Lab 3.1 Report: Event-Driven Cybersecurity Pipeline

Team 2

1. Introduction

In this laboratory, we implemented a simplified cybersecurity analysis pipeline based on event streaming. The system demonstrates how to decouple data generation from processing using Kafka, trace events using Jaeger, and analyze attack patterns using MITRE ATT&CK classifications.

2. Conceptual Questions & Answers

Q1: Why is Kafka used instead of direct function calls?

In a direct function call (synchronous communication), the caller must wait for the receiver to finish processing before continuing. In a high-load cybersecurity pipeline, this is risky because a slow classification model would block data ingestion, leading to potential packet loss.

Kafka decouples the system components, offering three main benefits:

- Asynchronous Processing: The Producer can ingest thousands of events per second without waiting for the Classifier to respond.
- Buffering: Kafka acts as a buffer. If the system experiences a sudden burst of network traffic, Kafka holds the data until the Consumer is ready, preventing system crashes.
- Independence: The Producer does not need to know the identity or state of the consumer, allowing for easier maintenance and upgrades.

Q2: What happens if the consumer is slower than the producer?

This scenario creates Consumer Lag. Since the Producer pushes data faster than the Consumer can process it, messages accumulate in the Kafka topic (queue).

- System Stability: Unlike direct calls, the system does not crash. The Producer continues working at full speed.
- Latency: The "freshness" of the data decreases (latency increases) because events sit in the queue waiting to be processed.
- Resolution: To fix this, we can scale up the Consumer (add more instances) to process the backlog in parallel without modifying the Producer.

Q3: How does tracing help debug pipeline behavior?

In a distributed event-driven architecture, logs are scattered across different containers and services, making it difficult to track a single event's path.

Distributed Tracing (Jaeger) allows us to:

- End-to-End Visibility: Visualize the entire journey of a single packet from producer → kafka → classifier → storage.
- Performance Analysis: Identify exactly which stage is the bottleneck (e.g., distinguishing between network latency and model inference time).
- Error Tracking: If an event is lost or fails, the trace shows exactly where the chain broke.

Q4: Which pipeline stages could be scaled independently?

Because the components are decoupled by Kafka, every stage can be scaled independently based on its specific resource needs:

- The Consumer/Classifier: This is typically the bottleneck (CPU/GPU intensive). We can run multiple instances (Consumer Group) to process events in parallel from the same Kafka topic.
- The Producer: If data generation is slow, we can run multiple producers.
- Storage: The database writing process can also be parallelized. Scaling one component does not require restarting or modifying the others.

Q5: How would this pipeline change in a real SOC system?

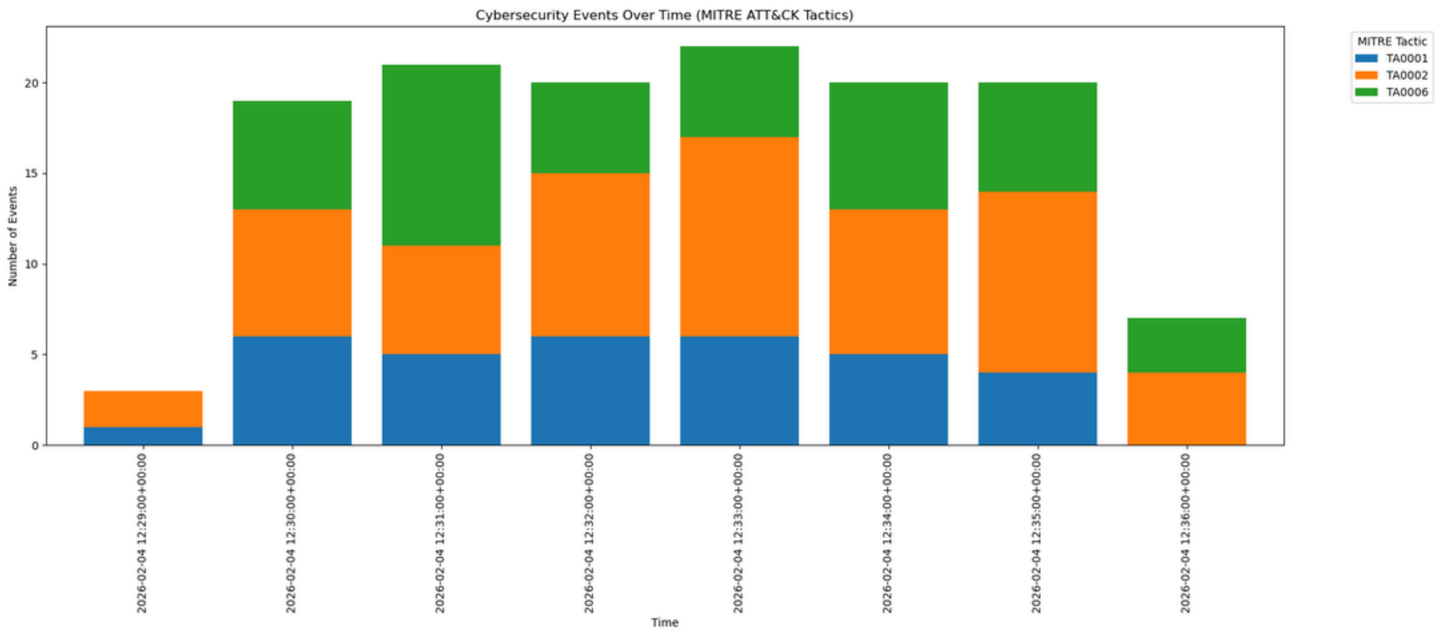
While this lab uses a simplified architecture, a real-world Security Operations Center (SOC) pipeline would include:

