Workload Identity Solutions

Table of Content

Terminology

1. Key-pair: A public-key and the corresponding private-key as required in any PKI based system
2. CSR (Certificate Signing Request): A CSR is a message sent from an applicant to a registration authority of the PKI in order to apply for a digital identity certificate. It usually contains the public key for which the certificate should be issued, identifying information (such as a domain name) and integrity protection (e.g., a digital signature).
3. TPM (Trusted Platform Module)

Need for Workload Identity

Applications talk to each other, more often than anything else, to get the work done. Applications typically have a fixed pattern of communication, for e.g, a backend server will most likely talk to a database server, a log server and take ingress traffic from the frontend server. These fixed patterns of communications usually translate to access control rules and setting up these access control/authorization rules requires one to identify these applications aka workloads. The Principle of Least Privilege (PoLP) states that every workload must be able to access **only** the information and resources that are necessary for its legitimate purpose. Thus the workloads need to be identified at granular level with a **unique and entire** set of attributes so that the authorization frameworks can have the flexibility to put in the access control rules as desired.

Identity and Zero-Trust (Identity boundary = Trust boundary)

Zero-trust requires that the services within the network use strong authentication and authorization policies and ensure that the principle of least privilege is effectively followed even for the east-west traffic and even for the services within a particular application provider space. A strong Identity solution forms the basis to achieve this.

Characteristics of a good Identity solution

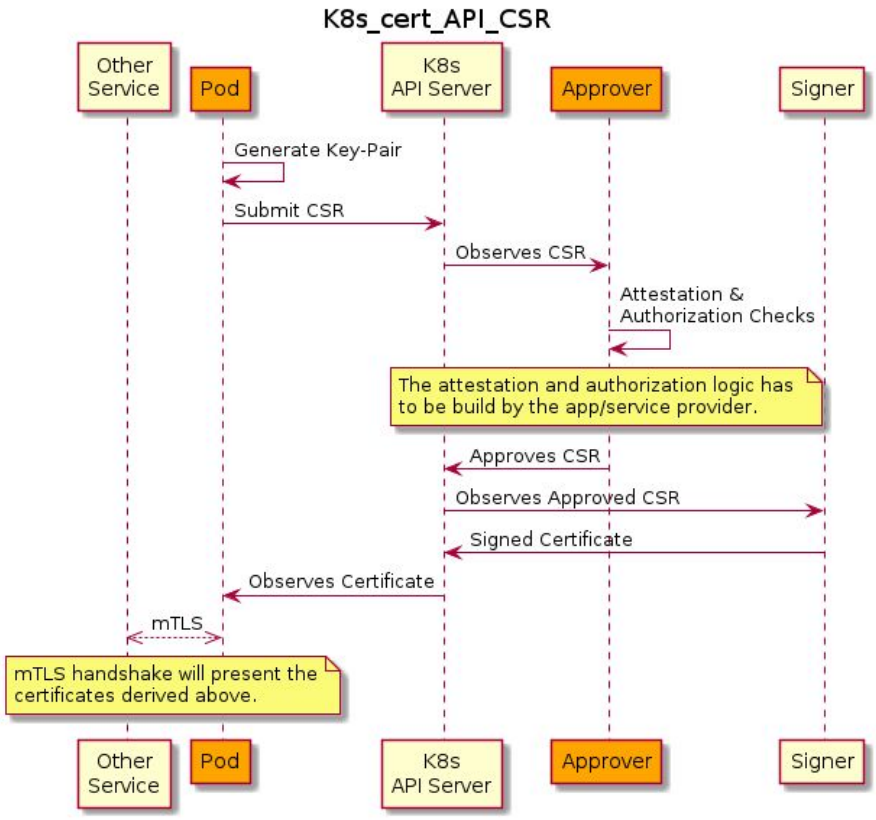
1. Strong Authentication
    a. Unforgeability of Identity
    b. Blast-radius? Impact of a node compromise
    c. Minting and isolation properties of keys.
        i. Are the private keys minted locally and kept locally?
2. Extensibility/Flexibility
    a. Ability to add new service/pod attributes in the future
        i. Will it be possible to differentiate multiple pods within the same service across different regions?
    b. Granularity, Can the identity system be extended to containers within the pod?
    c. Ability to include non-k8s workloads
    d. Single-side support
    e. Identity solution is application agnostic
        i. Can the identity system be used flexibly across any application or application-carrier protocols such as IPSec, WireGuard, TLS?
        ii. Can the identity system be used for another purpose (for e.g, Secrets Management)?
3. Ease of integration
    a. Ease of multi-cluster deployment
    b. Identity federation. Using the same identity system across multiple service providers.
    c. Ability to integrate with TPMs

