ONAP Security using trusted solutions

Intel & Tech Mahindra

Agenda

- Threats overview and Mitigations
- Certificate Management
- Secret Management

Typical Threats in Micro Service Architecture

Threats
Credential Stealing -- Attacker stealing passwords and Certificate private keys • From images (containers/VMs) – Files in images having credentials • From captured traffic • From ephemeral storage disk (swap disk, local disk for temporary storage) • From memory (After breaking into some vulnerable application) • From log files that log environment variables • Images using default user names and password
Denial of Service Attacks • Resource exhausts • Kernel exploits
Tampering • OS/VMM Tampering (by installing sniffing tools) • Container/VM tampering • Attacker images resembling genuine images
Data Stealing – Attacker stealing data that could be confidential, privacy information etc.… • From databases, file systems, thrown-away disks etc.…

Background – Attack examples that happened

Heart bleed attack – Vulnerability in OpenSSL, that expose private keys.
How widespread is this?
The most notable software using OpenSSL are the open source web servers like Apache and nginx. The combined market share of just those two out of the active sites on the Internet was over 66%

https://www.symantec.com/connect/blogs/how-attackers-steal-private-keys-digital-certificates

- Backdoor.Beasty
- Infostealer.Snifula
- Downloader.Parshell
- Trojan.Spyeye
- W32.Cridex
- W32.Qakbot
- Infostealer.Shiz
- Trojan.Carberp
- Trojan.Zbot

Onelogin breach – Key stealing in addition to authentication breach:

https://krebsonsecurity.com/2017/06/onelogin-breach-exposed-ability-to-decrypt-data/

https://blog.netspi.com/stealing-unencrypted-ssh-agent-keys-from-memory/ - *tools to extract keys from memory.*

https://wiki.xenproject.org/wiki/Virtual_Machine_Introspection - *Libvmi that enable VMM Software to look into the VM memory – Can be used to extract VM secrets.*

Implications : Financial loss and brand loss from web site spoofing, account misuse, money withdrawals and huge fines.

Mitigations

Mitigations
Threat : Credential Stealing -- Attacker stealing passwords and Certificate private keys Mitigation : - Enabling secure communication among services - Avoiding clear passwords and private keys using secure (HW) techniques. (HW Security include – usage of TPM, SGX, TZ etc…) - Avoid using default passwords - Avoid storing clear passwords and private keys in images - Avoid passing secrets/password using environment variables and using cloud-init user-data scripts
Threat : Denial of Service Attacks • Usage of various security techniques such as SECOMP, App-Armor, SELinux to filter out systems calls, provide mandatory access control for every process in containers. • Usage of cgroups for all shared resources.
Tampering • Secure boot, Measured boot with attestation. • Signed images
Data Stealing • Encryption of data while storing (in DB, files) and keeping encryption keys secure.

Focus Today

Mitigations
Threat : Credential Stealing -- Attacker stealing passwords and Certificate private keys Mitigation : - Enabling secure communication among services - Avoiding clear passwords and private keys using secure (HW) techniques. (HW Security include – usage of TPM, SGX, TZ etc…) - Avoid using default passwords - Avoid storing clear passwords and private keys in images - Avoid passing secrets/password using environment variables and using cloud-init user-data scripts
Threat : Denial of Service Attacks • Usage of various security techniques such as SECOMP, App-Armor, SELinux to filter out systems calls, provide mandatory access control for every process in containers. • Usage of cgroups for all shared resources.
Tampering • Secure boot, Measured boot with attestation. • Signed images
Data Stealing • Encryption of data while storing (in DB, files) and keeping encryption keys secure.

HW Security properties

TPM/SGX/TZ – Trusted Execution Environment
HW level fuses – Root keys are different for each chip, Can't be read by anybody other than the TEE, Factory level settings, Can't be modified.
Capability exposed: - Can generate keys internally (RSA, ECDSA, symmetric keys) – Keys are secured by parent key. - Perform crypto operations within the TEE (Key is never exposed) - Migration support (for high availability) - Even system administrator can't get the keys from the TEEs (If there is any breach, at the most it is possible to use the keys, but attacker can never steal the keys)
TPM: Standards created by TCG. Fixed functions as defined by standard. - Used also for attestation (Quote generation)
SGX/TZ : Programmable, can even be used for any application specific encryption and encoding. SGX: Possible to multiple TEEs (One for each container/VM or even process).

Certificate Management Service

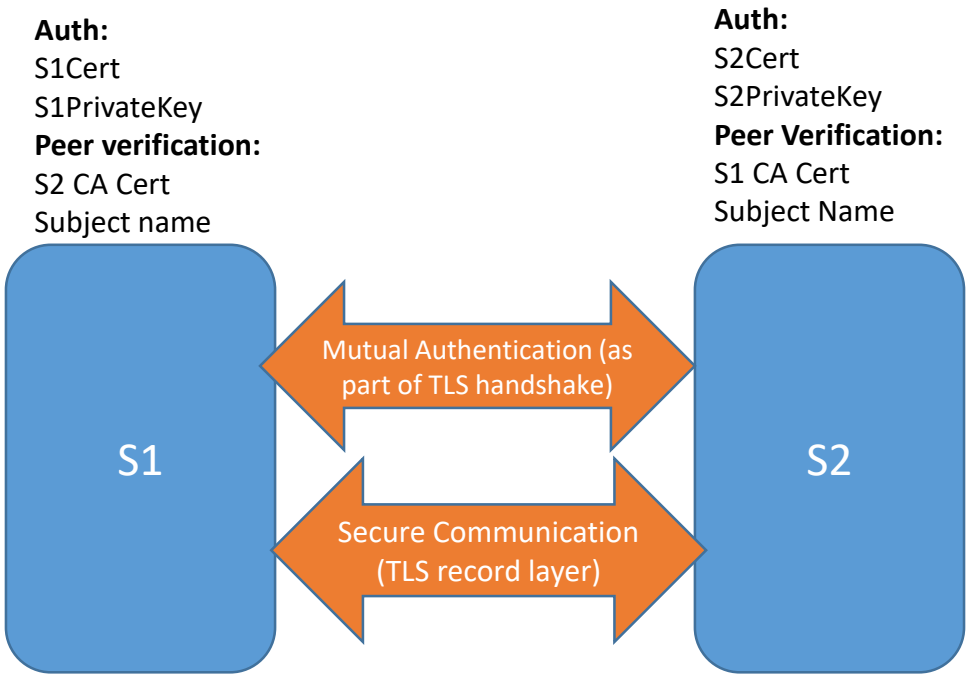
Secure Communication – Need for Mutual TLS (with Certificate based authentication)

Background
ONAP consists of multiple micro services
Two types of communication among micro services – REST API based and DMAPP publish/subscriber based communication.
Both use TCP transport
Hence TLS 1.2+ secure communication is good enough. No need for others such as IPSEC.

