Implementation and Evaluation of an Automation System for SOC Event Detection

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

The rapid growth of digital infrastructure has created both new opportunities and new risks. Organisations rely heavily on interconnected systems, cloud services, and remote access. While these technologies improve flexibility and efficiency, they also increase the number of possible entry points for cyber-attacks and make protecting networks more challenging.

Security teams now depend on Security Information and Event Management (SIEM) platforms and Intrusion Detection Systems (IDS) to monitor events and raise alerts about possible attacks. These tools are essential, but they also produce huge numbers of alerts. Many of these alerts are false positives, which means analysts must spend time checking harmless events while real attacks may be delayed or overlooked. This constant noise creates alert fatigue, slows response times, and increases the risk of missed threats.

This project develops a framework to help security teams handle alerts more effectively. It combines information from past incidents, different types of logs, and external threat intelligence to predict which alerts are likely to be true positives or false positives. The framework reduces unnecessary noise in the system and make it easier to focus on real attacks and improve the overall speed and accuracy of incident response.

1.1 Growth of Enterprise Security Solutions

Enterprise security solutions have evolved significantly to meet the growing challenges of complex IT networks and increasingly sophisticated cyber threats. In the early years, organizations relied mainly on manual log monitoring and basic security tools, which offered limited visibility and slow response to incidents. The introduction of Security Information and Event Management (SIEM) systems marked a major improvement, enabling centralized log collection, automated alerting, and correlation across multiple sources, which helped detect incidents more efficiently.

Over the last decade, SIEM platforms became more advanced, incorporating real-time monitoring, improved correlation, and early machine learning techniques for anomaly detection. Security Operations Centers (SOCs) also matured by implementing structured workflows, integrating multiple security tools, establishing dedicated incident response teams, and adopting automated alert triage processes. These enhancements allowed organizations to handle higher volumes of alerts more efficiently, prioritize critical incidents, and respond faster to genuine threats.

Building on these capabilities, next-generation SIEMs leverage artificial intelligence and advanced analytics to detect subtle anomalies and provide richer contextual insights for faster decision-making. SOC as a Service offering have expanded, providing flexible, cost-effective monitoring and easier integration with threat intelligence feeds and cloud platforms. These advancements allow security teams to respond more proactively, improve accuracy, and focus on high-priority incidents with greater efficiency.

1.2 Overview of the Project

The project focuses on helping SOC teams handle alerts more effectively. It collects data from different sources and uses past incident patterns and threat intelligence to assess each alert. The system predicts which alerts are likely genuine and which are likely false positives. Analysts can review these predictions on a dashboard and update rules based on their decisions. Over time, the system uses this feedback to improve its accuracy, reducing false positives and making it easier for analysts to focus on real threats.

1.3 Problem Statement

SOC Analysts rely on SIEM and IDS tools to detect network threats, but these systems face several challenges. They generate a large number of alerts, many of which are false positives, creating alert fatigue and slowing down response to real incidents. Some existing solutions try to reduce false positives by learning from historical data. However, many rely on complex black-box models, making it difficult for analysts to understand or trust the predictions. Accuracy also remains limited, especially in dynamic network environments or when new attack patterns appear, leaving organizations at risk.

This thesis addresses these challenges by improving alert classification and reducing false positives while maintaining transparency and reliability for SOC analysts.

1.4 Aims and Objectives

This thesis focuses on improving alert management in Security Operations Centers by increasing the accuracy of classification, reducing false positives, and providing clear, actionable information for analysts.

The project seeks to:

- Collect and combine data from multiple sources and threat intelligence feeds to give alerts proper context.
- Analyse historical incident patterns to identify trends and improve prediction accuracy.
- Build a system that predicts whether an alert is likely a True Positive, False Positive, False Negative or False Positive before it reaches analysts.
- Display predictions on a dashboard where analysts can validate results and update detection rules.
- Incorporate analyst feedback to refine the system continuously and enhance its reliability over time.

1.5 Scope and Limitations

This work is centred on improving how Security Operations Centers handle alerts. It looks specifically at reducing false positives and improving classification accuracy for network security events. The framework draws on existing incident logs and external threat intelligence to provide context around alerts. It is built with a feedback loop, so analysts can review predictions, adjust rules, and help refine the system over time. The focus is on enterprise environments, where the volume of alerts can overwhelm teams and slow down responses.

The work is carried out in a controlled environment using configured virtual machines and simulated network traffic. While this setup is suitable for testing the approach, it represents only a subset of possible network configurations. The current implementation is tailored to a specific set of log sources and would require adjustments to operate with other tools, environments, or data formats. The accuracy of the system depends on the quality and relevance of the input data, incomplete logs or outdated threat intelligence can reduce its effectiveness. The approach is designed to improve classification for attack patterns present in the available data and threat intelligence, and patterns outside this scope are not addressed here. The system reduces false positives but cannot eliminate them completely, and its continuous improvement relies on regular analyst interaction to review predictions and update detection rules.

1.6 Structure of the Report

1.6.1 Software methodology used

The project follows a hybrid Agile-Waterfall development methodology, combining upfront planning with iterative feature development. The overall system architecture, critical requirements, and compliance constraints were defined at the start using Waterfall principles, ensuring a stable foundation for log collection, alert processing, and dashboard visualization. Following this, the system was developed in short Agile cycles, with each cycle focusing on a specific functionality, such as data collection, alert prediction, and visualization. Within each cycle, features were designed, implemented, and evaluated, allowing improvements and adjustments to be incorporated progressively.

This hybrid approach was suitable for the project because SOC environments are dynamic, and system requirements may evolve while integrating multiple data sources and prediction logic. Iterative development enabled the project to handle large volumes of log data and complex alert processing efficiently, while upfront planning ensured critical system requirements were met. Regular evaluation at the end of each cycle resulted in a more adaptable, reliable, and well-structured alert management framework.

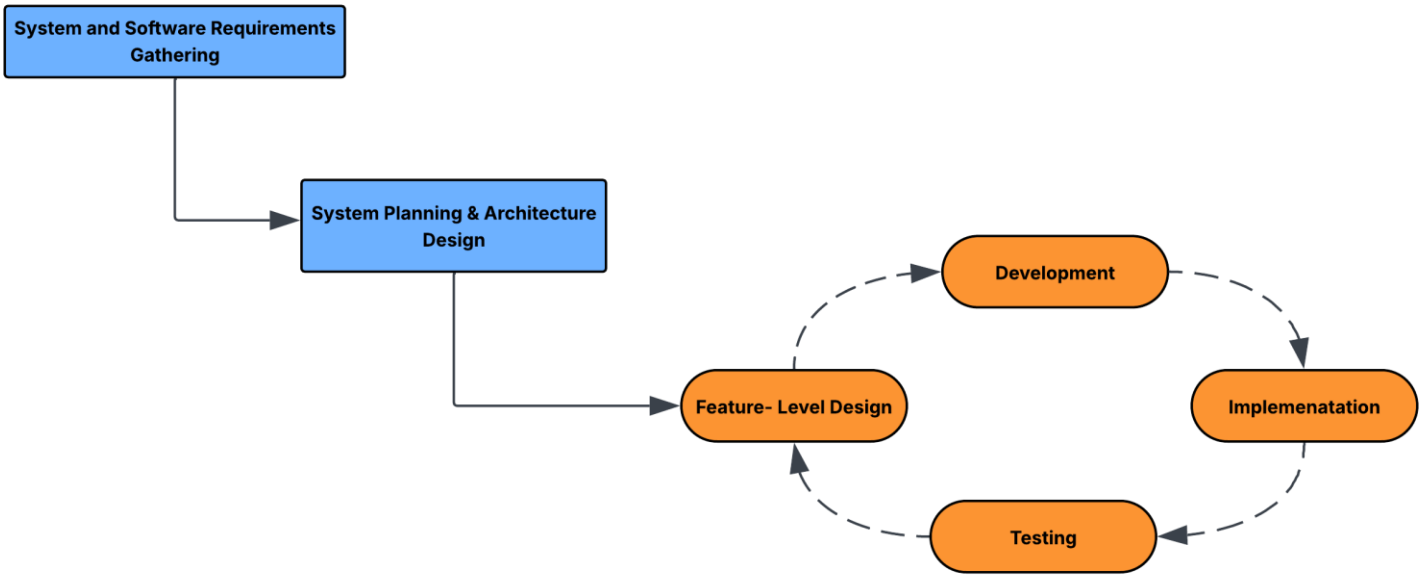

Figure 1 Hybrid Agile-Waterfall Development Model for SOC Alert Management

1.6.2 Report format

The chapters in this report correspond to the phases of the hybrid Agile-Waterfall model adopted for this project. Each chapter explains the processes and activities undertaken, detailing how the system was developed according to this methodology.

Chapter 2: System Design

2.1 System Design

This project implements a feedback-based detection system to provide end-to-end monitoring and analysis of network and endpoint activity within a controlled virtual lab. Network traffic is monitored using the Suricata intrusion detection system (IDS), while endpoint agents capture host-level events such as process activity and network connections. These data sources provide the foundation for detecting suspicious behaviour and potential threats in the virtual environment.

Collected logs and alerts are sent to a central server, where they are parsed, normalised, and enriched with relevant threat intelligence from sources like MISP and internal allowlists. This ensures that alerts carry sufficient context for prioritisation and further analysis. Centralised log management allows events from multiple sources to be correlated, forming a complete picture of activity across the virtual lab.

The decision engine evaluates each alert and assigns it a classification: true positive, true negative, false positive, or false negative. This high-level classification guides analysts in prioritising genuine threats while reducing unnecessary noise. The engine uses rules and historical patterns to make consistent, reliable decisions without overcomplicating the design.

Analysts review the classified alerts through a dashboard and provide feedback, which is used to refine detection logic and improve accuracy over time. This feedback-driven approach enables continuous improvement of the system, reducing false positives and increasing confidence in threat detection. Figure 2 illustrates the overall system design, showing the flow from monitoring and alert generation to log analysis, decision-making, and feedback-based refinement. The design emphasizes accuracy, efficiency, and adaptability while remaining suitable for experimentation within a virtual environment.

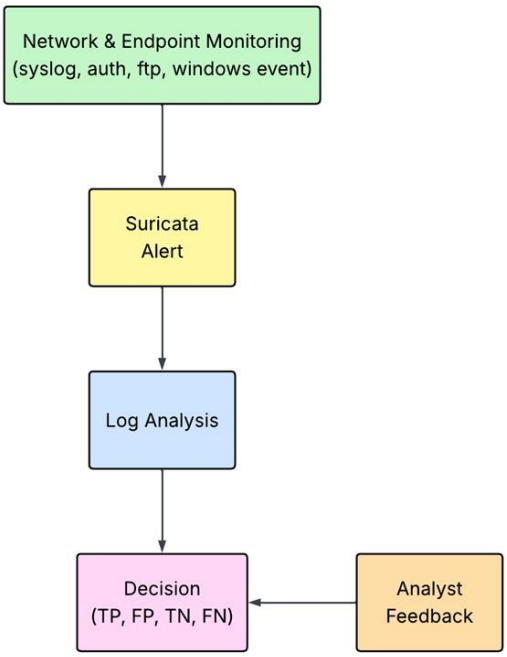

Figure 2 System Design

2.2 Virtual Environment Setup

A controlled virtual environment will be designed to simulate a realistic corporate network, including user endpoints, servers, monitoring tools, and simulated external threats. It will generate both normal and malicious activity to evaluate the proposed framework for reducing false positives in alerts, while focusing on realism, safety, and scalability, and remaining fully isolated from external networks. The lab will allow flexible testing of different attack scenarios and alert types, supporting interaction between monitoring systems, intrusion detection tools, and threat intelligence platforms for cohesive observation and analysis.

The network topology will consist of internal and external zones separated by a firewall. The internal zone will represent the organizational environment, generating normal network and user activity, while the external zone will simulate potential attackers introducing controlled malicious traffic. The firewall will regulate traffic between the zones and ensure safe testing conditions.

This setup will provide a safe and realistic environment to test how the system responds to different situations. It will allow observation of both normal and malicious activity, helping to see how accurately the framework detects threats and how effectively it handles alerts. The lab will also support refining and improving the system, ensuring it performs reliably in practical scenarios.

2.3 Data Collection and Enrichment

This section describes how the system will collect, organize, and standardize security-related logs for analysis. The virtual environment will capture everyday system activity as well as controlled attack scenarios, helping to assess the system's ability to differentiate between normal actions and security threats.

Logs will be gathered from multiple sources within the virtual environment, including user workstations, servers, firewalls, monitoring tools, and simulated attacker machines. External threat intelligence feeds, containing indicators of compromise such as malicious IP addresses, domains, and file hashes, will be integrated to enrich the collected information. The logs are intended to cover routine user activity, server processes, network flows, and controlled malicious events such as reconnaissance, exploitation attempts, brute-force attacks, malicious and unsecure URLs, and File Transfers.

Collected logs will be forwarded from endpoints and servers to a central system for aggregation, enrichment, and analysis. The collected logs in the central system will be structured and accessible, with key attributes such as IP addresses, ports, event types, timestamps, severity levels, and threat intelligence indicators available for correlation and alert analysis.

2.4 Correlation Logic and Event Classification

This section outlines how collected logs will be analysed to identify meaningful events and classify alerts. The system is designed to combine internal context, such as user activity and server behaviour, with external threat intelligence to highlight suspicious or malicious patterns. Correlation logic will link related events across multiple sources, providing a clearer view of potential threats.

Alerts will be classified into four categories: true positives (TP) for confirmed malicious activity, true negatives (TN) for benign activity correctly identified as safe, false positives (FP) for benign actions incorrectly flagged as threats, and false negatives (FN) for malicious events that go undetected. The classification process will consider attributes from the logs including source and destination IP addresses, ports, event types, timestamps, and severity levels along with contextual information from threat intelligence feeds.

The correlation and classification logic is designed to support iterative refinement. Observed patterns and feedback within the virtual environment will guide adjustments to improve accuracy, reduce false alerts, and ensure critical threats are reliably identified. This approach allows thorough evaluation of the system's ability to correctly classify events, minimize errors, and provide actionable insights.

2.5 Feedback Mechanism and Analyst Dashboard

This section describes how the system will use feedback from analysts to refine detection accuracy and improve alert handling. Alerts generated through correlation and classification will be reviewed by analysts via a central dashboard, which provides an organized view of events, their severity, and contextual information from internal logs and external threat intelligence.

The dashboard will allow analysts to validate alerts, confirming true positives (TP) and true negatives (TN), while identifying false positives (FP) and false negatives (FN). This feedback will be recorded and integrated into the system to adjust correlation logic, update alert criteria, and fine-tune detection rules. By capturing human judgment in a structured manner, the system will continuously learn to reduce false alerts without missing genuine threats.

The feedback process is designed to be iterative. Patterns observed through analyst validation will guide updates to classification criteria and correlation methods, gradually improving the system's performance. The dashboard will also provide summaries and visualizations to help analysts focus on critical alerts, track trends, and evaluate how well the detection framework is performing in the controlled environment.

This approach creates a closed-loop system where human expertise and automated detection work together. It ensures that alerts become more accurate over time, response is more efficient.

2.6 Required Resources

The project uses a combination of hardware and software resources to create a realistic and fully functional environment. Hardware provides sufficient computing power for continuous log generation and processing, while software tools enable monitoring, detection, visualization, and threat simulation. Together, they support accurate evaluation of detection workflows and system performance under both normal and malicious activity.

2.6.1 Hardware Resource

The project uses multiple virtual machines, set up in Oracle VirtualBox, to represent internal user workstations, servers, and an external attacker system. Each VM is configured with sufficient resources to ensure smooth operation and continuous log generation, collection, and analysis. Typical configurations allocate 4-8 GB of RAM and 50–90 GB of storage per VM, with CPU and network capacity sized to handle simultaneous processes and data flows, supporting realistic simulation of both normal user activity and controlled malicious events.

2.6.2 Software Resources

This project will utilize a combination of open-source tools and platforms, due to their reliability, efficiency, and widespread use in cybersecurity, log analysis, and automation. These tools will support the collection, enrichment, correlation, and analysis of security-related logs, as well as the simulation of alert response workflows.

- ***Operating Systems:*** The lab is setup with Windows 10, Ubuntu Server 24, Ubuntu Desktop 22 and 24 , and Kali Linux to represent a realistic corporate IT environment, like internal endpoints, servers, and an external attacker system.

- ***Security Tools:*** The setup uses security tools like Suricata, along with Winlogbeat and Filebeat, to capture network and system activity and forward structured logs to a central repository, forming the backbone of the detection and monitoring setup. Elasticsearch and Kibana handled aggregation, indexing, and visualization of the collected logs and alerts, enabling efficient correlation, analysis, and monitoring of system performance. MISP supplied external Indicators of Compromise to enrich alerts, helping the system more accurately distinguish between true positives and false positives.

- ***Automation Libraries:*** Python libraries such as Streamlit are used to create interactive dashboards for monitoring alerts, pandas helped organize and process log data, MySQL managed databases and stored alert information, matplotlib was used to visualize trends and patterns, and Elasticsearch APIs allowed querying, indexing, and handling of logs. Together, these tools made it easier to enrich alerts, process data, visualize activity, and test response workflows efficiently.

Chapter 3: System Setup and Configuration

This chapter outlines the technical setup of the system, detailing the configuration and interconnection of the virtual machines. It describes the installation and deployment of Suricata, Elasticsearch, Kibana, MISP, and MySQL, and explains how logs and alerts are collected, forwarded, and integrated across these components. The chapter provides the foundation for the practical implementation of monitoring, analysis, and testing of security events.

3.1 Virtual Lab Environment setup and Configuration

The virtual lab environment was created to simulate a realistic and controlled network for monitoring, logging, and threat intelligence analysis. Multiple virtual machines were deployed, each assigned specific roles and resources to support the integrated system. The environment was designed to allow smooth communication between components while remaining isolated from external networks, providing a secure and controlled setting.