K8s primitives for Workload Identity

K8s provides few primitives to identify such workloads namely, K8s Certificate API and K8s service accounts.
Note that there are apps who make use of baked secrets for Application Identification and the secrets are stored as part of K8s secrets. There are also cases which make use of hostnames + IP addresses as a means for identification. Such mechanisms obviously have a lot of flaws and are not discussed in this document.

K8s Certificate APIs

These APIs allow a pod to send a CSR based on a locally generated key-pair. The following call-flow shows how a pod can acquire a signed certificate using k8s certificate APIs.



The signed certificate eventually can be used for the mTLS handshake and at the end of the handshake the pods could use the certificate information to authorize the flow.
The application developer will have to handle following aspects of the design:

1. Defining an “Approver”. On what basis will the approver attest or authorize the pod’s CSRs? Here the service provider has to put in a strong attestation framework which would authorize and allow certificate signing. The security promise of the identity will depend on this attestation framework.
2. What part of the certificate would the pods use to identify each other? Pods can belong to certain nodes, cloud-service-provider, location, namespace etc and this needs to be

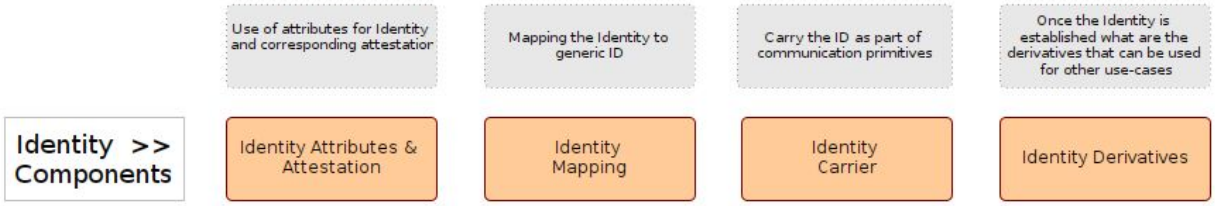
carried as part of the certificates. The application developer has to define this aspect and make sure that all the entities adhere to this certificate format.
3. Handling key rotations. The application developer has to put in place a system to rotate the keys on a short-lived, periodic basis.

K8s Service Accounts

K8s service accounts is the popular choice in k8s-orchestrated pods for service based identity. As an application developer, you could generate individual IAM service accounts for each application, and then download and store the keys as a Kubernetes secret that you manually rotate. Every service account has its corresponding certificate which would be used for service authentication. Service account keys expire every 10 years by default. Thus if you want short-lived secrets then you need to have a system to manually rotate the secrets. The possibility of long-term key exposure and the management overhead of key inventory and rotation usually is a major hindrance in the security promise of this system. The onus of handling security falls on the shoulders of application developers who might be less equipped to handle the security primitives with their corresponding best practices. K8s ServiceAccountToken allows you to generate a temporary token from a Service Account which can be configured to have a limited lifetime post which the token will be refreshed.
Service Account based solution also suffers from other problems:

1. Service accounts may not take into consideration all the attributes of the workload. For e.g, if an application developer has clusters across multiple regions then the workloads could be spawned across these regions. A service account based identity cannot provide a mechanism for finer attributes such as location of the pod to handle the authorization.
2. No server authentication!
3. Dependency on the k8s control plane. The validation of tokens needs to be done with the K8s API server which means the data-plane has an impact if there is any control plane downtime.