Need
Protect bad actors stealing the data on the wire Protect from receiving messages from bad actors -→ Secure communication among services that provide • Mutual Authentication of end points (Mutual TLS) • Confidentiality, Integrity and non-repudiation of transport → TLS1.2+ based transport (HTTPS)
Secure DMAPP messages using TLS1.2+ between brokers to publishers/subscribers. Possible End-to-End encryption/integrity of the data between publishers and subscribers.

How does Mutual TLS work?

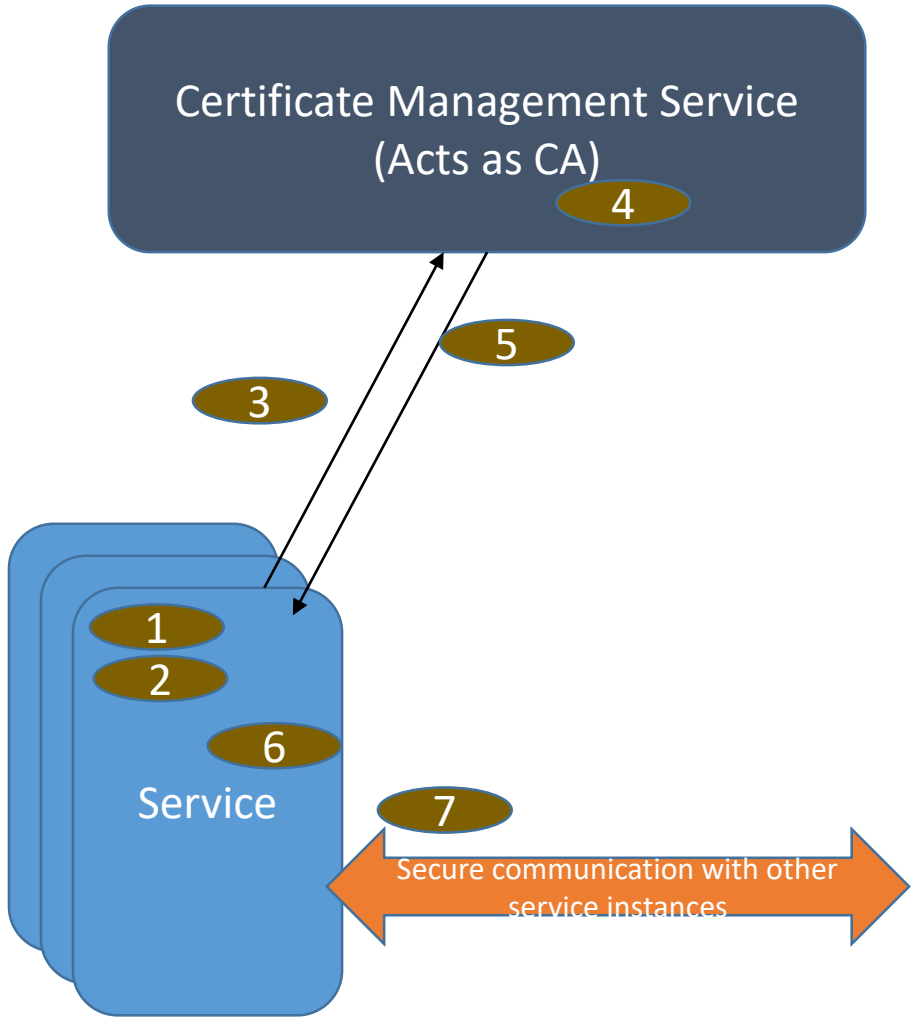
Java HTTPS library and underlying TLS1.2+ classes do support Mutual TLS (Certificate based authentication)
But, it requires certificate provisioning on each end point.



Each endpoint is expected to have private key and certificate with public key signed by CA. This information is used to authenticate itself with the peer.

It is also expected to have CA Certificate and subject names to verify the peer when presented with its certificate.

