EDGE X FOUNDRY™

EdgeX Hardware-based Secure Storage Design

Change History

Date	Author/Editor	List of Changes
Nov-29-2018	David Ferriera (ForgeRock)	Initial Draft Release
Dec-05-2018	James White (Dell)	Comments/Design details
Dec-06-2018	Alain Pulluelo (ForgeRock)	Comments/Design details
Dec-10-2018	David Ferriera (ForgeRock)	• Consolidate comments • Changes to API specification section • Changes to plugin architecture section
Dec-11-2018	David Ferriera (ForgeRock)	Incorporated comments from Jim and Tingyu

Introduction

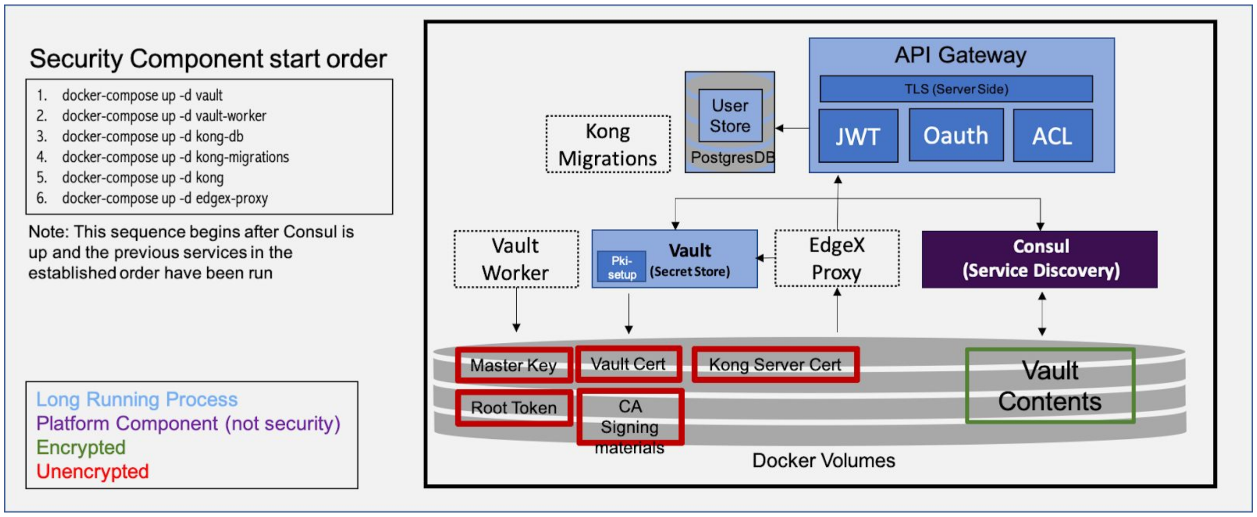
This document is a design for a pluggable interface for an EdgeX security component to enable initial EdgeX secrets to be encrypted using a hardware based secure element such as a TPM or a TEE.

Overview

The EdgeX platform contains (or will contain) many secrets such as database credentials, various encryption/signature keys, X.509 certificates, authentication tokens, username/password pairs, Oauth client credentials, IOT cloud platform credentials/keys (e.g., AWS IOT keys). EdgeX has chosen to use HashiCorp Vault for managing secrets. Currently, several of these "initial" secrets are stored on the file system in the clear. To enable the encryption of these secrets, Hardware based key storage is a critical component.

The EdgeX platform is intended for use on multiple hardware platforms. These platforms include Intel, ARM64 and ARM32. Within these platforms, there are different technologies to solve hardware-based trust, storage and related problems. For example, Intel has SDO, TPM, TXT , SGX and EPID.

Current State (Delhi)


Security Component start order

```
1.  docker-compose up -d vault
2.  docker-compose up -d vault-worker
3.  docker-compose up -d kong-db
4.  docker-compose up -d kong-migrations
5.  docker-compose up -d kong
6.  docker-compose up -d edgex-proxy
```

Note: This sequence begins after Consul is up and the previous services in the established order have been run

Upon initial startup, Vault needs to pull in a certificate for TLS. Currently this is generated by the pkisetup service inside of the Vault container and placed on a docker volume unencrypted. The pkisetup service also generates an external certificate (including a private key) for Kong and places it on the file system unencrypted. After, vault-worker initializes and unseals Vault. This process generates a root token and a master key which are stored on the file system unencrypted. Additionally, non-root tokens are generated by vault-worker for use by other EdgeX services to access vault. Vault-worker also configures paths and policies in Vault using HCL files.

After Vault setup is completed, Kong and PostgresDB are started and configured. As Kong starts, it requires a TLS certificate and it’s corresponding private key for the admin API. If they don’t exist in the configured location on the file system, Kong generates and installs a cert and places it in the expected path on the file system. The Edgex Proxy service configures Kong. As part of this configuration, the root token for Vault is retrieved from the file system and used to call Vault to retrieve and install the Kong external cert.

