Protecting EdgeX Secrets (Edinburgh release)

As of the California release, EdgeX has a standalone micro service secret store (provided by Hashicorp Vault) for centralize management of credentials and sensitive data across EdgeX. While the secret store has been in place, it has not yet been used to store the secrets of the other EdgeX micro services. With the Edinburgh release, that will change as EdgeX secrets will begin to be stored in Vault. Secrets include usernames, passwords, certificates, tokens, etc. used by the micro services.

This sounds straightforward, but can actually be a complex task. How do the secrets get into secret store? How does the micro service know where/how to get to its secrets and what permission need to be granted to the micro service so that it has access to retrieve its own secrets? How are secrets organized inside of the secret store?

In this document, we outline the overall design of the EdgeX Edinburgh secure storage usage.

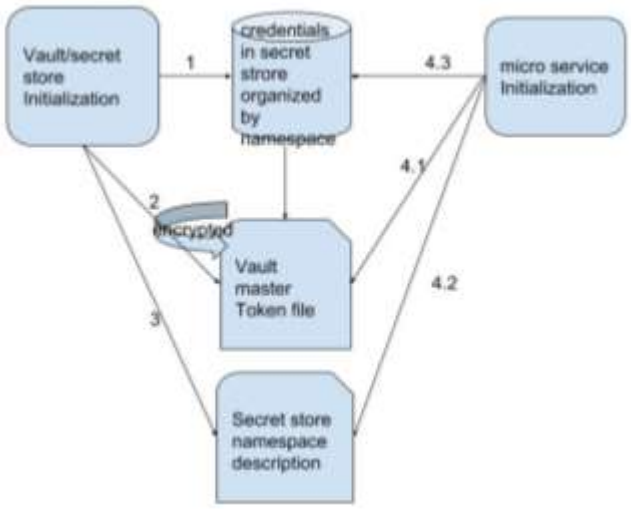
Secret Storage Architecture

This section covers the general design of how the secret store is setup, how its credentials are made available to the other EdgeX micro services and how the other services are able to access the secrets given the credentials. The numbered steps below correspond to the steps in the accompanying diagram. Details for each step are provided later in this document.

1. The secret store is seeded by an initialization program. This program will create different scopes/namespace for the different EdgeX micro services (see the Organization of Secrets section for more details on this organization). It will also populate the credentials and sensitive data for each micro service – storing the data into the appropriate namespace for each service. The namespaces will be separated and protected by configurable policies and an ACL that are back by Vault and implemented in the format of HCL or JSON.
2. A program (this could be the same initialization program for #1) encrypts the secret store access token and places it in a file on the file system (or shared volume when using containers). So that it can be used by the initialization programs and micro services later (see 4.1 below).
3. The secret store (Vault) offers a REST API interface that allows micro services or other initialization programs to get secrets from the secret store with an understanding of the applicable namespace and the access token. The namespace is used as part of the REST call URLs to access the appropriate secret from the store. The namespaces for available secrets for each service is shared with each service (or initialization program) via configuration.
4. When an EdgeX micro service (or initialization script) needs to obtain a credential from the secret store, it needs to perform following steps:
    - 4.1 Get and decrypt the access token from file system or volume
    - 4.2 Get the applicable namespace for the credential
    - 4.3 Make a REST call to the secret store using the appropriate URL derived from the namespace for the credential along with access token to retrieve the credential.

Commented [JW1]: Tingyu – what policies and ACL will we use? I assume this is configurable?

Commented [ZT2R1]: Yes it is configurable, and I updated the phase to include more details.



Vault Initialization

The secret store init script (currently written in Go) will be modified to create namespaces for each individual microservice as well as create secrets/credentials used by each service and store them in the namespace. Notably, this includes the creation of the username/password pairs for MongoDB (or another database like Redis).

The namespaces will be defined by configuration to the script. The script will be modified to take command line arguments to specify the needed secrets. The secrets can be passed as environment variables, or command line parameters when the init script starts up. For security enhancement, GUIDs or random strings can be generated within the init script and stored into their individual Vault namespaces without leaving the scope of Vault.

If the credentials need to be updated the secret store REST API calls can be used.

Commented [JW3]: Tingyu, I made this part up but tell me how the secrets are going to be provided to the vault script. We can't have the Vault script have the secrets otherwise we have just transferred the problem from one open text file to the other.

