stapled Documentation

Release 0.9

Chris Snijder (greenhost), Maarten de Waard (greenhost)

Apr 05, 2018

Table of Contents

CHAPTER 1

Why do I need `stapled`?

`stapled` is meant to be a helper daemon for HAProxy which doesn't do OCSP stapling out of the box. However HAProxy *can* serve staple files if they are place in the certificate directory, which is what we use to our benefit.

You may also be able to use `stapled` for any other proxy that supports serving `.ocsp` files but out of the box it will only save those files and optionally inform a running HAProxy instance of them.

1.1 Quick start

Table of Contents

1.1.1 Documentation

Read the full documentation on Read the docs.

1.1.2 System requirements

This application requires **Python 3.3+** or **Python 2.7.9** and an installed version of **PIP** for the Python version you are using. It is also convenient to have `virtualenv` installed so you can make a separate environment for stapled's dependencies.

1.1.3 Installation

Before installation make sure you have met the *System requirements*. You can install the ocsp daemon from the source code repository on our gitlab instance.

From github (for developers)

```
# Download the source from the repo
git clone --recursive https://github.com/greenhost/stapled.git
# OR, as a TIP, which downloads all the repos simultaneously in threads:
git clone --recursive -j5 https://github.com/greenhost/stapled.git
# Enter the source directory
cd stapled/
# Setup a virtualenv
virtualenv -p python3 env/
# Load the virtualenv
source env/bin/activate
# Install the current directory with pip. This allows you to edit the code
pip install -e .
```

Every time you want to run `stapled` you will need to run `source env/bin/activate` to load the virtualenv first. Alternatively you can start the daemon by running `stapled`

Upgrading

If you had previously installed a version of stapled from github, to upgrade run the following:

```
# Deactivate the virtualenv if active
deactivate
# Delete the virtualenv (we will start clean)
rm -rf ./env
# Make a new virtualenv
virtualenv -p python3 env/
# Update to the latest version
git pull
# Clone submodules too
git submodule upgrade --init --recursive
# Install the current directory with pip. This allows you to edit the code
pip install -e . --upgrade
```

1.1.4 Troubleshooting

In order to get HAPRoxy to serve staples, any valid staple file should exist at the moment it is started. If a staple file does not exist for your certificate stapling will remain disabled until you restart HAProxy. Even if *stapled* tries to send HAProxy a valid staple through its socket.

In order to get around this bootstrapping problem, add an empty staple file, which is also valid according to HAProxy's documentation by running:

```
touch [path-to-certificate].pem.ocsp
```

For each of your domains.

We tested this for HAProxy 1.6, perhaps this behaviour will change in future versions.

1.1.5 Compiling this package

There are 2 ways to compile the package and various target distributions.

Build locally

Assuming you have the following packages installed on a debian based system:

- build-essential
- python-cffi
- python3-cffi
- libffi-dev
- python-all
- python3-all
- python-dev
- python3-dev
- python-setuptools
- python3-setuptools
- python-pip
- rpm

- tar, gzip & bzip2
- git
- debhelper
- stdeb (`pip install --user stdeb`)

Or the equivalents of these on another distribution. You can build the packages by running one or more of the following `make` commands.

```
# Clear out the cruft from any previous build
make clean
# Source distribution
make sdist
# Binary distribution
make bdist
# RPM package (Fedora, Redhat, CentOS) - untested!
make rpm
# Debian source package (Debian, Ubuntu)
make deb-src
# Debian package (Debian, Ubuntu)
make deb
# All of the above
make all
```

Everything is tested under Debian Stretch, your mileage may vary.

Docker build

In order to be able to build a package reproducably by anyone, on any platform we have a `Dockerfile` that will install an instance of Debian Stretch in a docker container and can run the build process for you.

Assuming you have docker installed, you can simply run the below commands to build a package.

```
make docker-all
```

Remove any previous docker image and/or container named *stapled* then buil the image with the same dependencies we used. Then compile the packages, then place them in the *./docker-dist* dir.

```
make docker-nuke
```

Throw away any previous docker image and/or container named *stapled*. This is part of the *make docker-all* target.

```
make docker-build
```

Build the docker image. This is part of the *make docker-all* target.

```
make docker-compile
```

Assuming you have a built image, this compiles the packages for you and places them in *docker-dist*. This is part of the *make docker-all* target.

```
make docker-install
```

Assuming you have a built image and compiled the packages, this installs the packages in the docker container. This is part of the *make docker-all* target.

```
make docker-run
```

Assuming you have a built image and compiled the packages, and installed them in the docker container, this runs the installed binary to test if it works.

Packages

You can download packages here: https://github.com/greenhost/stapled/releases

1.1.6 Using stapled

Update OCSP staples from CA's and store the result so HAProxy can serve them to clients.

```
usage: stapled [-h] [-c CONFIG] [--minimum-validity MINIMUM_VALIDITY]
               [-t RENEWAL_THREADS] [--verbosity VERBOSITY] [-v] [-D]
               [--interactive] [--file-extensions FILE_EXTENSIONS]
               [-r REFRESH_INTERVAL] [-l [LOGDIR]] [--syslog] [-q]
               [-s HAPROXY_SOCKETS [HAPROXY_SOCKETS ...]]
               [--no-haproxy-sockets]
               [--haproxy-config HAPROXY_CONFIG [HAPROXY_CONFIG ...]]
               [-p CERT_PATHS [CERT_PATHS ...]] [-R] [--no-recycle]
               [-i IGNORE [IGNORE ...]] [-V]
               [-d DIRECTORIES [DIRECTORIES ...]]
```

Named Arguments

-c, --config Override the default config file locations (default=/home/docs/checkouts/readthedocs.org/user_builds/stapled/checkouts/latest/docs/source/stapled.con ~/.stapled.conf, /etc/stapled/stapled.conf)

--minimum-validity If the staple is valid for less than this time in seconds an attempt will be made to get a new, valid staple (default: 7200).

