EVERTRUST

Horizon Client

Version 1.19, 2026-10-29

Table of Contents

1. Introduction . . . 1
2. Requirements . . . 3
3. General Configuration and Usage . . . 4
4. Basic commands . . . 13
5. Discovery Operations . . . 14
6. Import operations . . . 20
7. EST Certificate Lifecycle Operations . . . 28
8. SCEP Certificate Lifecycle Operations . . . 40
9. WebRA Certificate Lifecycle Operations . . . 49
10. Updating a certificate . . . 61
11. Bulk Operations . . . 64
12. Automatic TLS Certificate Installation . . . 66
13. Release notes . . . 102
    13.1. Horizon-cli 1.19.0 release notes . . . 102

Chapter 1. Introduction

Description

Horizon Client is the client software associated to EverTrust Horizon. This client is developed in Golang, and compiled for the following platforms:

- Linux for x86-64 and arm64 processors
- Windows for x86-64 processors
- Darwin for x86-64 and arm64 processors
- AIX for ppc64 processors

System requirements

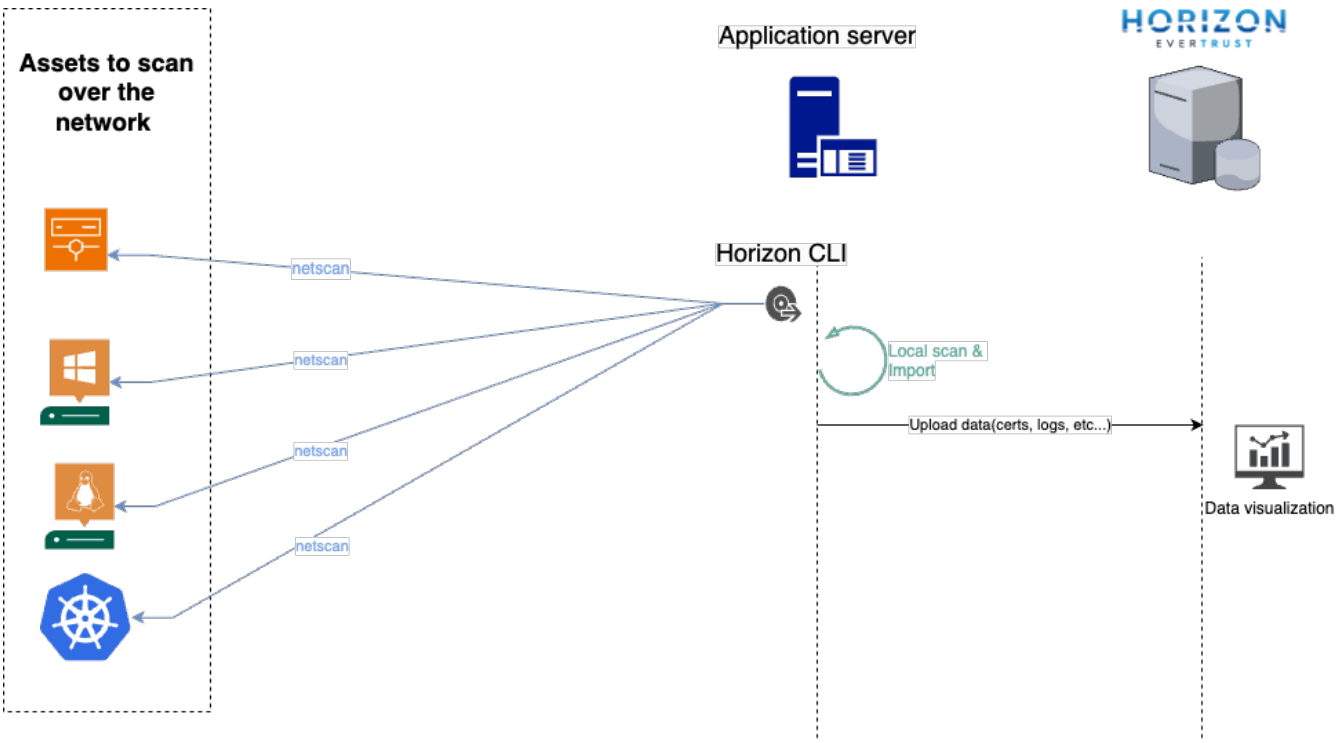
For system and operating system requirements, please refer to the Requirements page.

This document is specific to Horizon Client version **1.19**, which may be used with EverTrust Horizon 2.4.0 or later.

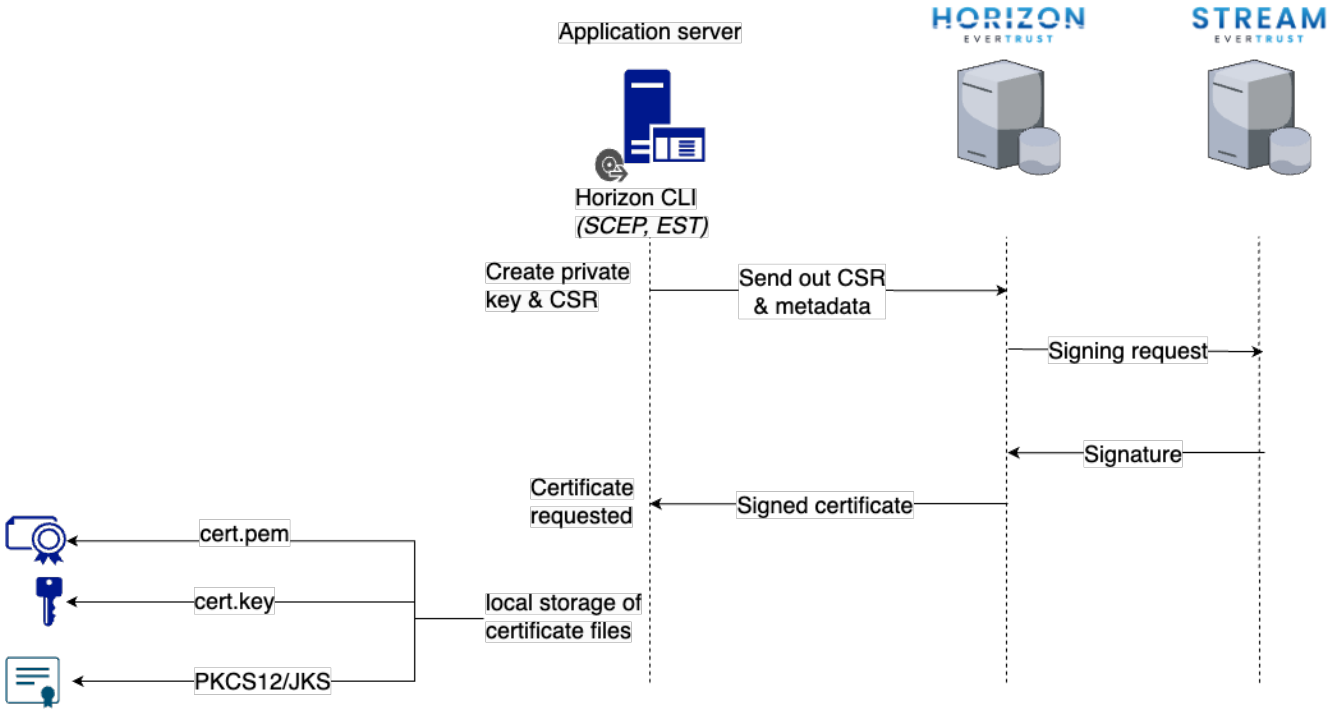
Scope

This document is a guide describing how to use the Horizon Client to perform the following tasks:

- Certificate discovery & import



- Certificate lifecycle management


Application server
HORIZON
EVERTRUST
STREAM
EVERTRUST
Horizon CLI
(SCEP, EST)
Create private key & CSR
Send out CSR & metadata
Signing request
Signature
Certificate requested
Signed certificate
cert.pem
cert.key
PKCS12/JKS
local storage of certificate files

Chapter 2. Requirements

Requirements

System requirements

To run Horizon Client the underlying system must comply with the following minimum requirements :

For certificate lifecycle purposes

- 1 gigahertz (GHz) or faster with 1 or more cores
- 1 gigabyte (GB) of RAM for Linux environments
- 2 gigabyte (GB) of RAM for Microsoft environments
- 10 gigabyte (GB) or larger storage device

For discovery purposes

- 2 gigahertz (GHz) or faster with 2 or more cores
- 2 gigabyte (GB) of RAM for Linux environments
- 4 gigabyte (GB) of RAM for Microsoft environments
- 20 gigabyte (GB) or larger storage device

Operating system requirements

The following elements are considered as operating system requirements:

- AIX (64-bit);
- Linux (64-bit);
- Microsoft Windows 10 (64-bit);
- Microsoft Windows Server 2016 (64-bit);
- Microsoft Windows Server 2019 (64-bit);
- Microsoft Windows Server 2022 (64-bit);
- Microsoft Windows Server 2025 (64-bit);

Chapter 3. General Configuration and Usage

Installation

Package install/uninstall

Installing the package

RPM

Installing from the Evertrust repository

Create a `/etc/yum.repos.d/horizon-cli.repo` file containing the EverTrust repository info:

```
[horizon-cli]
enabled=1
name=Horizon Client Repository
baseurl=https://repo.evertrust.io/repository/horizon-cli-rpm/
gpgcheck=1
gpgkey=https://evertrust.io/.well-known/rpm/gpg.pub
username=<username>
password=<password>
```

Replace `<username>` and `<password>` with the credentials you were provided.

Make sure the Evertrust GPG key is trusted:

```
# rpm --import https://evertrust.io/.well-known/rpm/gpg.pub
```

You can then run the following to install the latest Horizon Client version:

```
# yum install horizon-cli
```

To prevent unattended upgrades when running yum update, you should pin the Horizon Client version by adding

```
exclude=horizon-cli
```

at the end of the `/etc/yum.repos.d/horizon-cli.repo` file after installing Horizon Client.

Installing from the package file

Download the latest RPM for Horizon Client on the Official EVERTRUST repository.

Upload the file `horizon-cli-<latest>.x86_64.rpm` to the server;

Access the server with an account with administrative privileges;

Install the Horizon Client package with the following command:

```
# yum localinstall /root/horizon-cli-<latest>.x86_64.rpm
```

If you wish to verify the signature of the RPM package, the EVERTRUST key can be added to your trusted keys using the following command:

```
# rpm --import https://evertrust.io/.well-known/rpm/gpg.pub
```

The signature can then be verified using the following command:

```
# rpm -K /root/horizon-cli-<latest>.x86_64.rpm
```

DEB

The Debian package is available since version **1.18.0**, for the `amd64` architecture only. On other architectures, install the client from the binary distribution.

Installing from the Evertrust repository

If you haven't already, to add the EVERTRUST repository to your APT repositories, run the following commands:

1. Install the required tools (`gpg`)

   ```
   # sudo apt install gnupg
   ```

2. Download and install the EVERTRUST GPG key

   ```
   # curl https://evertrust.io/.well-known/apt/gpg.pub | sudo gpg -o
   /usr/share/keyrings/evertrust.gpg --dearmor
   ```

3. Add the repository

   ```
   # echo "deb [ arch=all,amd64 signed-by=/usr/share/keyrings/evertrust.gpg ]
   https://repo.evertrust.io/repository/apt all main" | sudo tee
   /etc/apt/sources.list.d/evertrust.list
   ```

Once the repository has been added, authentication to it must be provided. To do so, edit

the `/etc/apt/auth.conf` file and add the following lines:

```
machine repo.evertrust.io
login <your EVERTRUST login>
password <your EVERTRUST password>
```

Once the repository has been added, run the following command to update the APT repository list.

```
# sudo apt update
```

You can then run the following command to install the latest Horizon Client version:

```
# sudo apt install horizon-cli
```

To prevent unattended upgrades when running `apt upgrade`, you should pin the Horizon Client version by creating a `/etc/apt/preferences.d/horizon-cli` file:

```
Package: horizon-cli
Pin: version <installed-version>
Pin-Priority: 1001
```

Installing from the package file

Download the latest DEB for Horizon Client on the Official EVERTRUST repository.

Upload the file `horizon-cli_<latest>_amd64.deb` to the server;

Access the server with an account with administrative privileges;

Install the Horizon Client package with the following command:

```
# apt install /root/horizon-cli_<latest>_amd64.deb
```

After installing

The package installs the client in `/opt/horizon/bin/horizon-cli`, which is not added to the `PATH`, and creates the working directories used by the client. It does not create the configuration file: run the command below to generate it.

```
# /opt/horizon/bin/horizon-cli install
```

Windows

To install the package, double click on the MSI file and follow the instructions.

Binary

The linux binary file is usable on any linux distribution, to install it follow the steps below :

- Add the binary file to the `PATH`, in order to easily launch it on your shell.
- Apply the executable permission on the binary file

```
# chmod +x horizon-cli.bin
```

Uninstalling the package

RPM

```
# yum remove horizon-cli
```

DEB

```
# apt remove horizon-cli
```

Windows

To uninstall the package, simply browse to the Applications & program menu and uninstall the program.

Binary

Remove the binary file from your system, and remove it from `PATH`

Command line installation & initialization

Use the command below to install the client and generate interactively your configuration file:

```
$ horizon-cli install
```

The configuration file can also be created using command line parameters:

```
$ horizon-cli install --endpoint https://horizon-test.com
```

Use the help to get the full list of available parameters.

If you did not use an installer, this command should always be run first to ensure everything is set up correctly.

Configuration Location

General parameters of Horizon Client are configured through a file placed in one of the following locations:

Global configuration :

- `/opt/horizon/etc/horizon-cli.conf`
- `[C|D]:\ProgramData\EverTrust\Horizon\horizon-cli.conf`

Per-user configuration :

- `~/.horizon-cli/etc/horizon-cli.conf`
- `[C|D]:\Users\<username>\AppData\Local\horizon-cli\horizon-cli.conf`

In case the user running the Horizon Client is an administrator and the global configuration file is present and accessible by the user, the global configuration file will be used. Otherwise, the per-user configuration file will be used.

If the per-user configuration file is not present and the global configuration file is not accessible, the client will throw an error.

Configuration Content

Since version **1.10**, the configuration was migrated from JSON to YAML, if you are upgrading from an earlier version, the configuration migration will be done automatically and should be seamless.

The configuration file is in YAML format and contains the following:

```yaml
api_id: API-ID
api_key: API-Key
service_account_name: Service account name
service_account_token: Service account token
endpoint: endpoint url. e.g. https://horizon-test.evertrust.fr
debug: false
timeout: 2
proxy: proxy. e.g. http://myproxy.corp.local:3128
root_ca: Root CA PEM Certificate(s).
log_file: The log file of Horizon.
external_proxy: proxy. e.g. http://myproxy.corp.local:3128
sudo_commands:
  - command_one
  - another_command
```

These parameters may be instead specified or overridden using environment variables, as detailed in the table below.

Table 1. General configuration parameters

Parameter	Environment variable	Description
`api_id`	`HRZ_APIID`	The API ID: the identifier of a local account user defined in Horizon. Used for discovery, import modes and for the revocation in the EST module
`api_key`	`HRZ_APIKEY`	The API Key. Used together with API ID
`service_account_name`	`HRZ_SERVICE_ACCOUNT_NAME`	The name of the service account used to authenticate against Horizon. Used together with the service account token. Mutually exclusive with `api_id` /`api_key`.
`service_account_token`	`HRZ_SERVICE_ACCOUNT_TOKEN`	The token of the service account used to authenticate against Horizon. Used together with the service account name.
`endpoint`	`HRZ_ENDPOINT`	The URL of the Horizon instance, starting with `http` or `https` and without trailing "/"
`debug`	`HRZ_DEBUG`	Set to true to enable debug mode of the Horizon Client, defaults to false if unspecified.
`timeout`		Connection timeout in seconds, defaults to 2 seconds if unspecified.
`proxy`	`HRZ_HTTPS_PROXY`	HTTPS proxy used to reach Horizon (if any), in URL form which can contain login and password if needed.
`root_ca`		PEM chain of CA certificates that issued the TLS certificate exposed by Horizon. This parameter is optional, as preferred way is to put these CA certificates in the machine trust store.

Parameter	Environment variable	Description
log_file	HRZ_LOGFILE	Log file of horizon. This parameter is optional, but a default value is set as the Horizon Client displays useful messages on STDOUT and logs should always be kept.
external_proxy	HRZ_EXTERNAL_PROXY	HTTPS proxy used to reach Third Parties (if any), in URL form which can contain login and password if needed.
sudo_commands	HRZ_SUDO_COMMANDS	Array of commands that should be executed using sudo.

