Stealth V.4.01.02

Frank B. Brokken
Center for Information Technology, University of Groningen

2005-2015

Contents

Chapter 1

Introduction

Welcome to **stealth**. The program **stealth** implements a file integrity scanner. The acronym **stealth** can be expanded to

SSH-based Trust Enforcement Acquired through a Locally Trusted Host.

This expansion contains the following key terms:

- `SSH-based`: The file integrity scan is (usually) performed over an ssh-connection. Usually the computer being scanned (called the *client*) and the computer initiating the scan (called the `monitor`) are different computers.
- The client should accept incoming ssh-connections from the monitor. The monitor doesn't have to (and shouldn't, probably).
- `Trust Enforcement`: following the scan, 'trust' is enforced in the client, due to the integrity of its files.
- `Locally Trusted Host`: the client apparently trusts the monitor to use an ssh-connection to perform commands on it. The client therefore *locally trusts* the monitor. Hence, *locally trusted host.*

stealth is based on an idea by *Hans Gankema* and *Kees Visser*, both at the Center for Information Technology of the University of Groningen.

stealth's main task is to perform file integrity tests. However, the testing will leave virtually no sediments on the tested computer. Therefore, **stealth** has *stealthy* characteristics. I consider this an important security improving feature of **stealth**.

The monitor itself only needs two kinds of outgoing services: **ssh**(1) to reach its clients, and some mail transport agent (e.g., **sendmail**(1)) to forward its outgoing mail to some mail-hub.

Here is what happens when **stealth** is run:

- First, the *policy* file is read. For each client a policy file is defined, specifying the actions to be performed, and specifying the values of several variables used by **stealth**.

- If the command-line option `-daemon <uds>` is specified, **stealth** runs as a daemon process, using the Unix Domain Socket (`<uds>`) for communication with **stealth** processes running in IPC mode.

  If access to the Unix Domain Socket defined by Stealth running in daemon mode should be restricted, it can be defined in a directory with is only accessible to the user running Stealth (this will often be the root-user).

  When running in daemon mode, `-repeat <seconds>` may be specified to rerun the integrity scan every `<seconds>` seconds. If an integrity scan is being performed when, according to the repeat interval the next integrity scan is due, then the current scan is first completed. Once completed, the next integrity scan will be performed after `seconds` seconds.

- Next, the monitor opens a command shell on the client using **ssh**(1), and a command shell on the monitor computer itself using **sh**(1).

- Once the command shells are available, commands defined in the policy file are executed in their order of appearance. Examples are given below. Normally, return values of the programs are tested. When return values are to be tested **stealth** terminates when a non-zero return value is sensed. If this happens, a message stating the reason why **stealth** terminated is written to the report file (and into the mail sent by **stealth**). In some cases (e.g., when the report file could not be written), the message is written to the standard error stream.

- Very often integrity tests can be controlled using **find**(1), calling programs like **ls**(1), **sha1sum**(1) or its own `-printf` method to produce file-integrity related statistics. Most of these programs write file names at the end of generated lines. This characteristic is used by one of **stealth**'s internal routines to detect changes in the generated output. Such changes could indicate some harmful intent, like an installed *root-kit*.

- When changes are detected, they are logged in a *report file*, to which information is always appended. **Stealth** never reduces the report file's size or rewrites its contents. When information is added to the report file (beyond a plain time stamp) the newly added information is e-mailed to a configurable e-mail address for further (human) processing. Usually the e-mail is sent to the systems manager of the tested client. **Stealth** follows the 'dark cockpit' approach in the sense that no mail is sent when no changes were detected.

- Report and other log-files may safely be rotated between a pair of `-suspend` and `-resume` commands (see below at the section 5.7).

Instead of running in daemon mode, **stealth** may also run in 'foreground' mode. In foreground mode the option `-daemon` is not specified. When running in foreground mode **stealth** either performs one integrity scan (and terminates) or, if the `-repeat` option has been specified, it repeatedly performs integrity scans, at intervals determined by the `-repeat` and `-random-interval` options. When `-repeat` is specified with **stealth** running in foreground mode a prompt is shown (i.e., '`?  `') with **stealth** terminating after pressing the `Enter`-key.

Alternatively, **stealth** may run in 'inter process communication' mode (IPC mode). IPC mode is characterized by using one of the command-line options `-reload, -rerun, -suspend, -resume` or `-terminate`. In IPC-mode **stealth** communicates with an existing **stealth** daemon, using the Unix Domain Socket defined by the **stealth** daemon. These options require but one argument: the location of the Unix Domain Socket defined by a running **stealth** daemon.

- When started using the `-reload <uds>` command-line option, the stealth daemon that created the Unix Domain Socket reloads its policy file (and skip-file), immediately followed by another integrity scan;

- When started using the `-rerun <uds>` command-line option, the stealth daemon that created the Unix Domain Socket performs another integrity scan;

- When started using the `-terminate <usd>` command-line option, the stealth daemon that created the Unix Domain Socket terminates.

The options `-suspend` and `-resume` (see section 5.7) were implemented to allow safe rotations of **stealth**'s log-files.

1.1 What's new in Stealth V.4.01.02

With 4.00.00:

- Version 3.00.00 was only short-lived. The inter-process communication using signals never ran smoothly. Version 4.00.00 re-implements **stealth**'s inter-process communication using Unix Domain Sockets.
- Previously required absolute file paths are no longer required. When relative file paths are used with the **stealth** daemon or with **stealth** performing an integrity scan as foreground process they are interpreted relatively to the current working directory. Relative file locations for options specified in the second section of the policy file are interpreted relative to the location of the policy file, and relative path specifications used in the first section of the policy file are interpreted relative to the policy file's `USE BASE`.
- The README.flow file is provided with several separately provided illustrative images in the distribution-provided directory `documentation/images`.
- Specifications for the logrotate specifications should use '`copytruncate`' and '`sharedscripts`' (see section 5.7 for an example)
- Examples in the manual still use `sha1sum` when checking hash values. Stronger hash functions (like `sha256sum`) might be preferred in practice. When updating existing policy files to use `sha256sum` rather than `sha1sum` realize that `sha256sum's` hash values are longer than `sha1sum's` hash values, and that therefore log files obtained when `sha1sum` was used are incompatible with log files obtained when `sha256sum` was used. In practice this means that new log files need to be generated, disregarding any previously geneerated log files.

Chapter 2

Installation

This chapter describes **stealth**'s compilation and installation.

2.1 Compiling and installing Stealth

After downloading the **stealth** archive, it should be unpacked. The name of the archive is of the form `stealth-4.01.02.tar.gz`, where `4.01.02` is a version number. Below, `4.01.02` should be altered into the version of the archive that is actually used.

Required, non-standard software, for compiling **stealth** is summarized in the file `required`, which is included in the source file distribution. Currently, its contents are:

```
This file lists non-standard software only. Thus, standard utilities like cp,
mv, sed, etc, etc, are not explicitly mentioned. Neither is the gcc compiler
explicitly mentioned, but a fairly recent one is assumed.

Required software for building Stealth 4.00.00
----------------------------------------------

libbobcat-dev (>= 4.01.02),


To use the provided build-script:

    icmake (>= 7.22.00)

To construct the manual and map-page:

    yodl (>= 3.05.0)

To construct the manual:

    texlive-latex-base,
    texlive-generic-recommended,
    texlive-latex-recommended,
    texlive-fonts-recommended,
    cm-super-minimal,
```

```
ghostscript
```

- **Stealth** compilation is controlled by **icmake**(1). The program maintenance utility **icmake**(1) can be obtained at `https://fbb-git.github.io/icmake/`, and it is also available in several Linux distributions (e.g., Debian, Ubuntu);
- Determine a directory where the files in **stealth**'s archive ashould be stored. E.g., if the files in the archive should be stored under `/tmp` (and assuming the archive itself is stored in `/tmp` as well) then do:

```
cd /tmp
tar xzf stealth-4.01.02.tar.gz
```

  This creates a subdirectory `stealth` containing **stealth**'s sources;
- Change-dir to that directory:

```
chdir stealth
```

- Check the contents of the files `INSTALL.im` and `icmconf`, and verify that all `#defines` match your computer's file system and software.

  When compiling **stealth**, the `bobcat`[1] header files must be available. When **stealth** is run it is dynamically linked against the bobcat library. **Bobcat** can be downloaded from `https://github.com/fbb-git/bobcat/` (and follow its installation instructions). Alternatively, bobcat may be available in your distribution, (e.g., Debian or Ubuntu). Make sure to install both the run-time (**bobcat_...**) and the development (**bobcat-dev_...**) versions.
- Execute the command

```
./build program strip
```

  This command by default creates the program **./tmp/bin/binary**.
- To install **stealth** and its documentation, several commands are available:

```
./build install program [path] - installs the program as 'path'
                                 (by default as '/usr/bin/stealth')
./build install man [base]     - install the man pages below 'base'
                                 (by default below '/usr/share/man/man1')
./build install manual [base]- install the manual below 'base'
                                 (by default below
                                     '/usr/share/doc/stealth-doc/manual')
./build install std [base]     - install standard docs below 'base'
                                 (by default below '/usr/share/doc/stealth')
./build install extra [base] - install extra docs below 'base'
                                 (by default below
                                     '/usr/share/doc/stealth-doc')
```

[1] https://fbb-git.github.io/bobcat/

Chapter 3

Granting access

Access to clients should be granted using the `ssh` protocol.

Clients must allow the monitor to connect using `ssh`. **Stealth** connects to its clients using ssh certificates, after the monitor's public SSH-key has been transferred to the clients.

3.0.1 The monitor's user: creating an ssh-key

The monitor's user calling **stealth** to scan the client must first generate an `ssh-keypair`:

```
ssh-keygen -t rsa
```

By default this generates a public/private ssh key-pair in the directory `.ssh` in the user's home directory. The program asks for a *passphrase*. A passphrase can be defined (in which case it must be proviced when **stealth** is started) or, if the security if the monitor is sufficiently guaranteed, it can remain empty. To generate an ssh-key without passphrase simply press `Enter` in response to the question

```
Enter passphrase (empty for no passphrase):
```

(a confirmation is required: just press `Enter` again).

`Ssh-keygen` then returns a key fingerprint, e.g.,

```
03:96:49:63:8a:64:33:45:79:ab:ca:de:c8:c8:4f:e9 user@monitor
```

which may be saved for future reference.

In the user's `.ssh` directory the files `id_rsa` and `id_rsa.pub` are now created, which completes the preparations at the monitor.