-t, --renewal-threads Amount of threads to run for renewing staples. (default=2)

--verbosity Verbose output argument should be an integer between 0 and 4, can be overridden by the `-v` argument.

-v Verbose output, repeat to increase verbosity, overrides the `verbosity` argument if provided.

-D, --daemon Daemonise the process, release from shell and process group, run under new process group.

--interactive, --no-daemon Disable daemon mode, overrides daemon mode if enabled in the config file, effectively starting interactive mode.

--file-extensions Files with which extensions should be scanned? Comma separated list (default: crt,pem,cer).

-r, --refresh-interval Minimum time to wait between parsing cert dirs and certificates (default=60).

-l, --logdir Enable logging to '/var/log/stapled/'. It is possible to supply another directory. Traces of unexpected exceptions are placed here as well.

--syslog Output to syslog.

-q, --quiet Don't print messages to stdout.

-s, --haproxy-sockets Sockets to connect to HAProxy. Each directory you pass with the `directory` argument, should have its own haproxy socket. The order of the socket arguments should match the order of the directory arguments.Example:I have a directory `/etc/haproxy1` with certificates, and a HAProxy that serves these certificates and has stats socket `/etc/haproxy1/haproxy.sock`. I have another directory `/etc/haproxy2` with certificates and another haproxy instance that serves these and has stats socket `/etc/haproxy2/haproxy.sock`. I would then start stapled as follows:`./stapled /etc/haproxy1 /etc/haproxy2 -s /etc/haproxy1.sock /etc/haproxy2.sock`

--no-haproxy-sockets Disable HAProxy sockets, overrides `--haproxy-sockets` if specified in the config file.

--haproxy-config Path(s) to HAProxy config files, they will be scanned for certificates, certificate directories and HAProxy admin sockets based on `bind [..] crt [..]` directives and `stats [..] socket [..]` directives, the `crt-base` directive isrespected. Multiple config files may be specified separated by a space. See `--haproxy-sockets` for more information.

-p, --cert-paths Paths to certificates files or directories containing certificates used by HAProxy. Multiple paths may be specified separated by a space.

-R, --recursive Recursively scan given paths.

--no-recycle Don't re-use existing staples, force renewal.

-i, --ignore Ignore files matching this pattern. Multiple patterns may be specified separated by a space. You can put the pattern in quotes to let stapled evaluate it instead of letting your shell evaluate it. You can use globbing patterns with `*` or `?`. If a pattern starts with `/` it will be considered absolute, if it does not start with a `/`, the pattern will be compared to the last part of found files. e.g. the pattern `cert/snakeoil.pem` matches with path `/etc/ssl/cert/snakeoil.pem`. Don't define relative *paths* as patterns, paths are not patterns, e.g. `../certs/*.pem` will not cause pem files in a directory named `certs`, one directory up from `$PATH` to be ignored. Instead your pattern will cause a warning and will be ignored.

-V, --version Show the version number and exit.

-d, --directories DEPRECATED, please see `--cert-paths`.

The daemon will not serve OCSP responses, it can however inform HAPRoxy about the staples it creates using the `--haproxy-sockets.` argument. Alternatively you can configure HAPRoxy or another proxy (e.g. nginx has support for serving OCSP staples) to serve the OCSP staples manually.

1.1.7 Testing stapled

Testing an application like this is hard, but that is no excuse not to do testing. We want to have unit tests but to do that correctly we need to run an OCSP server locally, quite a setup. So until now we didn't do so yet. Note that if you have experience with this kind of setup and you want to help this project move forward, you are welcome to help.

Obviously we do test stapled, admittedly a little bit primitively. You can find a script in `scripts/` called `refresh_testdata.sh`. It will delete any directory named `testdata` in the root of the project and create a fresh one. Then it will download 3 certificate chains from live servers. These will be placed in subdirectories with the same name as the domain name.

Next you can run `python stapled -vvvv -d testdata/*` to get output printed to your terminal. The `testdata/[domain].[tld]` directories will be populated with `[domain].[tld].ocsp` files.

1.1.8 Caveats

In order to get HAPRoxy to serve staples, any staple valid file should exist at the moment it is started. If a staple file does not exist for your certificate stapling will remain disabled until you restart HAProxy. Even if `stapled` tries to send HAProxy a valid staple through its socket.

In order to get around this bootstrapping problem, add an empty staple file, which is also valid according to HAProxy's documentation by running:

```
touch [path-to-certificate].pem.ocsp
```

For each of your domains.

We tested this for HAProxy 1.6, perhaps this behaviour will change in future versions.

1.2 Module description

`stapled` consists of several modules that interact with each other in order to keep OCSP staples up-to-date. In short, these are the modules:

Scheduler It is possible to schedule a task with the scheduler. It will wait for the scheduled moment and add the task to a queue to be handled by one of the other modules.

