Konrad Scherer

Hashicorp Vault based PKI

Introduction

One of the trends I have noticed is that open source tools encrypt network connections by default. Some tools like Puppet even make it impossible to disable TLS encryption and provide tooling to build an internal Certificate Authority. Docker requires overrides for any registry that does not have a verified TLS cert. Many tools also generate self signed certs which Firefox and Chrome always require manual overrides to use.

The solution is to have an internal Certificate Authority with the root CA as part of the trusted store. This internal CA can then be used to generate certs which will be trusted. But there are always complications. Many programs do not use the OS trusted store and require extra configuration to add trusted certs. For example Java applications require several steps to generate a new trusted store file and configuration to make that available to the application. Docker has a special directory to place trusted certs for registries.

Options

There are many CA solutions available: OpenCA, CertStrap, CFSSL, Lemur and many others. As I looked through all these programs a couple things kept bugging me. Creating certs is easy, revocation is where it gets really messy. The critical question is how to handle revocation in a sensible way. How can the system recover from a root CA compromise? Once I started reading about CRLs and OSCP and cert stapling, I got really discouraged. That is why I was intrigued by Hashicorp Vault and its PKI backend.

Vault

Vault is a tool for managing secrets of all kinds, including tokens, passwords and private TLS keys. It is quite complex and the CLI is non obvious. It supports backends for Authentication, Secret Storage and Auditing. It has a comprehensive access control language and a generic wrapper concept that makes it possible to pass secrets without revealing secrets to the middle man.

Vault solves the revocation and CA compromise problem by making it unnecessary. It provides a secure audited out of band channel for distributing secrets like certs which enables very short lived certs and secure automated reissuing of certs.

Vault PKI

That is the theory, so I decided to try it in practice by creating a CA and some certs.

1) Start Vault server, initialize, unseal and authenticate as root:

```
> vault server -config config.hcl
> export VAULT_ADDR='http://127.0.0.1:8200'
> vault init -key-shares=1 -key-threshold=1
Unseal Key 1: LbOw129fyB3OAzZvxq9RMQefNH8fFm7twS3wlg5Zv2o=
Initial Root Token: d9e9d69b-5d49-e753-3ef2-e6b36c0fb45a
> vault unseal LbOw129fyB3OAzZvxq9RMQefNH8fFm7twS3wlg5Zv2o=
> vault auth
Token (will be hidden): d9e9d69b-5d49-e753-3ef2-e6b36c0fb45a
Successfully authenticated! You are now logged in.
```

Of course this is for development only. A production deployment would use more shares and higher threshold. The unseal keys should be encrypted using gpg. Note that the root token can be changed.

2) Create self signed Cert with 10 year expiration

```
> vault mount -path=wrlinux -description="WRLinux Root CA" -max-lease-ttl=87600h pki
Successfully mounted 'pki' at wrlinux'!
> vault write wrlinux/root/generate/internal common_name="WRlinux Root CA" \
    ttl=87600h key_bits=4096 exclude_cn_from_sans=true
certificate     -----BEGIN CERTIFICATE-----
MIIFBDCCAuygAwIBAgIUMt8NYFtqaYk8Q1OUfdOWuPjXI0IwDQYJKoZIhvcNAQEL
...
serial_number   32:df:0d:60:5b:6a:69:89:3c:43:53:94:7d:d3:96:b8:f8:d7:23:42
> curl -s http://localhost:8200/v1/wrlinux/ca/pem | openssl x509 -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            32:df:0d:60:5b:6a:69:89:3c:43:53:94:7d:d3:96:b8:f8:d7:23:42
...
```

Note that this is the only time private cert is exposed.

3) Keep root CA offline and create second vault for intermediate CA

Create CSR for Intermediate CA

```
> vault mount -path=lpd -description="LPD Intermediate CA" -max-lease-ttl=26280h pki
> vault write lpd/intermediate/generate/internal common_name="LPD Intermediate CA" \
ttl=26280h key_bits=4096 exclude_cn_from_sans=true
csr     -----BEGIN CERTIFICATE REQUEST-----
MIIEYzCCAksCAQAwHjEcMBoGA1UEAxMTTFBEIEludGVybWVkaWF0ZSBDQTCCAiIw
...
```

4) Sign CSR and import Certificate

Note: Intermediate private key never leaves Vault

```
> vault write wrlinux/root/sign-intermediate csr=@lpd.csr \
common_name="LPD Intermediate CA" ttl=8760h
Key             Value
---             -----
certificate     -----BEGIN CERTIFICATE-----
MIIFSzCCAzOgAwIBAgIUAY8RmTDEzwbkUQ0smevPPIPXOkYwDQYJKoZIhvcNAQEL
...
-----END CERTIFICATE-----
```

```
expiration      1523021374
issuing_ca      -----BEGIN CERTIFICATE-----
MIIFBDCCAuygAwIBAgIUMt8NYFtqaYk8Q1OUfdOWuPjXI0IwDQYJKoZIhvcNAQEL
...
-----END CERTIFICATE-----
serial_number   01:8f:11:99:30:c4:cf:06:e4:51:0d:2c:99:eb:cf:3c:83:d7:3a:46
> vault write lpd/intermediate/set-signed certificate=@lpd.crt
Success! Data written to: lpd/intermediate/set-signed
```

5) Create Role and generate Certificate

Vault uses roles to setup cert creation rules.

```
> vault write lpd/roles/hosts key_bits=2048 \
max_ttl=8760h allowed_domains=wrs.com allow_subdomains=true \
organization='Wind River' ou=WRLinux
Success! Data written to: lpd/roles/hosts
> vault write lpd/issue/hosts common_name="yow-kscherer-l1.wrs.com" \
ttl=720h
private_key             -----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAvxHQzyEjc13djntQfCo1ncpwU18a8c8iI4OdaOSQV72zbHf2
...
-----END RSA PRIVATE KEY-----
```

6) Final Steps

- Import root CA cert into trusted store
- Create Policy to limit role access to cert creation
- Use program like vault-pki-client to automate cert regeneration
- Audit that certs are only created at expected times
- Automate cert regeneration

Conclusion

Once this is setup, Heartbleed is a non event! As well as a PKI, i can also use Vault to manage other secrets as well.

Pages

- About
- Projects
- Archive
- RSS