Clam AntiVirus 0.100.1

User Manual

Contents

Introduction

Clam AntiVirus is an open source (GPL) anti-virus toolkit for UNIX, designed especially for e-mail scanning on mail gateways. It provides a number of utilities including a flexible and scalable multi-threaded daemon, a command line scanner and advanced tool for automatic database updates. The core of the package is an anti-virus engine available in a form of shared library.

Features

- Licensed under the GNU General Public License, Version 2
- POSIX compliant, portable
- Fast scanning
- Supports on-access scanning (Linux only)
- Detects over 1 million viruses, worms and trojans, including Microsoft Office macro viruses, mobile malware, and other threats
- Built-in bytecode interpreter allows the ClamAV signature writers to create and distribute very complex detection routines and remotely enhance the scanner's functionality
- Scans within archives and compressed files (also protects against archive bombs), built-in support includes:
  - Zip (including SFX)
  - RAR (including SFX)
  - 7Zip
  - ARJ (including SFX)
  - Tar
  - CPIO
  - Gzip
  - Bzip2
  - DMG
  - IMG
  - ISO 9660

  - PKG
  - HFS+ partition
  - HFSX partition
  - APM disk image
  - GPT disk image
  - MBR disk image
  - XAR
  - XZ
  - MS OLE2
  - MS Cabinet Files (including SFX)
  - MS CHM (Compiled HTML)
  - MS SZDD compression format
  - BinHex
  - SIS (SymbianOS packages)
  - AutoIt
  - InstallShield
- Supports Portable Executable (32/64-bit) files compressed or obfuscated with:
  - AsPack
  - UPX
  - FSG
  - Petite
  - PeSpin
  - NsPack
  - wwpack32
  - MEW
  - Upack
  - Y0da Cryptor
- Supports ELF and Mach-O files (both 32- and 64-bit)
- Supports almost all mail file formats
- Support for other special files/formats includes:

  - HTML
  - RTF
  - PDF
  - Files encrypted with CryptFF and ScrEnc
  - uuencode
  - TNEF (winmail.dat)
- Advanced database updater with support for scripted updates, digital signatures and DNS based database version queries

Mailing lists and IRC channel

If you have a trouble installing or using ClamAV try asking on our mailing lists. There are four lists available:

- **clamav-announce*lists.clamav.net** - info about new versions, moderated[1].
- **clamav-users*lists.clamav.net** - user questions
- **clamav-devel*lists.clamav.net** - technical discussions
- **clamav-virusdb*lists.clamav.net** - database update announcements, moderated

You can subscribe and search the mailing list archives at: `https://www.clamav.net/contact.html#ml`
Alternatively you can try asking on the `#clamav` IRC channel - launch your favourite irc client and type:

```
/server irc.freenode.net
/join #clamav
```

Virus submitting

If you have got a virus which is not detected by your ClamAV with the latest databases, please submit the sample at our website:

`https://www.clamav.net/reports/malware`

[1] Subscribers are not allowed to post to the mailing list

Base package

Supported platforms

Clam AntiVirus is regularly tested on:

- GNU/Linux
- Solaris
- FreeBSD
- macOS
- Windows

Binary packages

You can find the up-to-date list of binary packages at our website: `https://www.clamav.net/download.html#otherversions`

Installation

Requirements

The following components are required to compile ClamAV under UNIX: [2]

- zlib and zlib-devel packages
- openssl version 0.9.8 or higher and libssl-devel packages
- gcc compiler suite (tested with 2.9x, 3.x and 4.x series)
  If you are compiling with higher optimization levels than the default one (-O2 for gcc), be aware that there have been reports of misoptimizations. The build system of ClamAV only checks for bugs affecting the default settings, it is your responsibility to check that your compiler version doesn't have any bugs.
- GNU make (gmake)

The following packages are optional but **highly recommended**:

[2] For Windows instructions please see win32/README in the main source code directory.

- bzip2 and bzip2-devel library
- libxml2 and libxml2-dev library
- `check` unit testing framework [3].

The following packages are optional, but **required for bytecode JIT support**: [4]

- GCC C and C++ compilers (minimum 4.1.3, recommended 4.3.4 or newer) the package for these compilers are usually called: gcc, g++, or gcc-c++. [5]
- OSX Xcode versions prior to 5.0 use a g++ compiler frontend (llvm-gcc) that is not compatible with ClamAV JIT. It is recommended to either compile ClamAV JIT with clang++ or to compile ClamAV without JIT.
- A supported CPU for the JIT, either of: X86, X86-64, PowerPC, PowerPC64

The following packages are optional, but needed for the JIT unit tests:

- GNU Make (version 3.79, recommended 3.81)
- Python (version 2.5.4 or newer), for running the JIT unit tests

The following packages are optional, but required for clamsubmit:

- libcurl-devel library
- libjson-c-dev library

Installing on shell account

To install ClamAV locally on an unprivileged shell account you need not create any additional users or groups. Assuming your home directory is `/home/gary` you should build it as follows:

```
$ ./configure --prefix=/home/gary/clamav --disable-clamav
$ make; make install
```

To test your installation execute:

[3] See section 3.7 on how to run the unit tests

[4] if not available ClamAV will fall back to an interpreter

[5] Note that several versions of GCC have bugs when compiling LLVM, see `http://llvm.org/docs/GettingStarted.html#brokengcc` for a full list.

```
$ ~/clamav/bin/freshclam
$ ~/clamav/bin/clamscan ~
```

The `--disable-clamav` switch disables the check for existence of the *clamav* user and group but `clamscan` would still require an unprivileged account to work in a superuser mode.