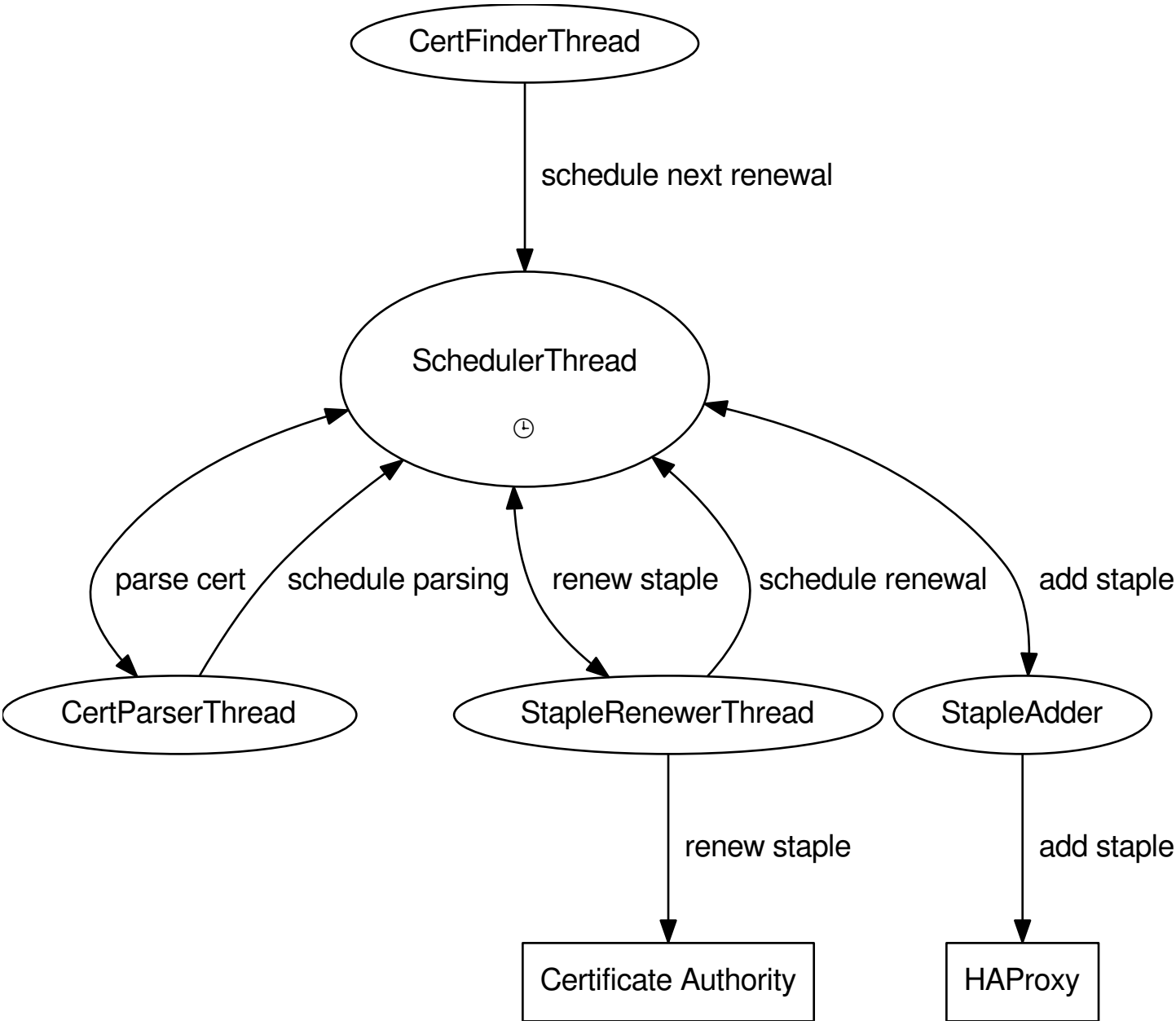
Finder Finds certificates in the specified directories. When new file are found, or existing files are changed it schedules a parsing for these certificates.

Parser Parses certificates and parses them. If certificates are correct, it schedules a renewal for these certificates.

Renewer The renewer takes input from the scheduler. It contacts the CA to renew an OCSP staple. After renewing the staple it schedules a new renewal and tells the scheduler to call the adder right away.

Adder This is a module that can talk to the HAProxy socket to add OCSP staples without restarting HAProxy.

This graph explains their interaction. Every arrow passes a *`StapleTaskContext`* instance to the other module.



1.3 Daemon documentation

Table of Contents

- *stapled.core.certfinder*
- *stapled.core.certparser*
- *stapled.core.staplerenewer*
- *stapled.core.stapleadder*
- *stapled.core.certmodel*

1.3.1 Source code

stapled.main

Initialise the stapled module.

This file only contains some variables we need in the `stapled` name space.

`stapled.`**`LOCAL_LIB_MODE = True`**
If local libs are in use this constant will be `True`

`stapled.`**`FILE_EXTENSIONS_DEFAULT = 'crt,pem,cer'`**
The extensions the daemon will try to parse as certificate files

`stapled.`**`DEFAULT_REFRESH_INTERVAL = 60`**
The default refresh interval for the *stapled.core.certfinder.CertFinderThread*.

`stapled.`**`MAX_RESTART_THREADS = 3`**
How many times should we restart threads that crashed.

`stapled.`**`LOG_DIR = '/var/log/stapled/'`**
Directory where logs and traces will be saved.

`stapled.`**`DEFAULT_CONFIG_FILE_LOCATIONS = ['/home/docs/checkouts/readthedocs.org/user_builds/`**
Default locations to look for config files in order of importance.

stapled.core.daemon

This module bootstraps the stapled process by starting threads for:

- 1x *stapled.scheduling.SchedulerThread*

  Can be used to create action queues that where tasks can be added that are either added to the action queue immediately or at a set time in the future.

