samhain -t init -e none** on the client to regenerate the database, with download of the configuration file from the server, while the daemon is suspended (normally you would get errors because of concurrent access to the server by two processes from the *same host*).

- *SIGTERM* Terminate the process.

- *SIGQUIT* Terminate the server process after processing all currently pending requests from clients. Terminate the client process after finishing the current task (from the terminal, *SIGQUIT* usually is **Ctrl**+\).

- *SIGHUP* Re-read the configuration file. Note that it is not possible to override command-line options given at startup.

- *SIGTTIN / SIGABRT* Unlock the log file, wait three seconds, then proceed. At the next access, the log file will be locked again and a fresh audit trail -- with a fresh signature key -- will be started. This allows log rotation without splitting an audit trail. See Sect.~ Section 5.1 (p. 27) .

- *SIGTTOU* Perform a file check. Only client/standalone, and only in daemon mode.

- *SIGTSTP* Perform a silent file check. Only client/standalone, and only in daemon mode. The config file option SetFullSilent (boolean) can be used to switch on/off any informational message (e.g. start/end file check). The result of the file check can still be queried from the message queue via **samhain -w** For security reasons, silent scans are not taken into consideration for the time till next file check.

5. PID file

samhain generates a PID file if it is run as a daemon process. You can configure the path to the PID file at compile time, either explicitly using the **./configure --with-pid-file=FILE** option, or via the **./configure --prefix=PREFIX** option.

6. Wait on file check

As of version 4.0, samhain allows to perform a *wait on check* operation which waits on the end of the current filesystem scan and indicates the result via the exit status:

samhain -w *seconds*

The argument for this option is the timeout. The maximum timeout value is 86400 (one day). A timeout of zero is interpreted as *no timeout*. A *negative* timeout is interpreted in the following way:

1. First, wait until the Samhain daemon has initalized.

2. Then, wait for the end of the filesystem scan with a timeout of (abs(seconds) - 1). I.e. a timeout of -1 will become 0, a timeout of -601 seconds will become 600 seconds.

The exit status has the following meaning:

0   The deamon currently performs no file system scan, and no issues have been reported in the last scan.

1   The deamon currently performs no file system scan, and one or more issues have been reported in the preceding scan.

2   The wait operation has timed out while the deamon was performing a file system scan.

3   An error has occured. Usually this would be either an access error (the wait operation needs at least group privilege), or a non-negative argument has been given and the deamon has not yet initialized (or isn't running).

7. Log file rotation

samhain locks the logfile using a lock file. This lock file has the same path as the log file, with *.lock* appended. After sending SIGTTIN or SIGABRT to the samhain daemon, it will first finish its current tast (this may take some time), then unlock the log file (i.e. remove the `logfile.lock` file), wait three seconds, then proceed. Thus, to rotate the log file, you should use something like the following script:

```
#! /bin/sh

if test -f /usr/local/var/run/samhain.pid; then \
  PIN=`cat /usr/local/var/run/samhain.pid`; \
  /bin/kill -TTIN $PIN; \
  sleep 1; \
  AA=0; \
  while test "x$AA" != "x120"; do \
    AA=$(( AA + 1 )); \
    if test -f /usr/local/var/log/samhain_log.lock; then \
      sleep 1; \
    else \
      break; \
    fi \
  done; \
fi
mv /usr/local/var/log/samhain_log /usr/local/var/log/oldlog
```

If you use the 'logrotate' tool, you could use the following (untested):

```
/usr/local/var/log/samhain_log {
weekly
rotate 52
nocreate
missingok
compress
prerotate
if test -f /usr/local/var/run/samhain.pid; then \
  PIN=`cat /usr/local/var/run/samhain.pid`; \
  /bin/kill -TTIN $PIN; \
  sleep 1; \
  AA=0; \
  while test "x$AA" != "x120"; do \
    AA=$(( AA + 1 )); \
    if test -f /usr/local/var/log/samhain_log.lock; then \
      sleep 1; \
    else \
      break; \
    fi \
  done; \
fi
endscript
        }
```

8. Updating the file signature database

The samhain daemon only reads the file signature database on startup (also see Section 4.4 (p. 43) on this). You can update the database while the daemon is running, as long as you don't interfere with its logging (i.e. you should run **samhain -t update -l none** to make sure the log file is not accessed). Interactive updates are supported with the command line flag **--interactive** , updates using a list of 'good' files are supported with the command line flag **--listfile**=**_path_to_listfile_** , where *path_to_listfile* should be the absolute path to a text file listing the 'good' files (absolute paths, one per line).

If you are using samhain in client/server mode *and* keep the baseline database on the server, then there are two ways to update the database:

- The preferred method is to use the web-based (PHP4) beltane frontend, which allows to review client messages and to perform *server-side* updates of baseline databases.

- Temporarily **scp** the baseline database to the client, run **samhain -t update** , and **scp** the baseline database back to the server. If you want to keep the client daemon running during the update, you need to avoid concurrent access to the log file (use '-l none' for the update process). Also, you need to avoid concurrent access to the server (use '-e none' for the update process).

  If you *must* access the server concurrently (e.g. to download the configuration file for the update process), you need to suspend the client daemon process temporarily using SIGUSR2 (note that SIGSTOP/SIGCONT will not do what you want, because the daemon must inform the server that it is about to suspend). Use SIGUSR2 again to wake up the daemon from suspend mode.

9. Improving the signal-to-noise ratio

To get a good signal-to-noise ratio (i.e. few false alerts), you need to know which files should be checked, and which not (looking at the 'last modified' timestamp may be helpful, if in doubt).

To see how to set recursion depths, implement 'check all but xxx' policies etc., have a look at Section 4.1 (p. 39) .

As samhain runs a a daemon, it is capable to 'remember' all file system changes, thus you won't get bothered twice about the same problem.

10. Runtime options: command-line & configuration file

All command line options are described in Appendix B (p. 114) . Note that depending on the **./configure** options used for compiling, not all options may be available. You can get a list of valid options with **samhain --help** .

All settings in the configuration file, are described in Appendix C (p. 117) . Note that depending on the **./configure** options used for compiling, not all options may be available. If you are using unsupported options, samhain will log warning messages upon startup, including the line number of the offending line in the configuration file.

11. Remarks on the dnmalloc allocator

As a proactive security measure, since version 2.4.5, samhain ships with dnmalloc (Dnmalloc Site), a safer allocator that isn't vulnerable by heap buffer overflows and/or double free errors. I.e. with dnmalloc, it's not possible to exploit such errors to run arbitrary code.

If you want to disable dnmalloc, you can do so at compile time with **./configure --disable-dnmalloc [more options]** .

Unsupported operating systems

The dnmalloc allocator doesn't work on: OpenBSD (problems with pthread internals), Cygwin (also pthread internals), and 64bit FreeBSD. On 64bit AIX, you need to compile as a 32bit application, or to forego dnmalloc.

Speed and memory overhead of dnmalloc:

Speed	Dnmalloc is as fast, or sometimes faster than, the GNU libc allocator (which is based on ptmalloc).
Memory overhead	The actual memory overhead of dnmalloc is in the range of 20 per cent or less.
	On top of that, dnmalloc allocates a huge (128MB/256MB for 32bit/64bit systems) table on startup. *This is basically a non-issue*, since this table is only sparsely used, and hence contributes very little to the "reserved memory", i.e. the actual resource usage of dnmalloc.
	Both 'top' and 'ps' include this table in the 'virtual size' (columns VIRT/VSZ in top/ps) of an application using dnmalloc, thus giving the incorrect impression that physical swap storage would be required to back this table, if it's not resident in RAM (columns RES/RSS in top/ps). In fact, since most parts of this table are never used, no physical storage (neither RAM nor swap) is ever reserved for them. Note that this is not true anymore if (on Linux) you've switched off overcommiting completely (**echo 2 > /proc/sys/vm/overcommit_memory**).

Reserved memory

"Reserved momory" is the amount of memory that the operating system has reserved for an application, is backed by physical reasources (RAM or swap), and hence is not available for other applications. In other words, "reserved momory" is the actual resource usage of an application.

Because of deferred memory allocation, reserved memory can be less than what an application has asked for, since memory is only reseved when it is used.

12. Support / Bugs / Problems

If you have problems getting samhain to run, or think that you have encountered a bug, then please check the FAQ first.

If your problem is not anwered there, you can visit the user forum(which is *searchable*, by the way) and ask there for help (recommended for questions of probably general interest), or send email to `<support@la-samhna.de>`.

Please remember that a useful problem report should at least include the following three items:

• What did you do?

• What result did you expect?

• What result did you obtain instead?

Please be sure to provide relevant details, such as:

• your operating system, its release version, and the machine (**uname -srm**).

• your operating system, its release version, and the machine (**uname -srm**).

• the version of samhain that you are using, and the options that you have supplied to **configure** ,

• the command line options used to start samhain, and

• the `samhainrc` runtime configuration file.

• If you think you have encountered a bug, it is usually *very helpful* if you run samhain *in the foreground*(i.e. not as daemon) with the command line switch **-p debug** to get some more information about the problem.

It would be *even more helpful* if you first re-compile samhain with **configure --enable-debug** , and then run it with the command line switch **-p debug** (again, not as daemon, but *in the foreground*).

Please compress the output using **gzip** , and send it as attachment to `<support@la-samhna.de>`.

12.1. If samhain appears to hang indefinitely

If you have the impression that samhain hangs indefinitely, this could be due to a deadlock caused by some rare circumstance (it is not possible to fully test all possible configurations of samhain).

The most useful thing to do then is to recompile samhain with **--enable-debug=gdb** , start it in the foreground under the control of the gdb debugger, and get a backtrace as soon as it seems to hang:

```
bash$ gdb ./samhain
(gdb) run -t check -p info --foreground --forever
(samhain output)
[Ctrl-C]
(gdb) thread apply all
(gdb) backtrace
```

Chapter 4. Configuration of logging facilities

The configuration file for samhain is named `samhainrc` by default. Also by default, it is placed in `/etc`. (Name and location is configurable at compile time). The distribution package comes with a commented sample configuration file. The layout of the configuration file is described in more details in Section 1 (p. 117) .

1. General

Events(e.g. unauthorized modifications of files monitored by samhain) will generate *messages* of some *severity*. These messages will be logged to all logging facilities, whose *threshold* is equal to, or lower than, the severity of the message.

1.1. Severity levels

The following severity levels are defined:

Level	Significance
none	Keyword to switch off a looging facility (*).
debug	Debugging-level messages.
info	Informational message.
notice	Normal conditions.
warn	Warning conditions.
mark	Timestamps.
err	Error conditions.
crit	Critical conditions.
alert	Program startup/normal exit, or fatal error, causing abnormal program termination.
inet	Incoming messages from clients (server only).

(*) the keyword 'none' can only be used as a threshold for logging facilities, to indicate that no messages shall be logged via that facility.

Most events (e.g. timestamps, internal errors, program startup/exit) have *fixed* severities. The following events have *configurable* severities: (*) the keyword 'none' can only be used as a threshold for logging facilities, to indicate that no messages shall be logged via that facility.

- (server only) failure to resolve a client address (section *[Misc]*, option *SeverityLookup*)

- policy violations (for monitored files)

- access errors for files

- access errors for directories

- obscure file names (with non-printable characters) and/or invalid UIDs/GIDs (no such user/group)

- login/logout events (if samhain is configured to monitor them)

Severity levels for events (see Section 1.1 (p. 19)) are set in the *EventSeverity* and (for login/logout events) the *Utmp* sections of the configuration file.

In the configuration file, these can be set as follows:

```
[EventSeverity]
#
# these are policies
#
SeverityReadOnly=crit
SeverityLogFiles=crit
SeverityGrowingLogs=warn
SeverityIgnoreNone=crit
SeverityIgnoreAll=info
#
# these are access errors
#
SeverityFiles=err
SeverityDirs=err
#
# these are obscure file names
# and/or invalid UIDs/GIDs (no such user/group)
#
SeverityNames=info
#
# This is the section for login/logout monitoring
#
[Utmp]
SeverityLogin=notice
SeverityLogout=notice
# multiple logins by same user
SeverityLoginMulti=err
```

1.2. Classes

Events of related type are grouped into *classes*. For each logging facility, it is possible to restrict logging to a subset of these classes (see Section 3 (p. 22)). The available classes are:

Class	Significance
EVENT	Events to be reported (i.e. policy violations, login/logout).
START	Startup/stop messages.
STAMP	Timestamp (heartbeat) messages.
LOGKEY	The key to verify the signed log file.
ERROR	Error messages.
OTHER	Everything else (e.g. informational messages).
AUD	System calls (for debugging).

The aforementioned classes represent a new, simplified classification scheme since version 1.8.2. The previous scheme (listed below) will still work, and both can be mixed.

Class	Significance
AUD	System calls.
RUN	Normal run messages (e.g. startup, exit, ...)
STAMP	Timestamps and alike.
FIL	Messages related to file integrity checking.

Class	Significance
TCP	Messages from the client/server subsystem.
PANIC	Fatal errors, leading to program termination.
ERR	Error messages (general).
ENET	Error messages (network).
EINPUT	Error messages (input, e.g. configuration file).

1.3. Error message customization

It is possible to customize the initial part (the *header* of messages via the **MessageHeader=** `"for-mat"` directive. *Enclosing quotes are only required to protect leading/trailing space, and there is no need to escape quotes within the format string.* The following placeholders are recognized:

%S   Severity of the message.

%T   Timestamp of the message.

%C   Class of the message.

%F   Source file where the message originates from.

%L   Line number (in the source file) where the message originates from.

%E   An error code that may provide additional information in case of internal errors.

In the configuration file, these can be set as follows:

```
[Misc]
#
# This is the default without XML
#
MessageHeader="%S %T "
#
# Default for XML-style messages.
# Note that quotes within the format strings are NOT escaped in any way.
#
# MessageHeader="<log sev="%S" tstamp="%T"
```

2. Available logging facilities

samhain supports the following facilities for logging:

- *e-mail* — samhain uses built-in SMTP code, rather than an external mailer program. E-mails are signed to prevent forging.

- *syslog* — The system logging utility.

- *console* — If running as daemon, /dev/console is used, otherwise stderr. /dev/console can be replaced by other devices, including a FIFO.

- *log file* — Entries are signed to provide tamper-resistance.

- *log server* — samhain uses TCP/IP with strong authentication and signed and encrypted messages.

- *external* — samhain can be configured to invoke external programs for logging and/or taking some action upon certain conditions.

- *SQL db* — Currently samhain supports MySQL, PostgreSQL, Oracle, and unixODBC.

- *Prelude* — samhain can be compiled with support for the Prelude IDS, i.e. it can be used as a Prelude sensor.

Each of these logging facilities has to be activated by setting an appropriate threshold on the messages to be logged by this facility.

Note

In addition, some of these facilities require proper settings in the configuration file (see next sections).

3. Activating logging facilities and filtering messages

All messages have a *severity* level (see Section 1.1 (p. 19)) and a *class* (see Section 1.2 (p. 20)), with somewhat orthogonal meaning:

The *severity* ranks messages with respect to their importance. Most events (e.g. timestamps, internal errors, program startup/exit) have *fixed* severities. However, as importance sometimes is a matter of taste, some events have *configurable* severities (see Section 1 (p. 19)).

Classes refer to the purpose/category of a message. As such, they should (ideally) be useful to exclude messages that are not interesting in some context (e.g. startup/stop messages may seem useless noise if samhain is run from cron).

Obviously, as *severity* is a rank, the most natural way to exclude unwanted messages is to set a *threshold*. On the other hand, as the message *class* is a category, the most natural way to exclude messages is to *list* those message classes that you want.

Messages are only logged to a log facility if their severity is at least as high as the threshold of that facility, and their class is one of those wanted (by default: all). Thresholds and class lists can be specified individually for each facility.

Switching on/off

Most log facilities are *off by default*, and need to be enabled by setting an appropriate threshold.

A threshold of *none* switches off the respective facility.

Logging of client messages by the server

By default, messages received by the server are treated specially, and are always logged to the logfile, and never to mail or syslog. If you don't like that, use the option *UseClientSeverity=yes*(section *[Misc]*).

Thresholds and class lists are set in the *Log* section of the configuration file. For each threshold option `Facility`**Severity** there is also a corresponding option `Facility`**Class** to limit that facility to messages within a given set of class. The argument must be a list of valid message classes, separated by space or comma.

Actually, the `Facility`**Severity** can take a list of severities with optional specifiers '*', '!', or '=', which are interpreted as 'all', 'excluding', and 'only', respectively. Examples: specifying '*' is equal to specify 'debug'; specifying '!*' is equal to specifying 'none'; 'info,!crit' is the range from 'info' to 'err' (excluding crit and above); and 'info,!=err' is info and above, but excluding (only) 'err'. This is the same scheme as used by the Linux syslogd (see man 5 syslogd).

System calls: certain system calls (execve, utime, unlink, dup (+ dup2), chdir, open, kill, exit (+ _exit), fork, setuid, setgid, pipe) can be logged (only to console and syslog). You can determine the set of

system calls to log via the option **LogCalls**= *call1, call2, ...* . By default, this is off (nothing is logged). The priority is *notice*, and the class is AUD.

Example:

```
[Log]
#
# Threshold for E-mails (none = switched off)
# MailSeverity=none
#
# Threshold for log file
#
LogSeverity=err
LogClass=RUN FIL STAMP
#
# Threshold for console
#
PrintSeverity=info
#
# Threshold for syslog (none = switched off)
#
SyslogSeverity=none
#
# Threshold for logging to Prelude (none = switched off)
#
PreludeSeverity=none
#
# Threshold for forwarding to the log server
#
ExportSeverity=crit
#
# Threshold for invoking an external program
#
ExternalSeverity=crit
#
# Threshold for logging to a SQL database
#
DatabaseSeverity=err
#
# System calls to log
#
LogCalls=open, kill
```

4. E-mail

It is possible to define email recipients at compile-time, but it is also possible to define recipients, or aliases (lists of recipients) in the configuration file. Each recipient (list) definition starts with either:

SetMailAddress= *recipient*

or:

SetMailAlias= *listname: addresslist*

Filters and/or a threshold severity for the recipient (list) may follow. The definition of a recipient is ended (a) explicitly when terminated with the line **CloseAddress** , or (b) implicitly when another recipient (list) definition is started.

Items that can/must be configured are:

Recipients address	**SetMailAddress=** *`username@hostname`*

Each address must on a separate line in the configuration file.

> ### Tip
>
> it is recommended to use numerical IP addresses instead of host names (to avoid DNS lookups).

Recipients address list	**SetMailAlias=** *`listname`*: *`addresslist`*

Define an alias for a list of (already defined) recipients. The format is *listname*":" *addresslist*, where addresses in *addresslist* can be separated by comma, tab, or space. Logging threshold and filters (see below) can be set for a list as for an individual recipient, but will take effect only for email that is specifically targeted at the list (e.g. via a per-queue rule in the logfile monitoring module).

Logging threshold	**SetAddrSeverity=** *`severity`*

This defines a logging threshold severity for the last defined recipient (list). The syntax is the same as for **MailSeverity** .

> ### MailSeverity and SetAddrSeverity
>
> The MailSeverity setting in the [Log] section defines an upper bound for *all recipients*. Messages not included by the MailSeverity setting will never be emailed.

NOT Filter	**SetMailFilterNot=** *`list_of_regexes`*

Defines a filtering condition for the last defined recipient (list). If there is no recipient (list) defined yet, it applies to the compiled-in recipients.

List items are POSIX regular expressions. As whitespace (blank or tab) is a valid separator in a list, strings with whitespace must be enclosed in single or double quotes. If a string begins with a double quote, enclose it in single quotes (and vice versa).

If used, then NONE of the regular expressions in *list* can occur in a message, otherwise it will not be sent by email.

AND Filter	**SetMailFilterAnd=** *`list`*

> ### Order of evaluation
>
> AND conditions are evaluated after all NOT conditions.

If used, then ALL strings in *list* must occur in a message, otherwise it will not be sent by email. The syntax is the same as for **SetMailFilterNot** .

OR Filter	**SetMailFilterOr=** *list*

Order of evaluation

OR conditions are evaluated after all AND conditions.

If used, then AT LEAST ONE of the strings in *list* must occur in a message, otherwise it will not be sent by email. The syntax is the same as for **SetMailFilterNot** .

Closing a recipient (list) definition	**CloseAddress**

This explicitly closes the definition of a recipient (list). However, this is optional syntactic sugar (i.e. not really required), since recipient (list) definitions are closed implicitly by the beginning of another recipient (list) definition (i.e. **SetMailAddress** or **SetMailAlias**).

Relay host / Mail exchanger	**SetMailRelay=** *mail.some_domain.com*

You may need this option because some sites don't allow outbound e-mail connections from any arbitrary host. If the recipient is offsite, and your site uses a mail relay host to route outbound e-mails, you need to specify the relay host.

Maximum interval	**SetMailTime=** *86400*

You may want to set a maximum interval between any two consecutive e-mails, to be sure that samhain is still 'alive'.

Maximum pending	**SetMailNum=** *10*

Messages can be queued to send several messages in one e-mail. You may want to set the the maximum number of messages to queue. (Note: messages of highest priority (alert) are always sent immediately. At most 128 messages can be queued.

Multiple recipients	**MailSingle=** *yes/no*

If there are multiple recipients, whether to send a single mail with the recipient list, or send multiple mails. If all recipients are on same domain, a single mail may suffice, otherwise it depends on whether the mail server supports forwarding (for security, most don't).

Subject line	**MailSubject=** *string*

Here, *string* may contain the placeholders %T, %H, %S, and/ or %M that will get replaced by the time, hostname, message severity and message text, respectively. The default subject line is equivalent to "%T %H". This option may be useful if you want to send emails to an email-to-sms gateway.

Sender	**SetMailSender=** *string*

Here, *string* is the address that is inserted in the From: field. If a name without domain is given (i.e. without '@xyz.tld'), the FQDN of the local host will be added automatically.

SMTP port	**SetMailPort=** *port_number*

This option allows to specify a custom port for SMTP (the default is 25).

Example:

```
[Misc]
#
# Do not send messages about added files, and startup messages.
# We have no recipient defined yet, thus this applies to
# compiled-in recipients only (if there are any).
#
SetMailFilterNot = 'POLICY ADDED', START
#
# E-mail recipient (offsite in this case).
#
SetMailAddress=username@host.some_domain.com
SetMailFilterNot = LOGKEY
CloseAddress
#
# Need a relay host for outgoing mail.
#
SetMailRelay=relay.mydomain.com
#
# Number of pending mails.
#
SetMailNum=10
#
# Maximum time between e-mails.
# Want a message every day, just to be sure that the
# program still runs.
#
SetMailTime=86400
#
# Do not send messages about added files, and startup messages
#
SetMailFilterNot = 'POLICY ADDED', START
#
# To all recipients in a single mail.
MailSingle=yes
```

4.1. E-mail reports and their integrity

The subject line contains timestamp and local hostname, which are repeated in the message body. samhain uses its own built-in SMTP code rather than the system mailer, because in case of temporary connection failures, the system mailer (e.g. sendmail) would queue the message on disk, where it may become visible to unauthorized persons.

During temporary connection failures, messages are stored in memory. The maximum number of stored messages is 128. samhain will re-try to mail every hour for at most 48 hours. In conformance with RFC 821, samhain will keep the responsibility for the message delivery until the recipient's mail server has confirmed receipt of the e-mail (except that, as noted above, after 48 hours it will assume a permanent connection failure, i.e. e-mailing will be switched off).

The body of the mail may consist of several messages that were pending on the internal queue (see Section 2 (p. 21)), followed by a signature that is computed from the message and a key. The key is initialized with a random number, and for each e-mail iterated by a *hash chain*.

The initial key is revealed in the first email sent (obviously, you have to believe that this first e-mail is authentic). This initial key is not transmitted in cleartext, but encrypted with a one-time pad (Section 2 (p. 106)).

The signature is followed by a unique identification string. This is used to identify separate audit trails (here, a *trail* is a sequence of e-mails from the same run of samhain), and to enumerate individual e-mails within a trail.

The mail thus looks like:

```
   -----BEGIN MESSAGE-----
 first message
second message
...
-----BEGIN SIGNATURE-----
signature
ID TRAIL_ID:hostname
-----END MESSAGE-----
```

Integrity verification

To verify the integrity of an e-mail audit trail, a convenience function is provided:

samhain -M */mailbox/file/path*

The mailbox file may contain multiple and/or overlapping audit trails from different runs of samhain and/or different clients (hosts).

CAVEATS

Verification will fail, if the compiled-in key of the verifying executable is different from the one that generated the message(s) (see Section 2 (p. 106)).

If you use a pre-compiled executable from some binary distribution, be sure to read Section 2 (p. 106) carefully.

5. Log file

Trusted users	**TrustedUser=** *username*
	If some element in the path to the log file is writeable by some-one else than *root* or the *effective user* of the process, you have to include that user in the list of *trusted users*(unless their UIDs are already compiled in).
Separate log files for clients	**UseSeparateLogs=** *yes/no*
	Only relevant on the server. Use a separate log file for (reports from) each client. The root name of these log files will be the same as the main log file, with the client name appended.

5.1. The log file and its integrity

The log file is named samhain_log by default, and placed into /var/log by default (name and location can be configured at compile time). If samhain has been compiled with the **./configure -- enable-xml-log** option, it will be written in XML format.

Note

If you have compiled for stealth (Chapter 10 (p. 94)), you won't see much, because if obfuscated, then both a 'normal' and an XML logfile look, well ... obfuscated. Use **samhain -jL** `/path/to/logfile` to view the logfile.

The log file is created if it does not exist, and locked by creating a *lock file*, which has the same path as the logfile, with a ".lock" appended. The lock file holds the PID of the process, which allows samhain to recognize and remove a stale lock if there is no process with that PID.

On the log server, it is possible to use separate log files for individual clients. This can be enabled with **UseSeparateLogs=** `yes/no` in the Misc section of the server configuration file. No locking will be performed for client files (only one instance of the server can listen on the TCP port, thus there will be no concurrent access).

The directory where the logfile and its lock file are located must be writeable only by trusted users (see Section 10.1 (p. 9)). This requirement refers to the *complete* path, i.e. all directories therein. By default, only *root* and the *effective user* of the process are trusted.

Audit trails (sequences of messages from individual runs of samhain) in the log file start with a *[SOF]* marker. Each message is followed by a signature, that is formed by hashing the message with a key.

The first key is generated at random, and sent by e-mail, encrypted with a one-time pad as described in the previous section on e-mail. Further keys are generated by a hash chain (i.e. the key is hashed to generate the next key). Thus, only by knowing the initial key the integrity of the log file can be assured.

The mail with the key looks like:

```
-----BEGIN MESSAGE-----
message
-----BEGIN LOGKEY-----
Key(48 chars)[timestamp]
-----BEGIN SIGNATURE-----
signature
ID TRAIL_ID:hostname
-----END MESSAGE-----
```

Integrity verification

To verify the log file's integrity, a convenience function is provided:

samhain -L `/log/file/path`

When encountering the start of an audit trail, you will then be asked for the key (as sent to you by e-mail). You can then: (i) hit **return** to skip signature verification, (ii) enter the key (without the appended timestamp), or (iii) enter the path to a file that contains the key (e.g. the mail box).

If you use option (iii), the path must be an absolute path (starting with a '/', not longer than 48 chars. For each audit trail, the file must contain a two-line block with the ----- *BEGIN LOGKEY*----- line followed by the line (*Key(48 chars)[timestamp]*) from the mail. Additional lines before/after any such two-line block are ignored (in particular, if you collect all e-mails from samhain in a mailbox file, you can simply specify the path to that mailbox file).

CAVEATS

Verification will fail, if the compiled-in key of the verifying executable is different from the one that generated the message(s) (see Section 2 (p. 106)).

If you use a pre-compiled executable from some binary distribution, be sure to read Section 2 (p. 106) carefully.

6. Log server

| Server address | **SetLogServer=** `my.server.address` |

You have to specify the server address, unless it is already compiled in. It is possible to specify a second server that will be used as back-up.

Note

If you want to store the configuration file on the server, the server address must be compiled in.

| Throughput throttling | **SetThrottle=** `milliseconds` |

An option to throttle the throughput when downloading the database from the server. The allowed maximum of 1000 msec throttles to about 64 kB/sec, less throttle means higher throughput.

6.1. Details

During temporary connection failures, messages are stored in a FIFO queue in memory. The maximum number of stored messages is 128. After a connection failure, samhain will make the next attempt only after a deadtime that starts with 1 sec and doubles after each unsuccessful attempt (max is 2048 sec). A re-connection attempt is actually only made for the next message after the deadtime -- you should send timestamps (i.e. set the threshold to *mark*) to ensure re-connection attempts for failed connections.

It is possible to specify two log servers in the client configuration file. The first one will be used by default (primary), and the second one as fallback in case of a connection failure with the primary log server.

7. External facilities

samhain can invoke external scripts/programs for logging (i.e. to implement support for pagers etc.). This is explained in detail in Chapter 7 (p. 83) .

8. Console

Up to two console devices are supported, both of which may also be named pipes. If running as daemon, samhain will use /dev/console for output, otherwise stdout. On Linux, *_PATH_CONSOLE* will be used instead of /dev/console, if it is defined in the file /usr/include/paths.h.

You can override this at compile time, or in the **[Misc]** section of the configuration file with the **SetConsole=** `device` option. Up to two console devices are supported, both of which may also be named pipes (use the **SetConsole** option twice to set both devices).

Switching off

Invariably, some users set **SetConsole=** `/dev/null` to switch off console logging. This is highly ineffective, as the device will be opened, and the message written to it, for

every log message. The correct way is to use **PrintSeverity**= *none* in the *[Log]* section of the configuration file (or the command line switch '-p none').

