Secrets

# Introduction

This document describes how Neon Research manages the secrets required to operate its services.

# Environments

Secrets are managed separately by environment. Currently, Neon Research operates two environments:

**neontest.com** Used for the development and testing purposes.

**neoncluster.io** Used for production purposes.

# Secrets

Each environment will require a number of secrets to operate. These secrets may include:

* TLS/SSL certificates and keys
* Database credentials
* Hashicorp Vault unseal keys
* Hive node root credentials (e.g. SSH client keys or passwords)

A neonHIVE currently makes the following trust assumptions:

* Node root accounts are secure
* Node RAM is secure
* Root home directory is encrypted
* Hashicorp Vault is secure

To make things easier for system operators, the **neon-cli** tool collects all of the information required into **Hive Secrets** files, one for each hive being managed. Each hive secrets file includes the hive definition, SSH/SCP client credentials, Vault unseal keys, Vault root token.

The **neon-cli** tool persists hive credential files to an application folder[[1]](#footnote-1) on the operator’s Windows workstation. These files are formatted as JSON and their names match the corresponding hive name. The files are encrypted at rest using Windows filesystem encryption, which means that the operator must be logged into the Windows workstation for these files to be readable. In the future, we plan to implement smart-card based encryption via [YubiKey](https://www.yubico.com/products/yubikey-hardware/yubikey4/) for additional safety.

The **neon** **login** command, specifying the hive name is used to choose which hive where subsequent commands will be performed. The **neon-cli** tool will extract any required credentials from the selected hive’s secrets file. The **add**, **key**, **list**, **logout**, and **remove** commands can be used to manage hive authorities.

# SSH Public Key Authentication

For better security, neonHIVESs by default disable username/password authentication in favor of SSH Public Key authentication. Public key authentication is generally much more secure because it’s often relatively easy to use brute force attacks against passwords. It is also possible to have **neon-cli** generate and configure a cryptographically random SSH password on the hive hosts.

# HashiCorp Vault

All hives rely on HashiCorp vault for secret management. This is deployed as a Docker service to all hive management nodes and securely persists its secrets to the local Consul hive. Vault starts up as sealed—which means it cannot access its persisted secrets before being manually unsealed by one or more operators. Once Vault is unsealed, it will able to begin processing requests.

Services access Vault using tokens or AppRoles. The overall process works something like this:

## Operator Steps

The operator performs the following steps on his/her workstation using the **neon-cli** tool to remotely submit command to the hive’s Vault as well as Docker:

1. Write one or more secrets to Vault, like an AWS API key.
2. Create a Vault AppRole with policies to read the secrets.
3. Write the AppRole **role-id** and **secret-id** to the Docker secret store (a Docker 1.13 feature)
4. Create a Docker service, passing the Docker secret.

## Docker Steps

At this point, Docker takes over, and for each service container spun up:

1. Docker mounts a **tmpfs** at **/run/secrets** and writes the secrets there.
2. The container is started.
3. The internal container entry point script used the secret to authenticate the AppRole against Vault to retrieve the Vault token to be used for subsequent operations.
4. The script uses the token to retrieve the original secret from Vault (e.g. the AWS key).

## Self-Signed Certificates

A neonHIVE currently requires 3 self-signed certificates:

General **\*.NAME.hive**

Registry Cache **\*.neon-registry-cache.NAME.hive**

Vault **\*.neon-vault.NAME.hive**

where **NAME** is the hive name.

All of these certificates are configured to be implicitly trusted on each hive node.

The registry cache certificate is only used internally within the hive so that the local Docker engines can securely talk to the local registry caches (if enabled for the hive). The General and Vault certificates are configured by all hive nodes and are also to be trusted on the hive operator's Windows or OSX machines so that neon-cli, hive web portals, and other tooling will work seamlessly from the operator’s workstation. neon-cli quietly manages the adding and removing these operator certificates when the user logs into or out of a hive.

In the future, it sure would be nice if we didn’t need to separate Vault certificate. This is currently required due to the **vault-direct** script installed on the hive managers used for initializing and sealing/unsealing specific vault instances.

We’re using self-signed certificates by default to make hive setup super easy.

At some point, it would also be nice if operators could use a certificated signed by a real trusted 3rd party authority and then use this when deploying a cluster using a real DNS name. This would also require that the operator actually register DNS host names for the hive and configure the proper DNS records. This is complicated and I’m not entirely sure that it would actually work. I’m hoping to defer this as long as possible (like until some enterprise is willing to pay me real money 😊).