- 1x *stapled.core.certfinder.CertFinderThread*
  - Finds certificate files in the specified certificate paths at regular intervals.
  - Removes deleted certificates from the context cache in `stapled.core.daemon.run.models`.
  - Add the found certificate to the the parse action queue of the scheduler for parsing the certificate file.
- 1x *stapled.core.certparser.CertParserThread*
  - Parses certificates and caches parsed certificates in `stapled.core.daemon.run.models`.
  - Add the parsed certificate to the the renew action queue of the scheduler for requesting or renewing the OCSP staple.
- 2x (or more depending on the `-t` CLI argument) *stapled.core.staplerenewer.StapleRenewerThread*

  - Gets tasks from the scheduler in `self.scheduler` which is a `stapled.scheduling.Scheduler` object passed by this module.
  - **For each task:**
    * Validates the certificate chains.
    * Renews the OCSP staples.
    * Validates the certificate chains again but this time including the OCSP staple.
    * Writes the OCSP staple to disk.
    * Schedules a renewal at a configurable time before the expiration of the OCSP staple.

  The main reason for spawning multiple threads for this is that the OCSP request is a blocking action that also takes relatively long to complete. If any of these request stall for long, the entire daemon doesn't stop working until it is no longer stalled.

- 1x *stapled.core.stapleadder.StapleAdder* **(optional)**

  Takes tasks `haproxy-add` from the scheduler and communicates OCSP staples updates to HAProxy through a HAProxy socket.

stapled.core.taskcontext

This module defines an extended version of the general purpose `scheduling.ScheduledTaskContext` for use in the OCSP daemon.

class stapled.core.taskcontext.**StapleTaskContext**(*task_name*, *model*, *sched_time=None*, ***attributes*)

Adds the following functionality to the `scheduling.ScheduledTaskContext`:

- Keep track of the exception that occurred last, and how many times it occurred.
- Renames `ScheduledTaskContext`'s `subject` argument to `model`.

__init__(*task_name*, *model*, *sched_time=None*, ***attributes*)

Initialise a StapleTaskContext with a task name, cert model, and optional scheduled time.

Parameters

- **task_name** (*str*) – A task name corresponding to an existing queue in the scheduler.
- **model** (stapled.core.certmodel.CertModel) – A certificate model.
- **sched_time** (*datetime.datetime* | *int*) – Absolute time (datetime.datetime object) or relative time in seconds (int) to execute the task or None for processing ASAP.
- **attributes** (*kwargs*) – Any data you want to assign to the context, avoid using names already defined in the context: scheduler, task_name, subject, model, sched_time, reschedule.

set_last_exception(*exc*)

Set the exception that occurred just now, this function will return the amount of times the same exception has occurred in a row.

Parameters **exc** (*Exception*) – The last exception.

Return int Count of same exceptions in a row.

Todo: Make sure two similar exceptions are treated as identical, e.g. ignore attributes that will be different every time. https://code.greenhost.net/open/stapled/issues/15

stapled.core.certfinder

This module locates certificate files in the supplied paths and parses them. It then keeps track of the following:

- If cert is found for the first time (thus also when the daemon is started), the cert is added to the `stapled.core.certfinder.CertFinder.scheduler` so the *CertParserThread* can parse the certificate. The file modification time is recorded so file changes can be detected.
- If a cert is found a second time, the modification time is compared to the recorded modification time. If it differs, if it differs, the file is added to the scheduler for parsing again, any scheduled actions for the old file are cancelled.
- When certificates are deleted from the paths, the entries are removed from the cache in `stapled.core.daemon.run.models`. Any scheduled actions for deleted files are cancelled.

The cache of parsed files is volatile so every time the process is killed files need to be indexed again (thus files are considered "new").

class stapled.core.certfinder.**CertFinderThread**(**args*, ***kwargs*)

This searches paths for certificate files. When found, models are created for the certificate files, which are wrapped in a *stapled.core.taskcontext.StapleTaskContext* which are then scheduled to be processed by the *stapled.core.certparser.CertParserThread* ASAP.

Pass `refresh_interval=None` if you want to run it only once (e.g. for testing)

__init__(**args*, ***kwargs*)

Initialise the thread with its parent `threading.Thread` and its arguments.

Parameters

- **models** (*dict*) – A dict to maintain a model cache **(required)**.
- **cert_paths** (*iter*) – The paths to index **(required)**.
- **scheduler** (`stapled.scheduling.SchedulerThread`) – The scheduler object where we add new parse tasks to. **(required)**.
- **refresh_interval** (*int*) – The minimum amount of time (s) between search runs, defaults to 10 seconds. Set to None to run only once **(optional)**.
- **file_extensions** (*array*) – An array containing the file extensions of file types to check for certificate content **(optional)**.

run()

Start the certificate finder thread.

refresh()

Wrap up the internal `CertFinder._update_cached_certs()` and `CertFinder._find_new_certs()` functions.

Note: This method is automatically called by `CertFinder.run()`

_find_new_certs(*paths*, *force_cert_path=None*)

Locate new files, schedule them for parsing.

Parameters

- **paths** (*list|tuple*) – Paths to scan for certificates.
- **force_cert_path** (*str|Nonetype*) – Parent path as specified in the CLI arguments. Necessary to link certificates found in *paths* to any configured sockets.

Raises ***stapled.core.exceptions.CertFileAccessError*** – When the certificate file can't be accessed.

_del_model (*filename*)
Delete model from `stapled.core.daemon.run.models`.

This is done in a thread-safe manner, if another thread deleted it, we should ignore the KeyError making this function omnipotent.

Parameters **filename** (*str*) – The filename of the model to forget about.

_update_cached_certs ()
Check for deleted or changed certificate files.

Loop through the list of files that were already found and check whether they were deleted or changed.

If a file was modified since it was last seen, the file is added to the scheduler to get the new certificate data parsed.

Deleted files are removed from the model cache in `stapled.core.daemon.run.models`. Any scheduled tasks for the model's task context are cancelled.