9. Prelude

REQUIREMENTS

This facility requires that you have compiled with the *--with-prelude* option to include support for prelude. Of course you need the libprelude client library for this to work.

Note

The following configuration options can only be used. They should be placed the [Misc] section of the configuration file, if you use them. The 'PreludeMapTo...' options do not affect in any way whether a message is reported by samhain to the prelude manager (for this there is 'PreludeSeverity' in the [Log] section); they only affect the 'Impact severity' shown on the prelude side.

PreludeProfile	**PreludeProfile**= *profile_name*

Specify the profile to use. The default is 'samhain'.

PreludeMapToInfo	**PreludeMapToInfo**= *list of samhain severities*

The severities that should be mapped to impact severity 'info' for prelude. (default: none).

PreludeMapToLow	**PreludeMapToInfo**= *list of samhain severities*

The severities that should be mapped to impact severity 'low' for prelude. (default: debug, info).

PreludeMapToMedium	**PreludeMapToMedium**= *list of samhain severities*

The severities that should be mapped to impact severity 'medium' for prelude. (default: notice, warn, err).

PreludeMapToHigh	**PreludeMapToHigh**= *list of samhain severities*

The severities that should be mapped to impact severity 'high' for prelude. (default: crit, alert).

9.1. Prelude-specific command-line options

The following prelude-specific command-line options are accepted:

1. *--prelude* Prelude generic options are following. This option must be given before the following options are used.

2. *--profile <arg>* Profile to use for this analyzer

3. *--heartbeat-interval <arg>* Number of seconds between two heartbeats

4. *--server-addr <arg>* Address where this sensor should report to (addr:port)

5. *--analyzer-name <arg>* Name for this analyzer

9.2. Registering to a Prelude manager

Sensor name/profile

The default sensor name/profile is 'samhain'. However, version 2.0.6 of samhain still had 'Samhain' For versions of samhain later than 2.0.6, there is an option **PreludeProfile=***profile* (in the [Misc] section) to set a user-defined name/profile.

In order to register samhain as a Prelude sensor, you need to run on the sensor host and on the manager host the **prelude-admin** command.

```
sensor # prelude-admin register samhain "idmef:w admin:r" <manager host> \
--uid=prelude --gid=prelude

You now need to start "prelude-admin" registration-server on 127.0.0.1:
example: "prelude-admin registration-server prelude-manager"

Enter the one-shot password provided on 127.0.0.1:
```

```
manager # prelude-admin registration-server prelude-manager

The "76g4h8au" password will be requested by "prelude-admin register"
in order to connect. Please remove the quotes before using it.

Generating 1024 bits Diffie-Hellman key for anonymous authentication...
Waiting for peers install request on 0.0.0.0:5553...
Waiting for peers install request on :::5553...
```

You now have to type in the *one-shot password* generated on "manager" at the password prompt on "sensor", (twice, for confirmation). Then on "manager" you will be asked to approve the registration. Type 'y', and you are finished.

The configuration file for the samhain sensor is /etc/prelude/profile/samhain/config

10. Using samhain with nagios

After running **./configure** , you will find the script check_samhain.pl in the subdirectory scripts/ of the samhain distribution, which you can copy to the nagios libexec/ directory. Before doing so, you may want to edit the script to set the correct path to the libexec/ directory in the following line:

```
use lib "/usr/local/nagios/libexec";
```

The following recipe to use this script has been kindly provided by kiarna:

Nagios runs as user 'nagios'. However, in order to check the filesystem, you typically want to run samhain as 'root'. You can use sudo to fix this problem. In your /etc/sudoers file, add the line:

```
nagios ALL = NOPASSWD:/path/to/check_samhain
```

Next, add the service to the nagios file `objects/commands.cfg`:

```
# 'check_samhain' command definition
define command{
command_name check_samhain
command_line /usr/bin/sudo -u root $USER1$/check_samhain -t 100
      }
```

Another option would be to install the check_samhain script with SUID permissions instead of using sudo (this requires that the suidperl package is installed and that samhain has been compiled with **./ configure --enable-suid ...**

```
# 'check_samhain' command definition
define command{
command_name check_samhain
command_line $USER1$/check_samhain -t 100
 }
```

Checking the filesystem may take some time, so you may want to increase the nagios plugin timeout by changing the following line in `nagios.cfg` from 60 to 600:

```
service_check_timeout=600
```

Then add the service to the appropriate section in the nagios `service.cfg` file.

11. Syslog

samhain will translate its own severities into *syslog priorities* as follows:

Severity	Syslog priority
debug	LOG_DEBUG
info	LOG_INFO
notice	LOG_NOTICE
warn	LOG_WARNING
mark	LOG_ERR
err	LOG_ERR
crit	LOG_CRIT
alert	LOG_ALERT

Messages larger than 959 chars will be split into several messages. By default, samhain will use the identity 'samhain', the *syslog facility* LOG_AUTHPRIV, and will log its PID (process identification number) in addition to the message.

The syslog facility can be modified via the directive **SyslogFacility**= *LOG_xxx* in the *Misc* section of the configuration file.

The syslog priority to be used for heartbeat messages (timestamps) can be selected with the directive **SyslogMapStampTo**= *LOG_xxx* in the *Misc* section of the configuration file. The default is LOG_ERR.

12. SQL Database

Requirements

To build an executable with support for logging to a SQL database, the relevant development packages (e.g. libmysqlclient-dev on Ubuntu for MySQL) need to be installed in the build machine.

This facility requires that you have compiled with the *--enable-xml-log* option to format log messages in XML (*also for the client*, even if you do SQL logging on the server), and of course with the *--with-database=XXX* option (where 'XXX' may be any of: mysql, postgresql, oracle, or odbc).

If you are using the *MessageHeader* directive in the configuration file for a user-defined message header, make sure that the log messages are still valid XML, and that all the default entities are still present.

Currently MySQL, PostgreSQL, and Oracle are implemented and tested. Support for unixODBC is implemented, but not fully tested. If the header file 'mysql.h' ('libpq-fe.h') is not found during compilation ('mysql.h: No such file or directory'), you can use the option *--with-cflags=-I/dir/where/mysql.h/ is*. If the library libmysqlclient.a (libpq.a) is not found ('/usr/bin/ld: cannot find -lmysqlclient'), you can use the option *--with-libs=-L/dir/where/libmysqlclient.a/is*.

Note

The --enable-static option to compile a static executable is incompatible with PostgreSQL and MySQL.

By default, the database server is assumed to be on localhost, the db name is 'samhain', the db table is 'log', and inserting is possible for any user without password. To *create* the database/table with the required columns, the distribution includes the scripts 'samhain.mysql.init', 'samhain.postgres.init', and 'samhain.oracle.init'. E.g., for PostgreSQL you would setup the database like:

```
$ su postgres
$ createdb samhain
$ createuser -P samhain
Enter password for new role:
Enter it again:
$ psql -d samhain < samhain.postgres.init
$ exit
```

... and for MySQL:

```
$ mysql -p -u root < samhain.mysql.init
$ mysql -p -u root
> CREATE USER 'samhain'@'localhost' IDENTIFIED BY 'password';
> GRANT SELECT,INSERT ON samhain.log TO 'samhain'@'localhost';
> FLUSH PRIVILEGES;
```

Permissions

The PostgreSQL init script will grant INSERT permission only to a user 'samhain'. Please take note that for PostgreSQL, inserting also requires SELECT and UPDATE permission for the sequence 'log_log_index_seq' (see bottom of init script). The MySQL init script will create the database, but not the user, and will not grant any permissions.

As with all logging facilities, logging to the SQL database must be enabled in the configuration file by setting an appropriate threshold, e.g.:

```
[Log]
DatabaseSeverity=warn
```

In the *Database* section of the configuration file, you can modify the defaults via the following directives:

```
[Database]
SetDBName=db_name
SetDBTable=db_table
SetDBHost=db_host
SetDBUser=db_user
SetDBPassword=db_password
UsePersistent=yes/no
```

The default is to use a persistent connection to the database. You can change this with **UsePersistent=no**

Note re. PostgreSQL

For PostgreSQL, db_host must be a numerical IP address.

When logging client messages, yule will wrap them into a server <log sev="RCVT" tstamp=[...] > [...] </log> message. The parser will then create a separate database entry for this server timestamp. If you don't like this, you can use the option **SetDBServerTstamp= *false*** .

The table field 'log_ref' is NULL for client messages, 0 for server messages, and equal to 'log_index' of the client message for the aforementioned server timestamp of a client message.

Log records can be tagged via a special (indexed) table field 'log_hash', which is the MD5 checksum of (the concatenation of) any fields registered with **AddToDBHash= *field*** . The beltane web-based console can use these tags to filter messages. There is no default set of fields over which the MD5 hash is computed, so by default the tag is *equal* for all rows.

Tip

For security, you may want to set up a user/password for insertion into the db. However, as the password is in cleartext in the config file (and the connection to the db server is not encrypted), for remote logging this facility is less secure than samhain's own client/ server system (it is recommended to run the db server on the log host and have the log server, i.e. yule, log to the db).

12.1. Upgrade to samhain 2.3

Version 2.3 of Samhain supports checking of SELinux attributes and/or Posix ACLs. For backward compatibility, this is off by default. If you upgrade Samhain and enable this option, you need to update the database scheme as follows:

Mysql:

```
ALTER TABLE samhain.log ADD COLUMN acl_old BLOB;
```

```
ALTER TABLE samhain.log ADD COLUMN acl_new BLOB;
```

PostgreSQL:

```
ALTER TABLE samhain.log ADD COLUMN acl_old TEXT;
ALTER TABLE samhain.log ADD COLUMN acl_new TEXT;
```

Oracle:

```
ALTER TABLE samhain.log ADD acl_old VARCHAR2(4000);
ALTER TABLE samhain.log ADD acl_new VARCHAR2(4000);
DROP TRIGGER trigger_on_log;
```

12.2. Upgrade to samhain 2.4.4

Version 2.4.4 of Samhain supports storing the content of files. If you have created your Oracle database using the database scheme from a previous version, you need to change at least the 'link_old' and 'link_new' columns from VARCHAR2 to CLOB:

```
ALTER TABLE samhain.log ADD tmp_name CLOB;
UPDATE samhain.log SET tmp_name=link_old;
ALTER TABLE samhain.log DROP COLUMN link_old;
ALTER TABLE samhain.log RENAME COLUMN tmp_name to link_old;

ALTER TABLE samhain.log ADD tmp_name CLOB;
UPDATE samhain.log SET tmp_name=link_new;
ALTER TABLE samhain.log DROP COLUMN link_new;
ALTER TABLE samhain.log RENAME COLUMN tmp_name to link_new;
```

12.3. Upgrade to samhain 2.8.0+

As of version 2.8.0, Samhain supports IPv6 now, which means that the size of the 'ip' column in the database must be increased from VARCHAR(16) to VARCHAR(46).

Mysql:

```
ALTER TABLE samhain.log MODIFY ip VARCHAR(46);
```

PostgreSQL:

```
ALTER TABLE samhain.log ALTER COLUMN ip TYPE VARCHAR(46);
```

Oracle:

```
ALTER TABLE samhain.log MODIFY ip VARCHAR(46);
```

12.4. Upgrade to samhain 4.0

As of version 4.0, Samhain optionally logs the checking policy for a file if the option *ReportCheckflags* is set in the configuration file. If you enable this option, you need to update your database scheme as follows:

Mysql:

```
ALTER TABLE samhain.log ADD COLUMN checkflags_old BIGINT UNSIGNED;
ALTER TABLE samhain.log ADD COLUMN checkflags_new BIGINT UNSIGNED;
```

PostgreSQL:

```
ALTER TABLE samhain.log ADD COLUMN checkflags_old NUMERIC(20);
ALTER TABLE samhain.log ADD COLUMN checkflags_new NUMERIC(20);
```

Oracle:

```
ALTER TABLE samhain.log ADD checkflags_old NUMBER(20);
ALTER TABLE samhain.log ADD checkflags_new NUMBER(20);
```

12.5. MySQL configuration details

To pass the location of the MySQL Unix domain socket (for connections on localhost) to samhain, you can use the environment variable MYSQL_UNIX_PORT (the value must be the path of the socket).

Alternatively, as of samhain version 2.2, you can set options for the group "samhain" in `my.cnf`. See the MySQL manual for the proper syntax of the `my.cnf` file, as well as for possible options.

Note

It is not possible for an application (like e.g. samhain) to detect whether `my.cnf` is readable (because the application does not know where the file resides). Interesting errors may result...

Chapter 5. Configuring samhain, the host integrity monitor

The samhain file monitor checks the integrity of files by comparing them against a database of file signatures, and notify the user of inconsistencies. The level of logging is configurable, and several logging facilities are provided.

samhain can be used as a client that forwards messages to the server part (yule) of the samhain system, or as a standalone program (for single hosts).

samhain can be run as a background process (i.e. a daemon), or it can be started at regular intervals by cron.

> ### Tip
>
> It is recommended to run samhain as daemon, because
>
> - samhain can remember file changes, thus while running as a a daemon, it will not bother you with repetitive messages about the same problem, and
>
> - using cron opens up a security hole, because between consecutive invocations the executable could get modified or replaced by a rogue program.

1. Usage overview

To use samhain, the following steps must be followed:

1. The configuration file must be prepared (Section 4 (p. 38) , Section 1 (p. 19) , and Section 11 (p. 53) for details).

   - All *files and directories* that you want to monitor must be listed. Wildcard patterns are supported.

   - The *policies* for monitoring them (i.e. which modifications are allowed and which not) must be chosen.

   - Optionally, the *severity* of a policy violation can be selected.

   - The *logging facilities* must be chosen, and the *threshold level* of logging should be defined To activate a logging facility, its threshold level must be different from *none*.

   - Eventually, the *address* of the e-mail recepient and/or the *IP address* of the log server must be given.

2. The database must be initialized. If it already exists, it should be deleted (samhain will not over-write, but append), or *update* instead of *init* should be used:

   samhain -t `init/update`

3. Start samhain in *check* mode. Either select this mode in the configuration file, or use the command line option:

   samhain -t `check`

   To run samhain as a background process, use the command line option

   samhain -D -t `check`

2. Available checksum functions

A cryptographic hash function is a one-way function **H(*foo*)** such that it is easy to compute **H(*foo*)** from *foo*, but infeasible to compute *foo* from **H(*foo*)** , or to find *bar* such that **H(*bar*)** = **H(*foo*)** (which would allow to replace *foo* with *bar* without changing the hash function).

One common usage of a such a hash function is the computation of *checksums* of files, such that any modification of a file can be noticed, as its checksum will change.

For computing checksums of files, and also for some other purposes, samhain uses the TIGER hash function developed by Ross Anderson and Eli Biham. The output of this function is 192 bits long, and the function can be implemented efficiently on 32-bit and 64-bit machines. Technical details can be found at this page.

As of version 1.2.10, also the MD5 and SHA-1 hash functions are available. (You need to set the option *DigestAlgo=MD5* or *DigestAlgo=SHA1* in the config file to enable this). Note that MD5 is somewhat faster, but because of security concerns it is not recommended anymore for new applications.

As of version 3.1.0, also the SHA2-256 hash function is available. (You need to set the option *DigestAlgo=SHA256* in the config file to enable this).

3. File signatures

samhain works by generating a database of *file signatures*, and later comparing file against that database to recognize file modifications and/or added/deleted files.

File signatures include:

- a 192-bit cryptographic checksum computed using the TIGER hash algorithm (alternatively SHA-1, MD5, or SHA2-256 can be used),

- the inode of the file,

- the type of the file,

- owner and group,

- access permissions,

- on Linux only: flags of the ext2 file system (see **man chattr**),

- the timestamps of the file,

- the file size,

- the number of hard links,

- minor and major device number (devices only)

- and the name of the linked file (if the file is a symbolic link).

Depending on the policy chosen for a particular file, only a subset of these may be checked for modifications (see Section 4.1 (p. 39)), but usually all these informations are collected.

4. Defining file check policies: what, and how, to monitor

This section explains how to specify in the configuration file, which files or directories should be monitored, and which monitoring policy should be used.

4.1. Monitoring policies

samhain offers several pre-defined monitoring policies. Each of these policies has its own section in the configuration file. Placing a file in one of these sections will select the respective policy for that file.

The available policies (section headings) are:

ReadOnly	All modifications except access times will be reported for these files.
	Checked: owner, group, permissions, file type, device number, hardlinks, links, inode, checksum, size, mtime, ctime.
LogFiles	Modifications of timestamps, file size, and signature will be ignored.
	Checked: owner, group, permissions, file type, device number, hardlinks, links, inode.
GrowingLogFiles	Modifications of timestamps, and signature will be ignored. Modification of the file size will only be ignored if the file size has *increased*.
	Checked: owner, group, permissions, file type, device number, hardlinks, links, inode, size >= previous_size, checksum(file start up to previous size) equals previous checksum.
	If the size has shrunk (e.g. because of log rotation), samhain will look for a file with the same inode as before and check whether it has a size >= previous_size, and checksum(file start up to previous size) equals previous checksum. No report will be issued if this check succeeds. Thus log rotation will be handled gracefully as long as the inode is kept (i.e. the old file is moved rather than copied) and the first rotated file is not compressed (the logrotate tool can be told to compress only after the second cycle, and on Debian this seems to be standard anyway).
Attributes	Only modifications of ownership, access permissions, and device number will be checked.
	Checked: owner, group, permissions, file type, device number.
IgnoreAll	No modifications will be reported. However, the *existence* of the specified file or directory will still be checked.
IgnoreNone	All modifications, including access time, but excluding ctime, will be reported - checking atime *and* ctime would require to play with the system clock.
	Checked: owner, group, permissions, file type, device number, hardlinks, links, inode, checksum, size, mtime, atime.
User0	Initialized to: report all modifications.
User1	Initialized to: report all modifications.
User2	Initialized to: report all modifications.
User3	Initialized to: report all modifications.
User4	Initialized to: report all modifications.
Prelink	Modifications of timestamps, size, and inode will be ignored Checksums will be verified by calling **/usr/sbin/prelink --verify** . This policy is intended for verification of prelinked executables/libraries and/or directories containing such files. For details and further configuration options see Section 4.8 (p. 45) .

Checked: owner, group, permissions, file type, device number, hardlinks, links, checksum.

Note

Each policy can be modified in the config file section *Misc* with entries like **RedefReadOnly= *+XXX[,...]*** or **RedefReadOnly= *-XXX[,...]*** to add (+XXX) or remove (-XXX) a (a comma-separated list of) tests XXX, where XXX can be any of CHK (checksum), TXT (store file content in database), LNK (link), HLN (hardlink), INO (inode), USR (user), GRP (group), MTM (mtime), ATM (atime), CTM (ctime), SIZ (size), RDEV (device numbers), MOD (file mode), PRE (Linux; prelinked binary), SGROW (file size is allowed to grow), and/or AUDIT (Linux; report who changed the file)

This must come before any file policies are used in the config file.

4.2. File/directory specification

Entries for files have the following syntax:

file= */full/path/to/the/file*

Entries for directories have the following syntax:

dir= *[recursion depth]/full/path/to/the/directory*

The specification of a (numerical) recursion depth is optional (see Section 4.5 (p. 43)). (Do not put the recursion depth in brackets — they just indicate that this is an optional argument!).

Wildcard patterns ('*', '?', '[...]') as in shell globbing are supported for paths. The leading '/' is mandatory. Since version 2.7.1, it is allowed to enclose the value of the directive (i.e. the path for files, the optional recursion depth and the path for directories) within matching single or double quotes, which allows to have trailing blanks (*note: it is not neccessary to escape quotes in between - the algorithm does not scan forward to find the matching quote, rather it uses the last character*). Also since version 2.7.1, C quoting style is supported ('\a' [bell], '\b' [backspace], '\f' [form feed], '\n' [newline], '\r' [carriage return], '\t' [horizontal tab], '\v' [vertical tab], '\\' [backslash], '\'' [single quote], '\"' [double quote], '\nnn' [dree digit octal value], \xNN' [two digit hexadecimal value]). Example:

```
[ReadOnly]
# valid examples
dir = /u01/oracle/archive00
dir = 7/u01/oracle/archive02
dir = "7/u01/oracle/archive03  "
dir = "7/u01/oracle/archi"ve"
dir = /u01/oracle/archive\v04
dir = /u01/oracle/archive\\04
dir = /u01/oracle/archive\076
file = '/u01/oracle/archive\x0a'

# valid (no quote at start, thus quote at end
#        is considered part of filename)
file = /u01/oracle/archive_0"

#invalid (no matching quote at end)
file = "/u01/oracle/archive_0

#invalid (\03 is bad, must be 3 digits [octal]
#         or \x03 for hexadecimal)
```

```
file = /u01/oracle/archive_\03

#invalid (\g is undefined escape sequence)
file = /u01/oracle/archive_\g
```

Note on directories

A directory is (a) a collection of files, with (b) a directory special file where a listing of all files in the directories is kept. This directory special file will be modified in case of a file addition, removal, or renaming. Depending on the chosen policy, samhain will report on such modifications of the directory special file.

The addition and/or deletion of files from a directory modifies the directory special file (mtime/ctime). The addition/deletion of subdirectories will also modify the number of hardlinks of the directory special file. A modification of a file *may* modify a directory special file (mtime/ctime), if this modification is done by first creating a temporary file, followed by renaming this temporary file to the original one.

4.2.1. Rules

1. For the file check, samhain does not follow symlinks. If the argument for a file=... directive is a symlink, then the symlink itself is checked, not the location it points to.

2. The argument for a dir=... directive must be a *directory*. Using a symlink to a directory as argument is incorrect.

3. Precedence is given to the most specific location in the filesystem regardless of the order listed in the config file. I.e.,

   - a policy for a specific file overrides the policy for its directory

   - a policy for a subdirectory overrides the policy for its parent directory

   - if a directory or file path are explicitly listed twice in two different policy sections, Samhain will print a warning and honor only the first stanza processed. "First matching rule wins." Note however that it is perfectly ok to list a directory both as file=/path and dir=/path (see next rules).

4. Checking a directory with dir=... will check both the *content of the directory* as well as the *directory special file* itself, honoring a local and global recursion depth, giving local preference.

5. Using a directory as argument for both a file=... and a dir=... directive will have the effect that

   - the file=... directive will override the dir=... directive for the directory special file itself,

   - while the dir=... directive remains in effect for the directory content.

6. The presence of a **file=*/parent/subdir***, which is more specific of a path entry than that of the parent directory in another policy section with a "deeper" recursion depth as **dir=*N/parent*** will not prevent Samhain from descending into /parent/subdir and applying the higher level directory with the "deeper" recursion policy to the contents of /parent/subdir The more-specific rule will only apply to the directory special file and does not "truncate" the higher level policy in any way.

7. To determine if you config file syntax is working as expected, increase the verbosity of debugging when running samhain with "-t init" using "-p info" or even "-p debug".

Example 1: If you only want to check files in a directory, but not the directory inode itself, use:

```
[ReadOnly]
dir = /u01/oracle/archive00
[IgnoreAll]
file = /u01/oracle/archive00

# Note: /u01/oracle/archive00/archive01.dbf -> archive99.dbf *should* be
# mounted in the DB as a read-only tablespace and should never be
# changed, however, the DBA thinks he's God and does not need to consult
# with the Admin, so he may be adding new, deleting, or renaming the
# DBFs using SQLPlus without consulting with the admin, so tell me about
# changes to the files inside that we know about at Samhain INIT but
# such as when he adds a file.
```

Example 2: If you want to monitor a directory, but not the dynamic contents inside it:

```
[Attributes]
file = /var/spool/mqueue
file = /tmp
[IgnoreAll]
dir=-1/var/spool/mqueue
dir=-1/tmp
```

Example 3: If you want to monitor a directory special file, while ensuring no files within are removed but not the actual attributes of those files:

```
[Attributes]
file = /root
[IgnoreAll]
dir=0/root
```

Thanks to Brian A. Seklecki for his effort to clarify these rules and provide examples.

4.3. Suppress messages about new/deleted/modified files

If you want to suppress messages about the creation of certain files (e.g. rotated log files), you can use the options **IgnoreAdded=*/fullpath/with_some_regex_inside*** and/or **IgnoreMissing=*/fullpath/with_some_regex_inside*** (to be placed in the [Misc] section of the configuration files. If you want to add more regular expressions, you can use these options multiple times. Since transient files might get modified during their lifetime, there is also the option **IgnoreModified=*/fullpath/with_some_regex_inside***

Note

The argument to **IgnoreAdded**, **IgnoreMissing**, and **IgnoreModified** must be a regular expression that matches the full path. *In particular, it has to start with a forward slash..* To test your regex before putting in samhain, you do something like this:

```
# This regex matches all files added by logrotate
# (e.g: messages.1 or messages.2.gz, etc.)
cd /var/log
for file in *; do
```

```
          echo $file| \
          egrep "(cron|messages|rpmpkgs|secure|up2date|wmtp)\.[0-9](\.gz)
      done
```

Once it's work this way, you can add it to your samhainrc file, but don't forget to add the full path. e.g:

```
      IgnoreAdded = your_tested_regex
```

This tip has been provided by jim at aegis hyphen corp dot org.

Alternative

If a directory is added to [Attributes] as a file=/dir, then only the directory special file is monitored for permissions/ownership. The advantage is that additions/removals of files to that subdirectory can happen without recourse, but the integrity of that directory is defended. Assuming the administrator doesn't want to get granular level of detail.

Good for such directories as: `/var/mail /var/cron/tabs /var/tmp /tmp`

This tip has been provided by Brian A. Seklecki

4.4. Dynamic database update (modified/disappeared/new files)

samhain reads the file signature database at startup and creates an in-memory copy. This in-memory copy is then dynamically updated to reflect changes in the file system.

I.e. for each modified/disappeared/new file you will receive an alarm, then the in-memory copy of the file signature database is updated, and you will only receive another alarm for that file if it is modified again (or disappears/appears again).

Note that the on-disk file signature database is *not* updated (if you have signed it, the daemon could not do that anyway). However, as long as the machine is not rebooted, there should be no need to update the on-disk file signature database.

If files disappear after initialization, you will get an error message with the severity specified for file access errors (*except* if the file is placed under the *IgnoreAll* policy, in which case a message of *SeverityIgnoreAll*— see Section 1.1 (p. 19) — is generated).

If new files appear in a monitored directory after initialization, you will get an error message with the severity specified for that directory's file policy (*except* if the file is placed under the *IgnoreAll* policy, in which case a message of *SeverityIgnoreAll*— see Section 1.1 (p. 19) — is generated).

The special treatment of files under the *IgnoreAll* policy allows to handle cases where a file might be deleted and/or recreated by the system more or less frequently.

4.5. Recursion depth(s)

Directories can be monitored up to a maximum recursion depth of 99 (i.e. 99 levels of subdirectories. The recursion depth actually used is defined in the following order of priority:

1. The recursion depth specified for that individual directory (Section 4 (p. 38)). As a special case, for directories with the policy *IgnoreAll*, the recursion depth should be set to *0*, if you want to monitor (the existence of) the files within that directory, but to *-1*, if you do not want samhain to look into that directory.

2. The global default recursion depth specified in the configuration file. This is done in the configuration file section *Misc* with the entry **SetRecursionLevel=** `number`

3. The default recursion depth, which is zero.

4.6. Hardlink check

As of version 1.8.4, samhain will by default compare the number of hardlinks of a directory to the number of its subdirectories (including "." and ".."). Normally, these numbers should be equal. The idea here is that a (kernel) rootkit may hide a directory, but fail to "fix" the parent directory hardlink count (actually, I am not aware of any kernel rootkit that would care to fix the hardlink count of the parent directory). This is an experimental feature; if there are any problems, it can be disabled with the option **UseHardlinkCheck=** `no` in the [Misc] section of the configuration file.

Errors will be reported at the same severity as directory access errors option **SeverityDirs=** `severity` in section [EventSeverity]).

MacOS X

This feature is not supported on MacOS X (because the resource fork is implemented as an invisible directory, it modifies the parent directory hardlink count.)

4.6.1. Specify exceptions for the hardlink check

Some filesystems do not always follow the rule mentioned above (directory hardlink equals number of subdirectories). E.g. the root directory of reiserfs partitions generally seems to have two additional hardlinks. To account for such exceptions, you can specify exceptions with the option **HardlinkOffset=** `N: /path` in the [Misc] section of the configuration file. Here, N is the numerical offset (actual - expected hardlinks) for `/path`. For multiple exceptions, use this options multiple times (note that '/path N:/path2' would itself be a valid path, so using the option only once with multiple exceptions on the same line would be ambiguous).

Note

Please note that samhain will not check for an exception if the standard rule (offset = 0) is true for a directory. Thus it will not warn if a directory that once was exceptional is not anymore.

4.7. Check for weird filenames

Samhain checks for weird filenames (containing control/nonprintable characters, newlines or tabs) and warns about them at a severity level that is set with **SeverityNames=** `severity` in section [EventSeverity]. The rationale is: most of the time, such names are either the result of user errors, buggy scripts, or questionable activity.

If you want to add characters to the set of 'good' ones, you can do so with the option: **AddOKChars=** `N1, N2, ...` in the [Misc] section of the configuration file. Nn should be the unsigned byte value of the character(s) in hex (leading '0x': 0xNN), octal (leading zero: 0NNN), or decimal.

UTF-8 filenames

To specify that filenames are UTF-8 rather than ASCII, use **FileNamesAreUTF8=yes** . Samhain will check for invalid UTF-8 sequences, and for filenames ending with invisible characters.

Tip

This check will not be performed for files under the IgnoreAll policy. To completely disable this check, use **AddOKChars=all** .

4.8. Support for prelink

prelink is a tool available on modern Linux systems that can significantly reduce the startup time of applications. It does this by performing some of the work of the dynamic linker in advance. As this changes both executables and shared libraries, file integrity verification will fail unless prelink is supported, in particular as prelinking has to be redone if libraries are updated (so initializing the checksum database after prelinking may not be good enough).