3.1.1 Network Topology

The lab environment's virtual network configuration is illustrated in Figure 3. The network is connected to the internet and protected by a firewall which serves as the main point of control for incoming and outgoing traffic.

The traffic is routed to two VLANs an internal and external VLAN. The internal VLAN hosts the SIEM, MISP platform, an Ubuntu server acting as a central organization server, and a Windows VM representing a regular user workstation. These machines generate regular system operations and logs for monitoring and analysis. The external VLAN contains a Kali Linux attacker machine, which is used to simulate malicious activities.

This setup creates a realistic environment for testing. It allows controlled simulation of attacks and shows how traffic moves between the external attacker, the internal systems, and the internet, providing a clear view of the network for evaluating threat detection and monitoring.

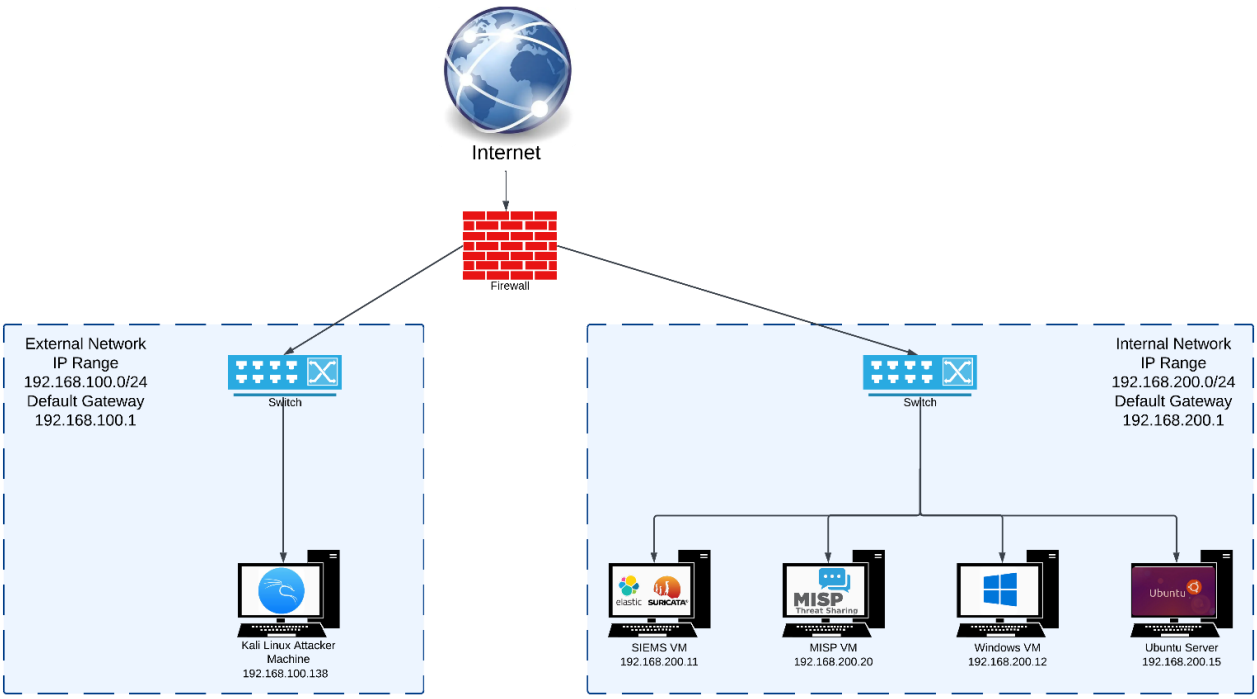

Figure 3 Network Diagram

3.1.2 Network Setup and Connectivity

The routing and connectivity of each virtual machine in the lab were carefully configured to ensure reliable communication within the internal and external VLANs, as well as controlled access to the internet through the WAN.

3.1.2.1 Router VM

The router VM serves as the central gateway, managing traffic between the WAN, internal VLAN, and external VLAN. Network interfaces were configured using the Netplan configuration file at **"sudo nano /etc/netplan/50-cloud-init.yaml"**. The WAN interface used DHCP for internet access, while the internal and external VLAN interfaces were assigned static IP addresses — 192.168.200.1/24 for the internal network and 192.168.100.1/24 for the external network (Figure 4). These addresses served as the default gateways for the respective VLANs, enabling controlled routing of traffic between networks and towards the internet.

IP forwarding was enabled to allow inter-VLAN and WAN traffic, and Network Address Translation (NAT) was implemented to securely route internal VM traffic to external networks while preserving private Ips. Forwarding rules were applied to regulate outgoing traffic, and all configurations were made persistent to maintain connectivity after system reboots (Figure 5).

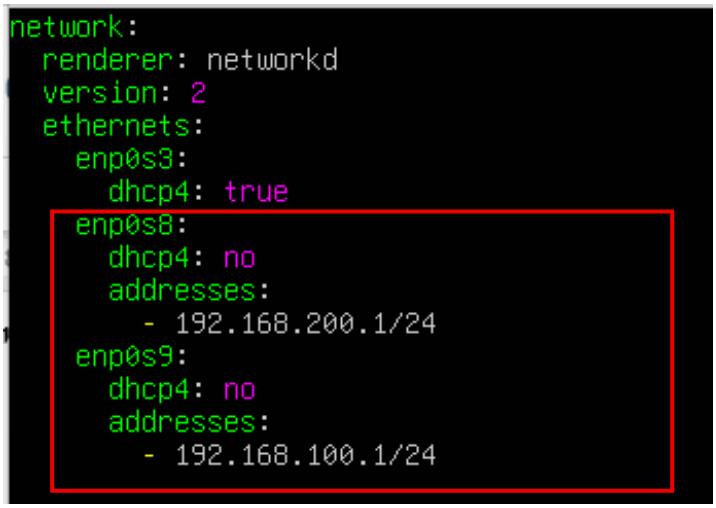

Figure 4 Router VM Network Routing and Configuration

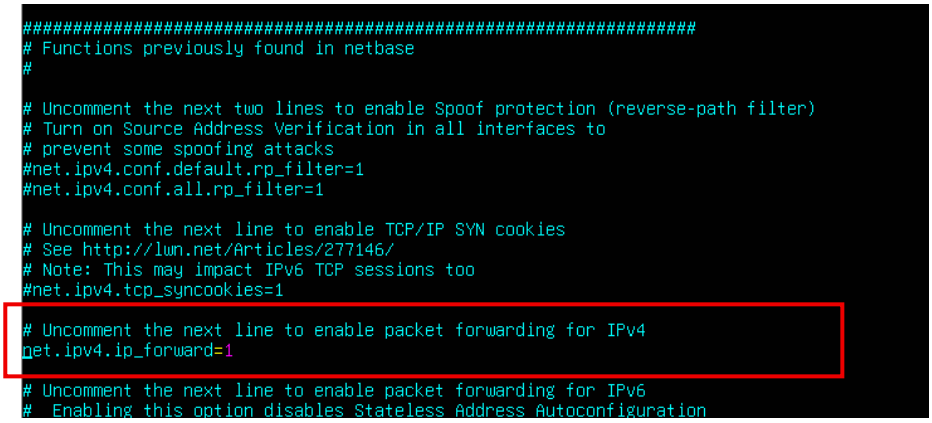

Figure 5 Enabling IP Forwarding

This setup ensures the router effectively isolates network segments while providing controlled WAN access for internal and external communications.

3.1.2.2 SIEMS, MISP and Ubuntu Server VM

Internal VMs were configured with static IP addresses to ensure predictable and uninterrupted communication within the internal VLAN. The Ubuntu server VM (Figure 6) was assigned 192.168.200.15, the SIEM VM (Figure 7) was set to 192.168.200.11, and the MISP VM (Figure 8) was set to 192.168.200.20. The SIEM and Ubuntu server network configurations were managed via **"sudo nano /etc/netplan/01-network-manager-all.yaml"** , while the MISP VM used **"sudo nano /etc/netplan/50-cloud-init.yaml"**. Changes were applied to activate the new network settings.

These configurations enabled the internal VMs to communicate seamlessly with each other and with the router.

```
  GNU nano 6.2        /etc/netplan/01-network-manager-all.yaml
# Let NetworkManager manage all devices on this system
network:
  version: 2
  renderer: NetworkManager
  ethernets:
   enp0s3:
    dhcp4: no
    addresses:
     - 192.168.200.15/24
    routes:
     - to: default
       via: 192.168.200.1
```

Figure 6 Ubuntu Server Network Configuration

```
  GNU nano 6.2        /etc/netplan/01-network-manager-all.yaml
# Let NetworkManager manage all devices on this system
network:
  version: 2
  renderer: networkd
  ethernets:
   enp0s3:
    dhcp4: no
    addresses:
     - 192.168.200.11/24
    routes:
     - to: default
       via: 192.168.200.1
```

Figure 7 SIEMS VM Config Network Configuration

```
  GNU nano 7.2              /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
         - 192.168.200.20/24
      routes:
        - to: default
          via: 192.168.200.1
```

Figure 8 MISP VM Network Configuration

3.1.2.3 Windows VM

The Windows VM was configured with a static IP address to maintain consistent connectivity within the internal VLAN. The network adapter was updated with the IP 192.168.200.12 and default gateway, allowing communication with the router and other internal VMs (Figure 9). The Windows Defender Firewall was temporarily disabled on the internal network to allow network testing and monitoring. After configuration, the Windows VM could communicate seamlessly with the internal systems.

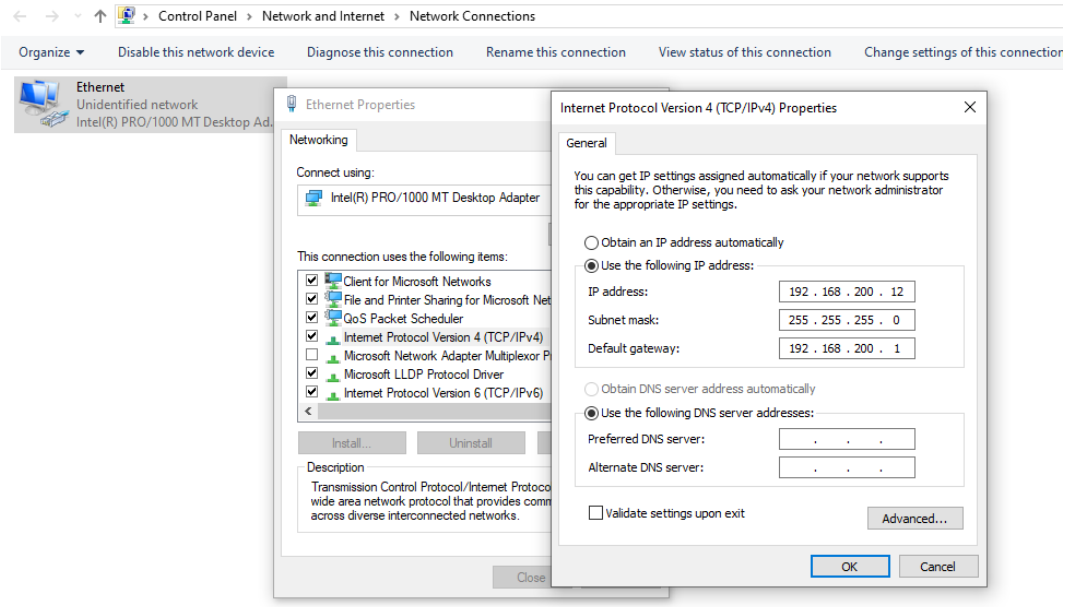

Figure 9 Windows VM Network Configuration

3.1.2.4 Attacker (Kali Linux) VM

The Kali Linux VM was configured with a static IP 192.168.100.138 in the external VLAN. The network configuration was configured via the **"sudo nano /etc/network/interfaces"** file (Figure 10), and the changes were activated by restarting the networking service using **"sudo systemctl restart networking"**.

This configuration allowed the attacker VM to communicate with the router and simulate external threats, while maintaining isolation from the internal VLAN. It provided a controlled environment for testing attack scenarios.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto eth0
iface eth0 inet static
address 192.168.100.138
netmask 255.255.255.0
gateway 192.168.100.1
```

Figure 10 Kali Linux VM Network Configuration

3.2 Suricata Installation and Configuration

Suricata, the network intrusion detection(IDS), is systematically configured to monitor network traffic and detect potential threats. Installation includes configuring the network interfaces, setting up log files, and loading rule sets. Suricata inspects traffic in real time and generates alerts, helping to distinguish between genuine threats and benign activity.

3.2.1 Suricata Installation

Suricata was installed using the steps outlined in the official Suricata guide to ensure a stable and fully supported deployment (Suricata, n.d.). Before installation, system packages were updated and upgraded using **"sudo apt-get update && sudo apt-get upgrade -y"** to ensure compatibility and stability. Before installing Suricata, essential dependencies were installed to support real-time packet inspection and processing. These included **libnetfilter-queue-dev** and **libnetfilter-queue1** for kernel packet queue interaction, **libnfnetlink-dev** and **libnfnetlink0** for Netfilter integration, and **jq** for parsing Suricata's JSON output, ensuring proper operation of the IDS (Figure 11). After installing the dependencies, the Suricata stable repository was added to the system to ensure access to the latest stable and supported version of the tool (Figure 12). Suricata was then installed using the system's package manage as seen in figure 13.

```
nayomi@SIEMS:~$ sudo apt -y install libnetfilter-queue-dev libnetfilter-queue1 libnfnetlink-dev libnfnetlink0 jq
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libnfnetlink0 is already the newest version (1.0.1-3build3).
libnfnetlink0 set to manually installed.
The following additional packages will be installed:
  libdpkg-perl libfile-fcntllock-perl libjq1 libonig5 pkg-config
Suggested packages:
  debian-keyring gcc | c-compiler binutils git bzr libnetfilter-queue-doc dpkg-dev
The following NEW packages will be installed
  jq libdpkg-perl libfile-fcntllock-perl libjq1 libnetfilter-queue-dev libnetfilter-queue1 libnfnetlink-dev libonig5 pkg-config
0 to upgrade, 8 to newly install, 0 to remove and 0 not to upgrade.
```

Figure 11 Installing dependencies for Suricata

```
nayomi@SIEMS:~$ sudo add-apt-repository ppa:oisf/suricata-stable
Repository: 'deb https://ppa.launchpadcontent.net/oisf/suricata-stable/ubuntu/ jammy main'
Description:
Suricata IDS/IPS/NSM stable packages
https://suricata.io/
https://oisf.net/

Suricata IDS/IPS/NSM - Suricata is a high performance Intrusion Detection and Prevention System and Network Security Monitoring engine.

Open Source and owned by a community run non-profit foundation, the Open Information Security Foundation (OISF). Suricata is developed
```

Figure 12 Suricata Repository