Raises ***stapled.core.exceptions.CertFileAccessError*** – When the certificate file can't be accessed.

check_ignore (**args*, ***kwargs*)
Check if a file path matches any pattern in the ignore list.

Parameters **path** (*str*) – Path to match a pattern in `self.ignore`.

stapled.core.certparser

This module parses certificate in a queue so the data contained in the certificate can be used to request OCSP responses. After parsing a new *stapled.core.taskcontext.StapleTaskContext* is created for the `stapled.core.oscprenewe.StapleRenewer` which is then scheduled to be processed ASAP.

class `stapled.core.certparser.`**CertParserThread** (**args*, ***kwargs*)
This object makes sure certificate files are parsed, after which a task context is created for the `stapled.core.oscprenewer.OCSPRenewer` which is scheduled to be executed ASAP.

__init__ (**args*, ***kwargs*)
Initialise the thread with its parent `threading.Thread` and its arguments.

Parameters

- **models** (*dict*) – A dict to maintain a model cache **(required)**.
- **minimum_validity** (*int*) – The amount of seconds the OCSP staple should be valid for before a renewal is scheduled **(required)**.
- **scheduler** (`stapled.scheduling.SchedulerThread`) – The scheduler object where we can get parser tasks from and add renew tasks to. **(required)**.
- **no_recycle** (*bool*) – Don't recycle existing staples (default=False)

run ()
Start the certificate parser thread.

parse_certificate(*model*)
Parse certificate files and check whether an existing OCSP staple that is still valid exists. If so, use it, if not request a new OCSP staple. If the staple is valid but not valid for longer than the `minimum_validity`, the staple is loaded but a new request is still scheduled.

stapled.core.staplerenewer

This module takes renew task contexts from the scheduler which contain certificate models that consist of parsed certificates. It then generates an OCSP request and sends it to the OCSP server(s) that is/are found in the certificate and saves both the request and the response in the model. It also generates a file containing the respone (the OCSP staple) and creates a new *stapled.core.taskcontext.StapleTaskContext* to schedule a renewal before the staple expires. Optionally creates a *stapled.core.taskcontext.StapleTaskContext* task context for the `stapled.core.oscpadder.StapleAdder` and schedules it to be run ASAP.

class stapled.core.staplerenewer.**StapleRenewerThread**(**args*, ***kwargs*)
This object requests OCSP responses for certificates, after which a new task context is created for the `stapled.core.oscprenewer.StapleRenewer` which is scheduled to be executed before the new staple expires. Optionally a task is created for the *stapled.core.stapleadder.StapleAdder* to tell HAProxy about the new staple.

__init__(**args*, ***kwargs*)
Initialise the thread's arguments and its parent `threading.Thread`.

Parameters

- **minimum_validity** (*int*) – The amount of seconds the OCSP staple is still valid for, before starting to attempt to request a new OCSP staple **(required)**.
- **scheduler** (`stapled.scheduling.SchedulerThread`) – The scheduler object where we can get tasks from and add new tasks to. **(required)**.

run()
Start the renewer thread.

schedule_renew(*model*, *sched_time=None*)
Schedule to renew this certificate's OCSP staple in `sched_time` seconds.

Parameters

- **context** (`stapled.core.certmodel.CertModel`) – CertModel instance None to calculate it automatically.
- **shed_time** (*int*) – Amount of seconds to wait for renewal or None to calculate it automatically.

Raises **ValueError** – If `context.ocsp_staple.valid_until` is None

stapled.core.stapleadder

Module for adding OCSP Staples to a running HAProxy instance.

class stapled.core.stapleadder.**StapleAdder**(**args*, ***kwargs*)
Add OCSP staples to a running HAProxy instance by sending it over a socket.

It runs a thread that keeps connections to sockets open for each of the supplied haproxy sockets. Code from collectd haproxy connection under the MIT license, was used for inspiration.

Tasks are taken from the *stapled.scheduling.SchedulerThread*, as soon as a task context is received, an OCSP response is read from the model within it, it is added to a HAProxy socket found in self.socks[<certificate directory>].

TASK_NAME = 'proxy-add'
The name of this task in the scheduler

OCSP_ADD = 'set ssl ocsp-response {}'
The haproxy socket command to add OCSP staples. Use string.format to add the base64 encoded OCSP staple

CONNECT_COMMANDS = ['prompt', 'set timeout cli 86400']
Predefines commands to send to sockets just after opening them.

__init__(**args*, ***kwargs*)
Initialise the thread and its parent `threading.Thread`.

Parameters

- **haproxy_socket_mapping** (*dict*) – A mapping from a directory (typically the directory containing TLS certificates) to a HAProxy socket that serves certificates from that directory. These sockets are used to communicate new OCSP staples to HAProxy, so it does not have to be restarted.
- **scheduler** (`stapled.scheduling.SchedulerThread`) – The scheduler object where we can get "haproxy-adder" tasks from **(required)**.

_re_open_socket(*path*)
Re-open socket located at path, and return the socket. Closes open sockets and wraps appropriate logging around the `_open_socket` method.

Parameters **path** (*str*) – A valid HAProxy socket path.

Return socket.socket An open socket.

:raises :exc:stapled.core.exceptions.SocketError: when the socket can not be opened.

_open_socket(*path*)
Open socket located at path, and return the socket.

Subsequently it asks for a prompt to keep the socket connection open, so several commands can be sent without having to close and re-open the socket.

Parameters **path** (*str*) – A valid HAProxy socket path.

Return socket.socket An open socket.

:raises :exc:stapled.core.exceptions.SocketError: when the socket can not be opened.

__del__()
Close the sockets on exit.

run()
Send any commands that enter the command queue.