Commented [ZT4R3]: Updated this section to add a couple of options how the secretes are created and stored.

Vault Master Token File Protection

As shown in the step 2 of the diagram above, the Vault master token file needs to be protected as this is highly sensitive file which includes the key to access the secret store. Either symmetric or asymmetric encryption cab be used to protect the file in the file system (or volume when containerized). Eventually this file needs to be protected with some hardware protection implementation like TPM, TEE, etc.

The secret store init script (written in Go) will further be modified to encrypt and store the master token in the file system or volume. Something like GnuPG (free implementation of the OpenPGP standard – see https://www.gnupg.org/) will be used to protect the file using a passphrase and unencrypted token file provided as a command line argument to the init script. Here is an example call to GnuPG that the init script has to make programmatically

```
gpg --yes --batch --passphrase=$PASS -c $TOKENFILE
```

where $PASS is the passphrase used to encrypt the file and $TOKENFILE is the file to be encrypted. The location of the file would be provided by configuration to the init script. The location would be configuration also needed by any other service that needs to access the secret store.

Organization of Secrets (Namespaces)

The credentials and sensitive data will be organized in the secret store by EdgeX micro service. A different namespace path will be used for each micro service. In general, credentials will be organized under a namespace of v1/secret/edgex/:path where path is a string that represents the individual micro service. For database initialization, specifically Mongo, the initial username/password pair would be accessed through the path of v1/secret/edgex/mongodbinit, and a REST API request of the secret store would return JSON like this:

```
{
        “auth”: null,
        “data”: {
                “initusername”: “user”,
                “initpasswd”: “passwd”
},
        “lease_duration”: 3600,
        “lease_id”: “”,
        “renewable”: false
}
```

As additional examples, core secrets (if it has any) would be in v1/secret/edgex/coredata and a modbus device service secrets (if any) would be in v1/secret/edgex/devicemodbus. Assuming core data needs credentials like username/password pair to access core data MongoDB, and need a GUID to identify itself when sending data to another microservice, an HTTP request to {SECRET-STORE-IP}:8200/v1/secret/edgex/coredata will be given response something like this:

```
{
        “auth”: null,
        “data”: {
                “mongousername”: “user”,
                “mongopasswd”: “passwd”,
                “guid”: ”xxxxxxxxx-xxxxxxxx-xxxxxxxx”
},
        “lease_duration”: 3600,
        “lease_id”: “”,
        “renewable”: false
}
```

Database Initialization

Mongo initialization is a standalone program and service that creates the MongoDB data structures (and access) for EdgeX micro services (https://github.com/edgexfoundry/docker-edgex-mongo). Similar initialization scripts and services may exist for other database implementations (such as Redis). When the database initialization program executes, it requires an initial access username/password pair to

Commented [JW5]: Tingyu – make sure I have this part correct. I kind of made it up from several versions of your doc. Rather than having another script, let’s just say we are using the existing script – unless you can think of a reason of why not to.

Commented [ZT6R5]: This explains the purpose very well.

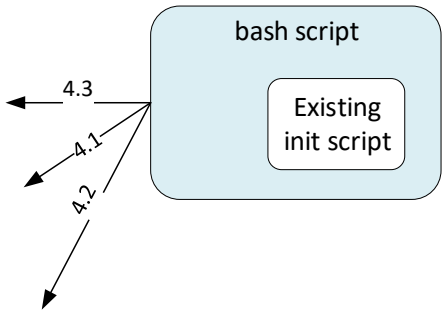
Commented [JW7]: Tingyu – check this. I wanted to provide a few more examples.

Commented [ZT8R7]: Add one more http request/response pair example.

Commented [JW9]: Tingyu – be more generic. We have to consider Redis or other database option here. We will have a Redis implementation for Edinburgh too.

