Multi-Stage Cyber Attack Detection Through Log Abstraction and Temporal Correlation

Author: Ansh Agrawal

Project Date: Sophomore Year, TJHSST

Field: Cybersecurity / Software Engineering

Dec 2025

Abstract

Security analysts frequently experience "alert fatigue" because many existing security tools treat logs as isolated events, rather than as parts of a larger attack sequence. A single failed login attempt is often benign, but dozens of failures followed by a successful authentication may indicate a serious breach.

This paper presents a two-layer detection system designed to address this problem. The first layer, the Log Analyzer, normalizes raw logs and detects individual suspicious events using explainable, rule-based logic. The second layer, the NetFlowAttackSequencer, performs temporal correlation across network and host-based events to reconstruct multi-stage attack chains.

The system was evaluated using the CIC-IDS-2017 dataset as well as controlled Linux lab environments. It successfully identified brute force and infiltration scenarios while significantly reducing alert volume. By mapping raw log data to a unified event schema and correlating events over time, the system allows defenders to observe attacker intent rather than isolated symptoms.

---

1. Introduction

Modern cybersecurity systems generate massive volumes of logs from operating systems, applications, and network devices. While these logs contain valuable security information, they are often noisy, fragmented, and difficult to interpret in isolation. As a result, security analysts are frequently overwhelmed by alerts that lack meaningful context.

A single failed authentication attempt is usually harmless and may be caused by user error. However, hundreds of failed attempts followed by a successful login and privilege escalation represent a coordinated attack. Traditional log-based detection systems often fail to recognize this progression because they analyze each event independently.

This project addresses that limitation by focusing on *behavior over time* rather than individual log entries. Instead of asking whether a single event is malicious, the system asks whether a *sequence* of events forms a recognizable attack pattern.

Summary of Contributions:

- **Log Abstraction**: Converting unstructured log messages into structured event objects that can be queried and analyzed.
- **NetFlow Sequencing**: Integrating network flow data with host-based logs to capture both external and internal attacker activity.

- **Correlation Logic**: Implementing a state-based model that tracks attackers over time to identify multi-stage attack chains such as "Brute Force → Successful Login."

---

2. Problem Statement and Goals

Modern cyber attacks, especially Advanced Persistent Threats (APTs), rarely occur as a single action. Instead, attackers operate in stages, including reconnaissance, exploitation, and persistence. Tools that only inspect the current packet or log entry often miss the broader context of these attacks.This system intentionally avoids machine learning in favor of deterministic, explainable logic. In security operations, alerts must be trusted, justified, and actionable. The goal of this project is not anomaly detection in the abstract, but reconstruction of attacker behavior.

Several challenges make detection difficult:

- **Noise**: Legitimate system activity, such as cron jobs and background services, produces log entries that can obscure malicious behavior.
- **Volume**: Enterprise systems can generate millions of log lines per day, making manual analysis impossible.
- **Fragmentation**: Network telemetry (NetFlow) and host-based logs (such as authentication logs) are typically stored and analyzed separately.

The goal of this project is to build a detection system that is:

- **Explainable**: Every alert can be traced back to the exact logs and flows that caused it.
- **Modular**: New detection rules and attack patterns can be added without redesigning the system.
- **Context-Aware**: Alerts represent attacker behavior over time, not isolated incidents.

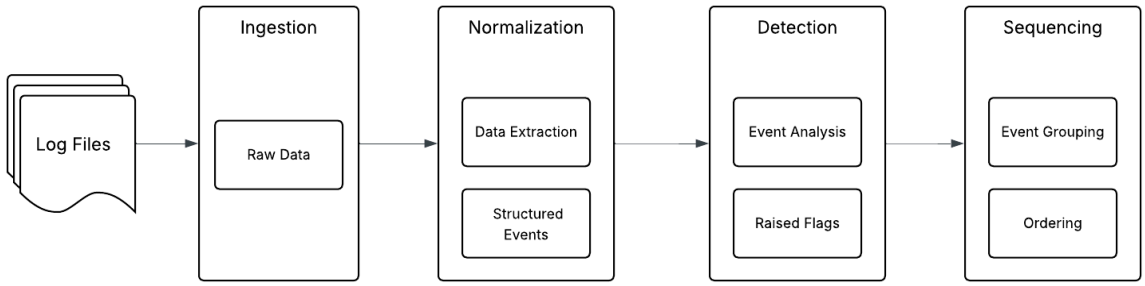
---

3. System Overview

The system is designed as a modular pipeline that transforms raw telemetry into correlated attack narratives. Each stage performs a specific function and passes structured data to the next layer.

