Vault: PKI Made Easy

Disclamer

Well, not quite **PKI Made Easy**, but definitely a bit more **fun**!
I've done all this work on OSX, but I believe the Linux setup should be very similar.
Vault 0.3 was the version used.

Containerize all the things

Last week I was tinkering with Docker and wanted to get Hashicorp Vault running on a container, this w
learning more about Vault.

The Docker stuff went pretty well and you have available a public container to prove it, check it out at:

hashicorp-vault on a container

Regarding the plan, it worked flawlessly and I've got really interested in the application.

So, what's Vault?

*Vault secures, stores, and tightly controls access to tokens, passwords, certificates, API keys, and other secr
handles leasing, key revocation, key rolling, and auditing. Vault presents a unified API to access multiple ba
databases, raw key/value, and more. (source)*

I'm not going into depth about how the application works and all the features it provides, firstly becaus
secondly the documentation does a very good job on that. Instead I'll talk about what I've learned rega
and usage.

These are the points I'll cover:

- Install Vault
- Configure Vault
- Generate a Root Certification Authority (CA) and Intermediate CA
- Create a PKI backend
- Configure the PKI backend
- Issue a couple of server certificates
- Issue a Certificate Revocation List (CRL) on Vault
- Revoke a certificate
- Vault using TLS

Setup

Create the following `vault.conf` file:

```
backend "file" {
  path = "/tmp/vault/backend"
}

listener "tcp" {
  address = "127.0.0.1:8200"
  tls_disable = 1
  #tls_cert_file = "/tmp/vault/localhost.pem"
  #tls_key_file = "/tmp/vault/localhost.key"
}

disable_mlock = true
```

vault.conf hosted with ♡ by GitHub

Create and run the following setup script on the same path as the vault.conf file:

```
#!/usr/bin/env bash

VERSION="0.3.1"
PLATFORM="darwin"
#PLATFORM="linux"
ARCH="amd64"
#ARCH="386"

wget https://releases.hashicorp.com/vault/"$VERSION"/vault_"$VERSION"_"$PLATFORM"_"$ARCH".zip -O /tmp/
unzip -o /tmp/vault_"$VERSION"_"$PLATFORM"_"$ARCH".zip -d /usr/local/bin/ && rm /tmp/vault_"$VERSION"_

if [[ ! -d /tmp/vault ]]; then
  mkdir /tmp/vault
  cp vault.conf /tmp/vault
fi

export VAULT_ADDR=http://127.0.0.1:8200

vault server -config="/tmp/vault/vault.conf" &
```

setup.sh hosted with ♡ by GitHub

You now should have a running instance of Vault using the `/tmp/vault/vault.conf` configuration.

Initialize Vault

```
cd /tmp/vault

vault init > credentials.txt

# check if initialized
curl http://127.0.0.1:8200/v1/sys/init

# keep your credentials safe
cat credentials.txt
```

Unseal Vault

Vault is protected by M-of-N so you'll need to run the unseal command 3 times using a different key ea

The M of N feature provides a means by which organizations employing cryptographic modules for sensitive person control over access to the cryptographic module. (source)

```
vault unseal
```

Export the Root Token

This will authenticate your vault client against the Vault server.

```
export VAULT_TOKEN=use-your-generated-root-token
```

Check the current mount points

```
vault mounts
```

Mount the PKI backend

```
vault mount pki
vault mounts
vault path-help pki
```

Get your hands on a CA certificate

You'll need a CA for the next steps. Don't have one?
Here you go (thank me later):

dummy_ca

You should never use a Root CA to issue client/server certificates, if it's compromised you're screwed! Instea and if that one it's compromised just revoke it and issue a new one, keeping the Root CA offline.

With your certificates generated, now build a certificate bundle with the Intermediate CA certificate an

```
export DUMMY_CA=/PATH/TO/dummy_ca

cat $DUMMY_CA/pki/intermediate/certs/intermediate.pem > \
  /tmp/vault/ca_bundle.pem

# vault does not accept encrypted keys
openssl rsa -in $DUMMY_CA/pki/intermediate/private/intermediate.key >> \
  /tmp/vault/ca_bundle.pem
```

Configure the PKI backend

Carefully read the documentation regarding the API endpoints `/pki/config/` , `/pki/roles` and `/p`

```
vault write pki/config/ca pem_bundle="@/tmp/vault/ca_bundle.pem"

vault write pki/roles/test-dot-local allow_any_name="true" \
  allow_subdomains="true" allow_ip_sans="true" max_ttl="420h" \
  allow_localhost="true" allow_ip_sans="true"

vault write pki/issue/test-dot-local common_name=localhost \
  alt_names="vault.test.local,*.vault.test.local" \
  ip_sans="127.0.0.1,192.168.1.77" > /tmp/vault/localhost.certs

vault write pki/issue/test-dot-local \
  common_name=sheep.test.local > /tmp/vault/sheep.certs
```

Split the `localhost.certs` into a separated key and certificate files:

- `localhost.pem`
- `localhost.key`

Split the `sheep.certs` into a separated key and certificate files:

- `sheep.pem`
- `sheep.key`

Test the CRL

This shouldn't return any revoked certificates yet.

```
curl -v http://127.0.0.1:8200/v1/pki/crl/pem
*   Trying 127.0.0.1...
* Connected to 127.0.0.1 (127.0.0.1) port 8200 (#0)
> GET /v1/pki/crl/pem HTTP/1.1
> Host: 127.0.0.1:8200
> User-Agent: curl/7.43.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/pkix-crl
< Date: Sun, 15 Nov 2015 12:14:40 GMT
< Content-Length: 0
<
* Connection #0 to host 127.0.0.1 left intact
```

Revoking a certicate

To revoke a certificate you first need its Serial Number.

```
export SHEEP_SN=$(openssl x509 -in /tmp/vault/sheep.pem -text | \
  grep -A1 "Serial Number" | grep -v "Serial Number" | \
  awk {'print $1'})

curl -v -X POST http://127.0.0.1:8200/v1/pki/revoke \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -d '{"serial_number":"'$SHEEP_SN'"}'
```

Test the CRL

```
curl -v http://127.0.0.1:8200/v1/pki/crl/pem > \
  /tmp/vault/crl.pem

openssl crl -inform PEM -in /tmp/vault/crl.pem -text
```

You should see the revoked Serial Number.

Vault with TLS

This bit took me quite a while to figure out.

The documentation doesn't mention how to do it. The Vault server doesn't send the Intermediate CA c
the vault client, this way you can't just trust the Root CA, you'll need to trust the Intermediate one…

I even tried providing a ca_bundle with the Root CA certificate in it, but no luck. Then there was was th
truststore to the vault client…

```
# enable the truststore
export VAULT_CAPATH=$DUMMY_CA/pki/intermediate/certs/intermediate.pem
```

Uncomment the lines `tls_*_file` and comment out `tls_disable` on `vault.conf`

```
pkill vault

vault server -config="/tmp/vault/vault.conf" &

export VAULT_ADDR=https://127.0.0.1:8200

vault unseal
```

If it doesn't give you a TLS error, you're golden! You can check the certificate the server is using and th

```
openssl s_client -showcerts -connect 127.0.0.1:8200
```

This blog post only scratches the surface of what Vault is capable of. I'm currently looking into High Av
backends to try out, but I hope I've piqued your curiosity.