Certgrinder

Thomas Steen Rasmussen

Jan 23, 2022

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Certgrinder is a client/server system written in Python to handle Letsencrypt certificate issuing on a central host (the Certgrinder server), rather than on the machines which need the certificates (the Certgrinder clients). This is accomplished by redirecting the LetsEncrypt challenges to the Certgrinder server. For `DNS-01` challenges this is done with a `CNAME record` and for `HTTP-01` challenges it is done with a `HTTP 301 redirect`.

To get a certificate the Certgrinder client calls the Certgrinder server (typically over `SSH`) with a `CSR` on `stdin` and (if all goes well) get a signed certificate in return on `stdout`.

Certgrinder support both `RSA` and `ECDSA` keys and certificates, and defaults to getting both kinds.

CHAPTER

TWO

ADVANTAGES

- The approach with a central host serving all challenges simplifies getting certificates for stuff like loadbalanced or anycast services, where it can be impossible to predict which cluster node the LetsEncrypt challenge checker will hit when using `HTTP-01`.
- Using `DNS-01` with a separate delegated zone dedicated to serve the challenges is safer than opening up dynamic updates of your primary zone(s) with your provider.
- Migrating services to new infrastructure becomes simpler because the new infrastructure can get real certificates before changing DNS to point to the new infrastructure.
- Certgrinder makes it trivial to get certificates for infrastructure behind firewalls or even on networks with no Internet connection. As long as the Certgrinder client can reach the Certgrinder server it is possible to use `DNS-01` to issue certificates for the client.
- Certgrinder does not rotate the RSA/ECDSA keypair on each certificate renewal, which makes `TLSA` and similar public key pinning easy. The Certgrinder client can output and check such `TLSA` and `SPKI` pins for the keypairs it manages, as well as checking correctness of `TLSA` records in the DNS.
- Certgrinder supports fetching OCSP responses via the Certgrinder server. Having Certgrinder fetch the OCSP response makes it possible to configure OCSP stapling without relying on the various TLS servers own OCSP-fetching implementation. It also means that OCSP stapling can be done on servers behind strict firewalls or with no Internet connection, as the communication with CA is done via the Certgrinder server.
- Certgrinder supports alternate chain selection and understands the longer chain used by LetsEncrypt since May 2021.

CHAPTER

THREE

TERMINOLOGY

The central host with the LetsEncrypt signing stack is called the "Certgrinder server". The individual servers (the ones that need the certificates) are called "Certgrinder clients". These match the two Python packages `certgrinderd` and `certgrinder`, respectively.

CHAPTER

FOUR

CHALLENGES

Certgrinder supports the two ACME challenge types DNS-01 and HTTP-01. Certgrinderd will try both challenge types (DNS-01 first, then HTTP-01), so your clients can use whatever is the best fit. Usually my webservers use HTTP-01 and everything else uses DNS-01. YMMV.

The following sections describes how Certgrinder handles these two challenge types.

4.1 DNS-01

With the DNS-01 challenge type the Certgrinder server serves the challenge over DNS, which means you need to run an authoritative DNS server on the Certgrinder server. You can also use an external DNS server or provider, as long as you can make a hook script to add and delete records from the Certgrinder server as needed.

To prepare the Certgrinder server for serving DNS-01 challenges you first need a zone to serve the challenges. Invent and delegate a new zone (like acme.example.com) to your Certgrinder server or DNS provider. Use an NS record to delegate, or follow your providers instructions. The zone name then needs to be configured in certgrinderd.conf. This zone will be used to serve all DNS-01 challenges, it will be updated automatically by certgrinderd as needed.

The default manual-auth-hook script is made for the bind DNS server. It creates and deletes the DNS record using nsupdate and an rndc.key file in the path /usr/local/etc/namedb/rndc.key. If you want to use other paths or another script for a local or external DNS provider you can configure it in certgrinderd.conf. The same goes for the cleanup script manual-cleanup-hook.

Note: Since certbot is responsible for calling the hooks they are run as root, just like certbot.

This concludes the server part of the DNS-01 configuration.

A client wanting a certificate must now create a CNAME record called _acme-challenge.${DOMAIN} pointing at ${DOMAIN}.${ACMEZONE} for each domain in the CSR.

For example, to get a certificate for smtp.example.org you would create _acme-challenge.smtp.example.org CNAME smtp.example.org.acme.example.com if your acme challenge zone was acme.example.com. certgrinderd will create the smtp.example.org.acme.example.com TXT record containing the validation string, and delete it afterwards.

4.2 HTTP-01

With the `HTTP-01` challenge type the Certgrinder server serves the challenge over HTTP, which means it needs a webserver somewhere to serve the challenges. It can be on the Certgrinder server or it can be an external webserver or provider, as long as you can make a hook script to add and delete files in the webroot from the Certgrinder server as needed. The hostname of this webserver will be the target of the Certgrinder clients HTTP redirects.

Each Certgrinder client then implements an HTTP redirect from `/.well-known/acme-challenge/` to the Certgrinder server like so (nginx syntax):

```
location /.well-known/acme-challenge/ {
    return 301 http://acme.example.com$request_uri;
}
```

When requesting a certificate the Certgrinder server receives the challenge and path from Certbot (which in turn gets it from LetsEncrypt of course). The challenge is then passed to the `manual-auth-hook` script which writes it in the webroot under `/.well-known/acme-challenge/`.

In another datacenter somewhere LetsEncrypts challenge checker then loops over the domains in the `CSR` and does a HTTP request to each for `/.well-known/acme-challenge/${path}` and expects the response to contain the challenge.

CHAPTER

FIVE

CERTGRINDERD

The Certgrinder server `certgrinderd` takes care of getting certificates and OCSP responses on behalf of the calling clients. `certgrinderd` doesn't run always like a daemon, so it never acts on its own. It only does something when a Certgrinder client runs it, usually over SSH.

The following sections explain the steps you need to setup a Certgrinder server.

5.1 Install Certgrinder Server

Create a VM or Jail or Docker thing or whatever somewhere. This will be your Certgrinder server. Give it a proper public hostname like `certgrinder.example.com`. You can use real proper IP addresses or port forwarding, whichever you prefer. The relevant ports are `TCP/22` (so the Certgrinder clients can reach the Certgrinder server), `TCP/53` and `UDP/53` if you want to serve `DNS-01` challenges locally, and `TCP/80` if you use `HTTP-01` challenges locally.

Create DNS records for the new hostname (A+AAAA, and an SSHFP record wouldn't hurt) and you should be ready to begin the install.

The hostname of your Certgrinder server will be the hostname your Certgrinder clients use to SSH into (if you use SSH), and also the hostname you use to serve HTTP challenges locally (if you use `HTTP-01` challenges).

5.2 Create User

Create a dedicated user to run the Certgrinder server, usually the username is just `certgrinderd`. The user needs `sudo` access to run the `certbot` binary, and to set a couple of environment variables. This works:

```
certgrinderd ALL=(ALL) NOPASSWD: /usr/local/bin/certbot
Defaults env_keep += "ACMEZONE WEBROOT"
```

5.3 Install certgrinderd

You can install `certgrinderd` from pip with `pip install certgrinderd`. It will pull in the dependencies it needs automatically. Create a venv for it if you don't want to pollute the global Python env.

You can also checkout the Github repo and install the deps from `requirements.txt` by hand if you prefer. If you want to install with pip directly from Github the following may help: `pip install "git+https://github.com/tykling/certgrinder/#egg=certgrinderd&subdirectory=server"`

The Certgrinder server needs to be reachable from the outside world on port 53/80 if you plan to serve DNS/HTTP challenges locally. It also needs to be accessible over SSH from all your Certgrinder clients if you plan to use SSH.

5.4 Configuration

Configuration of `certgrinderd` can be done using command-line options, or a configuration file, or a combination of the two.

The `certgrinderd` configuration file is in YAML format. An example config named `certgrinderd.conf.dist` can be found in the distribution. use `--config-file` or `-f` to specify the config file location.

Each config item can be specified either in the YAML config file as a `key: value` pair, or on the commandline as `--key value` - the latter overriding the former if both are present. For example, if the configfile has `log-level: INFO` and the command-line has `log-level: DEBUG` then the effective log-level would be `DEBUG`.

This is an alphabetical list of the configurable options:

acme-email The email to use for the ACME account creation. Only required for the first run.

Default: `None`

acme-server-url The URL for the ACME server.

Default: `https://acme-v02.api.letsencrypt.org/directory`

acme-zone The DNS zone to pass to auth-hook script as environment variable ACMEZONE. Leave this unset to disable DNS-01 challenges.

Default: `None`

auth-hook The script to run to prepare challenges before running Certbot.

Default: `manual-auth-hook.sh`

certbot-command The Certbot command to run between the auth hook and the cleanup hook.

Default: `/usr/local/bin/sudo /usr/local/bin/certbot`

certbot-config-dir The path to pass to Certbot as `--config-dir`.

Default: `None`

certbot-logs-dir The path to pass to Certbot as `--logs-dir`.

Default: `None`

certbot-work-dir The path to pass to Certbot as `--logs-dir`.

Default: `None`

cleanup-hook The script to run to cleanup challenges after running Certbot.

Default: `manual-cleanup-hook.sh`

config-file The path to the configuration file. The file is in YAML format.

Default: `None`

debug Enables debug mode. This is the same as setting –log-level to DEBUG. Outputs lots info about the internal workings of certgrinderd.

Default: `False`

log-level Sets the verbosity level for console and syslog logging. One of DEBUG, INFO, WARNING, ERROR, CRITICAL.

Default: `INFO`

pid-dir The directory to place the certgrinderd PID file in.

Default: `/tmp`

skip-acme-server-cert-verify Set to skip verification of the ACME servers TLS certificate. Used for testing, do not use in real world.

Default: `False`

staging Enable staging mode. To make Certbot use the LetsEncrypt staging servers.

Default: `False`

syslog-facility Set this and syslog-socket to enable logging to syslog. Must be a value supported by `logging.handlers.SysLogHandler` like `LOG_USER` or `LOG_LOCAL0`.

Default: `None`

syslog-socket Set this and syslog-facility to enable logging to syslog.

Default: `None`

temp-dir Set this to the directory to use for temporary files (CSR and certificates). Directory should be owned by the user running `certgrinderd`. A directory will be created and deleted inside this tempdir for each run. Leave blank to create one automatically.

Default: `None`

web-root The path to pass to the auth-hook script as environment variable WEBROOT. Leave this blank to disable HTTP-01 challenges.

Default: `None`

Finally the permitted domains for the current client must be specified as an environment variable (see next section).

5.5 Restricting Client Hostnames

To determine whether a Certgrinder client is authorised to get a certificate for a given list of hostnames `certgrinderd` checks the environment variable named `CERTGRINDERD_DOMAINSETS` which must contain a semicolon-separated list of comma-separated lists of hostnames permitted for the current client.

For example, if the Certgrinder client was a webserver with two vhosts, one with the name `example.net` and another vhost with the two names `example.com` and `www.example.com`. In this case the environment variable `CERTGRINDERD_DOMAINSETS="example.net;example.com,www.example.com"` would permit the client to get the two certificates it needs, and nothing else.

The list of hostnames is case insensitive. IDN names can be in either IDNA or unicode format, meaning `xn--plse-gra.example` and `pølse.example` will both work. The order of the hostnames in the list does not matter.

5.6 Configure SSH Access

Usually Certgrinder clients connect to the Certgrinder server using SSH, but other connection methods can be used if needed. The rest of this section is about configuring SSH access for clients.