The disadvantage is that prelinking modifies libraries and executables, and may need to be redone (potentially modifying all or many executables again) if a library is updated. This is a major problem for file integrity checkers.

Version 2.0 of samhain and later support prelink. To use this support, you need to place prelinked executables and libraries (or directories holding them) under the *[Prelink]* policy rather than under the (e.g.) *[ReadOnly]* policy. For all files under the *[Prelink]* policy, inode, size, and timestamps will be ignored (prelinking changes them). In addition, for ELF binaries under the *[Prelink]* policy, **/usr/sbin/prelink --verify** will be used to compute checksums (i.e. the checksum will be computed on the output of this command). For other files, checksums are computed as usual.

Speed

Obviously, invoking **prelink** results in a significant overhead, and slows down file integrity checking (tests indicate a factor of three - your mileage may vary).

Verification failures (zero checksum)

It seems that **prelink --verify** fails if the dependencies of a prelinked binary have changed. This results in a zero checksum, and can be fixed by re-prelinking the affected binary.

There are two configuration options in the *[Misc]* section that can are relevant for prelink support:

SetPrelinkPath= *fullpath* sets the path to the prelink executable. The default is `/usr/sbin/prelink`.

SetPrelinkChecksum= *checksum* sets the TIGER192 checksum for the prelink executable. You can compute this with **samhain -H /usr/sbin/prelink** (remove whitespace from the computed checksum). If the checksum is set, samhain will verify the prelink executable immediately before using it, otherwise prelink will be used without this special precaution.

4.9. SELinux attributes and Posix ACLs

Note for users of SQL database logging

You need to update the database scheme before using this feature, if you are upgrading from a version below 2.3.0. See Section 12.1 (p. 34) for details.

As of version 2.3, samhain supports checking and verifying of SELinux attributes and/or Posix ACLs, if the operating system supports these features. SELinux attributes are a Linux-specific feature, while Posix ACLs are supported by multiple operating systems.

These features will only get compiled if the required development environment is available on the host where samhain is compiled (e.g. on Debian Linux, packages libattr1-dev and libacl1-dev).

For backward compatibility, these features are disabled by default, even if they are compiled in. To enable them, use the configuration directives:

```
[Misc]
UseACLCheck = yes
```

```
UseSelinuxCheck = yes
```

4.10. Codes in messages about reported files

As of version 1.8.2, reports about modified files include a short code in the message field to describe which properties have been modified. The codes are: 'C' for 'checksum', 'L' for (soft) 'link', 'D' for 'device number', 'I' for 'inode', 'H' for (number of) 'hardlinks', 'M' for 'mode', 'U' for 'user' (owner), 'G' for 'group' (owner), 'T' for 'time' (any), and finally 'S' for 'size'.

As an example, 'C--I----TS' would indicate that a file has been replaced by one with different check-sum, inode, timestamp, and size, but (e.g.) same mode (type and access permissions) and same own-ership.

4.11. Loose directory checking

If files are added to, or removed from a directory, or modified by writing a temporary file and renam-ing it to the original, samhain will report the changed file as well as the changed directory inode. If you regard the report on the directory inode as redundant, you can suppress it with the option: **LooseDirCheck**= *true* in the [Misc] section of the configuration file. This will cause samhain to ignore modified directory inodes if nothing else but size and timestamps has changed.

4.12. Storing the full content of a file

This is discussed in Section 20 (p. 68) .

4.13. Who made changes to a file?

First of all, the UID of the user who changed or created a file is not stored in the file metadata, and hence *in general not available*. However, some operating systems may have non-standard security enhancements to log such information.

In particular, sufficiently recent versions of the Linux kernel provide an audit subsystem that can be used to gather such information if the required userspace tools are installed and the system is properly configured.

It should be noted that the Linux audit subsystem does NOT audit every file access by default. Rather, files are only audited if a *watch* is placed on them. What Samhain can do is making sure that watches are indeed placed on all files of interest for you (as defined in the Samhain configuration file), and collecting relevant information in case of an event.

Samhain supports the Linux audit system insofar as it can automatically mark files of interest for logging, and automatically collects and reports the log information after a change has been detected. This implies that you don't need to maintain two separate configurations (one for Samhain and one for the audit daemon).

4.13.1. Compiling

If you compile a Samhain standalone or client executable on a Linux system, support for the Linux audit system will be compiled in automatically if the required *development* packages for libaudit and libauparse are present (Debian: libaudit-dev, RedHat: audit-libs-devel).

To verify that the compiled executable has audit support, use **samhain --version**, which should list "optionally report auditd record of changed file".

Not supported with static compiling

This extension is *incompatible with static compiling*, i.e. it is not available if '--en-able-static' has been used for compiling.

4.13.2. Audit System Configuration

(1) If you want Samhain to report who changed a file, you need to have the Linux audit daemon (Debian: auditd, RedHat: audit) installed. You also need to have the libaudit and libauparse libraries installed.

The audit daemon must be running. You can check that with the command **auditctl -s** , which should show a non-zero PID for the audit daemon. If it says 'pid=0', you need to enable the audit daemon. First, make sure the daemon will autostart on boot: on RedHat/CentOS use **chkconfig auditd on** , on Debian/Ubuntu use **update-rc.d auditd defaults** . Second, start the daemon with **/etc/init.d/auditd start** .

(2) You are most likely interested in the *auid*, i.e. the audit UID which tracks the *login user*. This UID is only set correctly if you are using the pam_loginuid PAM module, and in a correct way even. Please read the man page for pam_loginuid carefully, and add the line

session required pam_loginuid.so

to the files `/etc/pam.d/gdm`, `/etc/pam.d/login`, `/etc/pam.d/atd`, `/etc/pam.d/cron`, `/etc/pam.d/sshd`.

Do NOT add this line to `/etc/pam.d/su` *or* `/etc/pam.d/sudo`*, as that will set auid=0 and hence erase the track of the login user.* If your system has a `/etc/pam.d/common-session` file, don't add it there if that file is included by the `/etc/pam.d/su` or `/etc/pam.d/sudo` file(s).

(3) The audit daemon enables the audit system in the kernel. Therefore all processes starting earlier than the audit daemon itself may get an auid=4294967295 (unknown). To avoid this problem, add **audit=1** to the kernel boot parameters.

(4) Because Samhain needs to fetch the audit information from the audit log, this log needs to be current. I.e. in the audit deamon configuration (`/etc/audit/auditd.conf` on Ubuntu), the parameter 'flush' must be set to DATA or SYNC, otherwise it might happen that the log entry has not been written yet when Samhain tries to fetch it.

4.13.3. Usage

In order to activate this feature for some particular file or directory, you have to add the AUDIT flag to the policy under which you place the file or directory (see Section 4.1 (p. 39)). Note that placing an audit rule on a directory will cause the whole file hierarchy under that directory to be audited. Also note that you cannot place an audit rule on the root directory itself. This is a limitation of the Linux audit system itself, not of Samhain. You have to place audit rules on the individual directories in the root directory.

```
[Misc]
RedefReadOnly = +AUDIT
#
[ReadOnly]
file = /etc/login.defs
dir = /bin
```

By default, Samhain will set the flags "wa" (write|attribute change) on which the watch will trigger. If you want to use other flags (e.g. "r" for read), you can do so with the **SetAuditdFlags=** *r|w|x|a* (i.e. any combination of these four characters, see also **man 8 auditctl**). Please note that order matters (as with Redef... directives), i.e. this will affect only files/directories specified afterwards in the config file.

The rules set by Samhain are flagged with *samhain*, i.e. you can check them with **auditctl -l -k samhain**, and delete them manually with **auditctl -D -k samhain**. See the man page of the **auditctl** command for further reference.

If you want to verify that the audit system works properly, you can e.g. use **ausearch -k samhain** to see all audit log entries generated by rules flagged with 'samhain', i.e. inserted by Samhain.

Persistance of audit rules

If Samhain runs in the foreground, it will not delete the inserted rules upon exit. This is to ensure that file changes are still audited as desired when Samhain is run only occasionally or at fixed intervals (e.g. as a cron job). If you want to delete the rules, use the command **auditctl -D -k samhain** .

4.14. Skip checksumming for particular files

Checksumming can put a high I/O load on a machine, and in some cases one might want to skip this for particular files. As of version 2.8.2, Samhain allows to specify certain conditions for which checksumming of a file should be skipped. These are:

match_prefix(string)	Skip checksumming if the full path of the file starts with the given string (e.g.: /home/someuser).
match_regex(regular_expression)	Skip checksumming if the full path of the file matches the given POSIX regular expression (e.g.: .*\.mpg$).
size_exceeds(bytes)	Skip checksumming if the filesize exceeds the given size (in bytes).
match_permission(octal_perm)	Skip checksumming if the file permissions *exactly* match the given one (as octal number, e.g. 0755 for rwxr-xr-x, or 4755 for rwsr-xr-x).
have_permission(octal_perm)	Skip checksumming if the file permissions *include* the given one (as octal number, e.g. 0100 for execute by owner).
match_filetype(filetype)	Skip checksumming if the file is of some particular type. See appendix for a list of supported file types.

Files that should *not* be checksummed are specified with **SkipChecksum=** *list of conditions* in the [Misc] section of the configuration file. The following rules apply:

1. To *negate* a condition, place an exclamation mark ('!') in front of it.

2. All conditions in the given list are *and* ed, i.e. checksumming for a file is skipped only if *all* conditions in the list are true. E.g. you can place a match_prefix(string) condition at the start of the list to avoid evaluation of the following condition(s) for files that should not be skipped.

3. If more than one **SkipChecksum=...** directives are given, then they are *or* ed, i.e. checksumming for a file is skipped if *one* of the directives matches.

4.14.1. User-defined file types

It is possible to add (at most 16) user-defined filetype descriptions to the compiled-in list. This can be done with the directive **FileType=** *description* where the format of *description* is 8 fields, separated by ':'.

The 8 fields are offset:type:length:G1:G2:G3:Name:Teststring, which describe:

1. An optional offset into the file, can be at most 3072-length. Counting starts at 0, thus '6' would mean the 7th byte of the file.

2. Type is 0 for a C string, 1 if binary, i.e. if the teststring contains NULL bytes,

3. Length should be 0 if the type is 0, otewise the length of the teststring if it is of type 1 (binary).

4. G1, G2, G2 give the filetype as GENERIC:MORE_SPECIFIC:EXACT, e.g. IMAGE:COM-PRESSED:JPG

5. The name field is currently unused and should hold a human-understandable description, e.g. 'Jpeg image'

6. The teststring is a string or byte pattern that is found at the given offset in files of this type. You can use quoted-printable (qp) encoding (which is often used for e-mail) for arbitrary binary patterns. A qp-encoded character (byte) consists of 3 characters: a "=" followed by the two-digit hexadecimal value if the byte. Please note that NULL bytes MUST be qp-encoded as '=00', and the equal sign ('=') MUST be qp-encoded as '=3D'.

A valid example would be **FileType=** *6:0:0:IMAGE:COMPRESSED:JPG:JFIF Jpeg:JFIF* (this one is already compiled in). This would recognize any file with the string 'JFIF' starting at the 7th byte (counting starts at '0') as a Jpeg image.

4.15. Graceful handling of log rotation

Growing log files should be placed under the [GrowingLogFiles] policy which (as of 3.0.11+) uses the following logic to handle log rotation:

If the size has shrunk (e.g. because of log rotation), samhain will look for a file with the same inode as before and check whether it has a size >= previous_size, and checksum(file start up to previous size) equal to the previous checksum. No report will be issued if this check succeeds. Thus log rotation will be handled gracefully as long as the inode is kept (i.e. the old file is moved rather than copied) and the first rotated file is not compressed (the logrotate tool can be told to compress only after the second cycle, and on Debian this seems to be standard anyway).

4.16. No such user/group

Samhain will report on files that are unowned (i.e. belong to a UID/GID that is not associated with an existing user or group. The rationale is that this is a security risk. An administrator (or administrative script) may get tricked into creating a user/group for that UID/GID, thus allowing the freshly created user/group to hijack the unowned files.

As of version 4.3.0, on Linux Samhain will ignore unowned files if the UID/GID is mapped in a user namespace via the /etc/subuid and /etc/subgid files respectively.

5. Excluding files and/or subdirectories (All except...)

To exclude individual files from a directory, place them under the policy *IgnoreAll*. Note that the *existence* of such files will still be checked (see next section).

To exclude subdirectories from a directory, place them under the policy *IgnoreAll* with an individual recursion depth of *-1*(see Section 4.5 (p. 43)).

Note

Changes in a directory may also modify the directory inode itself (i.e. the special file that holds the directory information). If you want to check all but a few files in a directory (say, /etc), and you expect some of the excluded files to get modified, you should use a setup like:

```
[ReadOnly]
#
```

```
                    dir=/etc
                    #
                    [Attributes]
                    #
                    # less restrictive policy for the directory file itself
                    #
                    file=/etc
                    #
                    [IgnoreAll]
                    #
                    # exclude these file and directories
                    #
                    file=/etc/resolv.conf.save
                    dir=-1/etc/calendar
                    #
```

6. Timing file checks

In the *Misc* section of the configuration file, you can set the interval (in seconds) between succesive file checks:

SetFilecheckTime= *value*

Alternatively, you can specify a crontab-like schedule with:

FileCheckScheduleOne= *schedule*

The schedule follows the same rules as crontab(5) entries, with two noteable exceptions: (a) *lists* are not allowed, and (b) *ranges* of names (like Mon-Fri) are allowed. See **man 5 crontab** for details. You can specify a list of schedules, with separate FileCheckScheduleOne=... directives on separate lines.

Note

If you need a list in your schedule, you can either use steps (like */2 for 'every two minutes/hours/...), or you can specify a list of schedules, with separate FileCheckScheduleOne=... directives on separate lines.

6.1. Using a second schedule

If you want to check some files rather often, while doing a more extensive check only sometimes, this is supported as follows:

• Enclose all directories for the more extensive check in a %SCHEDULE_TWO ... !%SCHEDULE_TWO block like:

```
    %SCHEDULE_TWO
    dir=/check/only/once/per/day
    !%SCHEDULE_TWO
```

• Define an optional second schedule as follows (similar to FileCheckSchedule, you can specify a list of schedules):

FileCheckScheduleTwo= *schedule2*

Rules:

1. All files and directories will always be checked at FileCheckScheduleTwo.

2. All single files (file=...) will always be checked at both FileCheckScheduleOne and FileCheck-
   ScheduleTwo (rationale: this is required to check for missing/added files in directories).

3. All directories outside the %SCHEDULE_TWO block will be checked at both FileCheckSched-
   uleOne and FileCheckScheduleTwo.

4. All directories inside the %SCHEDULE_TWO block will be checked at FileCheckScheduleTwo
   only.

Missing files

If you are using a second schedule, the full check for missing files will only be done
at FileCheckScheduleTwo. For paths directly defined in the configuration, e.g. with
file=... , samhain will detect immediately if the file is missing, if the path is checked at
FileCheckScheduleOne.

7. Initializing, updating, or checking

In the *Misc* section of the configuration file, you can choose between initializing the database, updating
it, or checking the files against the existing database:

ChecksumTest= `init|update|check|none`

If you use the mode *none*, you should specify on the command line one of *init*, *update*, or *check*, like:
samhain -t `check`

As of version 1.8.1, there is a new command line flag *--interactive* to enable interactive updates. If
you use this flag together with *-t update*, you will be asked if the database entry should be updated,
whenever samhain encounters a modified file.

As of version 2.8.6, updates can also be done using a text file listing paths (one per line) of files to
update, with the command line flag **--listfile**= `path_to_listfile`

8. The file signature database

The database file is named `samhain_file` by default, and placed into `/usr/local/var/lib/
samhain` by default (name and location can be configured at compile time).

The database is a binary file. For security reasons, it is recommended to store a backup copy of the
database on read-only media, otherwise you will not be able to recognize file modifications after its
deletion (by accident or by some malicious person).

samhain will compute the checksum of the database at startup and verify it at each access. samhain
will first open() the database, compute the checksum, rewind the file, and then read it. Thus it is not
possible to modify the file between checksumming and reading.

9. Checking the file system for SUID/SGID bi-
naries

To compile with support for this option, use the configure option

./configure --enable-suidcheck

If enabled, this will cause the samhain daemon to check the whole file system hierarchy for SUID/
SGID files at user-defined intervals, and to report on any that are not included in the file database. Upon
database initialization, all SUID/SGID files will automatically be included in the database. Excluded

are nfs, proc, msdos, vfat, and iso9660 (CD-ROM) file systems, as well as file systems mounted with the 'nosuid' options (the latter is not supported on all OSes, but at least on Linux).

On Linux, files that are marked as candidates for mandatory locking (group-id bit set, group-execute bit cleared) will be ignored.

You can manually exclude one directory (see below); this should be used only for obscure problems (e.g.: /net/localhost on Solaris - the automounter will mirror the root directory twice, as '/net/localhost' and '/net/localhost/net/localhost', and any nfs file system in '/' will be labelled as ufs system in '/net/localhost/net/localhost' ...).

Note

The SUID check is very I/O expensive. Using 'nice' may not help, if the CPU is waiting for I/O all the time anyway. To limit the load, the following options are provided:

You can *schedule* execution at fixed times with **SuidCheckSchedule=** *`schedule`* .

You can *limit I/O* with the **SuidCheckFps=** *`fps`* option (fps: files per second).

As an alternative to the **SuidCheckFps** option, you can use **SuidCheckYield=** *`yes`* . This will cause the SuidCheck module to yield its time slice after each file. If **Suid-CheckYield** is used, the **SuidCheckFps** option will not take effect.

The schedule should have the same syntax as a crontab entry (see crontab(5) and example below), with the following exceptions: (a) lists are not allowed, and (b) ranges of names are allowed. If a schedule is given, the **SuidCheckInterval** option will not take effect. You can specify a list of schedules with successive SuidCheckSchedule=... directives.

9.1. Quarantine SUID/SGID files

As of version 1.8.4, it is possible to *quarantine* new SUID/SGID files detected by samhain. To use this option, you must first enable it with **SuidCheckQuarantineFiles=** *`yes`* . This tells the SuidCheck module to quarantine any SUID/SGID files found after the initialization of the database using the method selected in **SuidCheckQuarantineMethod** (see next paragraph). If this is used, the file will be logged each time it is found and not added to the memory resident database.

You must also choose a method to be used to quarantine a SUID/SGID file: **SuidCheckQuarantineMethod=** *`0/1/2`* . Currently, there are 3 methods implemented: 0 - Delete the file from the system. 1 - Remove the SUID/SGID permissions from the file. 2 - Move the SUID/SGID file to a quarantine directory. The quarantine directory is `DEFAULT_DATAROOT/.quarantine`. Each file moved there has an additional file created that contains information about the SUID/SGID file. For example, if a file `/foo` is an unauthorized SUID/SGID file, then it will be removed and moved to `/var/lib/samhain/.quarantine` and another file, `foo.info`, will be created in `/var/lib/samhain/.quarantine` with information about `/foo`.

Important remarks

Methods 0 and 2 will by default not remove the original file, but rather truncate to zero size and remove suid/sgid properties. If you really want to remove the original file rather than truncate, you need to set the option **SuidCheckQuarantineDelete=** *`yes`*

The rationale for this behaviour is that removing a file in an arbitrary directory is considered to be *dangerous*, because the object that is unlinked may not be the same object anymore that has been determined to be a suid/sgid file before. You have been warned.

For additional security, samhain will recursively chdir into the parent directory of the file to make sure there are no symlinks in the path. Also, a file will not be truncated if it is a hardlink to another one.

No quarantining will be done if samhain is run in 'update' mode, since it is assumed that the current filesystem state is ok, and the database should be updated to reflect the current state.

9.2. Configuration

This facility is configured in the *SuidCheck* section of the configuration file.

```
[SuidCheck]
# activate (0 for switching off)
SuidCheckActive=1
# interval between checks (in seconds, default 7200)
# SuidCheckInterval=86400
# scheduled check at 01:30 each night
SuidCheckSchedule=30 1 * * *
# this is the severity (see Section 1.1 (p. 19))
SeveritySuidCheck=crit
# you may manually exclude one directory
SuidCheckExclude=/net/localhost
#
# limit on files per seconds
SuidCheckFps=250
# alternatively yield time slice after each file
# SuidCheckYield=yes
#
# Quarantine detected SUID/SGID files
# SuidCheckQuarantineFiles=no
#
# Quarantine Method
# 0 - Delete the file from the system.
# 1 - Remove the SUID/SGID permissions from the file.
# 2 - Move the SUID/SGID file to a quarantine directory.
#     The quarantine directory is DEFAULT_DATAROOT/.quarantine.
# SuidCheckQuarantineMethod = 1
#
# Really delete if using methods 0 or 2
# SuidCheckQuarantineDelete = no
```

10. Detecting Kernel rootkits

This option has been removed as of samhain 4.0 because it has been obsoleted by modern kernel developments.

11. Monitoring login/logout events

To compile with support for this option, use the configure option

./configure --enable-login-watch

samhain can be compiled to monitor login/logout events of system users. For initialization, the system utmp file is searched for users currently logged in. To recognize changes (i.e. logouts or logins), the system wtmp file is then used.

Optionally, it is possible to perform further checks for login events. All these additional checks are off by default. The following checks are provided:

First login

Report on the first login from a host or a domain / subnet. This option is configured with the directive:

LoginCheckFirst = `no`/`yes`/`domain`

If set to *yes*, samhain will issue a report when a user logs in from some host they haven't logged in from before. If set to *domain*, the domain (or C-class subnet, if the host cannot be resolved) is checked instead of the host.

Statistical outlier

Report unusual login times. This option will only take effect once a user has logged in several times, and a database of login times has been built which can be analyzed for statistical outlier detection. Since this is based on statistics, it will inevitably cause false positives (legitimate logins reported as outliers). This option is configured with the directive:

LoginCheckOutlier = `no`/`yes`/`paranoid`

If set to *yes*, samhain will issue a report when a login time is found to be an outlier with 99 per cent probability. If set to *paranoid*, the required outlier probability is lowered to 95 per cent, resulting in more reports and more false positives (legitimate logins reported as outliers).

Login date (global)

Report login events occuring outside some given date restrictions. This option is configured with the directive:

LoginCheckDate = `date`

Possible values for `date` are: *always*, *never*, and *workdays/saturday/sunday(list of time ranges)*, e.g. workdays(8:00-10:00,13:00-16:00) or saturday(08:10-17:20). To set date restriction for workdays (Mo-Fr) and saturday and/or sunday, use LoginCheckDate multiple times. The internal time resolution is ten minutes, i.e. 8:09 will be interpreted as 8:00.

Login date (individual)

Report login events occuring outside some date restrictions defined for the given individual user. This option, if defined for a given user, overrides the global setting above, and is configured with the directive:

LoginCheckUserDate = `user:date`

Here, `user` must be the login name for a user, and `date` has to be given as in the global option.

This facility is configured in the *Utmp* section of the configuration file:

```
[Utmp]
#
# activate (0 for switching off)
#
LoginCheckActive=1
#
# interval between checks (in seconds)
#
LoginCheckInterval=600
#
```

```
# these are the severities (see section Section 1.1 (p. 19))
#
SeverityLogin=info
SeverityLogout=info
#
# multiple logins by same user
#
SeverityLoginMulti=crit
```

12. Checking mounted filesystem policies

To compile with support for this option, use the configure option

./configure --enable-mounts-check

samhain can be compiled to check if certain filesystems are mounted, and if they are mounted with the appropriate options. This module currently supports Linux, Solaris, HP-UX (mount options as in /etc/mnttab), and FreeBSD. The configuration of the module is done in the *Mounts* section of the configuration file:

```
[Mounts]
#
# Activate (default is no).
#
MountCheckActive = yes
#
# Interval between checks (default = one day).
#
MountCheckInterval=7200
#
# Logging severities. We have two checks:
# 1) to see if a mount is there (default = err), and
# 2) to see if it is mounted with the correct options (default = err).
#
SeverityMountMissing=warn
SeverityOptionMissing=warn
#
# Mounts to check for, (optionally) followed by lists of
# mount options to check on them.
#
checkmount=/
checkmount=/var
checkmount=/usr
checkmount=/tmp noexec,nosuid,nodev
checkmount=/home noexec,nosuid,nodev
```

This module by the eircom.net Computer Incident Response Team.

13. Checking sensitive files owned by users

To compile with support for this option, use the configure option

./configure --enable-userfiles

samhain can be compiled to support checking of files that are specified as being relative to the a user's home directory. It is intended to detect interference with files that influence process behaviour such

as `.profile` It simply adds the appropriate file entries to the main samhain list, at the specified alerting level.

```
[UserFiles]
#
# Activate (0 is off).
#
UserfilesActive=1

#
# Files to check for under each $HOME
# A specific level can be specified.
# The allowed values are:
# allignore
# attributes
# logfiles
# loggrow
# noignore
# readonly
# user0
# user1
# user2
# user3
# user4
#
# The default is noignore
#
UserfilesName=.login noignore
UserfilesName=.profile readonly
UserfilesName=.ssh/authorized_keys
#
# A list of UIDs where we want to check.
# The default is all.
# IF THERE IS AN OPEN RANGE, IT MUST BE LAST
#
UserfilesCheckUids=0,100-500,1000-
```

This module by the eircom.net Computer Incident Response Team.

14. Checking for hidden/fake/missing processes

To compile with support for this option, use the configure option

./configure --enable-process-check

This module enables samhain to check for processes that are:

(a) hidden from ps, i.e. running processes that are not listed by ps,

(b) fake, i.e. listed by ps although they don't exist, and

(c) missing, i.e. processes that are required to run (as specified by the user), but are actually not running.

The module works by searching the complete range of possible PIDs for processes, and comparing the list of processes thus found against the output of ps. Note that the range of possible PIDs is OS-specific, and in general must be configured by the user (except for Linux, where it is determined automatically).

Threads

Threads (including kernel threads) may be detected as well; thus **ps** must be called with the proper argument such that threads are listed as well, otherwise they will be reported as hidden. On Linux, this is handled automatically by the code, for other operating systems, you can use the configuration option **ProcessCheckPSArg=** *arg* to set the argument to **ps** .

E.g. OpenBSD needs **ProcessCheckPSArg=** *axk* such that kernel threads are listed as well.

The ProcessCheckPSArg option

Samhain expects that the first numerical column is the PID. On Linux, the PID column needs to be followed by the SPID (thread id). I.e., on Linux **ProcessCheckPSArg=** *-efT* will work if none of the usernames is numeric.

OpenVZ

The OpenVZ virtualisation has one hidden process for each visible process (within the container). If you run samhain within an OpenVZ container, use **ProcessCheckIsOpen-VZ=** *true* to automatically avoid false positives.

14.1. Example configuration

```
[ProcessCheck]
#
# Activate (default is on)
#
ProcessCheckActive = yes

# The severity of reports: debug/info/notice/warn/err/crit/alert
# (default is crit)
#
SeverityProcessCheck = crit

# The PID range (default is 0 to 32767)
#
ProcessCheckMinPID = 0
ProcessCheckMaxPID = 32767

# The interval (in seconds) for process checks (default is 300 sec)
#
ProcessCheckInterval = 300

# Specify a process that is required to run. The argument
# must be a POSIX regular expression that matches the
# output of ps (samhain will check whether the PID in the
# output of 'ps' actually runs). You can use this option
# multiple times. Note that each matching substring in a line
# from the 'ps' output is considered a successful match.
#
ProcessCheckExists = syslogd

# The 'configure' script determines automatically
# the location of 'ps' as well as whether it is
# Posix or BSD style. Therefore, these options may
```

```
# not be required. For 'ProcesscheckPSArg', note
# that the first numerical column must be the PID, except on
# Linux, where the format 'PID SPID ...' is expected
# (spid = thread id), as shown by 'ps -eT' or 'ps -efT'.
#
# ProcessCheckPSPath = /usr/bin/ps
# ProcessCheckPSArg = -e
```

15. Checking for open ports

To compile with support for this option, use the configure option

./configure --enable-port-check

This module enables samhain to check for open ports (services) on the local machine, and report ports
that are open, but not listed in the configuration. Reports are like:

interface:portnumber/protocol (maybe_servicename)	This is a non-RPC service, e.g. 192.168.1.2:22/tcp (maybe_ssh). The service name is taken from /etc/ser-vices, and prepended by *maybe_*, because samhain cannot determine whether it really is the SSH daemon that is listening on this port.
interface:portnumber/protocol (servicename)	This is an RPC service, e.g. 192.168.1.2:2049/tcp (nfs). The service name is obtained by querying the portmapper daemon. The portmapper daemon may return a service name as listed in /etc/rpc, or just a number (if there is no name for the service). If the portmapper daemon only returns the number of the RPC service, samhain will list *RPC_number* as servicename.

15.1. Options

By default, (only) the interface corresponding to the 'official name' of the host will be scanned. Additional interfaces can be added via the option **PortCheckInterface=*(list of) IP address(es)***, where 'IP address' is the address of the interface that should be scanned. You can use this options multiple times to specify up to 15 additional interfaces, or supply a list of interfaces.

Don't specify external interfaces

While it is possible to misuse this option to specify an external IP address, the check will only work for interfaces on the local machine.

If addresses are dynamically assigned, you can use instead **PortCheckDevice=*(list of) de-vice(s)*** to specify device(s) to scan, regardless of address(es) assigned to them (each address counts towards the maximum of 63 interfaces for PortCheckInterface). Devices will be re-checked before scan to account for address changes.

Services (open ports) that are required or optional (allowed, but not required) can be specified with the options **PortCheckRequired=*interface:service list***, and/or **PortCheckOptional=*interface:service list***.