```
nayomi@SIEMS:~$ sudo apt install suricata
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libevent-core-2.1-7 libevent-pthreads-2.1-7 libhiredis0.14 libhyperscan5 libluajit-5.1-common libnet1
The following NEW packages will be installed
  libevent-core-2.1-7 libevent-pthreads-2.1-7 libhiredis0.14 libhyperscan5 libluajit-5.1-common libnet1 suricata
0 to upgrade, 7 to newly install, 0 to remove and 0 not to upgrade.
Need to get 7,278 kB of archives.
After this operation, 33.8 MB of additional disk space will be used.
```

Figure 13 Installing Suricata

3.2.2 Suricata Configuration

The main configuration file, **/etc/suricata/suricata.yaml**, was configured to match the lab environment and ensure comprehensive monitoring of all network endpoints. The home network was defined to include all internal IP ranges, while external traffic was set to include all other sources, allowing Suricata to observe traffic across the entire network and distinguish trusted internal activity from external sources (Figure 14). Paths for rule files were configured to include both predefined and custom rules, enabling detection of established threats as well as scenarios specific to the lab environment (Figure 16). The network interface **enp0s3** was selected to capture traffic from the network range, ensuring that all relevant communications were monitored (Figure 15). Log file locations were configured to ensure that alerts and system events are recorded in an organized and easily accessible for review and analysis (Figure 17). The rule sets were updated using the suricata-update utility to ensure the latest threat signatures were applied, and the Suricata service was started and enabled to run in the background, providing continuous monitoring that persists after reboots (Figure 18 & 19).

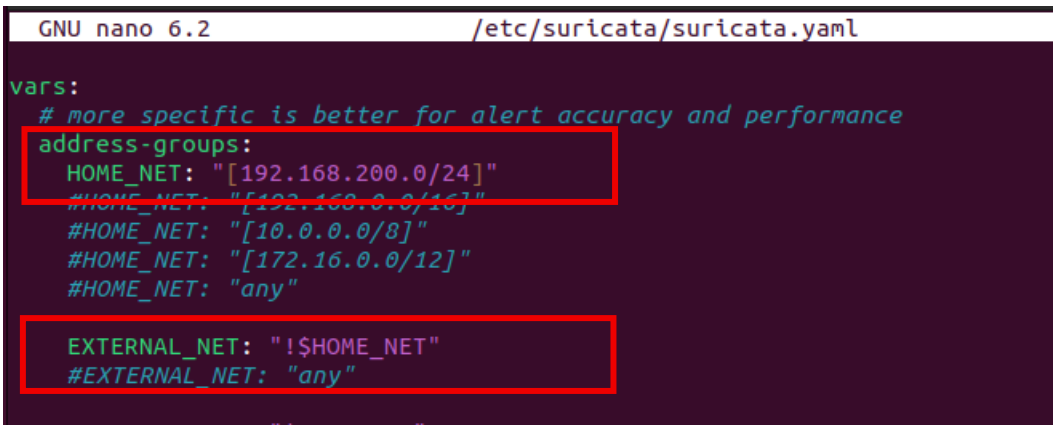

Figure 14 Configuration of IP Address range in Suricata

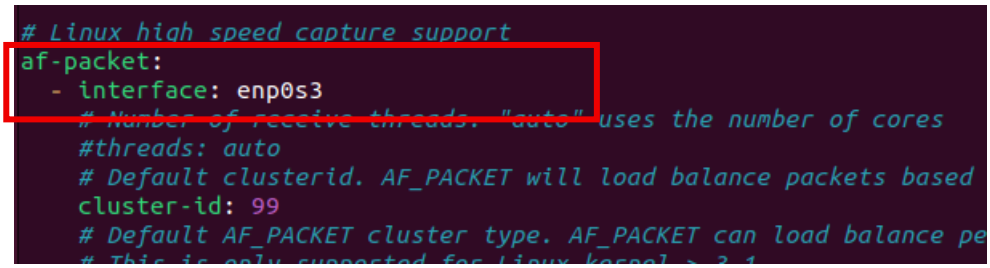

Figure 15 Configuration of Network interface in Suricata

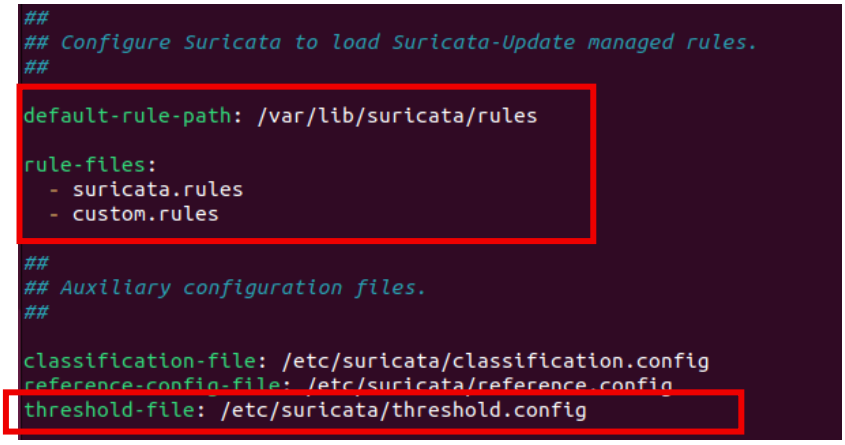

Figure 16 Configuration of Rule File Paths in Suricata

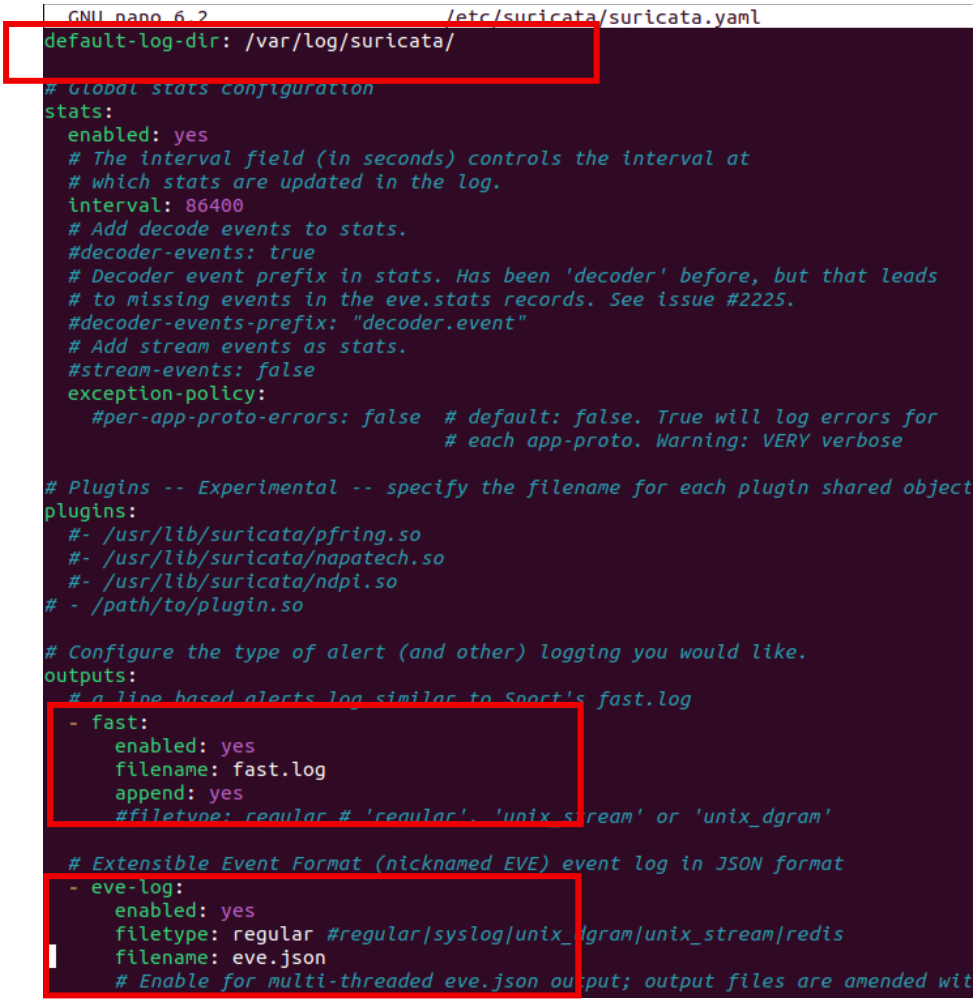

Figure 17 Configuration of Log Paths in Suricata

```
nayomi@SIEMS:~$ sudo suricata-update
24/7/2025 -- 14:20:32 - <Info> -- Using data-directory /var/lib/suricata.
24/7/2025 -- 14:20:32 - <Info> -- Using Suricata configuration /etc/suricata/suricata.yaml
24/7/2025 -- 14:20:32 - <Info> -- Using /usr/share/suricata/rules for Suricata provided rules.
24/7/2025 -- 14:20:32 - <Info> -- Found Suricata version 8.0.0 at /usr/bin/suricata.
24/7/2025 -- 14:20:32 - <Info> -- Loading /etc/suricata/suricata.yaml
24/7/2025 -- 14:20:32 - <Info> -- Disabling rules for protocol pgsql
24/7/2025 -- 14:20:32 - <Info> -- Disabling rules for protocol modbus
24/7/2025 -- 14:20:32 - <Info> -- Disabling rules for protocol dnp3
24/7/2025 -- 14:20:32 - <Info> -- Disabling rules for protocol enip
24/7/2025 -- 14:20:32 - <Info> -- No sources configured, will use Emerging Threats Open
24/7/2025 -- 14:20:32 - <Info> -- Fetching https://rules.emergingthreats.net/open/suricata-8.0.0/emerging.rules.tar.gz.
```

Figure 18 Update Suricata

```
nayomi@SIEMS:~$ sudo systemctl start suricata
nayomi@SIEMS:~$ sudo systemctl enable suricata
```

Figure 19 Start and enable suricata

This setup provides continuous visibility into network activity, helping to detect potential threats quickly and support informed decisions. By keeping alerts organized and events clearly logged, the system provides a reliable foundation for continuous threat monitoring.

3.3 Elasticsearch Setup and Configuration

Elasticsearch acted as the central data store and indexing engine, providing fast search and correlation capabilities for logs collected from multiple sources, including Suricata, Filebeat, and Winlogbeat. It played a key role in the adaptive feedback loop by normalizing all incoming data and storing it securely. This setup ensured efficient handling of large volumes of log data while maintaining accurate and timely insights for detection and analysis.

3.3.1 Elasticsearch Installation

Elasticsearch v9.0.3 was installed as a core component of the log management and analysis stack. The official Debian package was obtained directly from the Elastic website using the **wget** utility (Figure 20). This ensured that the latest stable version, was retrieved from a trusted source. Once downloaded, the package was installed using the **sudo dpkg -i** command, which handled the unpacking and integration of Elasticsearch into the SIEMS VM environment (Figure 21).

```
nayomi@SIEMS1:~$ wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-9.0.3-amd64.deb
--2025-07-21 14:35:39--  https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-9.0.3-amd64.deb
Resolving artifacts.elastic.co (artifacts.elastic.co)... 34.120.127.130, 2600:1901:0:1d7::
Connecting to artifacts.elastic.co (artifacts.elastic.co)|34.120.127.130|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 650233914 (620M) [application/vnd.debian.binary-package]
Saving to: 'elasticsearch-9.0.3-amd64.deb'

elasticsearch-9.0.3-amd64.de 100%[=========================================>] 620.11M  6.19MB/s    in 90s

2025-07-21 14:37:09 (6.92 MB/s) - 'elasticsearch-9.0.3-amd64.deb' saved [650233914/650233914]
```

Figure 20 Downloading Elasticsearch in SIEMS VM

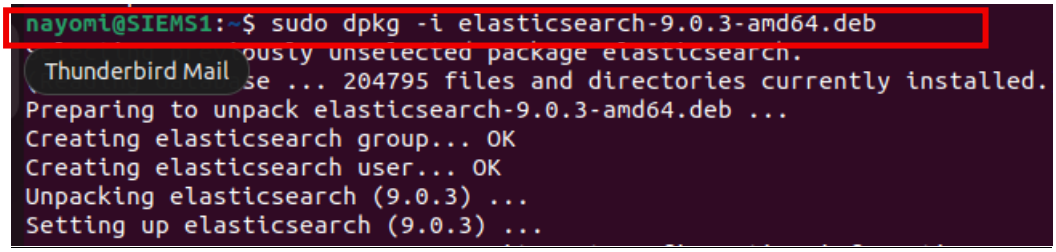

Figure 21 Installing Elasticsearch

During the installation, Elasticsearch automatically enabled security features and generated initial credentials for the elastic superuser (Figure 22). Along with the username and password, the installation also provided a file path that can be used to generate enrollment tokens, enabling secure connections for Kibana and additional Elasticsearch nodes. An **enrollment token** is a secure, time-limited token that allows Kibana and Elasticsearch nodes to join the cluster without exposing administrative credentials. It ensures that only authorized services can connect to Elasticsearch securely, establishing trust and encrypted communication. This leads to safe integration of other services without compromising the credentials.

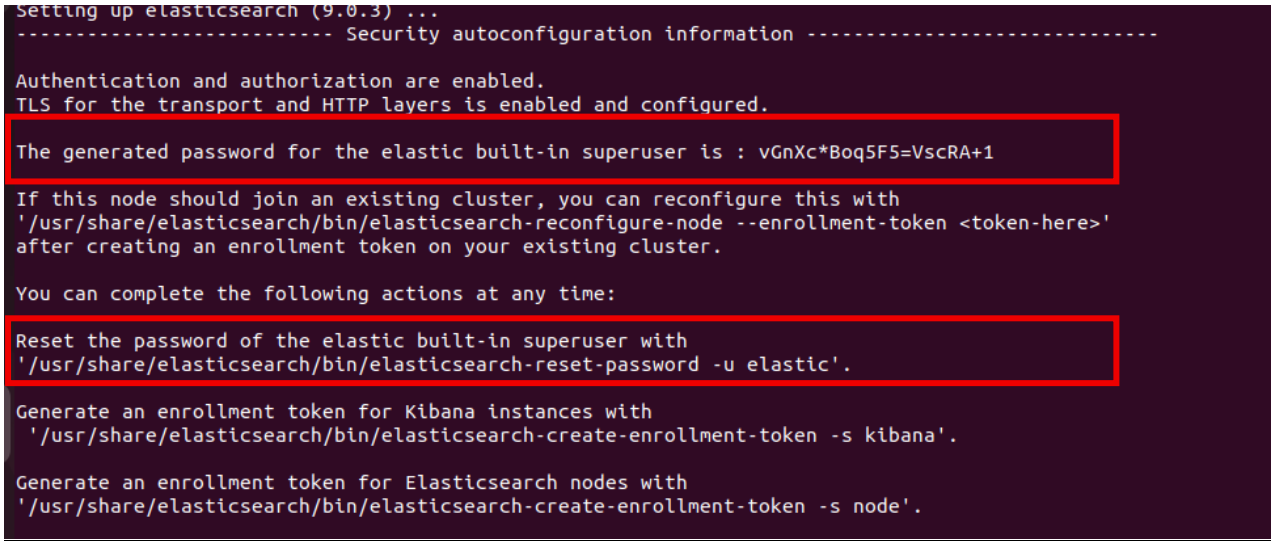

Figure 22 Elasticsearch Security Information

3.3.1 Elasticsearch Configuration

For configuring Elasticsearch, the main configuration file located at **"/etc/elasticsearch/elasticsearch.yml"** was updated to set **network.host** to **192.168.200.11** and **http.port** to **9200** (Figure 23). These settings ensured that the Elasticsearch node could be accessed over the network and respond to HTTP requests from other services and clients. After updating the configuration, the Elasticsearch service was started and enabled to run at boot using **"sudo systemctl start**

elasticsearch” and **“sudo systemctl enable elasticsearch”**. This made sure that the node was operational, would automatically start after system reboots.

```
# ---------------------------------- Network -----------------------------------
#
# By default Elasticsearch is only accessible on localhost. Set a different
# address here to expose this node on the network:
#
network.host: 192.168.200.11
#
# By default Elasticsearch listens for HTTP traffic on the first free port it
# finds starting at 9200. Set a specific HTTP port here:
#
http.port: 9200
#
# For more information, consult the network module documentation.
#
```

Figure 23 Elastics Configuration

To ensure that the configuration was applied correctly and that the Elasticsearch node was functioning as expected, connectivity was tested using the curl command. The -k option was used to bypass certificate verification, and the -u option provided the elastic superuser credentials. The response displayed important details about the node, including its name (SIEMS1), cluster name (elasticsearch), cluster UUID, version information, and Lucene version (Figure 24). This confirmed that the node was running properly, accessible securely over HTTPS, and ready to interact with clients such as Kibana, Filebeat, and Winlogbeat for log ingestion, indexing, and analysis. This configuration ensured secure communication between Elasticsearch and other services while providing centralized storage and search capabilities for all collected logs.

```
nayomi@SIEMS1:~$ curl -k -u elastic:NDBNPsfs9egQNJe-9Nwp "https://192.168.200.11:9200"
{
  "name" : "SIEMS1",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "kr1xmE6mTt6dfm6c8isTdw",
  "version" : {
    "number" : "9.0.3",
    "build_flavor" : "default",
    "build_type" : "deb",
    "build_hash" : "cc7302afc8499e83262ba2ceaa96451681f0609d",
    "build_date" : "2025-06-18T22:09:56.772581489Z",
    "build_snapshot" : false,
    "lucene_version" : "10.1.0",
    "minimum_wire_compatibility_version" : "8.18.0",
    "minimum_index_compatibility_version" : "8.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Figure 24 Testing Elasticsearch connectivity

3.4 Kibana Setup and Configuration

Kibana was set up as the visualization and management interface for Elasticsearch, providing a user-friendly way to explore, analyze, and monitor the collected log data. Through its graphical interface, analysts could easily search, filter, and visualize logs from multiple sources, helping them quickly spot patterns, detect anomalies, and identify potential security threats.

3.4.1 Installing Kibana

Kibana v9.0.3 was installed as the visualization and management interface for the log analysis stack. The official Debian package was downloaded directly from the Elastic website using the **wget** command, ensuring that the latest stable version was obtained from a trusted source (Figure 25). After downloading, the package was installed using **sudo dpkg -i**, which unpacked Kibana and integrated it into the SIEMS VM environment (Figure 26).

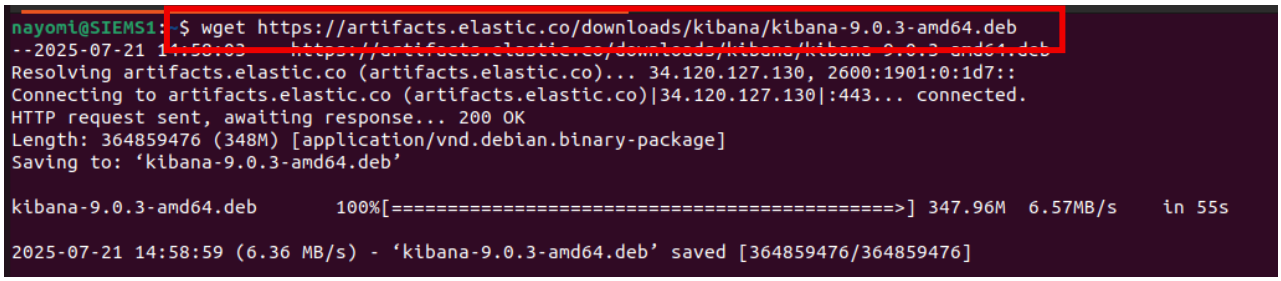

Figure 25 Downloading Kibana in SIEMS VM

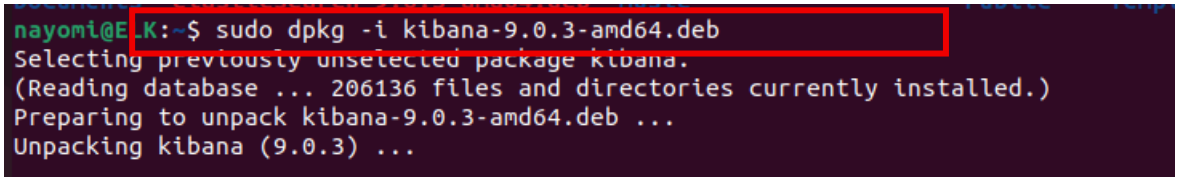

Figure 26 Installing Kibana

3.4.2 Configuring Kibana

The Kibana configuration file is located at **"/etc/kibana/kibana.yml"** and was updated to set the server parameters. The **server.host** was set to **192.168.200.11** so that Kibana would be reachable from the network at that IP address, and the **server.port** was set to **5601** to define the port through which users could access the dashboard (Figure 27). These settings ensure that Kibana runs as a network service and can communicate with Elasticsearch and clients securely. After saving these settings, the Kibana service was started and enabled using systemctl, ensuring it would automatically launch on system startup (Figure 28).