Each Certgrinder client must generate an SSH key which is to be added to `~/.ssh/authorized_keys` on the Certgrinder server. Each entry must be restricted with:

- A `from=` specifying the IP the Certgrinder client connects from (optional but recommended).
- An `environment=` restricting which names it may ask for, see above (required).
- `command=` to restrict the command it can run (optional but recommended). Remember `$SSH_ORIGINAL_COMMAND` so `certgrinder` can set `certgrinderd` command-line arguments.

- The `restrict` keyword to limit tunneling and forwarding and such (optional but recommended). The `restrict` option was added to OpenSSH in version 7.4, it might not be available everywhere.

Something like this works:

```
from="2001:DB8::15",environment="CERTGRINDERD_DOMAINSETS=example.com,www.example.com;
↪example.net",command="/path/to/certgrinderd $SSH_ORIGINAL_COMMAND",restrict ssh-
↪ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIOegnR+qnK2FEoaSrVwHgCIxjFkVEbW4VO31/Hd2mAwk ansible-
↪generated on webproxy2.example.com
```

To make the `environment=` foo work the option `PermitUserEnvironment=CERTGRINDERD_DOMAINSETS` needs to be added to `sshd_config`.

5.7 Auth and Cleanup Hooks

The configured `auth-hook` and `cleanup-hook` scripts can be adapted as needed to update whatever local or remote web- or DNS-server you decide to use to serve challenges.

Both scripts get the same environment variables to work with:

$CERTBOT_DOMAIN The domain being authenticated, like www.example.com

$CERTBOT_VALIDATION The validation string (the secret which LE looks for)

$CERTBOT_TOKEN The filename containing the secret (only relevant for HTTP-01)

$ACMEZONE The DNS zone used for challenges (only relevant for DNS-01)

$WEBROOT The path to the webroot used for challenges (only relevant for HTTP-01)

Both scripts must be able to handle the challenge type(s) you use. The same script will be called first for DNS-01 (if enabled), then for HTTP-01 (if enabled).

5.8 Testing

When the server has been configured with hooks you can test from a client using just SSH and a manually generated CSR, with something like: `cat mail4.example.com.csr | ssh certgrinderd@certgrinder.example.org -T -- --staging get certificate` where `-T` is to prevent SSH from allocating a TTY on the server, `--` is to mark the end of the SSH args, and `--staging` is to make `certgrinderd` use the LetsEncrypt staging servers. If all goes well it should output some logging and a certificate chain.

5.9 Command Line Usage

certgrinderd version 0.17.2. See the manpage certgrinderd(8) or ReadTheDocs for more info.

```
usage: certgrinderd [-h] [--acme-email ACME-EMAIL]
                    [--acme-server-url ACME-SERVER-URL] [-z ACME-ZONE]
                    [-A AUTH-HOOK] [--certbot-command CERTBOT-COMMAND]
                    [--certbot-config-dir CERTBOT-CONFIG-DIR]
                    [--certbot-logs-dir CERTBOT-LOGS-DIR]
                    [--certbot-work-dir CERTBOT-WORK-DIR] [-C CLEANUP-HOOK]
```

(continues on next page)

(continued from previous page)

```
                    [-c CONFIG-FILE] [--certificate-file CERTIFICATE-FILE]
                    [--csr-file CSR-FILE] [-d]
                    [-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}] [-p PID-DIR]
                    [--preferred-chain PREFERRED-CHAIN]
                    [--skip-acme-server-cert-verify] [-s]
                    [--syslog-facility SYSLOG-FACILITY]
                    [--syslog-socket SYSLOG-SOCKET] [-t TEMP-DIR]
                    [-w WEB-ROOT]
                    {get,show,help,ping} ...
```

5.9.1 Positional Arguments

command Possible choices: get, show, help, ping

Command (required)

5.9.2 Named Arguments

--acme-email The email for the ACME account.

--acme-server-url The url for the ACME server to use.

-z, --acme-zone The DNS zone to pass to the auth hook script as env. var. ACMEZONE. For DNS-01 challenges.

-A, --auth-hook The hook script to call to prepare auth challenges before calling Certbot

--certbot-command The Certbot command to call between auth hook and cleanup hook

--certbot-config-dir The path to pass to Certbot as –config-dir

--certbot-logs-dir The path to pass to Certbot as –logs-dir

--certbot-work-dir The path to pass to Certbot as –work-dir

-C, --cleanup-hook The hook script to call to clean up auth challenges after calling Certbot

-c, --config-file The path to the certgrinderd config file to use, in YML format.

--certificate-file The path to the PEM formatted certificate chain file to use instead of getting it from stdin.

--csr-file The path to the PEM formatted CSR file to use instead of getting it from stdin.

-d, --debug Debug mode. Equal to setting –log-level=DEBUG.

-l, --log-level Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL

Logging level. One of DEBUG, INFO, WARNING, ERROR, CRITICAL. Defaults to INFO.

-p, --pid-dir The directory to store the PID file in

--preferred-chain The preferred chain to use. Adds –preferred-chain to the Certbot command. Use to pick preferred signing chain when alternatives are available. Replace spaces with underscores in the chain name, so DST_Root_CA_X3 or ISRG_Root_X1 for prod or Fake_LE_Root_X1 or Fake_LE_Root_X2 for staging.

--skip-acme-server-cert-verify Do not verify the ACME servers certificate

-s, --staging	Staging mode. Equal to setting –acme-server-url https://acme-staging-v02.api.letsencrypt.org/directory
--syslog-facility	The facility to use for syslog messages
--syslog-socket	The socket to use for syslog messages
-t, --temp-dir	The directory to store temporary files in
-w, --web-root	The path to pass to the auth hook script as env WEBROOT to use for HTTP-01 challenges.

5.9.3 Sub-commands:

get

Use the "get" command to get certificates or OCSP responses

```
certgrinderd get [-h] {certificate,ocsp} ...
```

Positional Arguments

subcommand	Possible choices: certificate, ocsp Specify what to get using one of the available get sub-commands

Sub-commands:

certificate

Get a new certificate. Requires a CSR.

```
certgrinderd get certificate [-h]
```

ocsp

Get an OCSP response for the provided certificate.

```
certgrinderd get ocsp [-h]
```

show

Use the "show" command to show configuration, CSR info, or certificate info.

```
certgrinderd show [-h] {certificate,csr,configuration} ...
```

Positional Arguments

subcommand Possible choices: certificate, csr, configuration

Specify what to show using one of the available show sub-commands

Sub-commands:

certificate

Tell certgrinder to output information about the provided certificate.

```
certgrinderd show certificate [-h]
```

csr

Tell certgrinder to output information about the provided CSR.

```
certgrinderd show csr [-h]
```

configuration

Tell certgrinder to output the current configuration

```
certgrinderd show configuration [-h]
```

help

The "help" command just outputs the usage help

```
certgrinderd help [-h]
```

ping

The "ping" command is used by the certgrinder client to verify connectivity to the server. It just outputs the word "pong" to stdout.

```
certgrinderd ping [-h]
```

5.10 Class Methods

class certgrinderd.**Certgrinderd**(*userconfig: Optional[Dict[str, Optional[Union[str, bool]]]] = None*)

The Certgrinderd server class.

__init__(*userconfig: Optional[Dict[str, Optional[Union[str, bool]]]] = None*) → None

Merge userconfig with defaults and configure logging.

Parameters **userconfig** – A dict of configuration to merge with default config

Returns None

static check_csr(*csr: cryptography.hazmat.backends.openssl.x509._CertificateSigningRequest*) → bool

Check that this CSR is valid, all things considered.

First check that the CSR has exactly one `CommonName`, and that that CN is also present in the list of `SubjectAltNames`.

Then make sure that the environment var `CERTGRINDERD_DOMAINSETS` exists and contains all the names from the CSR in one of the domainsets.

Parameters **csr** – The CSR object

Returns True if the CSR is OK, False otherwise

classmethod check_ocsp_response(*ocsp_request: cryptography.hazmat.backends.openssl.ocsp._OCSPRequest*, *ocsp_response: cryptography.hazmat.backends.openssl.ocsp._OCSPResponse*, *certificate: cryptography.x509.base.Certificate*, *issuer: cryptography.x509.base.Certificate*) → bool

Check that the OCSP response is valid for the OCSP request and cert/issuer.

Return True if the OCSP response is good, regardless of the certificate revocation status. Implements all the checks in RFC2560 3.2.

Parameters

- **ocsp_request** – The OCSP request object to check
- **ocsp_response** – The OCSP response object to check
- **certificate** – The certificate the OCSP request is for
- **issuer** – The issuer of the certificate

Returns True if the OCSP response is valid, False if not

static check_ocsp_response_issuer(*ocsp_request: cryptography.hazmat.backends.openssl.ocsp._OCSPRequest*, *ocsp_response: cryptography.hazmat.backends.openssl.ocsp._OCSPResponse*) → bool

Check that the response matches the request.

Parameters

- **ocsp_request** – The OCSP request object
- **ocsp_response** – The OCSP response object

Returns Boolean - True if all is well, False if a problem was found

classmethod check_ocsp_response_signature(*ocsp_response: cryptography.hazmat.backends.openssl.ocsp._OCSPResponse*, *issuers: List[cryptography.x509.base.Certificate]*) → bool

Check the signature of the OCSP response.

Parameters

- **ocsp_response** – The OCSP response to check
- **issuers** – A list of issuer(s)

Returns Boolean - True if all is well, False if a problem was found

static check_ocsp_response_timing(*ocsp_response: cryptography.hazmat.backends.openssl.ocsp._OCSPResponse*) → bool

Check the timestamps of the OCSP response.

Parameters **ocsp_response** – The OCSP response object to check

Returns Boolean - True if all is well, False if a problem was found

classmethod create_ocsp_request(*certificate: cryptography.x509.base.Certificate*, *issuer: cryptography.x509.base.Certificate*) → cryptography.hazmat.backends.openssl.ocsp._OCSPRequest

Create and return an OCSP request based on the cert+issuer.

Parameters

- **certificate** – The certificate to create an OCSP request for
- **issuer** – The issuer of the certificate

Returns The OCSP request

get_certbot_command(*challengetype: str*, *csrpath: str*, *fullchainpath: str*, *certpath: str*, *chainpath: str*) → List[str]

Put the certbot command together.

Start with `self.conf["certbot-command"]` and append all the needed options.

Optionally add `--email` and a bunch of certbot settings as needed.

Parameters

- **challengetype** – The type of challenge, `dns` or `http`
- **csrpath** – The path to the CSR
- **fullchainpath** – The path to save the certificate+issuer
- **certpath** – The path to save the certificate (without issuer)
- **chainpath** – The path to save the issuer (without certificate)

Returns The certbot command as a list

get_certificate(*csrpath: str*) → None

Get a cert using `DNS-01` or `HTTP-01` by calling `self.run_certbot()` for each.

If `self.conf["acme-zone"]` is set then `DNS-01` is attempted. Return if it results in a new certificate.

If `self.conf["web-root"]` is set then `HTTP-01` is attempted. Return if it results in a new certificate.

If there is still no certificate log an error and return anyway.

Parameters **csrpath** – The path to the CSR

Returns None

get_certificate_command() → None
This method is called when the *get certificate* subcommand is used.

Parameters **None** –

Returns None

get_ocsp_command() → None
This method is called when the *get ocsp* subcommand is used.

It simply prints the DER formatted OCSP response to stdout if we get one.

Parameters **None** –

Returns None

get_ocsp_response(*certpath: Optional[str]*) → cryptography.hazmat.backends.openssl.ocsp._OCSPResponse
Parse certificate, get and return OCSP response.