Configuration customization

Changing the configuration file location

In case you want to change the configuration file location, the HRZ_CONFIG environment variable can contain an absolute path to the configuration file and will try to read it before defaulting to the standard configuration as detailed above.

Changing all horizon-client files location

Additional files are used by the client (automation state, log files, etc). In case you want to change the path to these configurations, the HRZ_LOCAL_DATA environment variable can contain an absolute path to a folder, and will create all necessary files starting from this folder.

In order to keep backward compatibility, legacy environment variables are still available and are the same as the one above without the HRZ_ prefix. These should not be used and should be migrated to HRZ-prefixed one.

You can use the “--help” parameter to get command line help on any command or sub-command.

```
$ horizon-cli <command> <subcommand> --help
```

Authentication

The Horizon Client authenticates against the Horizon instance using either a **service account** or a **login/password** local account. These two methods are mutually exclusive: only one may be configured at a time. If both are defined, the client will return an error.

Service Account

A service account is identified by a name and a token, both defined in Horizon. When configured, the client sends these credentials to Horizon on every request.

```
service_account_name: my-service-account
service_account_token: my-service-account-token
```

The name and token can be set with `horizon-cli install` (as the `--service-account-name` and `--service-account-token` flags or through the interactive prompts), or through the `HRZ_SERVICE_ACCOUNT_NAME` and `HRZ_SERVICE_ACCOUNT_TOKEN` environment variables, which take precedence over the configuration file.

The name and token do not have to be provided at the same time: the client accepts a partial configuration (for example a name now and a token delivered later through an environment variable or a token file). Both must, however, be resolvable when a request is actually made, otherwise the request will fail.

Token formats

The token can be provided in two forms:

- An **inline** value, sent to Horizon as-is.
- A `file://` **reference** to a token file (for example `file:///etc/horizon/token`). The path must be absolute. The file is read, its contents trimmed, and re-read **before each request**, so a token rotated by an external secret manager is always picked up without re-running `install`. If the file is missing, unreadable, or empty, the request will fail.

Service account authentication is **not compatible** with the SCEP and EST authorized enrollment modes, due to protocol limitations. To use SCEP or EST authorized, you must authenticate using a login/password local account (`api_id` /`api_key`).

Login/Password

A login/password local account is identified by an API ID and an API Key, both defined in Horizon.

```
api_id: my-api-id
api_key: my-api-key
```

Both `api_id` and `api_key` must be provided together. This authentication method is required for the SCEP and EST authorized enrollment modes.

No authentication

The Horizon Client can be used without authentication for certificate enrollments.

This is possible using EST/SCEP challenge or Webra Auto-Validation workflows.

Chapter 4. Basic commands

Ping

The Ping command can be useful to ensure the client configuration to access Horizon is correct. It will exit in error if the client could not join the Horizon server.

```
horizon-cli ping
```

The `--permissions` flag will display the permissions associated with the account the client is using, or if it is not, it will log a message indicating it.

```
horizon-cli ping --permissions
```

Chapter 5. Discovery Operations

These operations aim at feeding Horizon with certificates discovered on the network through different means. These certificates will be fed along with appropriate Discovery metadata, such as IP address or Hostname of the machine on which the discovered certificate is held.

Local Scan

In local scan mode, the Horizon Client will scan the machine it is installed on for certificates, and reports them to Horizon. Certificates are discovered if they match following conditions:

- They are saved in PEM or DER format in a file that is pointed in a configuration file
- They are contained in a Machine or User "MY" certificate store (Windows Only)
- They are not CA certificates

In local scan mode, Horizon client should be launched with root or administrator rights, or it will probably fail to discover all certificates.

```
horizon-cli localscan --campaign=test
```

When detecting a path to a certificate file containing an environment variable, a warning event with code `HCL-LOCALSCAN-ENV-001` will be raised

Unreferenced certificates

By default, a certificate file is only reported if a configuration file found during the scan points to it. To report every certificate file found in the scanned paths, whether a configuration file references it or not, use the `--all-certs` option.

```
horizon-cli localscan --campaign=test --all-certs
```

CA certificates are not reported, even with the `--all-certs` option.

Keystores

To handle keystores, the `--containers-passwords` option allows to specify keystore passwords to try on encountered keystore.

When a keystore cannot be opened, a warning event with code `HCL-LOCALSCAN-KS-001` will be raised

Scheduling

In order to perform local scans on a recurring schedule, the Horizon Client offers the possibility to create periodic tasks to run a scan.

The three supported options for the `period` parameter are:

- daily - runs the task everyday between 0-4 AM UTC
- weekly - runs the task every Sunday between 0-4 AM UTC
- monthly - runs the task on the first day of the month between 0-4 AM UTC

```
horizon-cli localscan --campaign=test --create-periodic-task --period
=monthly
```

This periodic task can be run with a specific user identity on Linux using the `user` parameter.

```
horizon-cli localscan --campaign=test --create-periodic-task --period
=monthly --user=horizon-cli
```

The created task can then be removed using:

```
horizon-cli localscan --campaign=test --remove-periodic-task
```

Scanned folders

By default, `localscan` will scan these commonly used paths:

- On Unix: `/usr/local/etc`, `/etc` and `/opt`
- On Windows: `user store`, `machine store`, `program data`, `program files`, `program files x86`

To disable scanning these paths, use the `--exclude-default-paths` option.

Network Scan

In network scan mode, the Horizon Client will first connect to Horizon to get the campaign's scanning parameters (Hosts and Ports), then perform the network scanning and feed Horizon with the scan results.

The following algorithm is used for network scanning:

1. If `--ping-first` flag is given, perform ICMP ping on the defined hosts and discard hosts that are not reachable
2. Scan the hosts and ports for an open TCP port

3. If TCP port is opened:
    - If port is not '25', try a TLS handshake. If handshake succeeds, retrieve the certificate and report it to Horizon
    - If port is '25', perform SMTP STARTTLS, retrieve the certificate and report it to Horizon

The "timeout" global configuration variable has an impact on both open ports discovery and TLS handshake. In case you get unexpected handshake errors or EOF, try to increase the timeout. However, this will also make the network scan perform slower.

```
horizon-cli netscan --campaign=test
```

In order to perform network scans on a recurring schedule, the Horizon Client offers the possibility to create periodic tasks to run a scan.

The three supported options for the `period` parameter are:

- daily - runs the task everyday between 0-4 AM UTC
- weekly - runs the task every Sunday between 0-4 AM UTC
- monthly - runs the task on the first day of the month between 0-4 AM UTC

```
horizon-cli netscan --campaign=test --create-periodic-task --period
=monthly
```

This periodic task can be run with a specific user identity on Linux using the `user` parameter.

```
horizon-cli netscan --campaign=test --create-periodic-task --period
=monthly --user=horizon-cli
```

The created task can then be removed using:

```
horizon-cli netscan --campaign=test --remove-periodic-task
```

If you wish to also scan the TLS versions and supported cipher suites of the scanned ports, you can use the `--scan-supported-tls` option.

Due to technical limitations in the GO runtime, some cipher suites cannot be tested.

The recommended approach is to import an nmap scan using nmap import.

▼ *Supported cipher suites*

- TLS_AES_128_GCM_SHA256

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
- TLS_ECDHE_RSA_WITH_RC4_128_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

Source : https://go.dev/src/crypto/tls/cipher_suites.go

```
horizon-cli netscan --campaign=test --scan-supported-tls
```

nmap import

In nmap import mode, the discovery itself is performed by nmap, using the `ssl-cert` plugin. Horizon Client then has the ability to import the nmap scanning results into Horizon using the nmap import mode.

To be able to do so, nmap needs to be launched with the `-oX` option, in order to export its scan result as XML file. This XML file is then passed on to Horizon Client.

If the `ssl-enum-ciphers` was also used on the scan, the TLS ciphers for the certificates will also be sent to Horizon.

```
horizon-cli importscan nmap --campaign=test --xmlfile=nmapresults.xml
```

Qualys Certificate View import

In Qualys Certificate View (CV) import mode, the discovery itself is performed by Qualys CV. Horizon Client then has the ability to import the Qualys CV scanning results into Horizon using the `qualyscv` import mode.

To be able to do so, a technical account must have been created into Qualys CV for Horizon Client, with appropriate rights to be able to view the scanning results. You need also to identify your Qualys CV API Gateway URL using the following link.

```
horizon-cli importscan qualyscv --campaign=test --endpoint
=https://gateway.qg1.apps.qualys.eu --username=testlogin --password
=testpassword
```

Nessus Scan Import

In Nessus scan import mode, Horizon Client enables the importation of scanning results from Nessus into Horizon. This mode allows for a seamless integration of Nessus vulnerability scans into the Horizon environment.

To utilize this feature, you need to ensure that you have valid credentials for Nessus with the necessary permissions to access and export scan data and the scan id on which you want to perform the import. Additionally, you must know your Nessus URL through which Horizon Client will communicate with the Nessus API and use the "SSL Certificate Information" plugin output to get the certificates into Horizon.

```
horizon-cli importscan nessus --campaign=test --endpoint
=https://cloud.tenable.com --username=testlogin --password=testpassword
--scan-id=5
```

Tenable.sc Import

In Tenable.sc scan import mode, Horizon Client enables the importation of scanning results from Tenable.sc into Horizon.

To utilize this feature, you need to ensure that you have a valid API access key and secret key for Tenable.sc with the necessary permissions to access and export scan data and the scan name on which you want to perform the import. Additionally, you must know your Tenable.sc hostname

through which Horizon Client will communicate with the Tenable.sc API and use the "SSL Certificate Information" plugin output to get the certificates into Horizon.

```
horizon-cli importscan tenable-sc --campaign=<campaign> --scan-name
=<name of the scan> --hostname=<tenable.sc instance> --access-key
=<access key> --secret-key=<secret key>
```

To import the results of every scan at once, use `--cumulative` instead of `--scan-name`. The certificates are then read from the Tenable.sc cumulative database, which aggregates the results of all the scans. One of `--scan-name` or `--cumulative` must be given, and they cannot be used together.

```
horizon-cli importscan tenable-sc --campaign=<campaign> --cumulative
--hostname=<tenable.sc instance> --access-key=<access key> --secret-
key=<secret key>
```

Chapter 6. Import operations

Import operations are designed to import certificate into Horizon without any metadata. This is useful mainly when installing Horizon, e.g. to import all certificates from an existing PKI database.

Local Import

In order to be able to import certificates, you need to put them as PEM files in a folder, and launch Horizon Client by pointing at that folder. Horizon Client will recurse on the folder, find all PEM files, and import certificates into Horizon. It is advised to use sub-folders to store certificates, so that you avoid to hit any file-per-folder file system limit.

```
horizon-cli localimport --campaign=test --path=/path/to/certificates
--source=MyADCS
```

By default, this command does not import CA certificates. To import CA certificates, use the --enable -ca-import flag.

```
horizon-cli localimport --campaign=test --path=/path/to/certificates
--source=MyADCS --enable-ca-import
```

If you wish to import certificates along with their private keys (e.g. when importing from a PKI escrow), you need to put them as PKCS#12 files in a folder, and launch Horizon Client by pointing at that folder. Horizon Client will recurse on the folder, find all PEM files, and import certificates into Horizon. It is advised to use sub-folders to store certificates, so that you avoid to hit any file-per-folder file system limit. All the PKCS#12 files must be encrypted using the same password that will be passed to Horizon Client using the command line.

```
horizon-cli localimport --campaign=test --path=/path/to/certificates
--source=MyADCS --pfx-password=<pkcs12_password>
```

You can also import certificates from a csv file. Certificates must be in a column named "certificate". As of now, three formats are supported:

1. DERBase64: Certificate in DER (binary) Base 64 encoded (default);
2. DERHex: Certificate in DER (binary) Hex String encoded;
3. PEM: Certificate in PEM (with or without the certificate header and footer).

```
horizon-cli localimport --campaign=test --csv /path/to/csv/file.csv
--csv-separator ";"
```

In order to add technical metadata to the imported certificate, the `--csv-metadata` flag can be used to import metadata from a column with the same name. For example, to configure a `pki_connector` on each certificate with a file containing the `pki_connector` column:

```
horizon-cli localimport --campaign=test --csv /path/to/csv/file.csv
--csv-separator ";" --csv-metadata pki_connector
```

Supported metadata are:

- `pki_connector`
- `certeurope_id`
- `digicert_id`
- `digicert_order_id`
- `entrust_id`
- `fcms_id`
- `gsatlas_id`
- `metapki_id`

Network Import

DigiCert CertCentral

You can import all your valid certificates from DigiCert CertCentral. Please note that only certificates in "issued" state can be imported. Certificates that are revoked will not be imported.

```
horizon-cli netimport digicert --campaign=test --digicert-api-key=<api-
key>
```

The API key can also be provided through the `HRZ_NETIMPORT_DIGICERT_API_KEY` environment variable.

By default, requests target the global CertCentral endpoint (`https://www.digicert.com`). The `--api-url` flag can be used to target another CertCentral instance, for example the EU instance (`https://certcentral.digicert.eu`) when your API key is restricted to it.

```
horizon-cli netimport digicert --campaign=test --digicert-api-key=<api-
key> --api-url=https://certcentral.digicert.eu
```

GlobalSign MSSL

You can import all your certificates from GlobalSign Managed SSL (MSSL). Certificates are retrieved through the GlobalSign GAS SOAP API, so a username and password for that API are required.

By default, only orders in the `4` (issued) order status are imported. This can be changed using the `--order-status` flag.

```
horizon-cli netimport globalsign --campaign=test --username=<GlobalSign
API username> --password=<GlobalSign API password>
```

The credentials can also be provided through the `HRZ_NETIMPORT_GLOBALSIGN_USERNAME` and `HRZ_NETIMPORT_GLOBALSIGN_PASSWORD` environment variables.

The following options are available:

- `--order-status` filters the GlobalSign orders to import by their order status (default `4`, i.e. issued)
- `--api-url` sets the GlobalSign GAS API endpoint to use, allowing you to target the test environment or a custom endpoint (default `https://system.globalsign.com/kb/ws/v1/GASService`)
- `--test` (**deprecated**) uses the GlobalSign test endpoint (`https://test-gcc.globalsign.com/kb/ws/v1/GASService`) instead of the production one. This flag is deprecated in favor of `--api-url` and will be removed in a future release. It cannot be used together with `--api-url`.

Sectigo

You can import all your SSL certificates from a Sectigo Certificate Manager (SCM) account. Only currently-issued SSL certificates are imported; revoked and expired certificates, and non-SSL certificate types (client/S-MIME, code signing, device), are not imported.

Authentication uses your SCM API credentials: a login, a password and the customer URI of your SCM account.

```
horizon-cli netimport sectigo --campaign=test --login=<SCM login>
--password=<SCM password> --customer-uri=<SCM customer URI>
```

The password can also be provided through the `HRZ_NETIMPORT_SECTIGO_PASSWORD` environment variable.

AWS ACM

You can import all your valid certificates from AWS ACM. Please note that only certificates in "issued" state can be imported. Certificates that are revoked will not be imported.

```
horizon-cli netimport aws-acm --campaign=test --aws-region=<aws-region>
```

```
--access-key-id=<aws-access-key-id> --secret-access-key=<aws-secret-access-key>
```

AWS Role Assumption is supported. You need to provide the ARN of the role you wish to assume using the `--assume-role-arn` option.

It is also possible to attach the AWS ACM third-party connector metadata to the discovered certificates, so that when the certificate is later managed in Horizon, renewals update the existing ACM object instead of creating a new one.

In order to activate this behavior, the `--connector` flag must reference a valid AWS ACM Connector in Horizon.

```
horizon-cli netimport aws-acm --campaign=test --connector=<Horizon AWS ACM Connector name> --aws-region=<aws-region> --access-key-id=<aws-access-key-id> --secret-access-key=<aws-secret-access-key>
```

If the certificate is already managed on a WebRA profile, the "Synchronize data from discovery" option must be enabled in the profile configuration, and the profile must reference the connector given as argument.

Azure Key Vault

You can import all your valid certificates from Azure Key Vault. Please note that only certificates in "issued" state can be imported. Certificates that are in pending state will not be imported.

```
horizon-cli netimport akv --campaign=test --vault-name=<vault FQDN> --azure-tenant=<tenant name> --client-id=<client app Id> --client-secret=<client app secret>
```

It is also possible to attach the AKV third-party connector metadata (connector reference and vault object name) to the discovered certificates, so that when the certificate is later managed in Horizon, renewals update the existing AKV object instead of creating a new one.