Adding new system user and group

If you are installing ClamAV for the first time, you have to add a new user and group to your system:

```
# groupadd clamav
# useradd -g clamav -s /bin/false -c "Clam AntiVirus" clamav
```

Consult a system manual if your OS has not *groupadd* and *useradd* utilities. **Don't forget to lock access to the account!**

Compilation of base package

Once you have created the clamav user and group, please extract the archive:

```
$ zcat clamav-x.yz.tar.gz | tar xvf -
$ cd clamav-x.yz
```

Assuming you want to install the configuration files in /etc, configure and build the software as follows:

```
$ ./configure --sysconfdir=/etc
$ make
$ su -c "make install"
```

In the last step the software is installed into the /usr/local directory and the config files into /etc. **WARNING: Never enable the SUID or SGID bits for Clam AntiVirus binaries.**

Compilation with clamav-milter enabled

libmilter and its development files are required. To enable clamav-milter, configure ClamAV with

```
$ ./configure --enable-milter
```

See section /refsec:clamavmilter for more details on clamav-milter.

Using the system LLVM

Some problems have been reported when compiling ClamAV's built-in LLVM with recent C++ compiler releases. These problems may be avoided by installing and using an external LLVM system library. To configure ClamAV to use LLVM that is installed as a system library instead of the built-in LLVM JIT, use following:

```
$ ./configure --with-system-llvm=/myllvm/bin/llvm-config
$ make
$ sudo make install
```

The argument to `--with-system-llvm` is optional, indicating the path name of the LLVM configuration utility (llvm-config). With no argument to `--with-system-llvm`, `./configure` will search for LLVM in /usr/local/ and then /usr.

Recommended versions of LLVM are 3.2, 3.3, 3.4, 3.5, and 3.6. Some installations have reported problems using earlier LLVM versions. Versions of LLVM beyond 3.6 are not currently supported in ClamAV.

Running unit tests

ClamAV includes unit tests that allow you to test that the compiled binaries work correctly on your platform.

The first step is to use your OS's package manager to install the `check` package. If your OS doesn't have that package, you can download it from `http://check.sourceforge.net/`, build it and install it.

To help clamav's configure script locate `check`, it is recommended that you install `pkg-config`, preferably using your OS's package manager, or from `http://pkg-config.freedesktop.org`.

The recommended way to run unit-tests is the following, which ensures you will get an error if unit tests cannot be built: [6]

```
$ ./configure --enable-check
$ make
$ make check
```

When `make check` is finished, you should get a message similar to this:

```
==================
All 8 tests passed
==================
```

If a unit test fails, you get a message similar to the following. Note that in older versions of make check may report failures due to the absence of optional packages. Please make sure you have the latest versions of the components noted in section /refsec:components. See the next section on how to report a bug when a unit test fails.

```
========================================
1 of 8 tests failed
Please report to https://bugzilla.clamav.net/
========================================
```

If unit tests are disabled (and you didn't use –enable-check), you will get this message:

```
*** Unit tests disabled in this build
*** Use ./configure --enable-check to enable them

SKIP: check_clamav
PASS: check_clamd.sh
PASS: check_freshclam.sh
PASS: check_sigtool.sh
PASS: check_clamscan.sh
======================
All 4 tests passed
(1 tests were not run)
======================
```

Running `./configure --enable-check` should tell you why.

[6] The configure script in ClamAV automatically enables the unit tests, if it finds the check framework, however it doesn't consider it a fatal error if unit tests cannot be enabled.

Reporting a unit test failure bug

If `make check` says that some tests failed we encourage you to report a bug on our bugzilla: `https://bugzilla.clamav.net`. The information we need is:

- The exact output from `make check`
- Output of `uname -mrsp`
- your `config.log`
- The following files from the `unit_tests/` directory:
  - `test.log`
  - `clamscan.log`
  - `clamdscan.log`
- `/tmp/clamd-test.log` if it exists
- where and how you installed the check package
- Output of `pkg-config check --cflags --libs`
- Optionally if `valgrind` is available on your platform, the output of the following:

```
$ make check
$ CK_FORK=no ./libtool --mode=execute valgrind unit_tests/check_clamav
```

Obtain Latest ClamAV anti-virus signature databases

Before you can run ClamAV in daemon mode (clamd), 'clamdscan', or 'clamscan' which is ClamAV's command line virus scanner, you must have ClamAV Virus Database (.cvd) file(s) installed in the appropriate location on your system. The default location for these database files are /usr/local/share/clamav (in Linux/Unix).