- Persistent Storage: Instead of a local CSV file, data would be stored in a high-performance database optimized for logs, such as Elasticsearch or ClickHouse.
- Data Enrichment: An intermediate stage would add context to events (e.g., resolving IP addresses to Geo-locations or checking IPs against Threat Intelligence feeds).
- Real-time Alerting: A separate consumer would listen for critical findings (e.g., "Brute Force Detected") and immediately push notifications to Slack, PagerDuty, or a SIEM dashboard.
- Data Retention Policies: Implementing mechanisms to archive old data to cold storage (e.g., AWS S3) to reduce costs.

3. Visual Evidence

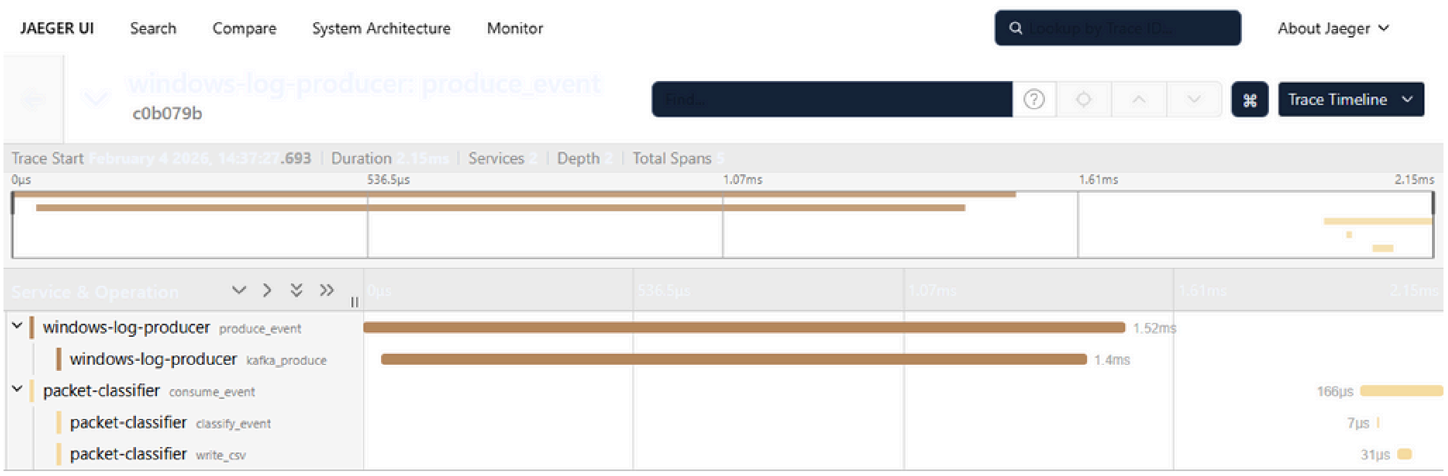
Pipeline Output (Statistics)

The following graph demonstrates the distribution of detected MITRE ATT&CK tactics over time, processed by our pipeline.



Distributed Tracing (Jaeger)

The following trace captures the end-to-end latency of a single event flowing through the system.



windows-log-producer: produce_event
c0b079b

Trace Start February 4 2026, 14:37:27.693 | Duration 2.15ms | Services 2 | Depth 2 | Total Spans 5

0µs 536.5µs 1.07ms 1.61ms 2.15ms

windows-log-producer produce_event 1.52ms
windows-log-producer kafka_produce 1.4ms
packet-classifier consume_event 166µs
packet-classifier classify_event 7µs
packet-classifier write_csv 31µs