Signing certificates using Vault provided CA, and the CA imported into Vault

This page describes on how to sign the certificates(generated using openssl) with two different approaches. One is with Vault CA(in-built) and the second one is with the CA uploaded into Vault.

Method 1: Signing the certificates with Vault generated Certificate Authority(CA)

Creating CA:

Using Vault CLI commands, Create CA with self-signed CA certificate.

Using CA to get the certificates signed:

Using openssl command, generate user key pair, generate CSR.

Using Vault CLI command, provide CSR and get the certificate signed.

Method 2: Signing the certificates with the CA certificates imported/supplied to Vault

Creating CA:

Using OpenSSL command, generate self signed CA certificate.

Import self signed CA certificate and associated private key to Vault using Vault CLI commands.

Using CA to get the certificates sigend:

Using openssl command, generate user key pair, generate CSR.

Using Vault CLI command, provide CSR and get the certificate signed.

Step 1: Steps to configure, start and unsealing Vault

- Install Vault

  Follow this page to install Vault.

  Once the installation is over, ensure of vault installed properly with below command.

  ```
  vault version   -> this results which version of vault got installed.
  ```

- Mount Vault backend with configuration file

  ```
  cat vault.conf
  backend "file" {
    path = "/tmp/vault/backend"
  }
  listener "tcp" {
  ```

```
    address = "127.0.0.1:8200"

    tls_disable = 1

  }
```

- Starting Vault server

```
vault server -config=vault.conf
```

- Initialize Vault with the new backend.

```
vault init
```

This command will give us list of vault unseal keys and root token. We must record/store them as these details plays significant role in unsealing vault and while making REST API requests to Vault server.

- Unseal the Vault since by default it is sealed.

```
vault unseal <unseal-key1>
```

This command prompts us to enter the unseal keys the number of times "Key Threshold" configured. By default it's value is 3, so it takes three keys to unseal. Repeat the command by providing different keys till "Unseal Progress" equals to "Key Threshold" and till you see "Sealed: false". By default it is true.

You may check vault parameters with the command ***vault status***

Step 2 : Configure environment variables

Ensure of following two environment variables have configured in the machine because Vault make use of them while you're playing with the commands using vault cli.

- ***VAULT_ADDR='http://127.0.0.1:8200'*** *→ **t**his has to be set so that we can use vault CLI; vault CLI commands internally make use of REST API so it looks for VAULT_ADDR.*
- ***VAULT_TOKEN=3803c395-f7fd-8338-03b9-fc67ec728d87*** *→ this indicates the root token which you can see while initializing vault*

Step 3 : Signing the certificates with Vault generated Certificate Authority(CA)

- **a. Creating CA:**

  1. Creating a 'pki' backend "onap-csm-pki ", the name could be anything. Ideally we have single CA installed in a backend.

```
vault mount -path=onap-csm-pki1 pki
```

  2. Check the available backbends, we must find the one what we've created above.

```
vault mounts
```

  3. Configure vault with CA certificate and associated private-key

```
    vault write onap-csm-pki1/root/generate/internal common_name=myvault.com
ttl=87600h

    Ensure of CA cert is created with the command below

    curl $VAULT_ADDR/v1/onap-csm-pki1/ca/pem   → this will result the created cert at
onap-csm-pki1 pki backend.
```

- **b. Creating rule :**

  Create a role with name onap-csm-rule (the name could be anything) with by providing role.json. Here we're using Vault REST API, unlike earlier where we've used Vault CLI.

```
curl --header "X-Vault-Token: 14e24083-402b-2829-106a-df069879563c" --request POST  --da
```

```
role.json

{

 "allow_any_name": true,

 "allow_subdomains": true

 }
```

- **c. Create Certificate Signing Request(CSR) using openssl that we wish to sign using the Vault CA created above:**

  openssl genrsa out onap-csm.key 2048

  openssl req -new -key onap-csm.key -out onap-csm.csr

- **d. Create the payload:**

  Create a json file onap-csm-root-ca-bundle.json (could be of any name) consisting of "csr" attribute having value of the result of csr file created above.

