31 OCTOBER 2017 / VAULT

How to use Vault's Cubbyhole Secret Backend and Response Wrapping

I just started a new Club, it's called the Spicy Coderz Club and it is the newest most exclusive club for spicy coderz ONLY. Naturally everyone wants to join, and many people try and sneak in to our clubhouse everyday. But we take security VERY seriously. Everyday we generate a new password, and only by saying that day's password can you enter the clubhouse. As head of security at the SCC, I had to figure out a way to distribute the daily password to all members in the most secure manner. Vault was the obvious choice. This post will give you an basic overview of how we use the Cubbyhole Secret Backend and Response Wrapping, so you can secure your own clubhouse!

Setting up a Vault Server and Logging

The actual Spicy Coderz Club Vault server lives in the "Cloud", but for this post, we are going to use Vault's Dev mode. This is NOT how you would use Vault in a production environment, but will suffice for demonstration purposes.

This post assumes you already have Vault installed on your local computer, if you need more help installing Vault please consult the Documentation.

Set Vault Address to HTTP instead of HTTPS:

```bash
export VAULT_ADDR='http://127.0.0.1:8200'
```

Note: We have to update the Vault Address because we are using Dev mode. This is not something you would run in production.

Start the Vault server:

```bash
vault server -dev
```

This will result in some output like:

```
==> WARNING: Dev mode is enabled!
```

```
In this mode, Vault is completely in-memory and unsealed.
...
Unseal Key: l+L/zOZVw+aNEaMGSyQLAV4iAAGPoFPX2SrkYsazWmk=
Root Token: 00dfe942-df0f-4cbd-66a2-db99c1212654

==> Vault server started! Log data will stream in below:
```

The important information right now is the root token. We will use this to finish setting up Vault and storing secrets.

Auditing

Since we take security so seriously at the Spicy Coderz Club, we want to use one of Vault's Audit Backends to track all activity with our Vault server. For the purpose of this post, we will use the File Audit Backend, but you also have the option of Syslog and Socket for your own implementation.

Enable Audit Logging:

```bash
export VAULT_TOKEN=00dfe942-df0f-4cbd-66a2-db99c1212654
vault audit-enable file file_path=vault_audit.log log_raw=true hmac_accessor=false
```

```
Successfully enabled audit backend 'file' with path 'file'!
```

Note: For the purpose of this post we are not encrypting the data, so we can more easily see who is interacting, with what on our Vault server. This is NOT what you would do in a production environment. The two settings we modified are:

> ***log_raw optional*** *A string containing a boolean value ('true'/'false'), if set, logs the security sensitive information without hashing, in the raw format. Defaults to false.*
>
> ***hmac_accessor*** *optional A string containing a boolean value ('true'/'false'), if set, enables the hashing of token accessor. Defaults to true. This option is useful only when log_raw is false.*

Also Note: If you enable vault auditing without a fully qualified file path (`filepath=vault_audit.log` versus `filepath=/var/logs/vault_audit.log`), it will place that file in folder where the Vault Server was started, not the folder where the command was run from.

After we enable Audit logging, we see a file appear called `vault_audit.log` with some information looking like the following:

```
{
    "time": "2017-10-22T16:25:00Z",
    "type": "response",
    "auth": {
        "client_token": "00dfe942-df0f-4cbd-66a2-db99c1212654",
        "accessor": "7c2a6947-5dd5-f6c6-5594-971d5b883907",
        "display_name": "root",
        "policies": [
            "root"
        ],
        "metadata": null
    },
    "request": {
        "id": "51455a80-91eb-0571-6348-93a726708727",
```

```
        "operation": "update",
        "client_token": "00dfe942-df0f-4cbd-66a2-db99c1212654",
        "client_token_accessor": "7c2a6947-5dd5-f6c6-5594-971d5b883907",
        "path": "sys/audit/file",
        "data": {
            "description": "",
            "local": false,
            "options": {
                "file_path": "vault_audit.log",
                "hmac_accessor": "false",
                "log_raw": "true"
            },
            "type": "file"
        },
        "remote_address": "127.0.0.1",
        "wrap_ttl": 0,
        "headers": {}
    },
    "response": {},
    "error": ""
}
```


This is the basic format of Vault Audit logs: A request and response, with all relevant information like what token was used for the request, and what the response contained, as well as any errors.

Tail the Audit Logs:

```bash
tail -F vault_audit.log | while read line; do echo "$line" | jq; done
```

Note: You will need jq installed for this command. We use jq to help format and colorize the Audit logs. If you don't want formatting/coloring, you can just tail the logs with `tail -F vault_audit.log`.

Disabling and Updating Auditing

If you want to update where you mounted the Vault Audit backend or change the file path you will need to disable the Vault backend first.

Disable Audit Logging:

```
vault audit-disable file
```

We now have a Vault server setup with some audit logging, we are ready to start storing and sharing secrets! But first we need to meet our Club members.

Introducing the Spicy Coderz Club

Aside from myself, we currently have 3 members in the Spicy Coders Club:

Pappas