Simulating a Man-in-the-Middle Attack in a Kubernetes + Istio Environment

Author: K M Asifur Rahman

Date: June 28, 2025

1. Introduction

This report outlines the methodology used to simulate a Man-in-the-Middle (MITM) attack within a service mesh managed by Istio on a local Kubernetes cluster using Minikube. The attack demonstrates how service impersonation and traffic interception can be used to manipulate and analyze intra-cluster communication.

2. Objective

To intercept and analyze internal service-to-service communication by introducing a malicious proxy pod in place of a legitimate service, effectively executing a MITM attack inside a service mesh environment.

3. Environment Setup

- Kubernetes Platform: Minikube
- Service Mesh: Istio
- Application Stack: Python Flask-based microservices
- Pods/Services Involved:
  - flaskapp: The source microservice making HTTP requests
  - flask-app-2: The legitimate destination microservice
  - mitm-proxy: The malicious interceptor pod

4. Attack Methodology

Step 1: Deploy the Attacker Pod

Command:

```
kubectl apply -f mitm-attacker.yml
```

A deployment named mitm-proxy is created, which is designed to act as a reverse proxy that captures, logs, and forwards requests.

Step 2: Replace the Legitimate Service

Commands:

```
kubectl delete service flask-app-2
kubectl delete service mitm-flask-app-2
```

The original service endpoint that flaskapp communicates with (flask-app-2) is deleted to make room for the malicious proxy.

Step 3: Deploy MITM Proxy as the Legitimate Service

Command:

```
kubectl expose deployment mitm-proxy
--name=flask-app-2 --port=5001 --target-port=5001
```

The mitm-proxy deployment is now exposed under the original name flask-app-2, effectively impersonating the service.

Step 4: Expose the Legitimate Service Under a New Name

Command:

```
kubectl expose deployment flask-app-2
--name=flask-app-2-backup --port=5001
--target-port=5001
```

The original service is still deployed but is now accessed by the MITM proxy under the name flask-app-2-backup.

Step 5: Interception and Forwarding

- flaskapp continues making requests to flask-app-2.
- These requests are intercepted by mitm-proxy, which:
  1. Logs the request
  2. Forwards it to flask-app-2-backup
  3. Receives the response and sends it back to flaskapp
- Logged intercepted requests are made available at:
  http://localhost:5001/intercepted

5. Results and Observations

Full request/response cycles were successfully intercepted, logged, and forwarded without causing service disruption. The proxy service captured all inbound traffic transparently.

6. Mitigation Strategy

To mitigate the MITM attack, an Istio AuthorizationPolicy was applied to restrict access to the `flask-app-2` service. This policy allows only the `flaskapp` workload, within the same namespace, to communicate with `flask-app-2`. Requests originating from other namespaces, such as the `default` namespace where the `mitm-proxy` pod resides, are denied by this policy.

Example AuthorizationPolicy YAML:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-flaskapp-only
  namespace: your-namespace
spec:
  selector:
    matchLabels:
      app: flask-app-2
  rules:
  - from:
    - source:
        principals:
["cluster.local/ns/your-namespace/sa/flaskapp-servi
ce-account"]
```

This policy ensures that even if a malicious pod like `mitm-proxy` is deployed, it will not be able to intercept traffic unless it also resides in the same namespace and uses the allowed service account. As a result, unauthorized requests to `flask-app-2` from the `mitm-proxy` pod were successfully blocked.