Parameters **certpath** – The path of the certificate chain to get OCSP response for (optional)

Returns The OCSPRequest object

classmethod parse_certificate(*certificate_bytes: bytes*) → cryptography.x509.base.Certificate
Parse and return individual certificate, or calls sys.exit(1) if something goes wrong.

Parameters **certificate_bytes** – A chunk of bytes representing a PEM certificate

Returns A cryptography.x509.Certificate object.

classmethod parse_certificate_chain(*certpath: Optional[str]*, *expected_length: Optional[int] = None*) → List[cryptography.x509.base.Certificate]
Parse certificate chain from path or stdin.

Parameters

- **certpath** – The path of the certificate chain to parse (optional), chainbytes are taken from stdin if not provided.
- **expected_length** – The number of certificates to expect. Optional.

Returns A list of cryptography.x509.Certificate objects in the order they appear in the input.

static parse_csr(*csrstring: str = ''*) → cryptography.hazmat.backends.openssl.x509._CertificateSigningRequest
Parse CSR with cryptography.x509.load_pem_x509_csr(), return CSR object.

Takes the CSR data from `sys.stdin` if the `csrstring` argument is empty.

Parameters **csrstring** – The PEM formatted CSR as a string (optional)

Returns The CSR object

static ping_command() → None
Reply to the ping command by outputting the string 'pong' to stdout.

Args: None Returns: None

process_csr(*csrpath: str = ''*) → None
Load the CSR, use it to get a certificate, and cleanup.

Calls `self.parse_csr()` followed by `self.check_csr()`, and then exists if any problems are found with the CSR.

Then `self.get_certificate()` is called, which in turn calls Certbot, which writes the certificate to stdout.

Finally the CSR is deleted.

Parameters `None` –

Returns None

`run_certbot`(*command: List[str]*, *env: Dict[str, str]*, *fullchainpath: str*) → bool
Call certbot, check exitcode, output cert, return bool success.

Parameters

- **command** – A list of certbot command elements
- **env** – A dictionary of the environment to pass to subprocess.run()
- **fullchainpath** – The path to read the certificate+chain from after Certbot runs

Returns True if Certbot command exitcode was 0, False otherwise

`static save_csr`(*csr: cryptography.hazmat.backends.openssl.x509._CertificateSigningRequest*, *path: str*) → None
Save the CSR object to the path in PEM format.

Parameters

- **csr** – The CSR object
- **path** – The path to save it in

Returns None

`static split_pem_chain`(*pem_chain_bytes: bytes*) → List[bytes]
Split a PEM chain into a list of bytes of the individual PEM certificates.

Parameters `pem_chain_bytes` – The bytes representing the PEM chain

Returns A list of 0 or more bytes chunks representing each certificate

`static verify_signature`(*pubkey: Union[cryptography.hazmat.primitives.asymmetric.dsa.DSAPublicKey, cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PublicKey, cryptography.hazmat.primitives.asymmetric.ed448.Ed448PublicKey, cryptography.hazmat.primitives.asymmetric.ec.EllipticCurvePublicKey, cryptography.hazmat.primitives.asymmetric.rsa.RSAPublicKey]*, *signature: bytes*, *payload: bytes*, *hashalgo: cryptography.hazmat.primitives.hashes.HashAlgorithm*) → bool
Verify a signature on a payload using the provided public key and hash algorithm.

Supports RSA and EC public keys. Assumes PKCS1v15 padding for RSA keys.

Parameters

- **pubkey** – The public key
- **signature** – The bytes representing the signature
- **payload** – The bytes representing the signed data
- **hashalgo** – The hashing algorithm used for the signature

CHAPTER

SIX

CERTGRINDER

The `certgrinder` client is responsible for generating a keypair and a CSR, which it uses to contact the Certgrinder server `certgrinderd` over SSH to get a signed certificate. The following section explains how to install and configure it to get certificates from the Certgrinder server.

6.1 Installation

This section explains the steps to install a Certgrinder client. Repeat these steps on each server in need of certificates!

6.1.1 Install certgrinder

You can install certgrinder from pip with `pip install certgrinder`. This will install the latest `certgrinder` release. It will pull in the dependencies it needs automatically.

You can also checkout the Github repo and install the deps from requirements.txt by hand if you prefer. If you want to install with pip directly from Git the following may help: `pip install "git+https://github.com/tykling/certgrinder/#egg=certgrinder&subdirectory=client"`

6.1.2 Create Certgrinder User

Since `certgrinder` is designed to be run under a separate system user one should be created. The user needs sudo access if (and only if) it is to be able to reload/restart services after renewing certificates. Sometimes it is also necessary to add some system users to the certgrinder group so they can read certificates. More on that later.

The user also needs to run `ssh-keygen` and the SSH key needs to be added to the `authorized_keys` file on the Certgrinder server. Make sure to test the SSH access works (hint: check firewalls, v4 vs v6 etc).

6.1.3 Configuration

Configuration of `certgrinder` can be done using command-line options, or a configuration file, or a combination of the two.

The `certgrinder` configuration file is in YAML format. An example config named `certgrinder.conf.dist` can be found in the distribution. use `--config-file` or `-f` to specify the config file location.

Each config item can be specified either in the YAML config file as a `key: value` pair, or on the commandline as `--key value` - the latter overriding the former if both are present. For example, if the configfile has `log-level: INFO` and the command-line has `--log-level: DEBUG` then the effective log-level would be `DEBUG`.

This is an alphabetical list of the configurable options:

alternate-chain Instruct certgrinder to request the alternate chain for signing.

For production this means using the short chain with 1 intermediate signed by `ISRG Root X1` instead of using the default long chain with 2 intermediates signed by `DST Root CA X3`.

For staging it means using `Fake LE Root X2` (1 intermediate) instead of the usual `Fake LE Root X1` (2 intermediates).

Default: `False`

certgrinderd The command to run as `certgrinderd`. Usually this will be something like `ssh certgrinderd@certgrinder.example.com -T`, possibly also with a `--config-file` for `certgrinderd` if needed.

Default: `None`

cert-renew-threshold-days A certificate will be renewed when it has less than this many days of lifetime left.

Default: `30`

domain-list Comma-separated lists of domains for the certificates. Can be specified multiple times on the command-line, `--domain-list example.com,www.example.com --domain-list example.net` means two certificates, the first with two names, the second with one name.

Default: `None`

invalid-ca-cn-list List of CommonName of certificate issuers to consider invalid. This is not a regular CA certificate validity check, it is used to detect certificates issued by LetsEncrypt staging servers as invalid.

Default: `["Fake LE Intermediate X1", "Fake LE Intermediate X2"]`

log-level Sets the verbosity level for console and syslog logging. One of DEBUG, INFO, WARNING, ERROR, CRITICAL.

Default: `INFO`

key-type-list List of key types to enable. Supported choices are *rsa* and *ecdsa*. Files for each keytype will be suffixed with *.rsa.ext* and *ecdsa.ext*, respectively.

Default: `["rsa", "ecdsa"]`

name-server Set this to a DNS server IP (v4 or v6, no hostnames) to use that DNS server instead of the system resolver.

Default: `None`

ocsp-renew-threshold-percent The amount of time in percent between `produced_at` and `next_update` that must have passed before an OCSP response is considered too old. As of January 2021 LetsEncrypt has 7 days between `produced_at` and `next_update` in OCSP responses, so the default of 50% means OCSP responses will be renewed after 3.5 days (half of the validity period) has passed.

As of January 2021 LetsEncrypt produces new OCSP responses after half of the validity period has passed, so any setting lower than that will be pointless. Setting this lower than 50 will just result in Certgrinder fetching the same OCSP response over and over.

Set to 0 to always renew OCSP responses regardless of their age.

Default: `50`

path The directory used for keys, CSRs and certificates. Must exist and be writable by the user running Certgrinder.

Default: `None`

periodic-sleep-minutes Certgrinder will pick a random number of minutes between 0 and this number and sleep for that long before doing periodic actions. Set to 0 to disable sleeping.

Default: `60`

pid-dir The directory to place the certgrinderd PID file in.

Default: `/tmp`

post-renew-hooks A list of commands which `certgrinder` must run after renewing one or more certificates or OCSP responses. Use this to reload/restart services which need to be poked after the certificate changes. Can be specified multiple times on the command-line. Remember to include sudo or whatever if needed. Wrap complex commands in a small shell script to avoid quoting issues.

Default: `None`

post-renew-hooks-dir A path to a hooks.d style directory containing files to be executed after renewing one or more certificates or OCSP responses. Each executable file in this path will be run in the order returned by `os.listdir()`. Set `post-renew-hooks-dir-runner` if something like `sudo` is needed to elevate privileges before running the hooks.

Default: `None`

post-renew-hooks-dir-runner When this is set it will be executed in place of each executable in `post-renew-hooks-dir` with the executable as argument.

Example: If `post-renew-hooks-dir` contains two executable files `hook1` and `hook2` and `post-renew-hooks-dir-runner` is set to `/usr/local/bin/sudo` then certgrinder will execute `/usr/local/bin/sudo /path/to/hooks/dir/hook1` and then `/usr/local/bin/sudo /path/to/hooks/dir/hook2` instead of executing the two hooks directly.

Default: `None`

staging Enable staging mode. Adds `--staging` to the `certgrinderd` command, and considers certificates issued by LE staging servers as valid.

Default: `False`

syslog-facility Set this and syslog-socket to enable logging to syslog. Must be a value supported by `logging.handlers.SysLogHandler` like `LOG_USER` or `LOG_LOCAL0`.

Default: `None`

syslog-socket Set this and syslog-facility to enable logging to syslog.

Default: `None`

tlsa-port Set this to the port (like `443`) when using `show tlsa` or `check tlsa` subcommands.

Default: `None`

tlsa-protocol Set this to the protocol (like `tcp`) when using `show tlsa` or `check tlsa` subcommands.

Default: `None`

tlsa-type-list Set this to enable a TLSA type (can be specified multiple times). The TLSA type must be specified as three integers, one of: `310`, `311` or `312`. Default: is all three pubkey types.

Default: `["310", "311", "312"]`

6.1.4 ACME Challenges

Finally you need to choose which challenge type to use for this `certgrinder` client. If `DNS-01` you need to create one or more `CNAME` record pointing somewhere. If `HTTP-01` you need to create an HTTP redirect. See the section on challenges for more info.

6.1.5 Testing

At this point you should be ready to test! Start by checking with SSH manually to see that the SSH key is installed properly on the Certgrinder server, and firewalls are open. Certgrinder has a `--staging` switch which makes `certgrinderd` use the LetsEncrypt staging environment. Use this until everything works! `certgrinder` outputs some info on what happens, and can output more with `-d / --debug`, but sometimes you need to check syslog on the Certgrinder server.

6.1.6 Crontab job

I run Certgrinder daily, although by default it only attempts certificate renewal when less than 30 days validity remains, and OCSP response renewal when half the validity period has passed.