Here is a listing of currently available ClamAV Virus Database Files:

- bytecode.cvd (signatures to detect bytecode in files)
- main.cvd (main ClamAV virus database file)
- daily.cvd (daily update file for ClamAV virus databases)
- safebrowsing.cvd (virus signatures for safe browsing)

These files can be downloaded via HTTP from the main ClamAV website or via the 'freshclam' utility on a periodic basis. Using 'freshclam' is the preferred method of keeping the ClamAV virus database files up to date without manual intervention (see section 4.4 for information on how to configure 'freshclam' for automatic updating and section 5.7 for additional details on freshclam).

Configuration

Before proceeding with the steps below, you should run the 'clamconf' command, which gives important information about your ClamAV configuration. See section 5.8 for more details.

clamd

Before you start using the daemon you have to edit the configuration file (in other case `clamd` won't run):

```
$ clamd
ERROR: Please edit the example config file /etc/clamd.conf.
```

This shows the location of the default configuration file. The format and options of this file are fully described in the *clamd.conf(5)* manual. The config file is well commented and configuration should be straightforward.

On-access scanning

One of the interesting features of `clamd` is on-access scanning based on fanotify, included in Linux since kernel 2.6.36. **This is not required to run clamd**. At the moment the fanotify header is only available for Linux.

Configure on-access scanning in `clamd.conf` and read the 5.3 section for on-access scanning usage.

clamav-milter

ClamAV $\geq$ 0.95 includes a new, redesigned clamav-milter. The most notable difference is that the internal mode has been dropped and now a working clamd companion is required. The second important difference is that now the milter has got its own configuration and log files.

To compile ClamAV with the clamav-milter just run `./configure --enable-milter` and make as usual. In order to use the '–enable-milter' option with 'configure', your system MUST have the milter library installed. If you use the '–enable-milter' option without the library being installed, you will most likely see output like this during 'configure':

```
checking for libiconv_open in -liconv... no
checking for iconv... yes
checking whether in_port_t is defined... yes
checking for in_addr_t definition... yes
checking for mi_stop in -lmilter... no
checking for library containing strlcpy... no
checking for mi_stop in -lmilter... no
configure: error: Cannot find libmilter
```

At which point the 'configure' script will stop processing.

Please consult your MTA's manual on how to connect ClamAV with the milter.

Testing

Try to scan recursively the source directory:

```
$ clamscan -r -l scan.txt clamav-x.yz
```

It should find some test files in the clamav-x.yz/test directory. The scan result will be saved in the `scan.txt` log file [7]. To test `clamd`, start it and use `clamdscan` (or instead connect directly to its socket and run the SCAN command):

```
$ clamdscan -l scan.txt clamav-x.yz
```

Please note that the scanned files must be accessible by the user running `clamd` or you will get an error.

Setting up auto-updating

`freshclam` is the automatic database update tool for Clam AntiVirus. It can work in two modes:

[7] To get more info on clamscan options run 'man clamscan'

- interactive - on demand from command line
- daemon - silently in the background

`freshclam` is advanced tool: it supports scripted updates (instead of transferring the whole CVD file at each update it only transfers the differences between the latest and the current database via a special script), database version checks through DNS, proxy servers (with authentication), digital signatures and various error scenarios. **Quick test: run freshclam (as superuser) with no parameters and check the output.** If everything is OK you may create the log file in /var/log (owned by *clamav* or another user `freshclam` will be running as):

```
# touch /var/log/freshclam.log
# chmod 600 /var/log/freshclam.log
# chown clamav /var/log/freshclam.log
```

Now you *should* edit the configuration file `freshclam.conf` and point the *UpdateLogFile* directive to the log file. Finally, to run `freshclam` in the daemon mode, execute:

```
# freshclam -d
```

The other way is to use the *cron* daemon. You have to add the following line to the crontab of **root** or **clamav** user:

```
N * * * * /usr/local/bin/freshclam --quiet
```

to check for a new database every hour. **N should be a number between 3 and 57 of your choice. Please don't choose any multiple of 10, because there are already too many clients using those time slots.** Proxy settings are only configurable via the configuration file and `freshclam` will require strict permission settings for the config file when `HTTPProxyPassword` is turned on.

```
HTTPProxyServer myproxyserver.com
HTTPProxyPort 1234
HTTPProxyUsername myusername
HTTPProxyPassword mypass
```

Closest mirrors

The `DatabaseMirror` directive in the config file specifies the database server `freshclam` will attempt (up to `MaxAttempts` times) to download the database from. The default database mirror is `database.clamav.net` but multiple directives are allowed. In order to download the database from the closest mirror you should configure `freshclam` to use `db.xx.clamav.net` where xx represents your country code. For example, if your server is in "Ascension Island" you should have the following lines included in `freshclam.conf`:

```
DNSDatabaseInfo current.cvd.clamav.net
DatabaseMirror db.ac.clamav.net
DatabaseMirror database.clamav.net
```

The second entry acts as a fallback in case the connection to the first mirror fails for some reason. The full list of two-letters country codes is available at `http://www.iana.org/cctld/cctld-whois.htm`

Usage

Clam daemon

`clamd` is a multi-threaded daemon that uses *libclamav* to scan files for viruses. It may work in one or both modes listening on:

- Unix (local) socket
- TCP socket

The daemon is fully configurable via the `clamd.conf` file [8]. `clamd` recognizes the following commands:

- **PING**
  Check the daemon's state (should reply with "PONG").
- **VERSION**
  Print program and database versions.
- **RELOAD**
  Reload the databases.

