Lab 5.3 - Configure Mutual TLS for Istio

Overview

In this lab, you'll be configuring mutual TLS (mTLS) with Istio. Istio has two configuration modes for mTLS:

- **`PERMISSIVE`** allows both encrypted and unencrypted (plaintext) traffic. While all requests between service proxies are encrypted by default, this mode allows services outside of the mesh to connect to services within the mesh without using encryption or authentication.

- **`STRICT`** requires that all connections be encrypted and that all services connecting to a service in the mesh authenticate themselves with certificates. This mode makes the mesh a closed environment that only services with the right certificates can connect to.

The default for Istio is **`PERMISSIVE`** mode because that makes migrations easy. Since you have already meshed your application, you can enable **`STRICT`** mode to prevent services outside of the mesh from connecting.

1. If you don't still have an active session from the previous Istio lab, perform these steps to reestablish it:

    a. Connect to the VM that hosts your Kubernetes clusters.

    b. To start the cluster that contains the Istio service mesh, issue this command:

    ```
    yourname@ubuntu-vm:~$ docker start $(docker ps -a -f
    name=istio-control-plane -q)
    ```



```
2d1d09fadf21
```

c. To switch the **kind** context to the Istio cluster, use this command:

```
yourname@ubuntu-vm:~$ kubectl config use-context kind-istio

Switched to context "kind-istio".
```

2. To demonstrate that **STRICT** mode requires all services to authenticate, you first need to have a service running outside of the mesh. Since all services in your demo application and your ingress controller are already establishing secure connections, you need to deploy another service. To do that, you can run a container that has the **curl** binary in the default namespace so the sidecar is not auto injected. Use the following command to do that:

```
yourname@ubuntu-vm:~$ kubectl run -n default curl-pod \
  --image curlimages/curl --command -- sleep infinity

pod/curl-pod created
```

3. Create a forwarder to the **web-svc** emoji service

```
yourname@ubuntu-vm:~$ kubectl port-forward service/web-svc -n
emojivoto 8080:80
```

4. Now you can send requests from a service outside the mesh. Make sure it works by sending a request to the **web-svc**. You should receive a **200** response code, which indicates you can successfully connect to the **web-svc** service with **PERMISSIVE** mode.

```
yourname@ubuntu-vm:~$ kubectl exec -n default curl-pod \
  -- curl -s -o /dev/null -w "Upstream Response Code:
%{http_code}\n" \
  http://web-svc.emojivoto/

Upstream Response Code: 200
```

5. Enable **STRICT** mode with a **PeerAuthentication** configuration.

```
yourname@ubuntu-vm:~$ kubectl apply -n istio-system -f - <<EOF
apiVersion: "security.istio.io/v1beta1"
kind: "PeerAuthentication"
```



```
metadata:
  name: "default"
spec:
  mtls:
    mode: STRICT
EOF

peerauthentication.security.istio.io/default created
```

6. To clean up from this lab, remove the **PeerAuthentication** configuration:

```
yourname@ubuntu-vm:~$ kubectl delete -n istio-system
peerauthentication default

peerauthentication.security.istio.io "default" deleted
```

7. Finish the lab cleanup by removing the **curl-pod** service:

```
yourname@ubuntu-vm:~$ kubectl delete pod -n default curl-pod

pod "curl-pod" deleted
```

8. You've reached the end of this lab. If you're not starting the next lab right away, you can stop the Istio cluster by running this command:

```
yourname@ubuntu-vm:~$ docker stop $(docker ps -a -f
name=istio-control-plane -q)

2d1d09fadf21
```