When everything above works it is time to automate it by adding it to crontab. The following line works for me (the `periodic` command sleeps a random number of minutes before doing its thing, so all the clients don't contact the Certgrinder server at once):

```
0 2 * * * certgrinder /usr/home/certgrinder/virtualenv/bin/certgrinder -c /usr/home/
↪certgrinder/certgrinder.conf periodic
```

6.2 Client Commands

All the functionality in Certgrinder can be accessed by using `commands` and `subcommands`. The following `commands` are available:

- *check certificate command*
- *check connection command*
- *check ocsp command*
- *check tlsa command*
- *get certificate command*
- *get ocsp command*
- *help command*
- *periodic command*
- *show certificate command*
- *show configuration command*
- *show ocsp command*
- *show paths command*
- *show spki command*
- *show tlsa command*

- *version command*

6.2.1 check commands

All the subcommands for the `check` commands return exit code 1 if a problem is found and 0 if everything is fine.

check certificate command

The `check certificate` subcommand loops over the configured domainsets and checks the validity of the certificate for each. If a problem is found `certgrinder` will exit with exit code 1, if all is well the exit code will be 0.

check connection command

The `check connection` subcommand simply checks that the connection to the `certgrinderd` server works as expected. It calls the `ping` command on the `certgrinderd` server and expects to see the string `pong` on stdout. If the expected string is found the exit code will be 0, if a problem is found the exit code will be 1.

check ocsp command

The `check ocsp` subcommand loops over the configured domainsets and checks for the existance of an OCSP response for each. If an OCSP response for a certificate is missing or too old `certgrinder` will exit with exit code 1, if all is well the exit code will be 0.

An OCSP response is considered too old when more than `ocsp-renew-threshold-percent` percent of the time between `producedAt` and `nextUpdate` has passed. As of January 2021 LetsEncrypt has 7 days (one week) between `producedAt` and `nextUpdate` which means OCSP responses will be renewed after 3.5 days with the default `ocsp-renew-threshold-percent` setting of `50`.

check tlsa command

The `check tlsa` subcommand is like the `show tlsa` subcommand but it goes one step further and actually checks in the DNS if the records could be found, and prints some output accordingly. The following example shows two runs of `check tlsa` mode. The first run finds no TLSA records and outputs what needs to be added:

```
[certgrinder@znc ~]$ ./virtualenv/bin/certgrinder -f certgrinder.conf check tlsa 443 tcp
2018-02-16 08:59:39 +0000 INFO: Processing domains: znc.tyknet.dk
2018-02-16 08:59:39 +0000 INFO: Looking up TLSA records for _443._tcp.znc.tyknet.dk
2018-02-16 08:59:39 +0000 WARNING: No TLSA records for name _443._tcp.znc.tyknet.dk of␣
↪type 3 1 0 was found in DNS. This record needs to be added:
2018-02-16 08:59:39 +0000 WARNING: _443._tcp.znc.tyknet.dk 3 1 0␣
↪30820222300d06092a864886f70d0101010500038202f003082020a0282020100bb852c1035ee7ce08d69a13f5cca95374dc
2018-02-16 08:59:39 +0000 WARNING: No TLSA records for name _443._tcp.znc.tyknet.dk of␣
↪type 3 1 1 was found in DNS. This record needs to be added:
2018-02-16 08:59:39 +0000 WARNING: _443._tcp.znc.tyknet.dk 3 1 1␣
↪5b95cb6ea387570f1f3dc4508794ca13a17a665733bab5f76b1e330f2fa13361
2018-02-16 08:59:39 +0000 WARNING: No TLSA records for name _443._tcp.znc.tyknet.dk of␣
↪type 3 1 2 was found in DNS. This record needs to be added:
2018-02-16 08:59:39 +0000 WARNING: _443._tcp.znc.tyknet.dk 3 1 2␣
↪24d49f3c974129b9c28b5e6213892a404d8e9777c5a2e977333b88442d4e16ac0bc732001ec783df795c194704149bd18bbca
2018-02-16 08:59:39 +0000 INFO: Done processing domains: znc.tyknet.dk
[certgrinder@znc ~]$
```

The second run is after adding the suggested records to DNS:

```
[certgrinder@znc ~]$ ./virtualenv/bin/certgrinder -f certgrinder.conf check tlsa 443 tcp
2018-02-16 09:16:27 +0000 INFO: Processing domains: znc.tyknet.dk
2018-02-16 09:16:27 +0000 INFO: Looking up TLSA records for _443._tcp.znc.tyknet.dk
2018-02-16 09:16:27 +0000 INFO: TLSA record for name _443._tcp.znc.tyknet.dk type 3 1 0␣
↪found in DNS matches the local key, good.
2018-02-16 09:16:27 +0000 INFO: TLSA record for name _443._tcp.znc.tyknet.dk type 3 1 1␣
↪found in DNS matches the local key, good.
2018-02-16 09:16:27 +0000 INFO: TLSA record for name _443._tcp.znc.tyknet.dk type 3 1 2␣
↪found in DNS matches the local key, good.
2018-02-16 09:16:27 +0000 INFO: Done processing domains: znc.tyknet.dk
[certgrinder@znc ~]$
```

All TLSA records for this public key can now be found in the DNS.

NOTE: As there might be additional records for the same name which do not belong to this server/key (for example in a loadbalanced or anycast setup), no attempts are made to warn about wrong/old/superfluous TLSA records. This might be added in a future version as a switch to tell Certgrinder that the local public key is the only one in existence for this service.

6.2.2 get commands

The `get` subcommands do all the real work.

get certificate command

The `get certificate` subcommand loops over the configured domainsets and gets a new certificate for each, regardless of the current status of existing certificates. Use with care, only for troubleshooting. *Do not use from cron. Use the periodic command instead.*

get ocsp command

The `get ocsp` subcommand loops over the configured domainsets and gets a new OCSP response for each, regardless of the current status of existing OCSP responses. *Do not use from cron. Use the periodic command instead.*

6.2.3 help command

The `help` command is just a shortcut for `-h` which shows commandline usage and help.

6.2.4 periodic command

The `periodic` command sleeps for a random number of minutes between 0 and the config setting `periodic-sleep-minutes` before doing anything. Set this setting to 0 to disable sleeping.

After sleeping the certificates and OCSP responses are checked and renewed as needed. This command is meant to be run daily from cron or similar.

6.2.5 show commands

The `show` subcommands show information but never change anything.

show certificate command

The `show certificate` subcommand loops over configured domainsets and outputs information about each certificate (if any).

show configuration command

The `show configuration` subcommand just dumps the active configuration as a pretty printed JSON object and exits. Useful for testing or debugging configuration issues.

show ocsp command

The `show ocsp` subcommand loops over the configured domainsets and shows info about each OCSP response.

show paths command

The `show paths` subcommand loops over the configured domainsets and outputs the paths used for keys, certificates and OCSP responses.

show spki command

The `show spki` subcommand outputs pin-sha256 spki pins for the public keys. The `HPKP` standard https://en.wikipedia.org/wiki/HTTP_Public_Key_Pinning defined the `pin-sha256` format for public key pins. While the `HPKP` standard didn't get much traction the pinning format is used in various places now, so `certgrinder` can generate them.

The operation is pretty simple:

```
[certgrinder@znc ~]$ ./virtualenv/bin/certgrinder -f certgrinder.conf show spki
2018-02-16 09:28:37 +0000 INFO: Processing domains: znc.tyknet.dk
2018-02-16 09:28:37 +0000 INFO: pin-sha256="W5XLbqOHVw8fPcRQh5TKE6F6ZlczurX3ax4zDy+hM2E="
2018-02-16 09:28:37 +0000 INFO: Done processing domains: znc.tyknet.dk
[certgrinder@znc ~]$
```

show tlsa command

The `show tlsa` subcommand loops over the configured domainsets and generates `TLSA` records for the public keys. The result is printed to the terminal in a format suitable for putting in the `DNS`. It looks something like this:

```
[certgrinder@znc ~]$ ./virtualenv/bin/certgrinder -f certgrinder.conf show tlsa 443 tcp
2018-02-16 08:42:18 +0000 INFO: Processing domains: znc.tyknet.dk
2018-02-16 08:42:18 +0000 INFO: TLSA records for _443._tcp.znc.tyknet.dk:
2018-02-16 08:42:18 +0000 INFO: _443._tcp.znc.tyknet.dk 3 1 0␣
↪30820222300d06092a864886f70d01010105000382020f003082020a0282020100bb852c1035ee7ce08d69a13f5cca95374dc
2018-02-16 08:42:18 +0000 INFO: _443._tcp.znc.tyknet.dk 3 1 1␣
↪5b95cb6ea387570f1f3dc4508794ca13a17a665733bab5f76b1e330f2fa13361
```

(continues on next page)

(continued from previous page)

```
2018-02-16 08:42:18 +0000 INFO: _443._tcp.znc.tyknet.dk 3 1 2␣
↪24d49f3c974129b9c28b5e6213892a404d8e9777c5a2e977333b88442d4e16ac0bc732001ec783df795c194704149bd18bbca
[certgrinder@znc ~]$
```

Shown above is the `show tlsa` subcommand in action. The value supplied should be the port and protocol of the service, in the example above it is a HTTPS service, so the TLSA record is the service hostname prefixed with `_443._tcp.`

6.2.6 version command

The `version` command is just a shortcut for `-v` which shows the Certgrinder version and exits.

6.3 Command Line Usage

Certgrinder version 0.17.2. See the manpage or ReadTheDocs for more info.

```
usage: certgrinder [-h] [-a] [--certgrinderd CERTGRINDERD]
                   [--cert-renew-threshold-days CERT-RENEW-THRESHOLD-DAYS]
                   [-c CONFIG-FILE] [-d] [-D DOMAIN-LIST]
                   [--invalid-ca-cn-list INVALID-CA-CN-LIST]
                   [-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}] [-k {rsa,ecdsa}]
                   [-n NAME-SERVER] [--now] [-o OCSP-RENEW-THRESHOLD-PERCENT]
                   [--path PATH]
                   [--periodic-sleep-minutes PERIODIC-SLEEP-MINUTES]
                   [-p PID-DIR] [--post-renew-hooks POST-RENEW-HOOKS]
                   [--post-renew-hooks-dir POST-RENEW-HOOKS-DIR]
                   [--post-renew-hooks-dir-runner POST-RENEW-HOOKS-DIR-RUNNER]
                   [-q] [-s] [--syslog-facility SYSLOG-FACILITY]
                   [--syslog-socket SYSLOG-SOCKET] [--tlsa-port TLSA-PORT]
                   [--tlsa-protocol TLSA-PROTOCOL]
                   [--tlsa-type-list {310,311,312}] [-v]
                   {check,get,help,periodic,show,version} ...
```

6.3.1 Positional Arguments

command Possible choices: check, get, help, periodic, show, version

Command (required)

6.3.2 Named Arguments

-a, --alternate-chain Use alternate chain. For production this means using the short chain with 1 intermediate signed by 'ISRG Root X1' instead of using the long chain with 2 intermediates signed by 'DST Root CA X3'. For staging it means using 'Fake LE Root X2' (1 intermediate) instead of the usual 'Fake LE Root X1' (2 intermediates).

--certgrinderd The command to reach the certgrinderd server, will get the input (CSR or cert chain) on stdin. Usually something like 'ssh certgrinderd@server -T'

--cert-renew-threshold-days A certificate is renewed when it has less than this many days of lifetime left. Default: *30*

-c, --config-file The path to the certgrinder.yml config file to use

-d, --debug Debug mode. Equal to setting –log-level=DEBUG.

-D, --domain-list Comma separated list of domains for a certificate. Can be specified multiple times.

--invalid-ca-cn-list The CommonName of an issuer (CA intermediate) to consider invalid. Can be specified multiple times.

-l, --log-level Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL

Logging level. One of DEBUG, INFO, WARNING, ERROR, CRITICAL. Defaults to INFO.

-k, --key-type-list Possible choices: rsa, ecdsa

The keytypes to enable. Valid values are 'rsa' and 'ecdsa'. Can be specified multiple times. Defaults to both rsa and ecdsa.