If, instead of running **stealth** in deamon mode it is preferred to let **stealth** perform single, but automated integrity scans, then new **ssh**(1) connections may be difficult to establish if the used ssh-key is passphrase-protected. To implement this scenario (i.e., automated integrity scans using passphrase protected ssh-keys) the program **ssh-cron**(1) can profitably be used.

3.0.2 The client's account: accepting ssh from the monitor's user

Next, at the client's account where **stealth**'s `ssh` command connects to (see also the `USE SSH` specification in section 4.2) ssh-access must be granted to the monitor's user. To do so, the monitor user's file `~/.ssh/id_rsa.pub` is added to the client account user's file `~/.ssh/authorized_keys`:

```
# transfer user@monitor's file id_rsa.pub to the client's /tmp
# directory. Then do:

cat /tmp/id_rsa.pub >> /home/account/.ssh/authorized_keys
```

This allows the user at the monitor to login at the account at the client without specifying the client account's password (of course, if the ssh-key is passphrase protected that passphrase must still be provided at the monitor when starting **stealth**).

3.0.3 Logging into the account@client account

When user@monitor now issues, for the first time, the command

```
ssh account@monitor
```

`Ssh` responds like this:

```
The authenticity of host 'monitor (xxx.yyy.aaa.bbb)' can't be
established.
RSA key fingerprint is c4:52:d6:a3:d4:65:0d:5e:2e:66:d8:ab:de:ad:12:be.
Are you sure you want to continue connecting (yes/no)?
```

Answering `yes` results in the message:

```
Warning: Permanently added 'monitor,xxx.yyy.aaa.bbb' (RSA) to the
list of known hosts.
```

The next time a login is attempted, the authenticity question is not asked anyore. However, the proper value of the host's RSA key fingerprint (i.e., the key fingerprint of the *client* computer) should *always* be verified to prevent *man in the middle* attacks. The proper value may be obtained at the client by issuing there the command

```
ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
```

This should show the same value of the fingerprint as shown when the first `ssh` connection was established. E.g.,

```
1024 c4:52:d6:a3:d4:65:0d:5e:2e:66:d8:ab:de:ad:12:be ssh_host_rsa_key.pub
```

3.0.4 Using the proper shell

On order to minimize the amount of clutter and possibly complications when only a simple command-shell is required for executing commands, it is suggested to use a **bash**(1) shell when logging into `account@client`'s account.

When another shell is already used for `account@client`, then an extra account (optionally using the same `UID` as the original account, but using **sh**(1) as the shell), could be defined in the client's `/etc/passwd` file. In the **passwd**(5) file this could, e.g., be realized for *root* as *rootsh* as follows:

```
rootsh:x:0:0:root:/root:/bin/bash
```

If shadow passwording is used, a matching entry in the `/etc/shadow` file is required as well.

Chapter 4

The 'policy' file

Stealth uses a policy file consisting of two sections, the second section is optional, and starts at a line merely containing `%%`.

Each policy file is uniquely associated with a host to be tested. Each host may have multiple policy files, though. In that case, each policy file defines its own set of checks to be performed.

- The policy file's first section consists of two sets of data: *use directives* (starting with the keyword **USE**) and *commands*.
- The (optional) second section starts at a line merely containing `%%`. Following this separating line several long option specifications can be entered (cf. section 5.2). Options specified on the command-line take priority over options specified in the policy file. Although the `-reload` option reloads the policy file, it will not change option values originally specified as command-line options.

  This section may contain specifications of the `skip-files` and `log` options. Relative file locations specified for these options are interpreted relative to the location of the policy file. E.g., if the policy file argument is specified as `/root/client/policy` then the specification `log: client.log` results in **stealth** writing its logs into the file `/root/client/client.log`.

The policy file's first section consists of three sets of data: *define directives* (starting with the keyword **DEFINE**), *use directives* (starting with the keyword **USE**) and *commands*.

Directives are written in capitals, and should appear exactly as written below.

Blank lines and information beyond hash-marks (#) are ignored, while lines following lines terminating in backslashes (\) will be concatenated (*en passant* removing these backslashes). Leading white space on lines of the policy file is ignored.

4.1 DEFINE directives

`DEFINE` directives can be used to define symbols for longer strings. A `DEFINE` directive is constructed as follows:

```
DEFINE name     that what is defined by 'name'
```

Here,

- the `name` following `DEFINE` is the symbol that may be used in `USE` directives (see below) and `commands` (see below).
- `DEFINE` symbols can be used in other `DEFINE` symbols. However, it is the responsibility of the policy file's author to make sure that (indirect) circular definitions are avoided. E.g., after:

```
DEFINE A    ${B}
DEFINE B    ${A}
DEFINE C    ${C}

USE MAILARGS ${A} ${B} ${C}
```

  `MAILARGS` will be expanded to

```
 ${A} ${A} ${C}
```

- The text following `DEFINE name` is then inserted literally into the `USE` directive or `command`.

  Example:

```
DEFINE  SSH         /usr/bin/ssh frank@localhost -q
DEFINE  EXECSHA1     -xdev -perm /111  -type f -exec /usr/bin/sha1sum {} \;
```

  The symbols defined by `DEFINE` directives may consist of letters, digits and the underscore characters (_). In the definition of the symbol any character can be used. However, initial and/or trailing blanks are removed from definitions.

  To insert a definition into a `USE` directive or `command` use the form

```
${name}
```

  E.g., `${EXECSHA1}`. Concrete examples are provided below.

4.2 USE directives

`USE` directives provide **stealth** with arguments which may be conditional to a certain installation. The following **USE** directives are supported:

- **USE BASE** `basedirectory`

  BASE defines the directory from where **stealth** operates. All relative path specifications in this section of the policy file are interpreted relative to **BASE**. *By default* this is the directory where **stealth** was started.

  If necessary, **stealth** creates all **BASE** and other directories below `BASE`.

  Example: using the specification

```
USE BASE /root/client
```

  results in all information generated by **stealth** being written in or below the directory `/root/client`.

- **USE DD** <`dd`>
  The **DD** specification uses `/bin/dd` as default, and defines the location of the **dd**(1) program, both on the server and on the client. The **DD** program is used to copy files between the client and the monitor without creating separate ssh-connections. The **DD** program is only used by **stealth** for the `PUT` and `GET` commands, described below.

  Example showing the default:

```
USE DD /bin/dd
```

- **USE DIFF** `path-to-diff`

  The **DIFF** specification uses `/usr/bin/diff` as default, and defines the location of the **diff**(1) program. The **diff**(1) program is used to compare a formerly created logfile of an integrity check to a newly created logfile.

  Example showing the default:

  ```
  USE DIFF /usr/bin/diff
  ```

- **USE DIFFPREFIX** <`prefix`>

  The **DIFFPREFIX** specification defines the size of the prefix added by the `DIFF` command to lines produced by commands executed through **stealth**. The default `/usr/bin/diff` program prefixes lines by either '> ' or '< '. The default value for <`prefix`> therefore equals 2.

  Example showing the default:

  ```
  USE DIFFPREFIX 2
  ```

- **USE EMAIL** `address`

  The **EMAIL** specification defines the email-address to e-mail the client's integrity scan report to. Mail is only sent when information has changed.

  Example showing the default:

  ```
  USE EMAIL root
  ```

- **USE MAILER** `mailer`

  The **MAILER** specification defines the program that is used to send the mail to the **EMAIL**-address. By default this is /**usr**/**bin**/**mail**(1). The **MAILER** program is called as follows:

  ```
  MAILER  MAILARGS EMAIL
  ```

  (`MAILARGS`: see below). The information to be mailed is read from `MAILER`'s standard input stream.

  Example showing the default:

  ```
  USE MAILER /usr/bin/mail
  ```

- **USE MAILARGS** `arguments` The **MAILARGS** specification defines the arguments to be to be passed to the `MAILER` program. By default this is

  ```
  USE MAILARGS -s "STEALTH scan report"
  ```

  Note that blanks may be used in the subject specification: use double or single quotes to define elements containing blanks. Use `\"` to use a double quote in a string that is itself delimited by double quotes, use `\'` to use a single quote in a string that is itself delimited by single quotes.

  Subtlety: in constructions like

  ```
  USE MAILARGS " 't was brillig " and 't went well
  ```

  the following arguments are passed to `MAILER`:

  - `" 't was brillig "`
  - `and`

- 't
- went
- well

So, when single- and double-quoted strings overlap, the first string is taken as a string, and the information beyond the first string is thereupon interpreted.

- **USE REPORT** <file-spec>
  REPORT defines the name of the reportfile. Information is always appended to this file. At each **stealth** integrity scan a *time marker line* is written to the report file. Only when (in addition to the marker line) additional information is appended to the report file the added contents of the report file are mailed to the mail address specified in the **USE EMAIL** specification. When a relative file specification is used it is interpreted a location relative to the `USE BASE` specification.
  Example showing the default:
  `USE REPORT report`

- **USE SH** <sh>
  The **SH** specification uses `/bin/sh` as default, and defines the command shell used by the monitor to execute commands on itself. This must be an absolute path specification.
  Example showing the default:
  `USE SH /bin/sh`

- **USE SSH** <user>
  The SSH specification has no default, and *must* be specified. This must be an absolute path specification.

  Assuming the client *trusts* the monitor (which is after all what this program is all about, so this should not be a very strong assumption), preferably the public ssh key of the monitor should be placed in the client's root `.ssh/authorized_keys` file, granting the monitor root access to the client. Root access is normally needed to gain access to all directories and files of the client's file system.

  In practice, connecting to an account using the **sh**(1) shell is preferred. When another shell is already used by that account, one should make sure that its shell doesn't define its own redirections for standard input and standard output. One way to accomplish that is for force the execution of `/bin/sh` in the **USE SSH** specification.

  An example of an `SSH` specification to scan a localhost is:

```
USE SSH root@localhost -T -q                # root's shell is /bin/sh
```

  The same, now explicitly using `/bin/bash`:

```
USE SSH root@localhost -T -q exec /bin/bash # root uses another shell
```

  Alternatively, `-noprofile` can be specified to prevent any profile-initialization:

```
USE SSH root@localhost -T -q exec /bin/bash --noprofile
```

In some installations **stealth** is used to inspect the monitor itself, even though this is *not* recommended, as it breaks one of the main reasons for **stealth**'s existence. But in those situations (so, where **stealth** is used to monitor the integrity of the `localhost`), `/bin/bash` could be specified at the `USE SSH` directive. For example:

```
# For stealth inspecting localhost:
    USE SSH /bin/bash --noprofile
```

4.3 Commands

Following the **USE** specifications, *commands* can be specified. The commands are executed in their order of appearance in the policy file. Processing continues until the last command has been processed or until a tested command (see below) returns a non-zero return value.