[8] man 5 clamd.conf

- **SHUTDOWN**
  Perform a clean exit.

- **SCAN file/directory**
  Scan file or directory (recursively) with archive support enabled (a full path is required).

- **RAWSCAN file/directory**
  Scan file or directory (recursively) with archive and special file support disabled (a full path is required).

- **CONTSCAN file/directory**
  Scan file or directory (recursively) with archive support enabled and don't stop the scanning when a virus is found.

- **MULTISCAN file/directory**
  Scan file in a standard way or scan directory (recursively) using multiple threads (to make the scanning faster on SMP machines).

- **ALLMATCHSCAN file/directory**
  ALLMATCHSCAN works just like SCAN except that it sets a mode where, after finding a virus within a file, continues scanning for additional viruses.

- **INSTREAM**
  *It is mandatory to prefix this command with **n** or **z**.*
  Scan a stream of data. The stream is sent to clamd in chunks, after INSTREAM, on the same socket on which the command was sent. This avoids the overhead of establishing new TCP connections and problems with NAT. The format of the chunk is: `<length><data>` where `<length>` is the size of the following data in bytes expressed as a 4 byte unsigned integer in network byte order and `<data>` is the actual chunk. Streaming is terminated by sending a zero-length chunk. Note: do not exceed StreamMaxLength as defined in clamd.conf, otherwise clamd will reply with *INSTREAM size limit exceeded* and close the connection.

- **FILDES**
  *It is mandatory to newline terminate this command, or prefix with **n** or **z**. This command only works on UNIX domain sockets.*
  Scan a file descriptor. After issuing a FILDES command a subsequent rfc2292/bsd4.4 style packet (with at least one dummy character) is sent to clamd carrying the file descriptor to be scanned inside the ancillary data. Alternatively the file descriptor may be sent in the same packet, including the extra character.

- **STATS**
  *It is mandatory to newline terminate this command, or prefix with **n** or **z**, it is*

*recommended to only use the **z** prefix.*
On this command clamd provides statistics about the scan queue, contents of scan queue, and memory usage. The exact reply format is subject to changes in future releases.

- **IDSESSION, END**
  *It is mandatory to prefix this command with **n** or **z**, also all commands inside **IDSESSION** must be prefixed.*
  Start/end a clamd session. Within a session multiple SCAN, INSTREAM, FILDES, VERSION, STATS commands can be sent on the same socket without opening new connections. Replies from clamd will be in the form `<id>: <response>` where `<id>` is the request number (in ASCII, starting from 1) and `<response>` is the usual clamd reply. The reply lines have the same delimiter as the corresponding command had. Clamd will process the commands asynchronously, and reply as soon as it has finished processing. Clamd requires clients to read all the replies it sent, before sending more commands to prevent send() deadlocks. The recommended way to implement a client that uses IDSESSION is with non-blocking sockets, and a select()/poll() loop: whenever send would block, sleep in select/poll until either you can write more data, or read more replies. *Note that using non-blocking sockets without the select/poll loop and alternating recv()/send() doesn't comply with clamd's requirements.* If clamd detects that a client has deadlocked, it will close the connection. Note that clamd may close an IDSESSION connection too if the client doesn't follow the protocol's requirements.

- **STREAM** (deprecated, use **INSTREAM** instead)
  Scan stream: clamd will return a new port number you should connect to and send data to scan.

It's recommended to prefix clamd commands with the letter **z** (eg. zSCAN) to indicate that the command will be delimited by a NULL character and that clamd should continue reading command data until a NULL character is read. The null delimiter assures that the complete command and its entire argument will be processed as a single command. Alternatively commands may be prefixed with the letter **n** (e.g. nSCAN) to use a newline character as the delimiter. Clamd replies will honour the requested terminator in turn. If clamd doesn't recognize the command, or the command doesn't follow the requirements specified below, it will reply with an error message, and close the connection. Clamd can handle the following signals:

- **SIGTERM** - perform a clean exit
- **SIGHUP** - reopen the log file
- **SIGUSR2** - reload the database

Clamd should not be started in the background using the shell operator `&` or external tools. Instead, you should run and wait for clamd to load the database and daemonize itself. After that, clamd is instantly ready to accept connections and perform file scanning.

Clamdscan

`clamdscan` is a simple `clamd` client. In many cases you can use it as a `clamscan` replacement however you must remember that:

- it only depends on `clamd`
- although it accepts the same command line options as `clamscan` most of them are ignored because they must be enabled directly in `clamd`, i.e. `clamd.conf`
- in TCP mode scanned files must be accessible for `clamd`, if you enabled LocalSocket in clamd.conf then clamdscan will try to workaround this limitation by using FILDES

On-access Scanning

There is a special thread in `clamd` that performs on-access scanning under Linux and shares internal virus database with the daemon. By default, this thread will only notify you when potential threats are discovered. If you turn on prevention via `clamd.conf` then **you must follow some important rules when using it:**

- Always stop the daemon cleanly - using the SHUTDOWN command or the SIGTERM signal. In other case you can lose access to protected files until the system is restarted.
- Never protect the directory your mail-scanner software uses for attachment unpacking. Access to all infected files will be automatically blocked and the scanner (including `clamd`!) will not be able to detect any viruses. In the result **all infected mails may be delivered.**
- Watch your entire filesystem only using the `clamd.conf` OnAccessMountPath option. While this will disable on-access prevention, it will avoid potential system lockups caused by fanotify's blocking functionality.
- Using the On-Access Scanner to watch a virtual filesystem will result in undefined behaviour.