Best Practices of Certificate provisioning – End point initiated

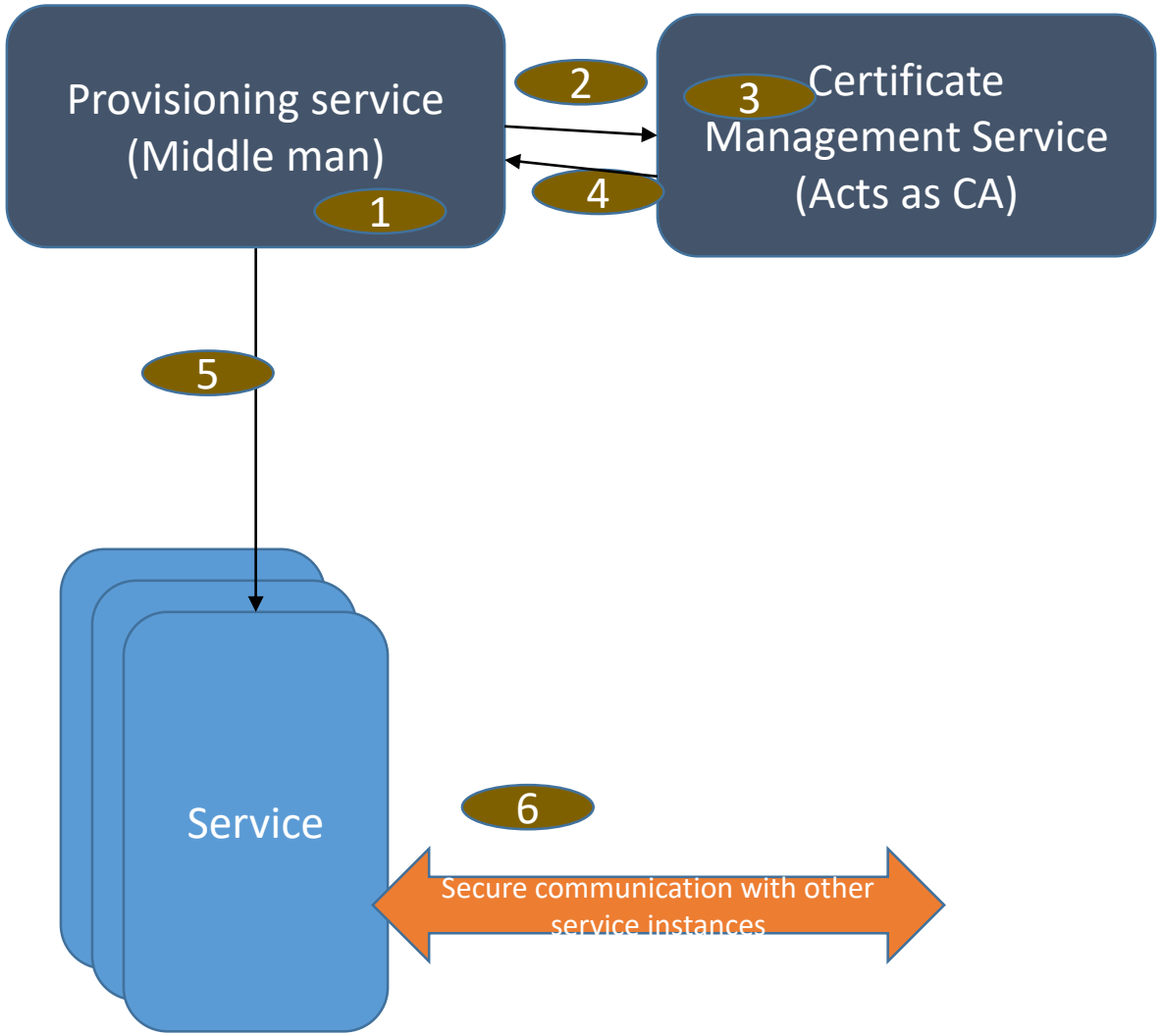


In Micro Services, before it communicates with other micro services, it needs to get certificate enrolled by CA.

1. Typically at startup, generates an RSA/ECDSA public/private key pair.
2. Generates PKCS10 CSR (Certificate request) – Which involves signing the message with private key.
3. Request Certificate by sending PKCS10 request to CA.
4. CA verifies that genuine service is requesting for certificate, verifies PKCS10 request, generates X.509v3 certificate, signs it using CA certificate-private key.
5. Sends signed X.509v3 certificate and CA certificate.
6. Service stores the information.
7. It uses this information during TLS handshake to establish secure communication channels.

- Good for Micro Services that are long lived.
- Possibility of keeping the private key secure within container service.
- Good for any workloads (Local/remote)
- Slow startup – Need for certificate auto provisioning.

Best Practices of Certificate provisioning – Middle Man initiated



As part of service instantiation, provisioning service provisions certificate and private key (either via environment variables, volume backend, cloud-init)

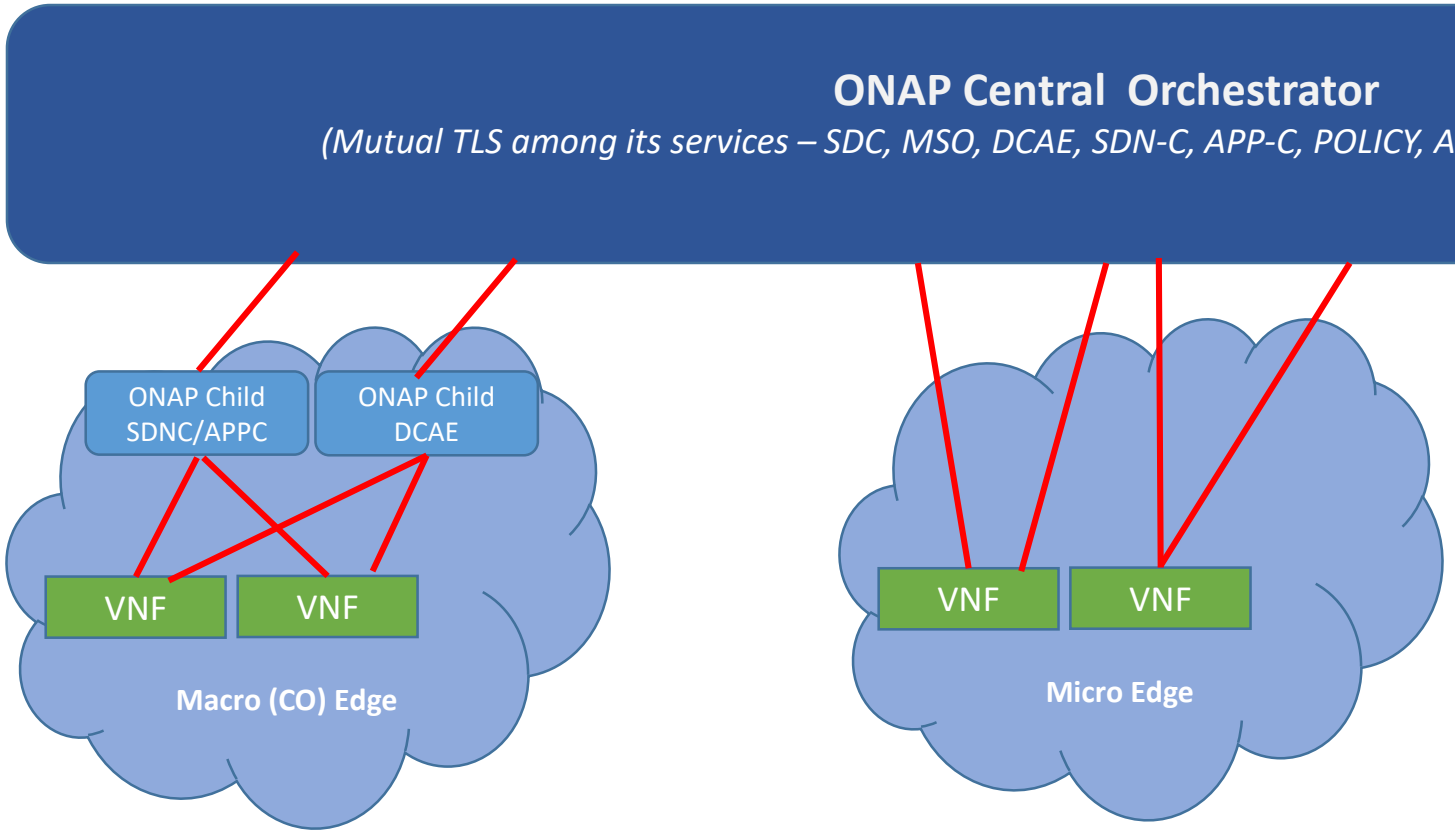
Provisioning Service is deployed with token/authentication credentials with permissions that allow CA to issue certificates.