4.3.1 LABEL commands

The following **LABEL** commands are available:

- **LABEL** `text`

  This defines a text-label which is written to the **REPORT** file, just before the output generated by the next **CHECK**-command. If the next **CHECK**-command generates no output, the label is not written to the **REPORT**-file. Once a **LABEL** has been defined, it is used until it is redefined by the next **LABEL** command. Use an empty **LABEL** command to suppress printing labels.

  The text may contain `\n` characters (two characters) which are transformed to a newline character.

- **LABEL**

  As noted, this clears a previously defined `LABEL` command.

Examples:

```
LABEL Inspecting files in /etc\nIncluding subdirectories
LABEL
```

The latter **LABEL** command clears the text of the former **LABEL** command.

4.3.2 LOCAL commands

LOCAL commands can be used to specify commands that are executed on the monitor itself. The following **LOCAL** commands are available:

- **LOCAL** `command`

  Execute `command` on the monitor, using the **SH** command shell. The command must succeed (i.e., must return a zero exit value). Example:

  ```
  LOCAL mkdir /tmp/client
  ```

  This command creates the directory `/tmp/client` on the monitor.

- **LOCAL NOTEST** `command`

  Execute `command` on the monitor, using the **SH** command shell. The command may or may not succeed. Example:

  ```
  LOCAL NOTEST mkdir /tmp/subdir
  ```

  This command creates `/tmp/subdir` on the monitor. The command fails if the directory cannot be created, but this does not terminate **stealth**.

- **LOCAL CHECK** [**LOG** =] `logfile [pathOffset] command`

  Execute `command` on the monitor, using the **SH** command shell. The command must succeed. The output of this command is compared to the output of this command generated during the previous integrity check run by **stealth**.

  The phrase **LOG** = is optional. When a relative file location is specified at <`file-spec`> it is interpreted relatively to the `USE BASE` path specification.

  `PathOffset` is also optional. If specified it defines the (0-based) offset where path-names of inspected files start in lines produced by <`command`>. By default **stealth** assumes that the first occurrence of a forward slash defines the first character of the path-names of inspected files.

  For example, if diff-output looks like this:

```
01234567890123456789012345678901234567890 (column offsets, not part of
                                           the diff-output)

33c33
< 90d8b506d249634c4ff80b9018644567  filename-specification
---
> b88d0b77db74cc4a742d7bc26cdd2a1e  filename-specification
```

  then the specification

```
LOCAL CHECK logfile 36 command-to-be-executed
```

  informs **stealth** where to find the filename specifications in the diff-output. Using the standard `/usr/bin/diff` command, this offset equals 2 + the offset of the `filename-specification` found in `command-to-be-executed`.

  If the command does not succeed a *warning* message is written to the report file. The warning message informs the reader that 'remaining results might be forged:

```
*** BE CAREFUL *** REMAINING RESULTS MAY BE FORGED
```

  This situation may occur, e.g., if an essential program (like `sha1sum`) was transferred to the monitor, and it was apparently modified since the previous check. Processing continues, but remaining checks performed at the client computer should be interpreted with *extreme* caution.

  The output of this command is compared to the output of this command generated during the previous run of **stealth**. Any differences are written to **REPORT**.

  If differences were found, the existing `logfile` name is renamed to `logfile.YYYYMMDD-HHMMSS`, with `YYYYMMDD-HHMMSS` the datetime-stamp at the time **stealth** was run.

  Over time, many `logfile.YYMMDD-HHMMSS` files could be accumulated. It is up to the monitor's systems manager to decide what to do with old datetime-stamped logfiles. For instance, the following script removes all **stealth** reports below the current directory that are older than 30 days:

```
#/bin/sh
FILES=`find ./ -path '*[0-9]' -mtime +30 -type f`

if [ "$FILES" != "" ] ; then
    rm -f $FILES
fi
```

  The `logfile` specifications may use relative and absolute paths. When relative paths are used, these paths are relative to **BASE**. When the directories implied by the `logfile` specifications do not yet exist, they are created first.

  Example:

```
LOCAL CHECK LOG = local/sha1sum sha1sum /tmp/sha1sum
```

This command will check the SHA1 sum of the `/tmp/sha1sum` program. The resulting output is saved at **BASE**`/local/sha1sum`. The program must succeed (i.e., `sha1sum` must return a zero exit-value).

- **LOCAL NOTEST CHECK** [**LOG** =] `logfile [pathOffset] command`

  Execute `command` on the monitor, using the **SH** command shell.

  The phrase **LOG** = is optional. When a relative file location is specified at <`file-spec`> it is interpreted relatively to the `USE BASE` path specification.

  The command may or may not succeed. Otherwise, the program performs exactly like the **LOCAL CHECK ...** command, discussed above.

  Example:

```
LOCAL NOTEST CHECK LOG=local/sha1sum sha1sum /tmp/sha1sum
```

  This command will check the SHA1 sum of the `/tmp/sha1sum` program. The resulting output is saved at **BASE**`/local/sha1sum`. The program may or may not succeed (i.e., `sha1sum` may or may not return a zero exit-value).

4.3.3 REMOTE commands

Plain commands can be executed on the client computer by merely specifying them. Of course, this means that programs called `LABEL`, `LOCAL USE` or `DEFINE`, cannot be executed, since these names are interpreted otherwise by **stealth**. It's unlikely that this will cause problems. Remote commands must succeed (i.e., their return codes must be 0).

Remote commands are commands executed on the client using the **SSH** shell. These commands are executed using the standard `PATH` set for the **SSH** shell. However, it is advised to specify the full pathname to the programs to be executed, to prevent "trojan approaches" where a trojan horse is installed in an 'earlier' directory of the `PATH`-specification than the intended program.

Two special remote commands are `GET` and `PUT`, which can be used to copy files between the client and the monitor. Internally, `GET` and `PUT` use the `DD` use-specification. If a non-default specification is used, one should ensure that the alternate program accepts **dd**(1)'s `if=, of=, bs=` and `count=` options. With `GET` the options `bs=, count=` and `of=` are used, with `PUT` the options `bs=, count=` and `if=` are used. Normally there should be no need to alter the default `DD` specification.

The `GET` command may be used as follows:

- **GET** <`client-path`> <`local-path`>
  Copy the file indicated by `client-path` at the client to `local-path` at the monitor. `client-path` must be the full path of an existing file on the client, `local-path` may either be a local directory, in which case the client's file name is used, or another file name may be specified, in which case the client's file is copied to the specified local filename. If the local file already exists, it is overwritten by the copy-procedure.

  Example:

```
GET /usr/bin/sha1sum /tmp
```

  The program `/usr/bin/sha1sum`, available at the client, is copied to the monitor's `/tmp` directory. If copying fails for some reason, any subsequent commands are skipped, and **stealth** terminates.

- **GET NOTEST** <client-path> <local-path>
  Copy the file indicated by `client-path` at the client to `local-path` at the monitor. `client-path` must be the full path of an existing file on the client, `local-path` may either be a local directory, in which case the client's file name is used, or another file name may be specified, in which case the client's file is copied to the specified local filename. If the local file already exists, it is overwritten by the copy-procedure.

  Example:

  ```
  GET NOTEST /usr/bin/sha1sum /tmp
  ```

  The program `/usr/bin/sha1sum`, available at the client, is copied to the monitor's `/tmp` directory. Remaining commands in the policy file are executed, even if the copying process wasn't successful.

The `PUT` command may be used as follows:

- **PUT** <local-path> <remote-path>
  Copy the file indicated by `local-path` at the monitor to `remote-path` at the client. The argument `local-path` must be the full path of an existing file on the monitor. The argument `remote-path` must be the full path to a file on the client. If the remote file already exists, it is overwritten by `PUT`.

  Example:

  ```
  PUT /tmp/sha1sum /usr/bin/sha1sum
  ```

  The program `/tmp/sha1sum`, available at the monitor, is copied to the client as `usr/bin/sha1sum`. If the copying fails for some reason, any subsequent commands are skipped, and **stealth** terminates.

- **PUT NOTEST** <local-path> <remote-path>
  Copy the file indicated by `local-path` at the monitor to `remote-path` at the client. The argument `local-path` must be the full path of an existing file on the monitor. The argument `remote-path` must be the full path to a file on the client. If the remote file already exists, it is overwritten by `PUT`.

  Example:

  ```
  PUT NOTEST /tmp/sha1sum /usr/bin/sha1sum
  ```

  Copy the file indicated by `local-path` at the monitor to `remote-path` at the client. The argument `local-path` must be the full path of an existing file on the monitor. The argument `remote-path` must be the full path to a file on the client. If the remote file already exists, it is overwritten by `PUT`. Remaining commands in the policy file are executed, even if the copying process wasn't successful.

Other commands to be executed on the client can be specified as follows:

- `command`

  Execute '`command`' on the client, using the **SSH** command shell. The command must succeed (i.e., must return a zero exit value). However, any output generated by the command is ignored. Example:

  ```
  /usr/bin/find /tmp -type f -exec /bin/rm {} \;
  ```

  This command will remove all ordinary files at and below the client's `/tmp` directory.

- **NOTEST** `command`

  Execute `command` on the client, using the **SSH** command shell. The command may or may not succeed.

  Example:

```
NOTEST /usr/bin/find /tmp -type f -exec /bin/rm {} \;
```

Same as the previous command, but this time the exit value of `/usr/bin/find` is not interpreted.

- **CHECK** [**LOG** =] `logfile [pathOffset] command`

  Execute `command` on the client, using the **SSH** command shell.

  The phrase **LOG** = is optional. When a relative file location is specified at <`file-spec`> it is interpreted relatively to the `USE BASE` path specification.

  `PathOffset` is also optional, and has the same meaning as for the `LOCAL CHECK` command, described above.

  The command must succeed. The output of this command is compared to the output of this command generated during the previous run of **stealth**. Any differences are written to **REPORT**. If differences were found, the existing `logfile` name is renamed to `logfile.YYYYMMDD-HHMMSS`, with `YYYYMMDD-HHMMSS` being the datetime-stamp at the time **stealth** was run.

  Note that the command is executed on the client, but the logfile is kept at the monitor. This command represents the core of the method implemented by **stealth**: there will be no residues of the actions performed by **stealth** on client computers.

  Several examples (note the use of the backslash as line continuation characters):

  ```
  CHECK LOG = remote/ls.root  /usr/bin/find / \
          -xdev -perm /6111 -type f -exec /bin/ls -l {} \;
  ```

  All suid/gid/executable files on the same device as the root-directory (/) on the client computer are listed with their permissions, owner and size information. The resulting listing is written on the file **BASE**`/remote/ls.root`.