The default configuration utilizes inotify to recursively keep track of directories. If you need to protect more than 8192 directories it will be necessary to change inotify's `max_user_watches` value.

This can be done temporarily with:

```
$ sysctl fs.inotify.max_user_watches=<n>
```

Where `<n>` is the new maximum desired.

To watch your entire filesystem add the following lines to `clamd.conf`:

```
ScanOnAccess yes
OnAccessMountPath /
```

Similarly, to protect your home directory add the following lines to `clamd.conf`:

```
ScanOnAccess yes
OnAccessIncludePath /home
OnAccessExcludePath /home/user/temp/dir/of/your/mail/scanning/software
OnAccessPrevention yes
```

For more configuration options, type 'man clamd.conf' or reference the example clamd.conf.

Clamdtop

`clamdtop` is a tool to monitor one or multiple instances of clamd. It has a (color) ncurses interface, that shows the jobs in clamd's queue, memory usage, and information about the loaded signature database. You can specify on the command-line to which clamd(s) it should connect to. By default it will attempt to connect to the local clamd as defined in clamd.conf.

For more detailed help, type 'man clamdtop' or 'clamdtop –help'.

Clamscan

`clamscan` is ClamAV's command line virus scanner. It can be used to scan files and/or directories for viruses. In order for clamscan to work proper, the ClamAV virus database files must be installed on the system you are using clamscan on.

The general usage of clamscan is: clamscan [options] [file/directory/-]

For more detailed help, type 'man clamscan' or 'clamscan –help'.

ClamBC

`clambc` is Clam Anti-Virus' bytecode testing tool. It can be used to test files which contain bytecode. For more detailed help, type 'man clambc' or 'clambc –help'.

Freshclam

`freshclam` is ClamAV's virus database update tool and reads it's configuration from the file 'freshclam.conf' (this may be overridden by command line options). Freshclam's default behavior is to attempt to update databases that are paired with downloaded cdiffs. Potentially corrupted databases are not updated and are automatically fully replaced after several failed attempts unless otherwise specified.

Here is a sample usage including cdiffs:

```
$ freshclam

ClamAV update process started at Mon Oct  7 08:15:10 2013
main.cld is up to date (version: 55, sigs: 2424225, f-level: 60, builder: neo)
Downloading daily-17945.cdiff [100%]
Downloading daily-17946.cdiff [100%]
Downloading daily-17947.cdiff [100%]
daily.cld updated (version: 17947, sigs: 406951, f-level: 63, builder: neo)
Downloading bytecode-227.cdiff [100%]
Downloading bytecode-228.cdiff [100%]
bytecode.cld updated (version: 228, sigs: 43, f-level: 63, builder: neo)
Database updated (2831219 signatures) from database.clamav.net (IP: 64.6.100.177)
```

For more detailed help, type 'man clamscan' or 'clamscan –help'.

Clamconf

`clamconf` is the Clam Anti-Virus configuration utility. It is used for displaying values of configurations options in ClamAV, which will show the contents of clamd.conf (or tell you if it is not properly configured), the contents of freshclam.conf, and display information about software settings, database, platform, and build information. Here is a sample clamconf output:

```
$ clamconf

Checking configuration files in /etc/clamav
```

```
Config file: clamd.conf
-----------------------
ERROR: Please edit the example config file /etc/clamav/clamd.conf

Config file: freshclam.conf
---------------------------
ERROR: Please edit the example config file /etc/clamav/freshclam.conf

clamav-milter.conf not found

Software settings
-----------------
Version: 0.98.2
Optional features supported: MEMPOOL IPv6 AUTOIT_EA06 BZIP2 RAR JIT

Database information
--------------------
Database directory: /xclam/gcc/release/share/clamav
WARNING: freshclam.conf and clamd.conf point to different database directories
print_dbs: Can't open directory /xclam/gcc/release/share/clamav

Platform information
--------------------
uname: Linux 3.5.0-44-generic #67~precise1-Ubuntu SMP Wed Nov 13 16:20:03 UTC 2013 i686
OS: linux-gnu, ARCH: i386, CPU: i686
Full OS version: Ubuntu 12.04.3 LTS
zlib version: 1.2.3.4 (1.2.3.4), compile flags: 55
Triple: i386-pc-linux-gnu
CPU: i686, Little-endian
platform id: 0x0a114d4d0404060401040604

Build information
-----------------
GNU C: 4.6.4 (4.6.4)
GNU C++: 4.6.4 (4.6.4)
CPPFLAGS:
CFLAGS: -g -O0 -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE
CXXFLAGS:
LDFLAGS:
Configure: '--prefix=/xclam/gcc/release/' '--disable-clamav' '--enable-debug' 'CFLAGS=-g -O0'
sizeof(void*) = 4
Engine flevel: 77, dconf: 77
```

For more detailed help, type 'man clamconf' or 'clamconf –help'.

