Cilium and SPIFFE Integration

Table of Contents

Pre-requisites

1. Identity Solutions Document
2. Understanding of how Cilium and Istio identity solutions work. The above document will help for this too.

Benefits SPIFFE could provide to Cilium

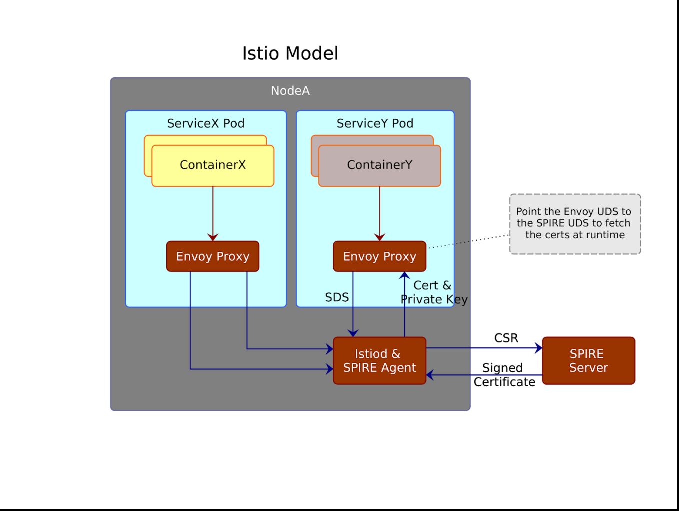
1. Strong attestation and authentication procedures for Identity. Strong cryptographic protection for the identity value.
2. Generic Identity solution which extends to non-k8s workloads and to edge/IoT/endpoint scenarios as well.
3. Ability to federate identity across multiple service providers. For e.g, if one service provider uses Istio based service mesh and another with Cilium+SPIFFE, it will be possible to federate the identity.
4. Ability to use the SVIDs for other purposes such as transparent encryption, WireGuard/IPSec tunnels. Solve the problem of certificate management in a right way across all the services/use-cases.

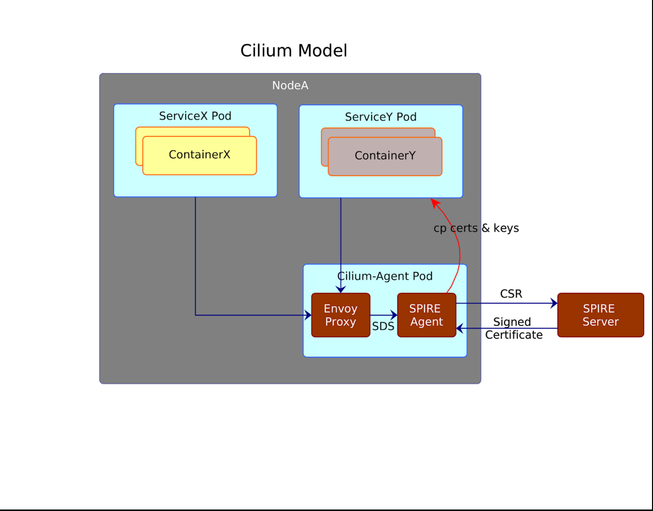
5. Ability to extend the identity solution with hardware based attestation service using confidential computing (enclaves, TPMs).

Design Considerations for SPIFFE Integration

1. Ability to fallback to existing Identity carrier mechanism in case SPIFFE is disabled/not available or if the policy is to allow for non-encrypted sessions to continue.
2. No impact (performance or functional) on Cilium data-path handling of identities.
3. Ability to use per-packet identity and mTLS handshake both for authz
4. Retain the ability to react to policy changes instantaneously for already-established connections by validating identity value on per-packet basis.

Difference in Envoy model between Istio and Cilium





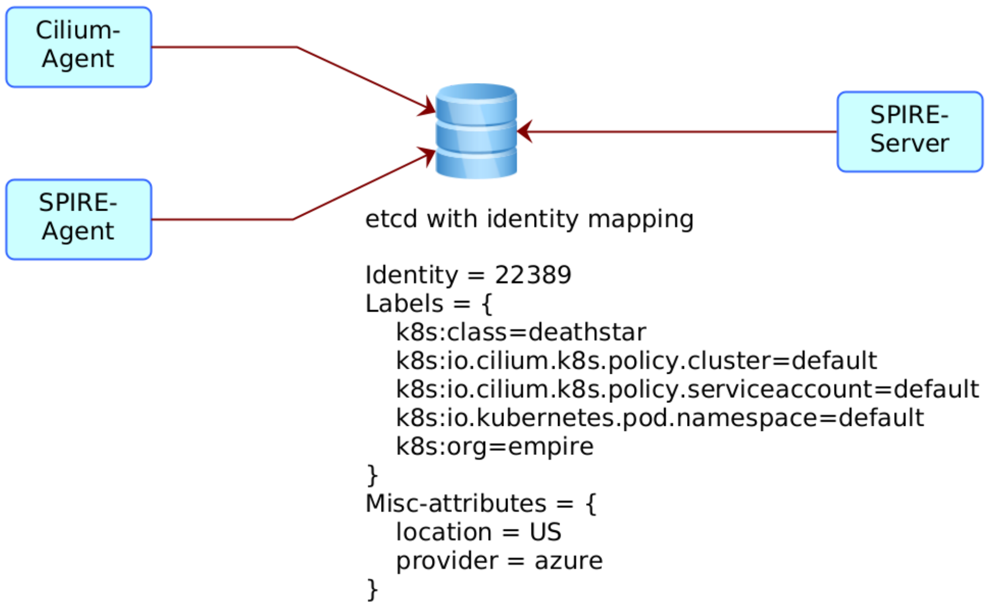
The Envoy proxy invocation model is different between Cilium and Istio. Istio spawns an Envoy proxy on a per pod basis in a typical sidecar model whereas in case of Cilium the Envoy proxy is started on a per node basis.
The envoy proxy needs to have access to the private keys associated with the Identity. Usually, the best practices define that the private keys of the containers should never leave the pod. In case of Cilium's current model, it may not be possible to follow this practice given the placement of the envoy proxy in its architecture.
[What are the implications of using one-proxy-per-node mode?]