  This long command could be formulated shorter using a `DEFINE`:

  ```
  DEFINE LSFIND   -xdev -perm /6111 -type f -exec /bin/ls -l {} \;
  CHECK remote/ls.root  /usr/bin/find / ${LSFIND}
  ```

  Another example:

  ```
  DEFINE SHA1SUM -xdev -perm /6111 -type f -exec /usr/bin/sha1sum {} \;
  CHECK remote/sha1.root /usr/bin/find / ${SHA1SUM}
  ```

  The SHA1 checksums of all suid/gid/executable files on the same device as the root-directory (/) on the client computer are determined. The resulting listing is written on the file **BASE**`/remote/sha1.root`.

- **NOTEST CHECK** [**LOG** =] `logfile [pathOffset] command`

  Execute `command` on the client, using the **SSH** command shell.

  The phrase **LOG** = is optional. When a relative file location is specified at <`file-spec`> it is interpreted relatively to the `USE BASE` path specification.

  `PathOffset` is also optional, and has the same meaning as for the `LOCAL CHECK` command, described above. The command may or may not succeed. Otherwise, the program acts identically as the **CHECK ...** command, described above. Example (using the same `${SHA1SUM}`)definition:

  ```
  NOTEST CHECK LOG = remote/sha1.root /usr/bin/find / ${SHA1SUM}
  ```

  The SHA1 checksums of all suid/gid/executable files on the same device as the root-directory (/) on the client computer are determined. The resulting listing is written on the file **BASE**`/remote/sha1.root`. **stealth** will not terminate if the `/usr/bin/find` program returns a non-zero exit value.

4.3.4 Preventing Controller Denial of Service (–max-size)

Either by malicious intent or by accident (as happened to me) the monitor may be a victim of a (self-inflicted) Denial of Service (DOS) attack. This DOS attack may occur when the client (apparently) sends a never ending stream of bytes in response to a `GET` or `REMOTE` command. Once one of my monitors fell victim to this attack when a client's power went down and the monitor kept on trying to read bytes from that client filling up the monitor's disk....

This problem was of course caused by a programming error: while reading information from a client **stealth** failed to check whether the reading had actually succeeded. This bug has now been fixed, but an intentional DOS attack could still be staged along this line when, e.g., the **find**(1) command is somehow replaced by a manipulated version continuously writing information to its standard output stream. Without further precaution **stealth** would receive a never ending stream of bytes to be written to its 'report' file, thus causing its disk to fill up.

To prevent this from happening **stealth** now offers the `-max-size` command line option allowing the specification of the maximum size of a stream of bytes received by **stealth** (e.g., a report or downloaded file). The maximum is used for each individual download and can be specified in bytes (using no suffix or the `B` suffix), kilo-bytes (using `K`), mega-bytes (using `M`) or giga-bytes (using `G`). The default is set at 10M, equivalent to the command line specification of `-max-size 10M`.

If a file or report received from the client exceeds its maximum allowed size then **stealth** terminates after writing the following message to the report file (which is sent to the configured mail address):

```
STEALTH - CAN'T CONTINUE: '<name of offending file>' EXCEEDS MAX.
                                                DOWNLOAD SIZE (<size shown>)
    STEALTH - THIS COULD SIGNAL A SERIOUS PROBLEM WITH THE CLIENT
    STEALTH - ONE OR MORE LOG FILES MAY BE INVALID AS A RESULT
    STEALTH - *** INVESTIGATE ***
```

Since a `-max-size` specification may cause **stealth** to terminate while receiving the output of a (remotely run) command, an empty or partial log file will be the result. Of course this partial result is spurious as it is a direct result of **stealth** terminating due to a size violation.

After investigating (and removing) the reasons for the size violation a new **stealth** run using the previous log file as the latest baseline should show only expected changes.

For example, assume the following situation represents a (valid) state of logfiles:

```
    etc                     stealth
    setuid                  stealth.20080316-105756
```

Now **stealth** is run with `-max-size 20`, prematurely terminating **stealth**. This results in the following set of logfiles:

```
    etc                     stealth
    setuid                  stealth.20080316-105756
                            stealth.20080316-110215
```

The file `stealth` now contains incomplete data with the (latest) file `stealth.20080316-110215` containing its previous contents.

The reasons for the size-violation should of course be investigated and removed. It is suggested to move

the file last saved (`stealth.20080316-110215`) to the file `stealth`, as it represents the state before the size violation was encountered. Following this **stealth** should operate normally again.

Chapter 5

Running 'stealth'

Now that **stealth** has been compiled, the construction of a policy file has been covered, and a service-account on the client has been defined, what must be done to run **stealth** in practice?

Here's what remains to be done:

- Install **stealth** at a proper location
- Construct one or more policy files
- Learn to interpret **stealth**'s output.
- Optionally, automate the removal of old log-files.
- Determine a schedule for running stealth automatically, e.g. using **cron**(1) or **ssh-cron**(1)

In this chapter, these topics are discussed.

5.1 Installing 'stealth'

As **stealth** is mainly a system administrator's tool, it could be installed in `/usr/bin`. In that case, do (as *root*) in the directory where **stealth** was compiled/unpacked:

```
./build install program
```

Alternatively, another default location may be specified in the `INSTALL.im` file or may be provided to the `build` script. E.g.,

```
./build install program /usr/local/bin/stealth
```

installing the binary program as `/usr/local/bin/stealth`.

The provided `icmake build` script can be started without arguments for an overview of possible commands.

5.2 Stealth command-line and policy file options

Short options are provided between parentheses, immediately following their long option equivalents.

Option descriptions showing (C) can only be used on the command-line, and are ignored when specified in the second section of the policy file.

In the overview of options '<uds>' represents the name of the *Unix Domain Socket* to use, and '<file-spec>' refers to a (relative or absolute) specification of a file location.

With the first and second synopses relative locations (of the Unix Domain Socket and of other file-specifications) are interpreted relative to the current working directory.

Command-line options overrule options defined in the policy-file.

- `-daemon (-d) <uds>`: (C) run as background (daemon) process. When the Stealth daemon process is started, the Unix Domain Socket (tt<uds>) may not already exist.
- `-dry-run`: (C) no integrity scans or reloads are performed, but are assumed OK. Remaining tasks are normally performed;
- `-help (-h)`: (C) Display help information and exit;
- `-log (-L) <file-spec>`: log messages are appended to 'file-spec'. If file-spec does not exist, it is first created;
- `-logmail`: mail sent by **stealth** is logged (requires `-log` or `-syslog`);
- `-max-size <size>[BKMG]`: files retrieved by `GET` commands may at most have <size> bytes (B), KBytes (K), MBytes (M), GBytes (G). The default size is 10M, the default unit is B.
- `-no-mail`: mail is not sent. By default mail is sent as configured in the policy-file (`-logmail` can be specified independently from `-no-mail`);
- `-parse-policy-file (-p)`: (C) parse the policy file, after which **stealth** ends.
  Specify once to see the numbered commands;
  twice to see the policy file parsing steps as well.
  Results are written to the std. output.
- `-ping <uds>`: (C) performs no actions, but is used to verify that a **stealth** daemon can be reached via its Unix Domain Socket (<uds>). The daemon will respond even if it's currently performing an integrity scan. It is used by the `/usr/bin/stealthcron` script to verify that a **stealth** daemon is alive.
- `-random-interval (-i) <interval>[m]`: start the scan a random interval of <interval> seconds (or minutes if an 'm' is appended (no blanks) to <interval>) following the delay specified at `-repeat` (see below). This option requires specification of the `-repeat` and `-daemon` options;
- `-reload <uds>`: (C) reloads the configuration and skip-files and restarts the scan of the **stealth** daemon process. Options defined in the policy file are also reloaded. However, command-line options always take priority over options defined in the policy file, so when command-line options were used when starting **stealth** in daemon mode, they cannot be modified by reloading the policy file.
- `-repeat <seconds>`: wake up and perform an integrity scan at interrupts or after <seconds> seconds (or minutes if an 'm' is appended (no blanks) to <seconds>) after completing the previous integrity scan. The option `-random-interval` can be used to add a random delay to <seconds> until the next integrity scan is performed. This option requires specification of the and `-daemon` option;

- `-rerun <uds>`: (C) start executing the integrity scan commands that are specifed in the **stealth** daemon process's policy file;
- `-resume <uds>`: (C) resume a suspended **stealth** process, implies `-rerun`;
- `-run-command (-r) <nr>`: (C) Only execute command number <nr> (natural number). Command numbers are shown by **stealth** `--parse-policy-file`. This option can only be specified using the second synopsis;
- `-skip-files (-s) <file-spec>`: all entries in `<file-spec>` are skipped. Their integrity is not monitored. If an entry is already present in a log file then **stealth** once generates an `IGNORING` message in the mail sent to the address specified at `EMAIL` in the policy file. Each entry mentioned in `file-spec` must be on a line of its own and must be specified using absolute file paths. Entries ending in a slash are assumed to be directories whose full contents must be skipped. Other entries are interpreted as the names of files to skip. Initial and trailing blanks, empty lines and lines having a `#` as their 1st non blank character are ignored. Here are some examples:

```
    # skip all files in user's Mail directory
/home/user/Mail/
    # skip user's .history file
/home/user/.history
```

- `-stdout (-o)`: messages are (also) written to the std. output stream (only available with the second synopsis);
- `-suspend <uds>`: (C) suspends a currently active **stealth** process. Following `-suspend` use `-resume` to re-activate an **stealth** daemon or `-terminate` to end an **stealth** daemon;
- `-syslog`: write syslog messages;
- `-syslog-facility <facility>`: syslog facility to use. By default facility DAEMON is used;
- `-syslog-priority <priority>`: syslog priority to use. By default priority NOTICE is used;
- `-syslog-tag <tag>`: `<tag>` specifies the identifier that is prefixed to syslog messages. By default the tag 'STEALTH' is used, see also the next section;
- `-terminate <uds>`: (C) terminate a currently active **stealth** process;
- `-time-stamp (-t) <type>`: the time-stamps to use. By default UTC. To use the local time specify `-time-stamp LT`. The `-time-stamp` option does not apply to time-stamps generated by syslog (see also the next section);
- `-usage`: (C) Display help information and exit;
- `-verbosity <value>`: determines the amount of logged information. Requires options `-log` or `-syslog`. Possible values are:
  0: nothing is logged
  1: (default) mode reports and policy commands
  2: also: ipc commands and actions
  3: also: integrity scan informative messages
- `-version (-v)`: (C) Display **stealth**'s version information and terminate;

- `policy`: file specification of the policy file. If a relative location is specified then this location is interpreted relative to the current working directory. **Stealth** converts this relative specification to an absolute file location, and an option like `-reload` will reload the policy file from the thus determined absolute file path.