In order to activate this behavior, the `--connector` flag must reference a valid AKV Connector in Horizon.

```
horizon-cli netimport akv --campaign=test --connector=<Horizon AKV Connector name> --vault-name=<vault FQDN> --azure-tenant=<tenant name> --client-id=<client app Id> --client-secret=<client app secret>
```

If the certificate is already managed on a WebRA profile, the "Synchronize data from discovery" option must be enabled in the profile configuration, and the

profile must reference the connector given as argument.

Google Certificate Manager

You can import the certificates of a Google Cloud project from Google Certificate Manager.

Authentication uses a GCP service account. Pass the path to its JSON key file with `--credentials`, otherwise the Application Default Credentials are used (for example, the `GOOGLE_APPLICATION_CREDENTIALS` environment variable).

The project is read from `--project`, then from the `GOOGLE_CLOUD_PROJECT` environment variable, and defaults to the project of the credentials. One of them is mandatory when the credentials do not define a project.

The `--location` flag selects the Certificate Manager location to read, `global` (default) or a region.

```
horizon-cli netimport gcm --campaign=test --credentials=<path to
service account JSON key> --project=<GCP project ID> --location=global
```

Google Cloud is reached through the proxy given with `--external-proxy`, or through the `HTTPS_PROXY`, `HTTP_PROXY` and `NO_PROXY` environment variables when no proxy is given.

It is also possible to attach the Google Certificate Manager third-party connector metadata to the discovered certificates, so that when the certificate is later managed in Horizon, renewals update the existing Google Certificate Manager object instead of creating a new one.

In order to activate this behavior, the `--connector` flag must reference a valid Google Certificate Manager Connector in Horizon.

```
horizon-cli netimport gcm --campaign=test --connector=<Horizon GCM
Connector name> --credentials=<path to service account JSON key>
--project=<GCP project ID>
```

If the certificate is already managed on a WebRA profile, the "Synchronize data from discovery" option must be enabled in the profile configuration, and the profile must reference the connector given as argument.

F5 BIG-IP

You can import all your valid certificates from F5 BIG-IP.

This feature requires the use of an administrator account on the F5 BIG-IP instance.

```
horizon-cli netimport bigip --campaign=test --hostname=<F5 BigIp
hostname> --login=<F5 BigIp login> --password=<F5 BigIp password>
```

It is also possible to import the certificates as managed certificates in Horizon. This will allow renewal and removal of the certificate upon revocation using Horizon's triggers mechanism.

In order to activate this behavior, the `connector` property must reference a valid F5 Connector in Horizon.

```
horizon-cli netimport bigip --campaign=test --connector=<Horizon F5
Connector name> --hostname=<F5 BigIp hostname> --login=<F5 BigIp login>
--password=<F5 BigIp password>
```

In order for the trigger mechanism to work correctly, an Horizon WebRA profile must use the F5 Connector trigger and a schedule task should reference the connector and the WebRA profile.

Global Options

- `--login-provider` specifies the login provider to use for TACACS to connect to the BIG-IP instance

IControl Options

- `--partition` specifies the F5 partition to retrieve the certificates from
- `--merge-third-party` enables fine tracking of the certificate across multiple SSL profiles. Enable this flag only with Horizon version superior or equal to `2.7.18`

AS3 Options

- `--as-3` enables AS3 compatibility
- `--filter-globs` is a list of globs to filter the certificates to import, based on the JSON path in the AS3 configuration

Akamai CPS

You can import all your valid certificates from Akamai Certificate Provisioning System. To do so, authentication credentials are required.

```
horizon-cli netimport akamai --campaign=test --host=<Akamai hostname>
--client-secret=<client secret> --client-token=<client token> --access
-token=<access token>
```

Gandi

You can import all your valid certificates from Gandi. To do so, authentication credentials are required.

```
horizon-cli netimport gandi --campaign=test --access-token=<access
token>
```

HashiCorp Vault

You can import all your certificates from HashiCorp Vault. The secret engines containing your certificates must be specified using the `--secrets-engines` option.

The required permissions for the scan operation are:

```
path "<engine>/certs/*" {
  capabilities = ["read", "list"]
}
```

As of the current version, both Token and AppRole authentication are supported. If another authentication method is required, a Token can be retrieved using Vault APIs and then used for the scan.

```
horizon-cli netimport vault --campaign=test --token=<token> --secrets
-engines pki
```

Nameshield

You can import all your certificates from Nameshield certificates.

```
horizon-cli netimport nameshield --campaign=test --token=<token>
```

Imported certificates carry the Nameshield certificate identifier in the `nameshield_cert_id` metadata.

Panorama

You can import all your certificates from PaloAlto Panorama.

The required permissions for the scan operation are:

```
XML API: Configuration
```

```
horizon-cli netimport panorama --campaign=test --api-key=<api key>
--hostname <panorama host>
```

By default, only the certificates deployed on the panorama (internal config and templates) will be scanned. If you wish to also scan each device managed by the panorama, use the `--scan-devices` option. If you do so, the `Operational Requests` permission must be added to the service account.

Chapter 7. EST Certificate Lifecycle Operations

EST Enrollment

Horizon Client is able to use the EST module of Horizon to enroll certificates.

Enrollment modes

The following enrollment modes are supported:

- Authorized user/password in decentralized mode
- Authorized user/password in centralized mode
- Challenge password in decentralized mode
- Challenge password in centralized mode
- Certificate swap in decentralized mode
- Certificate swap in centralized mode

Authorized user

In this enrollment mode, a local user account is created in Horizon for Horizon Client, and the EST profile on Horizon is configured in `authorized` mode thus a static username and password can be provided to Horizon Client for enrollment.They need to be set in general configuration as `APIID` and `APIKEY`.

```
horizon-cli est enroll --profile=test --dn=CN=TestCN [data parameters]
[key and certificate parameters]
```

Challenge password

In this enrollment mode, the EST profile on Horizon is set to `challenge` mode. A request must then be made on Horizon in order to retrieve the one-time password `challenge` to be used to authenticate the EST request.No `APIID` nor `APIKEY` need to be set.

Use the `--challenge` option.

```
horizon-cli est enroll --challenge=<challenge> --profile=test --dn=CN
=TestCN [data parameters] [key and certificate parameters]
```

Certificate swap

In this enrollment mode, the EST profile on Horizon is set to `x509` mode.The client is then able to make a request to Horizon by authenticating with an existing certificate.This certificate can be

specified either:

- by using the `--key` and `--cert` parameters, respectively pointing at the key and the certificate to be used to authenticate
- by using the `--win-store-auth` parameter (Windows only), that will look into the "MY" certificate store (user by default, unless `--win-machine-store` is specified) for a non-expired certificate whose CN matches the Common Name specified in `--cn` parameter

Use the `--in-cert`, `--win-user-store-auth` or `--win-computer-store-auth` option.

```
horizon-cli est enroll --in-cert=/path/to/cert/to/swap --in-key
=/path/to/key/to/swap  --profile=test --key=/path/to/key --cert
=/path/to/cert --cn=TestCN [data parameters] [key and certificate
parameters]

horizon-cli est enroll --win-user-store-auth --profile=test --cn=TestCN
[data parameters] [key and certificate parameters]
```

Decentralized mode

In decentralized mode, which is the default mode, Horizon Client generates a private key and a CSR.The CSR is generated according to the given certificate parameters, and the private key and the retrieved certificate are then stored according to the output parameters.

Centralized mode

In Centralized mode, triggered by adding the “--centralized” parameter to the command line, Horizon Client generates a fake private key and a CSR.The CSR is generated according to certificate parameters.The private key generated by Horizon Client is discarded.A random password is generated and inserted into the CSR.If the enrollment is successful, Horizon generates a private key and a certificate and sends them back to Horizon Client as PKCS#12, which Horizon Client decodes using the randomly generated password.The retrieved private key and the retrieved certificate are then stored according to the output parameters.

The random password generated has 16 characters, letters and numbers.If a password policy is enforced on Horizon side for the centralized mode in the considered EST profile, ensure that it is compatible with such characteristics.

General enrollment parameters

Table 2. General parameters

Parameter	Description
`--profile`	Horizon’s technical name of the profile to enroll on. Mandatory

`--challenge`	Challenge generated on Horizon on the profile. Mandatory in challenge mode
`--discovery`	Horizon's discovery campaign name to use in order to report the certificate to Horizon after enrollment
`--centralized`	Switches to centralized enrollment
`--script`	Path to the script to execute after enrollment. See script for more details

Input certificate parameters (x509 mode)

These parameters define how to find the certificate to swap in x509 mode. It can be stored in the following formats:

- Key and certificate stored separately in two files, in PEM format (`--in-cert` & `--in-key`)
- Key and certificate stored together in a PKCS#12 file (`--in-cert` & `--in-pfx-pwd`)
- Key and certificate stored together in a JKS file (`--in-cert` & `--in-jks-pwd` & `--in-jks-alias` & `--in-jks-alias-pwd`)
- Key and certificate stored together in Windows certificate store:
  - Using certificate thumbprint, available in the details tab of windows certificate explorer or in certutil (`--in-cert`)
  - Using the certificate CN (see Windows parameters)

Table 3. Input certificate parameters

Parameter	Description
`--in-cert`	Path to the certificate to renew (PEM file, PKCS#12 file, JKS file) or certificate thumbprint for Windows certificate store entries
`--in-key`	Path to the private key of the certificate to renew if --in-cert is a PEM file
`--in-pfx-pwd`	Password for the PKCS#12 file to renew
`--in-jks-pwd`	Password for the JKS file to renew
`--in-jks-alias`	Alias for the JKS file to renew
`--in-jks-alias-pwd`	Alias password for the JKS file to renew
`--in-kdb-pwd`	Password for the KDB file to renew
`--in-kdb-alias`	Alias for the KDB file to renew

Certificate parameters

Table 4. Data parameters

Parameter	Description
`--dn`	Requested subject DN. Single value. DN elements comma separated.
`--cn`	Requested subject Common Name. Single value
`--ou`	Requested subject `OU`. Can contain multiple values
`--dnsnames`	Requested subject alternative name DNS entries. Can contain multiple values
`--ip`	Requested subject alternative name IP entries. Can contain multiple values
`--emails`	Requested subject alternative name RFC822Name entries. Can contain multiple values

Table 5. Metadata parameters

Parameter	Description
`--contact-email`	Contact email of the request. Single value
`--owner`	Owner of the request. Single value
`--team`	Team of the request. Single value
`--labels`	Labels of the request. Can contain multiple values

Table 6. Crypto parameters

Parameter	Description
`--key-type`	Key-type of the certificate. See key types for more details

Output parameters

These parameters define how to store the retrieved certificate and its associated private key. The following alternatives are available:

- Key and certificate stored separately in two files, in PEM format. This is typically used by Apache or NGINX web servers;
- Key and certificate stored together in a PKCS#12 file. This is typically used by Tomcat application server;
- Key and certificate stored together in Windows certificate store. This is typically used by IIS web server (see Windows parameters)

Table 7. Output parameters

Parameter	Description

`--cert`	Path to the certificate to store
`--key`	Path to the private key to store
`--ca-chain`	Path to the chain to store
`--pfx`	Path to write the PKCS#12 output
`--pfx-pwd`	Password for the PKCS#12 output. Mandatory if `--pfx` is set
`--pfx-aes`	Enable AES encryption for PKCS#12, compatible with openssl v3
`--jks`	Path to write the JKS output
`--jks-pwd`	Password for the JKS output. Mandatory if `--jks` is set
`--jks-alias`	Alias for the JKS output. Mandatory if `--jks` is set
`--jks-alias-pwd`	Password for the alias in the JKS output. Mandatory if `--jks` is set
`--kdb`	Path to write the KDB output
`--kdb-pwd`	Password for the KDB output. Mandatory if `--kdb` is set
`--kdb-alias`	Alias for the KDB output. Mandatory if `--kdb` is set
`--overwrite`	Always overwrite existing files
`--update`	Update keystores when they already exist (JKS, KDB)

Windows parameters

These parameters define how to integrate with the Windows certificate store:

Table 8. Windows parameters

Parameter	Description
`--win-user-store-auth`	Triggers the use of Windows current user certificate store for certificate authentication. Most recent valid certificate with matching CN will be used
`--win-computer-store-auth`	Triggers the use of Windows local machine certificate store for certificate authentication. Most recent valid certificate with matching CN will be used
`--win-user-store-save`	Triggers the use of user Windows certificate store to save the certificate after enrollment

`--win-computer-store-save`	Triggers the use of computer Windows certificate store to save the certificate after enrollment
`--win-store-use-tpm`	Triggers the ability to store the certificate in the `Microsoft Platform Crypto Provider` KSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-use-legacy`	Triggers the ability to store the certificate in the legacy `Microsoft Enhanced Cryptographic Provider v1.0` CSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-set-exportable`	Marks the key as exportable from the Windows certificate store. If not specified, the key is not exportable

EST Renewal

The certificate renewal is performed by using the “renew” command.

- it is designed to renew a certificate already issued by Horizon on the same profile.
- it can be scheduled as a periodic task (cron or Scheduled Task), that will perform the renewal only when the certificate is N days before its expiration. N can be specified using the “--renewal-interval” parameter, and defaults to 30.

```
horizon-cli est renew --profile=test --in-cert=/path/to/cert/to/renew
[key and certificate parameters]

horizon-cli est renew --profile=test --win-store-auth --cn=TestCN [key
and certificate parameters]
```

General renewal parameters

Table 9. General parameters

Parameter	Description
`--profile`	Horizon’s technical name of the profile to enroll on. Mandatory
`--discovery`	Horizon’s discovery campaign name to use in order to report the certificate to Horizon after renewal
`--centralized`	Switches to centralized enrollment
`--key-type`	Key-type of the certificate. See key types for more details

`--script`	Path to the script to execute after renewal. See script for more details
`--renewal-interval`	Number of days before expiration to trigger the renewal. Defaults to 30

Input certificate parameters

These parameters define how to find the certificate to renew. It can be stored in the following formats:

- Key and certificate stored separately in two files, in PEM format (`--in-cert` & `--in-key`)
- Key and certificate stored together in a PKCS#12 file (`--in-cert` & `--in-pfx-pwd`)
- Key and certificate stored together in a JKS file (`--in-cert` & `--in-jks-pwd` & `--in-jks-alias` & `--in-jks-alias-pwd`)
- Key and certificate stored together in Windows certificate store:
  - Using certificate thumbprint, available in the details tab of windows certificate explorer or in certutil (`--in-cert`)
  - Using the certificate CN (see Windows parameters)

Table 10. Input certificate parameters

Parameter	Description
`--in-cert`	Path to the certificate to renew (PEM file, PKCS#12 file, JKS file) or certificate thumbprint for Windows certificate store entries
`--in-key`	Path to the private key of the certificate to renew if --in-cert is a PEM file
`--in-pfx-pwd`	Password for the PKCS#12 file to renew
`--in-jks-pwd`	Password for the JKS file to renew
`--in-jks-alias`	Alias for the JKS file to renew
`--in-jks-alias-pwd`	Alias password for the JKS file to renew
`--in-kdb-pwd`	Password for the KDB file to renew
`--in-kdb-alias`	Alias for the KDB file to renew

Output parameters

These parameters define how to store the retrieved certificate and its associated private key. The following alternatives are available:

- Key and certificate stored separately in two files, in PEM format. This is typically used by Apache or NGINX web servers;
- Key and certificate stored together in a PKCS#12 file. This is typically used by Tomcat

application server;

- Key and certificate stored together in Windows certificate store. This is typically used by IIS web server (see Windows parameters)

Table 11. Output parameters

Parameter	Description
`--cert`	Path to the certificate to store
`--key`	Path to the private key to store
`--ca-chain`	Path to the chain to store
`--pfx`	Path to write the PKCS#12 output
`--pfx-pwd`	Password for the PKCS#12 output. Mandatory if `--pfx` is set
`--pfx-aes`	Enable AES encryption for PKCS#12, compatible with openssl v3
`--jks`	Path to write the JKS output
`--jks-pwd`	Password for the JKS output. Mandatory if `--jks` is set
`--jks-alias`	Alias for the JKS output. Mandatory if `--jks` is set
`--jks-alias-pwd`	Password for the alias in the JKS output. Mandatory if `--jks` is set
`--kdb`	Path to write the KDB output
`--kdb-pwd`	Password for the KDB output. Mandatory if `--kdb` is set
`--kdb-alias`	Alias for the KDB output. Mandatory if `--kdb` is set
`--overwrite`	Always overwrite existing files
`--update`	Update keystores when they already exist (JKS, KDB)

Windows parameters

These parameters define how to integrate with the Windows certificate store:

Table 12. Windows parameters

Parameter	Description
`--dn`	Requested subject DN. Single value. DN elements comma separated.
`--cn`	CN of the certificate to renew in the Windows certificate store. Use with `--win-store-auth`

`--win-user-store-auth`	Triggers the use of Windows current user certificate store for certificate authentication. Most recent valid certificate with matching CN will be used
`--win-computer-store-auth`	Triggers the use of Windows local machine certificate store for certificate authentication. Most recent valid certificate with matching CN will be used
`--win-user-store-save`	Triggers the use of user Windows certificate store to save the certificate after enrollment
`--win-computer-store-save`	Triggers the use of computer Windows certificate store to save the certificate after enrollment
`--win-store-use-tpm`	Triggers the ability to store the certificate in the `Microsoft Platform Crypto Provider` KSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-use-legacy`	Triggers the ability to store the certificate in the legacy `Microsoft Enhanced Cryptographic Provider v1.0` CSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-set-exportable`	Marks the key as exportable from the Windows certificate store. If not specified, the key is not exportable

Key Types

Depending on your Horizon version, the following key types are supported:

RSA

To add a RSA key type, the following syntax must be used.