Output format

clamscan

`clamscan` writes all regular program messages to **stdout** and errors/warnings to **stderr**. You can use the option `--stdout` to redirect all program messages to **stdout**. Warnings and error messages from `libclamav` are always printed to **stderr**. A typical output from `clamscan` looks like this:

```
/tmp/test/removal-tool.exe: Worm.Sober FOUND
/tmp/test/md5.o: OK
/tmp/test/blob.c: OK
/tmp/test/message.c: OK
/tmp/test/error.hta: VBS.Inor.D FOUND
```

When a virus is found its name is printed between the `filename:` and `FOUND` strings. In case of archives the scanner depends on libclamav and only prints the first virus found within an archive:

```
$ clamscan malware.zip
malware.zip: Worm.Mydoom.U FOUND
```

When using the –allmatch(-z) flag, clamscan may print multiple virus `FOUND` lines for archives and files.

clamd

The output format of `clamd` is very similar to `clamscan`.

```
$ telnet localhost 3310
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
SCAN /home/zolw/test
/home/zolw/test/clam.exe: ClamAV-Test-File FOUND
Connection closed by foreign host.
```

In the **SCAN** mode it closes the connection when the first virus is found.

```
SCAN /home/zolw/test/clam.zip
/home/zolw/test/clam.zip: ClamAV-Test-File FOUND
```

CONTSCAN and **MULTISCAN** don't stop scanning in case a virus is found.
Error messages are printed in the following format:

```
SCAN /no/such/file
/no/such/file: Can't stat() the file. ERROR
```

LibClamAV

Libclamav provides an easy and effective way to add a virus protection into your software. The library is thread-safe and transparently recognizes and scans within archives, mail files, MS Office document files, executables and other special formats.

License

Libclamav is licensed under the GNU GPL v2 license. This means you are **not allowed** to link commercial, closed-source software against it. All software using libclamav must be GPL compliant.

Supported formats and features

Executables

The library has a built-in support for 32- and 64-bit Portable Executable, ELF and Mach-O files. Additionally, it can handle PE files compressed or obfuscated with the following tools:

- Aspack (2.12)
- UPX (all versions)
- FSG (1.3, 1.31, 1.33, 2.0)
- Petite (2.x)
- PeSpin (1.1)
- NsPack
- wwpack32 (1.20)
- MEW
- Upack
- Y0da Cryptor (1.3)

Mail files

Libclamav can handle almost every mail file format including TNEF (winmail.dat) attachments.

Archives and compressed files

The following archive and compression formats are supported by internal handlers:

- Zip (+ SFX)
- RAR (+ SFX)
- 7Zip
- Tar
- CPIO
- Gzip
- Bzip2
- DMG
- IMG
- ISO 9660
- PKG
- HFS+ partition
- HFSX partition
- APM disk image
- GPT disk image
- MBR disk image
- XAR
- XZ
- MS OLE2
- MS Cabinet Files (+ SFX)

- MS CHM (Compiled HTML)
- MS SZDD compression format
- BinHex
- SIS (SymbianOS packages)
- AutoIt
- NSIS
- InstallShield

Documents

The most popular file formats are supported:

- MS Office and MacOffice files
- RTF
- PDF
- HTML

In the case of Office, RTF and PDF files, libclamav will only extract the embedded objects and will not decode the text data itself. The text decoding and normalization is only performed for HTML files.

Data Loss Prevention

Libclamav includes a DLP module which can detect the following credit card issuers: AMEX, VISA, MasterCard, Discover, Diner's Club, and JCB and U.S. social security numbers inside text files.

Future versions of Libclamav may include additional features to detect other credit cards and other forms of PII (Personally Identifiable Information) which may be transmitted without the benefit of being encrypted.

Others

Libclamav can handle various obfuscators, encoders, files vulnerable to security risks such as:

- JPEG (exploit detection)
- RIFF (exploit detection)
- uuencode
- ScrEnc obfuscation
- CryptFF

API

Header file

Every program using libclamav must include the header file `clamav.h`:

```
#include <clamav.h>
```

Initialization

Before using libclamav, you should call `cl_init()` to initialize it. `CL_INIT_DEFAULT` is a macro that can be passed to `cl_init()` representing the default initialization settings. When it's done, you're ready to create a new scan engine by calling `cl_engine_new()`. To free resources allocated by the engine use `cl_engine_free()`. Function prototypes:

```
int cl_init(unsigned int options);
struct cl_engine *cl_engine_new(void);
int cl_engine_free(struct cl_engine *engine);
```

`cl_init()` and `cl_engine_free()` return `CL_SUCCESS` on success or another code on error. `cl_engine_new()` return a pointer or NULL if there's not enough memory to allocate a new engine structure.