Only one of the options `-daemon, -reload, -resume, -suspend` or `-terminate` can be specified. The options `-reload, -rerun, -resume, -suspend,` and `-terminate` ignore any other options.

The following options are still recognized for backward compatibility with **stealth** pre-3.00 versions and will be removed in a future **stealth** version. They generate error messages suggesting alternatives:

- `-echo-commands (-e)`: echo commands to std error when they are processed; use `-log` instead.
- `-keep-alive`: run as a daemon; use `-daemon` instead.
- `-only-stdout`: scan report is written to stdout; use `-stdout` instead.
- `-quiet (-q)`: suppresses progress messages written to stderr; use `-verbosity 0` instead.
- `-suppress <uds>`: suppresses a currently active **stealth** process; use `-suspend` instead.

The following options were discontinued starting since **stealth** version 3.00.00:

- `-debug` (option `-verbosity` or `-dry-run` could be used instead);
- `-no-child-processes`;
- `-parse-config-file`.

When specifying long options in policy files initial hyphens should be omitted. Here are some examples:

```
%%
log /tmp/stealth.log
verbosity 3
```

5.2.1 Rsyslog filtering

When using **rsyslogd**(1) property based filters may be used to filter syslog messages and write them to a file of your choice. E.g., to filter messages starting with the syslog message tag (e.g., `STEALTH`) use

```
:syslogtag, isequal, "STEALTH:"   /var/log/stealth.log
:syslogtag, isequal, "STEALTH:"   stop
```

Note that the colon is part of the tag, but is not specified with the `syslog-tag` option.

This causes all messages having the `STEALTH:` tag to be written on `/var/log/stealth.log` after which they are discarded. More extensive filtering is also supported, see, e.g., `http://www.rsyslog.com/doc/rsyslog_conf_filter.html` and `http://www.rsyslog.com/doc/property_replacer.html`

Time stamps written by `rsyslogd` are not controlled by **stealth**'s `-time-stamp` option, but, e.g., by a `TZ` specification in `/etc/default/rsyslog`. Simply add the line

```
export TZ=UTC
```

to `/etc/default/rsyslog`, followed by restarting `rsyslogd` configures `rsyslogd` to generate time stamps using UTC.

5.3 Construct one or more policy files

Here we assume that **stealth** is run by *root*, and that root wants to store information about the host `client` under the subdirectory `/root/stealth/client`.

Furthermore, we assume that reports of **stealth** integrity-scans should be sent to the user `admin@elsewhere`, who is only interested in receiving a short summary of changes, as the full report can always be read at the monitor itself. To accomplish this a small support-script was developed, filtering the report generated by **stealth** down to its essentials.

As the `sha1sum` program on the client may be compromised, it is a good idea to start the integrity scan by transferring the client's `sha1sum` program to the monitor first, to verify the integrity of that program locally (i.e., at the monitor), before trusting it to compute sha1sums of the client's files. The same holds true for any libraries and support programs (like `find`) that are used intensively during integrity scans.

Sha1sum checks should be performed on all setuid and setgid files on the `client`, and in order to be able reach all files on `client`, `root@monitor` is allowed to login to the `root@client` account using an `ssh` connection.

Furthermore, sha1sum checks should be performed on all configuration files, living under `/etc` and on the file `/usr/bin/find` (which is used intensively when performing the integrity checks).

Next, the construction of the required `policy` file, implementing the abovementioned requirements, is described in the following subsections.

5.3.1 DEFINE directives

First we write some `DEFINE` directives simplifying complex command specifications:

```
DEFINE  SSHCMD  /usr/bin/ssh root@client -T -q exec /bin/bash --noprofile
DEFINE  EXECSHA1 -xdev -perm +u+s,g+s \( -user root -or -group root \) \
                -type f -exec /usr/bin/sha1sum {} \;
```

The first `DEFINE` defines the `ssh` command to use: an ssh-connection will be made to the root account at the client.

The second `DEFINE` shows the arguments for **find**(1) when looking for all root setuid or setgid normal files. For all these files the **sha1sum**(1) program should be run.

5.3.2 USE directives

Next some `USE` directives, matching our specific, local, situation, are defined:

```
USE BASE        /root/stealth/client
USE EMAIL       admin@elswhere
USE MAILER      /root/bin/stealthmail
USE MAILARGS    "Client STEALTH report"
USE SSH         ${SSHCMD}
```

- All output is written under the `/root/stealth/client` directory;

- Mail is sent to the user `admin@elsewhere`;
- As mail program we use a filtering script (`stealthmail`), which is installed in `/root/bin`;
- The script handles its own argument. As it can be used by **stealth** performing integrity scans on other clients as well, it is given an argument which can be used as e-mail subject, identifying the client-computer that has been integrity-scanned;
- The ssh-command is defined by the `SSHCMD`. It's definition is used at the `USE SSH` specification;
- Default values of all remaining `USE` directives are OK, and thus were not explicitly specified. They are:

```
USE DD          /bin/dd
USE DIFF        /usr/bin/diff
USE REPORT      report
USE SH          /bin/sh
```

5.3.3 Commands

The following commands are now defined:

- first, we copy the client's `sha1sum` program to the monitor. In practice, this should also include the shared object libraries that are used by `sha1sum`, as they might have become corrupted as well;
- Once `sha1sum` is locally available its integrity is verified;
- Once the integrity of the client's `sha1sum` has been verified, it is used to verify the integrity of the client's `/usr/bin/find` program;
- Following this, all setuid and setgid root files are located and checked for integrity;
- Finally, the integrity of the client configuration files under `/etc` is verified;

In this manual the **sha1sum**(1) program is frequently used when checking hash values. Stronger hash functions (like **sha256sum**(1)) might be preferred in practice. `sha256sum's` hash values are remarkably longer than `sha1sum's` hash values. When using these longer hash values in the manual it often clobbers the layout of examples. Therefore in this manual **sha1sum**(1) is continued to be used.

Realize, however, that when updating existing policy files to use **sha256sum**(1) instead of **sha1sum**(1), that previously generated log files (that used **sha1sum**(1)) are incompatible with log files obtained when using **sha256sum**(1). In practice this means that new log files need to be generated, and any previously geneerated log files must be disregarded.

Obtaining the client's sha1sum program

To copy the client's `sha1sum` program to a local directory we specify:

```
GET /usr/bin/sha1sum /root/tmp
```

This command must succeed.

Checking the integrity of the client's sha1sum program

Next, we check the integrity of the received `sha1sum` program. For this, we use the monitor's `sha1sum` program:

```
LABEL \nCheck the client's sha1sum program
LOCAL CHECK LOG = local/sha1 /usr/bin/sha1sum /root/tmp/sha1sum
```

The `LABEL` command writes the label to the report file just before writing the `sha1sum` program's output.

The `LOCAL` command checks the sha1sum of the program copied from the client. The report is written on the file `/root/stealth/client/local/sha1`. If this fails, **stealth** terminates, alerting `admin@elsewhere` that the check failed. This is a serious event, as it indicates that either the monitor's `sha1sum` is behaving unexpectedly or that the client's `sha1sum` program has unexpectedly changed.

The `sha1sum` program *may* have changed due to a normal upgrade. If so, `admin@elsewhere` will know this, and can (probably) ignore the warning. The next time **stealth** is run, the (now updated) SHA1 value is used, and it again compares the obtained `SHA1` value to the one obtained for the downloaded `sha1sum` program.

Checking the client's /usr/bin/find program

The client normally uses its `find` command intensively: `find` is a great tool for producing reports about almost any conceivable combination of characteristics of sets of files. Of course, the client's `find` command must itself be OK, as well as the client's `sha1sum` program. Now that we know that the client's `sha1sum` program is OK, we can use it to check the client's `/usr/bin/find` program.

Note that the monitor itself no longer needs to invest any significant processing load: only the client itself is taxed for checking the integrity of its own files:

```
LABEL \nchecking the client's /usr/bin/find program
CHECK LOG = remote/binfind /usr/bin/sha1sum /usr/bin/find
```

Checking the client's setuid/setgid files

Having checked the client's `sha1sum` and `find` programs, sha1 checksum checks should be performed on all setuid and setgid files on the client. For this we activate the `sha1sum` program on the client. In order to check the setuid/setgid files, the following command is added to the policy file:

```
LABEL \nsuid/sgid/executable files uid or gid root on the / partition
CHECK LOG = remote/setuidgid /usr/bin/find / ${EXECSHA1}
```

Checking the configuration files in the client's /etc/ directory

Finally, the client's configuration files are checked. Some of these files change so frequently that we don't want them to be checked. E.g., `/etc/adjtime, /etc/mtab`. To check the configuration file, do:

```
LABEL \nconfiguration files under /etc
CHECK LOG = remote/etcfiles                          \
      /usr/bin/find /etc -type f -not -perm /6111    \
        -not -regex "/etc/\(adjtime\|mtab\)"         \
        -exec /usr/bin/sha1sum {} \;
```

5.3.4 The complete 'policy' file

Here is the complete policy file we've constructed so far:

```
DEFINE  SSHCMD  /usr/bin/ssh root@client -T -q exec /bin/bash --noprofile
DEFINE  EXECSHA1 -xdev -perm +u+s,g+s \( -user root -or -group root \) \
                -type f -exec /usr/bin/sha1sum {} \;

USE BASE        /root/stealth/client
USE EMAIL       admin@elswhere
USE MAILER      /root/bin/stealthmail
USE MAILARGS    "Client STEALTH report"
USE SSH         ${SSHCMD}

USE DD          /bin/dd
USE DIFF        /usr/bin/diff
USE REPORT      report
USE SH          /bin/sh

GET /usr/bin/sha1sum /root/tmp

LABEL \nCheck the client's sha1sum program
LOCAL CHECK LOG = local/sha1 /usr/bin/sha1sum /root/tmp/sha1sum

LABEL \nchecking the client's /usr/bin/find program
CHECK LOG = remote/binfind /usr/bin/sha1sum /usr/bin/find

LABEL \nsuid/sgid/executable files uid or gid root on the / partition
CHECK LOG = remote/setuidgid /usr/bin/find / ${EXECSHA1}

LABEL \nconfiguration files under /etc
CHECK LOG = remote/etcfiles                          \
      /usr/bin/find /etc -type f -not -perm /6111    \
        -not -regex "/etc/\(adjtime\|mtab\)"         \
        -exec /usr/bin/sha1sum {} \;
```

5.4 Running 'stealth' for the first time

When **stealth** is now run, it creates its initial report files under `root/stealth/client`.