3.1 Architecture

The system consists of four primary stages:



1. **Ingestion**: Raw data is collected from log files (such as Linux auth.log) or network flow datasets (such as CIC-IDS-2017 CSV files).
2. **Normalization**: Relevant fields such as timestamps, IP addresses, users, and actions are extracted and converted into structured event objects.
3. **Detection**: Individual events are analyzed using rule-based logic to flag suspicious behavior.
4. **Sequencing**: Events are grouped by actor (IP address or user) and ordered in time to reconstruct multi-stage attack chains.

This architecture allows detection logic to remain simple while enabling complex behavior to emerge through correlation.

3.2 Threat Model

This project focuses on detecting external attackers attempting to gain unauthorized access to a Linux system through network-based attacks. The primary threat considered is an attacker using automated tools such as Hydra or Nmap to perform brute force authentication attempts over protocols like SSH or FTP.

The attacker is assumed to:

- Operate remotely over the network
- Generate high-frequency, repetitive activity

- Attempt to gain access using guessed credentials rather than software exploits
- Leave detectable traces in authentication logs and network flow data

The system is designed to identify patterns of behavior over time, such as many failed login attempts followed by a successful authentication, rather than treating each event independently.

This threat model aligns with common real-world scenarios that cause alert fatigue in security operations, where individual events may appear harmless but their sequence reveals malicious intent.

Attacks that are out of scope for this project include insider threats, physical access attacks, and highly stealthy “low and slow” attacks that intentionally avoid triggering threshold-based detection. The system also assumes that logs remain intact; if an attacker gains root access and modifies log files, detection capability may be reduced.

By limiting the threat model, the project prioritizes explainability, low latency, and clarity of alerts, which are critical for practical security analysis.

---

4. Log Sources and Data Collection

The detection layer operates on both host-based and network-based telemetry in order to capture attacker behavior across multiple layers of the system.

Two primary log sources were used:

Linux auth.log:
The auth.log file records authentication-related activity, including SSH login attempts and privilege escalation events (such as sudo usage). This log is critical for identifying brute force attacks and successful compromises on the host.

CIC-IDS-2017 Dataset:
The CIC-IDS-2017 dataset provides labeled network traffic representing both benign and malicious activity. This project focuses on the dataset segments containing brute force behavior, specifically the SSH-Patator and FTP-Patator attack scenarios. Network flow records include metadata such as source IP, destination port, flow duration, and packet counts.

By combining these two data sources, the system captures both the external network behavior of an attacker and the internal system-level consequences of their actions.

---

5. Event Abstraction and Normalization

5.1 Motivation

Raw logs are stored as unstructured text strings, which makes automated analysis difficult. For example, a log entry such as:

Dec 27 14:20:01 server sshd[123]: Failed password for root from 192.168.1.50

contains useful information, but it cannot be easily queried or correlated without additional processing. Searching for keywords alone is unreliable and does not scale.
To enable systematic analysis, all logs must be converted into a structured representation that preserves only the relevant security information.

5.2 Unified Event Schema

Every log entry processed by the system is converted into a unified event object. This abstraction allows logs from different sources to be analyzed using the same detection and correlation logic.

Each event is represented as a structured object containing fields such as:

- timestamp
- source_ip
- event_type
- user
- severity

Example event structure:

```
{
  "timestamp": "2025-12-27 14:20:01",
  "source_ip": "192.168.1.50",
  "event_type": "AUTH_FAILURE",
  "user": "root",
  "severity": 3
}
```

By enforcing a consistent schema, the system can reason about authentication events, network flows, and privilege escalation using a common framework.

5.3 Parsing Logic

The Log Analyzer uses targeted parsing logic to extract structured fields from raw log messages. Regular expressions are used to isolate only the relevant components of each log entry while ignoring extraneous text.

For example, failed SSH login attempts are detected using the following pattern:

r"Failed password for (?:invalid user)?(\\w+) from ([\\d\\.]+) port (\\d+) ssh2"

This approach allows the system to extract the username and source IP address without being affected by variations in log formatting. Each matched log entry is then converted into a structured event and forwarded to the detection layer.