Database loading

The following set of functions provides an interface for loading the virus database:

```
const char *cl_retdbdir(void);

int cl_load(const char *path, struct cl_engine *engine,
    unsigned int *signo, unsigned int options);
```

`cl_retdbdir()` returns the default (hardcoded) path to the directory with ClamAV databases. `cl_load()` loads a single database file or all databases from a given directory (when `path` points to a directory). The second argument is used for passing in the pointer to the engine that should be previously allocated with `cl_engine_new()`. A number of loaded signatures will be **added** to `signo` [9]. The last argument can pass the following flags:

- **CL_DB_STDOPT**
  This is an alias for a recommended set of scan options.
- **CL_DB_PHISHING**
  Load phishing signatures.
- **CL_DB_PHISHING_URLS**
  Initialize the phishing detection module and load .wdb and .pdb files.
- **CL_DB_PUA**
  Load signatures for Potentially Unwanted Applications.
- **CL_DB_OFFICIAL_ONLY**
  Only load official signatures from digitally signed databases.
- **CL_DB_BYTECODE**
  Load bytecode.

`cl_load()` returns `CL_SUCCESS` on success and another code on failure.

```
    ...
    struct cl_engine *engine;
    unsigned int sigs = 0;
    int ret;

if((ret = cl_init(CL_INIT_DEFAULT)) != CL_SUCCESS) {
```

[9] Remember to initialize the virus counter variable with 0.

```
    printf("cl_init() error: %s\n", cl_strerror(ret));
    return 1;
}

if(!(engine = cl_engine_new())) {
    printf("Can't create new engine\n");
    return 1;
}

ret = cl_load(cl_retdbdir(), engine, &sigs, CL_DB_STDOPT);
```

Error handling

Use `cl_strerror()` to convert error codes into human readable messages. The function returns a statically allocated string:

```
if(ret != CL_SUCCESS) {
    printf("cl_load() error: %s\n", cl_strerror(ret));
    cl_engine_free(engine);
    return 1;
}
```

Engine structure

When all required databases are loaded you should prepare the detection engine by calling `cl_engine_compile()`. In case of failure you should still free the memory allocated to the engine with `cl_engine_free()`:

```
int cl_engine_compile(struct cl_engine *engine);
```

In our example:

```
if((ret = cl_engine_compile(engine)) != CL_SUCCESS) {
    printf("cl_engine_compile() error: %s\n", cl_strerror(ret));
    cl_engine_free(engine);
    return 1;
}
```

Limits

When you create a new engine with `cl_engine_new()`, it will have all internal settings set to default values as recommended by the ClamAV authors. It's possible to check and modify the values (numerical and strings) using the following set of functions:

```
int cl_engine_set_num(struct cl_engine *engine,
  enum cl_engine_field field, long long num);

long long cl_engine_get_num(const struct cl_engine *engine,
  enum cl_engine_field field, int *err);

int cl_engine_set_str(struct cl_engine *engine,
  enum cl_engine_field field, const char *str);

const char *cl_engine_get_str(const struct cl_engine *engine,
  enum cl_engine_field field, int *err);
```

Please don't modify the default values unless you know what you're doing. Refer to the ClamAV sources (clamscan, clamd) for examples.

Database checks

It's very important to keep the internal instance of the database up to date. You can watch database changes with the `cl_stat..()` family of functions.

```
int cl_statinidir(const char *dirname, struct cl_stat *dbstat);
int cl_statchkdir(const struct cl_stat *dbstat);
int cl_statfree(struct cl_stat *dbstat);
```

Initialization:

```
    ...
    struct cl_stat dbstat;

memset(&dbstat, 0, sizeof(struct cl_stat));
cl_statinidir(dbdir, &dbstat);
```

To check for a change you just need to call `cl_statchkdir` and check its return value (0 - no change, 1 - some change occurred). Remember to reset the `cl_stat` structure after reloading the database.

```
if(cl_statchkdir(&dbstat) == 1) {
    reload_database...;
    cl_statfree(&dbstat);
    cl_statinidir(cl_retdbdir(), &dbstat);
}
```

Libclamav ≥ 0.96 includes and additional call to check the number of signatures that can be loaded from a given directory:

```
int cl_countsigs(const char *path, unsigned int countoptions,
    unsigned int *sigs);
```

The first argument points to the database directory, the second one specifies what signatures should be counted: `CL_COUNTSIGS_OFFICIAL` (official signatures), `CL_COUNTSIGS_UNOFFICIAL` (third party signatures), `CL_COUNTSIGS_ALL` (all signatures). The last argument points to the counter to which the number of detected signatures will be added (therefore the counter should be initially set to 0). The call returns `CL_SUCCESS` or an error code.

Data scan functions

It's possible to scan a file or descriptor using:

```
int cl_scanfile(const char *filename, const char **virname,
unsigned long int *scanned, const struct cl_engine *engine,
unsigned int options);

int cl_scandesc(int desc, const char **virname, unsigned
long int *scanned, const struct cl_engine *engine,
unsigned int options);
```

Both functions will store a virus name under the pointer `virname`, the virus name is part of the engine structure and must not be released directly. If the third argument (`scanned`) is not NULL, the functions will increase its value with the size of scanned data (in `CL_COUNT_PRECISION` units). The last argument (`options`) specified the scan options and supports the following flags (which can be combined using bit operators):

- **CL_SCAN_STDOPT**
  This is an alias for a recommended set of scan options. You should use it to make your software ready for new features in the future versions of libclamav.