For Every Service instance

1. Generates RSA/ECDSA key pair, generates subject name, PKCS10 CSR
2. Request signed certificate by sending CSR to CMS
3. CMS verifies the token/auth-credentials, generates x.509v3 certificate and signs it using CA private key
4. CMS sends the x.509v3 certificate to Provisioning service
5. Provisioning service sends certificate + private key to service.
6. Service uses these credentials to create Mutual TLS session with others

- Good for Micro services that are short lived and where startup performance is very important.
- Disadvantage : Keys are clear

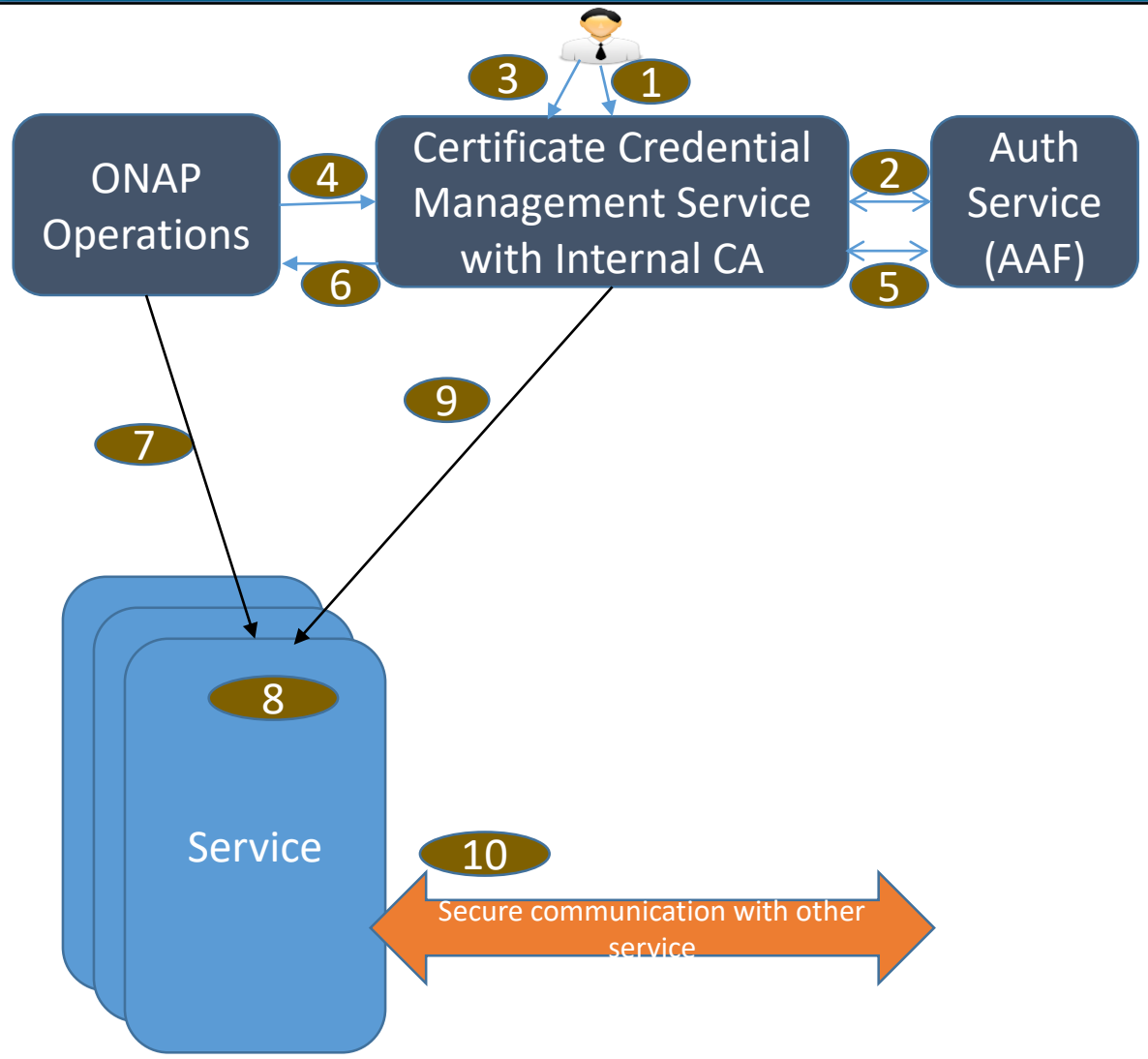
Mutual TLS opportunities among ONAP Components



- Secure communications among ONAP components in central location
- Secure communication between ONAP Child components in edges to central orchestrator services.
- Secure communication between VNFs and ONAP services (e.g. VES, Netconf)

All require certificate provisioning

Certificate provisioning Proposal – High level flow in ONAP



It is expected that Certificate management and Auth services are brought up first.