A Quick Vault Overview

Vault is a software based secret management service. Vault serves two primary functions:

1. Secrets storage – encrypted key/value pairs with secure key access, wrapping and management included (rotations, TTL, usage frequency, etc.)
2. Data encryption – Vault provides encryption as a service via REST based API calls. An application with the proper permissions can send unencrypted data to vault and have encrypted data returned and vice versa. In this case, vault does not store any data other than the encryption key.

Vault also has an extensive list of technology integrations via a plugin architecture including

- Public cloud providers - AWS, Azure, GCP, etc
- Databases – Oracle, Cassandra, MongoDB, HanaDB, MySQL, etc
- Miscellaneous – RabbitMQ, Consul, SSH and even X.509 v3 certs and CA

While EdgeX, by default runs a single Vault (and Consul for that matter) server, Vault also has the ability to run in a fault tolerant mode in conjunction with Consul. In this mode multiple vault servers and multiple consul servers are run simultaneously.

In EdgeX, Vault will initially be used for secrets storage. Eventually, almost (more on this in the next section) all EdgeX secrets will be stored in Vault including:

- Certificates
- Database credentials
- Cloud/IOT Platform credentials
- Encryption/Signature keys

Origin of Secrets

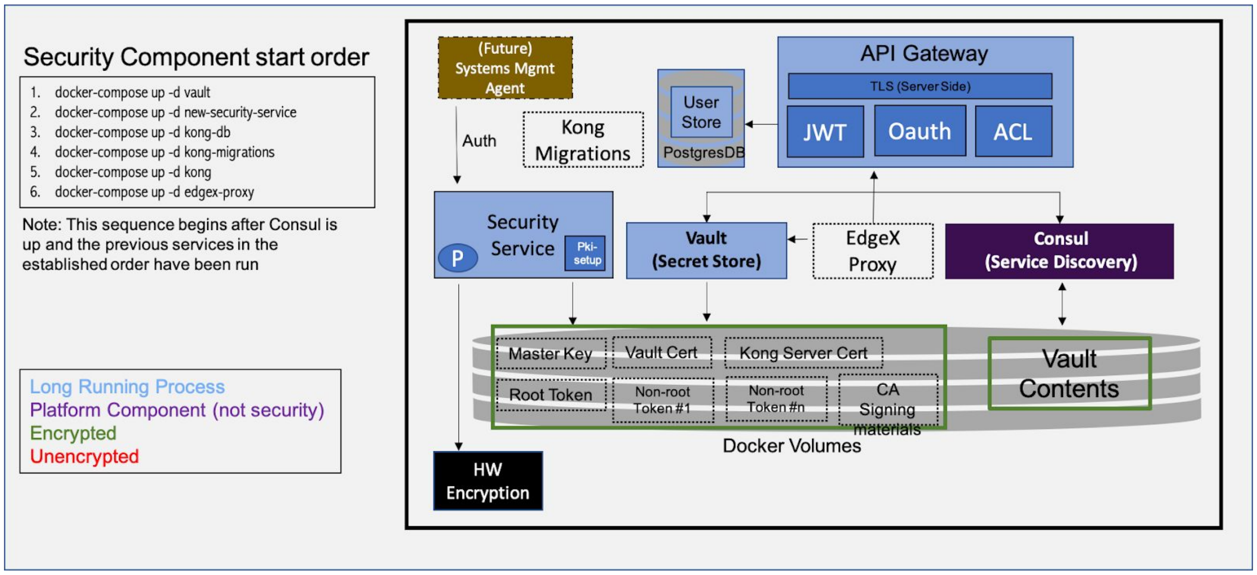
EdgeX secrets in need of encryption via Hardware based secure storage come from the following locations:

- Vault image – All certs are generated inside the Vault image
    - The TLS cert Kong will use on its Admin API port
    - The TLS cert Vault uses for serving the API – remember, Vault cannot store its own TLS cert and associated private key
    - CA Certificate Signing materials
- Vault worker – This process initializes and unseals vault. It also loads the various policies and creates service specific access tokens. As a result, the following secrets are generated:
    - Root Token
    - Non-root tokens (future)
    - Master Key(s)

Usage of Secrets

Initial secrets are only retrieved at start time (both initial and subsequent). This will include, in the future, any individual service that stores secrets in Vault. Anytime a service starts, it must be able to retrieve/store secrets from Vault using a valid token. Therefore, any Vault-consuming service must have a vault token at startup time. Non-root tokens have a Time to Live (TTL) barrier that implies a service to handle the refresh process, TTL cycles are also limited. These features cannot be disabled by Vault configuration, therefore have to be automated within the future usage procedures.