  Note: We must include "\n" in the payload json to indicate that there is a line break, the same is highlighted in the sample payload.

  sample payload json file :

```
{

    "csr" : "-----BEGIN CERTIFICATE REQUEST-----\n

     MIIC9jCCAd4CAQAwgZYxCzAJBgNVBAYTAklOMQ0wCwYDVQQIDARQdW5lMQ4wDAYD
    VQQHDAVLYXJ2ZTEZMBcGA1UECgwQVGVjaE1haGluZHJhIEx0ZDELMAkGA1UECwwC
    SVQxFDASBgNVBAMMC2V4YW1wbGUuY29tMSowKAYJKoZIhvcNAQkBFht2bjAwNDgw
    MjE1QHRlY2htYWhpbmRyYS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
    AoIBAQDfi17aJoSyeUitn7MweAbw3a1VEUHIvEnj/mEBxs9BlT34kDBOG0kaxIZ+
    bomIiQPjMNLx2HD1eqXbqU97clUBLFk3i9kCZeVPFjnhCmxoar9wqJNMsaLJyl4W
    KVzXwe3NxgD7GCEBxds1c3c+GO54fE6Cyrc6NcZIibTPAgUaLT1tIvNPaDX3U7ay
    vdZwiWE8JA3xEZNiz/yXxwSnnEpGv4fZfYuaPplVhddeGG9kYcJX6JK37XR7cNMT
    PV43eFXOdYMqCdCGH3UOCexGQOWIlXPJz2MgCxH2CuXqmf7S1Kfm0FpCwlO9MD4a
    ccUMdWOaSdMOeWKra19GLW0BB8B/AgMBAAGgGjAYBgkqhkiG9w0BCQcxCwwJMTIz
    NDU2Nzg5MA0GCSqGSIb3DQEBCwUAA4IBAQCXCTPMpXL8SYa9OMbrN1dcvFgfS/wy
    lVOGXyU4w9oovFDqbuqFtChrhLohPWNZjQNmOJdbaOvThNeQHnqkd4x6sPsJNCYO
    R+VDriMzkVrBwIPyiI0x5SDo6l3710cT31GDnjdON9Z/XmL4grFxmCFM0fvka+9U
    z96zHvq9LWTBPHReMS5r6i34j+JFlD4m1s56eJ6NzM5r0ok5DhMIttpf98A5BvZN
    7WbZEXthxaGmUWyrFYCzSDg+1Feha7FXv8oSiTsBwvmTPQ1op/FdFewvc+7/ERjR\n
    kq0P3qkxR0TJ4YoDZpFjyf2BSCml4r1eK+Z6118nn5x8Hd9USPZOPLZt\n

     -----END CERTIFICATE REQUEST-----"

 }
```

- **e. Get the certificate signed:**

```
curl --header "X-Vault-Token: 14e24083-402b-2829-106a-df069879563c" --request POST --da
```

Step 4 : Signing the certificates with the CA supplied to Vault.

- ***a. Importing/loading CA into Vault:***

  1. Creating a 'pki' backend "onap-csm-pki2 ", the name could be anything. Ideally we have single CA installed in a backend.

     ```
     vault mount -path=onap-csm-pki2 pki
     ```

2. Create self-signed certificate using openssl

```
openssl genrsa -out onap-csm-root-ca.key 2048
```

```
openssl req -x509 -new -nodes -key myrootca.key -sha256 -days 1024 -out onap-csm-ro
```

3. Create the payload

Create a json file onap-csm-root-ca-bundle.json (could be of any name) consisting of "pem_bundle" attribute having value of the concatenation result of **onap-csm-root-ca.key** & **onap-csm-root-ca.pem** created above.

Note: We must include "\n" in the payload json to indicate that there is a line break, the same is highlighted in the sample payload.