Creating CA Instance (Typically once per deployment)

1. Admin user creates authentication session by passing username/password OR grant token given by Auth Service.
2. CMS validates the Admin user credentials using Auth Service
3. Admin user instructs CMS to create self signed CA.

ONAP (Rest of) Service instance bring up (Getting the CMS-token)

4. Operations Admin provides username/password or Auth grant token to CMS to get the CMS-token.
5. CMS validates the credentials with Auth Service.
6. CMS returns the CMS-token.
7. CMS-token is passed to Service as part of its bring up

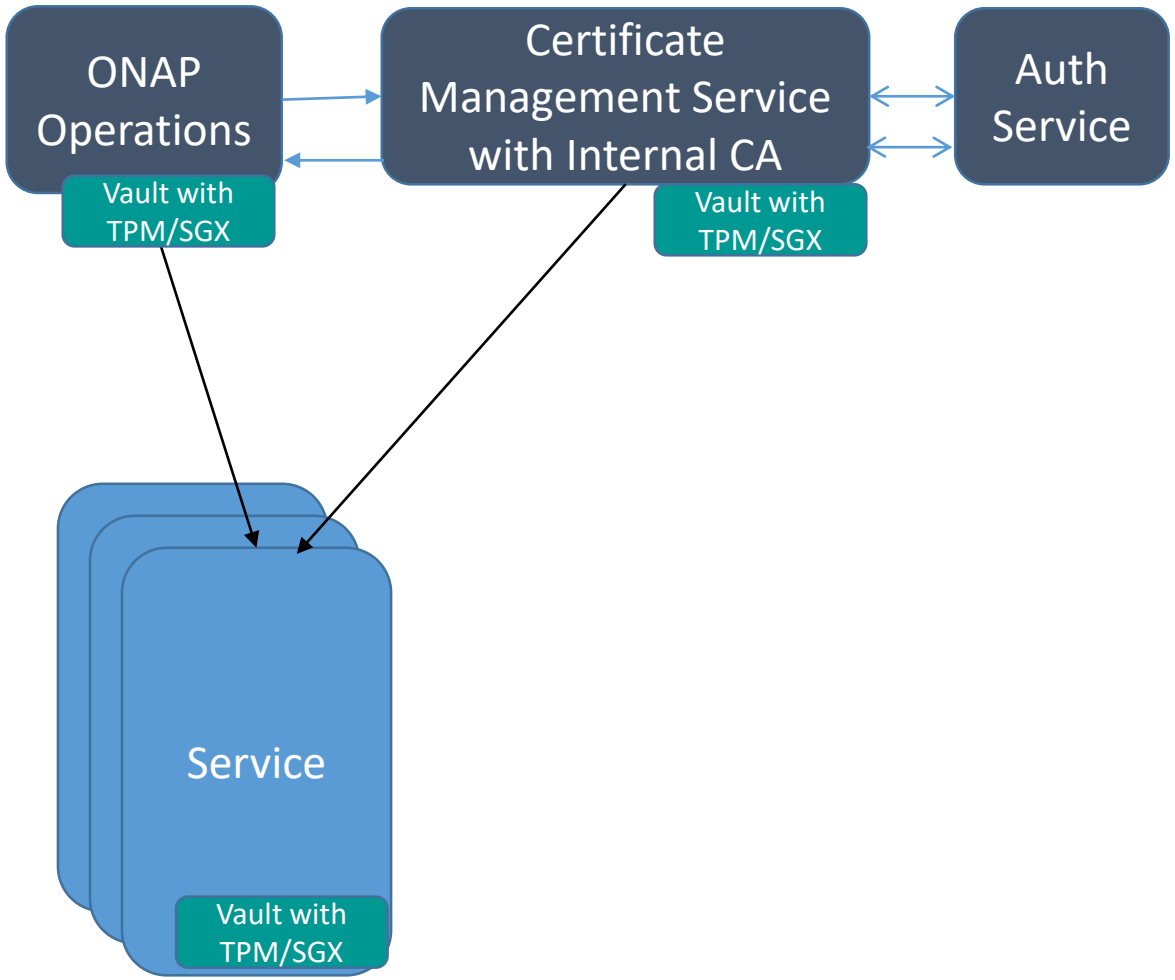
ONAP Service Certificate provisioning (Service startup time)

8. As part of bring up, generates RSA/ECDSA key pair and PKCS10 (CSR) request.
9. Sends PKCS10 CSR with CMS-token to CMS. CMS validates the token. If valid generates certificate and signs it with its CA. Returns certificate and CA certificate.

ONAP TLS Communication

10. ONAP service uses certificate and private key during TLS handshake

Answers to Security Concerns



In-Transit security

- TLS 1.2 (even for communication among CMS-to-Auth, ONAP Operations-to-CMS, Service-to-CMS etc.…

Authentication:

- Between Service-to-CMS : CMS-token, Certificate of CMS
- Between ONAP Operation-to-CMS : Grant-token given by Auth-Service/UN+PWD and Certificate of CMS.
- Usage count of CMS-token to 1 and very low life time.