Components of an Identity Solution



A. Identity Attributes and Attestation
   Every Identity system depends on a set of attributes and attestation of those attributes. Sometimes the attribute attestation may not be an explicit procedure (for e.g, in the case of Cilium, Aporeto) and is done implicitly for e.g, using a set of labels in k8s/docker environment. The attestation procedure ensures that the enlisted attributes indeed belong to the workload which claims it.
B. Identity Mapping
   The set of attributes may be mapped to an intermediate representation which essentially serves as “the ID”.
C. Identity Carrier
   The application network connection needs to communicate the ID to the remote peer.
D. Identity Derivatives
   The ID attestation procedure might eventually result in derivation of other credentials (such as certs) which could be used for other applications. For e.g, the certificates associated with the ID can be used for mTLS or for IPSec based authentication procedures.

Cilium’s Identity Solution

Cilium operates as a CNI and is a k8s-native solution. The identity of an endpoint is derived based on the Labels associated with the pod or container and is given a cluster wide unique identifier of length 24 bits. The etcd manages this mapping of 24 bit ID to the label set and is synchronized in the cluster nodes. Cilium performs authorization checks on a per-packet basis directly in the Linux’s kernel space using eBPF and thus having a compressed representation in the form of 24bits helps to conserve the valuable kernel-space memory requirement and aids in faster identity verification.
Cilium has two operational modes:

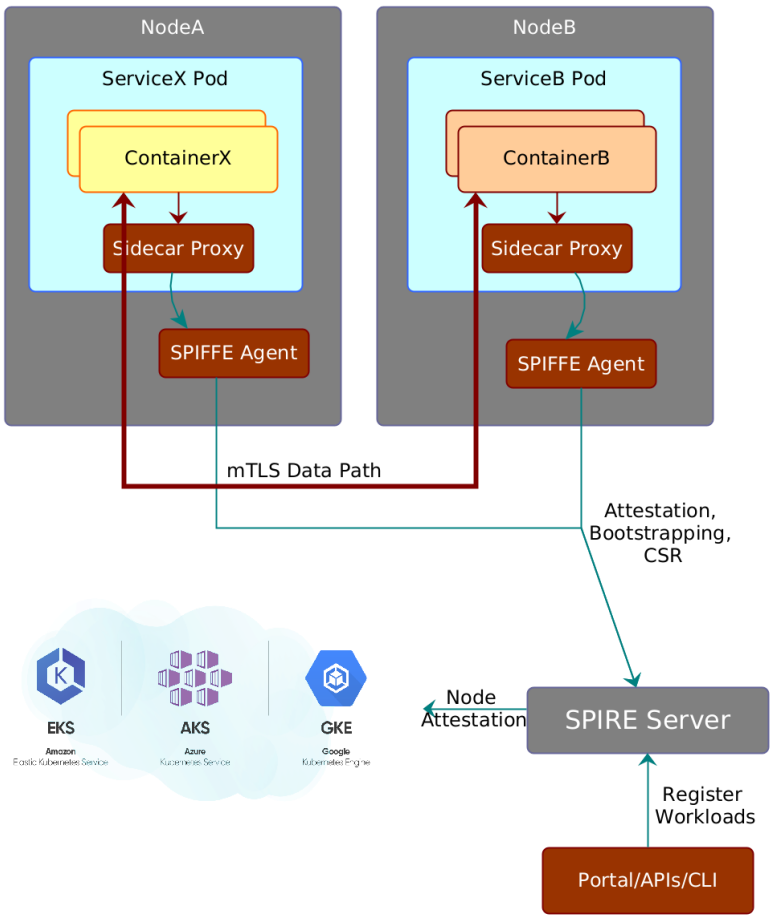
a) Tunneled mode which uses a tunneling protocol such as VXLAN or Geneve. This is the default mode for unmanaged k8s installation.
b) Direct-routing mode which does not use any tunneling protocol and depends on direct routing capabilities of the kernel. This mode is by default used for GKE and other managed k8s installations.