Services (open ports) that should be completely ignored can be specified with the option **PortCheck-Ignore=*interface:service list***.

Here, 'interface' should be the IP address of an interface, and 'service list' the comma-separated list of required/optional services. Each service must be listed as 'port/protocol' (e.g. 22/tcp) for a non-RPC service, and 'name/protocol' for an RPC service (e.g. portmapper/tcp). If an RPC service has no name, but just an RPC program number, then the name must be given as 'RPC_number' (e.g. RPC_100075).

Interface specification

The PortCheckRequired, PortCheckOptional, PortCheckIgnore options are considered whenever the port is checked on some specific interface, and thus the interface needs to match and is not optional.

By default, both TCP and UDP ports are scanned. To disable UDP scanning, the option **PortCheck-UDP=** *boolean* can be used.

Ports that should be skipped during the check can be specified with the option **PortCheckSkip=** *interface:port list*.

Here, 'interface' should be the IP address of an interface, and 'service list' the comma-separated list 'port/protocol' pairs (e.g.: 22/tcp,514/udp,...) to skip.

This option is different from **PortCheckIgnore=...** in three ways: (i) since it allows to skip ports only, it does not work for RPC services which have no fixed port, (ii) since the port is not probed, you can avoid error messages by obnoxious deamons, and (iii) it works without specifying the interface (equals to ALL).

MySQL (port 3306)

MySQL counts unsuccessful connection attempts and may refuse further connection if some limit is exceeded. You may want to use the **PortCheckSkip** option to avoid probing the MySQL port.

By default, all ports from 0 to 65535 are scanned. To change these limits, the options **PortCheckMin-Port=** *integer* and **PortCheckMaxPort=** *integer* can be used.

15.2. Example configuration

```
[PortCheck]
#
# Activate (default is on)
#
PortCheckActive = yes

# The severity of reports: debug/info/notice/warn/err/crit/alert
# (default is crit)
#
SeverityPortCheck = crit

# These are the defaults
#
PortCheckMinPort = 0
PortCheckMaxPort = 65535

# Services that are required. This example specifies ssl (22/tcp),
# smtp (25/tcp), http (80/tcp), and portmapper.
#
PortCheckRequired = 192.168.1.128:22/tcp,25/tcp,80/tcp,portmapper/tcp,portma

# Services that are optional. This example specifies
# mysql (3306/tcp).
#
PortCheckOptional = 192.168.1.128:3306/tcp

# Additional interfaces to scan. This example presumes that
```

```
    # the 'official hostname' corresponds to 192.168.1.128, and
    # that the machine has three more interfaces.
    # 127.0.0.1 (localhost) is not listed, hence not scanned.
    #
    PortCheckInterface = 192.168.1.129
    PortCheckInterface = 192.168.1.130
    PortCheckInterface = 192.168.1.131

    # The interval (in seconds) for port checks (default is 300 sec)
    #
    PortCheckInterval = 300

    # By default, UDP ports are checked as well as TCP ports.
    #
    PortCheckUDP = yes
```

16. Logfile monitoring/analysis

This option is available with samhain version 2.5.0 and higher. To compile with support for this option, use the configure option

./configure --enable-logfile-monitor

PCRE library required

This option requires the PCRE (Perl Compatible Regular Expressions) library. Many Linux distributions split library packages into a runtime package (required to run a dependent executable) and a development package (required to compile an executable). At least on the build host where samhain is compiled, the development package is required if you use this option.

This module enables samhain to monitor/analyze logfiles of other applications. Currently (samhain 2.5.0) the following logfile formats are supported:

• Syslog

• Apache (access and error log)

• Samba

• 'pacct' BSD-style process accounting (also available on Linux)

Logfile analysis will always start from the point the last one ended; the pointer into the file is stored persistently on disk. Logfile rotation is handled automatically as long as the rotated logfile remains in the same directory and is *not compressed*(usually log rotation tools can be configured to compress only after the second rotation, which is advisable for unrelated reasons - the logging application may still have an open file pointer after logfile rotation).

Logfile entries can be filtered with Perl-style regular expressions (filter rules). Regular expressions must match *the whole logfile record*. For efficiency, regular expressions can be grouped under a common regular expression, i.e. if the group expression fails to match, no RE in the group is tried. Furthermore, (groups of) regular expressions can be grouped by host, if the logfile(s) contain host information (such as host information in centralized syslog server logfiles, or virtual host information in Apache logfiles). Note that host->group->rule is supported (just as host->rule or group->rule), while group->host->rule isn't.

Each filtering rule (regular expression) is assigned to an output queue. Currently (samhain 2.5.0) queues only differ in the assigned severity of an event, but more options (per-queue mail addresses for alerts) are under development.

Filtering rules are processed in the order given in the configuration file, i.e. the first match wins.

> ### Blacklisting vs. whitelisting, and the 'trash' output queue
>
> Output queues are labelled. The label 'trash' is reserved and refers to the trash bin (no output, throw away log entries if the matching rule is assigned to the 'trash' queue).
>
> If a logfile entry does not match any rule, it is reported (i.e. the default is *whitelisting* known-good entries). To turn this into a *blacklisting* policy, simply add a catch-all rule at the end and assign it to the 'trash' queue.

16.1. Event Correlation

Sometimes it is desirable to report on the fact that several events happend at a similar time, possibly in a particular order. As of version 2.6.1, samhain supports this in the following way:

16.1.1. Marking individual events to be correlated

First, individual events to be correlated need to be marked for keeping them, under an arbitrary user-defined label, for an arbitrary user-defined time. So the rule for matching an event has to be modified like this:

LogmonRule=KEEP(`seconds,label` **):** `queue_label:(perl)regex` matches a logfile entry against the provided regular expression, AND keeps it for the specified time in *seconds*, with the specified *label*. In other words, processing of this rule will be no different than other rules, except for the fact that also a memory of the event is kept for the specified amount of time. So if you e.g. don't want a separate report for this individual event, just assign it to the *trash* queue.

16.1.2. Correlating the marked events

To correlate events labelled *label_one*, *label_two*, etc., just build a regular expression that matches the *labels*, in the temporal order you want to check for. E.g. if the temporal order is irrelevant, you may want to match *(label_one.*label_two)|(label_two.*label_one)*. Use this expression in a rule maked as CORRELATE(*description*), like this:

LogmonRule=CORRELATE(`description` **):** `queue_label:(perl)regex`

> ### Old records in existing logfiles
>
> Because the 'keep' timeout is relative to the current time, correlation of old entries in logfiles (i.e. when, at startup, an existing logfile with old entries is scanned) will only work if you specify 'keep' timeouts that are long enough to cover the whole timespan from the first logfile record until now.

16.2. Reporting non-occurence of an event

To check whether a given event occurs at least once within some given interval, the rule for matching an event can be modified like this:

LogmonRule=MARK(`seconds,description` **):** `queue_label:(perl)regex` matches a logfile entry against the provided regular expression, AND checks whether is occurs at least once within the specified interval (seconds).

Processing of this rule will be no different than other rules otherwise, so if you e.g. only want a report for this event if it is missing, just assign it to the *trash* queue. However, in the latter case the severity for reporting the messages must be set separately with the LogmonMarkSeverity directive, because the 'trash' queue has no severity assigned:

LogmonMarkSeverity= `severity` — Severity for reports on missing heartbeat messages if the messages themselves are assigned to the 'trash' queue (default: crit).

16.3. Reporting bursts of similar, repeated events

Samhain can automatically detect and report bursts of similar, repeated events in the monitored log-files. Here *similar, repeated events* refers to events that differ (only) in details that can be expected to differ for events of the same kind: IP adresses, FQDNs, email adresses, and numbers. The event history goes back 12 minutes, and thus a report is triggered if the number of similar events within the last 12 minutes exceeds a given threshold (default: 24).

This feature is *off* by default. In order to switch it on, you need to set a reporting queue:

LogmonBurstQueue= *queue* — Set the reporting queue for reporting bursts of similar log messages (default: don't report).

In addition, there are two more configurable parameters, one to set the triggering threshold (i.e. the number of messages within 12 minutes that need to be exceeded to raise an alert), and another one to indicate whether messages from the *cron* daemon should be considered as well (default: no):

LogmonBurstThreshold= *number* — The number of repeated messages within 12 minutes that must be exceeded to report a burst of repeated messages (default: 24).

LogmonBurstCron= *boolean* — Whether to report also on bursts of repeated cron messages (default: false).

16.4. Options

LogmonActive= *boolean* switches this module on or off (default: off).

LogmonSaveDir= */absolute/path* sets the directory where checkpoint data for logfiles is stored (default: same as for database file).

LogmonClean= *boolean* delete old checkpoint data unmodified for 30 days or more (default: off).

LogmonInterval= *seconds* sets the interval for logfile checking (default: 10 seconds).

LogmonMarkSeverity= *severity* — Severity for reports on missing heartbeat messages if the messages themselves are assigned to the 'trash' queue (default: crit).

LogmonBurstThreshold= *number* — The number of repeated messages within 12 minutes that must be exceeded to report a burst of repeated messages (default: 24).

LogmonBurstQueue= *queue* — Set the reporting queue for reporting bursts of similar log messages (default: don't report).

LogmonBurstCron= *boolean* — Whether to report also on bursts of repeated cron messages (defaul: false).

LogmonDeadtime= *seconds* — Do not report a correlated event again within the given time (default: 60 seconds).

LogmonWatch= *TYPE:path[:format]* advises the module to monitor the logfile with the specified `path`, which is of type 'TYPE' (logfile types are uppercase). Some logfile types (e.g. Apache access logs) can be customized, and hence some *format* information must be provided.

Do not quote the format

Please note that it's neither required nor supported to add quotes around the format string. Likewise, quotes within the format should not be escaped. Wrong:

LogmonWatch= *APACHE:/var/log/apache/access.log:"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\""*

Correct:

LogmonWatch= *APACHE:/var/log/apache/access.log:%h %l %u %t
"%r" %>s %b "%{Referer}i"*

Currently (samhain 2.6.4) the following logfile types are supported

SYSLOG   Standard UNIX style syslog files. Matching starts at the command (i.e. after the hostname). To select certain hostnames, place the rule under a LogmonHost directive (see below). If the LogmonHidePID option is used, the RE should not account for the process PID.

APACHE   Apache (or compatible) webserver access and/or error logs. Required *format* information: either one of *combined*, *common*, or *error*(error log), or the Apache custom log format specification used (also '%{X-Forwarded-For}i' is recognized). The whole log line is matched. If there are virtual hosts (%v), then the LogmonHost directive will match the virtual host.

In addition to the Apache format specifications, is possible to insert a *literal regular expression* as **RE{ *regex*}** (samhain 2.8.4+).

SAMBA   Samba logfile format (multiline, timestamp and origin within samba source code on first line, log message on continuation lines). The RE will match the continuation line (with the log message) only.

PACCT   BSD style process accounting (also available on Linux). This is a binary logfile. The module will build a text line like the 'last' command does, and match it against the RE.

What is pacct good for? Note that pacct records contain only the executable name, not the arguments. This may look somewhat useless for shell accounts, but is quite useful for servers: how many different commands can e.g. postfix legitimately execute? Just a handful, indeed, and certainly none of them is /bin/sh! So if pacct says that the 'postfix' user has executed a shell, then this would be rather alarming...

SHELL   A shell command. The full output on stdout will be read and matched. The PATH environment variable will be set to */sbin:/bin:/usr/sbin:/usr/bin:/usr/ucb*, and the SHELL, IFS, and TZ variables will be defined. The command is executed via **/ bin/sh -c *command*** .

LogmonHidePID= *boolean* is an option that only affects logfiles of type SYSLOG. It causes the PID to be stripped from the log line (before matching against the RE).

LogmonQueue= *label:[interval]:(sum|report):severity[:alias]* defines an output queue. Here, *label* is an arbitrary name which is used to assign rules to this queue; *interval* is the timespan over which messages are summarized if the queue is of type 'sum'; *sum*(summarize over some interval) or *report*(report each event separately and immediately) are the two queue type supported, and *severity* is the severity assigned to an event. Furthermore, optionally it is possible to specify an *alias*(must be defined in the email configuration) to direct email for this rule to a specific list of recipients.

Email

If you spefify a list alias, email will still go to all defined email recipients unless filtered, e.g. with

```
SetMailFilterNot = \[Logfile\]
```

I.e. you may want to define recipients, filter them as above, and then define list aliases to be used in an event queue. See Section 4 (p. 23) for more information.

LogmonHost= `(perl)regex` causes the following rules to be applied only to entries for this host(s). It is ended implicitly by another LogmonHost directive, or explicitly by a LogmonEndHost directive.

LogmonEndHost explicitely ends a preceding LogmonHost directive.

LogmonGroup= `group_label:(perl)regex` causes the following rules to be applied only if the group regex matches (i.e. rules within the group are skipped if the group regex doesn't match. This can be used to improve speed/efficiency of matching, i.e. you can group regexes by a common prefix. A group is ended implicitly by another LogmonGroup directive, or explicitly by a LogmonEndGroup directive.

LogmonEndGroup explicitely ends a preceding LogmonGroup directive.

LogmonRule= `queue_label:(perl)regex` matches a logfile entry against the provided regular expression. If the expression matches, then *captured* subexpressions are replaced by '___', and the logfile entry is reported as specified for the queue referenced by *queue_label*. Non-captured subexpressions (i.e. subexpressions where the opening bracket is followed by '?:') are *not replaced* by '___', but reported literally.

LogmonRule=KEEP(`seconds,label` **):** `queue_label:(perl)regex` as above, but additionally keep the event *label* for *seconds* to perform event correlation.

LogmonRule=CORRELATE(`description` **):** `queue_label:(perl)regex` perform event correlation by matching the *labels*(as specified in KEEP rules) of a sequence of events against the given regular expression.

LogmonRule=MARK(`seconds,description` **):** `queue_label:(perl)regex` matches a logfile entry against the provided regular expression, AND checks whether is occurs at least once within the specified interval (seconds).

16.5. Example configuration

```
[Logmon]

#
# Switch on the module
#
LogmonActive = yes

# Check every second
#
LogmonInterval = 1

# Strip PIDs from syslog messages
#
Logmonhidepid = true

# Define a queue with severity 'crit'.
# This is a 'report' queue, hence 'interval' (10)
# will be ignored.
#
LogmonQueue = q1:10:report:crit

# Define a second queue with severity 'alert'
#
LogmonQueue = q2:10:report:alert

# Monitor /var/log/messages, which is a syslog file
```

```
#
LogmonWatch = SYSLOG:/var/log/messages


# Monitor /var/log/samba/log.nmbd, which is a samba
# logfile
#
LogmonWatch = SAMBA:/var/log/samba/log.nmbd


# Monitor /var/log/apache2/access.log, which is
# an Apache logfile in 'combined' format
#
LogmonWatch = APACHE:/var/log/apache2/access.log:combined


# Monitor disks to check for full /dev/sda1
#
LogmonWatch = SHELL:df -h


# Syslog messages for the pppd deamon
#
LogmonGroup = g1:pppd.*
   #
   # Rules in this group
   #
   LogmonRule     = q1:pppd:\s+primary.*
   LogmonRule     = q1:pppd:\s+secondary.*
   #
LogmonEndGroup


# Warn about disk /dev/sda1 nearly full (80% or more. Use a
# non-capturing subexpression [the (?:8|9)] for the percentage full.
#
LogmonRule = q1:/dev/sda1\s+[0-9GM.]+\s+[0-9GM.]+\s+[0-9GM.]+\s+(?:8|9).%.*


# Messages starting with WARNING (some samba stuff)
#
LogmonGroup = g2:WARNING.*
LogmonRule     = q2:.*interfaces.*
LogmonEndGroup


# Report on these events if happening within 120 seconds.
# Set LogmonDeadtime to 120 seconds to avoid multiple reports.
# Use the 'trash' queue for the keep rules to avoid reports on
#    the individual events.
#
LogmonRule = KEEP(120,event1):trash:sshd: Accepted publickey for root.*
LogmonRule = KEEP(120,event2):trash:sshd: pam_unix\(sshd:session\).*
LogmonRule = CORRELATE(root_login):q1:(event1.*event2)|(event2.*event1)


LogmonDeadtime = 120


# Throw away all non-matching entries. This amounts
# to a blacklist policy (only report known bad).
#
# Usually considered bad practice!!! Use whitelisting!
#
# 'trash' is a built in queue, no definition needed.
#
LogmonRule = trash:.*
```

17. Checking the Windows registry

32bit vs. 64bit views

On 64bit Windows, the same key name may get mapped to different keys, depending on whether the lookup is done by a 32bit or 64bit application. Currently samhain does not check the alternate view.

This option is available with samhain version 2.8.0 and higher, when compiled on Cygwin/Windows. It enables samhain to verify the integrity of individual keys, or complete trees/hierarchies of keys, in the Windows registry.

Be careful what you ask for

The Windows registry is huge, i.e. it may contain a huge amount of keys, for which baseline data will get stored in the samhain baseline database if you desire to monitor all of them. There is the potential to blow up the size of the baseline database in a quite spectacular way.

17.1. Options

All options for this module go into the section **[Registry]** .

RegistryCheckActive= *boolean* switches this module on or off (default: off).

RegistryCheckInterval= *seconds* defines the interval (in seconds) between consecutive checks. The default is 300 seconds.

SeverityChange= *severity* defines the severity for reports on modifications to the registry.

IgnoreTimestampOnly= *boolean* to ignore changes where only the (write) timestamp has changed (default: off).

SingleKey= *key* defines a key to be monitored (of course it is possible to use this command multiple times). Valid key names must start with one of: HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, or HKEY_USERS. The Windows path separator ('\') must be used.

Hierarchy= *key* defines a key hierarchy in the registry, beginning at the specified key, to be monitored (of course it is possible to use this command multiple times). Valid key names must start with one of: HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, or HKEY_USERS. The Windows path separator ('\') must be used.

Escaping the path separator

The following two directives (StopAtKey, IgnoreKey) take a (POSIX) regular expression as argument. This implies that the path separator must be escaped by doubling it, i.e. you need to write '\\' instead of '\', because the '\' is a metacharacter in regular expressions (see example below).

StopAtKey= *regex* means that the check of a hierarchy will stop at the specified key, i.e. nothing below this key will be checked or monitored (but the key itself where the check stops will). *It is allowed to use a regular expression for the key.* Valid key names must start with one of: HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, or HKEY_USERS. The Windows path separator ('\') must be used.

IgnoreKey= `regex` differs from the *StopAtKey* option only insofar as the key where the check stops is *not* itself checked.

17.2. Example configuration

```
[Registry]

#
# Switch on the module
#
RegistryCheckActive = yes

# Check every 60 second
#
RegistryCheckInterval = 1

# Check this and everything below
#
Hierarchy = HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft

# Exclude this and anything below
# IgnoreKey and StopAtKey have a regex as argument, hence
# the path separator '\' must be escaped by doubling it.
#
IgnoreKey = HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVers

# Check this key
#
SingleKey = HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AFD\Paramet
```

18. Modules

samhain has a programming interface that allows to add modules written in C. Basically, for each module a structure of type *struct mod_type*, as defined in sh_modules.h, must be added to the list in sh_modules.c.

This structure contains pointers to initialization, timing, checking, and cleanup functions, as well as information for parsing the configuration file.

For details, in the source code distribution check the files sh_modules.h, sh_modules.c, as well as e.g. utmp.c, utmp.h, which implement a module to monitor login/logout events. There is also a HOWTO written by eircom.net Computer Incident Response Team.

19. Performance tuning

File checking is basically I/O-limited, i.e. typically most of the time the application waits for data from the disk. Most of the application runtime is spent in the checksum algorithm, but as the application is I/O-limited, using a faster algorithm does not neccessarily result in any noticable speed improvement.

Logging can be very expensive, so you should avoid enabling many different logging facilities. You should also avoid low logging thresholds (info/debug) on production systems — it tends to drown real problems in the noise of purely informational messages, and reduces performance quite noticably.

Other things you can do are:

- Build a static binary (use the **--enable-static** switch for configure). Static binaries are faster, and also more secure, because they cannot be subverted via libraries they depend on. On the other hand, if there is a bug in one of the statically compiled-in libraries, you would need to re-compile the executable after an update of the library.

Note

Some functionality may not be available with static binaries, in particular some name services (NIS, LDAP), the portmap service (used for the port check), and the support for the Linux audit daemon and SQL databases. On some operating systems (e.g. Solaris) static compiling may not be possible at all.

- Change the compiler switches to optimize more aggressively.

- If on a commercial UNIX, check whether the native compiler produces faster code (you need an ANSI C compiler). The **./configure** script honours CC (compiler) and CFLAGS environment variables.

On the other side, if you want to reduce the load caused by file checking, you can change the scheduling priority (see **man nice**), and/or limit the I/O:

```
[Misc]
# low priority (positive argument means lower priority)
SetNiceLevel=19
# kilobytes per second
SetIOLimit=1000
```

If you want to avoid thrashing the file cache, you can tell samhain to drop checksummed files from the cache (unless they were already cached). For performance reasons, this defaults to 'false'.

```
[Misc]
# drop checksummed files from cache
SetDropCache = True
```

Similarly, for the SUID check, you can limit the files per seconds:

```
[SuidCheck]
# limit on files per seconds
SuidCheckFps=250
```

20. Storing the full content of a file (aka: WHAT has changed?)

Consider using a revision control system

One of the most frequently requested features is the ability to determine *what* has changed in a file. This is not really within the scope of a *file integrity checker*; rather it would be the task of a *revision control system* like GIT, SVN (subversion) or CVS.

While samhain, as of version 2.4.4, supports storing the full content of files in the baseline database, this feature is limited to *small* files (smaller than 9200 bytes after zlib

compression). If you really think you need this feature, it is recommended to evaluate whether a *revision control system* does not fit your needs better.

As of version 2.4.4, samhain can optionally store the full literal content of regular files in the database, which allows to determine *what* has changed in a file. This feature will only get compiled if the required zlib development environment is available on the host where samhain is compiled (e.g. on Debian Linux, the package zlib1g-dev). This feature is subject to the following restrictions:

• Only small files can be stored, where 'small' means less than 9200 bytes after zlib compression (and less than 92000 bytes before compression, i.e. files 10 times larger than the limit are assumed to not compress below the limit).

• Only regular files can be stored; in particular, symlinks are not stored, since the *content* of a symlink inode actually is the target path (which is stored literally). It is safe to enable this for a directory, in the sense that it is silently ignored for file types where it does not apply.

• The feature must be explicitly enabled in the runtime configuration file by adding the '+TXT' to the monitoring policy of a file or directory.

To enable this feature, modify a policy to include 'TXT', and place the desired files under this policy (see example below).

In order to show the stored content of a file, use the following command:

> sh$ **samhain --list-file** *path* **-d** *database_path*

20.1. Example configuration

```
[Misc]
#
# UserN policies default to ReadOnly + ATM (access time). This
# makes the default (intentionally ;-) more or less useless.
#
# Redefine to ReadOnly + TXT (store file content)
#
RedefUser0 = -ATM, +TXT

[User0]
#
# Files for which we want to store the full content in the
# baseline database.
#
file=/etc/passwd
file=/etc/group
```

20.2. Implementation details

File contents are zlib compressed (RFC 1950), and the compressed data are base64 encoded. To avoid internal conflicts, samhain uses the letters '(', ')' and '?' instead of the letters '+', '/', and '=' used in standard base64 encoding. E.g. in PHP the following will decode the data:

```
$tmp1 = strtr($data, "()?", "+/=");
$tmp2 = base64_decode($tmp1);
```

```
$tmp3 = gzuncompress($tmp2);
```

21. Inotify support on Linux (instantaneous reports, no I/O load)

The Linux kernel offers an interface — called *inotify*— which allows an application to obtain change notifications for files and directories, if the application has registered *watches* for the files and directories it is interested in.

As of version 3.0, samhain optionally can use the *inotify* interface. The advantage of this is twofold: First, it is not neccessary to perform regular filesystem scans to detect file changes, i.e. the I/O load is drastically reduced. And second, changes can be reported immediately, thus allowing faster responses.

Please note

Your system may be configured with a too low default for the maximum inotify watches per user. You can see the default with:

cat /proc/sys/fs/inotify/max_user_watches

You can change the default temporarily (i.e. until reboot) with:

echo 1048576 /proc/sys/fs/inotify/max_user_watches

You can change the default permanently by placing the following line in /etc/ sysctl.conf(this will take effect at the next reboot):

fs.inotify.max_user_watches=1048576

Alternatively, samhain can be configured to reset the value by itself on startup (see example configuration below).

Full scans

(1) On startup, samhain will perform a full scan, first to set the inotify watches, and second to recover changes that happened after initialisation of the baseline database, but before starting the file check.

(2) The kernel will queue inotify events. If the queue overflows because there are too many events in too short a time, the application will be notified. In this case, samhain will automatically trigger a full scan to recover lost file system changes.

(3) Finally, even if inotify is enabled, samhain will still honour the configured intervals or schedules for full scans. If you want to rely only on inotify, you may want to configure a very large interval for filesystem checks, e.g. 'SetFilecheckTime = 315360000' (10 years).

Note that currently, *directories* specified via wildcard patterns can only be detected in a full scan if they appear newly in the filesystem (unless of course the parent directory is monitored anyway). Wildcard patterns for *files* are checked every 10 seconds.

21.1. Example configuration

```
[Inotify]
#
```

```
# Activate (default is off)
#
InotifyActive = yes

# Optionally set the maximum number of watches allowed per user.
# (default is leave as is). To see the default, use:
# cat /proc/sys/fs/inotify/max_user_watches
#
InotifyWatches = 1048576
```

Chapter 6. Configuring yule, the log server

yule is the log server within the samhain file integrity monitoring system. yule is part of the distribution package. It is only required if you intend to use the client/server capability of the samhain system for centralized logging to yule.

> ### Important
>
> Client and server are *distict* applications, and must be built separately. By default, installation names and paths (e.g. the configuration file) are different. Do not blame us if you abuse './configure' options to cause name clashes, if you install both on the same host.
>
> To compile yule, you must use **./configure --enable-network=server** . To compile a samhain client, you must use **./configure --enable-network=client** .

1. General

yule is a non-forking server. Instead of forking a new process for each incoming logging request, it multiplexes connections internally. Apart from samhain client reports (see below), yule (version 1.2.8+) can also collect syslog reports by listening on port 514/udp, if compiled with this option enabled (see also **man syslogd** .

Each potential client must be *registered* with yule to make a connection (see Section 1 (p. 37) and the example below). The client tells its host name to the server, and the server verifies it against the peer of the connecting socket. On the first connection made by a client, an authentication protocol is performed. This protocol provides *mutual authentication* of client and server, as well as a fresh *session key*.

By default, all messages are encrypted using Rijndael (selected as the Advanced Encryption Standard (AES) algorithm). The 192-bit key version of the algorithm is used. There is a compile-time option to switch off encryption, if your local lawmakers don't allow to use it (see Appendix).

yule keeps track of all clients and their session keys. As connections are dropped after successful completion of message delivery, there is no limit on the total number of clients. There is, however, a limit on the maximum number of *simultaneous* connections. This limit depends on the operating system, but may be of order 1000.

Session key expire after two hours. If its session key is expired, the client is forced to repeat the authentication protocol to set up a fresh session key.

Incoming messages are signed by the client. On receipt, yule will:

1. check the signature,

2. accept the message if the signature can be verified, otherwise discard it and issue an error message,

3. discard the clients signature,

4. log the message, and the client's hostname, to the console and the log file, and

5. add its own signature to the log file entry.

2. Important installation notes

As of version 1.7.0, yule will *always* drop root privileges after startup and initialization. You can use a privileged port (port number below 1024), because setting up the listening socket will occur as long as yule still has root privileges.

There are some special considerations that need to be taken into account when setting up an installation of yule. In particular:

The unprivileged user	By default, **configure** will check (in this order) for the existance of a user *yule*, *daemon*, or *nobody*, and use the first match.
	You can override this with the option **configure --enable-identity**= `user`. The user does not need to exist already; the install script knows how to create a new user (on Linux, FreeBSD, NetBSD, Solaris, HP-UX, OSF1).
	After successful installation, you will be asked to run **make install-user** in order to: (i) create the user that you specified to **configure** if it does not exist already (**make install-user** will check for this), and (ii) chown/chmod some directories.
	After running **make install** and **make install-user** , you should have a sane setup.
Logfile directory	The system logfile directory usually requires root privileges to write there (otherwise log files may easily get corrupted ...). To enable yule to write the log file and the HTML status file, a (sub-)directory should be used that is owned by yule. The **configure** script and the Makefile will do that automatically with the default layout (i.e. a directory `/var/log/yule` will be created).
Data files	The data file directory is now owned by root and world readable by default. If you chown it to a suitable *group* for the unprivileged yule user, you can make it group readable only. *Note that it is not required, and weakens the security, if the data file directory is writeable for the server.*
GnuPG signed configuration file	The unprivileged yule user must have a `.gnupg` subdirectory in its home directory, holding the public keyring with the key to verify the signature.
PID file	The PID file is written with before dropping root privileges. Therefore yule will not be able to overwrite it later (which is a GoodThing), or remove it upon exit (it will usually be able to recognize and handle a stale PID file on startup). Still, it may be a good idea to remove it after stopping yule. The provided start/stop scripts for various architectures will handle this.

