Lab 5: Secrets as a Service - Dynamic Secrets

Duration: 25 minutes

This lab demonstrates how Vault generates dynamic credentials for database on-demand.

- Task 1: Enable and Configure a Database Secret Engine
- Task 2: Generate Readonly PostgreSQL Credentials
- Task 3: Revoke Leases
- Challenge: Setup Database Secret Engine via API

The scenario is:

A privileged user (e.g. admin, security team) enables and configures the database secret engine. Also, creates a role which defines what kind of users to generate credentials for. Once the secret engine is set up, the Vault clients (apps, machine, etc.) can request a new set of database credentials. Since the clients don't need the database access for a prolong time, you are going to set its expiration time as well.

Task 1: Enable and Configure a Database Secret Engine

For a production environment, this task is performed by a privileged user.

Step 5.1.1

Most secret engines must be enabled and configured before use. Execute the following command to enable database secret engine:

```
$ vault secrets enable database
```

NOTE: By default, this mounts the database secret engine at `database/` path. If you wish the mounting path to be different, you can pass `-path` to set desired path.

Expected output:

```
Success! Enabled the database secrets engine at: database/
```

Step 5.1.2

Now that you have mounted the database secret engine, you can ask for help to configure it. Use the path-help command to display the help message.

```
$ vault path-help database/
```

Also, refer to the online API document: https://www.vaultproject.io/api/secret/databases/index.html.

Step 5.1.3

In this lab scenario, you are going to configure a database secret engine for PostgreSQL.

Execute the following command to configure the database secret engine:

```
$ vault write database/config/postgresql \
       plugin_name=postgresql-database-plugin \
       allowed_roles=readonly \
       connection_url=postgresql://postgres@localhost/myapp
```

NOTE: For the purpose of training, PostgreSQL has been installed and a database name, myapp, has been created on each student workstation. It is very common to give Vault the root credentials and let Vault manage the auditing and lifecycle credentials instead of having one person manage it manually.

Step 5.1.4

Notice that you set the `allowed_roles` to be readonly in previous step. Since Vault does not know what kind of PostgreSQL users you want to create. So, you supply the rule with the SQL to run and create the users.

Since this is not a SQL course, we've added the SQL on the student workstation. You can see the script:

```
$ cat readonly.sql
CREATE ROLE "{{name}}" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL
'{{expiration}}';
REVOKE ALL ON SCHEMA public FROM public, "{{name}}";
GRANT SELECT ON ALL TABLES IN SCHEMA public TO "{{name}}";
```

The values between the `{{ }}` will be filled in by Vault. Notice that we are using the VALID UNTIL clause. This tells PostgreSQL to revoke the credentials even if Vault is offline or unable to communicate with it.

Step 5.1.5

Next step is to configure a role. A role is a logical name that maps to a policy used to generate credentials. Here, we are defining a readonly role.

```
$ vault write database/roles/readonly db_name=postgresql \
        creation_statements=@readonly.sql \
        default_ttl=1h max_ttl=24h
```

NOTE: This command creates a role named, readonly which has a default TTL of 1 hour, and max TTL is 24 hours. The credentials for readonly role expires after 1 hour, but can be renewed multiple times within 24 hours of its creation. This is an example of restricting how long the database credentials should be valid.

Task 2: Generate Read-only PostgreSQL Credentials

As described earlier, privileged users (admin, security team, etc.) enable and configure the database secret engine. Therefore, Task 1 is a task that needs to be completed by the privileged users.

Now that the database secret engine has been enabled and configured, applications (Vault clients) can request a set of PostgreSQL credential to read from the database.

Step 5.2.1

Execute the following command to generate a new set of credentials:

```
$ vault read database/creds/readonly
```

The output should look similar to:

```
Key                Value
---                -----
lease_id           database/creds/readonly/86a2109c-780c...
lease_duration     1h
lease_renewable    true
password           A1a-u443zy2w14245784
username           v-token-readonly-x271s0zv6x42wsqx...
```

To generate new credentials, you simply read from the role endpoint.

Step 5.2.2

Copy the `lease_id`. You will use it later.