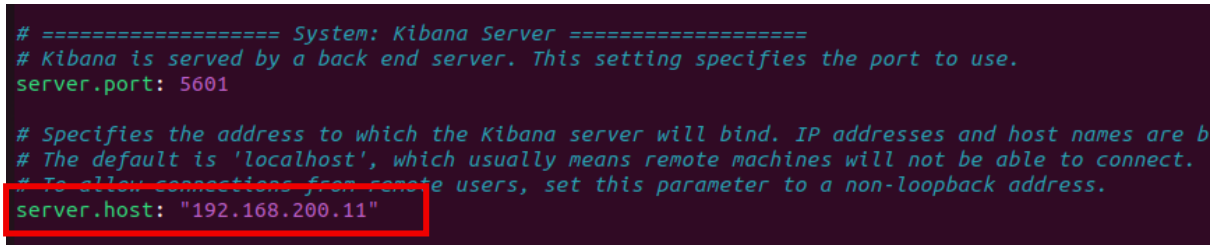

Figure 27 Kibana Configuration File

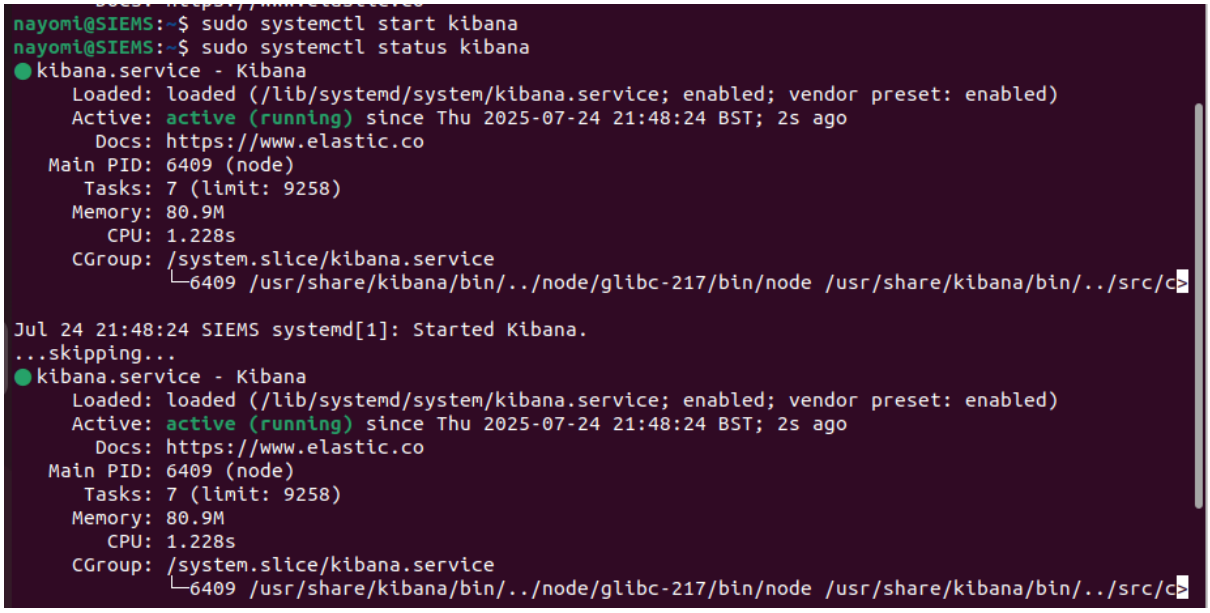

Figure 28 Start Kibana service

3.4.3 Loggin into Kibana

After completing the Kibana installation, the next step was to securely connect it to the Elasticsearch cluster. For this an enrollment token was generated specifically for Kibana using the command **“sudo /usr/share/elasticsearch/bin/elasticsearch-create-enrollment-token --scope kibana”** (Figure 29). This produces a secure, time-limited token that allows Kibana to join the Elasticsearch cluster without exposing administrative credentials. The token ensures that only authorized services can connect and communicate securely with Elasticsearch.

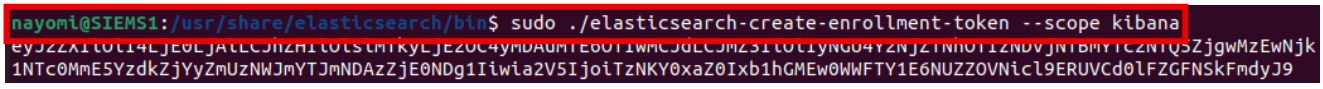

Figure 29 Kibana Enrollment token

Once the enrollment token was obtained, it was used in the Kibana setup process. During the initial login, Kibana required the token to establish trust with the Elasticsearch node (Figure 30). After providing the token, Kibana prompted for a one-time password (OTP) which was generated by **“sudo /usr/share/kibana/bin/kibana-verification-code”** , this adds an extra layer of security by ensuring that only an administrator with access to the system can complete the setup (Figure 31).

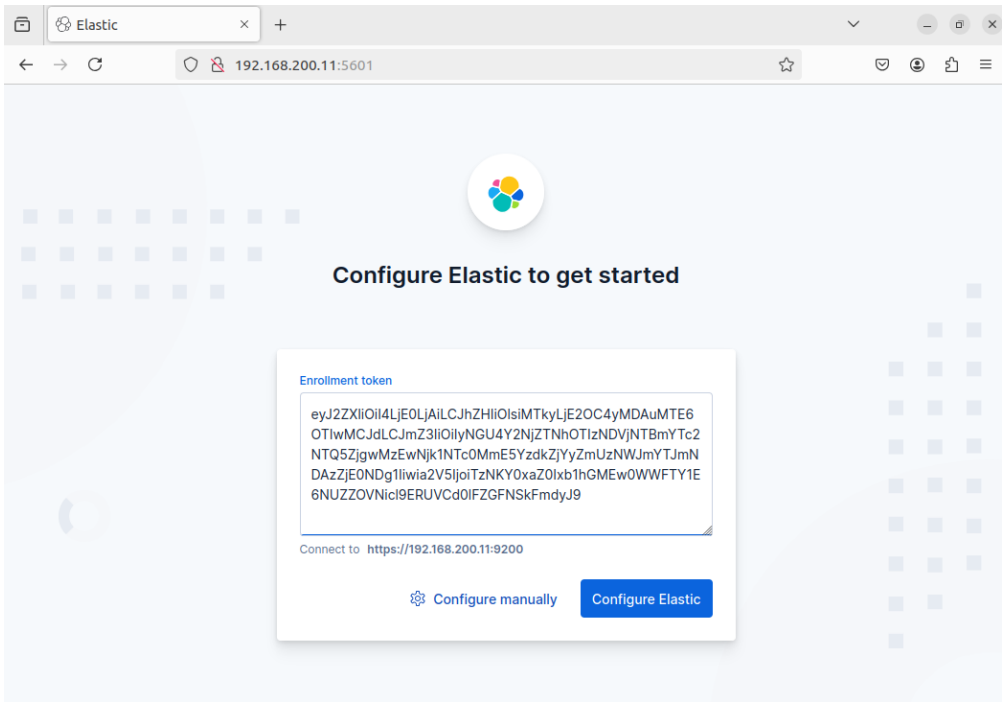

Figure 30 Adding token to Kibana

```
nayomi@SIEMS:~$ sudo /usr/share/kibana/bin/kibana-verification-code
Your verification code is:  512 835
nayomi@SIEMS:~$
```

Figure 31 Generating verification code for Kibana

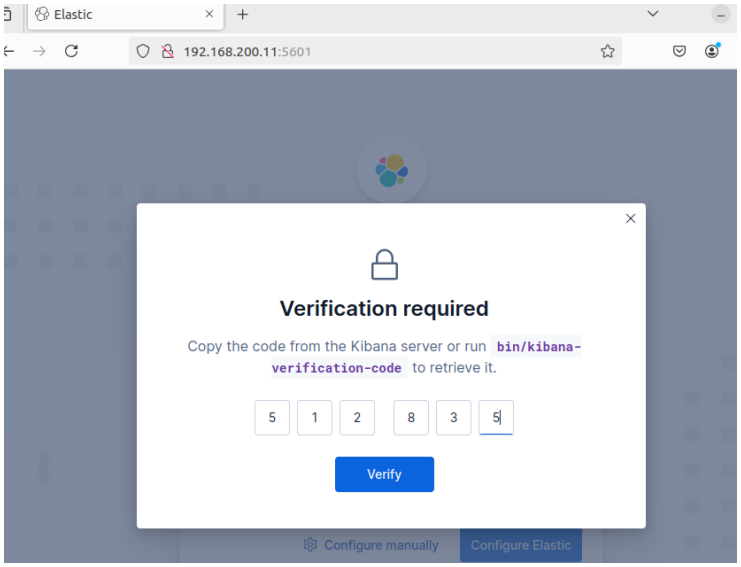

Figure 32 Kibana OTP verification

After entering the OTP, the initial credentials allowed successful login (Figure 32). Kibana could then be accessed via **https://192.168.200.11:5601**, providing a secure graphical interface to explore, search, and visualize logs stored in Elasticsearch (Figure 33).

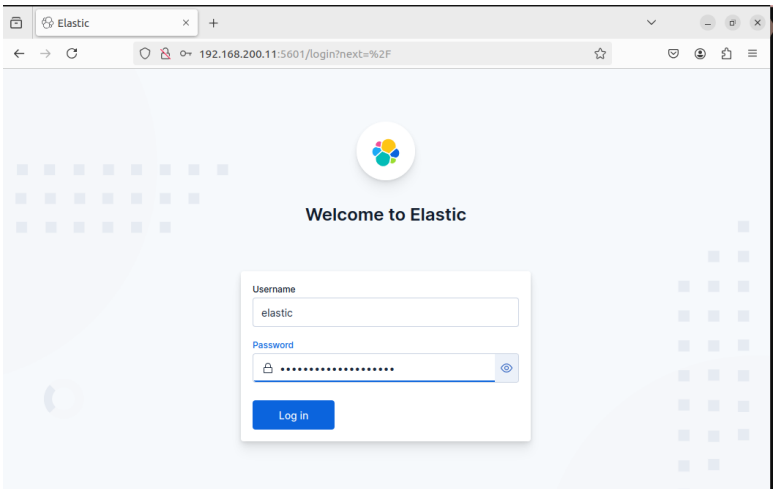

Figure 33 Login into Kibana

After logging in, a security alert prompted the generation of encryption keys (Figure 34), which were created using **"sudo /usr/share/kibana/bin/kibana-encryption-keys generate**". These keys were then added to the kibana-keystore to securely store sensitive information such as session data and credentials (Figure 35). Once the keys were added, the Kibana service was restarted to apply the encryption settings. This ensures that the sensitive data handled by Kibana remains confidential and protected, maintaining both system integrity and secure communication.

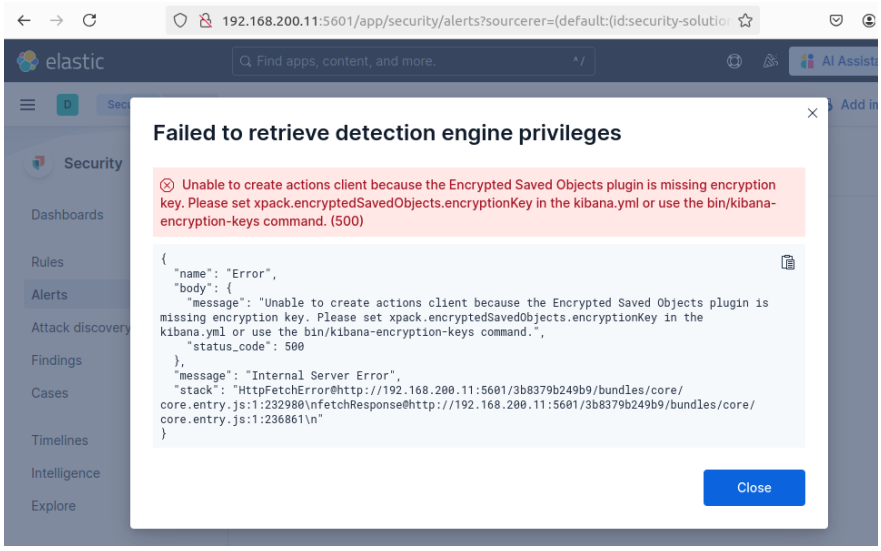

Figure 34 Kibana Security Alert

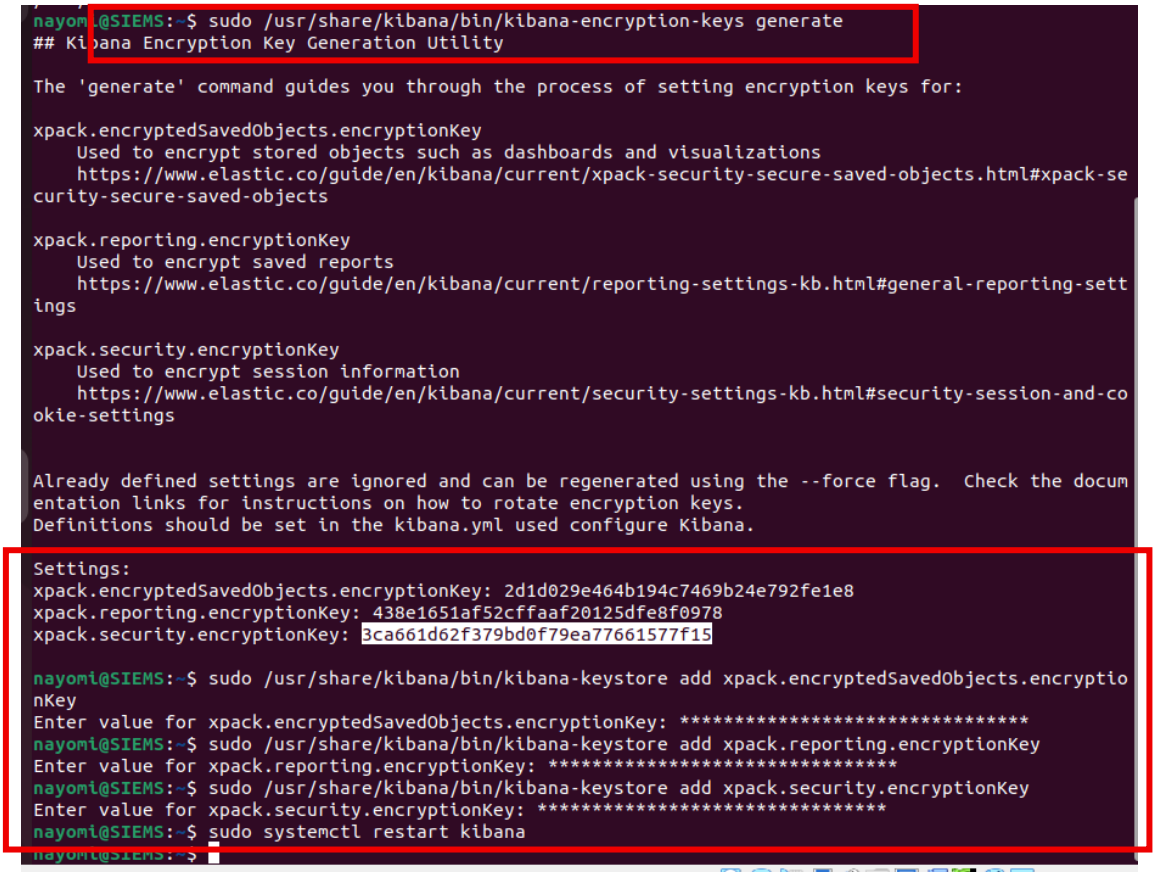

Figure 35 Kibana encryption keys

Kibana was successfully deployed and fully secured, providing a reliable and user-friendly interface for interacting with Elasticsearch. With encrypted communication, controlled access, and built-in visualization capabilities, it served as a central platform for exploring, analyzing, and monitoring log data from multiple sources, helping to turn raw events into actionable insights.

3.5 Installing and Configuring Beats

To collect and forward logs from endpoints and servers, Beats are deployed on all virtual machines in the environment. These lightweight agents efficiently capture system and network activity, structure the data, and forward it to a centralized Elasticsearch platform. This configuration provides continuous visibility into both routine operations and simulated attacks, enabling effective correlation, analysis, and visualization of events in Kibana. A detailed list of the logs gathered through this setup can be found in **Appendix A**.

3.5.1 Filebeat

Filebeat is deployed on the MISP, SIEM, and Ubuntu Server VMs following the same installation procedure, enabling it to collect logs from each system and provide comprehensive visibility for monitoring and evaluating detection workflows.

3.5.1.1 Filebeat Installation

The first step in deploying Filebeat involved downloading the official Debian package from Elastic's website. This was done using the **wget** command followed by the package URL, which retrieved the installer directly to each VM (Figure 36). This ensured that the correct and latest version of Filebeat was downloaded securely and consistently across all systems.

Once the package was downloaded, it was installed using the Debian package manager with the command **"sudo dpkg -i filebeat-9.0.4-amd64.deb"** (Figure 37), which unpacked all Filebeat files into the appropriate directories, set up the necessary permissions, and configured Filebeat as a service on the system. This process also ensured that all required dependencies were installed, allowing Filebeat to run reliably in the background.