This is the stapleadder thread's main loop.

add_staple(*model*)
Create and send base64 encoded OCSP staple to the HAProxy.

Parameters **model** – An object that has a binary string *ocsp_staple* in it and a filename *filename*.

static _send(*sock*, *command*)
Send the command through the `socket` and handle response.

Parameters

- **sock** (*list*) – An already opened socket.
- **command** (*str*) – String with the HAProxy command. For a list of possible commands, see the haproxy documentation

Return list List of tuples containing path and response from HAProxy.

:raises IOError if an error occurs and it's not errno.EAGAIN or errno.EINTR

send(*paths*, *command*)
Send the command through the sockets at `paths`.

Parameters

- **paths** (*str|list*) – The path(s) to the socket(s) which should already be open.
- **command** (*str*) – String with the HAProxy command. For a list of possible commands, see the haproxy documentation

Return list List of tuples containing path and response from HAProxy.

:raises IOError if an error occurs and it's not errno.EAGAIN or errno.EINTR

stapled.core.certmodel

This module defines the *stapled.core.certmodel.CertModel* class which is used to keep track of certificates that are found by the *stapled.core.certfinder.CertFinderThread*, then parsed by the *stapled.core.certparser.CertParserThread*, an OCSP request is generated by the `stapled.core.staplerenewer.StapleRenewer`, a response from an OCSP server is returned. All data generated and returned like the request and the response are stored in the context.

The following logic is contained within the context class:

- Parsing the certificate.
- Validating parsed certificates and their chains.
- Generating OCSP requests.
- Sending OCSP requests.
- Processing OCSP responses.
- Validating OCSP responses with the respective certificate and its chain.

class stapled.core.certmodel.**CertModel**(*filename*, *cert_path*)
Model for certificate files.

__init__(*filename*, *cert_path*)
Initialise the CertModel model object, and read the certificate data from the passed filename.

Raises ***stapled.core.exceptions.CertFileAccessError*** – When the certificate file can't be accessed.

parse_crt_file()
Parse certificate, wraps the *_read_full_chain()* and the *_validate_cert()* methods. Wicth extract the certificate (*end_entity*) and the chain intermediates*), and validates the certificate chain.

recycle_staple(*minimum_validity*)
Try to find an existing staple that is still valid for more than the `minimum_validity` period. If it is not valid for longer than the `minimum_validity` period, but still valid, add it to the context but still ask for a new one by returning `False`.

If anything goes wrong during this process, `False` is returned without any error handling, we can always try to get a new staple.

Return bool False if a new staple should be requested, True if the current one is still valid for more than `minimum_validity`

renew_ocsp_staple()

Renew the OCSP staple, validate it and save it to the file path of the certificate file (`certificate.pem.ocsp`).

Note: This method handles a lot of exceptions, some of then are non-fatal and might lead to retries. When they are fatal, one of the exceptions documented below is raised. Exceptions are handled by the *stapled.core.excepthandler.stapled_except_handle()* context.

Note: There can be several OCSP URLs. When the first URL fails, the error handler will increase the `url_index` and schedule a new renewal until all URLS have been tried, then continues with retries from the first again.

Raises

- ***RenewalRequirementMissing*** – A requirment for the renewal is missing.
- ***OCSPBadResponse*** – Response is empty, invalid or the status is not "good".
- **urllib.error.URLError** – An OCSP url can't be opened (Python3).
- **urllib2.URLError** – An OCSP url can't be opened (Python2).

Raises urllib.error.URLError/urllib2.URLError - when a URL/HTTP error occurs

Raises socket.error - when a socket error occurs

Todo: Send merge request to ocspbuider, for setting the hostname in the headers while fetching OCSP records. If accepted the request library won't be needed anymore.

_check_ocsp_response(*ocsp_staple*, *url*)

Check that the OCSP response says that the status is `good`. Also sets `stapled.core.certmodel.CertModel.ocsp_staple.valid_until`.

Raises ***OCSPBadResponse*** – If an empty response is received.

_read_full_chain()

Parses binary data in `self.crt_data` and parses the content. The server certificate a.k.a. *end_entity* is put in `self.end_entity`, anything else that has a CA extension is added to `self.intermediates`.

Note: At this point it is not clear yet which of the intermediates is the root and which are actual intermediates.

Raises ***CertParsingError*** – If the certificate file can't be read, it contains errors or parts of the chain are missing.

_validate_cert(*ocsp_staple=None*)

Validates the certificate and its chain, including the OCSP staple if there is one in `self.ocsp_staple`.

Parameters **ocsp_staple** (*asn1crypto.core.Sequence*) – Binary ocsp staple data.

Return array Validated certificate chain.

Raises ***CertValidationError*** – If there is any problem with the certificate chain and/or the staple, e.g. certificate is revoked, chain is incomplete or invalid (i.e. wrong intermediate with server certificate), certificate is simply invalid, etc.

Note: At this point it becomes known what the role of the certiticates in the chain is. With the exception of the root, which is usually not kept with the intermediates and the certificate because ever client has its own copy of it.

__repr__()
We return the file name here because this way we can use it as a short-cut when we assign this object to something.

__str__()
Return a formatted string representation of the object containing: `"<CertModel {}>".format("".join(self.filename))` so it's clear it's an object and which file it concerns.

__weakref__
list of weak references to the object (if defined)

1.4 Scheduler documentation

Table of Contents

1.4.1 Scheduler source code

scheduling

This is a general purpose scheduler. It does best effort scheduling and execution of expired items in the order they are added. This also means that there is no guarantee the tasks will be executed on time every time, in fact they will always be late, even if just by milliseconds. If you need it to be done on time, you schedule it early, but remember that it will still be best effort.