3. Registering a client

Clients must be registered with yule to make a connection. *Connection attempts by unknown clients will be rejected.* The respective section in the server configuration file looks like:

```
[Clients]
#
# A client
#
Client=HOSTNAME_CLIENT1@salt1@verifier1
#
# another one
#
```

```
Client=HOSTNAME_CLIENT2@salt2@verifier2
#
```

These entries have to be computed in the following way:

1. Choose a *password*(16 chars hexadecimal, i.e. only 0 - 9, a - f, A - F allowed. To generate a random password, you may use:

   sh$ **yule --gen-password**

2. Use the program **samhain_setpwd** to reset the password in the compiled *client* binary (that is, samhain, not yule) to the one you have chosen. **samhain_setpwd** takes three arguments: (1) the binary name, (2) an extension to append to the new binary, and (3) the password. It will read the executable binary (argument 1), insert the password (argument 3), and write a modified binary with the specified extension (argument 2). Run **samhain_setpwd** without arguments for usage information. Example:

   sh$ **samhain_setpwd samhain EXT 0123456789ABCDEF**

3. Use the server's convenience function '-P' to create a registration entry. Example:

   sh$ **yule -P 0123456789ABCDEF**

4. The output will look like:

   Client=HOSTNAME@salt@verifier

   You now have to replace *HOSTNAME* with the fully qualified domain name of the host on which the client should run (*exception*: if the server cannot determine the fully qualified hostname, you may need to use the numerical address instead. You will see the problem in a 'Connection refused' message from the server).

5. Put the registration entry into the servers's configuration file, under the section heading *[Clients]*(see Section 3 (p. 73)). You need to send SIGHUP to the server for the new entry to take effect.

6. Repeat steps (1) -- (5) for any number of clients you need (actually, you need a registration entry for each client's host, but you don't neccesarily need different passwords for each client. I.e. you may skip steps (1) -- (3)).

If you have a default directory layout, a *[Clients]* section right at the end of the server config file, and your client is client.mydomain.com, then you could e.g. do:

```
bash$ PASSWD=`yule --gen-password`
bash$ samhain_setpwd samhain new $PASSWD
bash$ scp samhain.new root@client.mydomain.com:/usr/local/sbin/samhai
bash$ ENTRY=`yule -P $PASSWD | sed s%HOSTNAME%client.mydomain.com%`
bash$ echo $ENTRY >>/etc/yulerc
bash$ kill -HUP `cat /var/run/yule.pid`
```

4. Enabling logging to the server

If the client is properly registered with the server, all you need to do is to set an appropriate threshold for remote logging in the client's configuration file, and give the IP address of the server (if not already compiled in). Of course, the client must be compiled with the **--enable-network=client** switch.

Example for client configuration:

```
[Log]
#
# Threshold for forwarding to the log server
#
ExportSeverity=crit
[Misc]
SetLogServer=IP address
```

Example for server configuration:

```
[Clients]
#
# Register a client to allow it to connect
#
Client=client.mydomain.com@salt@verifier
```

5. Enabling baseline database / configuration file download from the server

A significant advantage of samhain is the option to store baseline databases and configuration files on the central log server (yule), from where they can be downloaded by clients upons startup. In order to use this option, clients must be configured to retrieve these files from the server rather than from the local filesystem.

Tip

Obviously, retrieving the configuration file from the log server requires that the IP address of the log server is *compiled in*, using the option **./configure --with-logserver=HOST** .

Downloaded files are written to a temporary file that is created in the home directory of the effective user (usually *root*. The filename is chosen at random, the file is opened for writing after checking that it does not exist already, and immediately thereafter unlinked. Thus the name of the file will be deleted from the filesystem, but the file itself will remain in existence until the file descriptor referring it is closed (see **man unlink**), or the process exits (on exit, all open file descriptors belonging to the process are closed).

5.1. Configuration file

If the compiled-in path to the configuration file begins with the special value ``REQ_FROM_SERVER'', the *client* will request to download the configuration file from yule (i.e. from the server).

If ``REQ_FROM_SERVER'' is followed by a path, the *client* will use the path following ``REQ_FROM_SERVER'' as a fallback if (*and only if*) it is initializing the database. This is a convenience feature to allow initializing the database(s) before the client is registered with the server.

Example: **./configure --with-config-file=REQ_FROM_SERVER/etc/conf.samhain** In this case, the client will request to download the configuration file from the server. If the connection to the server fails, it will exit on error if run in 'check' mode, but fallback to `/etc/conf.samhain` as its configuration file, if run in 'init' mode.

Note

For obvious security reasons, the client cannot specify the path to the configuration file on the server side. The server will lookup the configuration file using only the hostname of the client and the compiled-in path for the 'localstatedir' (see below). The default for 'localstatedir' is `/var`.

The server will search for the configuration file to send in the following order of priority (paths are explained in Section 6 (p. 112)). *CLIENTNAME* is the hostname of the client's host, as listed in the server's config file in the **Clients** section:

1. `localstatedir/lib/yule/rc.CLIENTNAME`

2. `localstatedir/lib/yule/rc`

5.2. Database file

If the compiled-in path to the database file begins with the special value ``REQ_FROM_SERVER'', the *client* will request to download the database file from yule (i.e. from the server).

CAVEAT

``REQ_FROM_SERVER'' *must* be followed by a path that will be used for writing the database file when *initializing*. Upon initialization, the database is always written to a local file, and must be copied with **scp** to the server (the client cannot *upload* the database file to the server, as this would open a security hole).

Example: **--with-data-file=REQ_FROM_SERVER/var/lib/samhain/data.samhain** In this case, the client will request to download the database file from the server if *checking*, and will create a local database file `/var/lib/samhain/data.samhain` if *initializing*. You have to use **scp** to copy the file signature database to the server then.

Note

For obvious security reasons, the client cannot specify the path to the database file on the server side. The server will lookup the databse file using only the hostname of the client and the compiled-in path for the 'localstatedir' (see below). The default for 'localstatedir' is `/var`.

The server will search for the database file to send in the following order of priority (see Section 6 (p. 112)). *CLIENTNAME* is the hostname of the client's host, as listed in the server's config file in the **Clients** section:

1. `localstatedir/lib/yule/file.CLIENTNAME`

2. `localstatedir/lib/yule/file`

6. Rules for logging of client messages

As the log server may receive quite a large number of log messages from clients (depending on the number of clients and their threshold settings), client messages are treated specially and by default are only logged to facilities suitable for bulk logging: console, log file, relational database (if enabled), and external (if enabled).

To override this behavior, you can set the option **UseClientSeverity=** *yes* in the *[Misc]* section of the configuration file. In that case, the client message severity is used, and client messages are treated just like local messages (i.e. like those from the server itself).

If you also want to filter by message class, there is also an option **UseClientClass=** *yes*

All client messages are recorded in the main log file by default. However, it is possible to use separate log files for individual clients. This can be enabled with **UseSeparateLogs=** *yes/no* in the *Misc* section of the server configuration file. No locking will be performed for such separate client log files (only one instance of the server can listen on the TCP port, thus there will be no concurrent access).

7. Detecting 'dead' clients

It is possible to set a time limit for the maximum time between two consecutive messages of a client (option **SetClientTimeLimit** in the [Misc] section of the configuration file). If the time limit is exceeded without a message from the client, the server will issue a warning. The default is 86400 seconds (one day); specifying a value of 0 will switch off this option.

You may want to set **ExportSeverity = mark** (or any lower threshold) in the client configuration file in order to log timestamp ('heartbeat') messages to the server.

8. The HTML server status page

yule writes the current status to a HTML file. The default name of this file is `samhain.html`, and by default it is placed in `/var/log`.

The file contains a header with the current status of the server (starting time, current time, open connections, total connections since start), and a table that lists the status of all registered clients.

There are a number of pre-defined events that may occur for a client:

Inactive	The client has not connected since server startup.
Started	The client has started. This message may be missing if the client was already running at server startup.
Exited	The client has exited.
Message	The client has sent a message.
File transfer	The client has fetched a file from the server.
ILLEGAL	Startup without prior exit. May indicate a preceding abnormal termination.
PANIC	The client has encountered a fatal error condition.
FAILED	An unsuccessful attempt to set up a session key or transfer a message.
POLICY	The client has discovered a policy violation.
TIME_EXCEED-ED	No message (e.g. timestamp) has been received from the client for a defined amount of time (default 1 day, option SetClientTimeLimit).

For each client, the latest event of each given type is listed. Events are sorted by time. Events that have not occurred (yet) are not listed.

It is possible to specify templates for (i) the file header, (ii) a single table entry, and (iii) the file end. Templates must be named `head.html`, `entry.html`, and `foot.html`, respectively, and must be located in the data directory (i.e. `localstatedir/lib/yule/`, see Section 6 (p. 112)). The distribution package includes two sample files `head.html` and `foot.html`.

The following replacements will be made in the head template:

Placeholder	Significance
%T	Current time.
%S	Startup time.
%L	Time of last connection.
%O	Open connections.
%A	Total connections since startup.
%M	Maximum simultaneous connections.

The following replacements will be made in the entry template:

Placeholder	Significance
%H	Host name.
%S	Event.
%T	Time of event.

Tip

A literal '%' in the HTML output must be represented by a '% ' ('%' followed by space) in the template.

9. Chroot

As of version 1.7.0, yule is able to chroot itself after startup and initialization, either by using the command line option

```
bash$ yule --chroot=/chrootdir
```

or by requesting it in the configuration file:

```
[Misc]
SetChrootDir=path
```

In order to prepare for the chroot jail, the following is required:

Tip

In the scripts subdirectory of the source directory there is a script chroot.sh to perform steps (4) and (5) (only for Linux).

1. Compile normally. Make sure you use either dev/random(default if existing) or EGD (Entropy Gathering Daemon) for the entropy device. If dev/random does not exist, the default is the 'standard unix entropy gatherer', which uses the output of many system commands, and therefore is not suitable within a chroot jail.

2. Install with the command(s):

```
bash$ make DESTDIR=/chrootdir install
```

```
bash$ make DESTDIR=/chrootdir install-user
bash$ make install-boot
```

3. Fix the path to the yule binary in the runlevel start/stop script installed by the last command.

4. Prepare the chroot environment. Basically, you need under `/chrootdir`

    (a) an entropy device, either `dev/random`, `dev/urandom`, or an EGD (Entropy Gathering Daemon) socket,

    (b) minimum `etc/passwd`, `etc/group` files, at least with entries for root and the unprivileged yule user. Replace passwords with an asterix, and make sure the homedirectory of the unprivileged yule user is correct within the chroot jail.

    (c) files required for DNS: `etc/nsswitch.conf`, `etc/hosts`, `etc/host.conf`, `etc/resolv.conf`, `etc/services`, `etc/protocols`.

5. Create a symlink `/etc/yulerc` to `/chrootdir/etc/yulerc` (no, it will not work the other way round).

Because yule chroots after startup, there is no need to copy shared libraries into the chroot jail. They will be loaded upon startup, before the chroot() occurs.

Tip

If you are using syslog logging, you need a `dev/log` socket in the chroot jail. Modern syslog incarnations will allow you to have an additional socket using the command:

```
bash$ syslogd -a /chrootdir/dev/log
```

Tip

If you are using a GnuPG- or signify-signed configuration, you will need a working copy of gpg or signify, respectively, in the chroot jail.

10. Restrict access with libwrap (tcp wrappers)

As of version 1.8.0, yule can be build with support for libwrap, i.e. Wietse Venema's tcp wrappers libraries. To enable this, use the **configure** option *--with-libwrap*.

You can then restrict access to yule with appropriate entries in the `/etc/hosts.allow` and/or `/etc/hosts.deny` files.

Note

If you use the **configure** option *--enable-install-name=NAME*, then yule will be installed as 'NAME', and this is what you then need to use as the daemons name in the `/etc/hosts.allow` and/or `/etc/hosts.deny` files.

11. Sending commands to clients

It is generally not possible to send commands to clients, because the client does not listen on the network (the client needs root privileges to perform its tasks, and you don't want a root network daemon).

However, it is possible to send a command if and when a client connects to deliver a message. As of version 1.8.0, clients use a new version of the client/server protocol, which includes a set of pre-defined commands that are understood by the client. Currently implemented are *RELOAD* to reload the configuration, *SCAN* to request a file system check (ouside the regular schedule), and *STOP* to terminate the client.

Pre-1.8.0 clients, or clients build with the (optional) old protocol version, will simply ignore such commands.

11.1. Communicating with the server

As of version 1.8.0, yule can send a command to a client if and when a client connects to deliver a message, e.g. a timestamp message (clients are not listening on the network, and thus commands can only be sent together with the confirmation when a message is received).

Of course the server needs to know which (if any) command to send. Therefore it can open a unix domain socket upon startup (in the same directory as the PID file). Opening this command interface must be requested explicitly with the option **SetUseSocket=** *yes* (in the [Misc] section).

A separate application yulectl is compiled together with the server that provides a command-line interface to access this facility. Use **yulectl -h** for help.

11.2. Authenticating to the server

There are two methods to authenticate to the server. If supported by the OS, authentication is done by passing the credentials of the socket peer to the server (this is a special feature of unix domain sockets), and *requiring the UID* of the the socket peer (i.e. the user using the yulectl program) to match a UID as set with the **SetSocketAllowUid=** *UID* option (default is 0, i.e. only root can use the interface).

Note

If passing credentials over the socket is supported by the OS, it is not possible to fake these credentials - they are supplied by the kernel. Therefore, the server can rely on the fact that the user process writing to the socket has indeed the UID passed via the socket. Thus, the access rights to the socket are basically not important (on some systems, they are not even recognized/respected at all).

As of version 1.8.12, if (and only if) passing credentials over the socket is not supported, you can specify a password with the **SetSocketPassword=** *password* option. The password must be 14 characters or less, and must not include the '@' character.

Of course you must supply the password to yulectl if you want to communicate with the server. To do so, create a file `.yulectl_cred` in your home directory, and place the password there.

Note

Password authentication is not supported if the OS supports the aforementioned method. You can use **yule -v** to find out which of the two authentication methods is supported.

12. Syslog logging

yule (version 1.2.8+) can listen on port 514/udp to collect reports from syslog clients. This must be enabled by using the **--enable-udp** configure option when compiling. In addition, in the *Misc* section of the configuration file, you must set the option **SetUDPActive=** *yes* .

This option requires to run yule either as *root*, or as *SUID* root. For security, yule will drop root privileges irrevocably immediately after binding to port 514/udp. It will assume the credentials of some compiled-in user. The default is 'yule', 'daemon', or 'nobody' (i.e. the first of these that exists on

your system). You can override this with the **--enable-identity=** *USER* option. Note that each daemon should have its own user/group, such that an exploit will not give write access to files owned by other daemons.

13. Server-to-server relay

As of version 2.2.0, it is possible to relay messages from one yule server to another. This is implemented in the same way as client-to-server connectivity, i.e. the relaying server and the endpoint server must be set up in the same way as a samhain client and a server, respectively (see Section 3 (p. 73)).

14. Performance tuning

If connections time out because of slow network, you can raise the timeout with **SetConnectionTimeout=** *seconds* (the default is 900 seconds).

Even without tweaking, the server can probably handle some 100 connections per second on a 500Mhz i686. Depending on the verbosity of the logging that you wish, this should suffice even for some thousand clients.

Almost all time is spent (i) in the HMAC function that computes the message signatures, and (ii) if you do not have the gmp (GNU MP) multiple precision library, in the multiple precision arithmetic library (for SRP authentication).

The reason for (ii) is that samhain / yule will use a simple, portable, but not very efficient MP library that is included in the source code, if gmp is not present on your system.

To improve performance, you can:

* install gmp, remove the file config.cache in the source directory (if you have run **configure** before), and then run **configure** and **make** again. The **configure** script should automatically detect the gmp library and link against it.

* use a simple keyed hash (HASH-TIGER), which will compute signatures as HASH(message key) instead of the HMAC (HMAC-TIGER). This will save two of the three hash computations required for a HMAC signature.

> ### CAVEAT
>
> Make sure you use the same signature type on server and client !

```
[Misc]
#
# use simple keyed hash for message signatures
# Make sure you set this both for client and server
#
        MACType=HASH-TIGER
```

* build a static binary (use the **--enable-static** switch for configure). Static binaries are faster, and also more secure, because they cannot be subverted via libraries they depend upon. (However, if there is a bug in one of the compiled-in libraries, you'd need to re-compile a static executable after an update of the library).

> ### Note
>
> Some functionality may not be available with static binaries, in particular some name services (NIS, LDAP), the portmap service (used for the port check), and the support

for the Linux audit daemon and SQL databases. On some operating systems (e.g. Solaris) static compiling may not be possible at all.

- change the compiler switches to optimize more aggressively.

- if on a commercial UNIX, check whether the native compiler produces faster code than gcc (you need an ANSI C compiler). The **configure** script honours CC (compiler) and CFLAGS environment variables.

Chapter 7. Hooks for External Programs

samhain provides several hooks for external programs for (re-)processing the audit trail, including pipes, a System V message queue, and the option to call external programs.

1. Pipes

It is possible to use named pipes as 'console' device(s) (samhain supports up to two console devices, both of which may be named pipes. You can set the device path at compile time (see Section 6 (p. 112)), and/or in the configuration file (see Section 8 (p. 29)).

Pipe full

(Since version 4.0) samhain will do a nonblocking write. If the pipe is full, the write will silently fail and the message will be lost.

2. System V message queue

It is possible to have a SystemV IPC message queue (which is definitely more elegant than named pipes) as additional 'console' device. You need to compile with **--enable-message-queue=*MODE*** and use the option **MessageQueueActive=*T/F*** .

The default mode is 0700 (rwx------), but this is a compile option (message queues are kernel-resident, but have access permissions like files).

In your code, to get the System V IPC key for the message queue, use **ftok("*localstatedir*/lib/ samhain", '#');** (since Samhain Version 4.0) or **ftok("/tmp", '#');** (previous versions). Here, *localstatedir*/lib/samhain is the directory where samhain writes the database upon initialisation, i.e. `/var/lib/samhain` by default.

Messages logged / Queue full

The message queue is treated as an additional 'console' device, i.e. the same messages go there that go to the console.

(Since version 4.0) If the queue is full, the oldest message(s) will get deleted until there is enough free space for the current message. I.e. the queue will behave like a ring buffer.

3. Calling external programs

samhain may invoke external programs or scripts in order to implement logging capabilities that are not supported by samhain itself (e.g. pager support). This section provides an overview of this capability.

External programs/scripts invoked for logging will receive the formatted log message on *stdin*. The program should expect that *stdout* and *stderr* are closed, and that the working directory is the root directory.

Each external program must be defined in the configuration file, in a section starting with the header **[External]** . In addition, **ExternalSeverity** must be set to an appropriate threshold in the section **[Log]** .

Each program definition starts with the line

OpenCommand=*/full/path*

Options for the program may follow. The definition of an external program is ended (a) when explicitly terminated with the line **CloseCommand** , (b) when the section ends, or (c) when another **OpenCommand**= `/full/path` line for the next command is encountered.

Environment variables

By default, the environment is limited to the TZ (timezone) variable. If you need other variables (e.g. LD_LIBRARY_PATH), you can set them using the **Setenviron**= `KEY=value` option (see below).

- There are several places in samhain where external programs may be called. Each such place is identified by a *type*. Currently, valid types are:

  - *log*— An external logging facility, which is handled like other logging facilities. The program will receive the logged message on stdin, followed by a newline, followed by the string *[EOF]* and another newline.

  - *srv*— Executed by the server, whenever the status of a client, as displayed in the HTML status table, has changed. The program will receive the client hostname, the timestamp, and the new status, followed by a newline, followed by *[EOF]* and another newline.

- Any number of external programs may be defined in the configuration file. Each external program has a *type*, which is *log* by default. Whenever external programs are called, all programs of the appropriate *type* are executed. The *type* can be set with **SetType**= `type`

- External programs must be on a trusted path (see Section 10.1 (p. 9)), i.e. must not be writeable by untrusted users.

- For enhanced security, the (192-bit TIGER) checksum of the external program/script may be specified in the configuration file: **SetChecksum**= `checksum` (one string, no blanks in checksum)

- Command line arguments and environment variables for each external program are configurable (the default is no command line arguments, and a clean environment containing only the TZ (timezone) variable:

  SetCommandline= `full_command_line` (full command line starting with the name of the program)

  Setenviron= `KEY=value`

- The user whose credentials shall be used, can be specified: **SetCredentials**= `username`

- Some filters are available to make the execution of an external program dependent on the message content:

  SetFilterNot= `list` If any regular expression in 'list' matches the message, the program is not executed, else

  SetFilterAnd= `list` if any regular expression in 'list' is not matching the message, the program is not executed, else

  SetFilterOr= `list` if none of the regular expressions in 'list' matches the message, the program is not executed.

  For all filters, list items can be quoted with single or double quotes. It is also possible to use each filter option multiple times, although this does not affect the order (not, and, or) in which filters are evaluated. A maximum of 32 filter patterns for each of (not, and. or) are supported per defined external program. Any filter not defined is not evaluated.

- It is possible to set a 'deadtime'. Within that 'deadtime', the respective external program will be executed only once (if triggered): **SetDeadtime**= `seconds`

3.1. Example setup for paging

The distribution contains two example perl scripts for paging and SMS messages (example_pager.pl, example_sms.pl). The paging script will page via a web CGI script at www.pagemart.com (obviously will work only for their pagers), the SMS script is for any German 'free SMS' web site that outsources free SMS to pitcom (with a suitable query on Google you can find such sites; you can then inspect the HTML form to set proper values for the required form variables.)

If you know some Perl, both scripts can be adapted fairly easily to other providers. Below is an example setup for calling example_pager.pl as an external logging facility.

```
[External]
# start definition of first external program
OpenCommand=/usr/local/bin/example_pager.pl
  SetType=log
  # arguments
  SetCommandline=example_pager.pl pager_id
  # environment
  SetEnviron=HOME=/home/moses
  SetEnviron=PATH=/bin:/usr/bin:/usr/local/bin
  # checksum
  SetChecksum=FCBD3377B65F92F1701AFEEF3B5E8A80ED4936FD0D172C84
  # credentials
  SetCredentials=moses
  # filter
  SetFilterOr=POLICY
  # deadtime
  SetDeadtime=3600
#Optional
CloseCommand
```

Chapter 8. Change Control Process Integration

samhain 4.0 introduces a set of new features to allow seamless integration with an existing change control process. This feature set has been drafted as the result of a workshop, and is designed to meet some key criteria:

1. The whole procedure can be run in an automated way, i.e. it can be executed by scripts and without human intervention, once a *list of affected files* is available. It is expected that the change control process will yield the list of affected files from the development or quality assurance stage.

2. The feature set provided should be rather generic and not tied to any particular change control software.

3. Immediately before a change is implemented, the affected files can be tested for their integrity. I.e. it can be verified that the system is in a *known-good* state before the change is put into effect.

4. After a change is implemented, the baseline, i.e. the database of known-good file signatures, can be updated.

5. The approval of the change(s) performed can be *securely* communicated to the running Samhain daemon on the affected machine(s), such that a subsequent file integrity check will raise no alerts.

1. Use cases

The following list shows the use cases that were considered, and how they may be handled.

1.1. Case I: Machine taken offline for a large patch

This case is best handled by a comple re-initialisation of the baseline database. The running Samhain daemon performs an on-demand file system scan immediately before the machine is taken offline to ensure a valid state, the database will be initialized after the patch has completed, and the Samhain daemon will re-start when the machine goes online again.

1. Before taking the machine offline, a SIGTTOU signal is sent to the Samhain daemon to request a file check:

   ```
   sh# kill -s TTOU $( cat /var/run/samhain.pid )
   ```

   It is possible to perform a wait-on-completion (with an optional timeout) that provides an exit status to indicate whether the file system scan found any inconsistencies:

   ```
   sh# samhain -w <timeout>
   ```

2. The machine is taken offline for patching (e.g. by switching to another runlevel), and Samhain is shut down.

3. After the patch, a new baseline is initialized:

   ```
   sh# samhain -t init
   ```

4. The new baseline is transferred to the server and the Samhain daemon restarts when the machine is taken online again.

5. There may be no feedback between different parts of the process that implements the change, thus the new baseline may not be available yet at restart. Therefore Samhain has an option to delay the download of the baseline at startup:

```
# Delay database download at startup by N seconds
StartupLoadDelay = N
```

1.2. Case II: Installation of a new package

In this case, because the package is not installed yet a "pre-flight" scan may be deemed unneccessary. After installation of the package, a *DeltaDB* (delta database) containing the added files will be generated and transferred to the server. There, it will be merged into the existing baseline database.

The *approval* of the file system changes will be done by the server asking the client to download the DeltaDB. For security, there is no client-side mechanism to trigger an approval of the file system changes.

1. The list of files (package content + files affected by pre-/postinstall scripts) is generated in the QA stage.

2. The package is installed.

3. Based on the list of files, a *DeltaDB* (delta database) is generated:

```
sh# samhain --outfile <DeltaDB> --create-database <file_list>
```

File list

One pathname per line, optionally preceded by a '+' (plus) sign which, if present, indicates that the *content* of the file should be stored. The configuration file will *NOT* be read, and the policy recorded in the baseline database will be ReadOnly.

4. The DeltaDB is transferred to the server data directory, renamed with a fixed scheme of the form <baseline_file>.<UUID>, and merged with the baseline database. The *merging* function is provided as part of the Beltane II software:

```
sh# beltane_update --merge <UUID> --update <baseline_file>
```

5. The Samhain server (yule) is advised to inform the Samhain client that a DeltaDB of approved changes is available for download:

```
sh# yulectl -c DELTA:<UUID> <client_fqdn>
```

Location and naming scheme

The DeltaDB must be named file.`client_fqdn.UUID` and must be located in the yule data directory, i.e. the same directory where the baseline database file.`client_fqdn` is located. The client will *only ask for the UUID*. Similar to the baseline

database, the pathname of the file is constructed by the server, using the client FQDN and the requested *UUID*.

6. The client receives the UUID and requests the DeltaDB from the server. In case of a failure, retrying is controlled by the following two configuration options:

```
# Maximum retry count
SetDeltaRetryCount = N
# Time in seconds between re-tries
SetDeltaRetryInterval = N
```

1.3. Case III: Configuration change / Package upgrade

This case differs from Case II insofar as there are already installed files, and therefore it is desirable to verify the integrity of those files before the change is put into effect. To perform this check, on the server a *PartialDB* (partial database, containing only data for affected files) is generated from the full baseline database. This PartialDB is then transferred to the client and used to perform a verification scan.

If the affected files are found to be in a consistent state, the procedure continues as in Case II then.

1. The list of affected files is generated in the QA stage.

2. On the Samhain server, a *PartialDB* (partial baseline database) for the affected files is generated from the complete baseline:

```
sh# samhain --outfile <PartialDB> --list-filter <list_file> --binary -d
```

3. The PartialDB is transferred to the client.

4. The integrity of the affected files is verified before the change is implemented (success or failure indicated by exit status):

```
sh# samhain --verify-database <PartialDB>
```

5. After successful verification, the process continues as in Case II.

2. Limitations

The **--verify-database** requires that the policy under which a file is checked is stored in the baseline database. (Note that this affects only this command. For the normal file system monitoring, the checking policy is taken from the configuratiion file, not from the baseline database.) For this reason, the format of the baseline database has changed in samhain 4.0. However, it is possible that the information about the check policy becomes incorrect:

Added files	If files are added to the filesystem after baseline initialisation and reported by the client, the correct policy should be set. To ensure this, the option **ReportCheckflags = yes** should be set in the client configuration (for backward compatibility, this option is off by default.)
Merging a DeltaDB	The DeltaDB is generated with the policy set to *ReadOnly*, to collect a complete set of checksum and metadata. However, if

the actual policy should be less restrictive because some of that data is allowed to change, a later **--verify-database** may result in spurious failures.

Client configuration change

If the configuration file for the client is changed to alter the checking policy for the monitored files, it is recommended to re-initialize the baseline.

Chapter 9. Additional Features — Signed Configuration/Database Files

Both the configuration file (see Section 1 (p. 117)) and the database of file signatures (Section 8 (p. 51)) may always be cleartext signed by either GnuGP (**gpg**) or the OpenBSD signify tool (available as **signify-openbsd** on Debian-based Linux flavours). The *recommended* options are:

gpg -a --clearsign --not-dash-escaped *FILE*

signify-openbsd -Se -s *PRIVATE_KEY* **-m** *FILE*

> ### Tip
>
> Please take note that on Linux, there exists a Perl application named signify which is completely unrelated to the OpenBSD signify cryptographic application. For this reason, on Debian-based Linux flavours, OpenBSD signify is named signify-openbsd. *Do not confuse signify-openbsd and signify on Linux. The latter is not what you want.*

1. Compiling with support for signatures

1.1. If using GnuPG

If compiled with support for signatures, samhain will invoke **gpg** to verify the signature. To compile with **gpg** support, use the option:

./configure --with-gpg=/full/path/to/gpg --with-fp=FINGERPRINT [--with-keyid=0x<hex KeyID>]

- Note that **gpg --fingerprint** will only list the fingerprint of primary keys. If you are signing with a secondary key, you need to repeat the '--fingerprint' option (i.e. run gpg **gpg --fingerprint --fingerprint**) in order to obtain the fingerprint for the signing (secondary) key. (If you don't know what a secondary key is, then this note is probably irrelevant for you.)

  Example(spaces in FINGERPRINT do not matter): **--with-fp="EF6C EF54 701A 0AFD B86A F4C3 1AAD 26C8 0F57 1F6C"**

- The optional argument **--with-keyid=0x<hex KeyID>** allows to specify a key ID, if there is more than one key in your keyring. This is only used for the installation routine, and for configuring the **samhainadmin.pl** convenience script (see below).

  The installation routine ("[sudo] make install") will use the keyring of the user running it (in ~/.gnupg) for signing. At runtime, samhain will use the keyring of the runtime user (usually root) for verification.

- samhain will check that the path to the **gpg** executable is writeable *only by trusted users*(see Section 10.1 (p. 9)).

- The **gpg** program will be called without using the shell, with its full path (as compiled in), and *with an environment that is limited to the HOME variable.*

  If you need LD_LIBRARY_PATH, because your gpg executable relies on libraries that are not in the search path of the loader, you can either (i) use a wrapper script to set the environment and exec gpg (take care not to mess with file descriptors), (ii) update the system loader configuration file, or (iii) recompile with loader paths (-Wl,-r<path> or -Wl,-R<path>).

