Latency-Behavior MITM Detection for Kubernetes Microservices

Draft Report

Author: Asifur Rahman

Date: August 15, 2025

Abstract

This report proposes a lightweight, container-friendly machine learning (ML) detector that identifies man-in-the-middle (MITM) attacks in microservice architectures (MSA) on Kubernetes by profiling inter-service response-time behavior. The core hypothesis is that an on-path adversary forwarding or forging responses introduces characteristic timing shifts—including phase-specific delays and tail amplification—that can be learned from normal baselines and flagged online. We outline a threat model specific to Kubernetes service meshes (e.g., Istio) and present a deployable design that collects per-route latency features, trains per-endpoint models, and performs sliding-window post-processing to stabilize alarms. We also present an evaluation plan using Minikube/Istio with scripted MITM and benign load fluctuations, and discuss limitations (e.g., variance under bursty workloads) and attacker adaptivity. The proposed approach addresses gaps noted by recent surveys about insufficient detection mechanisms for inter-service threats in MSA.

Keywords: Kubernetes, Microservices, Service Mesh, Istio, Man-in-the-Middle, Latency Profiling, Anomaly Detection, Lightweight ML

1. Introduction

Microservice architectures (MSA) introduce complex, highly distributed communication patterns that expand the attack surface. Even with best practices such as mTLS, misconfigurations, legacy services, and staged rollouts can create windows where on-path manipulation is possible. This work investigates whether inter-service response-time behavior can provide a reliable, low-cost signal for detecting MITM attacks. Unlike generic performance anomaly detectors, the objective here is security-centric: to infer adversarial presence or response forgery from latency statistics, without deep packet inspection or invasive instrumentation. The resulting detector must be lightweight enough to run in sidecars or DaemonSets and resilient to normal workload variance.

2. Related Work

Intrusion detection for containers has explored system call analysis and specialized representations to improve learning efficacy. Graph-based encodings and sliding-window post-processing have shown strong gains in detection performance with reduced false alarms. Meanwhile, domain-specific timing-based MITM detection has been demonstrated in Bluetooth Low Energy (BLE) environments, where response-time behavior serves as a robust indicator of on-path mediation. However, the literature lacks a concrete adaptation of response-time behavioral profiling for detecting MITM in Kubernetes-based microservices or service meshes. Surveys of inter-service threats emphasize monitoring gaps, especially around practical detection mechanisms for communication-layer attacks such as MITM and forged responses.

3. Threat Model

Adversary Goal: Intercept or forge inter-service traffic to inject or alter responses.

Capabilities:
• On-path positioning within the cluster (e.g., compromised pod/node, misconfigured CNI/iptables, rogue sidecar, or malicious proxy).
• Ability to forward requests and respond with forged content while incurring small but non-zero processing delays.

Assumptions:
• mTLS may be disabled/misconfigured on some routes or during migration; or the adversary presents seemingly valid credentials due to supply-chain compromise.
• Normal operations exhibit measurable latency distributions per route (HTTP/gRPC), including phase components (DNS, TLS handshake, upstream handler time).

Out of Scope:
• Pure application-layer logic compromises at the responder that do not change network/processing timing.
• Attacks that perfectly mimic timing (unrealistic given variable compute and I/O along the attack path).

4. System Design

The proposed detector runs as a lightweight agent (sidecar or DaemonSet) that ingests per-request telemetry from the service mesh (e.g., Envoy/Istio metrics) and/or application instrumentation (timers around client calls). It computes features per route (serviceA→serviceB, method, status) and maintains rolling baselines. A compact model (e.g., one-class classifier or shallow autoencoder) is trained per route to model normal latency behavior; online inference flags anomalies, and sliding-window logic smooths alerts.

Architecture Components:
1) Telemetry Collector: extracts request/response timestamps, optional phase times (DNS, TLS, connect, TTFB), status codes, bytes, retries.
2) Feature Extractor: computes distributional features (p50/p90/p99, IQR, tail index, burstiness), change features (CUSUM/ED drift), and phase deltas.
3) Route-Scoped Models: one model per (caller, callee, endpoint) with adaptive baselines.
4) Alarm Aggregator: sliding-window voting, suppression by deployment events (rollouts, HPA scaling), and SLO-aware thresholds.
5) Notifier: emits Kubernetes Events and Prometheus alerts; optional webhook to SIEM.