---

6. Detection Logic

The detection layer is responsible for identifying individual suspicious events before correlation occurs. Rather than using machine learning, this project emphasizes explainable, rule-based detection to ensure transparency and ease of debugging.

6.1 Rule-Based Detectors

All detectors operate using a threshold-plus-time-window model. Events are grouped by source IP and evaluated over a fixed time window. If the number or pattern of events exceeds a predefined threshold, the activity is flagged as suspicious.

Example brute force rule:
If the number of authentication failures from the same source IP exceeds a threshold X within T seconds, the source is flagged for brute force behavior.

This model allows detection logic to remain simple while still capturing high-frequency automated attacks.

6.2 Implemented Detectors

The following detectors were implemented in the Log Analyzer:

- **SSH Brute Force Detection**:Repeated authentication failures from the same source IP within a short time window are flagged as a brute force attempt.

- **NetFlow Flooding Detection**:High packet rates and abnormal flow durations in network telemetry are flagged as suspicious behavior. These detections are passed to the correlation engine for further analysis.

Each detector produces a structured alert that includes the source IP, timestamp, and triggering events, allowing the correlation engine to reconstruct attacker behavior.

---

7. Motivation for Correlation

Generally, intrusion detection systems generate alerts in isolation, treating each suspicious event as an independent incident. This approach leads to "siloed" thinking, where analysts must manually piece together logs from different sources to understand what actually happened.

For example, a network monitoring tool may detect high-volume traffic from an IP address, while a host-based system independently logs multiple failed login attempts. Individually, these alerts may appear low priority. When viewed together, however, they may represent a coordinated attack.

The motivation for correlation is to link related events across time and across layers of the system. By correlating network-level telemetry with application-level and host-level logs, the system can identify attacker intent rather than isolated symptoms.

This approach reduces alert fatigue by grouping many low-level alerts into a single, meaningful attack narrative.

---

8. Correlation Model

The correlation engine, referred to as the NetFlowAttackSequencer, is responsible for reconstructing multi-stage attacks by analyzing the temporal relationships between events generated by the Log Analyzer and network flow data.

8.1 Temporal Correlation & The Sliding Window

Rather than evaluating events at fixed timestamps, the sequencer uses a sliding window model that continuously evaluates activity over time. Events are grouped by actor (typically a source IP address) and sorted by timestamp.

In addition to absolute time, the system measures inter-arrival time (IAT) between consecutive events. Short IAT values are indicative of automated tools and scripted attacks, while longer IAT values are more likely to be human-driven activity.

For this project, an IAT threshold of two seconds was used to identify potentially automated behavior. Events occurring faster than this threshold are flagged for further analysis.

8.2 State-Based Logic

To maintain context across time, the correlation engine implements a simple state machine for each observed actor.

State	Name	Trigger Condition
State 0	Idle	Default state; no suspicious activity recorded.
State 1	Reconnaissance	Detection of abnormal network flows or repeated connection attempts.
State 2	Attacking	Active exploitation attempts, such as a burst of SSH/FTP authentication failures.
State 3	Compromised	A successful login or privilege escalation (sudo) from the same source IP.

By maintaining state, the engine avoids confusion when attackers pause or change tactics. Events are interpreted relative to prior activity rather than in isolation.

---

9. Example Attack Chains

The correlation engine produces attack chains that summarize attacker behavior as a sequence of related actions. Each chain represents a high-level security incident rather than individual log entries.

Example SSH brute force attack chain:

- **Reconnaissance**:Network telemetry indicates repeated connection attempts to port 22 with short inter-arrival times.
- **Brute Force**:The Log Analyzer detects a burst of authentication failures from the same source IP within a short time window.
- **Compromise**:A successful login is observed from the same IP following repeated failures.

Without correlation, these events would appear as hundreds or thousands of separate alerts. The sequencer instead generates a single, high-severity incident that clearly describes the progression of the attack.

This approach allows defenders to quickly understand what occurred and take appropriate action without manually correlating logs.

---

10. Experimental Setup

The system was evaluated using both controlled lab experiments and publicly available datasets. This combination allowed testing under realistic conditions while also enabling reproducibility.