- The public key must be in in the subdirectory HOME/.gnupg, where HOME is the home directory of the effective user (usually *root*).

- From the command line, the signature must verify correctly with **/path/to/gpg --status-fd 1 --verify** **_FILE_** when invoked by the effective user of samhain (usually *root*).

> ### Tip
>
> There is a Perl script **samhainadmin.pl** to facilitate some tasks related to the administration of signed configuration and database files (see Section 3 (p. 92)).
>
> When signing, the option *--not-dash-escaped* is recommended, because otherwise the database might get corrupted. However, this implies that after a database update, you *must* remove the old signature first, before re-signing the database. Without 'dash escaping', gpg will not properly handle the old signature. See the tip just above.

As signatures on files are only useful as long as you can trust the **gpg** executable, the **configure** script will determine the TIGER192 *checksum* of the **gpg** executable, which will be compiled into samhain. In case of an error, you can specify the checksum by hand with:

--with-checksum=" *CHECKSUM***"** — or — **--without-checksum**

CHECKSUM should be the checksum as printed by

gpg --load-extension tiger --print-md TIGER192 */path/to/gpg* — or — **samhain -H** */path/ to/gpg* (the full line of output, *with spaces*).

Example: **--with-checksum="/usr/bin/gpg: 1C739B6A F768C949 FABEF313 5F0B37F5 22ED4A27 60D59664"**

> ### WARNING
>
> Compiling in the GnuPG checksum will tie the samhain executable to the gpg executable. If you upgrade GnuPG, you will need to re-compile samhain. If you don't like this, use **'--with-checksum=no'** (or **'--without-checksum'** , which is equivalent).

1.2. If using signify

If compiled with support for signify signatures, samhain will invoke the **signify** executable to verify the signature. To compile with **signify** support, use the option:

./configure --with-signify=/full/path/to/signify --with-pubkey-checksum=PUBKEY_CHECK-SUM

- To determine the (TIGER192) checksum of the public key, you can either use samhain, by running the command: **samhain -H `key.pub`** or you can use GnuPG with the command **gpg --load-extension tiger --print-md TIGER192 `key.pub`** You only need the (48 character long) checksum part of the output, with or without whitespace stripped.

- samhain will check that the path to the **signify** executable is writeable *only by trusted users*(see Section 10.1 (p. 9)).

- The **signify** executable will be called without using the shell, with its full path (as compiled in), and *with an environment that is limited to the HOME variable*.

  If you need LD_LIBRARY_PATH, because your gpg executable relies on libraries that are not in the search path of the loader, you can either (i) use a wrapper script to set the environment and exec signify (take care not to mess with file descriptors), (ii) update the system loader configuration file, or (iii) recompile with loader paths (-Wl,-r<path> or -Wl,-R<path>).

- The public key must be in in the subdirectory `HOME/.signify`, where HOME is the home directory of the effective user (usually *root*). It MUST be named *INSTALL_NAME*`.pub`, where

INSTALL_NAME is the name under which samhain is actually installed (defaults to 'yule' for the server, 'samhain' otherwise, but configurable with the **./configure --install-name=*NAME*** option.)

- From the command line, the signature must verify correctly with **signify -Vem /dev/null -p ~/.signify/*INSTALL_NAME*.pub -x *FILE*** when invoked by the effective user of samhain (usually *root*).

> **Tip**
>
> There is a Perl script **samhainadmin.pl** to facilitate some tasks related to the administration of signed configuration and database files (see Section 3 (p. 92)).

As signatures on files are only useful as long as you can trust the **signify** executable, the **configure** script will determine the TIGER192 *checksum* of the **signify** executable, which will be compiled into samhain. In case of an error, you can specify the checksum by hand with:

--with-checksum="*CHECKSUM*" — or — **--without-checksum**

CHECKSUM should be the checksum as printed by

gpg --load-extension tiger --print-md TIGER192 */path/to/gpg* — or — **samhain -H */path/to/signify*** (the full line of output, *with spaces*).

Example: **--with-checksum="/bin/signify-openbsd: 1C739B6A F768C949 FABEF313 5F0B37F5 22ED4A27 60D59664"**

> **WARNING**
>
> Compiling in the signify checksum will tie the samhain executable to the signify executable. If you upgrade signify, you will need to re-compile samhain. If you don't like this, use **'--with-checksum=no'** (or **'--without-checksum'** , which is equivalent).

2. Installation

make install will sign the configuration file before installation.

```
bash$ ./configure --with-gpg=/usr/bin/gpg \
--with-fp=EF6CEF54701A0AFDB86AF4C31AAD26C80F571F6C
bash$ make
bash$ su
bash$ make install
bash$ samhain -t init
bash$ gpg -a --clearsign /var/lib/samhain/samhain_file
bash$ mv /var/lib/samhain/samhain_file.asc \
/var/lib/samhain/samhain_file
```

samhain will report the signature key owner and the key fingerprint as obtained from **gpg** . If both files are present and checked (i.e. when checking files against the database), both must be signed with the same key. If the verification is successful, samhain will only report the signature on the configuration file. If the verification fails, or the key for the configuration file is different from that of the database file, an error message will result.

3. The samhainadmin script

In the subdirectory scripts/ of the source directory you will find a Perl script **samhainadmin.pl** to facilitate some tasks related to the administration of signed configuration and database files (e.g. examine/create/remove signatures). By default, this script is *not installed*.

```
bash$ samhainadmin.pl --help
  samhainadmin.pl { -m F | --create-cfgfile }      [options] [in.cfgfile]
    Sign the configuration file. If in.cfgfile is given, sign it
    and install it as configuration file.

  samhainadmin.pl { -m f | --print-cfgfile }     [options]
    Print the configuration file to stdout. Signatures are removed.

  samhainadmin.pl { -m D | --create-datafile }     [options] [in.datafile]
    Sign the database file. If in.datafile is given, sign it
    and install it as database file.

  samhainadmin.pl { -m d | --print-datafile }    [options]
    Print the database file to stdout. Signatures are removed. Use
    option --list to list files in database rather than printing the raw file.

  samhainadmin.pl { -m R | --remove-signature } [options] file1 [file2 ...]
    Remove cleartext signature from input file(s). The file
    is replaced by the non-signed file.

  samhainadmin.pl { -m E | --sign }              [options] file1 [file2 ...]
    Sign file(s) with a cleartext signature. The file
    is replaced by the signed file.

  samhainadmin.pl { -m e | --examine }           [options] file1 [file2 ...]
    Report signature status of file(s).

  samhainadmin.pl { -m G | --generate-keys }     [options]
    Generate a PGP keypair to use for signing.

Options:
  -c cfgfile    --cfgfile cfgfile
    Select an alternate configuration file.

  -d datafile    --datafile datafile
    Select an alternate database file.

  -p passphrase --passphrase passphrase
    Set the passphrase for gpg. By default, gpg will ask.

  -s gnupg_homedir --secretkeyring gnupg_homedir
    Select an alternate gpg homedirectory to locate the secret keyring.
    Will use '/home/rainer/.gnupg/' by default.

  -k keyid       --keyid keyid
   Select the keyid to use for signing.

  -l             --list
    List the files in database rather than printing the raw file.

  -v             --verbose
    Verbose output.
```

Chapter 10. Additional Features — Stealth

If an intruder does not know that samhain is running, s/he will make no attempt to subvert it. Hence, you may consider to run samhain in stealth mode, using some of the options discussed in this section.

1. Hiding the executable

samhain may be compiled with support for a stealth mode of operation, meaning that the program can be run without any obvious trace of its presence on disk. The following compile-time options are provided:

--enable-stealth= *xor_val* provides the following measures:

1. All embedded strings are obfuscated by XORing them with some value *xor_val* chosen at compile time. The allowed range for *xor_val* is 128 to 255.

2. The messages in the log file are obfuscated by XORing them with *xor_val*. The built-in routine for validating the log file (**samhain -L /path/to/logfile**) will handle this transparently. You may specify as path an already existing binary file (e.g. an executable, or a JPEG image), to which the log will get appended.

> ### Tip
>
> Use **samhain -jL /path/to/logfile** if you just want to view rather than verify the logfile.

3. Strings in the database file are obfuscated by XORing them with *xor_val*. You may append the database file to some binary file (e.g. an executable, or a JPEG image), if you like.

4. The configuration file must be steganographically hidden in a postscript image file (the image data must be uncompressed). To create such a file from an existing image, you may use e.g. the program **convert** , which is part of the ImageMagick package, such as **convert +compress** *ima.jpg ima.ps* .

> ### Tip
>
> **make install** will do this automatically before installation.

To hide/extract the configuration data within/from the postscript file, a utility program **samhain_stealth** is provided. Use it without options to get help.

> ### Note
>
> If **--enable-stealth** is used together with **--with-gpg** or **--with-signify** , then the config file must be signed before hiding it (rather than signing the PS image file afterwards).
>
> Furthermore, in the case of **--with-signify**, the config file MUST end with the '[EOF]' marker, as the only content of the LAST line. If there are blank lines afterwards, they will not get extracted, and the signature verification will fail.

--enable-micro-stealth= *xor_val* is like **--enable-stealth** , but uses a 'normal' configuration file (not hidden steganographically).

--enable-nocl[=ARG] will disables command line parsing. The optional argument is a 'magic' word that will enable reading command-line arguments from *stdin*. If the first command-line argument is

not the 'magic' word, all command line arguments will be ignored. This allows to start the program with completely arbitrary command-line arguments.

--enable-install-name= *NAME* will rename every installed file from *samhain* to *NAME* when doing a **make install** (standalone/client installation), and likewise rename installed files from *yule* to *NAME* when doing a **make install** (server installation). Also, the boot scripts will be updated accordingly. Files created by samhain (e.g. the database) will also have *samhain* replaced by *NAME* in their file-names.

> ### Tip
>
> The man pages have far too much specific information enabling an intruder to infer the presence of samhain. There is no point in changing *samhain* to *NAME* there — this would rather help an intruder to find out what *NAME* is. You probably want to avoid installing man8/samhain.8 and man5/samhainrc.5.

1.1. Using kernel modules to hide samhain (Linux/ix86 only)

This feature has been removed as of samhain version 4.0, because it does not work anymore on modern kernels due to enhanced kernel hardening.

2. Packing the executable

For even more stealthyness, it is possible to pack and encrypt the samhain executable. The packer is just moderately effective, but portable. Note that the encryption key of course must be present in the packed executable, thus this is no secure encryption, but rather is intended for obfuscation of the executable. There is a make target for packing the samhain executable:

make samhain.pk

On execution, **samhain.pk** will unpack into a temporary file and execute this, passing along all command line arguments. The temporary file is created in /tmp, if the sticky bit is set on this directory, and in /usr/bin otherwise. The filename is chosen at random, and the file is only opened if it does not exist already (otherwise a new random filename will be tried). The file permission is set to 700.

The directory entry for the unpacked executable will be deleted after executing it, but on systems with a /proc filesystem, the deleted entry may show up there. In particular, this is the case for Linux. You should be aware that this may raise suspicion.

On Linux, the /proc filesystem is used to call the unpacked executable without a race condition, by executing /proc/self/fd/NN, where NN is the file descriptor to which the unpacked executable has been written. On other systems, the filename of the unpacked executable must be used, which creates a race condition (the file may be modified between creation and execution).

The packed executable will not honour the SUID bit.

Chapter 11. Deployment to remote hosts

1. Method A: The deployment system

samhain includes a system to facilitate deployment of the client to remote hosts. This system enables you to: build and store binary packages for different operating systems, install them, create baseline databases upon installation, update the server configuration, and maintain the client database required by the beltane web-based console.

The system comprises a shell script **deploy.sh** that will be installed in the same directory as the samhain/yule (by default, `/usr/local/sbin`), and a directory tree that will be installed below the samhain/yule data directory (see Section 1.2 (p. 97)). The script and the directory tree can be relocated freely. There is a configuration file `~/.deploy.conf` that is created in the home directory of the user when **deploy.sh** is run for the first time, where you can specify the default for the top level directory of the system.

> ### Note
>
> In the following, an *architecture* is just a label for some group of hosts, typically a particular operating system (or some particular installation thereof). As long as you know what the label means, you can choose it freely (alphanumeric + underscore).
>
> The architecture for a build/install host (i.e. the association between a host and the architecture-specific configuration data) is currently specified via a command-line option.

The system allows to use per-architecture customized build options, as well as per-host customized runtime configuration files.

By default, the system will search for a sufficiently advanced incarnation of dialog to provide a nice user interface. You can switch this off in favour of a plain console interface, if you prefer (or if you want to run the script non-interactively).

To use this system, you must first install it with the command:

```
sh$ make install-deploy
```

> ### Installation tip
>
> This system is somewhat tied to the server (yule). While you can safely install it later, installing it together with the server will take care that the defaults are already correct. Upon first invocation a configuration file `~/.deploy.conf` will be written, where you can modify the defaults settings.

> ### Backward compatibility
>
> The deployment system has been completely revised in version 2.0 of samhain. It will *not* work with samhain versions below 2.0 (i.e. you cannot install them using this system). However, the default location and format of the client database (used by the beltane web-based console) has not changed.
>
> Installing the new version of the deploy system will not overwrite the old version (deploy.sh will be installed as deploy2.sh, if an old version is detected).

1.1. Requirements

1. You must have compiled and installed the server (yule) on the local host where you use the deploy system.

2. You must have installed the deployment system by using **make install-deploy** . This will install the script `deploy.sh` into the sbindir(default `/usr/local/sbin`, but depends on your configure options), and the deployment system into `localstatedir/install_name/profiles`(default `/var/lib/yule/profiles`, but depends on your configure options).

   If you already have installed the deprecated version 1 deployment system, the script will be installed as `deploy2.sh`.

3. For each architecture that you define, there must be (at least) one *build host* where development tools (C compiler, make, strip) are available to build the client executable.

4. On each remote where you want to build or install, you should be able to login as root with ssh using RSA authentication, such that ssh-agent can be used.

   > ## Tip
   >
   > To use RSA-based authentication in a *secure* way, you may proceed as follows:
   >
   > Use **ssh-keygen** to create a public/private key pair. Don't forget to set a passphrase for the private key (**ssh-keygen** will ask for it).
   >
   > Copy the public key (`HOME/.ssh/identity.pub` for the ssh protocol version 1, `HOME/.ssh/id_rsa.pub` for ssh protocol version 2) to `HOME/.ssh/authorized_keys` on any remote host where you want to log in. *Do not copy the private key* `HOME/.ssh/identity`(ssh protocol version 1) or `HOME/.ssh/id_rsa`(ssh protocol version 2) to any untrusted host !
   >
   > On your central host, execute the commands (use "ssh-agent -c" if you are using a csh-style shell):
   >
   > ```
   > bash$ eval `ssh-agent -s`
   > bash$ ssh-add
   > ```
   >
   > You can then ssh/scp without typing the passphrase again, until you exit the current shell.

1.2. Layout of the deployment system

```
(localstatedir)/(install_name)/profiles/
                      |
                      |
                      |-- source ------------> (tarballs)
                      |
                      |-- configs -----------> (default configs)
                      |
                      |-- archpkg
                      |    |
                      |    |-- architecture -> (compiled package, setup script)
                      |
                      |-- hosts
                      |    |
```

```
|      |-- hostname -----> (optional host-specific config)
|
|-- libexec ----------> (scripts)
|
|-- private ----------> (gpg key)
|
|-- tmp
```

1.2.1. The configs subdirectory

The `configs` subdirectory holds for each architecture at least two files (example files will be placed there upon installation of the deployment system):

<architecture>.configure (required)	The configure options for this architecture; one option per line, each enclosed in single quotes.
	If this file does not exist, it will be copied from `generic.configure`, which is created upon installation, and holds (only) some minimum options.
<architecture>.samhainrc (required)	The default runtime configuration file for a client running on this architecture. It is possible to override this on installation with a file `hosts/<hostname>/samhainrc`.
<architecture>.preinstall (optional)	The shell script to run before installing a client. Typically this script would shutdown the running client, if there is one.
	Defaults to `libexec/preinstall`.
<architecture>.postinstall (optional)	The shell script to run after installing a client. This script receives the client password as first (and only) argument, and typically would set the password in the client binary.
	Defaults to `libexec/postinstall`.
<architecture>.initscript (optional)	The shell script to initialize/update the baseline database file installing a client.
	Defaults to `libexec/initscript`.

1.2.2. The archpkg subdirectory

The `archpkg` directory holds for each architecture a subdirectory `archpkg/<architecture>`, where compiled binary installer packages are stored.

For each build, up to four files will be stored: (a) the binary installer package `samhain-<version>.<format>`, (b) the configure options used (`configure-<version>.<format>`), (c) the samhain-install.sh script generated during the build (`install-<version>.<format>`), and (only for packed executables) the client password set in the executable (`PASSWD-<version>.<format>`).

1.3. Customizing the system

1.3.1. Setting default options

If you want to change the default options, you can set some of them via a configuration file `~/.deploy.conf`, which is created upon the first invocation of **deploy.sh** .

1.3.2. Adding support for an architecture

To add support for another architecture <arch>, just create the two files `<arch>.configure`(configure options) and `<arch>.samhainrc`(runtime configuration) in the `configs` directory of the deployment system (see Section 1.2 (p. 97)).

Upon installation of the system, a template file `generic.configure` is created, which contains the minimum options for a client.

1.3.3. Per-architecture pre-/postinstallation scripts

The default scripts for preinstallation (shutting down the running client) and postinstallation (setting the client password, fixing the local configuration file), and the script for database initialization are located in the `libexec` directory. You can override them for some (or all) architectures by storing architecture-specific files `<arch>.preinstall`, `<arch>.postinstall`, `<arch>.initscrip` in the `configs` directory.

1.3.4. Per-host runtime configuration

If you want to override the runtime configuration file `configs/<arch>.samhainrc` on a per-host basis, you need to store a host-specific runtime configuration file as `hosts/<hostname>/samhainrc`, before you run **deploy.sh install** .

1.4. Using the deploy.sh script

Tip

When run for the first time, **deploy.sh** will create a configuration file `~/.deploy.conf` with some default configuration options. You may want to review this file. Note that you can override all options there with command-line options; the configuration file is just for convenience, if you don't like the defaults and don't want to type the corresponding option on the command line every time.

deploy.sh can be invoked in three ways:

```
        bash$ deploy.sh --help
 This will provide a general overview.
        bash$ deploy.sh --help command
 This will provide help on a specific command
 (where command can be any of: 'clean', 'download', 'checksrc',
 'build', or 'install'.
        bash$ deploy.sh [options] command
 This will run 'command' with the specified options.
```

A log of the last run will be kept in `tmp/logfile.lastrun`

command can be any of the following:

info   Provides information on installed clients, or available installer packages.

clean   Removes source tarballs from the `source` subdirectory of the deploy system. Removes unused installer packages from the `archpkg/<arch>` subdirectories of the deploy system.

download   Download a source tarball from the distribution site, verify the GnuPG signature (gpg must be installed), and install it into the `source` subdirectory of the deploy system. *Requires* one of: **wget** , **curl** , **links** , **lynx** , **fetch** , or **lwp-request** .

checksrc   Check the GnuPG signatures of available source tarballs in the `source` subdirectory of the deploy system (gpg must be installed). Optionally delete tarballs with no/invalid signature.

build   Build a binary installer package for the chosen architecture from one of the tarballs in the `source` subdirectory, and store it in the `archpkg/<architecture>` subdirectory (which will be created if it does not exist yet). Requires a file `<architecture>.configure` and a file `<architecture>.samhainrc` in the `configs` subdirectory.

install   Copy a pre-built binary package (built with **deploy.sh build**) to a remote host, stop the client running there (if any), install the (new) client, update the server configuration file and reload the server, initialize the file signature database and fetch it from the remote host.

uninstall   Remove a samhain client that was previously installed with **deploy.sh install** .

1.4.1. General options

-q | --quiet | --quiet=2 Produce output suitable for logging. Note that --quiet=2 implies --yes (see below).

-s | --simulate Print what would be done, but do not actually change the system.

-y | --yes Assume yes as answer to all prompts and run non-interactively.

-o <file> | --logfile=<file> Specify an output file for messages that would go to stdout otherwise. Has no effect on stderr (error messages).

-d <dialog> | --dialog=<dialog> Specify your preferred "dialog" clone (e.g. Xdialog). Use "no" to force plain text.

1.5. deploy.sh info

This command will show information for hosts in the client database (default), or for available binary installer packages.

1.5.1. Specific options

--packages Show information for available installer packages rather than for clients.

1.6. deploy.sh clean

This command will clean unused files: source tarballs in the `source` subdirectory, and unused installer packages in the `archpkg/>arch>` subdirectories.

1.6.1. Specific options

There are no specific options for this command.

1.7. deploy.sh download

This command will download a source tarball from the distribution website, verify its GnuPG signature, and install it into the `source` subdirectory. This command requires that either **wget** or **lynx** is in your PATH.

Manual installation of source

This note applies if you want to download source manually instead. Samhain distribution tarballs contain exactly two files: first, a source tarball with the source code, and second,

its GnuPG signature. For installation into the `source` subdirectory, the distribution tarball must be unpacked, and both the source source tarball and its GnuPG signature moved into the `source` subdirectory.

1.7.1. Specific options

--version=<version> The version of samhain to download. The default is "current" to download the current version.

1.8. deploy.sh checksrc

This command will check the GnuPG signatures of source tarballs in the `source` subdirectory.

1.8.1. Specific options

--delete Delete source tarballs if PGP signature cannot be verified.

1.9. deploy.sh build

This command will create a temporary directory on a remote build host, copy the selected version of the source there, build the selected format of the binary installer package, retrieve and store the package into the `archpkg/>architecture>` subdirectory, and remove the temporary build directory.

For each build, up to four files will be stored: (a) the binary installer package `samhain-<version>.<format>`, (b) the configure options used (`configure-<version>.<format>`), (c) the samhain-install.sh script generated during the build (`install-<version>.<format>`), and (only for packed executables) the client password set in the executable (`PASSWD-<version>.<format>`).

Package formats

Note that the build host must provide the required tools if you want to build a package for the native package manager (i.e. deb, rpm, tbz2, depot (HP-UX), or solaris pkg). On RPM-based Linux distributions and Gentoo Linux, building of RPMs and tbz2s, respectively, should just work. Debian requires additional packages for building debs.

The "run" binary package format does not require additional tools (it is a self-extracting tar package based on the makeself application, which is included in the samhain distribution). Use **/bin/sh <package> --help** for details.

1.9.1. Specific options

--host=<hostname> The build host.

--arch=<arch> The architecture to build for. This is used to get the "./configure" options from the file `configs/<arch>.configure`, and to store the binary package into the directory `archpkg/<arch>`.

--version=<version> The version of samhain you want to build. Must be in the `source` subdirectory.

--format=<run|rpm|deb|tbz2|depot|solaris-pkg> The format of the binary installer package. "run" is a portable (Unix) package based on makeself, "deb" is a Debian package, "tbz2" is a binary Gentoo Linux package, "rpm" is an RPM package, "depot" is an HP-UX binary package, and "solaris-pkg" for Sun Solaris.

--packed=<password> Build a packed executable, and set the client password before packing.

--user=<username> Login as <username> on the build host (defaults to root).

--add-path=<path> Append <path> to the PATH variable on the build host.

--tmpdir=<path> Temporary directory to use on the build host (defaults to `/tmp`).

1.10. deploy.sh install

This command will create a temporary directory on a remote host, copy the selected version of the installer package, its corresponding samhain-install.sh script, the runtime configuration file, and the preinstall, postinstall, initscripts scripts there. It will then:

(A) run the `preinstall` script on the client, which shuts down the running samhain daemon (if there is any).

(B) install the binary installer package on the client.

(C) run the `postinstall` script on the client, which sets the client password (unless the binary is packed), and replaces the default runtime configuration file with the proper one. The latter step is required, because **deploy.sh build** builds from the pristine source, so the runtime configuration file in the installer package is just the default one.

(D) copy the proper client runtime configuration file to the server data directory (as `rc.<client_name>`), fix the server configuration file, and restart the server (which will fail non-fatally if the server is not running).

(E) run the `initscript` script on the client, which initializes (or updates) the baseline database.

(F) retrieve the baseline database, copy it to the server data directory (as `file.<client_name>`), and remove the temporary directory on the client.

The runtime configuration file

If `hosts/<hostname>/<arch>.samhainrc` or `hosts/<hostname>/samhainrc` exists, this will be used (in this order of preference), otherwise `configs/<arch>.samhainrc` will be used. If the latter does not exist, the command will fail.

Transparent handling of particular build options

The build options '--enable-stealth=..' is handled by determining the argument from the configure options that were used for the build, and preparing the runtime configuration file appropriately. I.e., you should provide a 'normal', plain-text configuration file.

The build option '--with-nocl=..' is handled by determining the argument (which is required for database initialization) from the configure options that were used for the build, and passing it to the `initscript` script.

1.10.1. Specific options

--host=<hostname> The host on which to install.

--group=<foobar> The group to which you want to assign that client (default: none). This is used by the beltane web console to restrict access to users which are members of that group.

--arch=<arch> The architecture to install. This is used to get the installer package from the directory `archpkg/<arch>/`.

--version=<version> The version of samhain you want to install. An installer package for this version must exist in the `archpkg/<arch>/` subdirectory.

--format=<run|rpm|deb|tbz2|depot|solaris-pkg> The format of the binary installer package. "run" is a portable (Unix) package based on makeself, "deb" is a Debian package, "tbz2" is a binary Gentoo Linux package, "rpm" is an RPM package, "depot" is an HP-UX binary package, and "solaris-pkg" for Sun Solaris.

--yule_exec=<path> Path to the yule executable.

--yule_conf=<path> Path to the yule configuration file.

--yule_data=<path> Path to the yule data directory.

--no-init Do not initialize the file signature (baseline) database (and consequentially, do not replace the `file.<host>` file on server.

--no-rcfile Do not replace the `rc.<host>` file on server.

--no-start Do not start up the client after installation.

--local=<command> An optional command executed locally (i.e. on the server) twice (with the last argument set to 'first' and 'second', respectively. First is after client config file installation (i.e. before baseline database initialisation on the client), second is just before client startup. The script will be called with the following five arguments:

1. *hostname*, i.e. the host on which to install,

2. *arch*, the architecture to install,

3. the directory where the deployment system is installed (default is `/var/lib/yule/pro-files`, but depends on your configure options),

4. the server data directory where client configuration files and baseline databases are stored (default is `/var/lib/yule`, but depends on your configure options),

5. the literal word 'first' if the script is called the first time during client installation, the literal word 'second' otherwise.

--tmpdir=<path> Temporary directory to use on the installation host (defaults to `/tmp`).

1.11. deploy.sh uninstall

This command will remove a samhain client that was previously installed by using **deploy.sh install** .

1.11.1. Specific options

--host=<hostname> The host on which to uninstall.

--tmpdir=<path> Temporary directory to use on this host (defaults to `/tmp`).

1.12. Usage notes

Warning

On Solaris, the PATH environment variable on the remote host (where you build or deploy) may get set according to /etc/default/su, which may be different from what you would expect (noted by S. Bailey).

2. Method B: The native package manager

Samhain provides an easy method to create *custom binary packages* with the native package manager of your operating system. Basically, this works like:

```
bash$ ./configure [your preferred options]
bash$ make rpm|deb|tbz2|depot|solaris-pkg
```

I.e. the binary package will be built with the compile options chosen in the preceding **./configure** command. Supported package formats are: *rpm* (e.g. Redhat, SuSE, ...), *deb* (Debian), *tbz2* (Gentoo Linux), *depot* (HP-UX), and *solaris-pkg* (Solaris).

Customization

The binary package will use the OS-specific `samhainrc.OS` configuration file from the source directory, thus if you customize this, your package will contain your customized version.

Baseline initialization

Upon installation, the package will not automatically initialize the baseline database, and not start the daemon (though it will install the runlevel script to start upon boot).

Client packages

Samhain uses a password embedded in the binary for client/server authentication (for details see Section 3 (p. 73)). To avoid the need for changing the binary after installation (to set the password), it is possible to compile it in during package building (for RPM and DEB packages only). To do so, export the password in the PASSWORD environment variable before running the command **make rpm** or **make deb** .

Note

For reasons explained in Section 2 (p. 106) , we do not recommend to distribute binary packages to third parties. On the other hand, it is perfectly ok to use a self-built binary package to install/distribute samhain on your machine/within your own network.

2.1. Building an RPM

2.1.1. Custom RPM

If you run **./configure** in the source directory, a spec file `samhain.spec` will be created from `samhain.spec.in`. You can then use **make rpm** to create source and binary RPMs, or **make srpm** to create just the source RPM.

The RPM will be located in `/usr/src/(distribution-specific)/RPMS/i386`. Installing the RPM will *not initialize* the database automatically.

If anything fails during the build (and after installation has begun), just cd into the build directory and do a **make uninstall && make uninstall-boot** . If building for a non-RedHat system, the error messages will tell you which file paths in the spec file were incorrect.

2.1.2. Single-host

If you want to create an RPM for a single-host version of samhain without any fancy options, you can just run

```
bash$ rpmbuild -ta samhain-version.tar.gz
```

on the tarball (there is a default spec file in there).

The RPM will be located in `/usr/src/(distribution-specific)/RPMS/i386`. Installing the RPM will *not initialize* the baseline database automatically.

2.2. Building an HP-UX package

First run **./configure** in the source directory with your preferred options, then do a **make depot** . The result should be a package named `samhain.depot`, that can be installed with **swinstall** . Installing the package will *not initialize* the baseline database automatically.