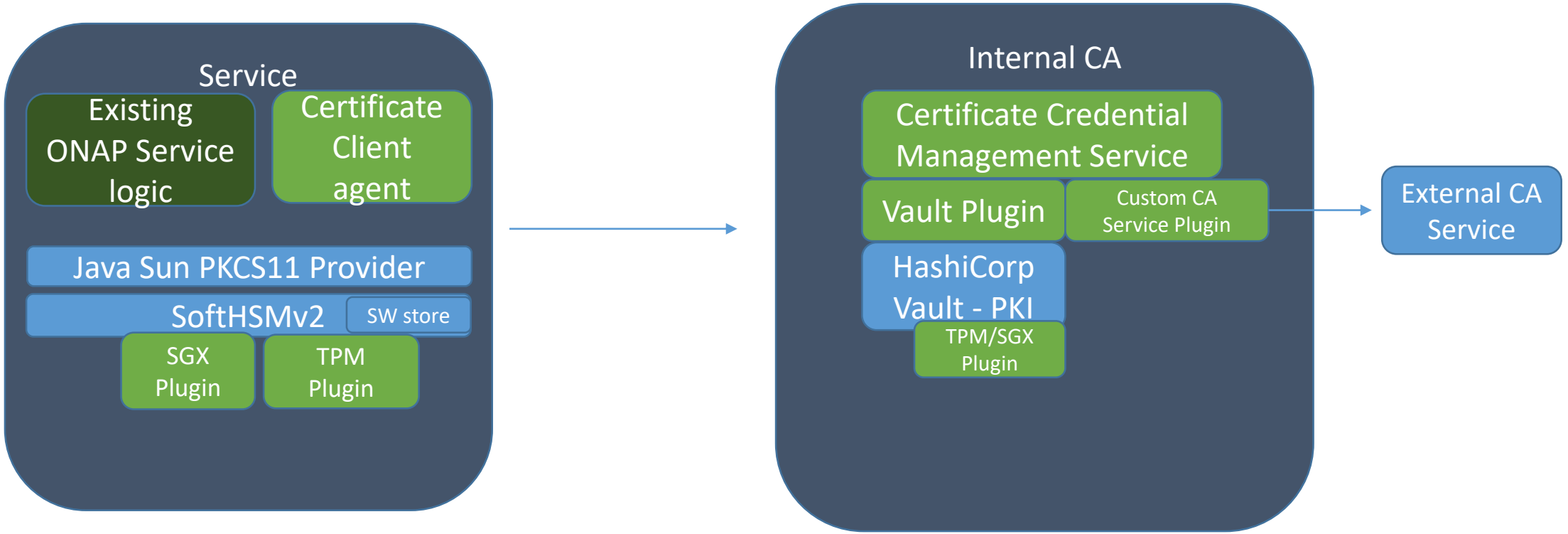
Additional Authentication:

- Using MAC addresses/IP addresses

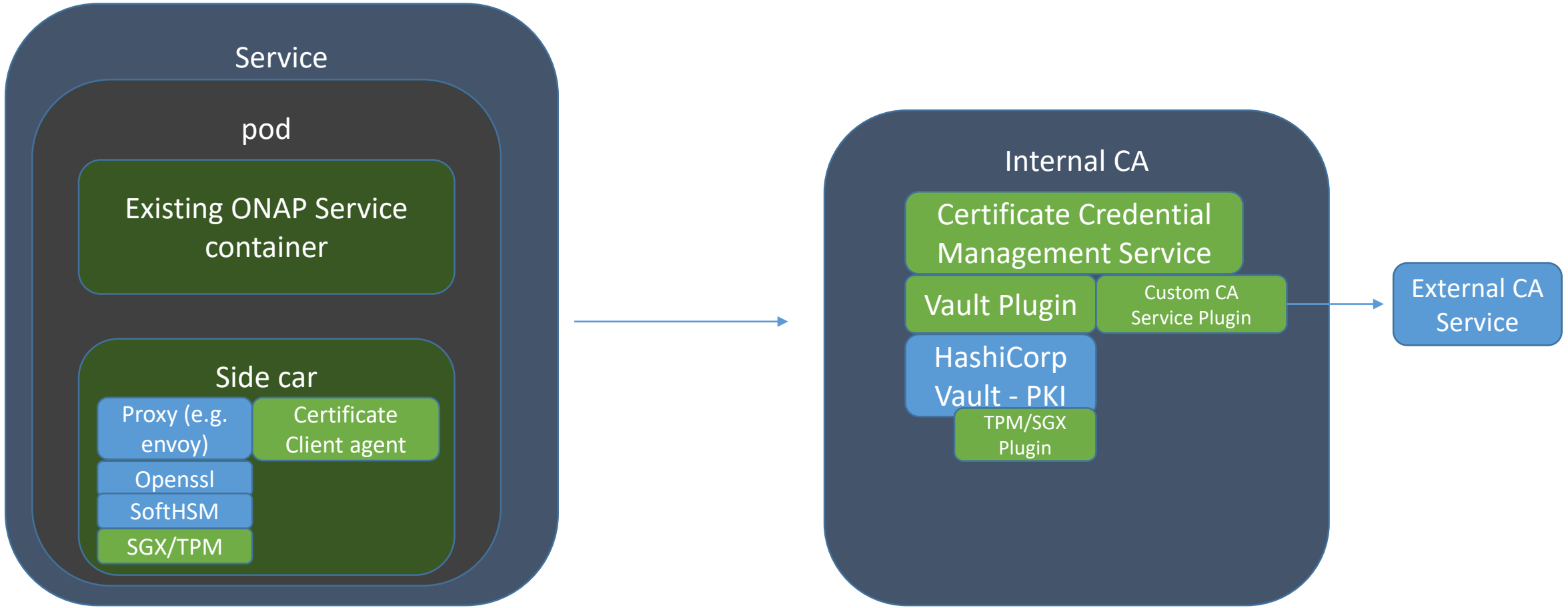
Security of Private keys + UN+PWD:

- HW based Security : TPM/SGX/TZ.
- Private key is not exposed even in memory.

