# Lab 3: Cubbyhole Secret Engine

Duration: 10 minutes

This lab demonstrates both CLI commands and API to interact with key/value and cubbyhole secret engines.

- Task 1: Test the Cubbyhole Secret Engine using CLI
- Task 2: Trigger a response wrapping
- Task 3: Unwrap the Wrapped Secret

## Task 1: Test the Cubbyhole Secret Engine using CLI

### Step 3.1.1

To better demonstrate the cubbyhole secret engine, first create a non-privileged token.

```
$ vault token create -policy=default
```

Expected output look similar to:

```
Key                  Value
---                  -----
token                da773bfc-24bd-a364-4cce-46560c4fdcf1
token_accessor       4f536d51-5084-25f1-3bb6-9ae2e4ecfcf9
token_duration       768h
token_renewable      true
token_policies       ["default"]
identity_policies    []
policies             ["default"]
```

### Step 3.1.2

Log into Vault using the newly generated token:

```
$ vault login <token>
```

Example:

```
$ vault login 9c247c5d-c2be-2ba2-c450-34f33f668ecf

Success! You are now authenticated. The token information displayed below is
already stored in the token helper. You do NOT need to run "vault login" again.
Future Vault requests will automatically use this token.

Key                  Value
```

```
---                  -----
token                697d340d-1d78-7497-cb97-72bb6b3f42a5
token_accessor       aa2e40ff-30be-1810-4bf2-2c151e4f4782
token_duration       767h59m46s
token_renewable      true
token_policies       ["default"]
identity_policies    []
policies             ["default"]
```

### Step 3.1.3

Execute the following command to write secret in the `cubbyhole/private` path:

```
$ vault write cubbyhole/private mobile="123-456-7890"
```

### Step 3.1.4

Read back the secret you just wrote. It should return the secret.

```
$ vault read cubbyhole/private

Key       Value
---       -----
mobile    123-456-7890
```

### Step 3.1.5

Login with root token:

```
$ vault login root
```

### Step 3.1.6

Now, try to read the `cubbyhole/private` path.

```
$ vault read cubbyhole/private
```

What response did you receive?

Cubbyhole secret backend provide an isolated secrete storage area for an individual token where no other token can violate.

## Cubbyhole Wrapping Token

Think of a scenario where a user does **not** have a permission to read secrets from the `secret/data/training` path. As a privileged user (admin), you have a permission to read the secret

in `secret/data/training`.

You can use response wrapping to pass the secret to the non-privileged user.

1. Admin user reads the secret in `secret/data/training` with response wrapping enabled
2. Vault creates a temporal token (wrapping token) and place the requested secret in the wrapping token's cubbyhole
3. Vault returns the wrapping token to the admin
4. Admin delivers the wrapping token to the non-privileged user
5. User uses the wrapping token to read the secret placed in its cubbyhole

Remember that cubbyhole is tied to its token that even the root cannot read it if the cubbyhole does not belong to the root token.

> **NOTE:** A common usage of the response wrapping is to wrap an initial token for a trusted entity to use. For example, an admin generates a Vault token for a Jenkins server to use. Instead of transmitting the token value over the wire, response wrap the token, and let the Jenkins server to unwrap it.

# Task 2: Trigger Response Wrapping

### Step 3.2.1

Execute the following commands to read secrets using response wrapping with TTL of 360 seconds.

```
$ vault kv get -wrap-ttl=360 secret/training
```

Output should look similar to:

```
Key                              Value
---                              -----
wrapping_token:                  0a728b26-7db7-3b2b-5c6a-9c09ac073c2e
wrapping_accessor:               e0731c2d-5c5e-fe16-d645-f10b80375bd3
wrapping_token_ttl:              6m
wrapping_token_creation_time:    2018-09-13 23:04:19.856332048 +0000 UTC
wrapping_token_creation_path:    secret/data/training
```

The response is the wrapping token; therefore, the admin user does not even see the secrets.

Make a note of this **`wrapping_token`**. You will use it later to unwrap the secret.

# Task 3: Unwrap the Wrapped Secret

Since you are currently logged in as a root, you are going to perform the following to demonstrate

the apps operations:

1. Create a token with default policy (non-privileged token)
2. Authenticate with Vault using this default token
3. Unwrap the secret to obtain the apps token
4. Verify that you can read `secret/data/dev` using the apps token

### Step 3.3.1

Generate a token with default policy:

```
$ vault token create -policy=default

Key                  Value
---                  -----
token                be17b1d0-ca70-d9a6-f44c-c0dfacf3ce37
token_accessor       2f9bda50-8c33-e0c1-ee9a-c2917b3fe8f0
token_duration       768h
token_renewable      true
token_policies       ["default"]
identity_policies    []
policies             ["default"]
```

### Step 3.3.2

Login with the generated token.

Example:

```
$ vault login be17b1d0-ca70-d9a6-f44c-c0dfacf3ce37

Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                be17b1d0-ca70-d9a6-f44c-c0dfacf3ce37
token_accessor       2f9bda50-8c33-e0c1-ee9a-c2917b3fe8f0
token_duration       767h58m3s
token_renewable      true
token_policies       ["default"]
identity_policies    []
policies             ["default"]
```

### Step 3.3.3

Test to make sure that you cannot read the `secret/data/training` path with default token.

```
$ vault kv get secret/training
```

```
Error making API request.

URL: GET http://127.0.0.1:8200/v1/sys/internal/ui/mounts/secret/training
Code: 403. Errors:

* Preflight capability check returned 403, please ensure client's policies grant
access to path "secret/training/"
```

### Step 3.3.4

Now, execute the following commands to unwrap the secret.

```
$ vault unwrap <WRAPPING_TOKEN>
```

Where `<WRAPPING_TOKEN>` is the `wrapping_token` obtained at ***Step 3.2.1***.

For example:

```
$ vault unwrap 0a728b26-7db7-3b2b-5c6a-9c09ac073c2e

Key         Value
---         -----
data        map[course:Vault 101 password:another-password]
metadata    map[destroyed:false version:4
created_time:2018-09-13T21:29:23.489065342Z deletion_time:]
```

Since the wrapping token is a **single-use** token, you will receive an error if you re-run the command.

### Step 3.3.5

Log back in as root:

```
$ vault login root
```

### Question

What happens to the token if no one unwrap its containing secrets within 360 seconds?

### Answer

To test this, generate a new token with short TTL (e.g. 15 seconds):

```
$ vault token create -wrap-ttl=15 -format=json \
        | jq -r ".wrap_info.token" > wrapping-token.txt
```

The above command stores the generated `wrapping_token` in a file.

Wait for 15 seconds and try to unwrap the containing secret:

```
$ vault unwrap $(cat wrapping_token.txt)

Error unwrapping: Error making API request.

URL: PUT http://127.0.0.1:8200/v1/sys/wrapping/unwrap
Code: 400. Errors:

* wrapping token is not valid or does not exist
```

> **NOTE:** The TTL of the wrapping token is separate from the wrapped secret's TTL (in this case, a new token you generated).

## Additional Exercises

To learn more about Cubbyhole secret engine, try additional hands-on exercises:

- Cubbyhole Response Wrapping guide: https://www.vaultproject.io/guides/secret-mgmt/cubbyhole.html
- Katacoda scenarios authored by HashiCorp: https://www.katacoda.com/hashicorp/scenarios/vault-cubbyhole

### End of Lab 3