## Observations

Here are the advantages of this scheme:

* At no point, are secrets persisted unencrypted to disk anywhere on the hive. Secrets are persisted only by Vault and the Vault unseal keys exist only in memory on the hive and the operator workstations encrypted with smart-cards.
* Operators can use Vault policies and roles to grant individual service tokens access to only the secrets they need. This means that it’s possible to prevent one service from reading the secrets of another.
* Vault will generate audit logs whenever tokens are created, apps authenticate and secrets are accessed. These will be useful for detecting and investigating security breaches.
* Tokens can be configured to expire or be explicitly revoked. Smarter containers can verify tokens against Vault and reload secrets for expired tokens or exit for revoked ones. This can provide the basis for periodically rolling out new secrets (e.g. updating TLS certificates).
* **Security Gap #1:** The AppRole secrets written to Docker are stored on disk on the manager nodes and assuming they’re encrypted, the key is somewhere on disk as well (Docker doesn’t have the equivalent of Vault’s unseal). This means that an attacker with access to a manager could conceivable examine the Docker AppRole secrets, and use them to authenticate with Vault and retrieve the real secret. Ultimately, Docker should allow a secrets plugin that could use Vault.
* **Security Gap #2:** The Vault TLS certificate and private key are persisted on the managers. An attacker with root access could configure a man-in-the-middle and record secrets as they are written.
* **Security Gap #3**: The **[~/.exec**] folder temporarily holds files while commands are executed remotely via the **rc.exe** tool. It’s possible that these files could hold sensitive information and an entity with root access could monitor this folder and grab files before they are purged. I’m not sure we can do anything about this.  
    
  It’s also possible for somebody to grab a drive or machine and use forensic techniques to capture this data. We could mitigate this by mounting a tmpfs (RAM) file system here.
* **Security Gap #4**: The **neon-registry-cache** containers are passed the remote registry’s username and password as an environment variable and then persist these in the config file on disk. The credentials should come from Vault and the config file should be hosted on a **tmpfs**. I’m not sure though if this is worth the trouble, since the credentials need to be specified in the Docker system unit file anyway.
* **Security Gap #5**: The hive currently deploys Vault with a self-signed certificate and will soon do this for Consul as well. This opens the potential for a man-in-the-middle attack. This shouldn’t be a huge issue for intra-hive communication because we’ve hardcoded the DNS lookups in the Docker host **/etc/hosts** and **/etc/neon/env-hosts** files, to the attacker would already be in our systems to take advantage of this. I believe the same holds for operators using the **neon-cli** to remotely manage the hive, because that also hardcodes the DNS lookups and assumes that the operator is working over a VPN into the datacenter.
* **Security Gap #6**: Hive VPN certificate generation is using **sha1** digests rather than **sha2** as described [here](http://www.macfreek.nl/memory/Create_a_OpenVPN_Certificate_Authority). Investigate whether this is a security issue.
* **Security Gap #7**: Azure swarm managers expose the hive API at <http://127.0.0.1:2375> so that the **neon-hive-manager** can access the API (this service uses host networking). The manager API will not be accessible from other servers but it’s possible for an attacker with credentials to deploy another service that does the same thing. The thing is though that an attacker with these credentials can do anything already.
* **Security Gap #8**: We’re not using notary to sign images yet, so it’s possible for somebody to hack one of our images and take over the hive.
* **Security Gap #9**: We’re using self-signed certificates to encrypt traffic to internal services like Consul, Vault and hive web dashboards and persisting these to the current user’s trusted certificate store/folder. Some enterprises may not like or even allow this as a domain policy.
* **Security Gap #10**: neon-cli needs to munge the **hosts** file so that the various **\*.hive** DNS names will resolve to the proper hive addresses.
* **Security Gap #11**: neon-cli needs to run with elevated permissions on the operator’s machine for two reasons: manage the trust for hive certificates and to munge the **hosts** file. Some enterprises may not like or even allow this as a domain policy.
* **Security Gap #11**: RabbitMQ and Ceph are both configured with fixed credentials (**sysadmin/password**).

# Secrets Archival

Secrets are archived using **encrypted 7-Zip** files, one archive for each environment. Each archive will be named for the environment whose secrets are being held, e.g. **neontest.com.7z** or **neoncluster.io.7z**. The password used to encrypt the secret archive files is currently written on a blue sticky note in my top desk drawer (soon to be relocated to my safe deposit box).

1. Hive credential files are saved to: **%LOCALAPPDATA%\Neon Research\neon-clusters\clusters** [↑](#footnote-ref-1)