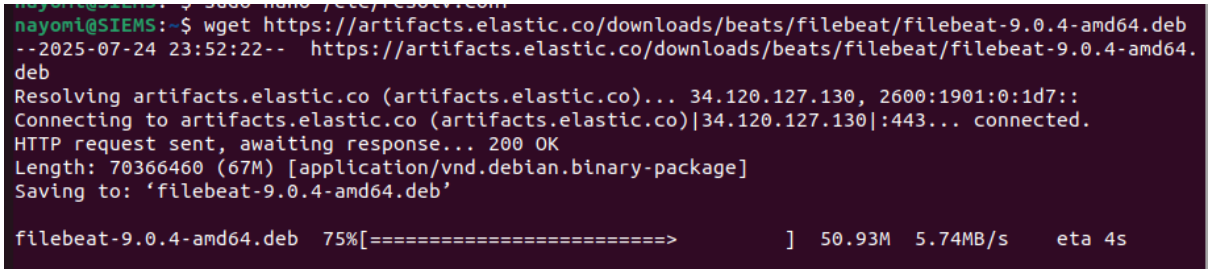

Figure 36 Downloading Filebeat in the VMs

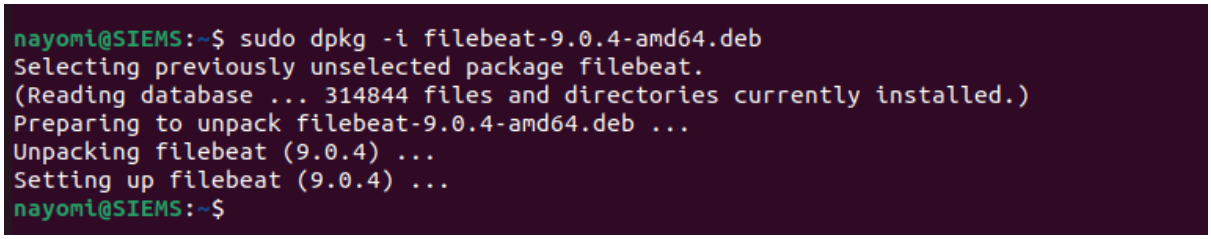

Figure 37 Installing Filebeat in the VMs

3.5.1.2 Filebeat configurations

Filebeat was configured to forward logs securely to the central Elasticsearch and Kibana setup, ensuring all collected activity could be monitored and analysed in real time. In the main configuration file **"/etc/filebeat/filebeat.yml"**, the Kibana output was set to the host IP **192.168.200.11** on port **5601**, enabling dashboards and visualizations to reflect real-time log activity (Figure 38). The Elasticsearch output was configured with the host IP **192.168.200.11** on port **9200**, with the Elasticsearch username and password to authenticate access. HTTPS was enabled for encrypted communication, and the verification mode was configured to ensure trusted connections between Filebeat and Elasticsearch (Figure 39). These changes ensured that all logs collected from the VMs were transmitted securely and could be indexed and visualized accurately.

```
# ================================== Kibana ===================================

# Starting with Beats version 6.0.0, the dashboards are loaded via the Kibana API.
# This requires a Kibana endpoint configuration.
setup.kibana:

  # Kibana Host
  # Scheme and port can be left out and will be set to the default (http and 5601)
  # In case you specify and additional path, the scheme is required: http://localhost:5601/path
  # IPv6 addresses should always be defined as: https://[2001:db8::1]:5601
  host: "192.168.200.11:5601"
```

Figure 38 Filebeat Configuration for Kibana Output

```
# ---------------------------- Elasticsearch Output ----------------------------
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["192.168.200.11:9200"]

  # Performance preset - one of "balanced", "throughput", "scale",
  # "latency", or "custom".
  preset: balanced

  # Protocol - either `http` (default) or `https`.
  protocol: "https"

  #certificate
  ssl.verification_mode: "none"

  # Authentication credentials - either API key or username/password.
  #api_key: "id:api_key"
  username: "elastic"
  password: "s4+qxaEymW3C8oI=qXDw"
```

Figure 39 Filebeat Configuration for Elasticsearch Output

The appropriate Filebeat modules were enabled on each virtual machine to ensure structured log collection. On the SIEM VM, the Suricata module was activated to capture alerts generated by the intrusion detection system, while the system module was enabled on both the MISP and Ubuntu Server VMs to collect authentication logs and syslog events. In the configuration files **"/etc/filebeat/modules.d/suricata.yml"** and **"/etc/filebeat/modules.d/system.yml"**, the modules were set to "**enabled: true**" (Figure 40 & 42). The modules were then activated using the commands **"sudo filebeat modules enable suricata"** and **"sudo filebeat modules enable system"**, allowing Filebeat to automatically parse and structure the log data (Figure 41 & 43) .

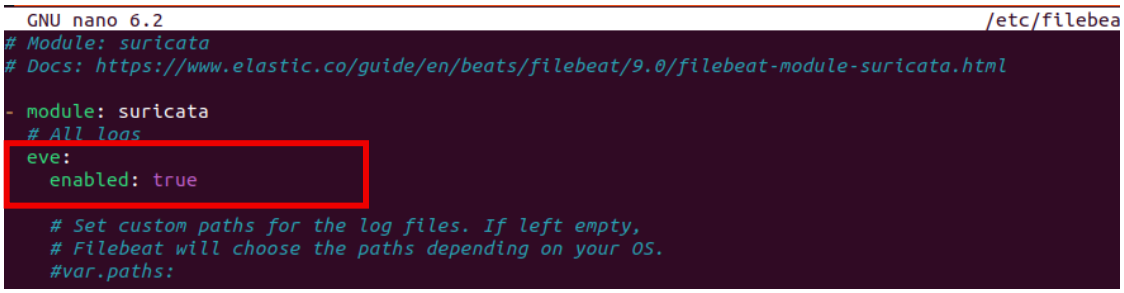

Figure 40 Enabling Suricata Module in Filebeat

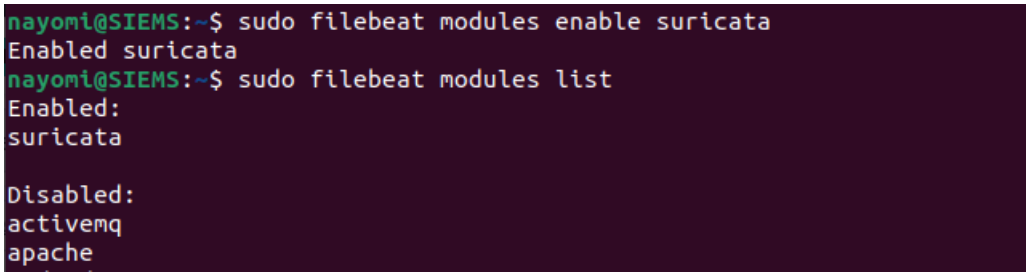

Figure 41 Activating Suricata Module in Filebeat

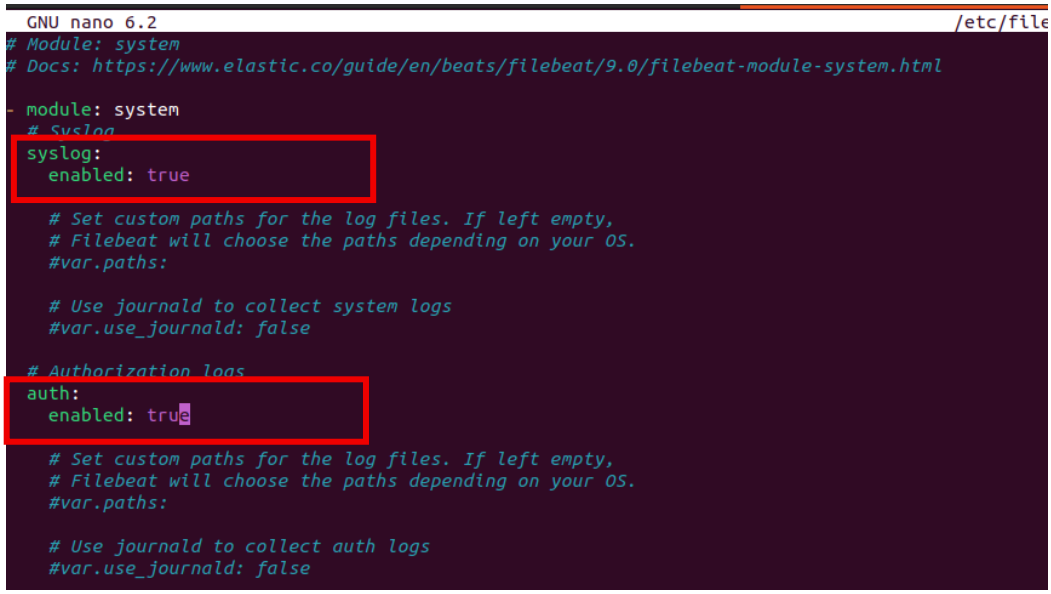

Figure 42 Enabling System Module in Filebeat

```
nayomi@victim:~$ sudo filebeat modules enable system
[sudo] password for nayomi:
Enabled system
nayomi@victim:~$ sudo nano /etc/filebeat/modules.d/system.yml
nayomi@victim:~$ sudo filebeat modules list
Enabled:
system

Disabled:
activemq
apache
```

Figure 43 Activating System Module in Filebeat

The configuration was then verified using **"sudo filebeat test config"**, and the connection to Elasticsearch was tested with **"sudo filebeat test output"** (Figure 44). After successful verification, the Filebeat service was started and enabled to run at boot, ensuring continuous log collection and forwarding (Figure 45). This setup allowed all logs to be collected in a consistent format, ready for analysis and visualization in Elasticsearch and Kibana.

```
nayomi@SIEMS:~$ sudo filebeat test config
Config OK
nayomi@SIEMS:~$ sudo filebeat test output
elasticsearch: https://192.168.200.11:9200...
  parse url... OK
  connection...
    parse host... OK
    dns lookup... OK
    addresses: 192.168.200.11
    dial up... OK
  TLS...
    security... WARN server's certificate chain verification is disabled
    handshake... OK
    TLS version: TLSv1.3
    dial up... OK
  talk to server... OK
  version: 9.0.3
```

Figure 44 Testing config and connectivity in Filebeat

```
nayomi@victim:~$ sudo systemctl enable filebeat
sudo systemctl start filebeat
sudo systemctl status filebeat
Created symlink /etc/systemd/system/multi-user.target.wants/filebeat.service → /lib/systemd/system/filebeat.service.
● filebeat.service - Filebeat sends log files to Logstash or directly to Elasticsearch.
     Loaded: loaded (/lib/systemd/system/filebeat.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2025-07-25 14:53:13 BST; 22ms ago
       Docs: https://www.elastic.co/beats/filebeat
   Main PID: 4891 (filebeat)
      Tasks: 1 (limit: 3443)
     Memory: 19.1M
        CPU: 13ms
     CGroup: /system.slice/filebeat.service
             └─4891 /usr/share/filebeat/bin/filebeat --environment systemd -c /etc/filebeat/filebeat.yml --path.home /u
```

Figure 45 Start and Enable Filebeat

3.5.2 Winlogbeat

Winlogbeat is deployed on the Windows VMs, it collect and forward event logs from various channels, including Security, System, and Application, enabling centralized monitoring and analysis of Windows endpoints for security, auditing, and operational insights.

3.5.2.1 Winlogbeat Installation

Winlogbeat was downloaded from the official Elastic website as a Windows package and extracted to the **C:/Program Files** directory. To install it as a Windows service, PowerShell was launched with administrative privileges, and the command **"PowerShell.exe -ExecutionPolicy Unrestricted -File .\install-service-winlogbeat.ps1"** was executed. Once the script completed, the Winlogbeat service was installed successfully (Figure 46).

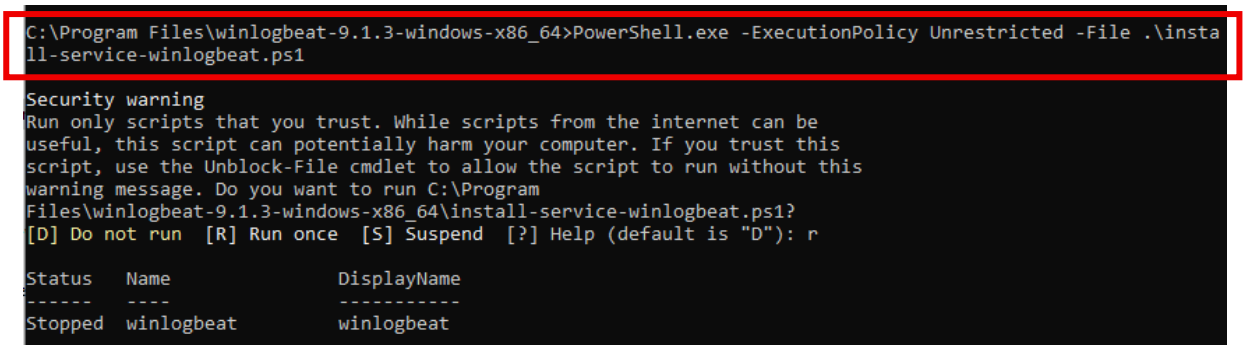

Figure 46 Installing Winlogbeat

3.5.2.2 Winlogbeat configuration

Winlogbeat was configured to collect relevant Windows event data from the virtual machine designated for endpoint monitoring. The configuration defined multiple event channels to ensure comprehensive visibility into system and security activities. Core channels such as Application, System, and Security were specified to capture operational events, service modifications, and authentication records. To extend detection capabilities, the Microsoft-Windows-Sysmon/Operational channel was included to provide detailed telemetry on process creation, network connections, and file modifications. Additionally, Windows PowerShell and Microsoft-Windows-PowerShell/Operational logs were configured with selected event IDs (400, 403, 600, 800, 4103, 4104, 4105, 4106) to monitor script execution, which is often leveraged in attack scenarios (Figure 47). The output settings for Elasticsearch and Kibana were aligned with those used for Filebeat (section 5.5.1.2), ensuring that all collected logs were forwarded to the same centralized Elasticsearch cluster (Figure 48). This uniform setup allowed the logs to be indexed, stored, and queried consistently. Finally,

the Winlogbeat service was started and managed directly through the Windows Services console (Figure 49).

```
winlogbeat.event_logs:
  - name: Application
    ignore_older: 72h

  - name: System

  - name: Security

  - name: Microsoft-Windows-Sysmon/Operational

  - name: Windows PowerShell
    event_id: 400, 403, 600, 800

  - name: Microsoft-Windows-PowerShell/Operational
    event_id: 4103, 4104, 4105, 4106

  - name: ForwardedEvents
    tags: [forwarded]
```

Figure 47 Winlogbeat configuration for event logs

```
# ================================== Kibana ===================================

# Starting with Beats version 6.0.0, the dashboards are loaded via the Kibana API.
# This requires a Kibana endpoint configuration.
setup.kibana:

  # Kibana Host
  # Scheme and port can be left out and will be set to the default (http and 5601)
  # In case you specify and additional path, the scheme is required: http://localhost:5601/path
  # IPv6 addresses should always be defined as: https://[2001:db8::1]:5601
  host: "192.168.200.11:5601"

  # Kibana Space ID
  # ID of the Kibana Space into which the dashboards should be loaded. By default,
  # the Default Space will be used.
  #space.id:

# ================================== Outputs ===================================

# Configure what output to use when sending the data collected by the beat.

# ---------------------------- Elasticsearch Output ----------------------------
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["192.168.200.11:9200"]

  # Protocol - either `http` (default) or `https`.
  protocol: "https"
  ssl.verification_mode : "none"

  # Authentication credentials - either API key or username/password.
  #api_key: "id:api_key"
  username: "elastic"
  password: "vO*YWo0MrsAys8n3c7IH"

  # Pipeline to route events to security, sysmon, or powershell pipelines.
  pipeline: "winlogbeat-%{[agent.version]}-routing"
```

Figure 48 Winlogbeat Configuration File

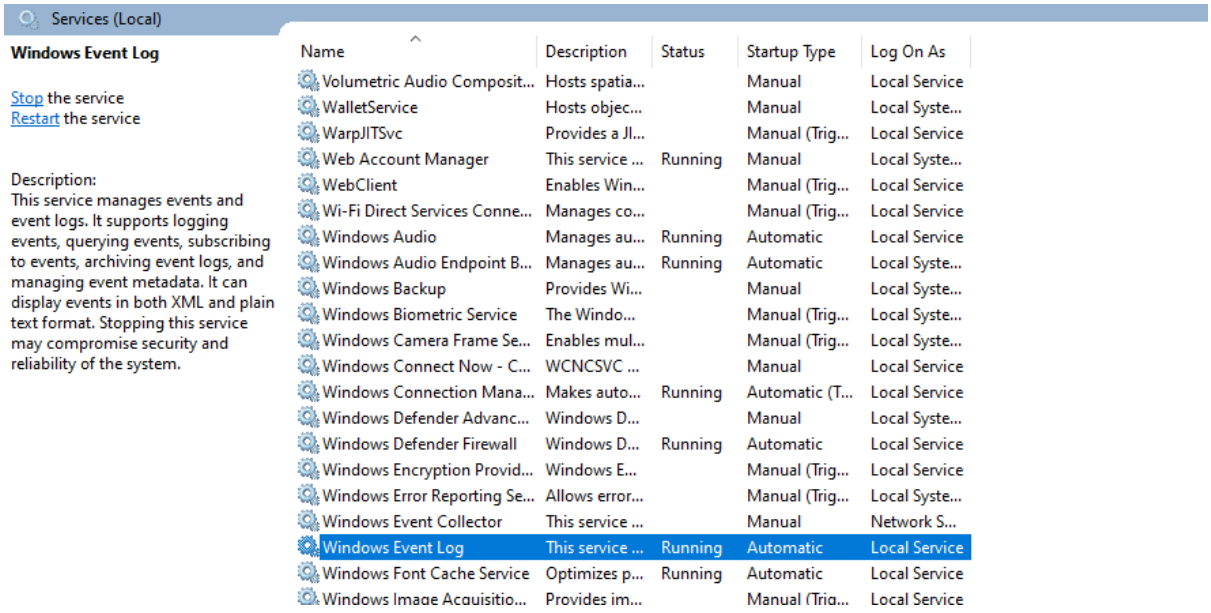

Figure 49 Starting Winlogbeat via Windows Service

3.5.3 Pipeline Creation

In Elasticsearch and Filebeat, a pipeline is a series of steps through which incoming log data passes to be parsed, enriched, and formatted before it is stored. Setting up pipelines is important because raw, unstructured logs from different sources can be inconsistent; pipelines normalize the data, remove unnecessary fields, and add useful metadata, making it easier to search, analyze, and visualize. To create these pipelines, the **command sudo filebeat setup --pipelines --dashboards --index-management** was executed, which automatically configured ingest pipelines, installed prebuilt Kibana dashboards, and set up index management (Figure 50)t. Once the setup was complete, all Filebeat and Winlogbeat instances from the networked VMs could forward their logs through the same processing workflow, ensuring consistent indexing, structured storage, and centralized visualization of all collected data (Figure 51 & 52).

```
nayomi@SIEMS:~$ sudo filebeat setup --pipelines --dashboards --index-management
Overwriting lifecycle policy is disabled. Set `setup.ilm.overwrite: true` to overwrite.
Index setup finished.
Loading dashboards (Kibana must be running and reachable)
Loaded dashboards
Loaded Ingest pipelines
nayomi@SIEMS:~$
```