cat onap-csm-ca-bundle.json

{

*"pem_bundle" : "-----BEGIN CERTIFICATE-----**\n***

*MIIEDTCCAvWgAwIBAgIJAKIttigupLHgMA0GCSqGSIb3DQEBCwUAMIGcMQswCQYD
VQQGEwJJTjENMAsGA1UECAwEUHVuZTEOMAwGA1UEBwwFS2FydmUxFjAUBgNVBAoM
DVRlY2ggTWFoaW5kcmExCzAJBgNVBAsMAklUMR0wGwYDVQQDDBR3d3cudGVjaG1h
aGluZHJhLmNvbTEqMCgGCSqGSIb3DQEJARYbdm4wMDQ4MDIxNUB0ZWNobWFoaW5k
cmEuY29tMB4XDTE4MDEwNTExMzg0NFoXDTIwMTAyNTExMzg0NFowgZwxCzAJBgNV
BAYTAklOMQ0wCwYDVQQIDARQdW5lMQ4wDAYDVQQHDAVLYXJ2ZTEWMBQGA1UECgwN
VGVjaCBNYWhpbmRyYTELMAkGA1UECwwCSVQxHTAbBgNVBAMMFHd3dy50ZWNobWFo
aW5kcmEuY29tMSowKAYJKoZIhvcNAQkBFht2bjAwNDgwMjE1QHRlY2htYWhpbmRy
YS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDcfXh+3XEx9hxz
SZHPDt5js/zNvewsg8PjCw3bv7mTV4qJVTxD3k6Bsor+FV5Hy8AR7M4G/7FLOmBr
9E9LnbqcN5pFIWyAP1WYzmsxY7Doa78A0KTNxZmydy2fA5nU93BgJBqbA9zp5Jep
Uw/jKeGV2XPAoPwpywfZpvotoTUmb4C06qZipZVyY4s7+fx9J5E7q2qMwAuK39vn
x8nJS0d65KOwqIaidrmMel/YBzo/0zRCOIlY1G5YpTeaPlDpdxMKOEZ8fAX09p3J
djC923p3tiMl/ZXU22RpMBBTO+pahwPPFTCm22KBs3sRHa98YXEHGRgHrlrf0uY5
20fPfkw5AgMBAAGjUDBOMB0GA1UdDgQWBBTf5ICVmH+IXBmHSXe1pJwvjFAuYTAf
BgNVHSMEGDAWgBTf5ICVmH+IXBmHSXe1pJwvjFAuYTAMBgNVHRMEBTADAQH/MA0G
CSqGSIb3DQEBCwUAA4IBAQADi6M71dDXZXehTM11i07ELkfNZ9gh27iMlxI6aZS0
9fHne4XDSutYU7xtVABcyD9n53n2s2lxx5rUTBuoB3HEllzsQPXAUNWzzgMhx2wo
qmTm+mBx+UCsDb2ntAqKpkHjehaj1ku2+ufWPt6edhzgX9Jwy01XjiykcGnJjLZf
lX7NSA+D5b0l4q1vzJpjZekMLoiIfc7NztNdzVz3+bH56p4MxLb3jdtOZdyVoarq
jKdIyjeETgePQEd0xU1vCCPvwQRNgJUWtDw8RlnzGoOyKjuU1/phzaJFn60bqfK6
OeDDrNlcUSrd6wf41gRTyPWS9MsWcc9TXIlUgMwfn9nA*<b>*\n*</b>

*-----END CERTIFICATE-----**\n***

*-----BEGIN RSA PRIVATE KEY-----**\n***

*MIIEpgIBAAKCAQEA3H14ft1xMfYcc0mRzw7eY7P8zb3sLIPD4wsN27+5k1eKiVU8
Q95OgbKK/hVeR8vAEezOBv+xSzpga/RPS526nDeaRSFsgD9VmM5rMWOw6Gu/ANCk
zcWZsnctnwOZ1PdwYCQamwPc6eSXqVMP4ynhldlzwKD8KcsH2ab6LaE1Jm+AtOqm
YqWVcmOLO/n8fSeRO6tqjMALit/b58fJyUtHeuSjsKiGona5jHpf2Ac6P9M0QjiJ
WNRuWKU3mj5Q6XcTCjhGfHwF9PadyXYwvdt6d7YjJf2V1NtkaTAQUzvqWocDzxUw
pttigbN7ER2vfGFxBxkYB65a39LmOdtHz35MOQIDAQABAoIBAQCmM+o1b0TZTVRq
zuUbOHEIpO8GQ4iYkYaCSZ3brKz9VPq3xMlVu2hgOa6uEntsETkqCd/PxMPnGgz+
sz1mmXHGOd+PBr/b+GHUepywsR30ROvIeH4SIkZWEaIRAEzgDNjnj6+CdCn9IPP1
jggmyzYhl7W6WV9bPZEgTs68wIzo97M3nwmVH7sOnMaHjo7rdcqhddethalkQLIz
/YWZ+NoD7C8GDa9t+nL2wTvRpWCPpX5YFAHsEByJSRXUYo73m/TGsqVYkRICZdR5
IirNV7YJgNMSf14QsoKJDk5YwsyowGVApPYNsa8JvQXfCA0LVGkYSgLxvc4fzOVE*