Certificate Credential Management : Architecture Blocks



Certificate Credential Management : Architecture Blocks with sidecar



Internal Certificate Management service- Requirements

1. RESTful API support for Certificate request agents
   - Generate Certificate request
   - Revocation status request
   - Usage report update
   - Token Authentication
2. Admin interface
   - Generate self signed CA
   - Upload CA cert + CA private key (In PEM/DER)
   - Get usage report on per key
   - Revoke certificate
   - Get CA Certificate in PEM/DER format.
   - Token Service to provide temporary tokens
3. Authenticate user using AAF
4. Role based access control using AAF
5. Settings using Distributed KV Store
6. Service registration using MSB
7. Reports and Logs
8. GUI/CLI support using Portal and CLI

- Security of CA private key – Using TPM/SGX/Trustzone if available.
- Ability to add new CA plugins to talk to deployment specific CA service.
- policy support such as algorithms allowed(For example to support NIST specified algorithms such as RSA 3K)

- Future:
  - Multiple CA instances
  - Validation of Genuine HW of client agents
  - Validation of attested OS
  - SCEP Support for Certificate enrollment.
  - OCSP support for Revocation status.
  - High availability of CMS/CA.
  - Additional Authentication mechanisms – JWT, AWS, GCP, K8S Service accounts, vendor cert/key etc…

Certificate request agent requirements

1. Ability to generate RSA/ECDSA key pair using PKCS11 interface
2. Secure storage of private key : Ability to use HW security under PKCS11 if TPM/SGX is available.
3. PKCS10 CSR generation
4. Communication with CA over REST API
5. Java Client and Python Client support
6. Periodic generation of usage report.
7. Service discovery of Certificate Credential Management service.
8. Certificate renewal

Secret Management Service

Secret Service - Need

Background
Many services still use password based authentication – Various database servers, Publish/Subscribe brokers, Log Service, VIM and many more…
Passwords are stored in files in many services.
With many services and service instances, attack surface area is becoming very big.

Need
Avoid passwords sprawl in files.
Avoid passing passwords via environment variables to services
Avoid clear passwords in storage.

Secret Server is nothing new….

K8S Orchestrator has built in Secret Server

- As an admin/user, create a secret in secret server, get the secret reference.
- Puts these references in yaml files instead of actual secrets.
- K8 controller on demand basis (while bringing up the POD), creates file system volumes with secrets using secret references.
- Application in Pods read the secrets from volumes.
- Volumes disappear when POD exits.

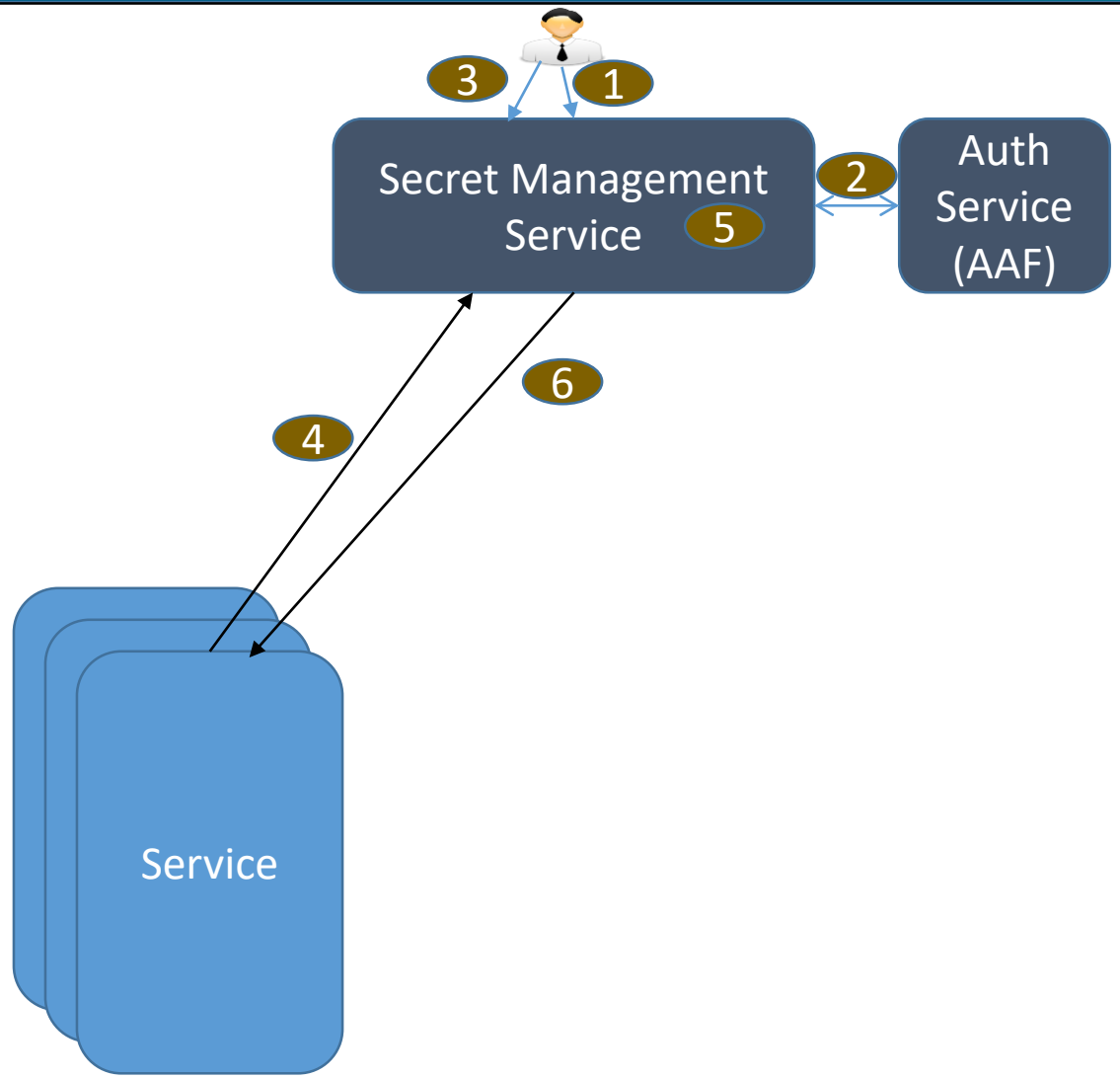
Openstack deployments typically have Barbican to store secrets.