-n, --name-server Tell certgrinder to use this DNS server IP to lookup TLSA records. Only relevant with -c / –checktlsa. Only v4/v6 IPs, no hostnames.

--now Run periodic command without delay. Equal to setting –periodic-sleep-minutes 0.

-o, --ocsp-renew-threshold-percent Possible choices: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100

An integer between 0 and 100 specifying the amount of time in percent between `produced_at` and `next_update` which must have passed before an OCSP response is considered too old. Defaults to 50.

--path Tell certgrinder to use the specified directory for keys, CSRs and certificates. The directory must exist and be writeable by the user running certgrinder.

--periodic-sleep-minutes Tell certgrinder to sleep for a random number of minutes between 0 and this number before doing anything when the periodic command is used. Set to 0 to disable sleeping.

-p, --pid-dir The directory to store the PID file in

--post-renew-hooks The list of commands to run after one or more certificates are renewed. Most such commands will need root access to run, remember to prefix the command with 'sudo' as needed. Can be specified multiple times. Default: *None*

--post-renew-hooks-dir Path to a folder containing executables to run after one or more certificates or OCSP responses are renewed. These will execute under the regular certgrinder

user uid, so make sure to use sudo/doas in scripts or suid executables as needed. Default: *None*

--post-renew-hooks-dir-runner Path to an executable like sudo to be used to run each of the executables in the post renew hooks dir. Default: *None*

-q, --quiet Quiet mode. No output at all if there is nothing to do, and no errors are encountered. Equal to setting –log-level=WARNING.

-s, --staging Staging mode. Sets –acme-server-url https://acme-staging-v02.api.letsencrypt.org/directory and –invalid-ca-cn-list empty. Use this while playing around to avoid hitting rate limits!

--syslog-facility The syslog facility to use. Set this and syslog-socket to enable logging to syslog.

--syslog-socket The syslog socket to connect to. Set this and syslog-facility to enable logging to syslog.

--tlsa-port The service port number (like 443) for TLSA operations.

--tlsa-protocol The service protocol (like tcp) for TLSA operations.

--tlsa-type-list Possible choices: 310, 311, 312

Enables a TLSA type for TLSA operations. Can be specified multiple times.

-v, --version Show version and exit.

6.3.3 Sub-commands:

check

Use the "check" command to check certificates, OCSP responses and TLSA records. Returns exit code 0 if all is well, and 1 if something needs attention.

```
certgrinder check [-h] {certificate,connection,ocsp,tlsa} ...
```

Positional Arguments

subcommand Possible choices: certificate, connection, ocsp, tlsa

Specify what to check using one of the available check sub-commands.

Sub-commands:

certificate

Tell certgrinder to check certificate validity for all configured domainsets. Returns exit code 1 if any problem is found, exit code 0 if all is well.

```
certgrinder check certificate [-h]
```

connection

Tell certgrinder to check the connection to the certgrinderd server by calling the certgrinderd 'ping' command which should return the string 'pong' if all is well.

```
certgrinder check connection [-h]
```

ocsp

Tell certgrinder to check the OCSP response validity for certificates for all configured domainsets. Returns exit code 1 if any problem is found, exit code 0 if all is well.

```
certgrinder check ocsp [-h]
```

tlsa

Tell certgrinder to lookup TLSA records for the given port and protocol in the DNS and compare with what we have locally, for example: 'certgrinder check tlsa 853 tcp'

```
certgrinder check tlsa [-h] tlsa-port tlsa-protocol
```

Positional Arguments

tlsa-port The port of the service, for example 443

tlsa-protocol The protocol of the service, for example tcp

get

Use the "get" command to get certificates and OCSP responses

```
certgrinder get [-h] {certificate,ocsp} ...
```

Positional Arguments

subcommand Possible choices: certificate, ocsp

Specify what to get using one of the available get sub-commands

Sub-commands:

certificate

Tell certgrinder to get new certificate(s), regardless of their current state. Rarely needed, use 'periodic' command instead.

```
certgrinder get certificate [-h]
```

ocsp

Tell certgrinder to get OCSP responses for the configured domainset(s). Rarely needed, use 'periodic' command instead.

```
certgrinder get ocsp [-h]
```

help

The "help" command just outputs the usage help

```
certgrinder help [-h]
```

periodic

The "periodic" command checks certificates and renews them as needed. Meant to be run from cron or similar daily.

```
certgrinder periodic [-h]
```

show

Use the "show" command to show certificates, TLSA records, SPKI pins or configuration.

```
certgrinder show [-h] {certificate,configuration,paths,ocsp,spki,tlsa} ...
```

Positional Arguments

subcommand Possible choices: certificate, configuration, paths, ocsp, spki, tlsa

Specify what to show using one of the available show sub-commands

Sub-commands:

certificate

Tell certgrinder to output information about certificates.

```
certgrinder show certificate [-h]
```

configuration

Tell certgrinder to output the current configuration

```
certgrinder show configuration [-h]
```

paths

Tell certgrinder to output the paths used

```
certgrinder show paths [-h]
```

ocsp

Tell certgrinder to output information about OCSP responses.

```
certgrinder show ocsp [-h]
```

spki

Tell certgrinder to generate and print the pin-sha256 spki pins for the public keys it manages.

```
certgrinder show spki [-h]
```

tlsa

Use the 'show tlsa' sub-command to tell certgrinder to generate and print TLSA records for the given service, for example: 'certgrinder show tlsa 443 tcp'

```
certgrinder show tlsa [-h] tlsa-port tlsa-protocol
```

Positional Arguments

tlsa-port	The port of the service, for example 443
tlsa-protocol	The protocol of the service, for example tcp

version

The "version" command just outputs the version of Certgrinder

```
certgrinder version [-h]
```

6.4 Class Methods

class certgrinder.**Certgrinder**

The Certgrinder client class.

__init__() → None

Define the default config.

check_certificate(*certificate: Optional[cryptography.x509.base.Certificate] = None*, *public_key: Optional[Union[cryptography.hazmat.backends.openssl.rsa._RSAPublicKey, cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PublicKey]] = None*) → bool

Check certificate validity and returns True or False.

This method is called by self.grind() once per domainset when the "check certificate" subcommand is invoked. It reads the certificate from self.certificate_path if there is no certificate arg

Parameters

- **certificate** – The certificate to be checked
- **public_key** – The keypair the certificate is based on

Returns True if everything is OK, False otherwise

static check_certificate_expiry(*certificate: cryptography.x509.base.Certificate*, *threshold_days: int*) → bool

Check the remaining validity of the certificate.

Parameters

- **certificate** – The certificate to check
- **threshold_days** – The lowest number of remaining days of validity that is considered valid

Returns True if remaining certificate lifetime is >= threshold_days, False if not

static check_certificate_issuer(*certificate: cryptography.x509.base.Certificate*, *invalid_ca_cn_list: List[str]*) → bool

Check the issuer of the certificate.

Parameters

- **certificate** – The certificate to check
- **invalid_ca_cn_list** – The list of CA CommonName strings to consider invalid

Returns True if the certificate issuer CN is not in invalid_ca_cn_list

static check_certificate_public_key(*certificate: cryptography.x509.base.Certificate*, *public_key: Union[cryptography.hazmat.backends.openssl.rsa._RSAPublicKey, cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PublicKey]*) → bool

Make sure certificate has the specified public key.

Parameters

- **certificate** – The certificate to check
- **public_key** – The public key

Returns True if the public key matches, False if not

static check_certificate_san_names(*certificate: cryptography.x509.base.Certificate*, *san_names: List[str]*) → bool

Make sure the certificate has the provided list of names as SAN.

Parameters

- **certificate** – The certificate to check
- **san_names** – A list of the names to expect

Returns True if all san_names were found in the cert, and no others.

static check_certificate_subject(*certificate: cryptography.x509.base.Certificate*, *subject: str*) → bool

Make sure the certificate has the specified subject.

Parameters

- **certificate** – The certificate to check
- **subject** – The subject to expect

Returns True if the subject matches the cert, False if not

classmethod check_certificate_validity(*certificate: cryptography.x509.base.Certificate*, *invalid_ca_cn_list: List[str]*, *threshold_days: int*, *san_names: List[str]*, *public_key: Optional[Union[cryptography.hazmat.backends.openssl.rsa._RSAPublicKey, cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PublicKey]] = None*, *subject: str = ''*) → bool

Perform a few sanity checks of the certificate.

- Check that the issuer is valid
- Check that the certificate expiry is not exceeded
- Check that the public key is correct
- Check that the subject is correct
- Check that the SubjectAltName data is correct

Parameters

- **certificate** – The certificate to check
- **invalid_ca_cn_list** – A list of CA CommonNames to consider invalid

- **threshold_days** – The minimum number of remaining days lifetime to considered valid.
- **san_names** – A list of domain names to expect in SubjectAltName of the certificate.
- **keypair** – The keypair the certificate is for.

Returns False if a problem is found, True if all is well.

check_connection(*stdout: Optional[bytes] = None*) → bool
The `check connection` subcommand method.

Parameters **stdout** – The certgrinderd response to use instead of calling certgrinderd (optional)

Returns None

check_ocsp() → bool
The `check ocsp` subcommand method, called for each domainset by `self.grind()`.

Returns True if the OCSP response was found and is not too old, False otherwise

check_tlsa() → None
The 'check tlsa' subcommand method, called for each domainset by `self.grind()`.

Loops over the configured TLSA types and calls `self.verify_tlsa_record()` which does the heavy lifting.

Returns None

configure(*userconfig: Dict[str, Union[str, int, bool, List[str]]]*) → None
Merge and check configuration and configure logging.

Merge the supplied userconfig dict with the default config, checks for missing required settings, and configures logging and syslog.

Parameters **userconfig** – dict of the config to be merged with the default config

Returns None

static generate_csr(*keypair: Union[cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey, cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PrivateKey]*, *domains: List[str]*) →
cryptography.hazmat.backends.openssl.x509._CertificateSigningRequest
Generate and return a new CSR based on the public key and list of domains.

Only set CN since everything else is removed by LetsEncrypt in the certificate anyway. Add all domains in subjectAltName, including the one put into CN.

Parameters

- **keypair** – The keypair to base the CSR on
- **domains** – A list of domains to put in the CSR. First in the list will be cert CN.

Returns The CSR object

static generate_private_key(*keytype: str*) →
Union[cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey,
cryptography.hazmat.backends.openssl.ec._EllipticCurvePrivateKey,
cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PrivateKey]
Generate and returns a private key.

Parameters **keytype** – "rsa" for RSA key, "ecdsa" for ECDSA and "ed25519" for ed25519

Returns The keypair object

Raises **ValueError** – For unsupported keytypes

static generate_spki(*derkey: bytes*) → str
Generate and return a pin-sha256 spki hpkp style pin for the provided public key.

OpenSSL equivalent command is: openssl x509 -in example.com.crt -pubkey -noout | openssl pkey -pubin -outform der | openssl dgst -sha256 -binary | openssl base64

Parameters **derkey** – The bytes representing the public key in DER format

Returns A string of the SPKI pin

static generate_tlsa_record(*derkey: bytes*, *tlsatype: str*) → str
Generate and return the data part of a TLSA record of the requested type.

TLSA record is generated from the DER formatted public key supplied. Returns an uppercase hex string.

Parameters

- **derkey** – The bytes representing the public key in DER format
- **tlsatype** – The TLSA type (like "310")

Returns String of the TLSA data

Raises **ValueError** – If an unknown TLSA type is passed

get_certgrinderd_command(*subcommand: List[str]*) → List[str]
Return the certgrinderd command to run.