*DD+r3A9VAoGBAPv6GYuSjGLf9sy+VMTbvnbrFHg5avw9m3Djx/GozJOqh6Xmv0uD
uzKMyppjH8do9OySWaVeB9l83QWzHU4tAZjQlSK4JDxqiobRgXc/lAueS2AYqjAS
0c0S6y3PjA8Flhre+j+t3ocQmqaBvSiepfFXOq+CntLW2o3b5kdPwUH3AoGBAOAC
rOgm6tpu8FJUJEI89mxAqE+fL5FfXNVpGmzdm4J4p84rWesAawDWvJ1eYw5Cm9CQ
Q2mdZarcDa5NLCfQJrUy8z/RBCFEB5gjJ9idPUOH+eX87BWof0e7a0A4vNySmOBv
FKbdjh4UxyRafTg4jRycwbgU4AXTsWmaWKpKEFdPAoGBAM4HFvAKaYNHAPM0BPfZ
fQia+me6+wE4FmrdtFSh4nQzESrTW0KReXTBrb9CoW9ZlDp3B5mxltXvxICujZ2o
KxAaLHbw/Z/wtUe68hLhB1ngmlz+jdk06hq2B1mzxB9cP/nEq/V5YuQo7WqL1nDq
F2EYI8HyGY7nYlhvnwBb8/bNAoGBALC265goO1Ud90+7OO9YED3Ns/k75taTmDRy
uXnwSGFgtWAbKtAMgF0lCZ5Le3EgcrLRW5zRogZrmg9Kqe6uchq3mtVZGhz0Admu
whxLzqybdDROlh9v0RjRbQY4vCR1MUy71FrepOJuGbs/91CGrCtKLjf9n8x495gL
Pq73xnRZAoGBANFJCrRLkjksaajZKmUD7s10nA0USP09j4JNSAYcG0lvYs4tRtOb
Cj9oOb3lt9GVH7tOhTIjE0KQdthO867ot8MoQ4rtNTIbTxA90DEDosiPcNRPLxUX
2vN7Q9NmWTzjZvrWdSnbU3YHEnnLMDKbc7GrUpHgQwrQLV7ftd40kREN\n*

-----END RSA PRIVATE KEY-----"

}

4. Upload the root certificate into the Vault

```
curl --header "X-Vault-Token: 14e24083-402b-2829-106a-df069879563c" --request POST

Ensure of CA cert is created with the command below

  curl $VAULT_ADDR/v1/onap-csm-pki2/ca/pem   → this will result the created cert at
onap-csm-pki1 pki backend.
```

- b. **Creating rule :**

Create a role with name onap-csm-rule (the name could be anything) with by providing role.json. Here we're using Vault REST API, unlike earlier where we've used Vault CLI.

```
curl --header "X-Vault-Token: 14e24083-402b-2829-106a-df069879563c" --request POST  --

 role.json

 {

   "allow_any_name": true,

   "allow_subdomains": true

 }
```

- **c. Create Certificate Signing Request(CSR) using openssl that we wish to sign using the Vault CA created above:**

  openssl genrsa out onap-csm.key 2048

  openssl req -new -key onap-csm.key -out onap-csm.csr

- **d. Create the payload:**

  Create a json file onap-csm-root-ca-bundle.json (could be of any name) consisting of "csr" attribute having value of the result of csr file created above.

  Note: We must include "\n" in the payload json to indicate that there is a line break, the same is highlighted in the sample payload.

  sample payload json file :

  {

  "csr" : "-----BEGIN CERTIFICATE REQUEST-----**\n**

MIIC9jCCAd4CAQAwgZYxCzAJBgNVBAYTAklOMQ0wCwYDVQQIDARQdW5lMQ4wDAYD
VQQHDAVLYXJ2ZTEZMBcGA1UECgwQVGVjaE1haGluZHJhIEx0ZDELMAkGA1UECwwC
SVQxFDASBgNVBAMMC2V4YW1wbGUuY29tMSowKAYJKoZIhvcNAQkBFht2bjAwNDgw
MjE1QHRlY2htYWhpbmRyYS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDfi17aJoSyeUitn7MweAbw3a1VEUHIvEnj/mEBxs9BlT34kDBOG0kaxIZ+
bomIiQPjMNLx2HD1eqXbqU97clUBLFk3i9kCZeVPFjnhCmxoar9wqJNMsaLJyl4W
KVzXwe3NxgD7GCEBxds1c3c+GO54fE6Cyrc6NcZIibTPAgUaLT1tIvNPaDX3U7ay
vdZwiWE8JA3xEZNiz/yXxwSnnEpGv4fZfYuaPplVhddeGG9kYcJX6JK37XR7cNMT
PV43eFXOdYMqCdCGH3UOCexGQOWIlXPJz2MgCxH2CuXqmf7S1Kfm0FpCwlO9MD4a
ccUMdWOaSdMOeWKra19GLW0BB8B/AgMBAAGgGjAYBgkqhkiG9w0BCQcxCwwJMTIz
NDU2Nzg5MA0GCSqGSIb3DQEBCwUAA4IBAQCXCTPMpXL8SYa9OMbrN1dcvFgfS/wy
IVOGXyU4w9oovFDqbuqFtChrhLohPWNZjQNmOJdbaOvThNeQHnqkd4x6sPsJNCYO
R+VDriMzkVrBwIPyiI0x5SDo6l3710cT31GDnjdON9Z/XmL4grFxmCFM0fvka+9U
z96zHvq9LWTBPHReMS5r6i34j+JFlD4m1s56eJ6NzM5r0ok5DhMIttpf98A5BvZN
7WbZEXthxaGmUWyrFYCzSDg+1Feha7FXv8oSiTsBwvmTPQ1op/FdFewvc+7/ERjR\n
kq0P3qkxR0TJ4YoDZpFjyf2BSCml4r1eK+Z6118nn5x8Hd9USPZOPLZt**\n**