Figure 50 Filebeat pipeline Creation

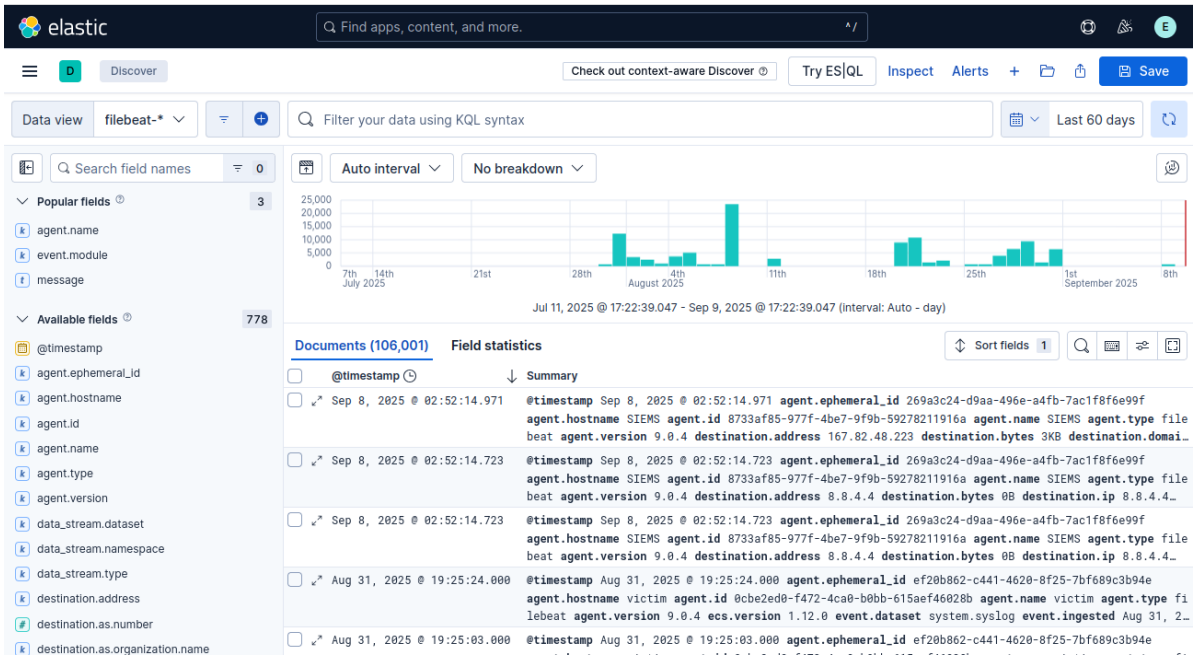

Figure 51 Filebeat logs in Kibana Dashboard

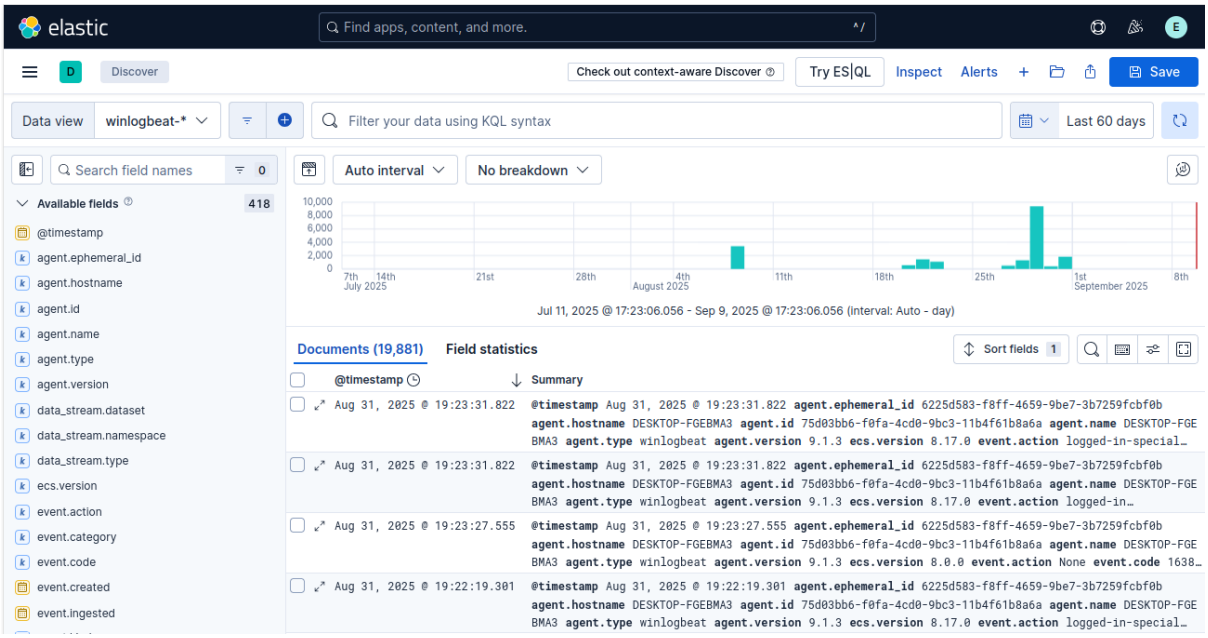

Figure 52 Winlogbeat data in Kibana Dashboard

3.6 MISP Installation and Event Configuration

MISP (Malware Information Sharing Platform) is a threat intelligence platform designed to collect, organize, and share indicators of compromise and other relevant threat data. In the lab environment, MISP is used to manage threat intelligence, provide realistic indicators for testing detection systems, and correlate this information with Suricata to help distinguish genuine threats from benign activity.

3.6.1 MISP installation

MISP was installed on the MISP VM to serve as a centralized platform for collecting, organizing, and sharing threat intelligence. The official installation script from the MISP GitHub repository was used to ensure a stable and supported deployment on Ubuntu (Figure 53). The script automates the installation of all required dependencies, configures the web server and database, and sets appropriate permissions, minimizing manual setup (Figure 54). The installation process involved downloading the official script, granting it execution permissions, and then running it to install MISP with administrative privileges, allowing the system to become fully operational. The system provided a URL and login credentials to access the MISP web interface, and the default password was updated to a secure value to ensure controlled access (Figure 55 & 56).

```
nayomi@MISP:~$ wget https://raw.githubusercontent.com/MISP/MISP/refs/heads/2.5/I
NSTALL/INSTALL.ubuntu2404.sh
--2025-07-18 00:37:35--  https://raw.githubusercontent.com/MISP/MISP/refs/heads/
2.5/INSTALL/INSTALL.ubuntu2404.sh
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.1
33, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.
133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 34604 (34K) [text/plain]
Saving to: 'INSTALL.ubuntu2404.sh'

INSTALL.ubuntu2404. 100%[===================>]  33.79K  --.-KB/s    in 0.02s

2025-07-18 00:37:36 (1.67 MB/s) - 'INSTALL.ubuntu2404.sh' saved [34604/34604]
```

Figure 53 MISP Installation Script from GitHub

```
misp@MISP:~$ chmod +x INSTALL.ubuntu2404.sh
misp@MISP:~$ sudo ./INSTALL.ubuntu2404.sh
```

```
v2.5 Setup on Ubuntu 24.04 LTS
[STATUS] Updating base system...
[OK] Base system update successfully completed.
[STATUS] Installing apt packages (git curl python3 python3-pip python3-virtualenv apache2 zip gcc sudo binutils openssl
supervisor)...
[OK] git installation successfully completed.
[OK] curl installation successfully completed.
[OK] python3 installation successfully completed.
```

Figure 54 Installing MISP

```
[OK] JSON structures ingestion successfully completed.
[OK] Apache restart successfully completed.
[OK] Settings configured.
[STATUS] Finalising MISP setup...
[NOTICE] Settings saved to /var/log/misp_settings.txt
[NOTICE] You can now access your MISP instance at https://misp.local
[NOTICE] The default admin credentials are:
[NOTICE] Username: admin@admin.test
[NOTICE] Password: PtWgRfu63JAvqKSCQuypZaoXM2nrn3QZ
[NOTICE] MISP setup complete. Thank you, and have a very safe, and productive day.
```

Figure 55 MISP URL and Credentials

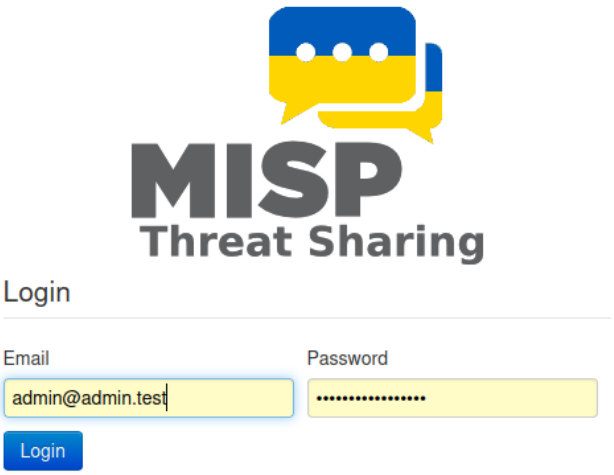

Figure 56 MISP Login Page

3.6.2 MISP Event Creation

After installing MISP, a private event was created to organize and manage threat intelligence relevant to the lab environment. The event was created to store indicators of compromise (IOCs) such as IP addresses, domain names, and other attributes, which were used for testing detection and correlation workflows. The event was given a descriptive title, and attributes were added, including external IP addresses, to provide meaningful data for evaluation and correlation within the lab environment (Figure 57 & 58). This setup enabled the evaluation of detection capabilities, the verification of alerts, and the differentiation between genuine threats and benign activity.

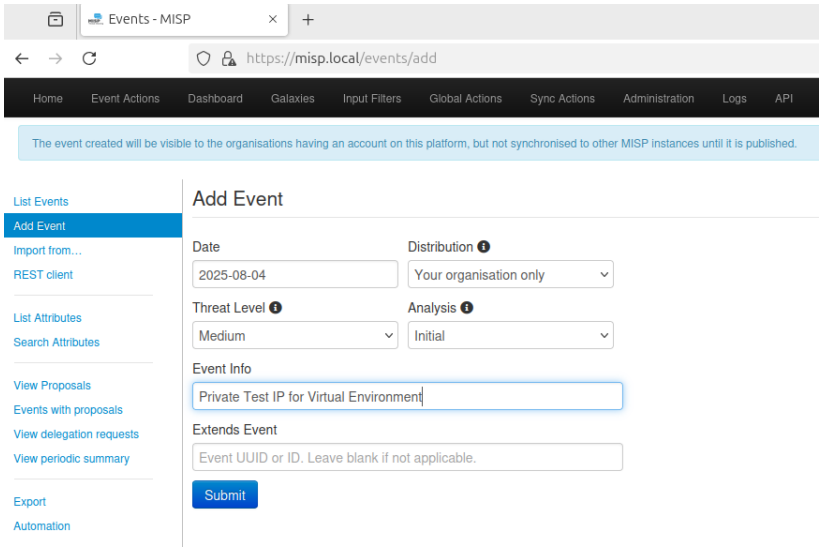

Figure 57 MISP Private Event Creation

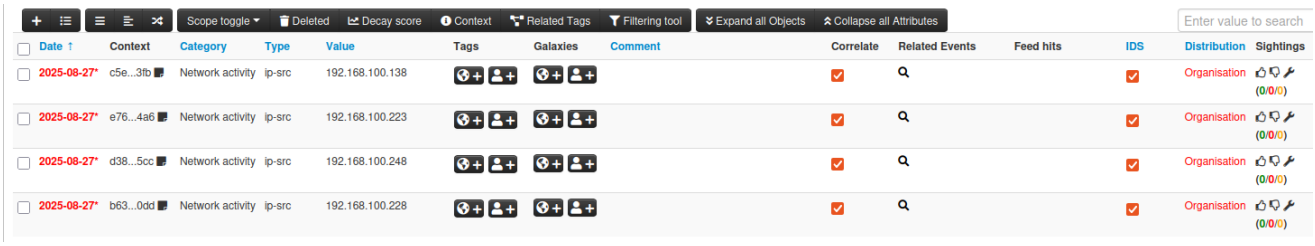

Figure 58 Adding Indicators of Compromise

3.7 MySQL Database Setup

MySQL was installed in MISP VM to provide a reliable and structured database for managing log data, user credentials, and configuration information essential for automation and event correlation. The installation was performed using the package manager with the sudo apt install mysql-server command, ensuring that a stable and secure version appropriate for the distribution is installed (Figure 59).

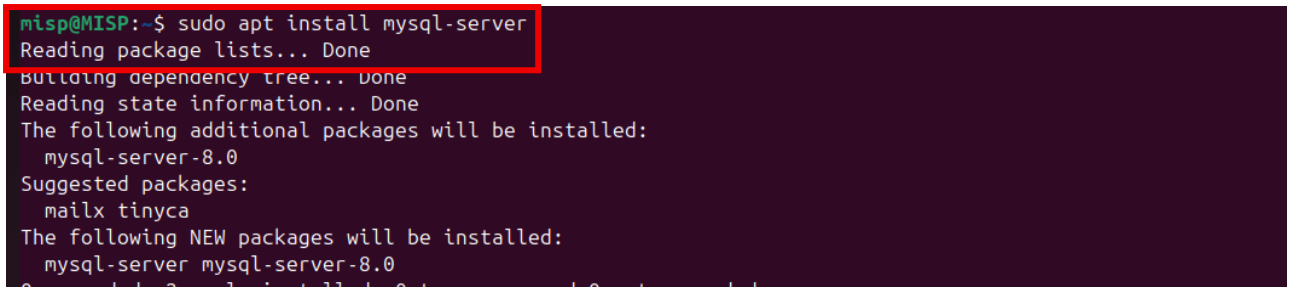

Figure 59 Installing MySQL Server

The MySQL root user password was first set to secure administrative access to the database server (Figure 60). A dedicated user account named **siemuser** was then created, along with a strong password, to handle day-to-day database interactions. This user was granted the necessary privileges to read from and write to the database while maintaining proper access control and minimizing exposure of root credentials (Figure 61). A dedicated database, **IDS_Feedback**, was created to store the log metadata, correlation rules, and feedback information used by the adaptive IDS logic (Figure 62). Firewall rules were updated to allow secure remote connections where required, ensuring that only authorized systems could communicate with the MySQL service (Figure 63).

```
misp@MISP:~$ sudo mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.42-0ubuntu0.24.04.2 (Ubuntu)

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'root1234';
Query OK, 0 rows affected (0.02 sec)
```

Figure 60 Setting root password

```
mysql>
mysql> CREATE USER 'siemuser'@'192.168.200.11' IDENTIFIED BY 'admin1234';
Query OK, 0 rows affected (0.02 sec)

mysql> GRANT ALL PRIVILEGES ON *.* TO 'siemuser'@'192.168.200.11' WITH GRANT OPTION;
Query OK, 0 rows affected (0.02 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.01 sec)
```

Figure 61 Creating user and granting privileges

```
mysql> create database IDS_Feedback;
Query OK, 1 row affected (0.02 sec)
```

Figure 62 Creating a database for Automation

```
misp@MISP:~$ sudo ufw allow from 192.168.200.11 to any port 3306
Rules updated
```

Figure 63 Configuring firewall rule

The database structure was established using MySQL Workbench, creating the necessary tables and schemas within the IDS_Feedback database. The tables are designed to store log data, feedback data, and correlation data, enabling smooth integration with Python automation scripts. This setup provided a reliable foundation for storing, processing, and correlating information, completing the backend infrastructure required for the adaptive feedback loop.

Chapter 4: System Implementation and Operation

4.1 Integration of Components

The integration phase focused on transforming individually configured tools into a unified monitoring environment. Each component was connected in a way that ensured seamless data flow, consistent formatting, and reliable communication across the system, as illustrated in Figure 64. Suricata serves as the intrusion detection system, continuously monitoring network traffic and host activity. It identifies suspicious behaviour or known attack patterns, generating alerts whenever potential threats are detected. These alerts form the foundation for further analysis.

To capture all relevant data, Filebeat and Winlogbeat collect logs from internal network, like Windows, SIEMS VM, MISP VM and Ubuntu Server VM, and forward them to Elasticsearch. Elasticsearch acts as the backbone of this setup. It takes the constant flow of raw security data, indexes it, and makes it easy to retrieve, filter, and correlate, even under heavy load. Built on this foundation, Kibana provides interactive dashboards and visualization tools, allowing analysts to monitor events in real time and quickly spot unusual patterns or anomalies.

Once the logs and alerts are ingested, a Python-based correlation engine processes them. The script enriches the alerts with threat intelligence from MISP (Malware Information Sharing Platform) and correlates multiple events to identify genuine attacks. Each event is then classified as a True Positive, False Positive, True Negative, or False Negative and then stored in a MySQL database along with supporting metadata. This ensures that the framework focuses on real threats while ignoring benign alerts, reducing false positives and improving response efficiency.

Finally, the dashboard is used to review alerts and predicted outcomes. Feedback from these reviews is then applied to refine the detection rules, gradually improving the system’s performance. This ongoing, iterative process allows the system to adapt to evolving threats, enhancing its accuracy and supporting timely, informed decisions for effective risk mitigation within the lab environment.