Adds `--log-level` with the current `self.conf["log-level"]`. Also adds –acme-server-url if configured, and –preferred-chain.

Parameters **subcommand** – The certgrinderd subcommand to run as a list, like ["get", "ocsp"]

Returns A list of the elements which make up the `certgrinderd` command

get_certificate(*csr: Optional[cryptography.hazmat.backends.openssl.x509._CertificateSigningRequest] = None*, *stdout: Optional[bytes] = None*) → bool
Get a new certificate for self.domainset.

This methods gets a new certificate regardless of the status of any existing certificate. It is called by `self.periodic()` as needed. It can also be called by the `get certificate` subcommand.

Parameters

- **csr** – The CSR to use instead of generating one
- **stdout** – The stdout bytes to use instead of calling self.run_certgrinderd(csr)

Returns False something goes wrong, True if all is well

static get_der_pubkey(*keypair: Union[cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey, cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PrivateKey]*) → bytes
Return the DER formatted publickey.

Parameters **keypair** – The keypair which contains the public key

Returns The bytes representing the DER formatted public key

get_filename(*hostname: str*) → str
Calculate the hostname string to be used for filenames.

Files are named after the ascii idna representation of the first hostname in the list (which is also the CN in the subject of the CSR and certificate).

Max filename length on some platforms is 255 bytes, but a hostname could be up to 253 bytes (RFC 1035 section 2.3.4), and we need some room for the usage and keytype and extension, so we only use the last 230 bytes of the ascii idna representation of the hostname for the filename, leaving 25 bytes for metadata.

Parameters **domainset** – The list of hostnames

Returns The string to use in filenames

get_ocsp(*certificate: Optional[cryptography.x509.base.Certificate] = None*, *issuers: List[cryptography.x509.base.Certificate] = []*, *stdout: Optional[bytes] = None*) → bool

The `get ocsp` subcommand method, called for each domainset by `self.grind()`.

Parameters

- **certificate** – The certificate to get OCSP response for (optional)
- **issuers** – The list of issuer(s) of the certificate to get OCSP response for (optional)
- **stdout** – The mock OCSP response to return instead of calling certgrinderd (optional, used for unit tests)

Returns None

grind(*args: argparse.Namespace*) → None

Loop over enabled keytypes and domainsets in `self.conf["domain-list"]` and call args.method for each.

load_certificates(*path: str*) → List[cryptography.x509.base.Certificate]

Reads PEM certificate data from the path, parses the certificate(s), and returns them in a list.

Parameters **path** – The path to read the PEM certificate(s) from

Returns A list of cryptography.x509.Certificate objects

load_domainset(*domainset: List[str]*, *keytype: str*) → None

Prepare paths and create/load private key.

Parameters

- **domainset** – The list of hostnames to load
- **keytype** – The keytype to use, “rsa” or “ecdsa”.

Returns None

static load_keypair(*path: str*) → Union[cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey, cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PrivateKey]

Load keypair bytes from disk, load key and return the object.

Fixes keypair permissions to 640 if they are not 640.

Parameters **path** – The path to load the keypair from

Returns The keypair object

static load_ocsp_response(*path: str*) → cryptography.hazmat.backends.openssl.ocsp._OCSPResponse

Reads OCSP response in DER format from the path and returns the object.

Parameters **path** – The path to read the OCSP response from

Returns The OCSP response object

static lookup_tlsa_record(*domain: str*, *port: int*, *protocol: str*, *tlsatype: Optional[str] = None*, *nameserver: str = ''*) → Optional[List[str]]

Lookup TLSA records in DNS for the configured domain, port, and protocol.

Loop over any responses and look for the requested tlsatype. Return a list of results, optionally limited to the specified tlsatype, or None. Use system resolver unless nameserver is specified.

Parameters

- **domain** – The service domain name (like `mail.example.com`)
- **port** – The service port (like `443`)
- **protocol** – The service protocol (like `tcp`)
- **tlsatype** – The TLSA type (like `312`)
- **nameserver** – The DNS server IP to use instead of system resolver (optional)

Returns A list of records or None

classmethod output_spki(*derkey: bytes*) → None

Get and print the spki pin for the supplied DER public key.

Parameters **derkey** – The bytes representation of the DER formatted public key

Returns None

classmethod output_tlsa_record(*derkey: bytes*, *domain: str*, *port: int*, *protocol: str*, *tlsatype: str*, *warning: bool = False*) → None

Output the TLSA record for the given DER key, domain, port, protocol and tlsatype.

Call `self.generate_tlsa()` and output the result formatted as a DNS record

Parameters

- **derkey** – The bytes representation the public key in DER format
- **domain** – The service domain name (like `mail.example.com`)
- **port** – The service port (like `443`)
- **protocol** – The service protocol (like `tcp`)
- **tlsatype** – The TLSA type (like `312`)
- **warning** – Set True to output at level `WARNING` (default `INFO`)

Returns None

static parse_certgrinderd_ocsp_output(*certgrinderd_stdout: bytes*) → Optional[cryptography.hazmat.backends.openssl.ocsp._OCSPResponse]

Parse a DER encoded binary OCSP response as returned by Certgrinderd.

Parameters **certgrinderd_output** – The bytes representing the OCSP response in DER format

Returns cryptography.hazmat.backends.openssl.ocsp._OCSPResponse

static parse_certificate(*certificate_bytes: bytes*) → Optional[cryptography.x509.base.Certificate]

Parse a bunch of bytes representing a PEM certificate and return.

Parameters **certificate_bytes** – The PEM certificate

Returns The parsed cryptography.x509.Certificate object or None

parse_certificate_chain(*certificate_chain: bytes*, *csr: cryptography.hazmat.backends.openssl.x509._CertificateSigningRequest*) → Optional[List[cryptography.x509.base.Certificate]]

Split a PEM chain into a list of certificates.

Parameters

- **certificate_chain** – The bytes representing the PEM formatted certificate chain
- **csr** – The CSR this certificate was issued from

Returns A list of certificates with the leaf certificate first, or None if an error happens

periodic() → bool
The periodic method performs periodic maintenance tasks.

This method is called by the 'periodic' command, from cron or similar. It starts out by sleeping for a random period and then checks certificates and renews as needed.

run_certgrinderd(*stdin: bytes*, *command: List[str]*, *certgrinderd_stdout: Optional[bytes] = None*, *certgrinderd_stderr: Optional[bytes] = None*) → Optional[bytes]
Run the configured `self.conf["certgrinderd"]` command.

The stdin argument will be passed to stdin of the command. A CSR is needed for the "get certificate" certgrinderd command, and a certificate chain is needed for the "get ocsp" command.

Parameters

- **stdin** – bytes representing CSR or cert chain to pass to the certgrinderd command
- **command** – The certgrinderd command and subcommand to call
- **certgrinderd_stdout** – Mocked certgrinderd stdout to use instead of calling the command
- **certgrinderd_stderr** – Mocked certgrinderd stderr to use instead of calling the command

Returns The bytes representing the stdout from the subprocess call

static run_post_renew_hook(*hook: List[str]*) → bool
Run a specific post renew hook.

Parameters **hook** – A list of string components of the command and arguments

Returns: True if exit code was 0, False otherwise.

run_post_renew_hooks() → bool
Loops over configured post_renew_hooks and executables in post_renew_hooks_dir and runs them.

Returns None

static save_certificate(*certificate: cryptography.x509.base.Certificate*, *path: str*, *issuers: List[cryptography.x509.base.Certificate] = []*) → None
Save the PEM certificate to the path, optionally with an issuer chain.

Parameters

- **certificate** – The certificate to save
- **path** – The path to save the certificate in
- **issuer** – The list of issuer certificates to write after the certificate (if any)

Returns None

classmethod save_concat_certkey(*keypair: Union[cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey, cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PrivateKey]*, *certificate: cryptography.x509.base.Certificate*, *issuers: List[cryptography.x509.base.Certificate]*, *path: str*) → None
Create a single file with the private key, the cert and the issuer(s), in that order.

Parameters

- **keypair** – The keypair to save in the concat file
- **certificate** – The certificate to save in the concat file
- **issuers** – The list of issuer(s) to save in the concat file
- **path** – The path to save the concat file in

Returns None

static save_csr(*csr: cryptography.hazmat.backends.openssl.x509._CertificateSigningRequest*, *path: str*) → None

Save the PEM version of the CSR to the path.

chmods the file 644 after writing.

Parameters

- **csr** – The CSR to be saved
- **path** – The path to save the CSR to

Returns None

static save_keypair(*keypair: Union[cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey, cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PrivateKey]*, *path: str*) → None

Save keypair to disk.

Parameters

- **keypair** – The keypair to save
- **path** – The path to save the keypair in

Returns None

Raises **ValueError** – For unsupported keytypes

static save_ocsp_response(*ocsp_response: cryptography.hazmat.backends.openssl.ocsp._OCSPResponse*, *path: str*) → None

Save the OCSP response to disk in DER format.

Parameters

- **ocsp_response** – The OCSP response to save
- **path** – The path to save in

Returns None

show_certificate() → None

The `show certificate` subcommand method, called for each domainset by `self.grind()`.

Returns None

show_ocsp() → None

The `show ocsp` subcommand method, called for each domainset by `self.grind()`.

Returns None

show_paths() → None

The `show paths` subcommand method, called for each domainset by `self.grind()`.

Returns None

show_spki() → None
The `show spki` subcommand method, called for each domainset by `self.grind()`.

Call `self.output_spki()` with the DER formatted public key and output the result.

Returns None

show_tlsa() → None
The 'show tlsa' subcommand method, called for each domainset by `self.grind()`.

Returns None

static split_pem_chain(*pem_chain_bytes: bytes*) → List[bytes]
Split a PEM chain into a list of bytes of the individual PEM certificates.

Parameters **pem_chain_bytes** – The bytes representing the PEM chain

Returns A list of 0 or more bytes chunks representing each certificate

classmethod verify_tlsa_record(*derkey: bytes*, *domain: str*, *port: int*, *protocol: str*, *tlsatype: str*, *nameserver: str = ''*) → bool
Check the TLSA records for the port/protocol/domain and DER key in the DNS.

Output the info needed to fix things when missing records are found.

Parameters

- **derkey** – The bytes representation the public key in DER format
- **domain** – The service domain name (like `mail.example.com`)
- **port** – The service port (like `443`)
- **protocol** – The service protocol (like `tcp`)
- **tlsatype** – The TLSA type (like `312`)
- **nameserver** – The DNS server IP to use instead of system resolver (optional)

Returns True if all is well, False if one or more problems are found

CHAPTER

SEVEN

CERTGRINDERD CHANGE LOG

This is the changelog for `certgrinderd`. The latest version of this file can always be found on Github

All notable changes to `certgrinderd` will be documented in this file.

This project adheres to Semantic Versioning.

7.1 v0.17.2 (27-nov-2021)

7.1.1 Changed

- Include Python 3.10 support
- Update setup.py to include license_file
- Update description in setup.py

7.2 v0.17.1 (21-nov-2021)

7.2.1 Changed

- Update dependency PyYAML==5.4.1 to PyYAML==6.0
- Update dependency certbot==1.15.0 to certbot==1.21.0
- Update dependency requests==2.25.1 to requests==2.26.0
- Update a bunch of development dependencies
- Switch to Github Actions instead of Travis CI

7.3 v0.17.0 (21-may-2021)

- No changes since v0.17.0-rc3

7.4 v0.17.0-rc3 (21-may-2021)

7.4.1 Fixed

- Replace underscores with spaces in the preferred-chain name
- Do not check number of intermediates when getting OCSP.