The first time **stealth** is run, it is usually run 'by hand'. The initial run by hand probably benefits from

the `-stdout` option, as it shows all executed commands on the standard output:

```
stealth --stdout policy
```

Furthermore, the reports are initialized. Running **stealth** this way for the `policy` file constructed in the previous sections produces the following output (lines were wrapped to improve readability):

```
GET /usr/bin/sha1sum /root/tmp
LABEL \nCheck the client's sha1sum program
LOCAL CHECK LOG = local/sha1 /usr/bin/sha1sum /root/tmp/sha1sum
LABEL \nchecking the client's /usr/bin/find program
CHECK LOG = remote/binfind /usr/bin/sha1sum /usr/bin/find
LABEL \nsuid/sgid/executable files uid or gid root on the / partition
CHECK LOG = remote/setuidgid /usr/bin/find / -xdev -perm +u+s,g+s
        \( -user root -or -group root \) -type f
        -exec /usr/bin/sha1sum {} \;
LABEL \nconfiguration files under /etc
CHECK LOG = remote/etcfiles                          /usr/bin/find /etc
        -type f -not -perm /6111   -not -regex "/etc/\(adjtime\|mtab\)"
        -exec /usr/bin/sha1sum {} \;
LOCAL /usr/bin/scp -q root@client:/usr/bin/sha1sum /root/tmp
LABEL \nCheck the client's sha1sum program
LOCAL CHECK LOG = local/sha1 /usr/bin/sha1sum /root/tmp/sha1sum
LABEL \nchecking the client's /usr/bin/find program
CHECK LOG = remote/binfind /usr/bin/sha1sum /usr/bin/find
LABEL \nsuid/sgid/executable files uid or gid root on the / partition
CHECK LOG = remote/setuidgid /usr/bin/find / -xdev -perm +u+s,g+s
        \( -user root -or -group root \) -type f
        -exec /usr/bin/sha1sum {} \;
LABEL \nconfiguration files under /etc
CHECK LOG = remote/etcfiles                          /usr/bin/find /etc
        -type f -not -perm /6111   -not -regex "/etc/\(adjtime\|mtab\)"
        -exec /usr/bin/sha1sum {} \;
```

5.4.1 The mailed report

The `/root/bin/stealthmail` script is called with the following arguments:

```
"Client STEALTH report" admin@elswhere
```

The mailed report contains information comparable to this:

```
STEALTH (4.00.00) started at Sat, 07 Feb 2015 22:10:56 +0100

Check the client's sha1sum program
Initialized report on local/sha1

checking the client's /usr/bin/find program
Initialized report on remote/binfind
```

```
suid/sgid/executable files uid or gid root on the / partition
Initialized report on remote/setuidgid

configuration files under /etc
Initialized report on remote/etcfiles
```

5.4.2 Files under /root/stealth/client

Under `/root/stealth/client` the following entries are now available:

- `local`: below this directory the reports of the locally performed checks are found. Using our demo `policy` file, only one logfile is found here: `sha1`, containing the client's SHA1 checksum of its `/usr/bin/sha1sum` program:

  ```
  45251e259bfaf1951658a7b66c328c52  /root/tmp/sha1sum
  ```

- `remote`: at this directory the reports of the remotely performed checks are found. Using our demo `policy` file, three files were created:

  The file `binfind`, containing the checksum of the client's `/usr/bin/find` program:

  ```
  fc62fc774999584f1e29e0f94279a652  /usr/bin/find
  ```

  The file `etcfiles`, containing the checksums of the client's configuration files under `/etc` (shown only partially):

  ```
  ced739ecb2c43a20053a9f0eb308b2b0  /etc/modutils/aliases
  a2322d7e2f95317b2ddf3543eb4c74c0  /etc/modutils/paths
  f9e3eac60200d41dd5569eeabb4eddff  /etc/modutils/arch/i386
  f07da2ebf00c6ed6649bae5501b84c4f  /etc/modutils/arch/m68k.amiga
  2893201cc7f7556160fa9cd1fb5ba56a  /etc/modutils/arch/m68k.atari
      ...
  bf73b4e76066381cd3caf80369ce1d0e  /etc/deluser.conf
  4cd70d9aee333307a09caa4ef003501d  /etc/adduser.conf.dpkg-save
  8c749353c5027d0065359562d4383b8d  /etc/gimp/1.2/gtkrc_user
  3ec404ec597ef5460600cccf0192f4d6  /etc/gimp/1.2/unitrc
  8c740345b891179228e3d1066291167b  /etc/gimp/1.2/gtkrc
  ```

  The file `setuidgid`, containing the checksums of the client's setuid/setgid root files (shown only partially):

  ```
  030f3f84ec76a8181cca087c4ba655ea  /bin/login
  b6c0209547d88928f391d2bf88af34aa  /bin/ping
  5d324ad212b2ff8f767637ac1a8071ec  /bin/su
  344dbedc398d5114966914419ef53fcc  /usr/bin/wall
  27b045bd7306001f9ea31bc18712d8b7  /usr/bin/rxvt-xpm
  ...
  3567b18ffc39c2dc6ec0c0d0fc483f4f  /usr/lib/ssh-keysign
  3383a7955ac2406311e9aa51c6ac9c2c  /usr/X11R6/bin/X
  3c99ea0425c6e0278039e16478d2fb57  /usr/X11R6/bin/xterm
  d590f7f5b4d6ae61680692a52235d342  /usr/local/bin/setuidcall
  4c17203d7d91ec4946dea2f0ae365d5b  /sbin/unix_chkpwd
  ```

Of course, the checksums and the filenames shown are only for documentation purposes. At other systems different files and/or checksums will be reported.

- The file `/root/client/report` **New lines are always appended to the** `/root/client/report` **file. It will never shorten, unless shorten by the systems administrator at 'monitor'**.

  This file contains the following:

```
STEALTH (3.00.00) started at Wed, 20 Aug 2014 11:06:50 +0000

Check the client's sha1sum program
Initialized report on local/sha1

checking the client's /usr/bin/find program
Initialized report on remote/binfind

suid/sgid/executable files uid or gid root on the / partition
Initialized report on remote/setuidgid

configuration files under /etc
Initialized report on remote/etcfiles
```

This completes the information generated by **stealth** during its first run.

5.5 Subsequent 'stealth' runs

5.5.1 All files unaltered

When **stealth** is subsequently run, it updates its report files under `root/stealth/client`. If nothing has changed, the log-files remain unaltered. Subsequent runs will, however, add some new info to the file `/root/client/report`:

```
STEALTH (4.00.00) started at Sat, 07 Feb 2015 22:10:56 +0100

Check the client's sha1sum program
Initialized report on local/sha1

checking the client's /usr/bin/find program
Initialized report on remote/binfind

suid/sgid/executable files uid or gid root on the / partition
Initialized report on remote/setuidgid

configuration files under /etc
Initialized report on remote/etcfiles

STEALTH (4.00.00) started at Sat, 07 Feb 2015 22:22:15 +0100
```

Note that just one extra line was added: a timestamp showing the date/time of the last run. The systems administrator may rotate the report file every once in a while to reclaim some disk space.

5.5.2 Modifications occurred

Basically, three kinds of modifications are possible: additions, modifications, and removals. Here we'll show the effects all these changes have on **stealth**'s output.

For illustrative purposes, the following changes were made to the `client`'s files:

- `/etc/motd` was changed
- the file `timezone~` was removed
- the file `/etc/motd.org` was created

Next, **stealth** was again run, producing the following output:

- The following new info is now added to file `/root/client/report`:

```
STEALTH (3.00.00) started at Wed, 20 Aug 2014 11:13:38 +0000

configuration files under /etc
ADDED: /etc/motd.org
    < 945d0b8208e9861b8f9f2de155e619f9  /etc/motd.org
MODIFIED: /etc/motd
    < 7f96195d5f051375fe7b523d29e379c1  /etc/motd
    > 945d0b8208e9861b8f9f2de155e619f9  /etc/motd
REMOVED: /etc/timezone~
    > 6322bc8cb3ec53f5eea33201b434b74b  /etc/timezone~
```

  Note that all changes were properly detected and logged in the file `/root/client/report`.

- Furthermore, a matching report was sent by *mail*:

```
STEALTH (3.00.00) started at Wed, 20 Aug 2014 11:13:38 +0000

configuration files under /etc
ADDED: /etc/motd.org
    < 945d0b8208e9861b8f9f2de155e619f9  /etc/motd.org
MODIFIED: /etc/motd
    < 7f96195d5f051375fe7b523d29e379c1  /etc/motd
    > 945d0b8208e9861b8f9f2de155e619f9  /etc/motd
REMOVED: /etc/timezone~
    > 6322bc8cb3ec53f5eea33201b434b74b  /etc/timezone~
```

  Note that the report *only* shows the info that was added to the */root/client/report* file.

  The report itself could be beautified further. E.g., I use the following script to mail the report to the addressee:

```
#!/bin/bash

NAME=‘basename $0‘

tee /root/stealth/lastreport/$NAME | egrep -v ’^([[:space:]]|[[:space:]]*$)’ |
    sort | uniq | mail -s $1 $2
```

For the `client` computer, this little script writes the mailed report on a file `/root/stealth/lastreport/client`, overwriting its previous contents, removes all lines beginning with blanks (thus trimming away the `diff`-generated lines), and e-mails the `sort`ed and `uniq`ed lines using `mail`. The addressee (`admin@elsewhere`) then receives the following information:

```
ADDED: /etc/motd.org
MODIFIED: /etc/motd
REMOVED: /etc/timezone~
STEALTH (3.00.00) started at Wed, 20 Aug 2014 11:13:38 +0000
configuration files under /etc
```

In practice this provides me with all the information I need if something out of the ordinary has happened.

- Finally, the file

  `/root/stealth/client/remote/etcfiles`

  was recreated, saving the old file as

  `/root/stealth/client/remote/etcfiles.20021028-112851`

  As remarked earlier (see section 4.3), many `logfile.YYMMDD-HHMMSS` files could eventually accumulate. As discussed in section 4.3, it might be considered to remove old log files every now and then.

5.5.3 Failing LOCAL commands

If the client's `sha1sum` program itself is altered, a serious situation has developed. In that case, further actions by **stealth** would be suspect, as their results might easily be currupted. Additional checks *will* be performed, but a warning is generated on the `report` file (and in the mail sent to `admin@elsewhere`):

```
STEALTH (4.00.00) started at Sat, 07 Feb 2015 22:27:15 +0100

Check the client's sha1sum program
MODIFIED: /root/tmp/sha1sum
    < fc62fc774999584f1e29e0f94279a652  /root/tmp/sha1sum
    > 45251e259bfaf1951658a7b66c328c52  /root/tmp/sha1sum

*** BE CAREFUL *** REMAINING RESULTS MAY BE FORGED

configuration files under /etc
REMOVED: /etc/motd.org
    > 945d0b8208e9861b8f9f2de155e619f9  /etc/motd.org
MODIFIED: /etc/motd
    < 945d0b8208e9861b8f9f2de155e619f9  /etc/motd
    > 7f96195d5f051375fe7b523d29e379c1  /etc/motd
```

(The report shows the removal of the previously added file `motd.org`, and the modification of `motd`. These are real, as the original `motd` file, modified earlier, was restored at this point).