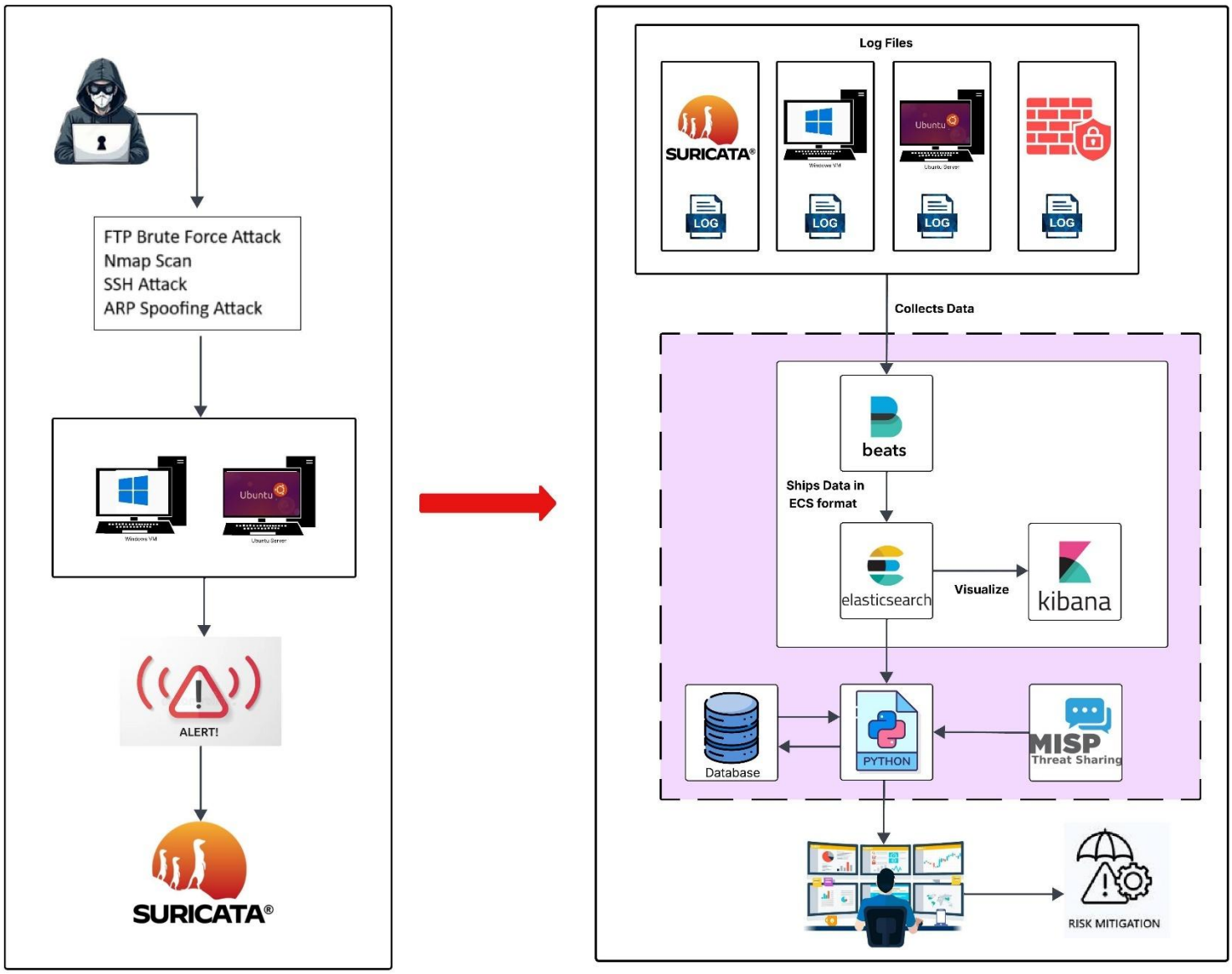

Figure 64 Integration of Components

4.2 Automation and Script Scheduling

This section describes how automation was implemented to streamline the alert correlation and classification workflow. Custom Python scripts were scheduled to run at defined intervals, ensuring continuous log retrieval, analysis, and feedback integration without manual intervention. The detailed scripts are provided in **Appendix B**.

4.2.1 Data Collection and Cleansing

The automated process begins with Python scripts that interact with Elasticsearch. A dedicated Elasticsearch client was configured using secure authentication, and logs were fetched incrementally based on the timestamp of the last successful fetch to prevent duplicates and capture new events in near real-time.

Once fetched, the logs were cleaned and standardized for consistency and easier analysis. Missing fields, such as source IPs in system or authentication logs, were extracted from the messages using patterns that identified valid IPs, while logs without sufficient reliable information were skipped, ensuring only accurate and usable data was stored in the database.

The message column contained the core details of each event but was stored in different formats like plain text, JSON, or lists, making it difficult to interpret directly. To address this, all messages were converted into clear, human-readable text summarizing the activity. Syslog messages were merged into single lines, such as service changes or blocked packets. Authentication messages were simplified to highlight the user, login type, source IP, and outcome, while FTP messages from vsftpd were condensed to show connection attempts, login results, and file transfers (Figure 65).

message	timestamp	process_name	log_file_path	host_ip	event_dataset	event_category	event_outcome
For sshd 78 authentication attempts, where 78 Failure and 0 Success were identified	2025-07-31 12:37:11	sshd	/var/log/auth.log	192.168.200.15	system.auth	authentication	failure
For vsftpd 15 authentication attempts, where 15 Failure and 0 Success were identified	2025-07-31 20:10:37	vsftpd	/var/log/auth.log	192.168.200.15	system.auth	authentication	failure
For vsftpd 55 authentication attempts, where 55 Failure and 0 Success were identified	2025-07-31 21:32:36	vsftpd	/var/log/auth.log	192.168.200.15	system.auth	authentication	failure
For sshd 1 authentication attempts, where 0 Failure and 1 Success were identified	2025-08-08 15:59:29	sshd	/var/log/auth.log	192.168.200.15	system.auth	authentication,se	success
For sshd 3 authentication attempts, where 0 Failure and 3 Success were identified	2025-08-08 16:03:26	sshd	/var/log/auth.log	192.168.200.15	system.auth	authentication,se	success

Figure 65 Converting text to Human readable format

Finally, all cleaned, normalized, and enriched logs were stored in structured MySQL tables. Separate tables were used for Suricata alerts, system events, authentication records, and FTP activities, making the data consistent, accessible, and ready for further analysis.

4.2.2 Log Correlation logic

After cleaning and formatting the logs, the next stage focused on identifying meaningful security events by correlating related data. This involved threat intelligence correlation through the Malware Information Sharing Platform (MISP). Source and destination IPs from the logs were compared against MISP indicators of compromise (IOCs), and matching logs were flagged and enriched with metadata such as event ID, category, type, and description (Figure 66). This integration allowed potentially harmful activity to be proactively identified and contextualized for incident response.

Matched IP	Event ID	Attribute ID	Category	Type	Value	Comment	Event Info	Event Link
192.168.100.248	1	4	Network activity	ip-src	192.168.100.248	Malicious and suspicious network activities	Private Test IP for Virtual Environment	https://misp.local//events/view/1
192.168.100.138	1	1	Network activity	ip-src	192.168.100.138	Malicious and suspicious network activities	Private Test IP for Virtual Environment	https://misp.local//events/view/1
192.168.100.228	1	3	Network activity	ip-src	192.168.100.228	Malicious and suspicious network activities	Private Test IP for Virtual Environment	https://misp.local//events/view/1
192.168.100.223	1	2	Network activity	ip-src	192.168.100.223	Malicious and suspicious network activities	Private Test IP for Virtual Environment	https://misp.local//events/view/1

Figure 66 MISP corelation

Suricata alerts were grouped based on common attributes, such as source IP, destination IP, timestamp range, and signature ID. This grouping reduced noise by combining multiple similar alerts from the same incident into a single logical event, making analysis more efficient.

These consolidated Suricata alerts were then cross-checked with other log sources, including system logs, authentication records, and FTP activity. The correlation engine looked for matching or related events across all sources that happened around the same time and involved the same IP addresses, making sure that even events not detected by Suricata but present in other logs were included. Each event was also tracked to indicate in which sources it appeared, providing a clear view of its presence and making analysis and incident investigation more straightforward (Figure 67).

incident_id	timestamp	source_ip	destination_ip	in_suricata	in_syslog	in_vsftpd	in_authlog	has_ioc
INC01	2025-07-30 22:54:23	192.168.100.228	192.168.200.15	TRUE	FALSE	FALSE	FALSE	TRUE
INC02	2025-07-31 11:37:07	192.168.100.248	192.168.200.15	TRUE	FALSE	FALSE	FALSE	TRUE
INC03	2025-07-31 12:10:25	192.168.100.248	192.168.200.15	TRUE	FALSE	FALSE	FALSE	TRUE
INC04	2025-07-31 12:11:58	192.168.200.15	192.168.200.11	TRUE	TRUE	FALSE	FALSE	FALSE
INC05	2025-07-31 12:30:33	192.168.100.248	192.168.200.15	TRUE	FALSE	FALSE	TRUE	TRUE
INC06	2025-07-31 16:27:19	185.125.190.82	185.125.190.82	TRUE	FALSE	FALSE	FALSE	FALSE
INC07	2025-07-31 19:47:33	192.168.200.11	167.82.48.223	TRUE	FALSE	FALSE	FALSE	FALSE
INC08	2025-07-31 19:47:33	192.168.200.11	8.8.4.4	TRUE	FALSE	FALSE	FALSE	FALSE
INC09	2025-07-31 20:37:58	192.168.100.128	192.168.200.15	TRUE	FALSE	FALSE	FALSE	FALSE
INC10	2025-08-01 21:12:46	192.168.100.128	192.168.200.15	TRUE	FALSE	FALSE	TRUE	FALSE
INC11	2025-08-02 14:38:26	192.168.100.138	192.168.200.15	TRUE	FALSE	FALSE	FALSE	TRUE
INC12	2025-08-03 23:29:02	192.168.100.138	192.168.200.15	TRUE	FALSE	FALSE	FALSE	TRUE
INC13	2025-08-05 12:30:48	192.168.200.138	192.168.200.15	TRUE	FALSE	FALSE	FALSE	TRUE
INC14	2025-08-08 00:59:54	192.168.200.100	192.168.200.15	TRUE	FALSE	FALSE	FALSE	FALSE
INC15	2025-08-08 18:58:23	192.168.200.12	192.168.200.15	TRUE	FALSE	TRUE	FALSE	FALSE
INC16	2025-07-31 11:23:55	0.0.0.0	192.168.200.15	FALSE	FALSE	FALSE	TRUE	FALSE
INC17	2025-07-31 13:10:27	192.168.100.248	192.168.200.15	FALSE	FALSE	FALSE	TRUE	TRUE
INC18	2025-07-31 13:30:36	192.168.100.248	192.168.200.15	FALSE	FALSE	FALSE	TRUE	TRUE
INC19	2025-07-31 20:09:20	0.0.0.0	192.168.200.12	FALSE	FALSE	FALSE	TRUE	FALSE
INC20	2025-07-31 20:10:37	192.168.100.128	192.168.200.15	FALSE	FALSE	FALSE	TRUE	FALSE
INC21	2025-08-01 22:12:43	192.168.100.128	192.168.200.15	FALSE	FALSE	FALSE	TRUE	FALSE

Figure 67 Log corelation Outcome

4.2.3 Classification of Alerts: TP, TN, FP, FN

After correlating events from multiple sources, the next step was to classify alerts as True Positives (TP), True Negatives (TN), False Positives (FP), or False Negatives (FN) which was implemented through a structured decision-making process, as illustrated in the Figure 68. The events are assessed across multiple data sources, where each event is first checked for detection by Suricata and for the presence of known Indicators of Compromise (IOCs). Historical analyst feedback was then applied where available to guide the classification of recurring or similar events. In addition, system logs, authentication logs, and FTP activity were analysed to identify suspicious behaviour that might not be evident from alerts alone. Based on this multi-source evaluation, each event was classified as a True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN).

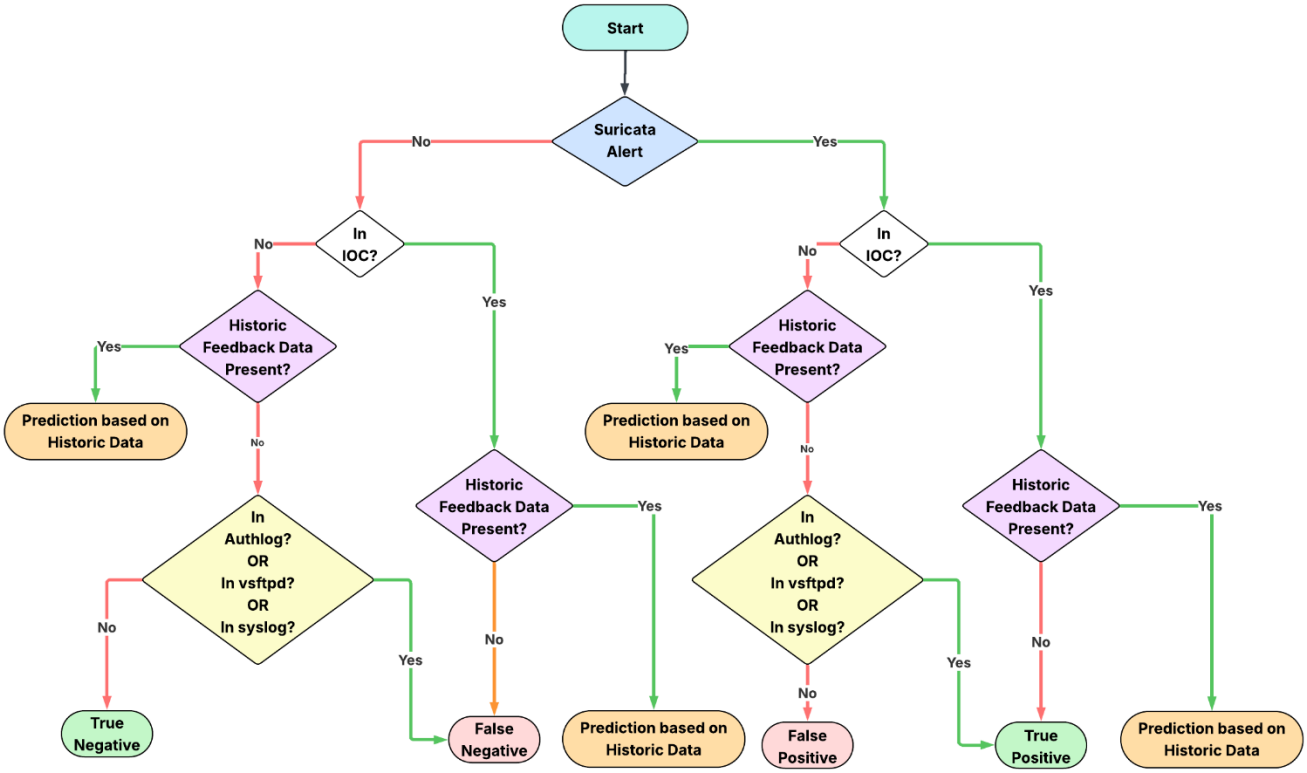

Figure 68 Prediction Flow Diagram

To refine the classification results, each event undergoes deeper verification using data from FTP, authentication, and syslog sources. FTP logs are checked for signs such as repeated failed login attempts, successful logins from unusual locations, and abnormal file transfer sizes. Authentication records are reviewed to identify patterns of excessive failed logins or unexpected successful accesses. Syslog entries are analysed for network events like BLOCK, DROP, and ACCEPT, paying close attention to sudden spikes in blocked or dropped traffic tied to the same IP address. Together, these additional checks help confirm whether an alert reflects a genuine incident (True Positive) or a False Positive as seen in Figure 69, and in cases where Suricata does not generate an alert, they assist in identifying missed detections (False Negatives) or verifying normal activity (True Negatives) as seen in Figure 70.

