TLS Certificate Auth Method (API)

This is the API documentation for the Vault TLS Certificate authentication method. For general information about the usage and operation of the TLS Certificate method, please see the Vault TLS Certificate method documentation.

This documentation assumes the TLS Certificate method is mounted at the `/auth/cert` path in Vault. Since it is possible to enable auth methods at any location, please update your API calls accordingly.

Create CA Certificate Role

Sets a CA cert and associated parameters in a role name.

Method	Path	Produces
`POST`	`/auth/cert/certs/:name`	`204 (empty body)`

Parameters

- `name` `(string: <required>)` - The name of the certificate role.
- `certificate` `(string: <required>)` - The PEM-format CA certificate.
- `allowed_names` `(string: "")` - DEPRECATED: Please use the individual `allowed_X_sans` parameters instead. Constrain the Common and Alternative Names in the client certificate with a globbed pattern. Value is a comma-separated list of patterns. Authentication requires at least one Name matching at least one pattern. If not set, defaults to allowing all names.
- `allowed_common_names` `(string: "" or array: [])` - Constrain the Common Names in the client certificate with a globbed pattern. Value is a comma-separated list of patterns. Authentication requires at least one Name matching at least one pattern. If not set, defaults to allowing all names.
- `allowed_dns_sans` `(string: "" or array: [])` - Constrain the Alternative Names in the client certificate with a globbed pattern. Value is a comma-separated list of patterns. Authentication requires at least one DNS matching at least one pattern. If not set, defaults to allowing all dns.
- `allowed_email_sans` `(string: "" or array: [])` - Constrain the Alternative Names in the client certificate with a globbed pattern. Value is a comma-separated list of patterns. Authentication requires at least one Email matching at least one pattern. If not set, defaults to allowing all emails.
- `allowed_uri_sans` `(string: "" or array: [])` - Constrain the Alternative Names in the client certificate with a globbed pattern. Value is a comma-separated list of URI patterns. Authentication requires at least one URI matching at least one pattern. If not set, defaults to allowing all URIs.
- `allowed_organizational_units` `(string: "" or array: [])` - Constrain the Organizational Units (OU) in the client certificate with a globbed pattern. Value is a comma-separated list of OU patterns. Authentication requires at least one OU matching at least one pattern. If not set, defaults to allowing all OUs.
- `required_extensions` `(string: "" or array: [])` - Require specific Custom Extension OIDs to exist and match the pattern. Value is a comma separated string or array of `oid:value`. Expects the extension value to be some type of ASN1 encoded string. All conditions *must* be met. Supports globbing on `value`.
- `policies` `(string: "")` - A comma-separated list of policies to set on tokens issued when authenticating against this CA certificate.
- `display_name` `(string: "")` - The `display_name` to set on tokens issued when authenticating against this CA certificate. If not set, defaults to the name of the role.

- `ttl` `(string: "")` - The TTL of the token, provided in either number of seconds (`3600`) or a time duration (`1h`). If not provided, the token is valid for the the mount or system default TTL time, in that order.
- `max_ttl` `(string: "")` - Duration in either number of seconds (`3600`) or a time duration (`1h`) after which the issued token can no longer be renewed.
- `period` `(string: "")` - Duration in either number of seconds (`3600`) or a time duration (`1h`). If set, the generated token is a periodic token; so long as it is renewed it never expires unless `max_ttl` is also set, but the TTL set on the token at each renewal is fixed to the value specified here. If this value is modified, the token will pick up the new value at its next renewal.
- `bound_cidrs` `(string: "", or list: [])` – If set, restricts usage of the certificates to client IPs falling within the range of the specified CIDR(s).

Sample Payload

```
{
  "certificate": "-----BEGIN CERTIFICATE-----\nMIIEtzCCA5+.......ZRtAfQ6r\nwlW975rYa1ZqEdA=\n-----END CERTIFICATE-----",
  "display_name": "test",
  "bound_cidrs": ["127.0.0.1/32", "128.252.0.0/16"]
}
```

Sample Request

```
$ curl \
    --header "X-Vault-Token: ..." \
    --request POST \
    --data @payload.json
    http://127.0.0.1:8200/v1/auth/cert/certs/test-ca
```

Read CA Certificate Role

Gets information associated with the named role.

Method	Path	Produces
`GET`	`/auth/cert/certs/:name`	`200 application/json`

Parameters

- `name` `(string: <required>)` - The name of the certificate role.