The way this scheduler is supposed to be used is to add a scheduling queue, then you can add tasks to the queue to either be put in a task queue ASAP, or at or an absolute time in the future. The queue should be consumed by a worker thread.

This module defines the following objects:

- ***stapled.scheduling.ScheduledTaskContext*** A context that wraps around any data you want to pass to the scheduler and which will be added to the task queue when the schedule time expires.
- ***stapled.scheduling.SchedulerThread*** An object that is capable of scheduling and unscheduling tasks that you can define with *stapled.scheduling.ScheduledTaskContext*.

class stapled.scheduling.**ScheduledTaskContext**(*task_name*, *subject*, *sched_time=None*, ***attributes*)

A context for scheduled tasks, this context can be updated with an exception count for the last exception, so it can be re-scheduled if it is the appropriate action.

__init__(*task_name*, *subject*, *sched_time=None*, ***attributes*)

Initialise a *ScheduledTaskContext* with a task name, subject and optional scheduled time. Any remaining keyword arguments are set as attributes of the task context.

Parameters

- **task** (*str*) – A task corresponding to an existing queue in the target scheduler.
- **sched_time** (*datetime.datetime | int*) – Absolute time (datetime.datetime object) or relative time in seconds (int) to schedule the task.
- **subject** (*obj*) – A subject for the context instance this can be whatever object you want to pass along to the worker.
- **attributes** (*kwargs*) – Any additional data you want to assign to the context, avoid using names already defined in the context: `scheduler`, `task`, `subject`, `sched_time`, `reschedule`.

scheduler = None

This attribute will be set automatically when the context is passed to a scheduler.

reschedule(*sched_time=None*)

Reschedule this context itself.

Parameters **sched_time** (*datetime.datetime*) – When should this context be added back to the task queue

__weakref__

list of weak references to the object (if defined)

class stapled.scheduling.**SchedulerThread**(**args*, ***kwargs*)

This object can be used to schedule tasks for contexts.

The context should be a `ScheduledTaskContext` or an extension of it.. When the scheduled time has *passed*, the context will be added back to the internal task queue(s), where it can be consumed by a worker thread. When a task is scheduled you can choose to have it added to the task queue ASAP or at a specified absolute or relative point in time. If you add it with an absolute time in the past, or a negative relative number, it will be added to the task queue the first time the scheduler checks expired tasks schedule times. If you want to run a task ASAP, you probably don't that, you should pass `sched_time=None` instead, it will bypass the scheduling mechanism and place your task directly into the worker queue.

__init__(**args*, ***kwargs*)

Initialise the thread's arguments and its parent `threading.Thread`.

Parameters

- **queues** (*iterable*) – A list, tuple or any iterable that returns strings that should be the names of queues.
- **sleep** (*int | float*) – The sleep time in seconds between checking the expired items in the queue (default=1)

Raises **KeyError** – If the queue name is already taken (only when queues kwarg is used).

schedule = None

The schedule contains items indexed by time.

scheduled_by_context = None

Keeping the tasks in reverse order helps for faster unscheduling.

scheduled_by_queue = None
Keeping the tasks per queue name helps faster queue deletion.

scheduled_by_subject = None
To allow removing by subject we keep the scheduled tasks by subject.

add_queue(*name*, *max_size=0*)
Add a scheduled queue to the scheduler.

Parameters

- **name** (*str*) – A unique name for the queue.
- **max_size** (*int*) – Maximum queue depth, [default=0 (unlimited)].

Raises **KeyError** – If the queue name is already taken.

remove_queue(*name*)
Remove a scheduled queue from the scheduler.

Parameters **name** (*str*) – The name of the existing queue.

Raises **KeyError** – If the queue doesn't exist.

add_task(*ctx*)
Add a `ScheduledTaskContext` to be added to the task queue either ASAP, or at a specific time.

If the context is not unique, the scheduled task will be cancelled before scheduling the new task.

Parameters **ctx** (`ScheduledTaskContext`) – A context containing data for a worker thread.

Raises

- **queue.Queue.Full** – If the underlying task queue is full.
- **TypeError** – If the passed context is not a `ScheduledTaskContext`
- **KeyError** – If the task queue doesn't exist.

cancel_task(*ctx*)
Remove a task from the scheduler.

Note: Tasks that were already queued for a worker to process can't be canceled anymore.

Parameters **ctx** (`ScheduledTaskContext`) – A context containing data for a worker thread.

Return bool True for successfully cancelled task or False.

get_task(*task_name*, *blocking=True*, *timeout=None*)
Get a task context from the task queue `task`.

Parameters

- **task_name** (*str*) – Task name that refers to an existsing scheduler queue.
- **blocking** (*bool*) – Wait until there is something to return from the queue.

Raises

- **Queue.Empty** – If the underlying task queue is empty and blocking is False or the timout expires.

- **KeyError** – If the task queue does not exist.

task_done(*task_name*)
Mark a task done on a queue, this up the queue's counter of completed tasks.

Parameters **task_name** (*str*) – The task queue name.

Raises **KeyError** – If the task queue does not exist.

run()
Start the scheduler thread.

run_all()
Run all tasks currently queued regardless schedule time.

_run(*all_tasks=False*)
Runs all scheduled tasks that have a scheduled time < now.

cancel_by_subject(*subject*)
Cancel scheduled tasks by the task's context's subject.

This comes down to: delete anything from the scheduler that relates to my object *X*.

Parameters **subject** (*obj*) – The object you want all scheduled tasks cancelled for.

1.5 Exception handling

During the OCSP renewal proces lots of things could go wrong, some errors are recoverable, others can be ignored, still others could be cause by temporary issues e.g.: a service interruption of the OCSP server in question. So extensive error handling is done to keep the daemons threads running.