Future State


Security Component start order

```
1.  docker-compose up -d vault
2.  docker-compose up -d new-security-service
3.  docker-compose up -d kong-db
4.  docker-compose up -d kong-migrations
5.  docker-compose up -d kong
6.  docker-compose up -d edgex-proxy
```

Note: This sequence begins after Consul is up and the previous services in the established order have been run

Hardware based Encryption

In order to securely encrypt the initial secrets, Hardware based encryption is a requirement. Otherwise, there is some form of key or secret sitting on the file system. There are multiple existing systems for Hardware based encryption with the most widely used being:

- TPM – The Trusted Platform Module 2.0 specification is provided by the Trusted Computing Group.
    - TPM boards can be manufactured by 3rd party vendors for inclusion on a system board. A list can be found here. Or the TPM can be manufactured and included on the system board by the same vendor such as Intel.
- TEE – Trusted Execution Environment – An implementation that provides a secure enclave or secure OS to execute trusted code in isolation from user space within the CPU.
    - ARM TrustZone is an architecture of the Hardware layer to support this implementation. TrustZone introduced a special CPU mode called “secure mode” in addition to the regular normal mode, the architecture includes the SoC and peripherals that are connected with the SoC.

While the architectures differ significantly, the general principle is that TPM hardware modules are separate from the SoC and TEE are built into the SoC. Both implementations have different ways to perform encryption, hashing, key generation and key storage functions. They also differ in cryptographic inputs like randomness and entropy. Storing a key in hardware provides yet another layer of security over software-based solutions. Simply having access to the filesystem does not provide an attacker an avenue for successfully decrypting data. Although some TEE implementations can provide slices of the filesystem which are themselves isolated and blinded from user space. These zones need supplemental TEE security mechanisms like monotonic clock to avoid alterations on replay and downgrade attacks.

Note: These types of system frequently have small CPU and small storage capacities. As such, they are meant to be used for encryption and storage of small amounts of data on an infrequent basis.

EdgeX usage of Hardware Based Encryption

EdgeX will be using the Encryption Module to encrypt data rather than to store data. The data will be encrypted by the Module then stored on the file system in an encrypted state. For decryption, the data will be sent to the Module for decryption. Upon return of the encrypted data, the data will be used in memory only. Communication with the Encryption Module will be secured by authentication/authorization and eventually supplemented by tunneling techniques.

A new component – Security Service

Communication with a hardware encryption module will require a new security component called the security service. There will be a single component in EdgeX that communicates with the encryption module. This component will perform several functions:

- Retrieve unencrypted secrets
    - This may involve merging in vault-worker functionality (see below)
    - Tokens, keys and certificates must be retrieved. Ideally, there will be a way to do this without the secrets ever residing on the file system unencrypted for even a moment
- Using the plugin, call the hardware encryption module to encrypt the secrets
    - Place the encrypted secrets on the file system.
- Using the plugin, call the hardware encryption module to decrypt the secrets
    - Grab the encrypted secrets from the files system and send them to the module
- Provide an API to return the unencrypted values
    - For example, in the future, the Systems Management Agent may need to call this component to retrieve the Vault tokens to provide to microservices at startup time.
    - The security service must also provide a method for authentication. Only known consumers should be able to call the security service.

It may make sense to merge some of the Delhi security functions into this single security service. This merge will serve the dual goals of simplifying the number of security services and interactions. Those functions are:

- Vault-worker functions
    - Initialize and unseal Vault – tokens and keys will be in memory (not fs)
    - HCL file integrity and authenticity
- Pkisetup
    - Generate all certs – private keys will temporarily be on the fs

The feasibility of this approach will have to be determined during development.

Plugin Architecture

A plugin architecture will be utilized to provide flexibility. An EdgeX design principle is to be language agnostic for service implementation. Therefore, the features of the plugin will be described. The plugin architecture will provide the following features:

- The plugins should be delivered as a binary. This will provide IP protection for 3rd party vendors.
- Externally expose the encrypt and decrypt methods specified below.
- Provide external configuration (if necessary) via standard EdgeX configuration patterns.

Proposal: Create an SDK similar to the Device SDK.

API Specification

Based on the above architecture, the new security component requires two methods provided by the plugin as described below:

Encrypt

The encrypt method should use an interface to provide future flexibility.

The interface
type secret struct { mode string, plaintext string}

mode – encryption mode
plaintext – unencrypted text

The method
func encrypt(secret) string

secret – interface struct that initially contains the encryption mode and the plaintext
return – encrypted string

Decrypt

func decrypt(cyphertext string) string

cyphertext – contains the encrypted string
return – unencrypted string

Authentication and Authorization

The architecture must include authentication and authorization. Hardware based encryption modules have authentication mechanisms. The plugin should satisfy the hardware module authentication and authorization requirements (e.g., pin or password).

Default Plugin

A software only plugin must be implemented for development purposes and situations where hardware encryption is not available.

Possible Future Features

- Expose hardware based encryption to other EdgeX services.
- Implementation with components other than Vault and Consul

Conclusion

While this API definition is very simple, it performs all of the functions required by EdgeX at this time. Remember, almost all EdgeX secrets should be in the secret store (Vault).