5.5.4 Skipping (some) integrity checks

Some files or directories may not require integrity checks. Automated processes may modify files which are not threatening the proper functioning of running programs or processes. In those cases a file can be prepared holding the absolute paths of entries to be skipped. Each entry should appear on a line of its own without any additional information.

Stealth can be informed about this file using the `-skip-files <file-spec>` option. When a relative file specification is used with `-skip-files` it is interpreted a location relative to the current working directory. The skip-files file itself must contain the paths of the entries to be skipped during file integrity scans. If an entry is already present in a log file then **stealth** once generates an `IGNORING` message in the mail sent to the address specified at `EMAIL` in the policy file. Each entry in a skip-file must be on a line of its own and must be specified using absolute file paths. Entries ending in a slash are assumed to be directories whose full contents must be skipped. Other entries are interpreted as the names of files to skip. Initial and trailing blanks, empty lines and lines having a `#` as their 1st non blank character are ignored.

Here is an example:

```
stealth -e --skip-files /root/stealth/remote/skipping remote.pol
```

If an entry `/etc/skipme` appears in the current logs which is thereafter added to the `skippath` file then the mail generated by **stealth** once contains a line like the following:

```
SKIPPING: /etc/skipme
    > a7695bb2d019e60988e757a4b692acfe  /etc/skipme
```

The reported hash-value is the hash-value at the time of the stealth-run reporting the `SKIPPING` message.

Entries ending in a slash are assumed to be directories whose contents must (recursively) be skipped.

5.6 Automating repeated 'stealth' runs

To automate **stealth**'s integrity scans, a file `/etc/cron.d/stealth` could be created, containing a line like

```
2,17,32,47 * * * *  root    test -x /usr/bin/stealth && \
                        /usr/bin/stealth /root/stealth/client.pol
```

This starts **stealth** 2 minutes after every hour. In this example the ssh-key must not require a passphrase, as **crontab**(1) cannot provide passphrases of ssh-keys. Ssh-keys requiring passphrases can, however, be used if repeated **stealth** runs are controlled by a program like **ssh-cron**(1).

In general, randomized events are harder to notice. For this **stealth** offers the `-repeat` and `-random-interval` options. Both options expect an argument in seconds (or in minutes, if an `m` is appended to the specification). After each integrity scan the next integrity scan starts after the time interval specified by the `-repeat` option plus a random time value selected from the time interval specified by the `-random-interval` option. For example, the **stealth** daemon started by the following command repeatedly performs integrity scans between two and five minutes after the last integrity scan completed:

```
stealth --daemon /root/stealth/client.uds \
        --repeat 2m --random-interval 3m /root/stealth/client.pol
```

Once this program has started its **ssh**(1) connection to the client host persists, and a possibly required ssh-passphrase is no longer required. Additional (intermediate) integrity scans can always be requested (also without requiring ssh-passphrase specifications) using the command

```
stealth --rerun /root/stealth/client.uds
```

5.7 Report File Rotation

When **stealth** performs integrity scans it appends information to the report file. This file therefore eventually grows to a large size, and the systems manager controlling **stealth** might want to *rotate* the report file every once in a while (e.g., using a program like **logrotate**(1), also see the upcoming section 5.7.2). To ensure that no log-rotation takes place while **stealth** is busy performing integrity scans (thus modifying the report file) the options –**suspend** and `-resume` were implemented. Both options require the process-ID file of currently active **stealth** process as their argument.

For example, if a **stealth** process was once started using the command

```
stealth --daemon /root/stealth/small.uds --repeat 900 \
                /root/stealth/small.pol
```

then the `-suspend` and `-resume` commands for this process should be called as:

```
stealth --suspend /root/stealth/small.uds
stealth --resume /root/stealth/small.uds
```

The **stealth** process identified in the files provided as arguments to the `-suspend` and `-resume` options is called the *daemon stealth process* below.

The `-suspend` option has the following effect:

- If the daemon **stealth** process is currently processing its policy file, performing an integrity scan, then the currently executing policy file command is completed, whereafter further commands are ignored, except for `-resume` (see below) and `-terminate`.
- Any scheduled integrity scans following the `-suspend` command are ignored by the daemon **stealth** process;
- The daemon **stealth** process writes a message that it is being suspended to the report file and then processes the report file as usual.

Now that the report file is no longer modified by the daemon **stealth** process, log-rotation may take place. E.g., a program like **logrotate**(1) allows its users to specify a command or script just before log-rotation takes place, and '**stealth** `-suspend udsfile`' could be specified nicely in such a pre-rotation section.

The `-resume` option has the following effect:

- The daemon **stealth** process resumes its activities by performing another integrity scan. Thus, `-resume` implies `-rerun`.
- Any scheduled integrity scans following the `-resume` command are again honored by the daemon **stealth** process.

Note that, once `-suspend` has been issued, all commands except `-resume` and `-terminate` are ignored by the daemon **stealth** process. While suspended, the `-terminate` command is acknowledged as a 'emergency exit' which may or may not interfere with, e.g., an ongoing log-rotation process. The daemon **stealth** process should not normally be terminated while it is in its suspended mode. The normal way to terminate a stealth process running in the background is:

- Wait for the daemon **stealth** process to complete an ongoing series of integrity scan commands;
- Issue the '**stealth** `-terminate udsfile`' command.

5.7.1 Status file cleanup

Whenever **stealth** is run and it encounters a modified situation the already existing status file summarizing that particular situation is saved and a new status file is created. Eventually, this will result in many status files. While report files can be rotated, it is pointless to rotate old status files, since they are never modified. Instead, status files exceeding a certain age could be removed and more recent files might be zipped to conserve space. In **stealth**'s binary distribution the file `/usr/share/doc/stealth/usr/bin/stealthcleanup` is provided which can be used to perform such a cleanup. The script expects one argument: a resource file defining the following shell variables:

- `directories`: the directories below which the status files are found;
- `gzdays`: the number of days a status file must exist before it is compressed using **gzip**(1);
- `rmdays`: the maximum age (in days) of compressed status files. Files exceeding this age are removed using **rm**(1).

Here is the `stealthcleanup` script as contained in the binary distribution's `/usr/share/doc/stealth/usr/bin` directory:

```
#!/bin/bash

usage()
{
    echo "
Usage: $0 [-v] rc-file
Where:
    -v: Show the actions that are performed
    rc-file: resource file defining:
            \'directories' - one or more directories containing status files
            \'gzdays'      - number of days status files may exist before they
                             are compressed
            \'rmdays'      - number of days gzipped status files may exist
                             before they are removed.
"
    exit 1
}
```

```
error()
{
    echo "$*" >&2
    exit 1
}

if [ "$1" == "-v" ]
then
    verbose=1
    shift 1
else
    verbose=0
fi

[ $# == 1 ] || usage

# now source the configuration file
. $1

for x in $directories
do
    cd $x || error "\‘$x’ must be a directory"
    if [ $verbose -eq 1 ]
    then
        echo "
cd $x"
    fi

    if [ $verbose -eq 1 ]
    then
        echo \
    /usr/bin/find ./ -mtime +$rmdays -type f -regex '.*[0-9]+-[0-9]+\.gz' \
        -exec /bin/rm {} \;
    fi
    /usr/bin/find ./ -mtime +$rmdays -type f -regex '.*[0-9]+-[0-9]+\.gz' \
        -exec /bin/rm {} \;

    if [ $verbose -eq 1 ]
    then
        echo \
    /usr/bin/find ./ -mtime +$gzdays -type f -regex '.*[0-9]+-[0-9]+' \
        -exec /bin/gzip {} \;
    fi
    /usr/bin/find ./ -mtime +$gzdays -type f -regex '.*[0-9]+-[0-9]+' \
        -exec /bin/gzip {} \;
done

exit 0
```

Assuming that the status files are written in `/var/stealth/target/local` and `/var/stealth/target/remote`; that status file should be compressed when older than 2 days and removed after 30 days, the resource file is:

```
directories="
    /var/stealth/target/local
    /var/stealth/target/remote
    "

rmdays=30
gzdays=3
```

Furthermore assuming that the resourcefile is installed in `/etc/stealth/cleanup.rc` and the `stealthcleanup` script itself in `/usr/bin/stealthcleanup`, the `stealthcleanup` script could be called as follows:

```
    /usr/bin/stealthcleanup /etc/stealth/cleanup.rc
```

Note that `stealthcleanup` may be called whether or not there are active **stealth** processes.

5.7.2 Using 'logrotate' to control report- and status files

A program like **logrotate**(1) allows its users to specify a command or script immediately following log-rotation, and '**stealth** `-resume pidfile`' could be specified nicely in such a post-rotation section.

Here is an example of a specification that can be used with **logrotate**(1). Logrotate (on Debian systems) keeps its configuration files in `/etc/logrotate.d`, and assuming there is a host `target`, whose report file is `/var/stealth/target/report`, the required **logrotate**(1) specification file (e.g., `/etc/logrotate.d/target`) could be:

```
/root/stealth/report /var/log/stealth/client-small.log {
    weekly
    rotate 12
    compress
    missingok
    copytruncate
    sharedscripts
    prerotate
        /usr/bin/stealth --suspend /root/stealth/small.uds
    endscript
    postrotate
        /usr/bin/stealth --resume /root/stealth/small.uds
    endscript
}
```

Using this specification file, **logrotate**(1) will

- perform weekly rotations of the report file;
- keep up to 12 rotated files, compressing them using **gzip**(1);

- suspend the **stealth** daemon, before rotating its report file; suspended;
- following the rotation, **stealth**'s actions are resumed.

Note thet `stealth -resume xxx` always initiates another file integrity scan.