Sample Request

```
$ curl \
    --header "X-Vault-Token: ..." \
    http://127.0.0.1:8200/v1/auth/cert/certs/test-ca
```

Sample Response

```
{
  "lease_id": "",
  "renewable": false,
  "lease_duration": 0,
  "data": {
    "certificate": "-----BEGIN CERTIFICATE-----\nMIIEtzCCA5+.......ZRtAfQ6r\nwlW975rYa1ZqEdA=\n-----END CERTIFICATE-----",
    "display_name": "test",
    "policies": "",
    "allowed_names": "",
    "required_extensions": "",
    "ttl": 2764800,
    "max_ttl": 2764800,
    "period": 0
  },
  "warnings": null,
  "auth": null
}
```

List Certificate Roles

Lists configured certificate names.

Method	Path	Produces
`LIST`	`/auth/cert/certs`	`200 application/json`

Sample Request

~~~
$ curl \
    --header "X-Vault-Token: ..." \
    --request LIST \
    http://127.0.0.1:8200/v1/auth/cert/certs

### Sample Response

```json
{
  "auth": null,
  "warnings": null,
  "wrap_info": null,
  "data": {
    "keys": [
~~~

```
        "cert1",
        "cert2"
      ]
    },
    "lease_duration": 0,
    "renewable": false,
    "lease_id": ""
  }
```

Delete Certificate Role

Deletes the named role and CA cert from the method mount.

Method	Path	Produces
`DELETE`	`/auth/cert/certs/:name`	`204 (empty body)`

Parameters

- `name` `(string: <required>)` - The name of the certificate role.

Sample Request

```
$ curl \
    --header "X-Vault-Token: ..." \
    --request DELETE \
    http://127.0.0.1:8200/v1/auth/cert/certs/cert1
```

Create CRL

Sets a named CRL.

Method	Path	Produces
`POST`	`/auth/cert/crls/:name`	`204 (empty body)`

Parameters

- `name` `(string: <required>)` - The name of the CRL.
- `crl` `(string: <required>)` - The PEM format CRL.

Sample Payload

```
{
  "crl": "-----BEGIN X509 CRL-----\n...\n-----END X509 CRL-----"
}
```

Sample Request

```
$ curl \
    --header "X-Vault-Token: ..." \
    --request POST \
    --date @payload.json \
    http://127.0.0.1:8200/v1/auth/cert/crls/custom-crl
```

Read CRL

Gets information associated with the named CRL (currently, the serial numbers contained within). As the serials can be integers up to an arbitrary size, these are returned as strings.

Method	Path	Produces
`GET`	`/auth/cert/crls/:name`	`200 application/json`

Parameters

- `name` `(string: <required>)` - The name of the CRL.

Sample Request

```
$ curl \
    --header "X-Vault-Token: ..." \
    http://127.0.0.1:8200/v1/auth/cert/crls/custom-crl
```

Sample Response

```
{
  "auth": null,
  "data": {
    "serials": {
      "13": {}
    }
  },
  "lease_duration": 0,
  "lease_id": "",
  "renewable": false,
  "warnings": null
}
```

Delete CRL

Deletes the named CRL from the auth method mount.

Method	Path	Produces
`DELETE`	`/auth/cert/crls/:name`	`204 (empty body)`

Parameters

- `name` `(string: <required>)` - The name of the CRL.

Sample Request

```
$ curl \
    --header "X-Vault-Token: ..." \
    --request DELETE \
    http://127.0.0.1:8200/v1/auth/cert/crls/cert1
```

Configure TLS Certificate Method

Configuration options for the method.

Method	Path	Produces
`POST`	`/auth/cert/config`	`204 (empty body)`

Parameters

- `disable_binding` `(boolean: false)` - If set, during renewal, skips the matching of presented client identity with the client identity used during login.

Sample Payload

```
{
  "disable_binding": true
}
```

Sample Request

```
$ curl \
    --header "X-Vault-Token: ..." \
    --request POST \
    --date @payload.json \
    http://127.0.0.1:8200/v1/auth/cert/certs/cert1
```

Login with TLS Certificate Method

Log in and fetch a token. If there is a valid chain to a CA configured in the method and all role constraints are matched, a token will be issued. If the certificate has DNS SANs in it, each of those will be verified. If Common Name is required to be verified, then it should be a fully qualified DNS domain name and must be duplicated as a DNS SAN (see https://tools.ietf.org/html/rfc6125#section-2.3)

Method	Path	Produces
`POST`	`/auth/cert/login`	`200 application/json`

Parameters

- `name` `(string: "")` - Authenticate against only the named certificate role, returning its policy list if successful. If not set, defaults to trying all certificate roles and returning any one that matches.

Sample Payload

```
{
  "name": "cert1"
}
```

Sample Request

```
$ curl \
    --request POST \
    --date @payload.json \
    http://127.0.0.1:8200/v1/auth/cert/login
```

Sample Response

```
{
  "auth": {
    "client_token": "cf95f87d-f95b-47ff-b1f5-ba7bff850425",
    "policies": [
      "web",
      "stage"
    ],
    "lease_duration": 3600,
    "renewable": true,
  }
}
```