2.3. Building a Solaris package

Note

This is experimental and not well tested. Constructive feedback from experienced Solaris administrators is welcome.

First run **./configure** in the source directory with your preferred options, then do a **make solaris-pkg** . The result should be a package named `samhain.pkg`.

2.4. Building a Gentoo Linux package

First run **./configure [your preferred options]** in the source directory (reminder: use **./configure --prefix=USR** , NOT **./configure --prefix=/usr** for standard paths), then do a **make tbz2** . The `.tbz2` package will be in `/usr/portage/packages/All`(this is just how Gentoo package building works).

The Gentoo package thus created will *not initialize* the database automatically upon installation. The `.tbz2` package file will be in `/usr/portage/packages/All`(this is just how Gentoo package building works).

Note

If you just want to install on your own system, rather than building a package for other machines, you can use the command **make emerge** (after running **./configure** , of course).

2.5. Building a Debian package

First run **./configure** in the source directory (reminder: use **./configure --prefix=USR** , NOT **./configure --prefix=/usr** for standard paths), then do a **make deb** . The `.deb` package and the corresponding `.dsc` file will be in the directory above the source directory (this is just how Debian package building works).

You will need the following additional Debian packages in order to build a Debian packages: **apt-get fakeroot** , **apt-get debmake** , **apt-get debhelper** , **apt-get devscripts** , and **apt-get cpio** .

The Debian package thus created will *not initialize* the database automatically upon installation. It will be located in the *parent directory* of the source directory (that's just the way the Debian build system works).

Chapter 12. Security Design

1. Usage

It is recommended to:

- *compile a static binary*(not linked to shared libraries), using the configure option **--enable-static** if possible. Some functionality may not be available with static binaries, in particular some name services (NIS, LDAP), the portmap service (used for the port check), and the support for the Linux audit daemon and SQL databases. On some operating systems (e.g. Solaris) static compiling may not be possible at all.

  The advantage is that a static binary is self-contained, i.e. if you are sure of its integrity (e.g. because it is on a write-protected medium) then you don't need to worry about the integrity of system libraries that dynamically linked binaries depend upon. The disadvantage is, if there is a bug in one of the compiled-in libraries, then you need to re-compile the static binary after a library upgrade.

- *strip the binary*. This will help somewhat against intruders that try to run it under a debugger ...

- *use signed database/configuration files*.

- *take a look at the stealth options*- while 'security by obscurity' *only* is a very bad idea, it certainly helps if an intruder does not know what defenses you have in place

- read the next chapter to understand how the *integrity of the samhain executable* can be verified.

1.1. Client security in a client/server system

In a client/server Samhain system, if an intruder has obtained root privileges on the server he may modify configuration files that are stored on the server and downloaded by the clients. Thus, if the client executes shell commands given in the configuration file (via the shell expansion option, or by logging events to external commands specified in the configuration file), this may allow the intruder to take over the clients as well.

As of version 2.8.5, there are two ways to protect against this scenario:

first, you can use signed configuration (and baseline database) files. The signature is checked on the client, after downloading the configuration file from the server. It is thus not possible to make the client perform any actions if the configuration file is not signed correctly (note that in versions before 2.8.5, the signature would be checked too late to prevent the attack).

second, you can just forego any execution of external programs by compiling with the options **--disable-shellexpand --disable-external-scripts** . No shell expansion will be performed on configuration file directives, and no logging to external programs will be supported.

2. Integrity of the samhain executable

Each samhain executable contains a compiled-in key, that is used when the signatures of emails and/or logfile entries are verified. By default, a cryptographically strong random key is generated by the **configure** script at compile time. Thus, each build is unique, and signature verification will fail if a different build is used, except if the compiled-in key was set to a common value for both builds.

To set a user-defined key, there is an option

./configure --enable-base= *B1,B2*

where B1,B2 should be two integers in the range 0...2147483647.

The key generated by **configure** is printed in the configure script's output. It is recommended that you save this key and use it for further builds.

Whenever you try to verify the integrity of e-mails or log file entries, this compiled-in key is used (to be more specific: the signature key is encrypted with a one-time pad generated from the message itself and the compiled-in key). As a result, if executable B is used to verify the integrity of e-mails sent by executable A, *integrity verification will fail* if the compiled-in keys of A and B do not match. This can be used to check the integrity of A in a straightforward way (check e-mails on another host, using a different executable compiled with the same key).

Obviously, this scheme can be broken, but it requires an intruder to disassemble/decompile and analyze the existing samhain executable, rather than simply replace it with a precompiled trojan.

However, if you use a *precompiled* samhain executable (e.g. from a binary distribution), in principle a prospective intruder could easily obtain a copy of the executable and analyze it in advance. This will enable her/him to generate fake audit trails and/or generate a trojan for this particular binary distribution.

For this reason, it is possible for the user to add more key material into the binary executable. This is done with the command:

samhain --add-key= ***key@/path/to/samhain_executable***

This will read the file `/path/to/samhain_executable`, add the key *key*, which can be a string of arbitrary length, except that it should not contain a '@' (because it has a special meaning, separating key from path), and write the new binary to the location `/path/to/executable.out`(i.e. with .out appended).

For Clarification

Please note that --add-key does not replace a compiled-in key but only adds to it. Integrity verification depends on both the compiled-in and any added key material, and integrity verification using different binaries will therefore only work if all were compiled with the same key and had the same extra key material (if any) added in.

WARNING

Using a precompiled samhain executable from a binary package distribution is not recommended unless you add in key material as described above.

3. Client executable integrity

If you use samhain in a client/server setup, the client needs to authenticate to the server using a password that is located within the client executable, at one of several possible places (where the valid place for your particular build is chosen at random at compile time). If the password is set, the alternative places are filled with random values.

Upon authentication to the server, client and server negotiate ephemeral keys for signing and encrypting further communication.

This implies that an intruder needs to analyse the running process to obtain knowledge of the signing/encryption keys in order to successfully fake a valid communication with the server, or she needs to analyse/disassemble the executable in order to find the password.

4. The server

The server does not need root privileges. Therefore, if it is started with root privileges, it will drop them irrevocably after startup. If a privileged port (below 1024) must be opened, the server will first open it, then drop root, and only thereafter accept any connection on the port.

The server can be chrooted, and actually has a config file option to do so by itself (which means that you don't need to copy shared libraries into the chroot environment).

(If your clients are configured to download baseline databases and configuration files from the server:) The server does not need write access to the directory where client baseline databases and configuration files are stored, and it would be wise to deny such access (chown to some other user, and allow group read access for the server).

5. General

Obviously, a security application should not open up security holes by itself. Therefore, an inportant aspect in the development of samhain has been the security of the program itself. While samhain comes with no warranty (see the license), much effort has been invested to identify security problems and avoid them.

As the client requires root privileges, while the server does not, the clients has no open socket to listen on the network. Consequently, all client/server connections are initiated by the client.

To avoid buffer overflows, only secure string handling functions are used to limit the amount of data copied into a buffer to the size of the respective buffer (unless it is known in advance that the data will fit into the buffer).

On startup, the timezone is saved, and all environment variables are set to zero thereafter. Signal handlers, timers, and file creation mask are reset, and the core dump size is set to zero. If started as daemon, all file descriptors are closed, and the first three streams are opened to `/dev/null`.

If external programs are used (in the entropy gatherer, if `/dev/random` is not available), they are invoked directly (without using the shell), with the full path, and with a limited environment (by default only the timezone). Privileged credentials are dropped before calling the external program.

With respect to its own files (configuration, database, the log file, and its lock), on access samhain checks the complete path for write access by untrusted users. Some care has been taken to avoid race conditions on file access as far as possible.

Critical information, including session keys and data read from files for computing checksums, is kept in memory for which paging is disabled (if the operating system supports this). This way it is avoided that such information is transfered to a persistent swap store medium, where it might be accessible to unauthorized users.

Random numbers are generated from a pseudo-random number generator (PRNG) with a period of 2^{88} (actually by mixing the output from three instances of the PRNG). The internal state of the PRNG is seeded from a strong entropy source (if available, `/dev/random` is used, else lots of system statistics is pooled and mixed with a hash function). The PRNG is re-seeded from the entropy source at regular intervals (one hour).

Numbers generated from a PRNG can be predicted, if the internal state of the PRNG can be inferred. To avoid this, the internal state of the PRNG is hidden by hashing the output with a hash function.

Appendix A. List of options for the configure script

1. General

--with-rnd=egd/dev/unix/default	The entropy gatherer to use. 'egd' is the Entropy Gathering Daemon (EGD), 'dev' is /dev/random, 'unix' is the built-in Unix entropy gatherer (similar to EGD), and 'default' will check for /dev/random first, and use 'unix' as fallback.
--with-egd-socket=NAME	The path to the EGD socket. Default is `localstatedir/lib/samhain/entropy`(see Section 6 (p. 112)).
--enable-identity=USER	The username to use when dropping root privileges (default *nobody*).
--with-sender=SENDER	The username of the sender for e-mail, or a complete e-mail address. If only a username is given, *SENDER@{FQDN_of_local_host}* will be used for the sender. Default is *daemon*.
--with-recipient=ADDR	The recepient(s) for e-mail, separated by whitespace (max. 8). You can add recepients in the configuration file as well.
--with-trusted=UID	Trusted users (must be a comma-separated list of numerical UIDs). Only required if the configuration file must be on a path writeable by others than *root* and the *effective* user.
--with-timeserver=HOST	Set host address for time server (default is to use own clock). You can set this in the configuration file as well. An address in the configuration file will take precedence. Note that the simple 'time' service (port 37/tcp) is used.
--with-alttimeserver=HOST	Set host address for an alternative (backup) time server.
--enable-stealth=XOR_VAL	Enable stealth mode, and set XOR_VAL. XOR_VAL must be decimal, in the range 127 -- 255, and will be used to obfuscate literal strings.
--enable-micro-stealth=XOR_VAL	As **--enable-stealth** , but without steganographic hidden configuration file.
--enable-nocl=PW	Command line parsing is disabled, but command-line arguments will be read from STDIN if the first command line argument is PW. PW="" (empty string) will disable command line parsing completely. This option may be used as addition to --enable(-micro)-stealth to prevent interactive enforcement of telltale output.
--enable-install-name=NAME	Upon installation, rename every file from `samhain`(or `yule` for the server) to `NAME`. To be used in conjunction with --with-(micro-)stealth.
--enable-base=B1,B2	Set compiled-in key for email and logfile signature verification. ONE string (no space) made of TWO comma-separated integers in the range 0 -- 2147483647. See Section 2 (p. 106) for details on this option.

--enable-db-reload	[CLIENT ONLY] Enable reload of file database on SIGHUP (otherwise, only the config file will be read again).
--enable-xml-log	Enable XML format for the log file.
--with-database=mysql/post-gresql/oracle/odbc	Support logging to a relational database (MySQL, PostgreSQL, Oracle or unixODBC). Oracle and unixODBC are not fully tested. Requires the relevant development packages (e.g. libmysql-client-dev on Ubuntu for MySQL).
--with-prelude	Support logging to the Prelude IDS system. Requires the libprelude library.
--with-libprelude-prefix=PFX	Prefix where libprelude is installed. This will be used to search **libprelude-config** in the PFX/bin/ directory.
--disable-ipv6	Disable IPv6 support.
--enable-debug[=gdb]	Enable debugging. Will slow down things, increase resource usage, and may leak information that should be kept secure. Will dump 'core' and 'samhain_backtrace' in the root directory on segfault. Do not use in production code.
	If used as --enable-debug=gdb, will *only* compile in debugging symbols for the GNU gdb compiler. This is more suitable for debugging the code itself.
--enable-ptrace	Periodically check whether a debugger is attached, and abort if yes. Only takes effect if **--enable-debug** is not used. Only tested on Linux.
--with-cflags=FLAGS	Additional flags to pass to the compiler.
--with-libs=LIBS	Additional libraries to link with.
--disable-largefile	Disable support for large files (> 2GB). Large file support is enabled automatically if your system supports it.
--enable-posix-acl	Compile with support for checking posix ACLs of files. This is compiled in by default if the required libraries and header files are present, but using this option will turn the 'soft fail' into a 'hard fail' if it can't be compiled in.
--enable-selinux	Compile with support for checking SELINUX attributes of files. This is compiled in by default if the required libraries and header files are present, but using this option will turn the 'soft fail' into a 'hard fail' if it can't be compiled in.
--enable-udp	This options enables code to listen on port 514/upd, i.e. the syslog port. Thus the server can receive syslog reports from remote hosts (if they are configured to send), and log them to any of the log facilities supported by samhain. If you compile in support for this, you still need to enable it in the runtime configuration file.
--disable-dnmalloc	This options disables use of the dnmalloc allocator that is the default since samhain 2.4.5, and reverts to using the standard allocator provided by your system.
--disable-ssp	This options disables use of the GCC stack protector.
--enable-suid	With this option, samhain will honour the SUID bit instead of resetting all privileges to the real UID of the process. Required

for Nagios if samhain is invoked by the Nagios plugin itself, unless you want to use sudo instead.

--disable-shellexpand This options disables the expansion of shell commands in the configuration file.

--disable-external-scripts This options disables the possibility to call an external program to log an event (or perform active response).

--enable-message-queue[=MODE] This options enables logging to a POSIX message queue, where MODE should be the octal permission for the queue (default is 0700).

2. Optional modules to perform additional checks

These are all client-only options, as the server does not perform any checks (if you want to run checks on the log server host, you need to run a client there as well).

--enable-login-watch [CLIENT ONLY] Compile in the module to watch for login/logout events.

--enable-mounts-check [CLIENT ONLY] Compile in the module to check for correct mount options.

--enable-userfiles [CLIENT ONLY] Compile in the module to check for files in user home directories (i.e. with paths relative to $HOME for all users).

--enable-suidcheck [CLIENT ONLY] Compile in the module to check file system for SUID/SGID binaries not in the database.

3. OpenPGP Signatures on Configuration/Database Files

--with-gpg=PATH Use GnuPG to verify database/configuration file. The public key of the effective user, usually root, (in `~/.gnupg/pubring.gpg`) will be used.

--with-keyid=0x<hex KeyID> This optional argument allows to specify a key ID, if there is more than one key in your keyring. This is only used for the installation routine, and for configuring the **samhainadmin.pl** convenience script.

--with-checksum=CHECKSUM Compile in TIGER checksum of the **gpg** binary. CHECKSUM must be the full line output by samhain or **gpg** when computing the checksum.

--with-fp=FINGERPRINT Compile in the fingerprint of the key used to sign the configuration/database file. If used, samhain will verify the fingerprint, but still report on the used public key.

4. OpenBSD Signify Signatures on Configuration/Database Files

--with-signify=PATH Use signify (signify-openbsd on Debian flavour Linuxes) to verify database/configuration file. The public key of the ef-

	fective user, usually root, (in `~/.signify/install_name.pub`) will be used (where *install_name* is the name under which samhain is installed).
--with-checksum=CHECKSUM	Compile in TIGER192 checksum of the **signify** binary. CHECKSUM can be determined either using samhain -H or **gpg --load-extension tiger --print-md TIGER192**
--with-pubkey-checksum=CHECK-SUM	Compile in TIGER192 checksum of the `~/.signify/install_name.pub` public key used to verify the signature of signed database/configuration files. CHECKSUM can be determined either using samhain -H or **gpg --load-extension tiger --print-md TIGER192**

5. Client/Server Connectivity

--enable-network=client/server	Compile a client or server, rather than a standalone version.
--disable-encrypt	Disable encryption for client/server communication.
--disable-srp	Disable the use of the zero-knowledge SRP protocol to authenticate to log server, and use a (faster, but less secure) challenge-response protocol. *This must be set to the same value for client and server, i.e. either disabled for client and server, or for none of both.*
--with-libwrap[=PATH]	[SERVER ONLY] Build the server with support for libwrap (Wietse Venema's TCP wrappers library). In `/etc/hosts.allow` and/or `/etc/hosts.deny`, use 'yule' or the name defined with --enable-install-name=NAME for the name of the daemon.
--with-port=PORT	The port on which the server will listen (default is 49777), or to which the client will connect, respectively. *This must be set to the same value for client and server.* Only needed if this port is already used by some other application. Port numbers below 1024 require *root* privileges for the server.
--with-logserver=HOST	[CLIENT ONLY] The host address of the log server. This can be set in the configuration file. A compiled-in address is only required if you want to fetch the configuration file from the log server. An address in the configuration file will take precedence.
--with-altlogserver=HOST	[CLIENT ONLY] The host address of an alternative (backup) log server.

6. Paths

Compiled-in paths may be as long as 255 chars. If the **--enable-(micro-)stealth** option is used, the limit is 127 chars. The paths to the database, log file, and pid/lock file can be overridden in the configuration file (see Section 1 (p. 117)).

Tip

If using NFS with clients on different hosts accesing the same files, you can set the database, log file, and pid/lock file names to "AUTO" in the configuration file to simply tack on the hostname on the compiled-in path. The same length limits apply.

--prefix=PREFIX	The install prefix. Default is none, and using the Filesystem Hierarchy Standard 2.2 directory layout. If you prefer the GNU layout (everything under /usr/local), use --prefix=/usr/local. See Section 10 (p. 9) for details.
--sbindir=DIR	The binary directory (default is `/usr/local/sbin`)
--localstatedir=DPFX	The state data directory prefix (default is `/var`). Data will be written to DPFX/lib/ *install_name*.
--with-state-dir=DIR	The state data directory (default is `DPFX/lib/install_name`). Data will be written to this directory.
--mandir=MPREFIX	The man directory directory prefix (default is `/usr/local/share/man`).
--with-tmp-dir=TPFX	The directory where tmp files are created (config/database downloads from server, extracted PGP-signed parts of config/database files) (default is *HOME*).
--with-config-file=FILE	The full path of the configuration file (default is `/etc/(install_name)rc`).
--with-log-file=FILE	The path of the log file (default is `DPFX/log/samhain_log`).
--with-pid-file=FILE	The path of the PID file (default is `DPFX/run/(install_name).pid`).
--with-html-file=FILE	[SERVER ONLY] The path of the HTML status file where the current status of clients is displayed (default is `DPFX/log/(install_name).html`).
--with-console=PATH	The path of the console (default is `/dev/console`). This may be a FIFO.
--with-altconsole=PATH	The path of a second console (default is none). This may be a FIFO. If defined, console output will always go to both console devices (but note that console devices are only used when running as daemon).

Appendix B. List of command line options

1. General

1. *-D, --daemon* Run as daemon.

2. *--foreground* Stay in the foreground, do not run as daemon.

3. *-f, --forever* Loop forever, even if not daemon.

4. *--bind-address=<IP-Address>* Use this IP address (i.e. interface) for outgoing connections (e.g. on multi-interface machines).

5. *--server-port=<port number>* Connect to this port on the server (client-side option for client-server connection).

6. *--server-host=<hostname/IP address>* Connect to this server host (client-side option for client-server connection). Using this option a second time will set the backup server host. *Reloading* the configuration with a server defined in the configuration file will overwrite the original CL values.

7. *-s <arg>, --set-syslog-severity=<arg>* Set the severity threshold for syslog. *arg* may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

8. *-l <arg>, --set-log-severity=<arg>* Set the severity threshold for logfile. *arg* may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

9. *-m <arg>, --set-mail-severity=<arg>* Set the severity threshold for e-mail. *arg* may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

10. *--set-database-severity=<arg>* Set the severity threshold for logging to a RDBMS. *arg* may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

11. *--set-prelude-severity=<arg>* Set the severity threshold for logging to the Prelude IDS system. *arg* may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

12. *-p <arg>, --set-print-severity=<arg>* Set the severity threshold for terminal/console. *arg* may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

13. *-x <arg>, --set-extern-severity=<arg>* Set the severity threshold for external program(s). *arg* may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

14. *-L <arg>, --verify-log=<arg>* Verify the integrity of the log file and print the entries (*arg* is the path of the log file).

15. *-j, --just-list* Modify -L to just list the logfile, rather than verify (to de-obfuscate the logfile if you have compiled for stealth mode). *Order matters:* this must come before -L.

16. *-M <arg>, --verify-mail=<arg>* Verify the integrity of e-mailed messages (*arg* is the path of the mail box).

17. *-V <arg>, --add-key=<arg>* Add key material to the compiled-in key (see Section 2 (p. 106)). *arg* must be of the form *key@/path/to/executable*. Output will be written to `/path/to/exe-cutable.out`.

18. *-H <arg>, --hash-string=<arg>* Print the hash of a string / the checksum of a file, and exit. If *arg* starts with a '/', it is assumed to be a file, otherwise a string. This function is useful to test the hash algorithm.

19. *-z <arg>, --tracelevel=<arg>* If compiled with --enable-debug: arg > 0 to switch on debug output. If compiled with --enable-trace: arg > 0 max. level for call tracing.

20. *-i <arg>, --milestone=<arg>* If compiled with --enable-trace: trace from milestone arg to arg+1. If arg = -1, trace all.

21. *-o <arg>, --outfile=<filename>* Pathname for the output filename (relevant for the options --list-database and --create-database).

22. *-d <arg>, --list-database=<arg>* List the database file *arg*(use ``default'' for the compiled-in path). The listing can be modified in various ways by prepending some other options (see next).

23. *--list-file=<path>* Modify --list-database to list the literal content of a file, if this has been stored. *Order matters:* this must come before --list-database.

24. *-a, --full-detail* Modify --list-database to list full details (numeric mode, owner, group, all three timestamps (ctime, mtime, atime), and the checksum. *Order matters:* this must come before --list-database.

25. *--delimited* Same as --full-detail, but with *comma-delimited* fields. *Order matters:* this must come before --list-database.

26. *--binary* This will list the database in the binary format of the database, i.e. it will output *another database*. Intended to be used together with --list-filter to create a partial database. *Order matters:* this must come before --list-database.

27. *--list-filter=<filename>* Modify --list-database to list only those files listed in the text file <filename> (one path per line). Together with the --binary option this allows to create a partial database. *Order matters:* this must come before --list-database.

28. *--verify-database=<arg>* Verify (the integrity of the files in) the database given as argument, and exit with a status of success or failure. The configuration file will *NOT* be read, the policy recorded in the baseline database will be used instead. For more verbosity, the option *--set-print-severity=<arg>>* can be prepended. Intended to be used with a partial database created by *--binary --list-filter=<filter_filename> --list-database=<database_filename>*

29. *--create-database=<arg>* Create a baseline database for the files listed in the textfile *arg*(one path per line). The configuration file will *NOT* be read, the policy recorded in the baseline database will be ReadOnly, and the content of files will optionally be stored if the path in the list is preceded by a '+' (plus) sign. For choosing the output pathname, the option --outfile=<filename> should be prepended. For more verbosity, the option *--set-print-severity=<arg>* can be prepended.

30. *--init-rootfs=<arg>* Use the directory given as *arg* as the root of the filesystem when initializing the database. Paths in the configuration file are considered to be relative to this directory, i.e. the leading '/' represents the specified filesystem root rather than the true root.

    The purpose of this option is to faciliate the generation of a baseline database for a root filesystem (e.g. of an embedded/IOT device) mounted at some path.

31. *-c, --copyright* Print copyright information and exit.

32. *-v, --version* Show version information and compiled-in options.

33. *-h, --help* Print a short help on command line options and exit.

34. *--trace-enable* Print a trace of the execution flow.

35. *--trace-logfile=<arg>* Use file *arg* to log the trace.

2. samhain

1. *-t <arg>, --set-checksum-test=<arg>* Set file checking to *init*, *update*, or *check*. Use *init* to create the database, *update* to update it, and *check* to check files against the database.

> **Tip**
>
> Yes, it is normal that *update* takes much more time than *init*.

2. *-i, --interactive* Use interactive mode for *update*(ask before updating an entry).

3. *--listfile=<path>* Use a text file comprising a list of files to *update*(one file per line).

4. *-e <arg>, --set-export-severity=<arg>* Set the severity threshold for forwarding messages to the log server. *arg* may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

5. *-r <arg>, --recursion=<arg>* Set the default recursion level for directories (0 -- 99).

6. *--init2stdout* Write the database to stdout when performing the initialization.

3. yule

1. *-S, --server* Run as server. Only required if the binary is dual-purpose.

2. *-q, --qualified* Log received messages with the fully qualified name of client host.

3. *--chroot=<arg>* Chroot to to the directory *arg*(should be an absolute path.

4. *-G, --gen-password* Generate a random password suitable for use in the following option (16 hexadecimal digits).

5. *-P <arg>, --password=<arg>* Compute a client registry entry. *arg* is the chosen password (16 hexadecimal digits).

Appendix C. Configuration file syntax and options

1. General

The configuration file for samhain is named `samhainrc` by default. Also by default, it is placed in `/etc`. (Name and location is configurable at compile time). The distribution package comes with a commented sample configuration file.

This section introduces the general structure of the configuration file. Details on individual entries in the configuration files are discussed in Section 4 (p. 38) (which files to monitor), Section 1 (p. 19) (what should be logged, which logging facilities should be used, and how these facilities are properly configured), and Section 11 (p. 53) (monitoring login/logout events).

The configuration file contains several *sections*, indicated by *headings* in *square brackets*(e.g. **[Database]**). Sections exist to group related directives and avoid eventual name clashes among options. Any particular section may occur multiple times.

Each section may hold zero or more **key=** *value* pairs. Keys are not case sensitive, and space around the '=' is allowed, as well as before the key and after the value. More specifically: the line is processed by splitting into key and value at the first '=', trimming whitespace from the beginning and end of both key and value, and converting the key to lowercase.

Blank lines and lines starting with '#' are comments. Everything before the first section and after an **[EOF]** is ignored. The **[EOF]** end-of-file marker is optional. The file thus looks like:

```
# this is a comment
[Section heading]
key1=value
key2=value
[Another section]
key3=value
key4=value
```

For boolean values the following are equivalent (case-insensitive): True, Yes, or 1. Likewise, the following are equivalent (case-insensitive): False, No, or 0.

In lists, values can be separated by space, tabs, or commas.

Tip

Each section may occur multiple times.

Note

You can explicitly end the configuration file with an **[EOF]** (on a separate line), but this is not required, unless there is some junk beyond that may confuse the parser. A PGP signature does *not* qualify as 'junk' if samhain is compiled to verify the signature.

1.1. Shell expansion

As of version 2.5.3, it is possible to use shell expansion to define the value of an option. For any configuration file option written as **Key = $(** *shell_command***)** , the string contained within the

$() will be passed literally to the shell (by invoking **/bin/sh -c `shell_command`**), and the *first* line returned by the shell - after stripping the newline char - will replace the $(..). If there is no output within 120 seconds, samhain will ignore the configuration option (and report an error).

Note

You cannot define just *part* of an option value this way. You need to write the shell expression such that it covers the *whole* option value (e.g. by including an 'echo -n foobar').

The PATH environment variable will be set to "/sbin:/bin:/usr/sbin:/usr/bin:/usr/ucb", the SHELL variable to "/bin/sh", the IFS variable to " \t\n", and the TZ variable will be copied from the startup environment. *No other environment variables will be set*.

In case you are unsure about the need for escaping: yes, the whole string will be passed as a single argument to the shell, like calling **/bin/sh -c ' `shell_command`'** from the shell, BUT since this is done from within a C program rather than from a shell, there are no single quotes surrounding the whole string.

In the following example, we parse the output of **ifconfig** to supply a list of all interfaces to the "PortCheckInterface" option.

```
#
# Lines broken for display purposes. Must be ONE line in config file!!!

# Linux/Solaris, FreeBSD, OpenBSD

$Linux:.*:.*
PortCheckInterface=$( /sbin/ifconfig | grep 'inet addr:' |
    sed 's/.*r:\([0-9.]*\).*/\1 /' | tr -d '\n'; echo )
$end

# Solaris, FreeBSD, OpenBSD

$(SunOS|FreeBSD|OpenBSD):.*:.*
PortCheckInterface = $( /sbin/ifconfig -a| grep 'inet ' |
    sed 's/.*t \([0-9.]*\) .*/\1 /' | tr -d '\n';echo )
$end
```

1.2. Conditionals

Conditional inclusion of entries for some host(s) is supported via any number of @if.. / @else / @fi directives. @if.., @else, and @fi must each be on separate lines. Configuration options in the @if.. (or the optional @else) branch will be read or ignored depending on the result of the test.

Supported tests are as follows:

hostname_matches	**@if hostname_matches `regex`** will succeed if the hostname matches the regular expression given.
system_matches	**@if system_matches `regex`** will succeed if the string sysname:release:machine — i.e. $(uname -s):$(uname -r):$uname - m) — matches the regular expression given.
file_exists	**@if file_exists `path`** will succeed if a file with the given absolute path exists. Wildcards/regular expression are not supported.
interface_exists	**@if interface_exists `address`** will succeed if a network interface with the given address exists.

command_succeeds   **@if command_succeeds** `command` will execute **/bin/sh -c** `command` and succeed if the exit status is zero. The PATH environment variable will be set to "/sbin:/bin:/usr/sbin:/usr/bin:/usr/ucb", the SHELL variable to "/bin/sh", the IFS variable to " \t\n", and the TZ variable will be copied from the startup environment. *No other environment variables will be set*.

You can negate a test by saying '@if not ..'. The 'not' may be replaced by a '!'. The following are all valid: '@if not file_exists /etc/motd', '@if !file_exists /etc/motd', and '@if ! file_exists /etc/motd'.

Note on backward compatibility

For backward compatibility, instead of **@if hostname_matches** `hostname` you can also say **@**`hostname` .

Likewise, instead of **@if system_matches** `sysname:release:machine` you can also say **$**`sysname:release:machine` .