4.1 Latency Feature Set

Feature	Description	Rationale for MITM

p50/p90/p99 latency	Robust central/tail latency	MITM adds small but consistent delay; tails amplify.
IQR & MAD	Dispersion/robust spread	Detects jitter inflation under on-path mediation.
Tail index (e.g., Hill)	Heaviness of tail	Forged/forwarded responses can shift tail behavior.
Burstiness (Fano factor)	Variance-to-mean of arrivals	On-path processing queues create bursty latencies.
CUSUM/ED drift	Online change scores	Captures sustained shifts beyond natural variance.
Phase timings	DNS/TLS/connect/TTFB splits	Adversary adds time to connect/TTFB phases.
Retry/backoff counts	Client retry metrics	MITM artifacts trigger retries/timeouts.
Status mix	2xx/4xx/5xx proportions	Forged responses may change status distribution.

4.2 Model Choices

Given container resource constraints, two families of models are suitable:

• One-Class Models: One-Class SVM, Isolation Forest, or Elliptic Envelope trained on normal data; simple to maintain per route.

• Shallow Autoencoders: Tiny feed-forward networks reconstructing feature vectors; reconstruction error provides anomaly score.

Both benefit from per-route training and periodic re-baselining (e.g., nightly) with guardrails to avoid learning attacks. Model outputs feed a sliding-window post-processor to stabilize decisions and reduce false positives during short-lived spikes.

5. Deployment Plan

• Sidecar Mode: Attach to critical caller pods; intercept client libraries or leverage mesh-generated metrics via /stats or Prometheus scraping.

• DaemonSet Mode: Node-local agent correlates Envoy metrics and application logs; lower overhead for large clusters.

• Configuration: Per-namespace allowlists of monitored routes; SLO/SLA metadata to scope sensitivity.

• Outputs: Prometheus metrics (detector_score, alert_state), Kubernetes Events, and Alertmanager routes.

6. Evaluation Methodology

Testbed: Minikube with Istio (or Linkerd) on a multi-service demo app (e.g., boutique, Hipster shop). Generate benign scenarios (rollouts, HPA scaling, cache cold starts) and adversarial scenarios:

• MITM via on-path proxy inside the cluster namespace (e.g., ARP spoof/iptables redirect) passing or forging responses.
• Variant MITM with minimized added delay (attacker attempts to hide within jitter).
• Non-MITM load anomalies (CPU throttling, network shaping) to test specificity.

Metrics: AUROC/PR for attack detection, false-alarm rate under benign spikes, detection delay, and overhead (CPU/mem). Ablations compare feature subsets and post-processing strategies (window sizes, voting rules).

7. Experimental Results

This draft reserves space for quantitative results once experiments are completed. We will present ROC/PR curves, alert timelines aligned to injected attacks, and resource overhead tables. We will also report sensitivity analyses across services, endpoints, and load levels.

8. Limitations and Attack Evasion

• High Variance Routes: Some endpoints naturally exhibit high jitter; per-route modeling and phase features mitigate but cannot eliminate ambiguity.
• Adaptive Adversary: An attacker may attempt to pad benign traffic or throttle to confound baselines. Combining phase decomposition and distributional features raises the bar.
• Encrypted/Optimized Paths: HTTP/2 multiplexing and connection pooling reduce handshake visibility; emphasize TTFB and server-time proxies where available.
• Concept Drift: Workload and topology evolve; require scheduled re-baselining with holdout windows to avoid learning attacks.

9. Ethical and Operational Considerations

Detectors must avoid excessive false alarms that can fatigue operators. Tuning should respect SLOs and maintenance windows. Data collection is limited to timing/metadata; no collection of payloads or PII. Open-sourcing detectors and datasets (synthetic where necessary) can aid reproducibility without exposing sensitive traffic.

10. Conclusion and Novelty Statement

We proposed a latency-behavior, lightweight ML approach to detect MITM and forged inter-service responses in Kubernetes microservices. To the best of our knowledge, this is the first system to adapt response-time profiling—previously shown effective in other domains—to service-mesh environments for security detection. By combining per-route modeling, phase-aware features, and sliding-window post-processing, the detector aims to distinguish attacker-induced delays from normal variance. Future work includes richer topology signals and integration with admission controls for rapid automated mitigation.

References

Araujo, C., & Vieira, M. 'Enhancing intrusion detection in containerized services: Assessing machine learning models and an advanced representation for system call data.'

Haindl, A., et al. 'A Systematic Literature Review of Inter-Service Security Threats and Mitigation Strategies in Microservice Architectures.'

Yurdagul, S., & Sencar, H. 'BLEKeeper: Response Time Behavior Based Man-In-The-Middle Attack Detection.'

Appendix A: Online Detection Pseudocode

```
for each route r:
   model_r = train_on(normal_feature_vectors[r])
   baseline_r = init_baseline_stats(r)

for each request q on route r:
   x = features(q)  # p50/p90/p99, IQR, tail index, phase times, retries, status mix
   s = anomaly_score(model_r, x)
   update_window(r, s)
   if window_decision(r) == ALERT:
      emit_alert(r, q.timestamp, s)
```