The following is an overview of what can be expected when exceptions occur.

Exception	Source	Raised when?	Action
IOError/OSError	certfinder	Directory can't be read.	Ignore, certfinder will try at every refresh.
CertFileAccessError	certfinder	Certificate file can't be read.	Schedule retry 3x *n**60s, then 3x, every hour, then ignore.[1]
CertParsingError	certparser	Can't access the certificate file, doesn't parse or part of the chain is missing.	Ignore, certfinder will try at every refresh.
StapleBadResponse	staplerenewer	The response is empty, invalid or the status is not "good".	Schedule retry 3x *n**60s, then 3x, every hour, then twice a day. indefinately. If it's not a server issue, wait for the file to change[1]
urllib.error.URLError	staplerenewer	An OCSP url can't be opened.	We can try again later, maybe there is a server side issue. Some certificates contain multiple URL's so we will try each one with 10 seconds intervals and then start from the first again. Schedule retry 3x *n**60s, then 3x, every hour, then then twice a day.
requests.exceptions.Timeout	staplerenewer	Data didn't reach us within the expected time frame.	
requests.exceptions.ReadTimeout	staplerenewer		
requests.exceptions.ConnectTimeout	staplerenewer	A connection can't be established because the server doesn't reply within the expected time frame.	
requests.exceptions.TooManyRedirects	staplerenewer	When the OCSP server redirects us too many times. Limit is quite high so probably something is wrong with the OCSP server.	
requests.exceptions.HTTPError	staplerenewer	A HTTP error code was returned, this can be a 4xx or 5xx status code.	
requests.exceptions.ConnectionError	staplerenewer	A connection to the OCSP server can not be established.	
SocketError	stapleadder	A HAProxy socket can not be opened	Log a critical error. Every "send" action will try to re-open the socket.
BrokenPipeError	stapleadder	A HAProxy socket consistently has a broken pipe	
StapleAdderBadResponse	stapleadder	HAProxy does not respond with 'OCSP Response updated!'	Schedule a retry 3x *n**60s, then 3x, every hour, then ignore.

1.5.1 stapled.core.exceptions

This module holds the application specific exceptions.

exception stapled.core.exceptions.**OCSPBadResponse**
Raised when a OCSP staple is not valid.

[1] When the certificate file is changed, *certfinder* will add the file back to the parsing queue.

exception stapled.core.exceptions.**RenewalRequirementMissing**
Raised when a OCSP renewal is run while not all requirements are met.

exception stapled.core.exceptions.**SocketError**
Raised by the StapleAdder when it is impossible to connect to or use its socket.

exception stapled.core.exceptions.**StapleAdderBadResponse**
Raised when HAProxy does not respond with "OCSP Response updated".

exception stapled.core.exceptions.**CertFileAccessError**
Raised when a file can't be accessed at all.

exception stapled.core.exceptions.**CertParsingError**(*msg*, **args*, ***kwargs*)
Raised when something went wrong while parsing the certificate file.

exception stapled.core.exceptions.**CertValidationError**
Raised when validation the certificate chain fails.

1.5.2 stapled.core.excepthandler

This module defines a context in which we can run actions that are likely to fail because they have intricate dependencies e.g. network connections, file access, parsing certificates and validating their chains, etc., without stopping execution of the application. Additionally it will log these errors and depending on the nature of the error reschedule the task at a time that seems reasonable, i.e.: we can reasonably expect the issue to be resolved by that time.

It is generally considered bad practice to catch all remaining exceptions, however this is a daemon. We can't afford it to get stuck or crashed. So in the interest of staying alive, if an exception is not caught specifically, the handler will catch it, generate a stack trace and save if in a file in the current working directory. A log entry will be created explaining that there was an exception, inform about the location of the stack trace dump and that the context will be dropped. It will also kindly request the administrator to contact the developers so the exception can be caught in a future release which will probably increase stability and might result in a retry rather than just dropping the context.

Dropping the context effectively means that a retry won't occur and since the context will have no more references, it will be garbage collected. There is however still a reference to the certificate model in core.daemon.run.models. With no scheduled actions it will just sit idle, until the finder detects that it is either removed – which will cause the entry in core.daemon.run.models to be deleted, or it is changed. If the certificate file is changed the finder will schedule schedule a parsing action for it and it will be picked up again. Hopefully the issue that caused the uncaught exception will be resolved, if not, if will be caught again and the cycle continues.

stapled.core.excepthandler.**LOG_DIR = '/var/log/stapled/'**
This is a global variable that is overridden by stapled.__main__ with the command line argument: --logdir

stapled.core.excepthandler.**stapled_except_handle**(**args*, ***kwds*)
Handle lots of potential errors and reschedule failed action contexts.

stapled.core.excepthandler.**handle_file_error**(*exc*)
Wrapper for handling IOError and OSError logging..

Can't use FileNotFoundError and PermissionError because they don't exist in Python 2.7.x yet. This won't be required after we remove Python 2.7.x support. :param Exception exc: OSError or IOError to handle logging for. :return str: Reason for OSError/IOError.

stapled.core.excepthandler.**delete_ocsp_for_context**(*ctx*)
When something bad happens, sometimes it is good to delete a related bad OCSP file so it can't be served any more.

Todo: Check that HAProxy doesn't cache this, it probably does, we need to be able to tell it not to remember it.

---

stapled.core.excepthandler.**dump_stack_trace**(*ctx*, *exc*)

Examine the last exception and dump a stack trace to a file, if it fails due to an IOError or OSError, log that it failed so the a sysadmin may make the directory writeable.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

C

D