-----END CERTIFICATE REQUEST-----"

}

- **e. Get the certificate signed:**

```
curl --header "X-Vault-Token: 14e24083-402b-2829-106a-df069879563c" --request POST --da
```

Troubleshoot:

Following I've kept few scenarios for the anticipated troubles we may face while doing this exercise.

1. {"errors":["certificate request could not be parsed: asn1: structure error: tags don't match (2 vs {class:2 tag:0 length:3 isCompound:true}) {optional:false explicit:false application:false defaultValue:\u003cnil\u003e tag:\u003cnil\u003e stringType:0 timeType:0 set:false omitEmpty:false} int @2"]}

   This is due to the csr I've created is a combination of private key & certificate. Instead, I should have created directly out of key.

2. {"errors":["common name example.com not allowed by this role"]}

   The role created like below (by not feeding json) is throwing this error vault write pki/roles/example-dot-com allowed_domains=example.com allow_subdomains=true max_ttl=72h

3. {"errors":["csr contains no data"]} – following suggestion resolved the issue.

   curl --header "X-Vault-Token: 3803c395-f7fd-8338-03b9-fc67ec728d87" --request POST --data "@payload.json" http://127.0.0.1:8200/v1/techm-pki-final/sign/techm-csm-role

   Adding new lines(\n) at the end of BEGIN CERTIFICATE REQUEST line and before END CERTIFICATE REQUEST line

   I've posted in groups for the solution on the similar query, however you've you have suggested the same. https://groups.google.com/d/msgid/vault-tool/CAORe8GHDfgfZHi77FzO9PEis8xaqgnwKc0xqRyOWvB6B5uRKAA%40mail.gmail.com?utm_medium=email&utm_source=footer

4. {"errors":["no data found"]}

   sudo curl --header "X-Vault-Token:3803c395-f7fd-8338-03b9-fc67ec728d87" --request POST --data "@techmrootcabundle.json" http://127.0.0.1:8200/v1/techm-pki-final/config/ca

This is when I tried above command. Proper placing of \n in the payload (since it is a combination a private key and certificate) resolved this trouble.

5. {"errors":["1 error occurred:\n\n* unsupported path"]}

   curl --header "X-Vault-Token:3803c395-f7fd-8338-03b9-fc67ec728d87" --request POST --data @payload-csr.json http://127.0.0.1:8200/v1/techm-pki/sign

   This is due to missing of rule name in the REST request URI above.

6. {"errors":["certificate could not be PEM-decoded"]}

   curl --header "X-Vault-Token:3803c395-f7fd-8338-03b9-fc67ec728d87" --request POST --data @payload.json http://127.0.0.1:8200/v1/techm-pki/root/sign-self-issued

   This is due to wrong request uri used (sign-self-issued) instead of sign.

7. vault auth 14e24083-402b-2829-106a-df069879563c

==> WARNING: VAULT_TOKEN environment variable set!

The environment variable takes precedence over the value set by the auth command. Either update the value of the environment variable or unset it to use the new token.

Error validating token: Error making API request.

URL: GET http://127.0.0.1:8200/v1/auth/token/lookup-self Code: 403. Errors:

* permission denied

This you may encounter even when you set VAULT_TOKEN with the right token value and passing the right token value as an argument to the "vault auth" command. It's because the vault unable to read env file(/etc/environment) as it is with root permissions.

Execute the "vault auth" with sudo permissions would resolve this issue.

```
sudo vault auth 14e24083-402b-2829-106a-df069879563c
```

8. {"errors":["cannot satisfy request, as TTL is beyond the expiration of the CA certificate"]}

curl --header "X-Vault-Token: 14e24083-402b-2829-106a-df069879563c" --request POST --data "@payload.json" $VAULT_ADDR/v1/onap-csm-pki1/sign/onap-csm-rule

```
vault mount-tune -max-lease-ttl=87600h onap-csm-pki1   -> Setting this immediately after cr
```

```
Maybe because TTL declared while creating root CA is mismatching with backend.
```

9. Every time we restart Vault server it gets sealed by default. We must unseal it to perform operations on it.

No labels