10.1 Home Lab

A small home lab was created using an Ubuntu virtual machine configured with OpenSSH. A custom Python script was used to simulate attacker behavior by generating multiple failed SSH login attempts followed by a successful login.

This setup allowed precise control over timing, source IPs, and attack sequences. The Log Analyzer processed the auth.log file in near real-time, while the correlation engine reconstructed the attack chain based on the generated events.

10.2 Dataset Simulation

In addition to the home lab, the CIC-IDS-2017 dataset was used to evaluate the system at scale. The dataset includes labeled attack scenarios, allowing ground-truth comparison.

For the correlation engine, the following columns from the dataset were used:

- Source IP
- Destination Port
- Flow Duration
- Total Forward Packets
- Attack Label

These fields provided sufficient metadata to reconstruct brute force and flooding attack patterns without inspecting packet payloads.

---

11. Results and Analysis

The primary objective of the evaluation was to determine whether the system could reduce alert volume while preserving meaningful security context.

11.1 Alert Reduction Performance

One of the major goals of the system is to mitigate alert fatigue. Traditional systems often generate an alert for every suspicious event, overwhelming analysts.

In contrast, the correlation engine groups related events into attack chains. Table 1 shows the reduction achieved across multiple scenarios.

Dataset Segment	Raw Flows	Correlated Attack Chains	Noise Reduction (%)
SSH-Patator (Brute Force)	5,892	12	99.7%
FTP-Patator	7,935	8	99.8%
Normal Traffic (Noise)	150,000+	0	100%
Manual Lab Test (SSH)	105	1	99.0%

As shown above, the system transforms thousands of low-level events into a small number of actionable alerts.

11.2 Detection Latency

Detection latency was measured in the home lab environment. The Log Analyzer processed authentication logs in near real-time.

The average time from an attacker generating a malicious event to the system producing a correlation alert was under 1.5 seconds. This latency is low enough to enable automated response mechanisms such as firewall rules or account lockouts.

11.3 Case Study: Anatomy of an Attack

During analysis of the CIC-IDS-2017 "Wednesday" dataset, the system identified a single IP address (172.16.0.1) responsible for a brute force attack.

The reconstructed attack chain was as follows:

1. **09:20** – The Log Analyzer flagged over 50 authentication failures from the same source IP.
2. **09:21** – The NetFlowAttackSequencer detected abnormal flow behavior with high packet rates.
3. **Result** – The system generated a single high-severity "SSH Brute Force" alert.

Without correlation, a security analyst would have been presented with thousands of individual alerts, making meaningful response impractical.

---

12. Solution Limitations

While the system performs well under the tested conditions, several limitations of the solution remain and are not addressed in the current solution.

- **Log Evasion**: If an attacker gains root access, they may modify or delete log files. Because the system depends on log integrity, post-compromise activity may go undetected.

- **Encrypted Payloads**:The correlation engine relies on metadata such as timing and packet counts. It cannot inspect encrypted payload contents, which limits visibility into specific commands or credentials.

- **Rule-Based Detection**: The current implementation uses hand-tuned thresholds. While this improves explainability, it may reduce adaptability to novel attack patterns.

---

13. Design Considerations

There were various tradeoffs that were considered during the design of the solution. Following are some of the key points that were considered and the reason for the choice.

- **Latency vs. Sliding Window:** A smaller sliding window ensures faster detection (average <1.5s) but may miss "low and slow" attacks where an attacker waits long intervals between attempts.

- **Sensitivity vs. Alert Volume:** The **two-second Inter-Arrival Time (IAT)** threshold effectively filters automated tools but may result in false negatives for high-skill human attackers who manually type passwords.

- **Explainability vs. Novelty:** By using hand-tuned thresholds, the system provides clear justification for every alert but requires manual updates to detect novel, non-repetitive attack patterns.

- **Integrity vs. Post-Compromise Visibility:** The system is highly effective at detecting the *process* of a breach but relies on log file integrity, which can be compromised if an attacker gains root access.

---

14. Comparative tools & Solutions

There are various other tools and solutions that exist, but each of those solutions has its own set of parameters/boundaries and I have tried to address them in this solution.

- **Log Parsing:** Traditional tools like **Drain** use fixed-depth trees for online log parsing. This solution uses a custom regex-based parser to maintain high explainability and precision for specific SSH and FTP attack patterns.