Chapter 6

Kick-starting 'stealth'

Here are the steps to take to kick-start **stealth**

- Install the stealth Debian package `stealth_4.01.02_i386.deb` and thus accept the provided binary program (skipping the next series of steps) or do not accept the provided binary, and compile **stealth** yourself, as per the following steps:
- Unpack `stealth_4.01.02.tar.gz`: `tar xzvf tealth_4.01.02.tar.gz`
- `cd stealth`;
- Inspect, and where necessary modify the values of the variables in the files `INSTALL.cf` and `icmconf`;
- Install a recent Gnu `g++` compiler;
- Install the bobcat library (both the binary and development version) (`https://fbb-git.github.io/bobcat/`);
- Install the `icmake` program (`https://fbb-git.github.io/icmake/`);
- Run '`./build program strip`' to compile **stealth**;
- Run (probably as root) '`./build install program`' to install;
- Optionally install documentation. See section 2.1.

Following the installation the `stealth` directory tree has become superfluous and can safely be removed.

Next, do:

- `cp share/usr/bin/stealthmail /usr/local/bin`
- `mkdir /root/stealth`
- `cp documentation/example-policies/localhost.pol /root/stealth`

`ssh` and `bash` (or another shell program) should be available. `root@localhost` should be able to login at `localhost` using `ssh root@localhost`, using the `/bin/bash`. Check (as 'root') at least

```
ssh root@localhost
```

as this might ask you for a confirmation that you've got the correct host. Now, run

```
stealth /root/stealth/localhost.pol
```

to initialize the stealth-report files for `localhost`. This initializes the report for

- all root setuid/setgid executable files on `localhost`,
- and for all files under `/etc/` on `localhost`.

The mail-report is sent to `root@localhost`.

Now change or add or remove one of these files, and rerun **stealth**. The file `/tmp/stealth-4.01.02.mail` should reflect these changes.

Chapter 7

Usage info

When **stealth** is started without arguments, it provides some help about how to start it. A message like the following is produced:

```
stealth V4.01.02
SSH-based Trust Enhancement Acquired through a Locally Trusted Host
Copyright (c) GPL 2005-2015

Usage 1 (activation modes):
   stealth [options] policy
Where:
   [options] - optional arguments (short options between parentheses,
               option descriptions starting with (C) can only be used
               on the command-line and are ignored when specified in the
               policy file).
      --daemon (-d) <uds>: (C) run as a background (daemon) process.
               <uds> is the location of the Unix Domain Socket that is
               used for communication with the stealth daemon process
      --dry-run: (C) no integrity scans or reloads are performed, but
               are assumed OK. Remaining tasks are normally performed
      --log (-L) <file-spec>: log messages are appended to '<file-spec>'.
               If <file-spec> does not exist, it is first created
      --logmail: mail is logged (requires --log or --syslog)
      --max-size value[BKMG]: files retrieved by GET may at most
               have 'value' bytes (B), Kbytes (K), Mbytes (M), Gbytes (G).
               By default: 10M; The default unit is 'B'
      --no-mail: mail is not sent. By default mail is sent as configured
               in the policy-file (--logmail can be specified independently
               from --no-mail)
      --parse-policy-file (-p): (C) parse the policy file, no further actions.
               Specify once to see the numbered commands,
               twice to see the policy file parsing steps as well.
               Results to std output.
      --random-interval (-i) value: start integrity scans within
               a random interval of 'value' seconds (minutes if an 'm'
               is appended to the specified value).
               Requires --repeat.
      --repeat value: start an integrity scan every 'value' seconds
```

```
            (minutes if an 'm' is appended to the specified value).
      --run-command (-r) value: (C) only execute command #'value'
            (not available in combination with --daemon)
      --skip-files (-s) <file-spec>: skip the integrity checks of the
            files having their absolute path names listed in '<file-spec>'
      --stdout (-o): messages are (also) written to stdout
            (not available in combination with --daemon)
      --syslog: write syslog messages
      --syslog-facility fac:  syslog facility to use. By default DAEMON
      --syslog-priority pri:  syslog priority to use. By default NOTICE
      --syslog-tag tag: identifier prefixed to syslog messages. By
            default 'STEALTH')
      --time-stamp <type>: the time-stamps to use. By default UTC.
            (does not apply to syslog-timestamps)
      --verbosity (-V) value: determines the amount of logged information.
            Requires --log or --syslog:
                0: nothing is logged
                1: mode reports and policy commands
                2: also: ipc commands and actions
                3: also: integrity scan informative messages
   policy: location of the policy file

Usage 2 (IPC modes, all options are command-line only):
   stealth {--reload,--rerun,--resume,--suspend,--terminate} <uds>
Where:
      --reload: reload a stealth process's policy and skip-file
                files
      --rerun: start an integrity scan
      --resume: resume stealth after --suspend
      --suspend: suspend stealth's activities
                to continue: --resume; to end: --terminate
      --terminate: terminate the stealth daemon
   <uds>: location of the Unix Domain Socket to connect to.

Usage 3 (support mode, all options are command-line only)
   stealth {--help,--version}
Where:
      --help (-h):    provide this help and terminate
      --version (-v): show version information and terminate
```

Note that with the second type of usage the policy file is not required: here only the `pidfile` must be specified.

Chapter 8

Errormessages

`Can't chdir to 'path'`

the directory `path` could not be created/used. This may be a permission problem. Check the permissions of `path` if `path` does actually exist. The problem may be in a path component, not necessarily in the last element of the path.

`Can't open '<fname>' to read (or write)`

When a `GET` or `PUT` command fails because the target file could not be read or written, **stealth** terminates afer logging this message. The file may not exist or you may not not have sufficient permissions to read or write it.

`Corrupt line in policy file: ...`

The apparently corrupted line is shown. The line is corrupted if the line could not be split into an initial word and its remainder. Normally this should not happen. As the line is mentioned, the message itself should assist you in your repairs.

`could not open <logname>`

This message is generated when the mentioned log-file cannot be written to. Check the permissions of the file, and check if the path to the file exists. The problem may be in a path component, not necessarily in the last element of the path or in the file itself.

`-daemon: missing uds-file or policy file`

The `-daemon` option requires the location for its Unix Domain Socket, and in addition a policy file must be specified. Most likely the location of the Unix Domain File was omitted, thus confusing the policy file with the Unix Domain File

`incompatible options: ...`

Two or more incompatible options are shown. E.g., `-daemon` is incompatible with IPC options, which themselves are mutually incompatible. Omit all but one of the mentioned options and restart **stealth**

`Inserting command '...' failed.`

the mentioned command could not be sent to a child-process (`sh` or `ssh`). Check the availability of the `ssh` connection to the client, and whether you have permissions to execute the specified command.

`Invalid -random-interval specified`

The –random-interval option was given an invalid (too large or negative) argument.

`invalid -<option> specified`

The specified option value (e.g., `-max-size` was specified, but its option value is not a valid integral number) is not supported

`LocalClientSocket::connect(): ...`

When running in IPC-mode **stealth** could not connect to the unix domain socket specified for the IPC-mode. Check whether the matching **stealth** daemon exists, and whether the location of the uds-file was correctly specified

`LocalServerSocket::accept(): ...`

When running in daemon mode incoming connections could not be accepted. This may have many causes, and a probably cause is mentioned with the error message

`LocalServerSocket::listen(): ...`

When running in daemon mode the unix domain socket failed to listen for incoming connections. This may have many causes, and a probably cause is mentioned with the error message

`LocalServerSocket::open(<uds>): ...`

When running in daemon mode the unix domain socket could not be opened. A possible reason is that an entry in the file system having the same name already exists. The error message will mention a probably cause of the failure

`-max-size incompatible with IPC calls`

The `-max-size` option can can only be used when **stealth** is also receiving a policy file. It cannot be used in combination with the Inter Process Communication (IPC) options `-reload, -rerun, -resume, -suspend` or `-terminate`.

`-max-size <value>: invalid option value`

An invalid specification for `-max-size` was received. Refer to the man-page or manual for information about valid options.

`No exit value for <cmd> ...`

No exit value was received for the logged command, after which **stealth** terminates.

`-<option> is only valid for a s() foreground process`

The indicated option cannot be specified in combination with a `-daemon` or IPC-option (like `-suspend`)

`-random-interval requires -repeat`

The `-random-interval` option can only be used when the `-repeat` option has also been specified.

`-repeat not available in IPC modes`

The `-repeat` option cannot be specified in combination with an IPC-option (like `-suspend`)

`-run-command 0:  not a valid (natural) command number`

The `-run-command` option requires an argument, which is a positive, integral value.

`-run-command <nr>:  invalid command number`

The `-run-command` option requires an argument, which is a positive, integral value at most equal to the number of commands listed in the policy file.

`-skip-files incompatible with IPC calls`

The `-skip-files` option can only be used when **stealth** is also given a policy file. It cannot be used in combination with IPC options (`-reload, -rerun, -resume, -suspend` or `-terminate`).

`-skip-files:  missing skip-file or policy file`

The `-skip-files` option requires a file name and the **stealth** command also requires a policy file. Most likely the name of the skip-file was not provided, thus confusing the policy file with the skip-file.

`-stdout incompatible with -daemon`

The `-stdout` option can only be used when **stealth** is not started as a daemon process.

`syslog facility <facility> not supported`

The option `-syslog-facility` requires the name of a standard **syslog**(1) facility. Supported facilities are `DAEMON, LOCAL0` through `LOCAL7` and `USER`. See, e.g., the **syslog**(3) man-page for an overview of their definitions.

`-syslog* options incompatible with IPC calls`

`-syslog*` options can can only be used when **stealth** is also receiving a policy file. It cannot be used in combination with the Inter Process Communication (IPC) options `-reload, -rerun, -resume, -suspend` or `-terminate`.

`syslog priority <priority> not supported`

The option `-syslog-priority` requires the name of a standard **syslog**(1) priority. All standard priorities are supported. See, e.g., the **syslog**(3) man-page for an overview of available priorities. The prefixes used with the priorities in this man-page (i.e., `LOG_` should not be used when specifying the `-syslog-priority`. E.g., use `-syslog-priority WARNING` rather than `-syslog-priority LOG-WARNING`)

`terminated: non-zero exit value for '...'`

A local command (not using the `CHECK` keyword), returned with a non-zero exit. This will terminate further processing of the policy file. Inspect and/or rerun the command 'by hand' to find indications about what went wrong. The report file or the standard error stream may also contain additional information about the reason of the failure.

`Unable to create the logfile '...'`

the mentioned log file could not be created. Check the permissions of the file, check if the path to the file exists. The problem may be in a path component, not necessarily in the last element of the path or in the file itself.

`Unix Domain Socket '<uds>': already in use, remove it first`

The intended Unix Domain Socket name (`<uds>`) is already in use. It could be a file that is or has been used by another process. Either use another name for the unix domain socket, or remove the existing file-system entry

`USE SSH ... entry missing in the policy file`

there is no default for the `USE SSH` specification in the policy file. The specification could not be found. Provide a specification like:

```
USE SSH     ssh -q root@localhost
```