> rsa-<key-size>

`rsa-2048`, `rsa-3072`, `rsa-4096`

ECDSA

To add a ECDSA key type, the following syntax must be used.

> ec-<curve>

The following curves are supported:

- secp256r1
- secp384r1
- secp521r1

`ec-secp256r1`, `ec-secp384r1`

EDDSA

To add a EDDSA key type, the following syntax must be used.

> ed-<curve>

The following curves are supported:

- Ed25519

`ed-Ed25519`

Script parameter

You can tell Horizon Client to launch a script upon successful certificate enrollment or renewal by using the `--script` parameter, which takes the path to the script as an argument.

The script will receive arguments passed by Horizon Client in the following order:

1. Issued certificate serial number
2. Issued certificate fingerprint (SHA-1 hash of the certificate in DER format - windows store thumbprint)
3. Issued certificate Subject DN
4. Issued certificate Issuer DN

Below is an example of a very simple bash script:

```
#!/bin/sh

echo $1
echo $2
echo $3
echo $4
```

Below is an example of a very simple PowerShell script:

```
param($serial, $fingerprint, $subject, $issuer)

Write-Output $serial
```

```
Write-Output $fingerprint
Write-Output $subject
Write-Output $issuer
```

Examples

You will find below a few examples detailing how to use the client for EST enrollment in various context

Decentralized enrollment with challenge, output as key and certificate

```
horizon-cli est enroll --challenge=<challenge> --profile=<profile> --key=/path/to/key
--cert=/path/to/cert --dn=CN=test.example.com --dnsnames
=test.example.com,www.test.example.com
```

Decentralized enrollment with challenge, output as PKCS#12

```
horizon-cli est enroll --challenge=<challenge> --profile=<profile> --dn=CN
=test.example.com --dnsnames=test.example.com,www.test.example.com --pfx
=/path/to/pkcs12 --pfx-pwd=<pkcs12_password>
```

Centralized enrollment with challenge, output as key and certificate

```
horizon-cli est enroll --centralized --challenge=<challenge> --profile=<profile> --dn
=CN=test.example.com --dnsnames=test.example.com,www.test.example.com --cert
=/path/to/cert --key=/path/to/key
```

Centralized enrollment with challenge, output as PKCS#12

```
horizon-cli est enroll --centralized --challenge=<challenge> --profile=<profile> --dn
=CN=test.example.com --dnsnames=test.example.com,www.test.example.com --pfx
=/path/to/pkcs12 --pfx-pwd=<pkcs12_password>
```

Decentralized enrollment with challenge, output in machine windows store

```
horizon-cli est enroll --challenge=<challenge> --profile=<profile> --dn=CN
=test.example.com --dnsnames=test.example.com,www.test.example.com --win-store-save
```

```
--win-machine-store
```

Decentralized renewal from certificate and key, output as key and certificate

```
horizon-cli est renew --profile=<profile> --in-cert=/path/to/old/cert --in-key
=/path/to/old/key --cert=/path/to/new/cert --key=/path/to/new/key
```

Decentralized renewal from PKCS#12, output as key and certificate

```
horizon-cli est renew --profile=<profile> --in-cert=/path/to/old/pkcs12 --cert
=/path/to/cert --key=/path/to/key
```

Decentralized renewal using machine windows store

```
horizon-cli est renew --profile=<profile> --cn=test.example.com --win-store-auth --win
-store-save --win-machine-store
```

Chapter 8. SCEP Certificate Lifecycle Operations

The Horizon Client includes a SCEP client to perform challenge based pre-validated enrollments and renewals. Its usage is similar to that of the EST client in challenge mode.

Usage:

```
horizon-cli scep [command] [flags]
```

SCEP Enrollment

The `enroll` command allows you to perform a SCEP enrollment operation. It will generate a new key pair and a CSR based on the content parameters, and send it to the SCEP server to obtain a certificate.

Enrollment modes

The following enrollment modes are supported:

- Authorized user/password in decentralized mode
- Challenge password in decentralized mode

Authorized user

In this enrollment mode, a local user account is created in Horizon for Horizon Client, and the SCEP profile on Horizon is configured in `authorized` mode thus a static username and password can be provided to Horizon Client for enrollment.They need to be set in general configuration as `APIID` and `APIKEY`.

```
horizon-cli scep enroll --profile=test --dn=CN=TestCN [data parameters]
[key and certificate parameters]
```

Challenge password

In this enrollment mode, the SCEP profile on Horizon is set to `challenge` mode. A request must then be made on Horizon in order to retrieve the one-time password `challenge` to be used to authenticate the SCEP request.No `APIID` nor `APIKEY` need to be set.

Use the `--challenge` option.

```
horizon-cli scep enroll --challenge=<challenge> --profile=test --dn=CN
=TestCN [data parameters] [key and certificate parameters]
```

General enrollment parameters

Table 13. General parameters

Parameter	Description
`--profile`	Horizon's technical name of the profile to enroll on. Mandatory
`--challenge`	Challenge generated on Horizon on the profile. Mandatory in challenge mode
`--discovery`	Horizon's discovery campaign name to use in order to report the certificate to Horizon after enrollment
`--script`	Path to the script to execute after enrollment. See script for more details

Certificate parameters

Table 14. Data parameters

Parameter	Description
`--dn`	Requested subject DN. Single value. DN elements comma separated.
`--dnsnames`	Requested subject alternative name DNS entries. Can contain multiple values
`--ip`	Requested subject alternative name IP entries. Can contain multiple values
`--emails`	Requested subject alternative name RFC822Name entries. Can contain multiple values

Table 15. Metadata parameters

Parameter	Description
`--contact-email`	Contact email of the request. Single value
`--owner`	Owner of the request. Single value
`--team`	Team of the request. Single value
`--labels`	Labels of the request. Can contain multiple values

Table 16. Crypto parameters

Parameter	Description
`--key-type`	Key-type of the certificate. See key types for more details

Output parameters

These parameters define how to store the retrieved certificate and its associated private key. The following alternatives are available:

- Key and certificate stored separately in two files, in PEM format. This is typically used by Apache or NGINX web servers;
- Key and certificate stored together in a PKCS#12 file. This is typically used by Tomcat application server;
- Key and certificate stored together in Windows certificate store. This is typically used by IIS web server (see Windows parameters)

Table 17. Output parameters

Parameter	Description
`--cert`	Path to the certificate to store
`--key`	Path to the private key to store
`--ca-chain`	Path to the chain to store
`--pfx`	Path to write the PKCS#12 output
`--pfx-pwd`	Password for the PKCS#12 output. Mandatory if `--pfx` is set
`--pfx-aes`	Enable AES encryption for PKCS#12, compatible with openssl v3
`--jks`	Path to write the JKS output
`--jks-pwd`	Password for the JKS output. Mandatory if `--jks` is set
`--jks-alias`	Alias for the JKS output. Mandatory if `--jks` is set
`--jks-alias-pwd`	Password for the alias in the JKS output. Mandatory if `--jks` is set
`--kdb`	Path to write the KDB output
`--kdb-pwd`	Password for the KDB output. Mandatory if `--kdb` is set
`--kdb-alias`	Alias for the KDB output. Mandatory if `--kdb` is set
`--overwrite`	Always overwrite existing files
`--update`	Update keystores when they already exist (JKS, KDB)

Windows parameters

These parameters define how to integrate with the Windows certificate store:

Table 18. Windows parameters

Parameter	Description
`--win-user-store-save`	Triggers the use of user Windows certificate store to save the certificate after enrollment
`--win-computer-store-save`	Triggers the use of computer Windows certificate store to save the certificate after enrollment
`--win-store-use-tpm`	Triggers the ability to store the certificate in the `Microsoft Platform Crypto Provider` KSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-use-legacy`	Triggers the ability to store the certificate in the legacy `Microsoft Enhanced Cryptographic Provider v1.0` CSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-set-exportable`	Marks the key as exportable from the Windows certificate store. If not specified, the key is not exportable

SCEP Renewal

The `renew` command is designed to work similarly to the `enroll` command, but with a few differences:

- It will enroll a certificate based on the `--in-cert` parameter (or similar, see below) instead of the content parameters. Only the `--key-type` parameter is used to generate a new key pair.
- No challenge is needed for a SCEP renewal operation

General renewal parameters

Table 19. General parameters

Parameter	Description
`--profile`	Horizon's technical name of the profile to enroll on. Mandatory
`--discovery`	Horizon's discovery campaign name to use in order to report the certificate to Horizon after renewal
`--key-type`	Key-type of the certificate. See key types for more details
`--script`	Path to the script to execute after renewal. See script for more details

`--renewal-interval`	Number of days before expiration to trigger the renewal. Defaults to 30

Input certificate parameters

These parameters define how to find the certificate to renew. It can be stored in the following formats:

- Key and certificate stored separately in two files, in PEM format (`--in-cert` & `--in-key`)
- Key and certificate stored together in a PKCS#12 file (`--in-cert` & `--in-pfx-pwd`)
- Key and certificate stored together in a JKS file (`--in-cert` & `--in-jks-pwd` & `--in-jks-alias` & `--in-jks-alias-pwd`)
- Key and certificate stored together in Windows certificate store:
  - Using certificate thumbprint, available in the details tab of windows certificate explorer or in certutil (`--in-cert`)

Table 20. Input certificate parameters

Parameter	Description
`--in-cert`	Path to the certificate to renew (PEM file, PKCS#12 file, JKS file) or certificate thumbprint for Windows certificate store entries
`--in-key`	Path to the private key of the certificate to renew if --in-cert is a PEM file
`--in-pfx-pwd`	Password for the PKCS#12 file to renew
`--in-jks-pwd`	Password for the JKS file to renew
`--in-jks-alias`	Alias for the JKS file to renew
`--in-jks-alias-pwd`	Alias password for the JKS file to renew
`--in-kdb-pwd`	Password for the KDB file to renew
`--in-kdb-alias`	Alias for the KDB file to renew

Output parameters

These parameters define how to store the retrieved certificate and its associated private key. The following alternatives are available:

- Key and certificate stored separately in two files, in PEM format. This is typically used by Apache or NGINX web servers;
- Key and certificate stored together in a PKCS#12 file. This is typically used by Tomcat application server;
- Key and certificate stored together in Windows certificate store. This is typically used by IIS web server (see Windows parameters)

Table 21. Output parameters

Parameter	Description
`--cert`	Path to the certificate to store
`--key`	Path to the private key to store
`--ca-chain`	Path to the chain to store
`--pfx`	Path to write the PKCS#12 output
`--pfx-pwd`	Password for the PKCS#12 output. Mandatory if `--pfx` is set
`--pfx-aes`	Enable AES encryption for PKCS#12, compatible with openssl v3
`--jks`	Path to write the JKS output
`--jks-pwd`	Password for the JKS output. Mandatory if `--jks` is set
`--jks-alias`	Alias for the JKS output. Mandatory if `--jks` is set
`--jks-alias-pwd`	Password for the alias in the JKS output. Mandatory if `--jks` is set
`--kdb`	Path to write the KDB output
`--kdb-pwd`	Password for the KDB output. Mandatory if `--kdb` is set
`--kdb-alias`	Alias for the KDB output. Mandatory if `--kdb` is set
`--overwrite`	Always overwrite existing files
`--update`	Update keystores when they already exist (JKS, KDB)

Windows parameters

These parameters define how to integrate with the Windows certificate store:

Table 22. Windows parameters

Parameter	Description
`--win-user-store-save`	Triggers the use of user Windows certificate store to save the certificate after enrollment
`--win-computer-store-save`	Triggers the use of computer Windows certificate store to save the certificate after enrollment
`--win-store-use-tpm`	Triggers the ability to store the certificate in the `Microsoft Platform Crypto Provider` KSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used

`--win-store-use-legacy`	Triggers the ability to store the certificate in the legacy `Microsoft Enhanced Cryptographic Provider v1.0` CSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-set-exportable`	Marks the key as exportable from the Windows certificate store. If not specified, the key is not exportable

Key Types

Depending on your Horizon version, the following key types are supported:

RSA

To add a RSA key type, the following syntax must be used.

> rsa-<key-size>

`rsa-2048`, `rsa-3072`, `rsa-4096`

ECDSA

To add a ECDSA key type, the following syntax must be used.

> ec-<curve>

The following curves are supported:

- secp256r1
- secp384r1
- secp521r1

`ec-secp256r1`, `ec-secp384r1`

EDDSA

To add a EDDSA key type, the following syntax must be used.

> ed-<curve>

The following curves are supported:

- Ed25519

`ed-Ed25519`

Script parameter

You can tell Horizon Client to launch a script upon successful certificate enrollment or renewal by using the `--script` parameter, which takes the path to the script as an argument.

The script will receive arguments passed by Horizon Client in the following order:

1. Issued certificate serial number
2. Issued certificate fingerprint (SHA-1 hash of the certificate in DER format - windows store thumbprint)
3. Issued certificate Subject DN
4. Issued certificate Issuer DN

Below is an example of a very simple bash script:

```
#!/bin/sh

echo $1
echo $2
echo $3
echo $4
```

Below is an example of a very simple PowerShell script:

```
param($serial, $fingerprint, $subject, $issuer)

Write-Output $serial
Write-Output $fingerprint
Write-Output $subject
Write-Output $issuer
```

Examples

You will find below a few examples detailing how to use the client for SCEP enrollment in various context

Enrollment with output as key and certificate

```
horizon-cli scep enroll --profile=<profile> --challenge=<challenge> --dn=CN
=test.example.com --dnsnames=test.example.com,www.test.example.com --cert
=/path/to/cert --key=/path/to/key
```

Enrollment with lots of metadata and output as PKCS#12

```
horizon-cli scep enroll \
  --profile=<profile> \
  --challenge=<challenge> \
  --key-type=rsa-2048 \
  --dn=CN=test.example.com,OU=it \
  --dnsnames=test.example.com,www.test.example.com \
  --owner="John Doe" \
  --team="IT" \
  --labels="env:prod" \
  --pfx=/path/to/pkcs12 \
  --pfx-pwd=<pkcs12_password>
```

Renewal with output as key and certificate

```
horizon-cli scep renew --profile=<profile> --in-cert /path/to/cert --cert
=/path/to/cert --key=/path/to/key
```

Chapter 9. WebRA Certificate Lifecycle Operations

The Horizon Client can perform post-validated lifecycle operations using the WebRA protocol. This includes certificate enrollment, renewal and revocation.

WebRA operations validation

Like SCEP and EST, WebRA operations requires the intervention of a third party to validate the request. Unlike SCEP and EST though, it is a post-validation protocol, meaning that no challenge is produced before the operation, instead a request is created and sent to the WebRA server, which will need to be validated or cancelled by an operator with the appropriate rights on the web app.

This means the Horizon Client performs enrollment and renewal operations in two steps:

1. Create the request
2. Once the request is validated, retrieve the certificate

Depending on the time it takes for the request to be validated, the Horizon Client can be configured to either enter a blocking loop and wait for the request to be validated, or merely create the request and exit.