In tunneled mode, the Identity value is carried as part of the vxlan network ID (VNI) field. While in case of direct-routing mode, an ipcache is maintained which synchronizes the Identity values with the corresponding pod IP addresses.

Cilium is the only CNI which does the authorization checks based on per-packet basis and the use of eBPF in kernel-space alleviates the performance impact of such checks. The per-packet checks allow the Cilium CNI to instantly react to policy changes for the already established connections/sessions.
Cilium does not employ explicit workload attestation procedures and does not need to have any additional control plane elements to attest the identities.

SPIFFE based Identity Solution

SPIFFE is a universal identity control plane for distributed systems. It is not tightly coupled with k8s but provides integration with k8s environment in the form of plugins. SPIRE, the SPIFFE Runtime Environment, is an extensible system that implements the principles embodied in the SPIFFE standards. SPIRE manages platform and workload attestation, provides an API for controlling attestation policies, and coordinates certificate issuance and rotation. The attestation procedure helps SPIFFE to do away with any static configuration of workload secrets/credentials.
The SPIFFE workload attributes are mapped to SPIFFE ID which is in the form of an URI (Uniform Resource Identifier). An example of SPIFFE ID is,
spiffe://acme.com/billing/payments.
The Identity attestation procedure results in deriving an Identity document called SVID (SPIFFE Verifiable Identity Document) which essentially is a x.509 signed certificate with few mandatory fields such as the presence of SAN (Subject Alternate Name) which carries the SPIFFE ID.



The Identity document essentially a certificate can be used across any and all applications from that workload pod/container.

Aporeto's Trireme based Identity Solution

Note: Trireme is not as actively maintained as other approaches mentioned here. However, Trireme provides yet another TCP-based approach for identity handling and hence is discussed here in detail.

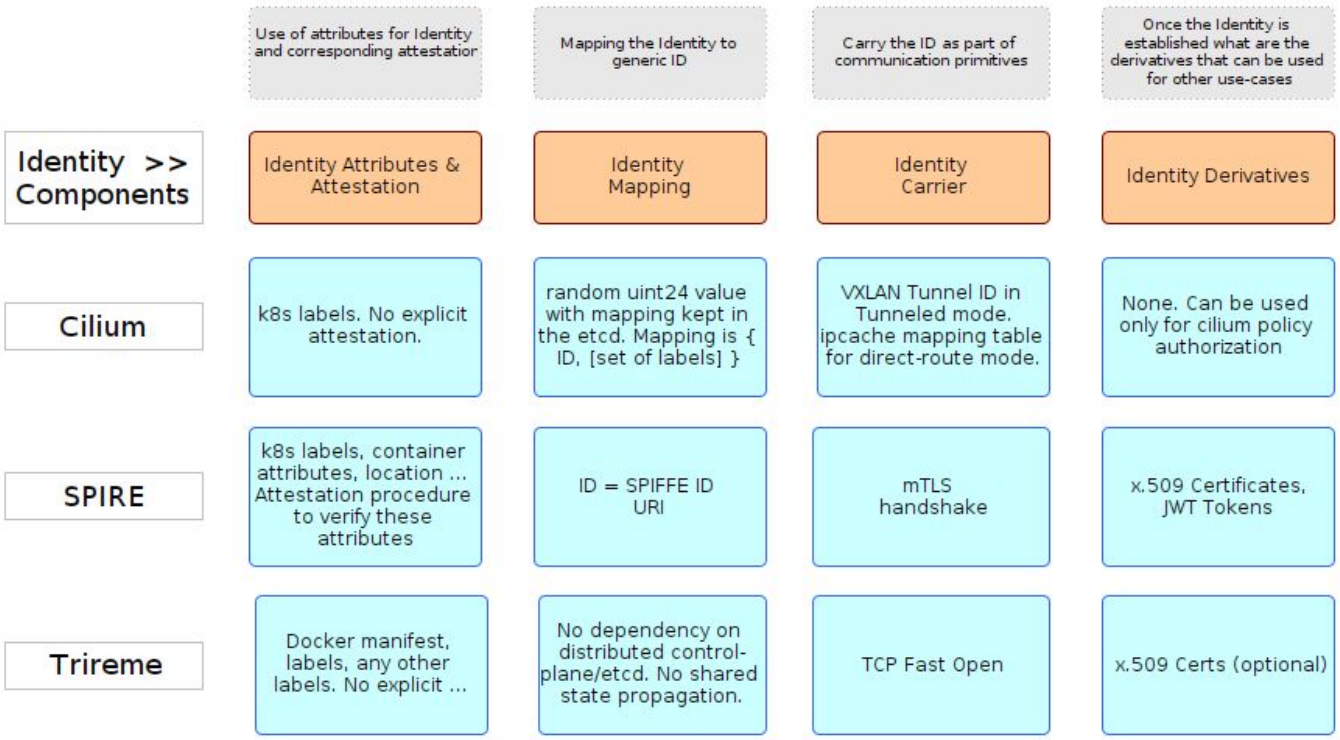
For Trireme, the identity is a set of attributes and metadata that describes the container as key/value pairs. Trireme provides an extensible interface for defining these identities. In case of k8s, the identity can be the set of labels identity a Pod.