7.5 v0.17.0-rc2 (20-may-2021)

7.5.1 Fixed

- Replace spaces with underscores in chain names to get around quoting woes in the SSH commands

7.6 v0.17.0-rc1 (20-may-2021)

7.6.1 Added

- New config and command-line option `preferred-chain` can be used to ask the ACME server (LetsEncrypt) to sign with the specified chain. This is used by the certgrinder clients `alternate-chain` option to ask for primary or alternate chain for staging or prod. The value of the option is passed directly to Certbot as `--preferred-chain`.

7.6.2 Changed

- Refactor a bunch of code to support the new two-intermediates chain from LetsEncrypt
- Upgrade dependencies

7.7 v0.16.0 (18-Jan-2021)

7.7.1 Added

- Certgrinderd now keeps a pidfile while running to prevent running multiple times simultaneously.
- New `ping` command used by the `certgrinder` command `check connection` to check connection to the `certgrinderd` server without doing anything else.
- Python3.9 support

7.7.2 Fixed

- IDN domain handling now works again

7.8 v0.15.1 (29-Sep-2020)

- No changes

7.9 v0.15.0 (29-Sep-2020)

- No changes

7.10 v0.15.0-beta2 (28-Sep-2020)

- No changes

7.11 v0.15.0-beta1 (28-Sep-2020)

7.11.1 Added

- Enabled `check-spelling` Github action and fixed a bunch of misspelled words all over.

7.11.2 Fixed

- Removed unused `--rsa-key-size` arg from certbot command
- Fix wrong requirements line for pre-commit (remove extra equal sign)
- Updated all dependencies in requirements.txt, and switch to pinning deps with == rather than >= so dependabot on github can do its thing

7.12 v0.14.2 (13-Sep-2020)

- No changes

7.13 v0.14.1 (13-Sep-2020)

7.13.1 Changed

- Change `intermediate` to `issuer` in the code and tests.

7.13.2 Fixed

- Fix a wrong error message in an assert in `test_parse_certificate_chain_path()`

7.14 v0.14.0 (29-Aug-2020)

7.14.1 Changed

- Change log message to `INFO` when getting a new OCSP response

7.15 v0.14.0-beta2 (29-Aug-2020)

- No changes

7.16 v0.14.0-beta1 (29-Aug-2020)

7.16.1 Added

- Introduce commands and subcommands (like “get certificate” or “get ocsp”)
- Add OCSP response fetching and verifying functionality
- Add requests to requirements (for getting OCSP responses)
- Tests for the new functionality

7.16.2 Changed

- Refactor code to fit the commands/subcommand structure
- Log certbot stderr at level ERROR
- Change some default config from None to “” to keep it as str

7.17 v0.13.2 (11-Jul-2020)

7.17.1 Added

- Manpage to MANIFEST.in to include it in the distribution

7.18 v0.13.1 (7-Jul-2020)

7.18.1 Changed

- Specify python3.7 and 3.8 as classifiers in setup.py

7.19 v0.13.0 (7-Jul-2020)

- No changes

7.20 v0.13.0-rc1 (1-Jul-2020)

7.20.1 Added

- Information about $SSH_ORIGINAL_COMMAND to docs

7.20.2 Changed

- Show current log-level setting in first log message

7.21 v0.13.0-beta2 (29-Jun-2020)

7.21.1 Added

- Dev requirements now has `sphinx-rtd-theme` which is the theme used on ReadTheDocs, so `make html` in `docs/` now produces the same-ish output.
- Dev requirements now include `sphinx-argparse` used for generating automatic usage documentation.
- Short command-line options for a bunch of things.
- Manpage certgrinderd.8
- Unittests for a few Certgrinderd() methods

7.21.2 Changed

- Move CHANGELOG.md to rst format and into `docs/`
- Split certbot-command related stuff into new methods get_certbot_command() and run_certbot()
- Split creating the argparse object into a separate function to assist sphinx-argparse
- Test suite now covers 100% of certgrinderd.py

7.21.3 Fixed

- Only try challenge types if we have the needed info (acme-zone for DNS-01, www-root for HTTP-01)

7.22 v0.13.0-beta1 (7-May-2020)

- No changes

7.23 v0.13.0-alpha8 (6-May-2020)

7.23.1 Changed

- Changed logformat to prefix messages with certgrinderd: and Certgrinderd. instead of nothing and %(name)s, making it more clear which messages are from certgrinder and which are from certgrinderd.

7.24 v0.13.0-alpha7 (6-May-2020)

- No changes

7.25 v0.13.0-alpha6 (6-May-2020)

7.25.1 Changed

- `certgrinderd` now creates a temporary directory for temporary CSR and CRT files per run. The directory and contents is at the end of each run. If –temp-dir is configured the temporary directory is created inside the path specified.

7.26 v0.13.0-alpha5 (6-May-2020)

7.26.1 Added

- -f and -S short options for –config-file and –staging
- MANIFEST.in file to include sample config and hook scripts

7.27 v0.13.0-alpha4 (5-May-2020)

7.27.1 Added

- New –log-level option to set logging verbosity. Must be one of DEBUG, INFO, WARNING, ERROR, CRITICAL, corresponding to the levels in the Python logging framework.
- A lot of new documentation about `certgrinderd`
- Command-line options for everything

7.27.2 Changed

- Configuration file and command-line options aligned so everything is configurable both places.

7.28 v0.13.0-alpha3 (5-May-2020)

7.28.1 Added

- Add missing PyYAML dependency in setup.py

7.28.2 Changed

- Fix so certgrinderd.conf certbot_commands with spaces in them work as expected

7.29 v0.13.0-alpha2 (4-May-2020)

7.29.1 Added

- Install `certgrinderd` binary using entry_points in setup.py

7.29.2 Changed

- Move CSR loading and testing to class methods in the Certgrinderd class
- Wrap remaining script initialisation in a main() function to support entry_points in setup.py better

7.30 v0.13.0-alpha (4-May-2020)

7.30.1 Added

- Create Python package `certgrinderd` for the Certgrinder server, publish on pypi
- Add isort to pre-commit so imports are kept neat
- Tox and pytest and basic testsuite using Pebble as a mock ACME server
- Travis and codecov.io integration

7.30.2 Changed

- Move client files into client/ and server files into server/, each with their own CHANGELOG.md
- Rename server from csrgrinder to certgrinderd
- Rewrite server in Python

CHAPTER

EIGHT

CERTGRINDER CHANGE LOG

This is the changelog for `certgrinder`. The latest version of this file can always be found on Github

All notable changes to `certgrinder` will be documented in this file.

This project adheres to Semantic Versioning.

8.1 v0.17.2 (27-nov-2021)

8.1.1 Changed

- Include Python 3.10 support
- Update setup.py to include license_file
- Update description in setup.py

8.2 v0.17.1 (21-nov-2021)

8.2.1 Changed

- Update dependency PyYAML==5.4.1 to PyYAML==6.0
- Cryptography 35.0.0 is incompatible with Certgrinder v0.17.x so the Cryptography dependency has been pinned to a version lower than <35 in setup.py. Next version of Certgrinder will support cryptography v35.0.0 and newer.
- Update a bunch of development dependencies
- Switch to Github Actions instead of Travis CI

8.3 v0.17.0 (21-may-2021)

- No changes since v0.17.0-rc3

8.4 v0.17.0-rc3 (21-may-2021)

- No changes since v0.17.0-rc2

8.5 v0.17.0-rc2 (20-may-2021)

8.5.1 Fixed

- Replace spaces with underscores in chain names to get around quoting woes in the SSH commands

8.6 v0.17.0-rc1 (20-may-2021)

8.6.1 Added

- New config option `alternate-chain` to tell certgrinderd to tell Certbot to tell LetsEncrypt to use the alternate chain. Sets the certgrinderd option `preferred-chain` to the appropriate value accordingly.

8.6.2 Fixed

- Support the new longer chain from LetsEncrypt (with two intermediates).
- Use `shlex` to parse certgrinderd command instead of just splitting on spaces

8.6.3 Changed

- Refactor a bunch of code to deal with multiple intermediates
- Upgrade dependencies

8.7 v0.16.0 (18-Jan-2021)

8.7.1 Added

- New config option `ocsp-renew-threshold-percent` to specify the amount of time in percent which must have passed before an OCSP response is considered too old. The new option defaults to 50% which matches when LetsEncrypt currently issues new OCSP responses, which is after half the time between produced_at and next_update has passed.
- Certgrinder now keeps a pidfile while running to prevent running multiple times simultaneously.
- New `check connection` command to check connection to the `certgrinderd` server without doing anything else.
- New config options `post-renew-hooks-dir` and `post-renew-hooks-dir-runner`. The former can be used to specify a path to a directory containing executables to run after a certificate or OCSP response has been renewed. The latter can be used to specify a runner like `sudo` to be used to run all the hooks. The existing `post-renew-hooks` setting will continue to work as expected.
- Python3.9 support

8.7.2 Removed

- Config option `ocsp-renew-threshold-seconds` was removed and replaced with `ocsp-renew-threshold-percent`.

8.7.3 Fixed

- Show keytype in `show ocsp` output
- The new `ocsp-renew-threshold-percent` code and default setting eliminates redundant OCSP response fetching
- IDN domain handling now works again

8.7.4 Changed

- Better logging when running post renew hooks - exit code is always logged, and the time spent running each hook is now logged.

8.8 v0.15.1 (29-Sep-2020)

8.8.1 Fixed

- Check OCSP response age and get a new one when needed

8.8.2 Added

- Configuration option `ocsp-renew-threshold-seconds` - defaults to 86400.

8.9 v0.15.0 (29-Sep-2020)

8.9.1 Changed

- Change output a bit for the `show tlsa` subcommand

8.9.2 Fixed

- The `show tlsa` command did not work due to type mismatch triggering an assert
- Show keytype in the `show certificate` output

8.10 v0.15.0-beta2 (28-Sep-2020)

8.10.1 Changed

- Check if files exist in the `show paths` subcommand.

8.11 v0.15.0-beta1 (28-Sep-2020)

8.11.1 Added

- Enabled ECDSA keys and certificates. Default to getting both RSA and ECDSA certificates. Control which keytypes are enabled with the new `key-type-list` configuration option. Curve for ECDSA is SECP384R1, this might be made configurable later.
- Added `show paths` subcommand to output the various filepaths used.
- Enabled `check-spelling` Github action and fixed a bunch of misspelled words all over.

8.11.2 Changed

- Changed filenames of keys and certificates. Run the following commands to rename existing RSA files from pre 0.15 installs:
  - The keypair: `mv example.com.key example.com-keypair.rsa.key`
  - The CSR: `mv example.com.csr example.com-request.rsa.csr`
  - The certificate chain: `mv example.com.crt example.com-chain.rsa.crt`
  - The certificate: `mv example.com-certonly.crt example.com-certificate.rsa.crt`
  - The concat key and chain: `mv example.com-concat.pem example.com-concat.rsa.pem`
  - The issuer certificate: `mv example.com-issuer.crt example.com-issuer.rsa.crt`
  - The OCSP response: `mv example.com.ocsp example.com-response.rsa.ocsp`

  In other words: - All files got the keytype (always `rsa` for pre-0.15 files) inserted just before the extension, so `.crt` becomes `.rsa.crt` and `.key` becomes `.rsa.key`. - Additionally the keypair files got `-keypair` inserted just after the hostname, so `example.com.rsa.key` becomes `example.com-keypair.rsa.key`. - Additionally the CSR files got `-request` inserted just after the hostname, so `example.com.rsa.csr` becomes `example.com-request.rsa.csr`. - Finally the OCSP response got `-response` inserted just after the hostname, so `example.com.rsa.ocsp` becomes `example.com-response.rsa.ocsp`.