If the latter is chosen, the Horizon Client will keep in its internal database the pending request, and will check for its validation each time the `horizon-cli automate routine` command is executed. If the request is validated, the certificate will be retrieved and stored in the appropriate location. If it is denied, the request will be removed from the database. In some cases you would want to configure a crontab or scheduled task to perform this check periodically. You can use the command `horizon-cli automate create-periodic-task <period>` to help you in the process, or create it manually.

By default, the behavior is to create the request and exit. If you wish for the client to enter a blocking loop until the request is validated, specify the `--now` flag.

Enrollment can also be performed with a WebRA challenge, see challenge enrollment.

WebRA Enrollment

General enrollment parameters

Table 23. General parameters

Parameter	Description
`--profile`	Horizon's technical name of the profile to enroll on. Mandatory
`--now`	Start a blocking loop to wait for request approval

`--discovery`	Horizon's discovery campaign name to use in order to report the certificate to Horizon after enrollment
`--script`	Path to the script to execute after enrollment. See script for more details
`--challenge`	WebRA challenge to use for enrollment. No credentials are required in that mode. See challenge enrollment
`--centralized`	Use with `--challenge` to generate the certificate in centralized mode

Challenge enrollment

With `--challenge`, the certificate is enrolled with a WebRA challenge obtained from Horizon beforehand. No credentials are required in that mode, and the enrollment is synchronous: the certificate is returned directly and `--now` is ignored.

The key is generated locally (decentralized mode) unless `--centralized` is given, in which case it is generated by Horizon. The key type defaults to `rsa-2048` when `--key-type` is not given.

The `--owner`, `--team`, `--contact-email` and `--labels` options are ignored in that mode: they are defined by the challenge request.

```
horizon-cli webra enroll --profile=<profile> --challenge=<challenge>
--dn="CN=www.example.com" --dnsnames=www.example.com --cert=cert.pem
--key=key.pem
```

Certificate parameters

Table 24. Data parameters

Parameter	Description
`--dn`	Requested subject DN. Single value. DN elements comma separated.
`--dnsnames`	Requested subject alternative name DNS entries. Can contain multiple values
`--ip`	Requested subject alternative name IP entries. Can contain multiple values
`--emails`	Requested subject alternative name RFC822Name entries. Can contain multiple values
`--ms-guid`	Requested Microsoft GUID. Single value
`--ms-sid`	Requested Microsoft SID. Single value

Table 25. Metadata parameters

Parameter	Description
`--contact-email`	Contact email of the request. Single value
`--owner`	Owner of the request. Single value
`--team`	Team of the request. Single value
`--labels`	Labels of the request. Can contain multiple values

Table 26. Crypto parameters

Parameter	Description
`--key-type`	Key-type of the certificate. Defaults to `rsa-2048` with `--challenge`. See key types for more details

Output parameters

These parameters define how to store the retrieved certificate and its associated private key. The following alternatives are available:

- Key and certificate stored separately in two files, in PEM format. This is typically used by Apache or NGINX web servers;
- Key and certificate stored together in a PKCS#12 file. This is typically used by Tomcat application server;
- Key and certificate stored together in Windows certificate store. This is typically used by IIS web server (see Windows parameters)

Table 27. Output parameters

Parameter	Description
`--cert`	Path to the certificate to store
`--key`	Path to the private key to store
`--ca-chain`	Path to the chain to store
`--pfx`	Path to write the PKCS#12 output
`--pfx-pwd`	Password for the PKCS#12 output. Mandatory if `--pfx` is set
`--pfx-aes`	Enable AES encryption for PKCS#12, compatible with openssl v3
`--jks`	Path to write the JKS output
`--jks-pwd`	Password for the JKS output. Mandatory if `--jks` is set
`--jks-alias`	Alias for the JKS output. Mandatory if `--jks` is set

`--jks-alias-pwd`	Password for the alias in the JKS output. Mandatory if `--jks` is set
`--kdb`	Path to write the KDB output
`--kdb-pwd`	Password for the KDB output. Mandatory if `--kdb` is set
`--kdb-alias`	Alias for the KDB output. Mandatory if `--kdb` is set
`--overwrite`	Always overwrite existing files
`--update`	Update keystores when they already exist (JKS, KDB)

Windows parameters

These parameters define how to integrate with the Windows certificate store:

Table 28. Windows parameters

Parameter	Description
`--win-user-store-save`	Triggers the use of user Windows certificate store to save the certificate after enrollment
`--win-computer-store-save`	Triggers the use of computer Windows certificate store to save the certificate after enrollment
`--win-store-use-tpm`	Triggers the ability to store the certificate in the `Microsoft Platform Crypto Provider` KSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-use-legacy`	Triggers the ability to store the certificate in the legacy `Microsoft Enhanced Cryptographic Provider v1.0` CSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-set-exportable`	Marks the key as exportable from the Windows certificate store. If not specified, the key is not exportable

WebRA Renewal

The `webra renew` command is designed to work similarly to the `webra enroll` command, except that it will enroll a certificate based on the `--in-cert` parameter (or similar, see below) instead of the content parameters.

General renewal parameters

Table 29. General parameters

Parameter	Description
`--key-type`	Key-type of the certificate. See key types for more details
`--now`	Start a blocking loop to wait for request approval
`--discovery`	Horizon's discovery campaign name to use in order to report the certificate to Horizon after renewal
`--script`	Path to the script to execute after renewal. See script for more details
`--renewal-interval`	Number of days before expiration to trigger the renewal. Defaults to 30

Input certificate parameters

These parameters define how to find the certificate to renew. It can be stored in the following formats:

- Key and certificate stored separately in two files, in PEM format (`--in-cert` & `--in-key`)
- Key and certificate stored together in a PKCS#12 file (`--in-cert` & `--in-pfx-pwd`)
- Key and certificate stored together in a JKS file (`--in-cert` & `--in-jks-pwd` & `--in-jks-alias` & `--in-jks-alias-pwd`)
- Key and certificate stored together in Windows certificate store:
  - Using certificate thumbprint, available in the details tab of windows certificate explorer or in certutil (`--in-cert`)

Table 30. Input certificate parameters

Parameter	Description
`--in-cert`	Path to the certificate to renew (PEM file, PKCS#12 file, JKS file) or certificate thumbprint for Windows certificate store entries
`--in-key`	Path to the private key of the certificate to renew if --in-cert is a PEM file
`--in-pfx-pwd`	Password for the PKCS#12 file to renew
`--in-jks-pwd`	Password for the JKS file to renew
`--in-jks-alias`	Alias for the JKS file to renew
`--in-jks-alias-pwd`	Alias password for the JKS file to renew
`--in-kdb-pwd`	Password for the KDB file to renew

`--in-kdb-alias`	Alias for the KDB file to renew

Output parameters

These parameters define how to store the retrieved certificate and its associated private key. The following alternatives are available:

- Key and certificate stored separately in two files, in PEM format. This is typically used by Apache or NGINX web servers;
- Key and certificate stored together in a PKCS#12 file. This is typically used by Tomcat application server;
- Key and certificate stored together in Windows certificate store. This is typically used by IIS web server (see Windows parameters)

Table 31. Output parameters

Parameter	Description
`--cert`	Path to the certificate to store
`--key`	Path to the private key to store
`--ca-chain`	Path to the chain to store
`--pfx`	Path to write the PKCS#12 output
`--pfx-pwd`	Password for the PKCS#12 output. Mandatory if `--pfx` is set
`--pfx-aes`	Enable AES encryption for PKCS#12, compatible with openssl v3
`--jks`	Path to write the JKS output
`--jks-pwd`	Password for the JKS output. Mandatory if `--jks` is set
`--jks-alias`	Alias for the JKS output. Mandatory if `--jks` is set
`--jks-alias-pwd`	Password for the alias in the JKS output. Mandatory if `--jks` is set
`--kdb`	Path to write the KDB output
`--kdb-pwd`	Password for the KDB output. Mandatory if `--kdb` is set
`--kdb-alias`	Alias for the KDB output. Mandatory if `--kdb` is set
`--overwrite`	Always overwrite existing files
`--update`	Update keystores when they already exist (JKS, KDB)

Windows parameters

These parameters define how to integrate with the Windows certificate store:

Table 32. Windows parameters

Parameter	Description
`--win-user-store-save`	Triggers the use of user Windows certificate store to save the certificate after enrollment
`--win-computer-store-save`	Triggers the use of computer Windows certificate store to save the certificate after enrollment
`--win-store-use-tpm`	Triggers the ability to store the certificate in the `Microsoft Platform Crypto Provider` KSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-use-legacy`	Triggers the ability to store the certificate in the legacy `Microsoft Enhanced Cryptographic Provider v1.0` CSP. If not specified, the `Microsoft Software Key Storage Provider` KSP will be used
`--win-store-set-exportable`	Marks the key as exportable from the Windows certificate store. If not specified, the key is not exportable

WebRA Import

The `webra import`command is designed to import a certificate and its key on a profile.

Import parameters

Table 33. WebRA import parameters

Parameter	Description
`--profile`	Existing profile on which to import

Table 34. Metadata parameters

Parameter	Description
`--owner`	Owner of the request. Single value
`--team`	Team of the request. Single value
`--contact-email`	Contact email of the request. Single value
`--labels`	Labels of the request. Can contain multiple values
`--metadata`	Technical metadata of the imported certificate, in key:value form

Input parameters

These parameters define how to find the certificate to import. It can be stored in the following formats:

- Key and certificate stored separately in two files, in PEM format (`--in-cert` & `--in-key`)
- Key and certificate stored together in a PKCS#12 file (`--in-cert` & `--in-pfx-pwd`)
- Key and certificate stored together in a JKS file (`--in-cert` & `--in-jks-pwd` & `--in-jks-alias` & `--in-jks-alias-pwd`)
- Key and certificate stored together in Windows certificate store:
  - Using certificate thumbprint, available in the details tab of windows certificate explorer or in certutil (`--in-cert`)

Table 35. WebRA input certificate parameters

`--in-cert`	Path to the Certificate to import (PEM file, PKCS#12 file, JKS file) or cert thumbprint for Windows certificate store entries
`--in-key`	Path to the private key of the certificate to import if it is not included in the certificate file
`--in-pfx-pwd`	Password for the PKCS#12 file to import
`--in-jks-pwd`	Password for the JKS file to import
`--in-jks-alias`	Alias for the certificate to import in the JKS file
`--in-jks-alias-pwd`	Alias password for the JKS file to import

WebRA Revocation

The `webra revoke` command takes the following parameters:

Revocation parameters

Define how to revoke this certificate:

Table 36. WebRA revocation parameters

Parameter	Description
`--reason`	Reason for revocation (`unspecified`, `keycompromise`, `cacompromise`, `affiliationchanged`, `superseded`, `cessationofoperation`)

Input parameters

These parameters define how to find the certificate to revoke. It can be stored in the following formats:

- Key and certificate stored separately in two files, in PEM format (`--in-cert` & `--in-key`)
- Key and certificate stored together in a PKCS#12 file (`--in-cert` & `--in-pfx-pwd`)
- Key and certificate stored together in a JKS file (`--in-cert` & `--in-jks-pwd` & `--in-jks-alias` & `--in-jks-alias-pwd`)
- Key and certificate stored together in Windows certificate store:
  - Using certificate thumbprint, available in the details tab of windows certificate explorer or in certutil (`--in-cert`)

Table 37. WebRA input certificate parameters

`--in-cert`	Path to the Certificate to revoke (PEM file, PKCS#12 file, JKS file) or cert thumbprint for Windows certificate store entries
`--in-key`	Path to the private key of the certificate to revoke if it is not included in the certificate file
`--in-pfx-pwd`	Password for the PKCS#12 file to revoke
`--in-jks-pwd`	Password for the JKS file to revoke
`--in-jks-alias`	Alias for the certificate to revoke in the JKS file
`--in-jks-alias-pwd`	Alias password for the JKS file to revoke

For a revocation operation, the `--now` flag is unavailable, and the `automate routine` command will not track the revocation status, as no actions are to be performed after the revocation is complete. This command thus merely creates the revocation request and exits.

Key Types

Depending on your Horizon version, the following key types are supported:

RSA

To add a RSA key type, the following syntax must be used.

> rsa-<key-size>

`rsa-2048`, `rsa-3072`, `rsa-4096`

ECDSA

To add a ECDSA key type, the following syntax must be used.

> ec-<curve>

The following curves are supported:

- secp256r1
- secp384r1
- secp521r1

`ec-secp256r1`, `ec-secp384r1`

EDDSA

To add a EDDSA key type, the following syntax must be used.

> ed-<curve>

The following curves are supported:

- Ed25519

`ed-Ed25519`

Script parameter

You can tell Horizon Client to launch a script upon successful certificate enrollment or renewal by using the `--script` parameter, which takes the path to the script as an argument.

The script will receive arguments passed by Horizon Client in the following order:

1. Issued certificate serial number
2. Issued certificate fingerprint (SHA-1 hash of the certificate in DER format - windows store thumbprint)
3. Issued certificate Subject DN
4. Issued certificate Issuer DN

Below is an example of a very simple bash script:

```
#!/bin/sh

echo $1
echo $2
echo $3
echo $4
```

Below is an example of a very simple PowerShell script:

```
param($serial, $fingerprint, $subject, $issuer)

Write-Output $serial
```

```
Write-Output $fingerprint
Write-Output $subject
Write-Output $issuer
```

Examples

You will find below a few examples detailing how to use the client for WebRA enrollment in various context

Enrollment with output as key and certificate, waiting for the certificate to be issued

```
horizon-cli webra enroll --profile=<profile> --dn=CN=test.example.com --dnsnames
=test.example.com,www.test.example.com  --cert=/path/to/cert --key=/path/to/key --now
```

Enrollment with lots of metadata, output as PKCS#12 and no blocking loop

```
horizon-cli webra enroll \
  --profile=<profile> \
  --dn=CN=test.example.com \
  --dnsnames=test.example.com,www.test.example.com \
  --owner="John Doe" \
  --ou="IT" \
  --team="IT" \
  --labels="env:prod" \
  --pfx=/path/to/pkcs12 \
  --pfx-pwd=<pkcs12_password>
```

after this command, run periodically:

```
horizon-cli automate routine > /path/to/my/logfile
```

Renewal with output as key and certificate, waiting for the certificate to be issued

```
horizon-cli webra renew --in-cert /path/to/old/cert --in-key /path/to/old/key --cert
/path/to/cert --key /path/to/key --now
```

Revocation of a certificate

```
horizon-cli webra revoke --in-cert /path/to/cert --in-key /path/to/key
```

Chapter 10. Updating a certificate

The horizon client can perform update operations on certificates using the `update-cert` command. This will modify the information associated with the certificate on Horizon.

This command can either be used to update a certificate present on your machine or to update certificates on Horizon using an account with sufficient permission.

To update a local certificate, the 'Update (pop)' common configuration permission must be enabled on the profile the certificate is linked to.

General Parameters

`--confirm`	The command asks for confirmation after the changes are computed. Use this flag to disable this behavior and proceed directly. (Optional)
`--prompt`	Use this flag to be prompted for edition of all the certificate fields. In this mode, using enter on an existing value means the value is not changed. (Optional)

Update Parameters

An update concerns only metadata fields, that is fields added by Horizon.

`--owner`	Set the owner of the certificate. An empty string means deletion of this information. (Optional)
`--team`	Set the team of the certificate. An empty string means deletion of this information. (Optional)
`--contact-email`	Set the contact email of the certificate. An empty string means deletion of this information. (Optional)
`--labels`	Set the labels of the certificate. An empty string means deletion of this information. (Optional)
`--metadata`	Set the technical metadata of the certificate. To use with caution. An empty string means deletion of this information. (Optional)

Certificate selection parameters

Local certificate

The update is only possible on local certificates for which you possess the key:

`--cert`	Path to the certificate to update (PEM file, PKCS#12 file, JKS file) or cert thumbprint for Windows certificate store entries. (Optional)
`--key`	Path to the private key of the certificate to update if it is not included in the certificate file. (Optional)
`--pfx-pwd`	Password for the PKCS#12 file to update. (Optional)
`--jks-pwd`	Password for the JKS file to update. (Optional)
`--jks-alias`	Alias for the JKS file to update. (Optional)
`--jks-alias-pwd`	Alias password for the JKS file to update. (Optional)

Certificate on Horizon server

An account must be configured on the client using `horizon-cli install` and it must have update permissions on the certificate

`--id`	Id of the certificate to update (Optional)

Examples

You will find below a few examples detailing how to use the client to update certificates in various contexts

Updating the owner of a certificate

```
horizon-cli update-cert --cert=/path/to/cert --key=/path/to/key --owner=newowner
```

Removing the team from a certificate stored in JKS file

```
horizon-cli update-cert --cert=/path/to/cert.jks --jks-pwd=<jks_password> --team=""
```

Updating labels and metadata of a certificate stored in windows certificate store

```
horizon-cli update-cert --cert=<certificate_thumbprint> --labels
="label1:value1,label2:value2" --metadata="metadata1:value1,metadata2:value2"
```

Updating contact email of a certificate referenced on Horizon

```
horizon-cli update-cert --id=<certificate_id> --contact-email="test@evertrust.fr"
```

Chapter 11. Bulk Operations

The horizon client allows you to perform bulk operations on certificates using the *Horizon Certificate Query Language* (HCQL).