- Secrets can be created using Barbican APIs. Reference to secret is returned.
- These references are passed to Openstack services.
- Secrets can be retrieved by calling Barbican APIs by services on demand basis.

Secret Service - High level Requirements

1. Shall be able to support multiple secret domains. (New key for every secret domain)
2. For each domain, multiple secrets can be stored.
3. Each secret can have multiple key value pairs
4. Each domain is associated with various policies.
5. Configuration of roles to permissions (policies)
6. Configuration of subject names (with wildcard, prefix) to permissions (policies)
7. Authenticate users with AAF
8. Certificate authentication
9. Issuing token with permissions (assigning policies that this token has permission to) upon successful authentication.
10. Key life cycle management (Keys used for every secret domain) – Key rotation, Key expiry, Key archival, Key deletion/destruction.

Secret Server – High level flow in ONAP



It is expected that Certificate management and Auth services are brought up first.

Creating Secret Domain

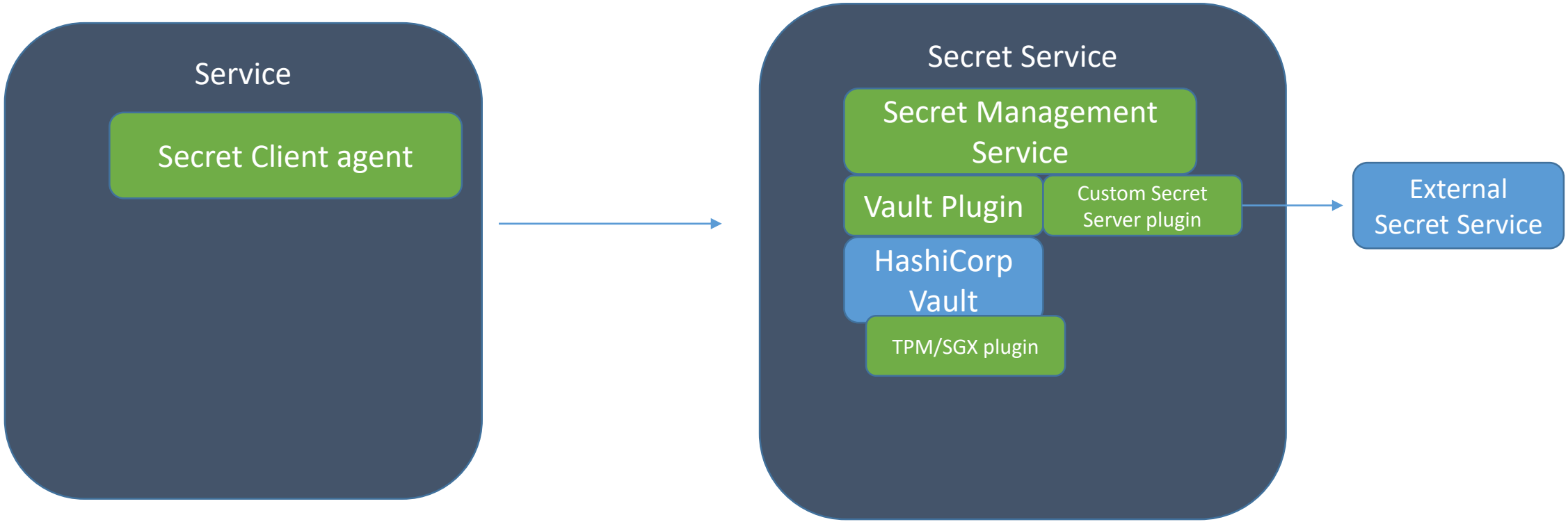
1. Admin user creates authentication session by passing username/password OR grant token given by Auth Service.
2. SMS validates the Admin user credentials using Auth Service
3. Admin user instructs SMS to create secret domain (Policies, CA-Cert, Set of Subject name prefix to allow vs permissions)

Service instance bring up (Assumes that service already got the certificate provisioned)

4. Service makes Secret service request via TLS. (Create Secret/Read Secret)
5. SMS validates the certificate credentials, if valid and if policy allows it, perform the operation. Also, create and return SMS-token for future operations.
6. SMS returns the results

Service uses secrets for service specific processing

Secret Service: Architecture Blocks



Q&A

Why not use istio

Istio (istio.io) is service mesh project, has all the features required for service/service-instance with versions/ registration, discovery, load balancing, Mutual TLS security with certificate credential provisioning

Challenges with istio:

- Yet to mature.
- Closely works with Kubernetes – Since ONAP may be also orchestrated via Openstack and others, this could be impediment.
- Security does not start from application : There is sidecar container in every POD which runs envoy HTTP proxy. Security starts from the envoy. Communication data between application container and sidecar container is in clear.
- CA software – Seems basic, may be good enough. Also, need to check how CA private key is stored securely.
- Support for new protocols (beyond HTTP) – Needs to upgrades to Envoy container

What are secret backend candidates

Choices:
- K8S Secret Server,
- Barbican
- Vault

K8S Secret Server:
- Tightly integrated with K8S, if ONAP needs to be brought up using other orchestrators, then this could be an issue.
- Secrets in the secret service are not protected.

Barbican :
- Good architecture, but….
- But only used in Openstack, Maturity/Scalability may be a concern. Community support is limited

Vault: (Recommended)
- Good architecture and well supported
- Widely used in multiple projects.
- Quorum based master key, but can be enhanced to use TPM/SGX/HSMs for hardware level security.

Secret Service - Requirements

1. RESTful API support
   - Adding secrets
   - Deletion of secrets
   - Update secrets
   - Token Authentication for above operations
   - User name/password based authentication too
3. Authenticate user using AAF
4. Role based access control using AAF
5. Settings using Distributed KV Store
6. Service registration using MSB
7. Reports and Logs
8. GUI/CLI support using Portal and CLI

- Security of secrets via AES encryption
- Usage of AES-256 based master key – Plugin approach to secure master key (e.g.. TPM/SGX/HSM/TrustZone)