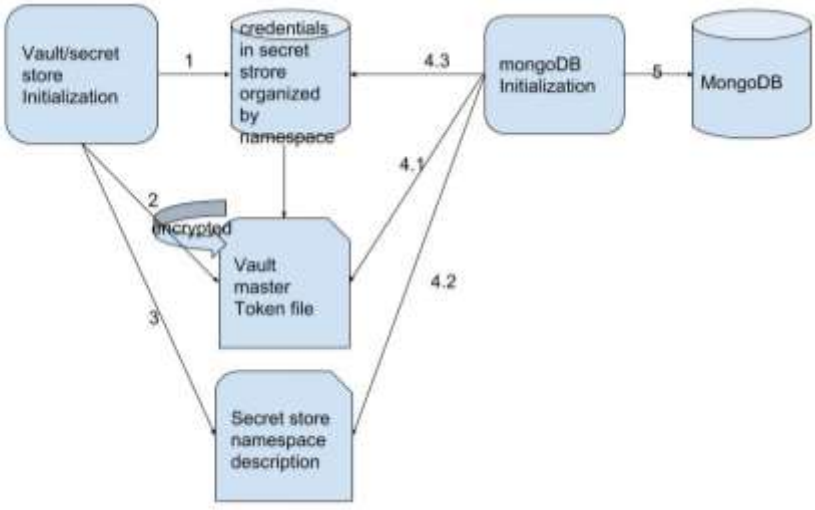
Commented [ZT10R9]: Redis is based on trusted model so the protection could be based on different methodology. More research is needed to provide options to secure Redis.

create the database instance. It may, as in the case of MongoDB, also create several collections or structures each requiring other username/password pairs. Typically, there is a separate (and uniquely protected) collection for each EdgeX micro service (such as a collection for coredata, metadata etc.). Currently the initial username/password as well as username/password pairs for micro services are saved in plaintext format in the initialization script. Under the new architecture, the database initialization script will query the secret store to obtain the credentials and use them to perform initialization.



The existing MongoDB initialization program is implemented using JavaScript (mongo_init.js). For Edinburgh, a bash script will get and decrypt the master token file, use the master token to get the appropriate username/password pairs and the call on a revised initialization script with the pairs to initialize the database and all appropriate collections. In other words, a bash script will perform operations 4.1-4.3 above for retrieving all database username/password pairs and then call to execute the initialization JavaScript with the username/password pairs.

Similar work will need to be accomplished for Redis initialization.



Master Token Access by Micro Services

To consume the secret service, each individual micro service or initialization script needs to obtain an access token. Services or initialization scrips that need to use the master token to get access to secrets in the store will have to use the same library and passphrase (passed as a command line argument to the service or initialization script) as used by the Vault initialization script to decrypt the secret obtained from the file. Programmatically, the service or script will have to do something similar to:

```
gpg –-yes –-batch –-passphrase=$PASS {$TOKENFILE}.gpg
```

Ideally, and in future releases, the access token need for each service will be different so that an access control and policies can be enforced per service. In the Edinburgh release, we will not distinguish the roles and provide each service with access to master token encrypted and store it in the file system or volume that is accessed equally by all micro services.

Commented [JW11]: Tingyu, check that I have interpreted your meaning correctly here.

Commented [ZT12R11]: Yes it is explained well.

Micro Service Secrets Retrieval (in Go)

Any credential that is kept in configuration file or other plaintext format currently needs to be moved to secret store, and a code module needs to be created and made available to all micro services needing to retrieve credentials/secrets. The general steps for the Go module will be similar to what is described in the step 4 of "Secret Store Architecture" previously, which is

- decrypt the access token to secret store
- obtain REST API path for the credential
- REST calls to API endpoint to retrieve the credential with access token

Edinburgh Security Work Outlined

To add the features outlined in this document to at least secure the database secrets (username/password pairs) for all consuming micro services, the following tasks will need to be completed:

1. Modify the secret store service initialization (implemented in Golang) to create the initial namespaces and credentials for Mongo.
2. Modify the secret store service initialization to encrypt the master token to a configuration defined file location to the file system or shared volume.
3. Define the secret namespaces and share the namespace definitions with the individual consuming micro service or init scripts. This can be either specified in a configuration file or within Consul (as with other configuration, it should be done by Consul when available or configuration file when Consul is not available).
4. Create a bash script to fetch and decrypt the master token and access the secret store to fetch the database username/password pairs, and then call the existing database (Mongo) initialization script to properly initialize the databases and their associated collections.
5. Create a "client" Go module to fetch and decrypt the master token file, get secret namespace information from configuration and access secrets (in particular the database username/password pairs) from the secret store using the token and namespace. The module will serve as a template for other language libraries in the future to access the secret store in the same fashion.

Commented [JW13]: I cleaned some of this up. Make sure I have it right.

Commented [ZT14R13]: Just made a bit update (remove Redis as it may need different strategy to protect).