  This rename must be done for each domainset. If a keypair with the old filename is found Certgrinder will quit with exit code 1 and refuse to run. Use the new `show paths` subcommand to figure out what the new filenames should be.
- Prefix certgrinderd output with `certgrinderd:` when not in debug mode.
- Updated all dependencies in requirements.txt, and switch to pinning deps with == rather than >= so dependabot on github can do its thing

8.11.3 Fixed

- Fix wrong requirements line for pre-commit (remove extra equal sign)

8.12 v0.14.2 (13-Sep-2020)

8.12.1 Added

- Make `show certificate` output certificate `not_valid_before` and `not_valid_after`

8.12.2 Changed

- Rename test `test_show_certificate()` to `test_show_certificate_file_not_found()`

8.13 v0.14.1 (13-Sep-2020)

8.13.1 Added

- Workaround to get certificate from chain in installations from before foo-certonly.crt was written separately. This makes the "get ocsp" subcommand work even if the current certificate was issued with an older version of certgrinder.

8.13.2 Changed

- Rename parse_certgrinderd_certificate_output() to parse_certificate_chain() and clean it up a bit
- Update some log messages and update tests to match
- Change "intermediate" to "issuer" in the code and tests.
- Rename intermediate cert path to example.com-issuer.crt instead of example.com-intermediate.crt. Existing intermediate/issuer certs will be renamed next time "get ocsp" is run, which is done automatically by the "periodic" command.

8.14 v0.14.0 (29-Aug-2020)

8.14.1 Changed

- Update log message when running post-renew hooks

8.15 v0.14.0-beta2 (29-Aug-2020)

8.15.1 Added

- Workaround to get intermediate from chain in installations from before foo-intermediate.crt was written separately. This makes the "get ocsp" subcommand work even if the current certificate was issued with an older version of certgrinder.

8.15.2 Changed

- Separated the PEM chain splitting logic into a new split_pem_chain method

8.16 v0.14.0-beta1 (29-Aug-2020)

8.16.1 Added

- OCSP response support
- Log certgrinderd output at the level certgrinderd logs it at, when possible (otherwise log at WARNING)
- Tests for the new functionality

8.16.2 Changed

- Support the new certgrinderd commands and subcommands
- Change short command for –config-file from -f to -c
- Set default certgrinder command to "certgrinderd"
- Use with for opening files a few places to avoid leaving open fds

8.16.3 Fixed

- Changed certgrinder syslog ident from "certgrinderd" to "certgrinder"

8.17 v0.13.2 (11-Jul-2020)

8.17.1 Added

- Manpage to MANIFEST.in to include it in the distribution

8.18 v0.13.1 (7-Jul-2020)

8.18.1 Changed

- Specify python3.7 and 3.8 as classifiers in setup.py

8.19 v0.13.0 (7-Jul-2020)

8.19.1 Changed

- Test suite now covers 100% of certgrinder.py

8.19.2 Fixed

- Fix broken test client/certgrinder/tests/test_certgrinder.py::test_check_certificate_not_cert
- Fix broken show_certificate() method, and make it output more useful info

8.20 v0.13.0-rc1 (1-Jul-2020)

8.20.1 Changed

- Writing the certificate only (without the intermediate) to `example.com-certonly.crt` is new in 0.13, so make the `check_certificate()` method checks the chain certificate to make sure upgrading 0.12 to 0.13 doesn't trigger needlessly renewing all existing certs.

8.21 v0.13.0-beta2 (29-Jun-2020)

8.21.1 Added

- Dev requirements now has `sphinx-rtd-theme` which is the theme used on ReadTheDocs, so `make html` in `docs/` now produces the same-ish output.
- Dev requirements now include `sphinx-argparse` used for generating automatic usage documentation.
- Very preliminary support for EC keys in addition to RSA keys.
- More tests
- Better validation of returned certificate and intermediate
- Save intermediate in separate file, save certificate only in separate file.
- Documentation for all config settings
- Manpage certgrinder.8
- periodic command to run from cron

8.21.2 Changed

- Move CHANGELOG.md to rst format and into `docs/`
- Rework command-line options, add commands, rework configuration and configfile. This is a backwards incompatible change. Run `/venv/bin/certgrinder periodic` from cron, `certgrinder help` for more info.
- Configuration is now a combination of command-line options (if any), config file (if any) and default config; in decreasing precedence order. A default setting will be overridden by a config file setting which will be overridden by a command-line setting.
- Update `certgrinder.conf.dist` with new options and better comments
- Mark most methods as `@staticmethod` or `@classmethod`, refactor code as needed. This makes the code more reusable and easier to test.
- Split certificate validity tests into separate methods
- Split parsing of `certgrinderd` output into separate method `parse_certgrinderd_output()`
- Split argparse stuff (which grew considerably with this change) into separate `get_parser()` func
- Support calling `certgrinder.main()` function and `certgrinder.Certgrinder.grind()` method with a list of mocked command-line args
- Update existing tests to deal with all the new stuff
- Make pytest logformat look like regular logging
- Split creating the argparse object into a separate function to assist sphinx-argparse
- Reorder argparse commands and subcommands in alphabetical order
- Re-add -v / –version to show version and exit
- Test suite now covers 100% of certgrinder.py

8.22 v0.13.0-beta1 (7-May-2020)

8.22.1 Fixed

- Made -q / –quiet mode work
- Made certgrinder always pass `--log-level LEVEL` to certgrinderd, so the effects of both `--quiet` and `--debug` are passed to the certgrinderd call.

8.23 v0.13.0-alpha8 (6-May-2020)

8.23.1 Changed

- Changed logformat to prefix messages with certgrinder: and Certgrinder. instead of nothing and %(name)s, making it more clear which messages are from certgrinder and which are from certgrinderd
- Output logging from certgrinderd call

8.24 v0.13.0-alpha7 (6-May-2020)

8.24.1 Fixed

- Old bug where permissions of private key would be fixed to 640 even if it was already 640
- –log-level didn't work without –debug

8.25 v0.13.0-alpha6 (6-May-2020)

- No changes

8.26 v0.13.0-alpha5 (6-May-2020)

8.26.1 Added

- MANIFEST.in file to include certgrinder.conf.dist in installs

8.26.2 Changed

- Default config file is now ~/certgrinder.conf instead of ~/certgrinder.yml

8.27 v0.13.0-alpha4 (5-May-2020)

8.27.1 Added

- There is now a –log-level=LEVEL command line argument to set loglevel more flexibly. It can be set to one of DEBUG, INFO, WARNING, ERROR, or CRITICAL.

8.27.2 Changed

- Config file path should be given with the -f flag
- Pass –staging and –debug flag to certgrinderd when given to certgrinder
- Prefix syslog messages with "certgrinder" instead of "Certgrinder" to match the package name

8.28 v0.13.0-alpha3 (5-May-2020)

- No changes

8.29 v0.13.0-alpha2 (4-May-2020)

8.29.1 Added

- Install `certgrinder` binary using entry_points in setup.py

8.29.2 Changed

- Wrap script initialisation in a main() function to support entry_points in setup.py better

8.30 v0.13.0-alpha (4-May-2020)

8.30.1 Added

- Create Python package `certgrinder` for the Certgrinder client, publish on pypi
- Add isort to pre-commit so imports are kept neat
- Tox and pytest and basic testsuite using Pebble as a mock ACME server
- Travis and codecov.io integration
- Add -C argument which simply checks if the certificates are present and valid and have more than 30 days validity left. Exit code 0 if all is well or exit code 1 if one or more certificates needs attention.

8.30.2 Changed

- Move client files into client/ and server files into server/, each with their own CHANGELOG.md, in preparation for Python packaging.
- Reorder commandline arguments alphabetically.
- Change a few imports to make mypy and isort happy

8.31 v0.12.1 (4-Jan-2020)

8.31.1 Added

- Add RELEASE.md so I don't forget how to do this

8.31.2 Fixed

- Fixed release date for v0.12.0 in CHANGELOG.md
- Add a few type: ignore for some of the cryptography imports and calls to make newer mypy happy

8.31.3 Changed

- Update mypy to 0.761 and add to requirements-dev.txt

8.32 v0.12.0 (4-Jan-2020)

8.32.1 Changed

- Support python3 instead of (NOT in addition to) python2
- Format code with Black
- Check code with flake8
- Add type annotations and check code with mypy –strict

8.32.2 Fixed

- pyyaml load deprecation warning: ./certgrinder.py:54: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the default Loader is unsafe. Please read https://msg.pyyaml.org/load for full details.

8.33 v0.11.0 (25-Dec-2018)

8.33.1 Added:

- Support for setting SSH user: in certgrinder.yml config file.

8.33.2 Changed:

- Remove OpenSSL dependency for key and X509 operations, use cryptography directly instead. This affects any method which deals with keys and/or X509.
- Do not use shell=True for the subprocess.pOpen SSH call.

8.33.3 Removed:

- Support for selfsigned certificates.

8.34 v0.10.2 (5-Apr-2018)

8.34.1 Added:

- Support setting syslog_facility and syslog_socket in certgrinder.yml (defaults to "user" and "/var/run/log" to maintain backwards compat)
- Warn in the last line when one or more selfsigned certificates has been created
- Show a counter with the number of domainsets being processed

8.34.2 Fixed:

- Typo in variable name in logoutput
- Only log SSH output and exception info when in debug mode
- Various improvements to logging

8.35 v0.10.1 (2-Mar-2018)

8.35.1 Fixed:

- Version number was wrong in certgrinder.py

8.36 v0.10.0 (2-Mar-2018)

8.36.1 Added:

- Move from webroot to manual Certbot authenticator, using hook scripts manual-auth-hook and manual-cleanup hook
- Add DNS-01 support in hook scripts. DNS-01 is now the recommended challenge type.
- csrgrinder got a config file
- Describe new features in README
- Many improvements to logging and error handling

8.36.2 Fixed:

- Language and typos and layout in README

8.37 v0.9.5 (16-Feb-2018)

8.37.1 Fixed:

- v0.9.4 had the wrong version number in the .py file.

8.37.2 Added:

- -p / –showspki switch to output pin-sha256 pins for the public keys. Useful for HPKP or other pinning that uses the same format.

8.38 v0.9.4 (17-Jan-2018)

8.38.1 Fixed:

- The showtlsa (-s) and checktlsa (-c) features did not work for multiple domain sets

8.39 v0.9.3 (17-Jan-2018)

8.39.1 Fixed:

- Custom nameserver functionality was not working due to an error
- Catch more types of exceptions when looking up DNS results, and exit if a serious error occurs.

8.40 v0.9.2 (17-Jan-2018)

8.40.1 Fixed:

- Typo in CHANGELOG.md

8.41 v0.9.1 (17-Jan-2018)

8.41.1 Fixed:

- Logic for using a custom nameserver with -n / –nameserver was inverted.
- Add example directory structure to README.md

8.41.2 Added:

- Show version number in usage and add -v / –version switch to show it.
- Add shebang line to certgrinder.py and make the script executable.

8.42 v0.9.0 (16-Jan-2018)

8.42.1 Added:

- This changelog. First numbered release.

INDEX

Symbols

C

G

L

O