Bulk update

The `bulk update` command allows update of certificate metadata en masse. The command takes a HCQL query as parameter and updates the matching certificates with the provided metadata. You can update the **owner**, the **team**, the **labels** and the **contact email** of the certificates. To unset an existing value use the value *"unset"*.

Table 38. Bulk update command parameters

Parameter	Description
`--query`	The HCQL query string. Update will be performed on results.
`--confirm`	Skip confirm.
`--owner`	The owner to set on certificates matching the query. (Optional)
`--team`	The team to set on certificates matching the query. (Optional)
`--labels`	The labels, in the comma separated key:value form, to set on certificates matching the query. (Optional)
`--contact-email`	The contact email to set on certificates matching the query. (Optional)

```
horizon-cli bulk update --query 'module equals "est" and status is
valid' --owner "myuser" --team "myteam" --labels "mylabel:myvalue"
--contact-email "unset"
```

Bulk migrate

The `bulk migrate` command allows certificate migration from one profile to another. The command takes an HCQL query as parameter and migrate the matching certificates to the provided profile. The command can also update the certificates metadata. You can update the **owner**, the **team**, the **labels** and the **contact email** of the certificates. To unset an existing value use the value *"unset"*.

Table 39. Bulk migrate command parameters

Parameter	Description
`--query`	The HCQL query string. Update will be performed on results.

Parameter	Description
`--confirm`	Skip confirm.
`--profile`	The target profile for the migration.
`--owner`	The owner to set on certificates matching the query. (Optional)
`--team`	The team to set on certificates matching the query. (Optional)
`--labels`	The labels, in the comma separated key:value form, to set on certificates matching the query. (Optional)
`--contact-email`	The contact email to set on certificates matching the query. (Optional)

```
horizon-cli bulk migrate --query 'module equals "est" and status is
valid' --profile new-est-profile --team myteam --labels mylabel:myvalue
```

Bulk revoke

The `bulk revoke` command allows certificate revocation en masse. The command takes an HCQL query as parameter and revoke the matching certificates.

Table 40. Bulk revoke command parameters

Parameter	Description
`--query`	The HCQL query string. Update will be performed on results.
`--confirm`	Skip confirm.

```
horizon-cli bulk revoke --query 'team equals "myterminatedteam" and
status is valid' --confirm
```

Chapter 12. Automatic TLS Certificate Installation

The horizon-cli automate command helps you automate the installation of your TLS certificates. It is designed to streamline the process of certificate enrollment, renewal, and installation. This functionality is particularly useful for managing Transport Layer Security (TLS) on web servers, ensuring secure and encrypted connections. It simplifies complex operations through its suite of subcommands, each tailored for specific aspects of certificate management.

Subcommands

To begin with Horizon CLI automation, use the `--help` flag with any subcommand for detailed usage information:

```
./horizon-cli automate <subcommand> --help
```

Table 41. Automation module subcommands

Subcommand	Description
enroll	This command is versatile, functioning similarly to the init command on servers without certificates, while also capable of re-enrolling or taking control of servers with existing certificates.
init	This command is designed to automatically configure SSL on any newly installed web server. When executed, it seamlessly sets up a secure connection by arranging SSL certificates and enabling SSL on the default HTTPS port (usually port 443). The process includes configuring the necessary SSL settings and restarting the server with SSL enabled.
control	This command is designed to locate certificates within a web server's configuration and offers to bring them under automation control. Taking a certificate under control involves adding necessary metadata and incorporating it into the local memory for routine management and oversight. This process ensures that the certificates are systematically monitored and managed as part of the automated workflow. Be aware that enrolling a new certificate will replace any certificate currently bound to an application running on your machine.

Subcommand	Description
modify	This command allows the user to select one of the managed certificates and modify its sans.
edit	This command allows the user to change values on the state of currently managed certificates.
create-periodic-task	Sets up a routine task for automated certificate renewal. The task frequency can be specified, with a default period of 6 hours. This helps in automating the renewal process to ensure certificates remain valid without manual intervention.
remove-periodic-task	Removes the previously set periodic task for certificate renewal. This is useful when automated renewal is no longer needed or needs to be reconfigured.
routine	Performs a routine check on all managed certificates to assess if any need renewal. This is a part of proactive certificate management to avoid service disruptions.
list	Lists all managed certificates. This provides an overview of all certificates under management, including details like expiration dates, domains covered, etc.
remove	Removes a managed certificate or a group of certificates. The <id> parameter specifies which certificates to remove, with the option to remove all services using a keyword like 'all'.

If you're looking for a more guided experience while using horizon-cli automate, the `--prompt` flag is an excellent tool. It enables interactive prompts that guide you step-by-step through the command's options, making it easier to configure and execute your commands accurately.

```
./horizon-cli automate <subcommand> --prompt
```

Supported Services

The horizon-cli automate feature supports a variety of web server and application server services. This ensures a wide range of compatibility and flexibility for users working with different server environments. The `--target` flag is used within the tool to specify the specific service on which to perform automation tasks, allowing for precise and targeted configuration and management.

In order for horizon-cli to function properly, the targeted service should be started

on the machine.

List of the supported services:

- `nginx` (Linux)
- `apache` (Linux)
- `haproxy` (Linux)
- `jboss wildfly` (Linux)
- `lighttpd` (Linux)
- `microsoft iis` (Windows)
- `evertrust winhorizon` (Windows)
- `evertrust adcsconnector` (Windows)
- `Windows` (Windows)
- `tomcat` (Linux & Windows)
- `generic` (Linux & Windows)

Example of a standard setup on the default HTTPS port (443) for NGINX:

```
./horizon-cli automate init --target=nginx --automation-policy=<POLICY_NAME>
```

To set up SSL on a custom port (e.g., 9000), use the `--PORT` flag. This option allows SSL configuration on a specific port of your Nginx server.

```
./horizon-cli automate init --target=nginx --automation-policy
=<POLICY_NAME> --PORT=9000
```

For `microsoft iis`, `evertrust winhorizon` and `evertrust adcsconnector` services, the renewal will remove the old certificates (except the original one for backup purposes) from the windows store.

Generic Service Automation

The Generic Service within the Horizon Client's automation module provides a versatile and user-friendly method for obtaining TLS certificates, eliminating the need to specify a pre-existing service or certificate file. This feature is especially beneficial in scenarios where certificates are required to be generated dynamically or for users who seek a more automated certificate management process.

Through the interactive mode:

```
./horizon-cli automate init --target=generic --prompt
```

Non-Interactive Mode: For an even more streamlined experience, certificates can be automatically generated with default parameters in non-interactive mode.

```
./horizon-cli automate init --target=generic --automation-policy=<POLICY_NAME> --cert
=my_cert.pem --key=my_key.key --chain-file=my_chain.pem --no-interactive
```

Default Storage Location: Newly generated certificates are automatically stored in a default location, which is **/opt/horizon/var/generic** on Unix systems, and **C:\ProgramData\EverTrust\Horizon\Var\Generic** on Windows systems.

Configuration Folder Override: Users have the flexibility to override the default storage location using the `--config-folder` option. This allows for customization of the storage path as per individual requirements or organizational standards.

```
./horizon-cli automate init --config-folder=/path/to/folder --target=generic
--automation-policy=<POLICY_NAME> --pfx=my_cert.p12 --pfx-pwd=pass123 --chain-file
=my_chain.pem --no-interactive
```

The Generic Service's emphasis on flexibility and user-friendliness makes it a valuable tool for a wide range of users, from those requiring on-the-fly certificate generation to those preferring a hands-off, automated approach.

Windows-Specific Features: On Windows systems, the Horizon CLI offers additional flags to specify the certificate store location:

`--win-user-store`: Save the certificate in the user store. This option is beneficial when certificates need to be accessible on a per-user basis.

`--win-computer-store`: Save the certificate in the computer store. Ideal for certificates that must be available system-wide.

These options provide flexibility in managing certificate storage, catering to different security and accessibility requirements on Windows systems.

Windows Service Automation

The `windows` target handles application that use a certificate in the Windows store without specific configuration files.

It supports:

- Remote Desktop certificates (`rdp`)
- Domain Controller certificates (`domaincontroller`)

To enroll a RDP certificate:

Through the interactive mode:

```
./horizon-cli automate init --target=windows --prompt
```

Non-Interactive Mode: Specific windows usages can be targeted using the `--win-usages` option

```
./horizon-cli automate init --target=windows --win-usages rdp,domaincontroller
```

Automation policies

Automation policies are central to the operation of the automation module in the Horizon system. These policies dictate how certificates should be enrolled and renewed, providing a customizable framework to suit various client requirements.

Before using automation policies, they must be pre-configured in the Horizon web app. Each policy is given a unique name for easy identification.

Profile Selection: Policies can be based on EST, SCEP, WebRA or ACME profiles, depending on the specific requirements of the enrollment and renewal process.

EverTrust recommends using EST for most use cases of server automation.

Execution Policy: Includes settings that define how and when the automation should be executed.

Compliance Settings: Specify which CAs are authorized for use within the policy.

Authorized Hash Algorithms: Determine which hash algorithms are acceptable.

Trust Chains: Configure the trust chains that are essential for establishing the trustworthiness of the certificates.

Parameter Specification: When performing automation operations, the relevant automation policy is specified using the `--automation-policy` parameter.

Existing WebServer certificates

The Horizon CLI's automation module is equipped with a discovery feature that scans and identifies certificates on your machine. It does this by parsing configuration files of your web server or TLS service.

By default, it searches for certificates across all supported services.

```
./horizon-cli automate enroll --automation-policy=<POLICY_NAME>
```

Limit the search to specific services. Separate multiple services with commas.

```
./horizon-cli automate enroll --automation-policy=<POLICY_NAME>
```

```
--target=apache,nginx
```

If you use a different config folder than the default one you can specify your custom folder using `--config-folder`

```
./horizon-cli automate enroll --automation-policy=<POLICY_NAME>
--target=apache,nginx --config-folder=/path/to/folder
```

Perform only the discovery phase and print the results without enrolling certificates using `--analyze-only`:

```
./horizon-cli automate enroll --automation-policy=<POLICY_NAME>
--target=apache,nginx --analyze-only
```

The `--discovery` parameter allows you to integrate certificate enrollment with a discovery campaign pre configured in the Horizon web app before running the CLI command. It includes additional information on the certificate's usage and location on the host machine.

```
./horizon-cli automate enroll --automation-policy=<POLICY_NAME> --discovery-campaign
=<DISCOVERY_CAMPAIGN_NAME>
```

Additional enrollment parameters

Challenge Passwords for Enrollment

The `--challenge` parameter is used to provide challenge passwords for EST, SCEP or WebRA during the enrollment process. Challenge passwords are crucial for the authentication phase in these protocols, ensuring secure communication and identity verification.

For WebRA, the challenge is used when the automation policy is configured with the challenge authorization mode. Without `--challenge` or `--request-challenge`, the challenge is prompted for in interactive mode, and the enrollment fails in non-interactive mode.

When enrolling certificates using EST, SCEP or WebRA protocols, the `--challenge` parameter allows you to specify one or more challenge passwords required by the enrollment server. Multiple challenge passwords can be provided, separated by commas, to support various scenarios or multiple servers enrollment.

You can also use the `--request-challenge` parameter to create a challenge request on Horizon. When the challenge request will be validated a periodic task will get the challenge and finish the enrollment. The periodic task can be configured with the `--challenge-routine-period` parameter.

You must be authenticated to use the `--request-challenge` parameter.

Post-Enrollment Script Execution

The `--script` parameter allows the execution of a custom script upon the successful completion of a certificate enrollment process. It supports both Bash scripts in Linux environments and PowerShell scripts in Windows environments.

```
./horizon-cli automate enroll --automation-policy=<POLICY_NAME> --script
=/home/user/post_enroll.sh
```

The script will receive arguments passed by Horizon Client in the following order:

1. Issued certificate serial number
2. Issued certificate fingerprint (SHA-1 hash of the certificate in DER format - windows store thumbprint)
3. Issued certificate Subject DN
4. Issued certificate Issuer DN
5. Storage information

Below is an example of a very simple bash script:

```
#!/bin/sh

echo $1
echo $2
echo $3
echo $4
echo $5
```

Below is an example of a very simple PowerShell script:

```
param($serial, $fingerprint, $subject, $issuer, $storage)

Write-Output $serial
Write-Output $fingerprint
Write-Output $subject
Write-Output $issuer
Write-Output $storage
```

Storage information

The storage information is a JSON array containing information about the storage, with one object per storage.

```
[
```

```
  {
    "type": "jks",
    "path": "/path/to/jks",
    "alias": "jks alias",
    "password": "jks password",
    "caChainPath": "/path/to/chain"
  }
]
```

Storage are often linked to a target. Here are the available storages for each target:

- `apache`: Certificate, chain and key in separate files or Chain and key in separate files
- `nginx`: Chain and key in separate files
- `tomcat`: Certificate, chain and key in separate files or JKS or PKCS#12
- `lighttpd`: Chain and key in separate files
- `wildfly`: JKS or PKCS#12
- `iis`: Windows store entry
- `winhorizon`: Windows store entry or PKCS#12
- `adcsconnector`: Windows store entry
- `windows`: Windows store entry
- `generic`: Certificate, chain and key in separate files or Windows store entry or JKS or PKCS#12
- `haproxy`: Chain and key in separate files or Chain and key in a single file

Storage Types

Chain and key in a single file

Description:

- `path` contains the path to the certificate chain (including leaf certificate, in leaf to root order) followed by the certificate private key in PEM format

Example:

```
[
  {
    "type": "bundle",
    "path": "/path/to/cert+chain+key"
  }
]
```

Chain and key in separate files

Description:

- `chainPath` contains the path to the certificate chain (including leaf certificate, in leaf to root order) in PEM format
- `keyPath` contains the path to the certificate key in PEM format

Example:

```
[
  {
    "type": "chainKey",
    "chainPath": "/path/to/cert+chain",
    "keyPath": "/path/to/key"
  }
]
```

Certificate, chain and key in separate files

Description:

- `certPath` contains the path to the certificate (without chain) in PEM format
- `keyPath` contains the path to the certificate key in PEM format
- `caChainPath` contains the path to the certificate chain (excluding leaf certificate, in leaf to root order) in PEM format
- `certFormat` contains the format of the certificate file (PEM or DER)

Example:

```
[
  {
    "type": "certChainKey",
    "certPath": "/path/to/cert",
    "keyPath": "/path/to/key",
    "caChainPath": "/path/to/chain",
    "certFormat": "PEM or DER"
  }
]
```

Windows store entry

Description:

- `thumbprint` contains the thumbprint of the certificate in the windows store
- `machineStore` if true, means the certificate is stored in the machine store, else in the user store

Example:

```
[
```

```
  {
    "type": "windowsStore",
    "thumbprint": "thumbprint of the certificate",
    "machineStore": false
  }
]
```

JKS

Description:

- `path` is the path to the JKS file
- `alias` contains the alias in which the certificate is stored
- `password` contains the JKS password
- `caChainPath`(optional) contains the path to the chain file (excluding leaf certificate, in root to leaf order) in PEM format

Example:

```
[
  {
    "type": "jks",
    "path": "/path/to/jks",
    "alias": "jks alias",
    "password": "jks password",
    "caChainPath": "/path/to/chain"
  }
]
```

PKCS#12

Description:

- `path` is the path to the pkcs12 file
- `password` contains the pkcs12 password
- `caChainPath`(optional) contains the path to the chain file (excluding leaf certificate, in root to leaf order) in PEM format

Example:

```
[
  {
    "type": "pkcs12",
    "path": "/path/to/pkcs12",
    "password": "pkcs12 password",
    "caChainPath": "/path/to/chain"
  }
```

```
]
```

Metadata parameters

***Add metadata to certificates during the enrollment process*:** These certificate information parameters enhance the management and traceability of TLS certificates. By using these optional fields, organizations can maintain better oversight and control over their certificate infrastructure.