SPIFFE ID format

Use of SPIFFE entails deciding on the SPIFFE identifier format which can take up several forms. The possible options are

1. Verbose format. Example: *spiffe://trust-domain/orgBST/clusterXYZ/nodeABC/endpointLMN*. Verbose format is the most widely used format in which every part of the path information (viz, org, cluster, node, …) is defined by the implementers. The verbose format allows ease of debuggability and maintenance.
2. Opaque format. Example: *spiffe://trust-domain/22839* ... where 22839 is the Identity value whose detailed information is known to internal systems. The SPIFFE path may be left opaque, carrying no visible hierarchical information. The metadata associated with the SPIFFE opaque value has to be provided by a secondary system.

The choice of trust-domain name is critical only when Identity federation is desired. The installation procedure can take the value of the trust-domain name as a configuration parameter. If the user does not explicitly pass the trust-domain, a default in the form of "identity.example.com" can be chosen. It should be possible to migrate to a different trust-domain name in the future and a tooling should be provided to handle this migration.



```
Identity = 22389
Labels = {
    k8s:class=deathstar
    k8s:io.cilium.k8s.policy.cluster=default
    k8s:io.cilium.k8s.policy.serviceaccount=default
    k8s:io.kubernetes.pod.namespace=default
    k8s:org=empire
}
Misc-attributes = {
    location = US
    provider = azure
}
```

Notice the use of misc-attributes which can be used additionally to augment the identity of the pod/container/service. All the attributes of the identity will be attested as part of the attestation procedure.

The use of opaque identifiers naturally fits in the Cilium's architecture. Cilium's use of eBPF for runtime policy enforcement in kernel-space requires use of identity which is easy to load/verify (i.e, using basic integer operation) and that is where the current uint24 bit value of Identity plays a crucial role. This model is central to Cilium's data path performance since the verification is done on a per-packet basis (and even with SPIFFE identity this per-packet verification may be retained).

Istio/Kuma's choice of SPIFFE ID

Following is the format chosen by Istio/Kuma:

Kuma: spiffe://<mesh name>/<service name>
Istio: spiffe://<trust-domain>/ns/<namespace>/sa/<service-account>
Both Istio and Kuma depend on k8s ServiceAccountToken attestation procedure to verify the pod's identity.
Even after the SPIFFE integration, Istio and Kuma attach the SPIFFE ID to the service account token itself and thus the identity of the services are still tied directly to the service-accounts. Istio-SPIFFE integrators have published a document which explains pitfalls of using service-accounts for workload identity. In one such example, it was mentioned that using service accounts based identity could result in unwanted behaviour when services are spawned across multiple regions and there could be regulatory limitations in allowing the data access. In such case, the identity solution may have to consider the region/location of the service as well to fetch the identity. However, the SPIFFE ID format of Istio still does not allow such attributes to be taken into consideration. [*Rahul - Need validation of this point*].

What would be the procedure to include new identity attributes in the future?

Attestation logic for new attributes. Updating the Identity map with a new Identity value (already present for the most part). Updating the SVID based on the updated attributes.

Fallback to classic Cilium Identity solution

By classic identity solution, I mean the current solution which maps the set of k8s labels to an identity value which is directly used for authorization. In case the cluster supports non-encrypted transport connections, such a fallback option might be beneficial. The design point where we make use of opaque identifiers in the form of uint24 bit value which is the same as the current identity value, allows this integration to easily fallback to classic Cilium identity solution without any additional checks during runtime (especially in the data path).
The design does not make any change in the way the identity is handled in the kernel-space i.e, based on u24_t value.

Implications of Transparent Encryption support in Cilium

Cilium currently supports transparent encryption using IPSec alone and does not have a way to enforce transparent encryption using TLS. This would have impact in following ways:

1. The SPIFFE derived certs needs to be used at IPSec layer in case transparent encryption is desired.
2. It would be important to handle un-encrypted sessions and thus support the current notion of Cilium Identity.
3. Handling workloads who already use TLS? [TODO]

Use of external CA

The default installation will provision a private CA along with the SPIRE-server. It should be possible to configure an external CA during SPIRE-agent installation or use some tooling in the future to change the CA.

Design/Implementation Decisions?

1. SPIFFE ID format to follow
2. Is the concern with respect to envoy proxy instantiation valid?
3. Mode to follow for transparent encryption with SPIFFE-derived certs.
4. Fallback option? General configuration options to be considered?
5. Tooling for registration of workloads