The identity values are not mapped to any intermediate representation and are carried as it is as part of TCP handshake. TCP Fast Open (TFO) primitive is used to handshake the set of key-value pairs. The elimination of any intermediate representation has resulted in simplification of the control-plane for Trireme since it does not require any state synchronization/propagation and thus does not depend on any distributed control plane entity (such as etcd in case of Cilium).

The use of TFO may have its own issues. What if the application already uses TFO for its own purpose? What if there are stateful TCP proxies in the path which may strip off the TFO options?

Trireme doesn't have a notion of explicit attestation procedure, but the TFO options are cryptographically protected. OPEN QUESTION: How are the TFO options cryptographically protected, which keys are used?

Summary



Detailed Comparison of Identity Solutions

Use the link to view in the spreadsheet.
https://docs.google.com/spreadsheets/d/14sM6EATOWyjAowL48li5Zm32pZVVzv-H7CDe_jERvUo/edit?usp=sharing

		Identity Solutions			Remarks
		Cilium	SPIFFE	Trireme	
Char	Identity Unforgeability	??		??	Can a malicious node spoof an identity?
	Keeping sensitive keys	NA		??	

acteristicsofIdentitySolution	isolated				
	Extending to new attributes				The identity attributes may be updated. Does the solution have a clean way of adding new attributes? Ideally the attribute addition should only impact the identity system and should not have any impact on other subsystems of othe underlying platform.
	Granularity				
	Extending to non-k8s workloads				
	Ability to change identity attributes at runtime and handling authz updates for in-progress conns				
	Single-side support	??			
	Dependency on control plane for state sync or identity certification				
	Use of Identity solution to aid other apps				
	Robustness of Identity carrier				Is the method to carry Identity a foolproof one? Does it depend on middleboxes? Does it depend on IPAM?
	Support for Identity for unecrypted sessions				Cilium and Trireme do not care about the use of TLS and their identity solution can work on unecrypted connections too.
	Ease of multi-cluster deployments				
	Identity Federation				
	Ability to integrate with				

	TPMs				
	Blast-radius				Without strong cryptographic protection, it won't be possible to limit the blast radius in case one of the nodes gets a privileged attacker. In case of SPIFFE, there are checks in the form of attestation logic and the possibility to integrate the attestation logic with TPMs.

References

1. https://www.youtube.com/watch?v=7N1PFdCEFE8
2. Identity Bootstrapping in Multi-tenant Multi-cluster Kubernetes - Manish Mehta, Derek Suzuki
3. https://developer.ibm.com/components/istio/articles/istio-identity-spiffe-spire/
4. https://github.com/aporeto-inc/trireme-lib