Background

Hardware Secure Storage Roadmap

The EdgeX Foundry has called for hardware secure storage:

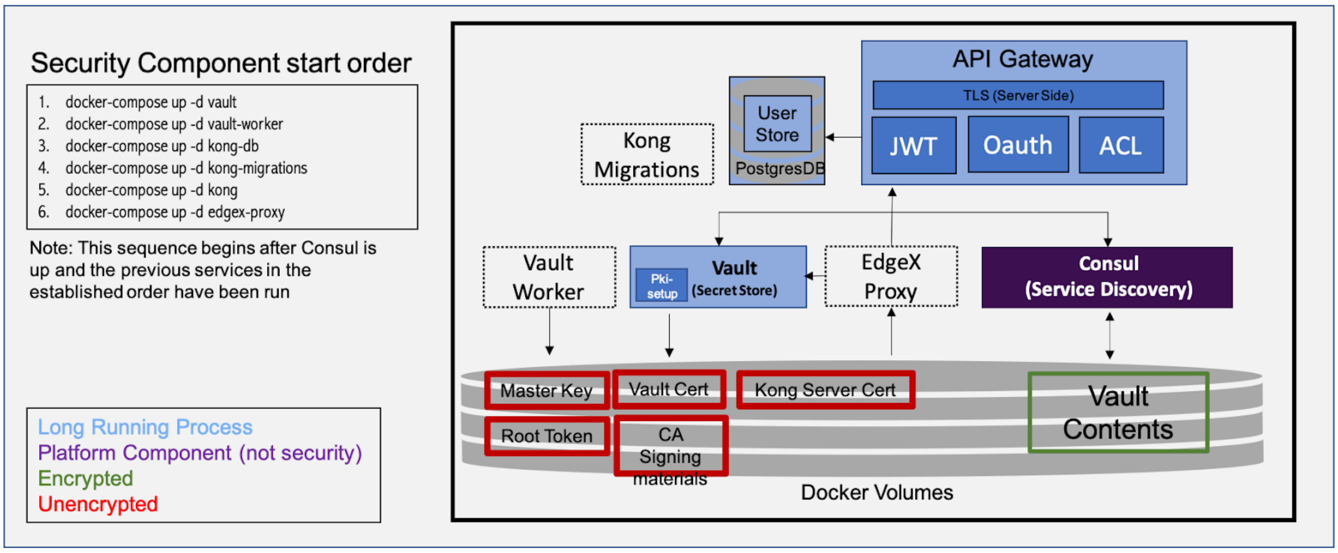
Initial EdgeX secrets (needed to start Vault/Kong) will be encrypted on the file system using a secure storage abstraction layer – allowing other implementations to store these in hardware stores (based on hardware root of trust systems)

- https://www.edgexfoundry.org/blog/2018/11/15/edgex-foundry-releases-delhi-and-plans-for-edinburgh/
- https://wiki.edgexfoundry.org/display/FA/Edinburgh+Release

Hardware Secure Storage Draft Proposal

The current state of secret storage is described in the Hardware Secure Storage Draft (https://docs.google.com/document/d/1MsTNdwtZp3zA-nPhCC3COakL3e5mrhJuFByy6ja5OxU/edit).

The AS-IS architecture resembles the following diagram:

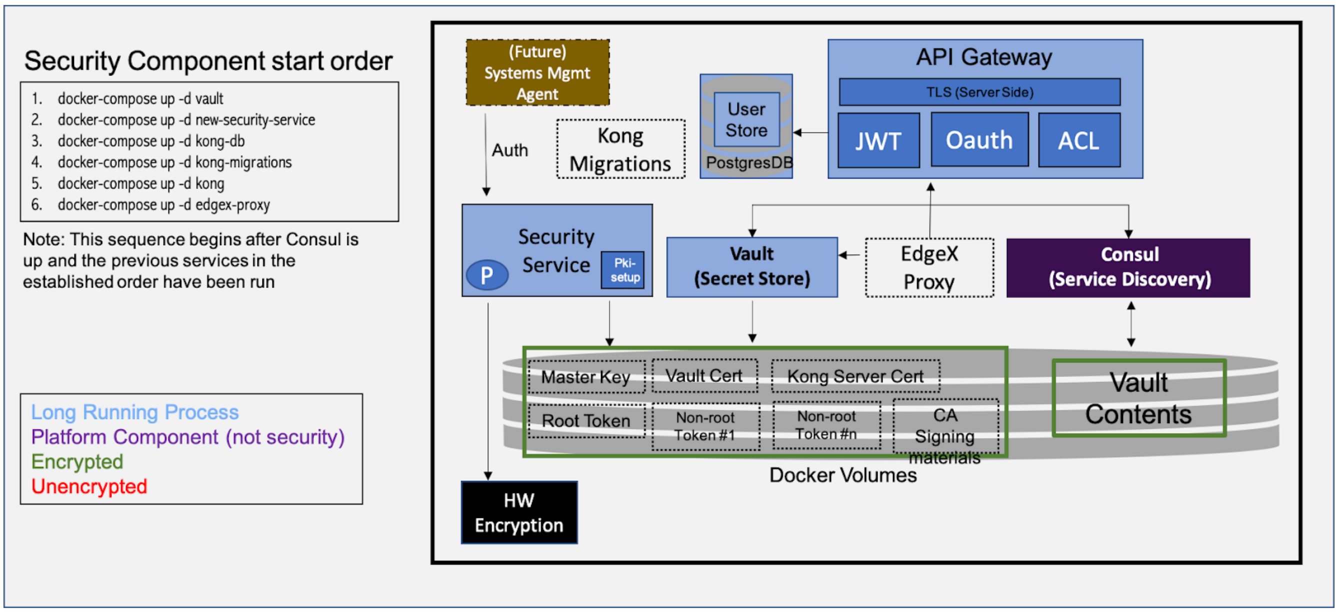


```
1. docker-compose up -d vault
2. docker-compose up -d vault-worker
3. docker-compose up -d kong-db
4. docker-compose up -d kong-migrations
5. docker-compose up -d kong
6. docker-compose up -d edgex-proxy
```

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

As the diagram notes, the critical secrets for securing the entire on-device infrastructure sit unencrypted on bulk storage media. That the Vault contents is shown as encrypted is misleading: the master key to decrypt the vault is in plaintext nearby. Moreover, the current state is actually worse pictured, as some of these root secrets, in the containerized incarnation of the EdgeX Framework, are mistakenly included in the container images themselves.

The Hardware Secure Storage Draft proposes the following 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

This future state proposes a security service that can encrypt the currently unencrypted data items. The proposal itself, however, has a number of unresolved flaws:

First, the diagram is not sufficiently detailed to cover initialization order. In a container-based system, the security service is not in a position to directly start its dependencies. Note the prescribed start order box on the left of the diagram. The proper way to handle initialization order is for containers to block startup until their prerequisites have been satisfied.

Secondly, the security service depends on an encryption plugin that implements a simple encrypt/decrypt interface. The community is hesitant to mandate a particular plugin architecture, such as a dynamic load C module, or a Golang loadable module, as it would introduce a language dependency. It is obviously not safe for the plugin to just decrypt any secret passed to it, but to do otherwise would mandate some kind of authentication mechanism such as a pin or password, which is problematic for a completely headless device. TPM's in particular have very flexible polices to determine whether or not a secret can be released, including local attestation of a known machine state. In general, without some root secrets already established, it is not possible for the security service or the plugin itself to authenticate its caller.

Third, the proposed solution overlooks the possibility of a hardware-rooted PKI for secure on-device communications. A hardware-rooted PKI is also useful for establishing an unique, immutable, and non-cloneable device identity, and for establishing protected channels to allow messages to be sent to the device that only the device can decrypt.

Fourth, the the security service has too many responsibilities and is likely to become overly complex over time. It is responsible for PKI generation, Vault initialization, and generating non-root Vault tokens for services. This will complicate the development and maintenance of the security service. The centralization of security-critical functions also means that any implementation bug in the security service is potentially a CVE-level issue. It also increases the likelihood that the entire process need be long-lived with an attackable surface area.

Lastly, the larger issue of distribution of Vault tokens to services is completely unaddressed.

A New Hardware Secure Storage Proposal

The replacement proposal is not particularly aimed at encrypting the currently unencrypted root secrets present in the Delhi release of EdgeX Framework. Instead, the new proposal is derived from a formal threat model to meet some critical security objectives, which include, among others:

- Ensure confidentiality, integrity, and availability of application secrets.
- Prevent off-line copying of application secrets by binding them to secure hardware storage.

The design of the proposed solution also seeks to address software complexity by defining components in terms of their roles and responsibilities in the context of the threat model rather than trying to define a one-side-fits-all model. In container-based architectures, the choice between software-only and various forms of hardware-backed security can be made at the container level: as long as the container fulfills its roles and responsibilities in a secure manner, the resulting system will meet its security objectives and continue to evolve over time. The proposed design also seeks to resolve some of the early-startup-phase initialization-order problems in the Delhi release of the framework.