A Better Way To Manage Secrets

JohnnyT - RubyHACK 2019

Hi!

- I’m JohnnyT
- Work locally at Nav
    - Currently on Engineering Services team (enable DevOps)
    - Have strong Ruby, Elixir and Go teams
- I Love Ruby

Thank You

Matz

What Are Secrets?

- Static Secrets (DB credentials, Billing API Keys, …)
- Encryption (Credit Card #, SSN, …)
- Many more that we won’t cover

Current Way - Static

- Sensitive config needs to be encrypted and protected
- Ansible Vault, Kubernetes Secrets
- Do you know when these were requested/viewed?
- What to do if a malicious actor has access to them?

Current Way - Encryption

- We shouldn’t roll our own encryption (use standard libs)
- How are your encryption keys stored and accessed?
- Have they ever been in a place that they could have been copied? (Can you audit this?)
- Do you know the crypto period of these keys?
- How do you deal with key rotation?
- Can you stop someone from decrypting an old DB backup?

HashiCorp Vault

- A system to “Manage Secrets and Protect Sensitive Data”
- In use:
  - Locally - Nav, MX, Adobe (I’m sure others too)
- Separate service - accessed via UI, CLI or HTTP API
- Has to be running and unsealed
- Out of band processes (policy management, management of secrets)

Vault is Complex

- There is a lot to HashiCorp Vault
- Running an HA Vault cluster is some work
- We are not covering all topics here

Example Setup

- Vault server (dev mode)
- RubyGem: vault
- When interacting with Vault in these examples:

```
export VAULT_ADDR=http://127.0.0.1:8200
export VAULT_TOKEN=root
```

Dev Server Setup

```
vault server -dev -dev-root-token-id=root
```

```
# Use v1 secret engine for this demo
vault secrets disable secret
vault secrets enable -path=secret -version=1 kv
```

Static Secret Setup

```
vault kv put secret/signup DB_USERNAME=foo DB_PASSWORD=bar
```

Static Secret Use

```ruby
ENV["VAULT_ADDR"] = "http://127.0.0.1:8200"
ENV["VAULT_TOKEN"] = "root"

require "vault"

secret = Vault.logical.read "secret/signup"
p secret.data
# {:DB_PASSWORD=>"bar", :DB_USERNAME=>"foo"}
```

Encryption Setup

```
vault secrets enable transit

vault write -f transit/keys/ssn
```

Encryption Use

```
ENV["VAULT_ADDR"] = "http://127.0.0.1:8200"
ENV["VAULT_TOKEN"] = "root"

require "vault"
require "base64"

plaintext = "123-45-6789"
encoded_plaintext = Base64.strict_encode64 plaintext

secret = Vault.logical.write "transit/encrypt/ssn",
  plaintext: encoded_plaintext

p secret.data[:ciphertext]
# "vault:v1:qgKoVHEYrxwl3BO1qymjZIaOa9mkoO8Qul/MUOeBpFrTvw8HSfCu"
```

Decryption Use

```
ENV["VAULT_ADDR"] = "http://127.0.0.1:8200"
ENV["VAULT_TOKEN"] = "root"

require "vault"
require "base64"

ciphertext = "vault:v1:qgKoVHEYrxwl3BO1qymjZIaOa9mkoO8Qul/MUOeBpFrTvw8HSfCu"

secret = Vault.logical.write "transit/decrypt/ssn",
  ciphertext: ciphertext

encoded_plaintext = secret.data[:plaintext]
Base64.decode64 encoded_plaintext
# "123-45-6789"
```

Remove Root

- Extremely powerful, can be dangerous
- Treat it like a captured enemy ninja
  - Don’t leave it alone
  - Remove it once you’ve extracted needed info
- We need two things in order to do this:
  - Authorization (policies)
  - Authentication

Policies

- Define what a token can do
- Path based

App Policy

```
path "secret/signup" {
  capabilities = ["read"]
}

path “transit/encrypt/ssn“ {
  capabilities = [“create”,”update"]
}
```

```
vault policy write signup_app policy_signup.hcl
```

Ops Policy

```
path "*" {
  capabilities = ["sudo"]
}
```

```
vault policy write ops policy_ops.hcl
```

Static Secret - Token

```ruby
ENV["VAULT_ADDR"] = "http://127.0.0.1:8200"

app_token = `vault token create -address=http://127.0.0.1:8200 \
  -policy=signup_app -field=token`

ENV["VAULT_TOKEN"] = app_token

require "vault"

secret = Vault.logical.read "secret/signup"
p secret.data
# {:DB_PASSWORD=>"bar", :DB_USERNAME=>"foo"}
```

Static Secret - Default

```
ENV["VAULT_ADDR"] = "http://127.0.0.1:8200"
require "vault"

default_token = `vault token create -address=http://127.0.0.1:8200 \
  -policy=default -field=token`

Vault.token = default_token
secret = Vault.logical.read "secret/signup"
# BOOM!
# The Vault server at `http://127.0.0.1:8200' responded with a 403.
# (Vault::HTTPClientError)
# Any additional information the server supplied is shown below:
#    * 1 error occurred:
#    * permission denied
```

Authentication

- Allow different entities to authenticate
- Vault has multiple Auth Methods
  - People (LDAP, GitHub, Username/PW, Okta, …)
  - Machines (AWS, Kubernetes, Google Cloud, …)
- Good practice to auth before launching process

Revisit Encryption

- After a while - we will start getting permission denied
- Our token has been revoked

Leases

- (Almost) Everything is tied to a lease and will expire
- Kitchen timer on a conveyer belt
  - Conveyer Belt - Max Time To Live
  - Timer - Time To Live (can renew)
  - Revoked when the bell rings (out of time, or falls off)

Renewing Leases

- Need something to renew
  - Envconsul (out of your process)
  - Health Check (in your process)

Dynamic Secrets

- Revisit our DB credentials - can we use a lease?
- Additional Vault Secret Backends:
  - Database
  - AWS

Audit Logs

```
vault audit enable file file_path=/var/log/vault_audit.log

# or in a container
vault audit enable file file_path=stdout
```

Other Vault Stuff

- Compliance - PCI DSS Section 3
- Storage Backends
- Shamir’s Secret Sharing - Master (or Key) Encryption Key
- Sealing / Unsealing Vault
- HA Cluster (Vault becomes a CRITICAL part of your system)

Secret Management

- More than just encryption
- There are systems out there to help

Thanks!

Questions / Resources

- Vault Project Website: https://www.vaultproject.io/
- Learn Vault: https://learn.hashicorp.com/vault/