Table 42. Metadata parameters

Parameter	Description
`--owner`	Owner of the certificate
`--contact-email`	Contact email of the certificate owner
`--team`	Team owning the certificate
`--labels`	Labels to attach to the certificate, in the form `key:value`

ACME Account Specification for Enrollment

The `--acme-account` parameter is mandatory when enrolling certificates using the ACME protocol. It specifies the ACME account to be used for the enrollment process.

```
./horizon-cli automate enroll --acme-account=myAcmeAccount
```

ACME IP identifiers

IP SANs given with `--ip` are requested as ACME IP identifiers (RFC 8738). This requires Horizon 2.11 or above. An IP identifier can only be validated with the `http-01` or `tls-alpn-01` challenge, so the enrollment fails when neither is available for it.

When both DNS names and IPs are requested, the first DNS name is used as CN. An IP is used as CN only when no DNS name is requested.

```
./horizon-cli automate enroll --automation-policy=<POLICY_NAME> --acme-account
=<ACME_ACCOUNT> --target=generic --dnsnames=www.example.com --ip=192.0.2.10
```

ACME DNS-01 propagation tuning

When solving an ACME DNS-01 challenge, horizon-cli runs an authoritative name server propagation pre-check against the DNS provider before notifying the ACME server. In private DNS environments (for example, AWS Route 53 private hosted zones or split-horizon DNS deployments), the public authoritative name servers exposed by the zone do not serve the private records, and the pre-check fails even though the ACME server-side validation would succeed.

Two options on the `enroll`, `init` and `control` commands allow tuning this behavior:

- `--dns-01-propagation-wait` (duration): skips the client-side authoritative name server pre-check and instead waits the configured duration for the DNS record to propagate before notifying the ACME server. The ACME server-side validation is unaffected.
- `--dns-01-timeout` (duration): sets the overall DNS-01 operation timeout, which sizes the propagation wait loop.

```
./horizon-cli automate enroll --automation-policy=<POLICY_NAME> --acme
-account=<ACME_ACCOUNT> --dns-01-provider=<PROVIDER_SCRIPT> --dns-01
-propagation-wait=2m --dns-01-timeout=5m
```

Installation

After a successful enrollment or renewal, the certificate will be installed on your machine. The impacted services will be restarted automatically after each certificate enrollment or renewal.

In certain scenarios, you might not want the Horizon CLI to automatically install the new certificate or restart the related services.

If you wish for the client to **not install** your new certificate, that is, not replace the old certificate and not restart the impacted services, you can use the `--no-install` option. Each new file (cert, key, CA chain, keystore...) will then be placed in the same folder as it's predecessor, with the `.new` extension.

Renewal

Each time a certificate is discovered and enrolled by the automation module of the Horizon Client, its details are stored in the internal database for future reference. Each time the `automate routine` command is run, the client will check if any of the locally known certificates need to be renewed. Reasons for renewal can be:

- The certificate is about to expire
- The certificate has been revoked
- Preferences such as key type or enrollment CA were changed in the profile or automation policy

If a certificate needs to be renewed, the client will perform the renewal according to the automation policy, and its corresponding profile.

We recommend that you run the `automate routine` command periodically as a cron job or scheduled task. You can use the command `horizon-cli automate create-periodic-task <period>` or the flag `--auto-renew` on the `automate enroll` command to help you in the process, or create it manually.

This mechanism allows for more resilient web servers, as the certificates will be renewed

automatically, before any interruption of service can happen because of an expired or revoked certificate. It also helps your organisation migrate your TLS certificates to a new CA quickly, by simply changing the preferred enrollment CA in the automation policy and waiting a few hours for all your instances of the Horizon Client to execute their routine tasks.

Using `--auto-renew` flag will check and check the certificate every 6 hours:

```
./horizon-cli automate enroll --target=nginx --automation-policy
=<POLICY_NAME> --auto-renew
```

Interactivity

Two options are available to control the interactivity of the `automate` commands:

- The `--no-interactive` flag will prevent any prompt from being displayed, and will use the default values or those provided in the command line arguments. It will:
  - select all discovered certificates for enrollment. In order to select specific certificates without interaction, the `--select-certs` flag can be used to specify a glob matching the website identifier.

Example: Running the `automate enroll --analyze-only --automation-policy <automation policy name>` returns the following result:

CN	Locations	Bindings
New Certificate	Not yet prompted	generic-893c8435-08c5-4d2d-b9f6-88b952b34ca4
New Certificate	/etc/pki/nginx/server.crt	nginx-*:443

To select and enroll only the nginx cert, the following command can be used `automate enroll --automation-policy <automation policy name> --no-interactive --select-certs nginx*`

- if a challenge is required, use the provided `--challenge` argument. If no challenge is provided, or the given challenge has already been used, the enrollment will fail.
- not add any additional SANs
  - The `--prompt` flag will force the client to prompt the user for any missing information. If specified, any other command line arguments are optional. It will:
- prompt the user to select which services to search for on the machine (equivalent to the `--target` option)
- prompt the user for the automation policy (equivalent to the `--automation-policy` option)
- prompt the user for the configuration folder (equivalent to the `--config-folder` option)

Certificate commands

Enroll

The `enroll` command can either setup a certificate and empty https configuration from scratch, or take control of certificates, reenrolling them to be compliant when necessary.

Parameters

Table 43. Enroll general parameters

Parameter	Mandatory	Type	Description
--automation-policy	☑	string	The automation policy to link the certificate to.
--challenge		string array (comma separated)	See challenge section.
--script		string (path to file)	See script section.
--auto-renew		boolean	Configures a status check every 6 hours.
--request-challenge		boolean	See challenge section.
--challenge-routine-period		duration	Period of execution of the periodic task.
--user		string	Name of the user to impersonate while running the request challenge periodically.
--discovery-campaign		string	Also add discovery info on the enrolled certificate on the campaign passed.
--force-enroll		boolean	Reenroll all selected certificates regardless of their compliance.
--analyze-only		boolean	Instead of executing the enrollments, only displays a summary of possible actions.
--no-interactive		boolean	Disables all interactive inputs. All parameters must then be given using cli flags.
--prompt		boolean	Enable all interactive inputs. All parameters will now be asked for, except ones given using cli flags.

Table 44. Enroll webServer configuration

Parameter	Mandatory	Type	Description
--target		string array (comma separated)	List of services to target. If not given, all available services are targeted.
--no-install		boolean	Disables installation.
--config-folder		string (path to folder)	Explicitly point your webserver configuration folder.
--port		integer	When initializing an empty https configuration on a blank server, choose on which port to listen for https (defaults to the standard https port for the webserver *443* or *8443*)
--keystore-password		string	Password of the webserver's keystore if it cannot be deduced from configuration

Table 45. Enroll certificate configuration

Parameter	Mandatory	Type	Description
--dn		DN as string, DN elements comma separated.	Requested subject DN. If no CN is specified here, it will be deduced from requested SANs
--dnsnames		string array (comma separated)	List of DNS SANs. If not given, the machine hostname is used.
--ip		string array (comma separated)	List of IP SANs. If not given, no IPs are set.
--labels		string array (comma separated, in label:value form)	Horizon labels to add to the certificate on enrollment.
--team		string	Horizon team to add to the certificate on enrollment
--owner		string	Horizon owner to add to the certificate on enrollment
--contact-email		string	Horizon contact email to add to the certificate on enrollment

Table 46. Enroll ACME options

Parameter	Mandatory	Type	Description
--acme-account	☑ (if using ACME)	string	The identifier (email) of the ACME account to use

Parameter	Mandatory	Type	Description
--http-01-port		int	The http 01 port on which to listen
--dns-01-provider		string	DNS provider script to use for ACME enrollments. ACME.sh based on Linux, Posh-ACME based on Windows
--dns-01-propagation-wait		duration	See DNS-01 propagation section.
--dns-01-timeout		duration	See DNS-01 propagation section.
--eab-kid		string	Kid for External Account Binding
--eab-key		string	Key for External Account Binding

Table 47. Enroll ACME External options

Parameter	Mandatory	Type	Description
--standalone		boolean	Use built in http server
--local		boolean	Use an existing http server
--document-root		string	Path of the document root where to put the well-known folder for the challenge

Table 48. Enroll generic options

Parameter	Mandatory	Type	Description
--destination-folder		string (path to folder)	Folder to write the new certificates to. Default to `/opt/horizon/var/generic` on linux and `C:\ProgramData\EverTrust\Horizon\Generic` on windows.
--pfx		string	Name of the PKCS#12 file to write the enrolled certificate to.
--pfx-pwd		string	Password of the PKCS#12 file to write the enrolled certificate to.
--pfx-strength		string	Encryption level for the generated PKCS#12. One of `weak`, `average`, `strong`.
--pfx-alias		string	Alias of the certificate in the PKCS#12 file.

Parameter	Mandatory	Type	Description
--jks		string	Name of the JKS file to write the enrolled certificate to.
--jks-pwd		string	Password of the JKS file to write the enrolled certificate to.
--jks-alias		string	Alias of the JKS file to write the enrolled certificate to.
--jks-alias-pwd		string	Password of the JKS alias entry. If not provided, the JKS password is used to protect the alias entry.
--kdb		string	Name of file to write the KDB output to.
--kdb-pwd	☑ (if using KDB)	string	Password for the KDB output.
--kdb-alias	☑ (if using KDB)	string	Alias for the KDB output.
--cert		string	Name of the file to write the enrolled certificate to in PEM format.
--key		string	Name of the file to write the enrolled key to.
--der		boolean	Save the certificate and key in DER format.
--chain-file		string	Name of the file to write the enrolled certificate chain to.
--chain-order		string	Order of the chain: `root-to-leaf` or `leaf-to-root`.
--chain-key-bundle		string	Name of the chain and key bundle file to write the enrolled chain, certificate and key to.
--win-user-store		boolean	[Windows certificate store] Save the certificate in the windows user store.
--win-computer-store		boolean	[Windows certificate store] Save the certificate in the windows computer store (LocalMachine).
--win-store-use-tpm		boolean	[Windows certificate store] Use the `Microsoft Platform Crypto Provider` for certificate store storage.

Parameter	Mandatory	Type	Description
--win-store-use-legacy		boolean	[Windows certificate store] Use the `Microsoft Platform Crypto Provider` for certificate store storage.
--win-store-set-exportable		boolean	[Windows certificate store] Set the private key as exportable from the certificate store.

Storage output for generic must choose between pfx, certificate, jks or windows store output.

Table 49. Enroll windows options

Parameter	Mandatory	Type	Description
--win-usages		string array (comma separated)	Windows usages for this certificate. See windows target.

Examples

Use the interactive mode

```
horizon-cli automate enroll --prompt
```

Enroll a new certificate in the default generic folder

```
horizon-cli automate enroll --automation-policy=<automation policy> --target=generic
--cert cert.pem --key key.pem --chain-file chain.pem
```

Add https or control a nginx service

```
horizon-cli automate enroll --automation-policy=<automation policy> --target=nginx
```

Add https to a nginx service without interaction and with custom SAN values

```
horizon-cli automate enroll --automation-policy=<automation policy> --target=nginx
--no-interactive --dnsnames="nginx.test,*.test" --ip "1.1.1.1"
```

Init

The `init` command can setup a certificate and empty https configuration from scratch.

The `init` command can only be used on webservers where no certificate is configured in order to avoid conflicts. See the other commands to match your use case.

Parameters

Table 50. Init general parameters

Parameter	Mandatory	Type	Description
--automation-policy	☑	string	The automation policy to link the certificate to.
--challenge		string array (comma separated)	See challenge section.
--script		string (path to file)	See script section.
--auto-renew		boolean	Configures a status check every 6 hours.
--request-challenge		boolean	See challenge section.
--challenge-routine-period		duration	Period of execution of the periodic task.
--user		string	Name of the user to impersonate while running the request challenge periodically.
--discovery-campaign		string	Also add discovery info on the enrolled certificate on the campaign passed.
--analyze-only		boolean	Instead of executing the enrollments, only displays a summary of possible actions.
--no-interactive		boolean	Disables all interactive inputs. All parameters must then be given using cli flags.
--prompt		boolean	Enable all interactive inputs. All parameters will now be asked for, except ones given using cli flags.

Table 51. Init webServer configuration

Parameter	Mandatory	Type	Description
--target		string array (comma separated)	List of services to target. If not given, all available services are targeted.

Parameter	Mandatory	Type	Description
--no-install		boolean	Disables installation.
--config-folder		string (path to folder)	Explicitly point your webserver configuration folder.
--port		integer	When initializing an empty https configuration on a blank server, choose on which port to listen for https (defaults to the standard https port for the webserver *443* or *8443*)
--keystore-password		string	Password of the webserver's keystore if it cannot be deduced from configuration

Table 52. Init certificate configuration

Parameter	Mandatory	Type	Description
--dn		DN as string, DN elements comma separated.	Requested subject DN. If no CN is specified here, it will be deduced from requested SANs
--dnsnames		string array (comma separated)	List of DNS SANs. If not given, the machine hostname is used.
--ip		string array (comma separated)	List of IP SANs. If not given, no IPs are set.
--labels		string array (comma separated, in label:value form)	Horizon labels to add to the certificate on enrollment.
--team		string	Horizon team to add to the certificate on enrollment
--owner		string	Horizon owner to add to the certificate on enrollment
--contact-email		string	Horizon contact email to add to the certificate on enrollment

Table 53. Init ACME options

Parameter	Mandatory	Type	Description
--acme-account	☑ (if using ACME)	string	The identifier (email) of the ACME account to use
--http-01-port		int	The http 01 port on which to listen

Parameter	Mandatory	Type	Description
--dns-01-provider		string	DNS provider script to use for ACME enrollments. ACME.sh based on Linux, Posh-ACME based on Windows
--dns-01-propagation-wait		duration	See DNS-01 propagation section.
--dns-01-timeout		duration	See DNS-01 propagation section.
--eab-kid		string	Kid for External Account Binding
--eab-key		string	Key for External Account Binding

Table 54. Init ACME External options

Parameter	Mandatory	Type	Description
--standalone		boolean	Use built in http server
--local		boolean	Use an existing http server
--document-root		string	Path of the document root where to put the well-known folder for the challenge

Table 55. Init generic options

Parameter	Mandatory	Type	Description
--destination-folder		string (path to folder)	Folder to write the new certificates to. Default to `/opt/horizon/var/generic` on linux and `C:\ProgramData\EverTrust\Horizon\Generic` on windows.
--pfx		string	Name of the PKCS#12 file to write the enrolled certificate to.
--pfx-pwd		string	Password of the PKCS#12 file to write the enrolled certificate to.
--pfx-strength		string	Encryption level for the generated PKCS#12. One of `weak`, `average`, `strong`.
--pfx-alias		string	Alias of the certificate in the PKCS#12 file.
--jks		string	Name of the JKS file to write the enrolled certificate to.

Parameter	Mandatory	Type	Description
--jks-pwd		string	Password of the JKS file to write the enrolled certificate to.
--jks-alias		string	Alias of the JKS file to write the enrolled certificate to.
--jks-alias-pwd		string	Password of the JKS alias entry. If not provided, the JKS password is used to protect the alias entry.
--kdb		string	Name of file to write the KDB output to.
--kdb-pwd	☑ (if using KDB)	string	Password for the KDB output.
--kdb-alias	☑ (if using KDB)	string	Alias for the KDB output.
--cert		string	Name of the file to write the enrolled certificate to in PEM format.
--key		string	Name of the file to write the enrolled key to.
--der		boolean	Save the certificate and key in DER format.
--chain-file		string	Name of the file to write the enrolled certificate chain to.
--chain-key-bundle		string	Name of the chain and key bundle file to write the enrolled chain, certificate and key to.
--chain-order		string	Order of the chain: `root-to-leaf` or `leaf-to-root`.
--win-user-store		boolean	[Windows certificate store] Save the certificate in the windows user store.
--win-computer-store		boolean	[Windows certificate store] Save the certificate in the windows computer store (LocalMachine).
--win-store-use-tpm		boolean	[Windows certificate store] Use the `Microsoft Platform Crypto Provider` for certificate store storage.
--win-store-use-legacy		boolean	[Windows certificate store] Use the `Microsoft Platform Crypto Provider` for certificate store storage.

Parameter	Mandatory	Type	Description
--win-store-set-exportable		boolean	[Windows certificate store] Set the private key as exportable from the certificate store.