Step 5.2.3

Let's check that the newly generated username exists by logging in as the postgres user and list all accounts.

```
$ psql -U postgres
```

At the `postgres` command prompt, enter `\du` to list all accounts.

```
postgres > \du
```

The username generated at ***Step 5.2.1*** should be listed.

Notice that the Attributes for your username has "password valid until" clause.
This means that even if an attacker is able to DDoS Vault or take it offline, PostgreSQL will still revoke the credential. When backends support this expiration, Vault will take advantage of it.

Step 5.2.4

Enter `\q` to exit.

Step 5.2.5

Now, let's renew the lease for this credential.

```
$ vault lease renew <lease_id>
```

While `<lease_id>` is what you copied at ***Step 5.2.2***.

Expected output:

```
Key                Value
---                -----
lease_id           database/creds/readonly/86a2109c-780c...
lease_duration     1h
lease_renewable    true
```

The lease duration for this credential is now reset.

For the clients to be able to read credentials and renew its lease, its policy must grants the following:

```
# Get credentials from the database backend
path "database/creds/readonly" {
   capabilities = [ "read" ]
}

# Renew the lease
path "/sys/leases/renew" {
   capabilities = [ "update" ]
}
```

Task 3: Revoke Leases

Under a certain circumstances, the privileged users may need to revoke the existing database credentials.

Step 5.3.1

When the database credentials are no longer in use, or need to be disabled, run the following command:

```
$ vault lease revoke <lease_id>
```

While `<lease_id>` is what you copied at ***Step 5.2.2***.

Expected output:

```
All revocation operations queued successfully!
```

Step 5.3.2

You can verify that the username no longer exists by logging in as postgres user and list all accounts as you did in ***Step 5.2.3***.

Step 5.3.3

Let's read a few more credentials from the postgres secret engine. Here, you will simulate a scenario where multiple applications have requested readonly database access.

```
$ vault read database/creds/readonly
Key                Value
---                -----
lease_id           database/creds/readonly/563e5e58-aa31-564c-4637-70804cc63fe1
lease_duration     1h
lease_renewable    true
password           A1a-zr9q5t79391w569z
username           v-token-readonly-0306y039q232wvr2y59p-1517945642


$ vault read database/creds/readonly
Key                Value
---                -----
lease_id           database/creds/readonly/67fdf769-c28c-eba7-0ac4-ac9a52f13e4c
lease_duration     1h
lease_renewable    true
password           A1a-89q59vqz83z892xs
username           v-token-readonly-74551qs2us5zzqwsqw56-1517945647
```

```
$ vault read database/creds/readonly
Key                Value
---                -----
lease_id           database/creds/readonly/b422c54b-2664-e0b4-1b6e-74badbd7ab1c
lease_duration     1h
lease_renewable    true
password           A1a-0wsw97r2x6s49qv9
username           v-token-readonly-838uu0r2vvzyw0p34qw4-1517945648
```

Now, you have multiple sets of credentials.

Step 5.3.4

Imagine a scenario where you need to revoke all these secrets. Maybe you detected an anomaly in the postgres logs or the vault logs indicates that there may be a breach!

If you know exactly where the root of the problem, you can revoke the specific leases as you performed in ***Step 5.3.1***. But what if you don't know!?

Execute the following command to revoke all readonly credentials.

```
$ vault lease revoke -prefix database/creds/readonly
```

Expected output:

```
All revocation operations queued successfully!
```

If you want to revoke all database credentials, run:

```
$ vault lease revoke -prefix database/creds
```

Challenge: Setup Database Secret Engine with API

Perform the same tasks using API.

1. Enable database secret engine at a different path (e.g. `postgres-db/`)
2. Configure the secret engine using the same parameters in Task 1
    - **plugin_name:** postgresql-database-plugin
    - **allowed_roles:** readonly
    - **connection_url:** postgresql://postgres@localhost/myapp
3. Create a new role named, readonly
    - **db_name:** postgresql
    - **creation_statements:** readonly.sql
    - **default_ttl:** 1h

    - **max_ttl:** 24h
4. Generate a new set of credentials for readonly role

Hint:

- Database Secret Engine API doc
- PostgreSQL Database Secret Plugin HTTP API