Also, the old method of negating by prepending a '!' to the '@' ('$') is still supported, as well as the use of '@end' ('$end') instead of '@fi'.

```
@if hostname_matches foobar
# only read if hostname is 'foobar'
@else
# read if hostname is NOT 'foobar'
@fi

@if not hostname_matches foobar
# not read if hostname is 'foobar'
@fi

@if system_matches Linux:2.6.24-21-generic:i686
# only read if $(uname -s):$(uname -r):$(uname -m)
#    matches Linux:2.6.24-21-generic:i686
@fi

@if !system_matches Linux:2.6.24-21-generic:i686
# not read if $(uname -s):$(uname -r):$(uname -m)
#    matches Linux:2.6.24-21-generic:i686
@fi
```

2. Files to check

Allowed section headings (see Section 4.1 (p. 39) for more details) are:

[Attributes] , **[LogFiles]** , **[GrowingLogFiles]** , **[IgnoreAll]** , **[IgnoreNone]** , **[ReadOnly]** , **[User0]** , **[User1]** , and **[User2]** , and **[User3]** , and **[User4]** , and **[Prelink]**

Placing an entry under one of these headings will select the respective policy for that entry (see Section 4.1 (p. 39)). Entries under the above section headings must be of the form:

dir=`[optional numerical recursion depth]path`

file=`path`

3. Severity of events

Section heading (see Section 1.1 (p. 19) for more details):

[EventSeverity]

Entries:

SeverityReadOnly= *severity*

SeverityLogFiles= *severity*

SeverityGrowingLogs= *severity*

SeverityIgnoreNone= *severity*

SeverityIgnoreAll= *severity*

SeverityAttributes= *severity*

SeverityUser0= *severity*

SeverityUser1= *severity*

SeverityUser2= *severity*

SeverityUser3= *severity*

SeverityUser4= *severity*

SeverityPrelink= *severity*

SeverityFiles= *severity*

SeverityDirs= *severity*

SeverityNames= *severity*

severity may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

4. Logging thresholds

Section heading (see Section 3 (p. 22) for more details):

[Log]

Entries:

MailSeverity= *list of [optional specifier]threshold*

PrintSeverity= *list of [optional specifier]threshold*

LogSeverity= *list of [optional specifier]threshold*

SyslogSeverity= *list of [optional specifier]threshold*

PreludeSeverity= *list of [optional specifier]threshold*

ExportSeverity= *list of [optional specifier]threshold*

ExternalSeverity= *list of [optional specifier]threshold*

DatabaseSeverity= *list of [optional specifier]threshold*

threshold may be one of **none, debug, info, notice, warn, mark, err, crit, alert** .

The optional specifier may be one of '*', '!', or '=', which are interpreted as 'all', 'excluding', and 'only', respectively. Examples: specifying '*' is equal to specify 'debug'; specifying '!*' is equal to specifying 'none'; 'info,!alert' is the range from 'info' to 'crit'; and 'info,!=mark' is info and above, but excluding 'mark'.

5. Watching login/logout events

Section heading:

[Utmp]

Entries:

LoginCheckActive= *boolean* — '1' to switch on, '0' to switch off.

LoginCheckInterval= *seconds* — Interval between checks.

SeverityLogin= *severity* — Severity for login events.

SeverityLoginMulti= *severity* — Severity for multiple logins by same user.

SeverityLogout= *severity* — Severity for logout events.

6. Checking for SUID/SGID files

Section heading:

[SuidCheck]

Entries:

SuidCheckActive= *boolean* — '1' to switch on, '0' to switch off.

SuidCheckExclude= *path* — A directory (and its subdirectories) to exclude from the check. Only one directory can be specified this way.

SuidCheckSchedule= *schedule* — Crontab-like schedule for checks.

SeveritySuidCheck= *severity* — Severity for events.

SuidCheckFps= *fps* — Limit files per seconds for SUID check.

SuidCheckNosuid= *boolean* — Check filesystems mounted as nosuid. Defaults to not.

SuidCheckQuarantineFiles= *boolean* — Whether to quarantine files. Defaults to not.

SuidCheckQuarantineMethod= *0/1/2* — Quarantine method. Delete = 1, remove suid/sgid flags = 1, move to quarantine directory = 2. Defaults to 1 (remove suid/sgid flags).

SuidCheckQuarantineDelete= *boolean* — Whether to delete rather than truncate, if method 0 (delete) is chosen. Default is truncate.

7. Checking for mount options

Section heading:

[Mounts]

Entries:

MountCheckActive= *`boolean`* — 'yes' to switch on, 'no' to switch off (default = no).

MountCheckInterval= *`seconds`* — Interval between checks (default one day).

SeverityMountMissing= *`severity`* — Severity for missing mounts (default = err).

SeverityOptionMissing= *`severity`* — Severity for missing mount options (default = err).

CheckMount= *`/path [mount options]`* — Mount point to check. Mount options must be given as comma-separated list, separated by a blank from the preceding mount point.

8. Checking for user files

Section heading:

[UserFiles]

Entries:

UserfilesActive= *`boolean`* — '1' to switch on, '0' to switch off.

UserfilesName= *`filename policy`* — Files to check for under each $HOME. Allowed values for 'policy' are: allignore, attributes, logfiles, loggrow, noignore (default), readonly, user0, user1, user2, user3, and user4.

UserfilesCheckUids= *`uid  list`* — A list of UIDs where we want to check. The default is all. Ranges (e.g. 100-500) are allowed. If there is an open range (e.g. 1000-), it must be last in the list.

9. Checking for hidden/fake/required processes

Section heading:

[ProcessCheck]

Entries:

ProcessCheckActive= *`boolean`* — 'true' to switch on, 'false' to switch off.

SeverityProcessCheck= *`severity`* — Severity for events (default is crit).

ProcessCheckMinPID= *`integer`* — Minimum PID (default is 0).

ProcessCheckMaxPID= *`integer`* — Maximum PID (default is 32767).

ProcessCheckInterval= *`seconds`* — Interval between checks.

ProcessCheckExists= *`POSIX regular expression`* — A process that is required to run. Must match a substring in a line of the 'ps' output.

ProcessCheckPSPath= *`path`* — The path to ps (default: autodetected at compile time).

ProcessCheckPSArg= *`path`* — The argument to ps (default: autodetected at compile time). Note that the first column must be the PID, except on Linux, where the format 'PID SPID ...' is expected (spid = thread id), as shown by 'ps -eT'.

10. Checking for open ports

Section heading:

[PortCheck]

Entries:

PortCheckActive= *boolean* — 'true' to switch on, 'false' to switch off.

SeverityPortCheck= *severity* — Severity for events (default is crit).

PortCheckRequired= *interface:portlist* — Services (open ports) that are required.

PortCheckOptional= *interface:portlist* — Services (open ports) that are optional (allowed, but not required).

PortCheckIgnore= *interface:portlist* — Services (open ports) that should be ignored (no reports for this port).

PortCheckInterface= *(list of) IP address(es)* — Additional interface to scan (up to 63 interfaces).

PortCheckDevice= *(list of) device(s)* — Device(s) to scan, regardless of address(es) assigned to them (each address counts towards the maximum of 63 interfaces for PortCheckInterface). Devices will be re-checked before scan to account for address changes.

PortCheckInterval= *seconds* — Interval between checks (default 300).

PortCheckUDP= *boolean* — Whether to scan UDP ports as well (default yes).

PortCheckMinPort= *integer* — The lowest port to be checked (defaults to 0).

PortCheckMaxPort= *integer* — The highest port to be checked (defaults to 65535).

11. Logfile monitoring/analysis

Section heading:

[Logmon]

LogmonActive= *boolean* — 'true' to switch on, 'false' to switch off.

LogmonSaveDir= */abslute/path* sets the directory where checkpoint data for logfiles is stored (default: same as for pid file).

LogmonClean= *boolean* delete old checkpoint data unmodified for 30 days or more (default: off).

LogmonInterval= *seconds* — Interval between checks (default 10).

LogmonWatch= *TYPE:path[:format]* — File to monitor.

LogmonHidePID= *boolean* — Suppress PID in syslog messages, 'true' to switch on, 'false' to switch off.is an option

LogmonMarkSeverity= *severity* — Severity for reports on missing heartbeat messages if the messages themselves are assigned to the 'trash' queue (default: crit).

LogmonBurstThreshold= *number* — The number of repeated messages within 12 minutes that must be exceeded to report a burst of repeated messages (default: 24).

LogmonBurstQueue= *queue* — Set the reporting queue for reporting bursts of similar log messages (default: don't report).

LogmonBurstCron= *boolean* — Whether to report also on bursts of repeated cron messages (defaul: false).

LogmonDeadtime= *seconds* — Do not report a correlated event again within the given time (default: 60 seconds).

LogmonQueue= *label:[interval]:(sum|report):severity[:alias]* — defines an output queue.

LogmonHost= *(perl)regex* — Causes the following rules to be applied only to entries for this host(s).

LogmonEndHost — Explicitly ends a preceding LogmonHost directive.

LogmonGroup= *(perl)regex* — Causes the following rules to be applied only if the group regex matches.

LogmonEndGroup — Explicitly ends a preceding LogmonGroup directive.

LogmonRule= *queue_label:(perl)regex* — matches a logfile entry against the provided regular expression.

12. Database

Section heading:

[Database]

Entries:

SetDBHost= *db_host* — Host where the DB server runs (default: localhost). Should be numeric IP address for PostgreSQL.

SetDBName= *db_name* — Name of the database (default: samhain).

SetDBTable= *db_table* — Name of the database table (default: log).

SetDBUser= *db_user* — Connect as this user (default: samhain).

SetDBPassword= *db_password* — Use this password (default: none).

SetDBServerTstamp= *boolean* — Log server timestamp for client messages (default: true).

UsePersistent= *boolean* — Use a persistent connection (default: true).

AddToDBHash= *field* — Add a database field to the set of fields that are used for tagging the log record with an MD5 hash.

13. Miscellaneous

Section heading:

[Misc]

Entries:

Daemon= *boolean* — Whether to become a daemon (default: no)

MessageHeader= *"%S %T "* — Specify custom format for message header. The following placeholders are supported: %S for the message severity, %T for the timestamp, %C for the message class, %F for the source file, %L for the source line number, and %E for the status (might provide additional information in case of internal errors).

VersionString= *string* — Set version string to include in file signature database (along with hostname and date).

SetReverseLookup= *boolean* — If false, skip reverse lookups when connecting to a host known by name rather than IP address.

AvoidBlock= *boolean* — Run stat/lstat system calls in a subprocess to avoid that a flaky NFS mount blocks the process (defaults to off for the server, on for the client/standalone executable, except off for Cygwin/Windows).

HideSetup= *boolean* — Don't log names of config/database files on startup.

SyslogFacility= *LOG_xxx* — Set syslog facility (default is LOG_AUTHPRIV).

SyslogMapStampTo= *LOG_xxx* — Set syslog priority for heartbeat messages (timestamps). Default is LOG_ERR.

MACType= *HASH-TIGER/HMAC-TIGER* — Set type of message authentication code (HMAC). Must be identical on client and server.

SetLoopTime= *seconds* — Interval between timestamp messages (60).

SetConsole= *device* — Set the console device (/dev/console).

SetReportFile= *path* — Set the path for file check reports (none). Can be an absolute path or 'none' to disable. Format is lines comprised of a timestamp string followed by number of seconds since the Epoch followed by six integers: bytes hashed, dirs checked, files checked, files reported, errors, files that should be but aren't directories.

SetReportGroup= *group* — Set the unix group (numeric or name, defaults to 0) for the file check reports.

SetSigtrapMaxDuration= *microseconds* — This directive allows to configure the timeout for handling the sigtrap signal in the antidebug code (enabled with the **--enable-ptrace** configure option) (500000, equal to 500ms). Set to a higher value if the antidebug handler is triggered under high load. Note that for security, you can set this value only once while the daemon runs.

MessageQueueActive= *boolean* — Use SysV IPC message queue (false).

PreludeMapToInfo= *list of samhain severities* — The severities that should be mapped to impact severity 'info' in prelude reports (default: none). This option is only available with libprelude 0.9.

PreludeMapToLow= *list of samhain severities* — The severities that should be mapped to impact severity 'low' in prelude reports (default: none). This option is only available with libprelude 0.9.

PreludeMapToMedium= *list of samhain severities* — The severities that should be mapped to impact severity 'medium' in prelude reports (default: none). This option is only available with libprelude 0.9.

PreludeMapToHigh= *list of samhain severities* — The severities that should be mapped to impact severity 'high' in prelude reports (default: none). This option is only available with libprelude 0.9.

PreludeProfile= *profile* — Set the profile (sensor name) for use with the Prelude IDS. This option is only available with libprelude 0.9. Default is 'samhain' (prelude 0.9) or 'Samhain' (prelude 0.8).

SetMailAddress= *recepient* — Add a recepient e-mail address.

SetMailAlias= *listname: username@hostname* — Add a list of recepient e-mail address.

SetAddrSeverity= `severity` — Defines a severity threshold for an individual recipient (list). Must be a subset of the global MailSeverity setting. Applies to the last defined recipient (list).

SetMailFilterAnd= `list` — Defines a list of strings all of which must match a message, otherwise it will not be mailed. Applies to the last defined recipient (list).

SetMailFilterOr= `list` — Defines a list of strings at least one of which must match a message, otherwise it will not be mailed. Applies to the last defined recipient (list).

SetMailFilterNot= `list` — Defines a list of strings none of which should match a message, otherwise it will not be mailed. Applies to the last defined recipient (list).

CloseAddress — Explicitly closes the definition of a recipient (list).

SetMailTime= `seconds` — Maximum time interval between mail messages (86400 sec).

SetMailNum= `0 -- 16383` — Maximum number of pending mails on internal queue (10).

SetMailRelay= `IP address` — The mail relay (for offsite mail; default: none).

MailSubject= `string` — Custom format for the email subject (none).

SetMailSender= `string` — Sender for the 'From:' field.

SetMailPort= `port number` — Port number to use for SMTP (default: 25).

SamhainPath= `path` — The path of the process image.

SetBindAddress= `IP address` — The IP address (i.e. interface on multi-interface box) to use for outgoing connections (e.g. e-mail).

SetTimeServer= `IP address` — The time server. Note that the simple 'time' service (port 37/tcp) is used.

TrustedUser= `username(,username,..)` . — List of additional trusted users.

SetLogfilePath= `AUTO or /path` — Path to log file (AUTO to tack hostname on compiled-in path).

SetLockfilePath= `AUTO or /path` — Path to lock file (AUTO to tack hostname on compiled-in path).

The following options are only relevant for standalone or client executables:

SetNiceLevel= `-19..19` — Set scheduling priority during file check. — (see 'man nice').

SetIOLimit= `bps` — Set IO limits (kilobytes per second) for file check.

SetDropCache= `boolean` — Drop checksummed files from cache (unless they were cached before). Defaults to false for performance reasons.

ReportCheckflags= `boolean` — Report checking policy (check flags) for new files, and if changed also for changed files (defaults to no). Added in version 4.0.

StartupLoadDelay= `seconds` — At startup, delay the download of the baseline databse from the server for the given time span (default is no delay).

SetDeltaRetryCount= `integer` — The number of times the client will retry to download a delta database from the server after the initial attempt has failed (default is 0, i.e. do not retry).

SetDeltaRetryInterval= `seconds` — The interval between successive tries to download a delta database (default is 60 seconds).

SetFilecheckTime= `seconds` — Interval between file checks (600).

FileCheckScheduleOne= `schedule` — Crontab-like schedule for file checks.

UseRsrcCheck= `boolean` — Check the ..namedfork/rsrc file on Mac OS X (defaults to no since this mechanism is deprecated by Apple).

UseHardlinkCheck= `boolean` — Compare number of hardlinks to number of subdirectories for directories.

HardlinkOffset= `N: /path` — Exception (use multiple times for multiple exceptions). N is offset (actual - expected hardlinks) for `/path`.

AddOKChars= `N1, N2, ..` — List of acceptable characters (byte value(s)) for the check for weird filenames. Nn may be hex (leading '0x': 0xNN), octal (leading zero: 0NNN), or decimal. Use 'all' for all.

FilenamesAreUTF8= `boolean` — If set, samhain will check for invalid UTF-8 encoding and for filenames ending in invisible characters.

IgnoreAdded= `path_regex` — Ignore if this file/directory is added/created. *The path_regex argument has to start with a forward slash and has to match the full path..*

IgnoreMissing= `path_regex` — Ignore if this file/directory is missing/deleted. *the path_regex argument has to start with a forward slash and has to match the full path.*

IgnoreModified= `path_regex` — Ignore if this file/directory is modified (3.0.11+, useful for transient files that get modified during their lifetime). *the path_regex argument has to start with a forward slash and has to match the full path.*

LooseDirCheck= `boolean` — Ignore changes of directory inodes if nothing but size and timestamps have changed.

SetAuditdFlags= `r/w/x/a` — Set the flags on which audit rules will trigger (defaults to wa [=write| change attributes]).

SkipChecksum= `list of conditions` — Skip checksumming if the list of condition holds true

FileType= `definition` — User-defined file type specification (to be used for the **SkipChecksum=** `...` command).

ReportOnlyOnce= `boolean` — Report only once on a modified file (yes).

ReportFullDetail= `boolean` — Report in full detail on modified files (no).

UseLocalTime= `boolean` — Report file timestamps in local time rather than GMT (no). Do not use this with Beltane.

ChecksumTest= `none/init/update/check` — The default action (default is none).

SetPrelinkPath= `path` — The path to the prelink binary (default is `/usr/sbin/prelink`).

SetPrelinkChecksum= `checksum` — The checksum of the prelink binary.

SetLogServer= `IP address` — The log server.

SetServerPort= `port number` — The port on the log server (defaults to the compiled-in port, which is 49777 unless redefined at compile time).

SetThrottle= `milliseconds` — An option to throttle the network throughput when downloading the database from the server. The allowed maximum of 1000 msec throttles to about 64 kB/sec, less is faster.

SetDatabasePath= `AUTO or /path` — Path to database (AUTO to tack hostname on compiled-in path).

DigestAlgo= `TIGER192/SHA1/MD5/SHA256` — Use SHA1, MD5, or SHA2-256 instead of the TIGER checksum (default: TIGER192).

RedefReadOnly= `+XXX or -XXX` — Add or subtract test XXX from the ReadOnly policy.

RedefAttributes= `+XXX or -XXX` — Add or subtract test XXX from the Attributes policy.

RedefLogFiles= `+XXX or -XXX` — Add or subtract test XXX from the LogFiles policy.

RedefGrowingLogFiles= `-XXX or ~XXX` — Add or subtract test XXX from the GrowingLogFiles policy.

RedefIgnoreAll= `+XXX or -XXX` — Add or subtract test XXX from the IgnoreAll policy.

RedefIgnoreNone= `+XXX or -XXX` — Add or subtract test XXX from the IgnoreNone policy.

RedefUser0= `+XXX or -XXX` — Add or subtract test XXX from the User0 policy.

RedefUser1= `+XXX or -XXX` — Add or subtract test XXX from the User1 policy.

RedefUser2= `+XXX or -XXX` — Add or subtract test XXX from the User2 policy.

RedefUser3= `+XXX or -XXX` — Add or subtract test XXX from the User3 policy.

RedefUser4= `+XXX or -XXX` — Add or subtract test XXX from the User4 policy.

UseAttributesCheck= `boolean` — Check file attributes on linux file systems (default=yes).

UseACLCheck= `boolean` — Check ACL policies for files.

UseSelinuxCheck= `boolean` — Check SELINUX attributes for files.

SetFullSilent= `boolean` — Also suppress informational messages during silent file scan triggered by SIGTSTP.

The following options are only relevant for the server:

SetUseSocket= `boolean` — If unset, do not open the command socket (server only). This socket allows to advise the server to transmit commands to clients as soon as they connect to the server next time.

SetSocketAllowUid= `UID` — Which user can connect to the command socket. The default is 0 (root).

SetSocketPassword= `password` — Password (max. 14 chars, no '@') for password-based authentication on the command socket (only if the OS does not support passing credentials via sockets).

SetChrootDir= `path` — If set, chroot to this directory (server only).

SetStripDomain= `boolean` — Whether to strip the domain from the client hostname when logging client messages (server only; default: yes).

SetClientFromAccept= `boolean` — If true, use client address as known to the communication layer. Else (default) use client name as claimed by the client, try to verify against the address known to the communication layer, and accept (with a warning message) even if this fails.

UseClientSeverity= `boolean` — If set to 'yes', don't assign a special severity (priority) to client messages.

UseClientClass= `boolean` — If set to 'yes', don't assign a special class to client messages.

SetServerPort= *`port number`* — The port that the server should use for listening (default is 49777).

SetServerInterface= *`IP address`* — The IP address (i.e. interface on multi-interface box) that the server should use for listening (default is all). Use INADDR_ANY to reset to all.

SeverityLookup= *`severity`* — Severity for name lookup errors when verifying (on the server side) that the socket peer matches the hostname claimed by the client. See the preceding option.

UseSeparateLogs= *`boolean`* — If true, messages from different clients will be logged to separate log files (the name of the client will be appended to the name of the main log file to construct the logfile name). Default: false.

SetClientTimeLimit= *`seconds`* — Maximum time limit until next client message (server-only). If no message is received from a client within that limit, the respective client will be reported as dead.

SetConnectionTimeout= *`seconds`* — Timeout after which a currently active connection to a client will be closed by the server (900 seconds). This timeout has the purpose to prevent bad clients from hogging server resources.

SetUDPActive= *`boolean`* — yule 1.2.8+: Listen on 514/udp (syslog). Default: false.

Remarks: (i) **root** and the effective user are always trusted. (ii) If no time server is given, the local host clock is used. (iii) If the path of the process image is given, the process image will be checksummed at startup and exit, and both checksums compared.

14. External

Definition of an arbitrary number of external programs/scripts (see Chapter 7 (p. 83)). Section heading:

[External]

Entries:

OpenCommand= *`/full/path/to/program`* — Starts new command definition.

CloseCommand — Ends new command definition (optional syntactic sugar).

SetType= *`log/srv`* — Type/purpose of the program.

SetCommandline= *`list`* — The command line.

SetEnviron= *`KEY=value`* — Environment variable (can be repeated).

SetChecksum= *`TIGER checksum`* — Checksum of the program.

SetCredentials= *`username`* — User whose credentials shall be used.

SetFilterNot= *`list`* — Regular expression patterns not allowed in message.

SetFilterAnd= *`list`* — Regular expression patterns required (ALL) in message.

SetFilterOr= *`list`* — Regular expression patterns required (at least one) in message.

SetDeadtime= *`seconds`* — Deadtime between consecutive calls.

SetDefault= *`boolean`* — Set default environment (HOME from /etc/passwd, SHELL=/bin/sh, PATH=/sbin:/usr/sbin:/bin:/usr/bin).

15. Clients

This section is relevant for yule only. Section heading:

[Clients]

Entries must be of the form:

Client= `hostname@salt@verifier`

See Section 3 (p. 73) on how to compute a valid entry.

The hostname must be the same name that the client retrieves from the host on which it runs. Usually, this will be a fully qualified hostname, no numerical address. However, there is no method that guarantees to yield the fully qualified hostname (it is not even guaranteed that a host has one ...). The only way to know for sure is to set up the client, and check whether the connection is refused by the server with a message like **Connection attempt from unregistered host `hostname`** In that case, *hostname* is what you should use.

> # CAVEAT
>
> Problems and oddities encountered in client/server setups (like client messages from *127.0.0.1*, server warnings about unknown/unresolved peer, etc. are *always*(at least so far) due to incorrect configuration of the DNS or the /etc/hosts file.
>
> A surprisingly large number of hosts are not able to determine the own hostname, or reverse lookup adresses on the own local network. Don't bother asking about such problems — fix your DNS.

Alias= `alias@hostname` — This option allows to define an alias for a hostname, which is evaluated when the 'SetClientFromAccept' option is active. This may be used to fix the problem that the remote samhain client reports itself with different hostname than known to the network layer on the side of the yule server, which e.g. may happen if you route the samhain traffic through a secondary management network that is different from the primary network of the system.

Appendix D. List of database fields

The database may hold (i) internal message from yule, the log server, and (ii) client messages. The latter result in *two* rows: one for the client message, and one for the server message recording the arrival of the client message, the originating remote host, and the timestamp. The different message types can be recognized by the *log_ref* field (see below).

Many database fields record details of files (see **man stat**), before (_old) and after (_new) a detected modification. For some items, both numeric (iXXX) and string values are reported, because the translation between both is host-specific. This allows to perform updates of the file signature database(s) on the server side. Other fields are listed below. Basically, most of the fields supply additional information for *log_msg* if relevant.

1. General

log_index	Unique index of the message (primary key).
log_ref	Zero for internal server messages, NULL for messages received from a client, log_index(client_message) for server timestamp of client message.
log_host	The host where the message originates.
log_time	The timestamp of the message.
log_sev	The severity/priority of the message.
log_msg	The message itself.
log_hash	A checksum over the union of user-defineable fields.
entry_status	NEW for new entries. Used by the Beltane frontend to track the status of a message.
path	Path of a file (whenever a message refers to a file).
userid	UID of the current user if relevant (e.g. if access to a file fails).
grp	Name of a group (for messages reporting problems with a GID, e.g. no entry in /etc/group).
program	Name of the current process (startup message).
subroutine	Name of an internal subroutine (in messages reporting failure of a subroutine).
status	Exit status value of samhain.
hash	Checksum of configuration file (if gpg not used). Startup message.
path_data, hash_data	Path and checksum of data file (if gpg not used). Startup message.
key_uid, key_id	User ID and key id of GPG key used to sign the configuration file. Startup message.
key_uid_data	User ID of GPG key used to sign the data file (different keys for configuration and data file cause program abort). Startup failure message.
peer	Address of a connecting host.

obj	Generic field to hold additional information. Occasionally used.
interface	Name of a library routine/interface (error messages).
dir	Name of a directory, if relevant.
linked_path	In reports about dangling symlinks.
port	Port number (in reports about connections errors).
service	Logging facility or remote service (failure reports).

2. Modules

module	Name of a samhain module (e.g. the module to watch login/logout events). Used in initialization/error reports for a module.
return_code	Return code from a module. Used in initialization/error reports for a module.
syscall	ID of bad syscall. Formerly used by kernel checking module.
ip	IP address. Login/logout watch. Also used in received syslog messages (see below).
tty	Terminal used. Login/logout watch.
time	Login/logout time. Also used in some other messages (e.g. time to complete file check).
fromhost	Host from which user is logged in. Login/logout watch.

3. Syslog

ip	IP of remote host received syslog reports. Also used in the login/logout watch module (see above).
facility	Syslog facility for received syslog reports.
priority	Syslog priority for received syslog reports.
syslog_msg	Syslog message for received syslog reports.

Appendix E. List of recognized file types

Filetypes have hierarchical names of the form G1:G2:G3, and in the "match(filetype)" condition you can specify filetypes as G1:G2:G3, or G1:G2 (less specific), or G1 (generic). The list of currently (version 2.8.2) recognized filetypes comprises:

```
IMAGE:COMPRESSED:JPG
    IMAGE:COMPRESSED:PNG IMAGE:COMPRESSED:JPG IMAGE:COMPRESSED:GIF
    IMAGE:COMPRESSED:TIFF IMAGE:COMPRESSED:PCX IMAGE:RAW:BMP
    IMAGE:RAW:XPM IMAGE:SPECIAL:AUTOCAD IMAGE:SPECIAL:COREL
    IMAGE:SPECIAL:FITS IMAGE:SPECIAL:VISIO IMAGE:SPECIAL:DICM
    IMAGE:SPECIAL:PHS IMAGE:SPECIAL:XCF MOVIE:COMPRESSED:RIFF
    MOVIE:RAW:MOV MOVIE:COMPRESSED:MPG MOVIE:COMPRESSED:QT
    MOVIE:COMPRESSED:FLI MOVIE:COMPRESSED:FLASH
    MOVIE:COMPRESSED:WMV AUDIO:RAW:SND AUDIO:RAW:EMOD AUDIO:RAW:MOD
    AUDIO:RAW:WAVE AUDIO:RAW:DEC AUDIO:STANDARD:MIDI
    AUDIO:COMPRESSED:REAL AUDIO:COMPRESSED:OGG
    AUDIO:COMPRESSED:FLAC AUDIO:COMPRESSED:MP3
    ARCHIVE:COMPRESSED:LHA ARCHIVE:COMPRESSED:RAR
    ARCHIVE:COMPRESSED:ZIP ARCHIVE:COMPRESSED:7Z
    ARCHIVE:COMPRESSED:COMPRESS ARCHIVE:COMPRESSED:GZIP
    ARCHIVE:COMPRESSED:BZIP2 ARCHIVE:COMPRESSED:ARJ
    ARCHIVE:COMPRESSED:HPAK ARCHIVE:COMPRESSED:JAM
    ARCHIVE:COMPRESSED:SQUISH ARCHIVE:COMPRESSED:CAB
    ARCHIVE:COMPRESSED:ZOO ARCHIVE:COMPRESSED:XPK
    ARCHIVE:PACKAGE:RPM ARCHIVE:PACKAGE:DEB ARCHIVE:UNIX:AR
    ARCHIVE:UNIX:TAR LIBRARY:JAVA:CLASS DOCUMENT:OFFICE:WORD
    DOCUMENT:OFFICE:EXCEL DOCUMENT:OFFICE:WORD DOCUMENT:OFFICE:ALL
    DOCUMENT:ADOBE:PDF DOCUMENT:ADOBE:EPS DOCUMENT:STANDARD:RTF
    DOCUMENT:ID:VCARD EXECUTABLE:DOS:EXE EXECUTABLE:DOS:COM
    EXECUTABLE:UNIX:SHELL EXECUTABLE:UNIX:ELF EXECUTABLE:DOS:COM
    EXECUTABLE:AMIGAOS:EXECUTABLE DATABASE:ANY:ACCESS
    DATABASE:ANY:MYSQL
```