Storage output for generic must choose between pfx, certificate, jks or windows store output.

Table 56. Init windows options

Parameter	Mandatory	Type	Description
--win-usages		string array (comma separated)	Windows usages for this certificate. See windows target.

Examples

Use the interactive mode

```
horizon-cli automate init --prompt
```

Enroll a new certificate in the default generic folder

```
horizon-cli automate init --automation-policy=<automation policy> --target=generic
--cert cert.pem --key key.pem --chain-file chain.pem
```

Add https to a nginx service

```
horizon-cli automate init --automation-policy=<automation policy> --target=nginx
```

Add https to a nginx service without interaction and with custom SAN values

```
horizon-cli automate init --automation-policy=<automation policy> --target=nginx --no
-interactive --dnsnames="nginx.test,*.test" --ip "1.1.1.1"
```

Control

The `control` command can take control of an existing certificate on your machine.

The `control` command can only be used on known and compliant with Horizon certificates. If the certificate needs to be enrolled, see the other commands to

match your use case.

Parameters

Table 57. Control general parameters

Parameter	Mandatory	Type	Description
--automation-policy	☑	string	The automation policy to link the certificate to.
--challenge		string array (comma separated)	See challenge section.
--script		string (path to file)	See script section.
--auto-renew		boolean	Configures a status check every 6 hours.
--discovery-campaign		string	Also add discovery info on the enrolled certificate on the campaign passed.
--analyze-only		boolean	Instead of executing the control, only displays a summary of possible actions.
--no-interactive		boolean	Disables all interactive inputs. All parameters must then be given using cli flags.
--prompt		boolean	Enable all interactive inputs. All parameters will now be asked for, except ones given using cli flags.

Table 58. Control webServer configuration

Parameter	Mandatory	Type	Description
--target		string array (comma separated)	List of services to target. If not given, all available services are targeted.
--no-install		boolean	Disables installation.
--config-folder		string (path to folder)	Explicitly point your webserver configuration folder.
--keystore-password		string	Password of the webserver's keystore if it cannot be deduced from configuration

Table 59. Control certificate configuration

Parameter	Mandatory	Type	Description
--labels		string array (comma separated, in label:value form)	Horizon labels to add to the certificate on enrollment.
--team		string	Horizon team to add to the certificate on enrollment
--owner		string	Horizon owner to add to the certificate on enrollment
--contact-email		string	Horizon contact email to add to the certificate on enrollment

Table 60. Control ACME options

Parameter	Mandatory	Type	Description
--acme-account	☑ (if using ACME)	string	The identifier (email) of the ACME account to use when renewing
--http-01-port		int	The http 01 port on which to listen
--dns-01-provider		string	DNS provider script to use for ACME enrollments. ACME.sh based on Linux, Posh-ACME based on Windows
--dns-01-propagation-wait		duration	See DNS-01 propagation section.
--dns-01-timeout		duration	See DNS-01 propagation section.

For generic control, it is as of now disabled outside the default folder (`/opt/horizon/var/genric` or `C:\ProgramData\EverTrust\Horizon\Generic`) to ensure no service interruption. Use post enrollment scripts to copy the certificate.

Table 61. Control generic options

Parameter	Mandatory	Type	Description
--pfx		string	Name of the PKCS#12 file to control.
--pfx-pwd		string	Password of the PKCS#12 file to control.
--pfx-strength		string	Encryption level for the generated PKCS#12. One of `weak`, `average`, `strong`.
--pfx-alias		string	Alias of the certificate in the PKCS#12 file.

Parameter	Mandatory	Type	Description
--jks		string	Name of the JKS file to control.
--jks-pwd		string	Password of the JKS file to control.
--jks-alias		string	Alias of the JKS file to control.
--jks-alias-pwd		string	Password of the JKS alias entry. If not provided, the JKS password is used to protect the alias entry.
--cert		string	Name of the certificate file to control.
--key		string	Name of the key file to control.
--der		boolean	If true, the certificate and key to control are in DER format.
--chain-file		string	Name of the file to write the enrolled certificate chain to.
--chain-order		string	Order of the chain: `root-to-leaf` or `leaf-to-root`.
--chain-key-bundle		string	Name of the chain and key bundle file to control.
--win-user-store		boolean	Control the certificate in the windows user store.
--win-computer-store		boolean	Control the certificate in the windows machine store.
--win-thumbprint		boolean	Thumbprint of the certificate to control. To use with --win-computer-store or --win-user-store

Input for generic must choose between pfx, certificate, jks or windows store.

Table 62. Control windows options

Parameter	Mandatory	Type	Description
--win-usages		string array (comma separated)	Windows usages for this certificate. See windows target.

Examples

Use the interactive mode

```
horizon-cli automate control --prompt
```

Control a certificate in the default generic folder

```
horizon-cli automate control --automation-policy=<automation policy> --target=generic
--cert cert.pem --key key.pem --chain-file chain.pem
```

Control a certificate on an already configured nginx

```
horizon-cli automate control --automation-policy=<automation policy> --target=nginx
```

Modify

The `modify` command allows to select a managed certificate and reenroll it, modifying its sans.

Automation policy cannot be modified

By default, the certificates to modify are selected interactively. They can also be selected with `--ids` or `--serials`, which skips the selection prompt. With `--no-interactive`, no input is prompted for: `--ids` or `--serials` is then mandatory, and `--reenroll` is required to apply `--dnsnames` and `--ip`.

Parameters

Table 63. Modify general parameters

Parameter	Mandatory	Type	Description
--challenge		string array (comma separated)	See challenge section.
--script		string (path to file)	See script section.
--request-challenge		boolean	See challenge section.
--challenge-routine-period		duration	Period of execution of the periodic task.
--user		string	Name of the user to impersonate while running the request challenge periodically.
--discovery-campaign		string	Also add discovery info on the enrolled certificate on the campaign passed.
--no-interactive		boolean	Do not prompt for user input. Requires `--ids` or `--serials`.
--reenroll		boolean	Re-enroll the modified certificates without asking. Required to apply `--dnsnames` and `--ip` in non-interactive mode.

Parameter	Mandatory	Type	Description
--prompt		boolean	Deprecated, has no effect.

Table 64. Modify selection parameters

Parameter	Mandatory	Type	Description
--ids		string array (comma separated)	Identifiers of the services whose certificate to modify. Supports glob syntax (ex: `apache-*:443`). Cannot be used with `--serials`.
--serials		string array (comma separated)	Serial numbers of the certificates to modify. Cannot be used with `--ids`.

Table 65. Modify webServer configuration

Parameter	Mandatory	Type	Description
--target		string array (comma separated)	List of services to target. If not given, all available services are targeted.
--no-reload		boolean	Do not reload webservers after certificate enrollment.

Table 66. Modify certificate configuration

Parameter	Mandatory	Type	Description
--dn		DN as string, DN elements comma separated.	Requested subject DN. If no CN is specified here, it will be deduced from requested SANs. Replaces the deprecated `--c`, `--o` and `--ou` options, and cannot be used with them.
--dnsnames		string array (comma separated)	List of DNS SANs. If not given, the machine hostname is used.
--ip		string array (comma separated)	List of IP SANs. If not given, no IPs are set.
--labels		string array (comma separated, in label:value form)	Horizon labels to add to the certificate on enrollment.
--team		string	Horizon team to add to the certificate on enrollment

Parameter	Mandatory	Type	Description
--owner		string	Horizon owner to add to the certificate on enrollment
--contact-email		string	Horizon contact email to add to the certificate on enrollment

Table 67. Modify ACME options

Parameter	Mandatory	Type	Description
--acme-account	☑ (if using ACME)	string	The identifier (email) of the ACME account to use.

Table 68. Modify Generic options

Parameter	Mandatory	Type	Description
--chain-order		string	Order of the chain: `root-to-leaf` or `leaf-to-root`.

Examples

Use the interactive mode

```
horizon-cli automate modify
```

Set sans for the generic certificates

```
horizon-cli automate modify --target=generic --dnsnames="nginx.test,*.test" --ip
"1.1.1.1"
```

Set sans for the apache certificates without interaction

```
horizon-cli automate modify --no-interactive --reenroll --ids="apache-*:443"
--dnsnames="www.example.com" --ip "192.0.2.10"
```

Edit

The `edit` command allows to change automation data on managed certificates.

Parameters

Table 69. Edit selection parameters

Parameter	Mandatory	Type	Description
--ids		string array (comma separated)	Identifiers of the services to edit. Supports glob syntax
--serials		string array (comma separated)	Serial numbers of the certificates to edit all its services

Table 70. Edit state parameters

Parameter	Mandatory	Type	Description
--no-install		boolean	Change the no-install value for targeted services

Backup

Each time a file is replaced by the Horizon Client, the old file is backed up. The backup files are stored in the `cert/backup/<HASH(BACKUPED FILE PATH)>` directory relative to the Horizon Client data folder (`/opt/horizon` on unix and `C:/ProgramData/EverTrust/Horizon` on Windows), as `filename_n.ext` where `n` is the number of the backup. Thus, the `filename_0.ext` is the original version, before any intervention of the Horizon Client.

Internal Database operations

The internal database is used to store the details of the certificates that are discovered and enrolled by the automation module. You can list them using the `automate list` command, and delete them using the `automate delete` command. Certificates are indexed by their bindings, which are the combination of all the services along with the hostnames and ports that use the certificate. For example, if you have a certificate that is used by *Apache* for all hosts on the port 443, it's "id" in the local database will be `apache-*:443`.

You can choose the output format of the `automate list` command. By default, it outputs a string, but you can use the `--json` option to output a JSON object. example:

```
horizon-cli automate list --json | jq
```

The `automate remove --ids <id1>, ⋯, <idn>` command erases certificates from the local database. This command will not remove the certificate files from your machine, only remove it from the "managed certificates" local database. This way, the client will not check its status at each routine execution anymore.

You can use the `automate remove --ids all` command to remove all certificates from the local database.

To select certificates to remove, another option is to use the `--serials` option with the serials of the certificates to remove.

The `--restore` option of the `automate remove` command can be used to restore a certificate from a backup file. The backup file to be restored will always be the older one, in most cases the `filename_0.ext`, that is, the original file before any tampering by the Horizon Client. For certificates stored in the Windows store, the store thumbprints will be stored in a file corresponding to the server type, like `iisbackups`.

The `--revoke` option of the `automate remove` command can be used to revoke the certificate after being removed from the "managed certificate" local database. You can add the option `--reason` to specify the revocation reason of the certificate.

System commands

`horizon-cli` uses system commands to manage webservers. In order to use them with `sudo`, the `SUDO_COMMANDS` configuration is available and the commands that might be executed on each flow are available below (linux only):

apache

- `systemctl`
- `apachectl`
- `a2enmod`
- `a2dismod`
- `which`
- `whereis`
- `ln`
- `bash` for script execution

nginx

- `systemctl`
- `nginx`
- `ln`
- `bash` for script execution

tomcat

- `systemctl`
- `bash` for script execution

lighttpd

- `systemctl`
- `ln`
- `bash` for script execution

wildfly

- `systemctl`
- `bash` for script execution

generic

- `bash` for script execution

haproxy

- `systemctl`
- `haproxy`
- `bash` for script execution

Routine

Table 71. Routine arguments

Argument	Mandatory	Type	Description
automation-policies		string array (comma separated)	List of automation policies to check for certificate renewal. If not given, all policies are checked.

Its usage is described in the renewal section.

Management commands

Periodic task

To run the routine at specified intervals, the periodic task command can create scheduled execution on windows (scheduled task) and linux (cron).

Two commands are available, to remove and add the task.

Parameters

Table 72. Create periodic task arguments

Argument	Mandatory	Type	Description
Period		duration	Run the routine command every `period` from midnight UTC. Must be superior to 1h and inferior to 24h. Defaults to 6h.

Table 73. Create periodic task parameters

Parameter	Mandatory	Type	Description
--user		string	**Linux only**: The user to impersonate while running the routine. Defaults to root user.

Examples

Create a periodic task to run routine every 7h

```
horizon-cli automate create-periodic-task 7h
```

Remove the periodic task

```
horizon-cli automate remove-periodic-task
```

List

The `list` command lists the currently managed certificates and various information about them, notably the associated automation policy.

Parameters

Table 74. List parameters

Parameter	Mandatory	Type	Description
--json		string	Output the current state as JSON instead of human readable format.

Examples

Show the currently managed certificates

```
horizon-cli automate list
```

Show the currently managed certificates as JSON

```
horizon-cli automate list --json
```

Remove

The `remove` command erases a certificate binding from the horizon-cli managed certificates.

Parameters

Table 75. Remove parameters

Parameter	Mandatory	Type	Description
--ids		string list	Ids of the services to remove. See automate list to get the id. Use `all` to erase every managed certificate.
--serials		string list	Serials of the certificates to remove. All linked services will be removed. See automate list to get the serial.
--restore		boolean	Enables restoration of the oldest backup for the removed certificate.
--revoke		boolean	Enables the revocation of the removed certificate
--reason		string	Reason for revocation (`unspecified`, `keycompromise`, `cacompromise`, `affiliationchanged`, `superseded`, `cessationofoperation`)

--acme-account	☑ (if revoking ACME certificate)	string	The identifier (email) of the ACME account to use

Examples

Remove all currently managed certificates

```
horizon-cli automate remove --ids all
```

Remove the certificate linked to nginx https and restore the original one

```
horizon-cli automate remove --ids nginx-*:443 --restore
```

Remove and revoke the certificate linked to nginx https one

```
horizon-cli automate remove --ids nginx-*:443 --revoke --reason affiliationchanged
```

Examples

Enroll certificates used by nginx and apache

```
horizon-cli automate enroll \
  --target=nginx,apache \
  --automation-policy=<POLICY_NAME>
```

Enroll certificates using the generic target

```
horizon-cli automate enroll \
  --target=generic \
  --automation-policy=<POLICY_NAME>
```

Use the interactive mode

```
horizon-cli automate enroll --prompt
```

Check if the previously enrolled certificates need to be renewed

```
horizon-cli automate routine
```

Get the DN of all the certificates enrolled by the automation module

```
horizon-cli automate list --json | jq -r '.[] | .certificate | .subject'
```

Remove the certificate used by nginx on the port 443 from the automatically renewed certificates

```
horizon-cli automate remove --ids nginx-*:443
```

Remove the certificate used by tomcat on the port 8443 from the automatically renewed certificates and restore the original certificate

```
horizon-cli automate remove --restore --ids tomcat-*:8443
```

Chapter 13. Release notes

13.1. Horizon-cli 1.19.0 release notes

Here are the release notes for EverTrust Horizon Client v[object Object], released on 2026-10-29. For the installation and upgrade procedure, please refer to the Installation and Upgrade guide.

New Features

- Netimport now supports Google Certificate Manager. The new `netimport gcm` command imports the certificates of a Google Cloud project into a Horizon discovery campaign. Learn more ...
- WebRA challenge mode is now supported. `webra enroll --challenge` enrolls a certificate with a WebRA challenge and no credentials, and the automation module can enroll on WebRA automation policies configured with the challenge authorization mode. Learn more ...
- ACME enrollments in the automation module now support IP identifiers (RFC 8738). IP SANs given with `--ip` are requested and validated with the `http-01` or `tls-alpn-01` challenge. Requires Horizon 2.11 or above. Learn more ...
- `automate modify` can now run without interaction. The new `--no-interactive`, `--ids`, `--serials` and `--reenroll` options select the certificates to modify and re-enroll them without prompts. Learn more ...
- Tenable.sc import can now read the cumulative database instead of a single scan with the new `--cumulative` option. Learn more ...

Enhancements

- Nameshield netimport: imported certificates now carry the Nameshield certificate identifier in the `nameshield_cert_id` metadata.
- `automate modify` now accepts `--dn`. The `--c`, `--o` and `--ou` options are deprecated in favor of `--dn`, and `--prompt` is deprecated and has no effect.
- Certificate chains going through the eMudhra (emSign) intermediate CAs distrusted by the Go project are now rejected.

Bug Fixes

- Fixed an issue where the `--no-install` option was not honored on two-step renewals in the automation module
- Fixed an issue where the IIS automation did not target the existing HTTPS binding when no certificate was attached to it
- Fixed an issue where the chain file of a KDB keystore was written in the wrong order
- Fixed a panic when the CA chain response was empty or returned a non-200 status
- Fixed a panic in `automate request` when a completed WebRA request had no certificate

Reworked features

[None]

Known Defects

[None]