- **Detection Philosophy:** Most Intrusion Detection Systems (IDS) treat logs as independent incidents, leading to high alert volume. This solution utilizes **temporal correlation**, similar to modern SIEM frameworks, but specifically optimized for **brute-force-to-compromise** sequences.

- **Rule-Based vs. ML:** While deep learning models like **Transformers** or **LSTM** are increasingly used for anomaly detection, they often lack interpretability. This solution prioritizes **rule-based detectors** for transparency and near real-time performance (<1.5s latency).

---

15. Possible Improvements

The following ideas are some of the areas where we can continue to build and improve the solution.

- **Visualization**: A web-based dashboard using frameworks such as Dash or Streamlit that can display attack chains graphically, making analysis easier for defenders.

- **Machine Learning**:The labeled CIC-IDS-2017 dataset could be used to train classifiers that identify anomalous flows not captured by rule-based detectors.

- **Automated Response**:Integration with firewalls or access control systems could enable automatic blocking of confirmed attackers.

---

16. Conclusion

This project demonstrates that analyzing attack behavior over time is more effective than treating logs as isolated events. By combining a Log Analyzer with a NetFlowAttackSequencer, the system reconstructs multi-stage attacks from noisy and fragmented data sources.

The key insight from this work is that time and identity (such as IP addresses) serve as the foundation for security reasoning. While the current system relies on explainable, rule-based logic, it provides a strong framework for future machine learning integration.

This project represents a foundational step toward building smarter, context-aware security operations centers capable of reducing alert fatigue while preserving critical security insight.

Appendix

Unified Event Schema JSON - Log entry is converted into a standard object to ensure interoperability between the Log Analyzer and the NetFlow Sequencer.

```
{
   "$schema": "http://json-schema.org/draft-07/schema#",
   "title": "UnifiedEvent",
   "type": "object",
   "properties": {
      "timestamp": {
         "type": "string",
         "format": "date-time"
      },
      "source_ip": {
         "type": "string",
         "format": "ipv4"
      },
      "event_type": {
         "enum": [
            "AUTH_FAILURE",
            "AUTH_SUCCESS",
            "FLOW_ABNORMAL",
            "SUDO_USE"
         ]
      },
      "user": {
         "type": "string"
      },
      "severity": {
         "type": "integer",
         "minimum": 1,
         "maximum": 5
      }
   },
   "required": [
      "timestamp",
      "source_ip",
      "event_type",
      "severity"
   ]
}
```

Logic Thresholds & Dataset Parameters - The following thresholds were utilized to flag malicious activity during evaluation:

- SSH Brute Force: ≥5 AUTH_FAILURE events from a single source_ip within a T=60s window.

- NetFlow Flooding: Any flow with a Flow Duration <100μs and a Total Forward Packets count exceeding the 95th percentile of baseline normal traffic in the CIC-IDS-2017 dataset.

- Temporal Correlation: An Inter-Arrival Time (IAT) threshold of ≤2.0s was used to differentiate between automated script behavior and manual human input

Benchmarking Environment - System performance was measured in a controlled environment to ensure the <1.5s latency was consistent.

- **Hardware:** NVIDIA GeForce RTX 3050, 12th Gen Intel i5-1240P, 8GB RAM, SSD Storage.
- **Software:** Ubuntu 24.04.3 LTS, Python 3.14, Regex engine (Standard Library).
- **Complexity:** The sequencing logic operates at O(NlogN)for sorting events by timestamp within the sliding window, where N is the number of events per actor.

References

1. Draper-Gil, G., et al. (2016). "Characterization of Encrypted Traffic with any-level Flow Statistics." *Proceedings of the 6th International Conference on Information Systems Security and Privacy.* (Foundational for NetFlow analysis).
2. Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). "Toward a Reliable Dataset for IDS Evaluation." *CIC-IDS-2017.* (The primary dataset used in this paper).
3. He, P., et al. (2017). "Drain: An Online Log Parsing Method with Fixed Depth Tree." *IEEE International Conference on Web Services.* (Contrast to your regex-based parser).
4. Osterweil, L. J. (1987). "Software processes are software too." *Proceedings of the 9th international conference on Software Engineering.* (Conceptual basis for treating security sequences as state-based processes).
5. Chuvakin, A. (2012). "SIEM: The Difference Between Logs and Events." *Gartner Blog.* (Context for "Alert Fatigue" and the need for normalization).