- **CL_SCAN_RAW**
  Use it alone if you want to disable support for special files.
- **CL_SCAN_ARCHIVE**
  This flag enables transparent scanning of various archive formats.
- **CL_SCAN_BLOCKENCRYPTED**
  With this flag the library will mark encrypted archives as viruses (Encrypted.Zip, Encrypted.RAR).
- **CL_SCAN_MAIL**
  Enable support for mail files.
- **CL_SCAN_OLE2**
  Enables support for OLE2 containers (used by MS Office and .msi files).
- **CL_SCAN_PDF**
  Enables scanning within PDF files.
- **CL_SCAN_SWF**
  Enables scanning within SWF files, notably compressed SWF.
- **CL_SCAN_PE**
  This flag enables deep scanning of Portable Executable files and allows libclamav to unpack executables compressed with run-time unpackers.
- **CL_SCAN_ELF**
  Enable support for ELF files.
- **CL_SCAN_BLOCKBROKEN**
  libclamav will try to detect broken executables and mark them as Broken.Executable.
- **CL_SCAN_HTML**
  This flag enables HTML normalisation (including ScrEnc decryption).
- **CL_SCAN_ALGORITHMIC**
  Enable algorithmic detection of viruses.
- **CL_SCAN_PHISHING_BLOCKSSL**
  Phishing module: always block SSL mismatches in URLs.
- **CL_SCAN_PHISHING_BLOCKCLOAK**
  Phishing module: always block cloaked URLs.
- **CL_SCAN_STRUCTURED**
  Enable the DLP module which scans for credit card and SSN numbers.

- **CL_SCAN_STRUCTURED_SSN_NORMAL**
  Search for SSNs formatted as xx-yy-zzzz.
- **CL_SCAN_STRUCTURED_SSN_STRIPPED**
  Search for SSNs formatted as xxyyzzzz.
- **CL_SCAN_PARTIAL_MESSAGE**
  Scan RFC1341 messages split over many emails. You will need to periodically clean up `$TemporaryDirectory/clamav-partial` directory.
- **CL_SCAN_HEURISTIC_PRECEDENCE**
  Allow heuristic match to take precedence. When enabled, if a heuristic scan (such as phishingScan) detects a possible virus/phish it will stop scan immediately. Recommended, saves CPU scan-time. When disabled, virus/phish detected by heuristic scans will be reported only at the end of a scan. If an archive contains both a heuristically detected virus/phishing, and a real malware, the real malware will be reported.
- **CL_SCAN_BLOCKMACROS**
  OLE2 containers, which contain VBA macros will be marked infected (Heuristics.OLE2.ContainsMacros).

All functions return `CL_CLEAN` when the file seems clean, `CL_VIRUS` when a virus is detected and another value on failure.

```
    ...
    const char *virname;

if((ret = cl_scanfile("/tmp/test.exe", &virname, NULL, engine,
CL_SCAN_STDOPT)) == CL_VIRUS) {
    printf("Virus detected: %s\n", virname);
} else {
    printf("No virus detected.\n");
    if(ret != CL_CLEAN)
        printf("Error: %s\n", cl_strerror(ret));
}
```

Memory

Because the engine structure occupies a few megabytes of system memory, you should release it with `cl_engine_free()` if you no longer need to scan files.

Forking daemons

If you're using libclamav with a forking daemon you should call `srand()` inside a forked child before making any calls to the libclamav functions. This will avoid possible collisions with temporary filenames created by other processes of the daemon. This procedure is not required for multi-threaded daemons.

clamav-config

Use `clamav-config` to check compilation information for libclamav.

```
$ clamav-config --libs
-L/usr/local/lib -lz -lbz2 -lgmp -lpthread
$ clamav-config --cflags
-I/usr/local/include -g -O2
```

Example

You will find an example scanner application in the clamav source package (/example). Provided you have ClamAV already installed, execute the following to compile it:

```
gcc -Wall ex1.c -o ex1 -lclamav
```

CVD format

CVD (ClamAV Virus Database) is a digitally signed tarball containing one or more databases. The header is a 512-bytes long string with colon separated fields:

```
ClamAV-VDB:build time:version:number of signatures:functionality
level required:MD5 checksum:digital signature:builder name:build time (sec)
```

`sigtool --info` displays detailed information on CVD files:

```
$ sigtool -i daily.cvd
File: daily.cvd
Build time: 10 Mar 2008 10:45 +0000
Version: 6191
Signatures: 59084
Functionality level: 26
Builder: ccordes
```

```
MD5: 6e6e29dae36b4b7315932c921e568330
Digital signature: zz9irc9irupR3z7yX6J+OR6XdFPUat4HIM9ERn3kAcOWpcMFxq
Fs4toG5WJsHda0Jj92IUusZ7wAgYjpai1Nr+jFfXHsJxv0dBkS5/XWMntj0T1ctNgqmiF
+RLU6V0VeTl4Oej3Aya0cVpd9K4XXevEO2eTTvzWNCAq0ZzWNdjc
Verification OK.
```

Graphics

The current ClamAV logo was created by Alicia Willet, Talos.

OpenAntiVirus

Our database includes the virus database (about 7000 signatures) from OpenAntiVirus (`http://OpenAntiVirus.org`).