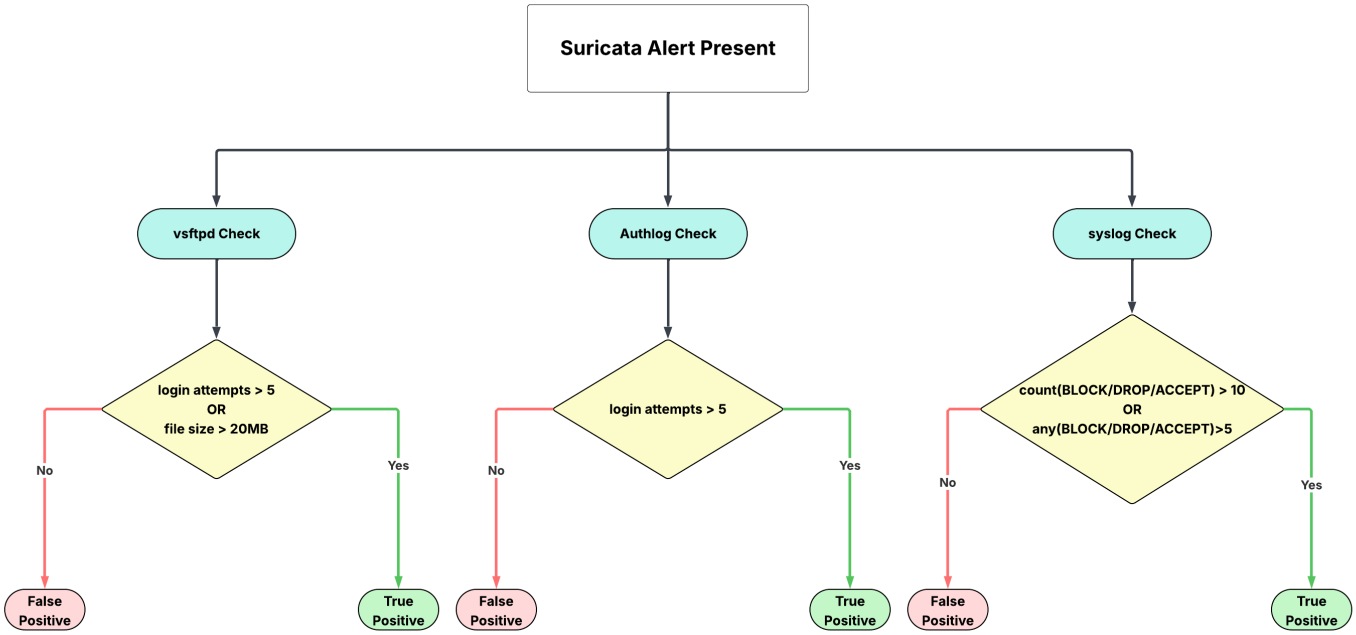

Figure 69 Log checking condition (Suricata Alert)

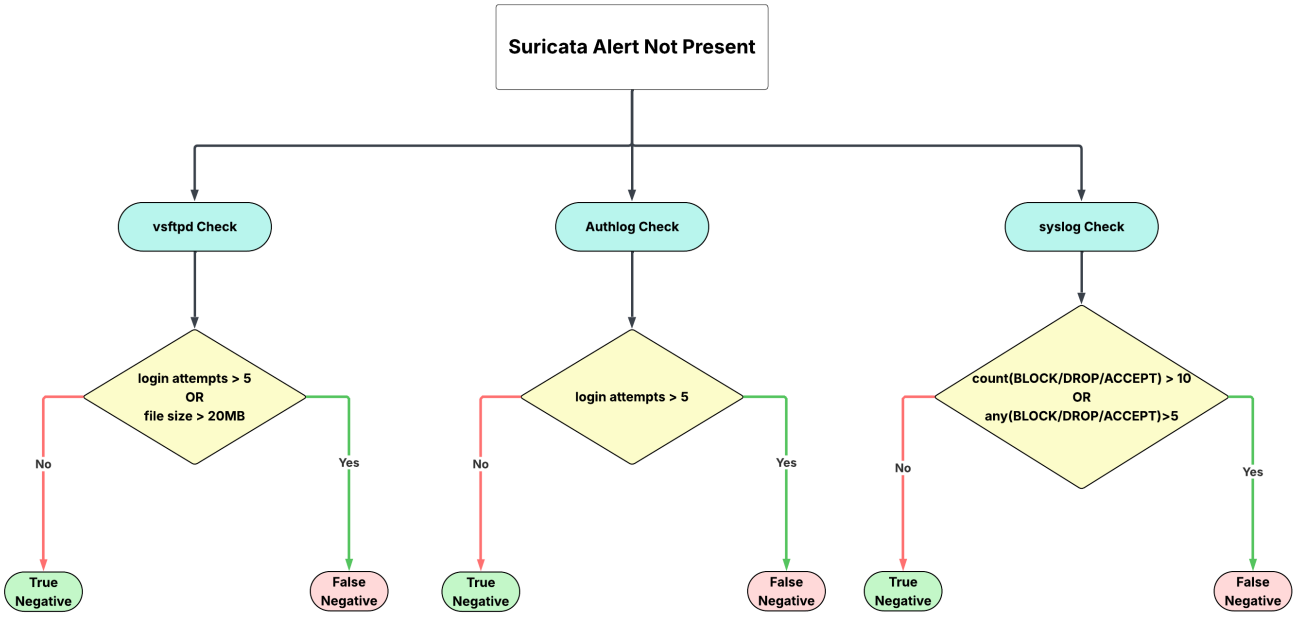

Figure 70 Log checking condition (No Suricata Alert)

4.2.4 Script Scheduling and Execution

To streamline the classification workflow, a Python script was set up to run automatically through the system scheduler, removing the need for manual execution of each analysis step. The process begins with fetching the latest logs from Elasticsearch, and once this retrieval is successful, the script proceeds to perform correlation and classification. If the log retrieval fails for any reason, the correlation step is skipped to prevent working with incomplete data, ensuring the results remain accurate. To keep the workflow running consistently, the script is scheduled as a cron job to execute every five minutes (Figure 71). All output generated by the script, including both standard messages and errors, was redirected to a dedicated log file, allowing easy monitoring of each run and quick troubleshooting in case of failures. This setup ensures that logs are continuously collected, analysed, and classified, keeping the alert state up to date and supporting timely detection and response to any incidents.

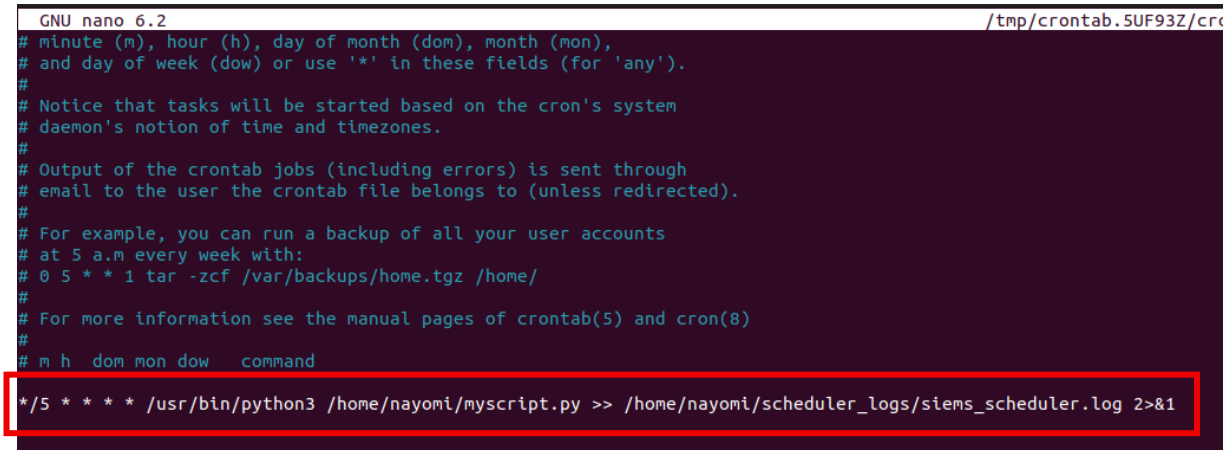

Figure 71 Cron job for Automation

4.3 User Interface

The user interface (UI) was implemented using Python and Streamlit, providing an interactive platform for monitoring and managing alerts. It was launched in sudo mode, which not only ensured the application had the necessary permissions to access protected system resources such as Suricata rules and firewall configurations, but also generated the URL through which the UI could be accessed via a web browser as seen in Figure 72.

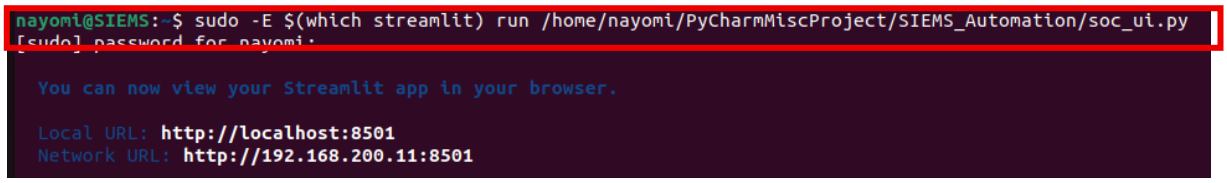

Figure 72 Starting Feedback Dashboard

The feedback module displays alerts categorized as True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) to provide insight into detection accuracy. Alerts are prioritized by prediction confidence, ensuring that the most critical ones are addressed first, and then ordered by timestamp so the most recent events appear at the top as seen in Figure 73. This approach ensures that critical alerts are checked promptly, improving response efficiency.

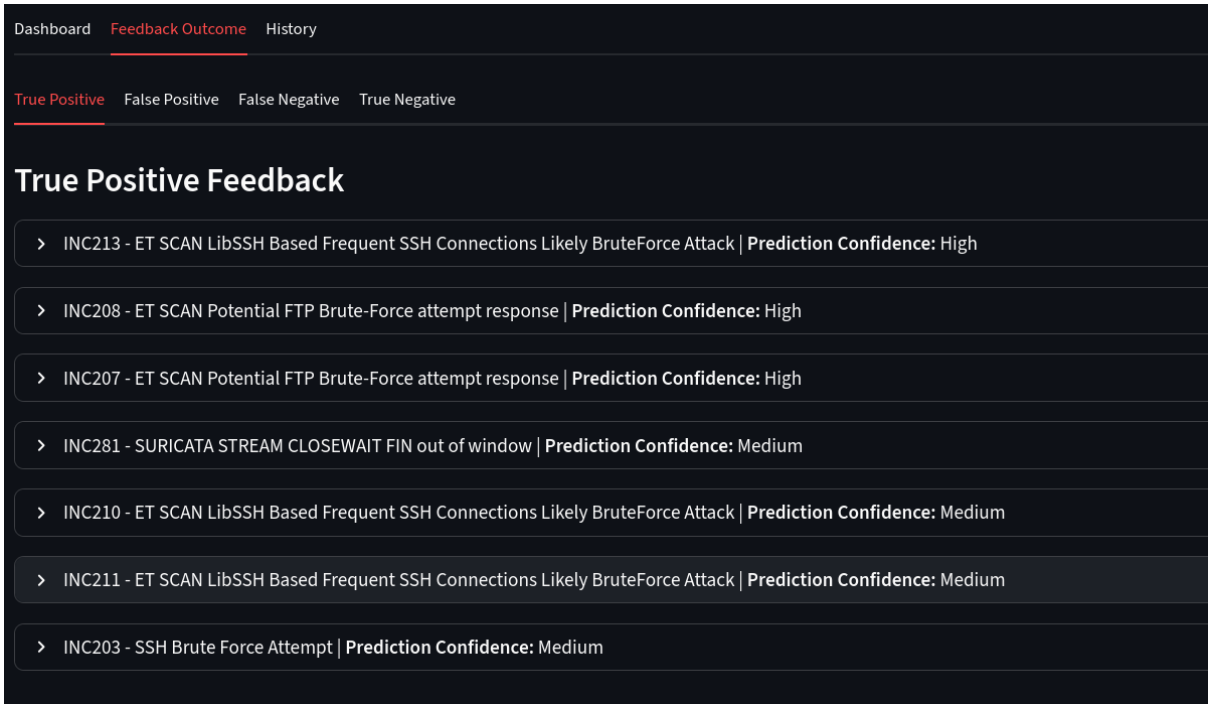

Figure 73 Feedback outcome View

Once an incident is selected, the system displays all relevant logs to provide comprehensive context for analysis. This includes Suricata alerts with details such as timestamp, source IP, destination IP, and event type, alongside logs from Syslog, Auth log, and FTP sources. Correlated threat intelligence from MISP is also presented, offering analysts additional insight into potential threats (Figure 74 & 75). All displayed data can be cross verified in the Kibana dashboard, allowing analysts to validate incidents, confirm patterns, and ensure accurate classification. This integrated view supports thorough investigation, timely response, and improved decision-making.

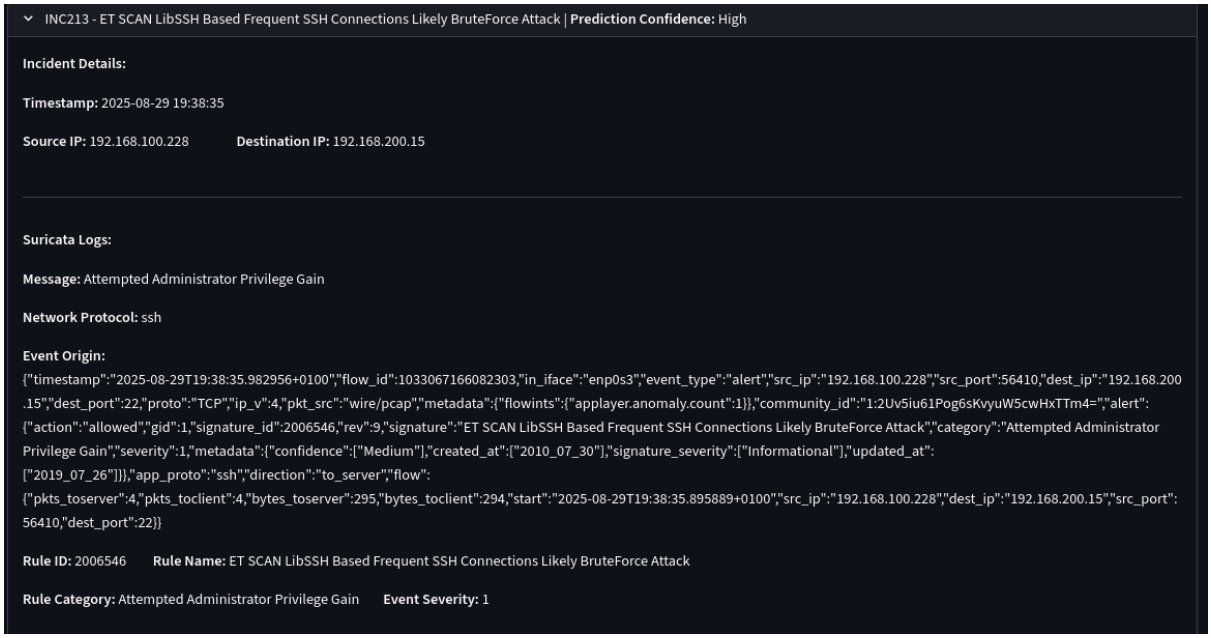

Figure 74 Incident details view

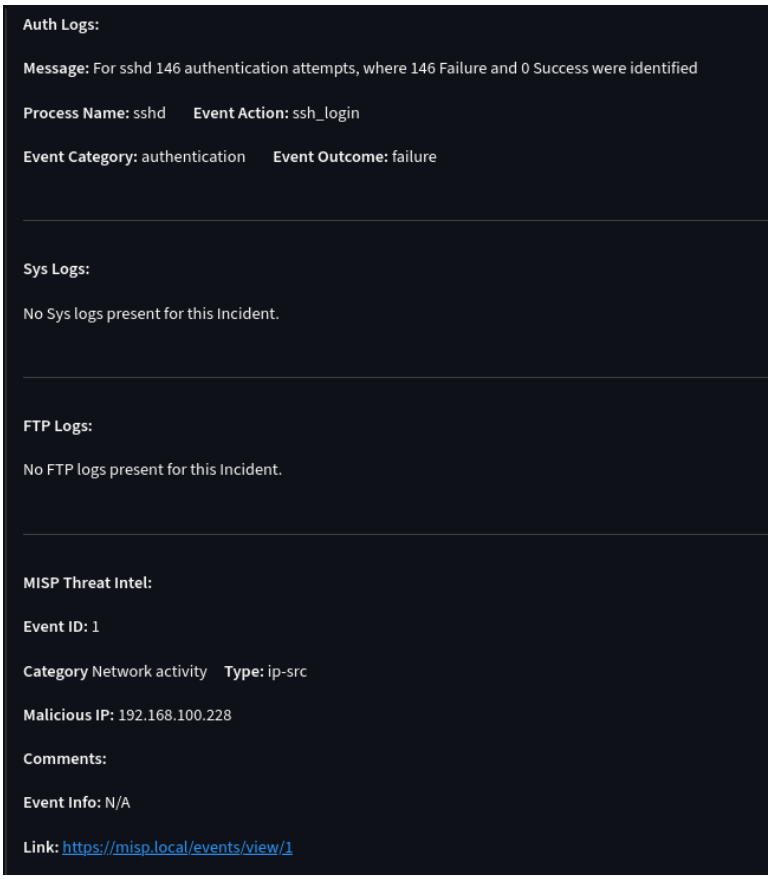

Figure 75 Incident details view (Syslog, FTP, MISP)

Based on the selected logs and any available historical data, as described in Section 6.2.3, the system predicts the alert outcome and provides the reasoning behind its classification. The predictions can then be reviewed, and input can be provided to indicate whether the alert is a alert is a True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN). Comments can also be added to justify the assessment, allowing the system to incorporate feedback and refine future prediction accuracy as seen in Figure 76.

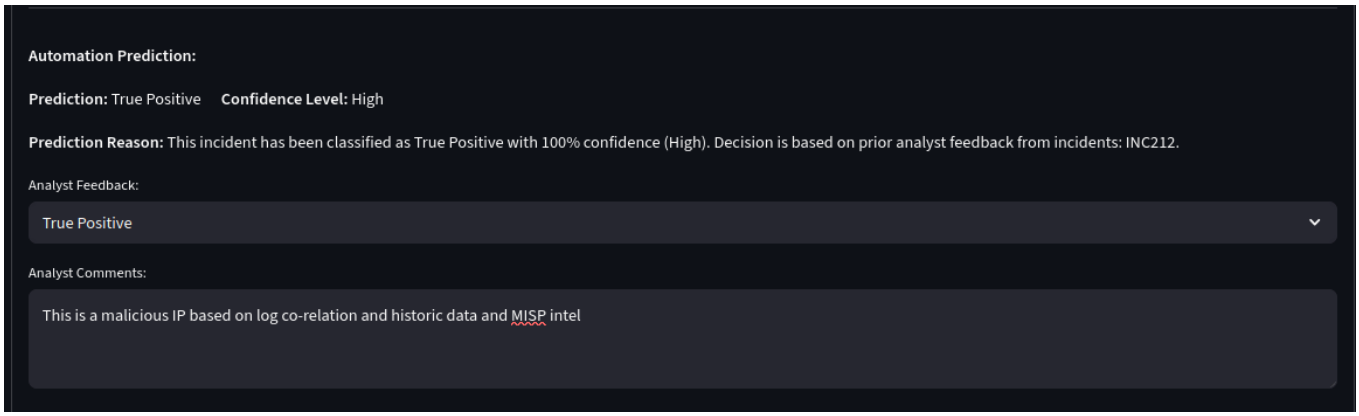

Figure 76 Prediction and Analyst Feedback

The interface was designed to allow adding a Suricata suppression rule, a firewall rule, or both. Firewall rules include actions such as accept, drop, or reject to control network traffic (Figure 77), while Suricata displays the specific rule that triggered the alert and provides an option to add suppression rules to prevent similar alerts in the future (Figure 78). After selecting the appropriate rules, a submit button is used to apply the changes. Upon execution, the system updates in real time, ensuring the chosen firewall or Suricata rules take immediate effect, while simultaneously recording the modifications in the backend database for logging and future reference.

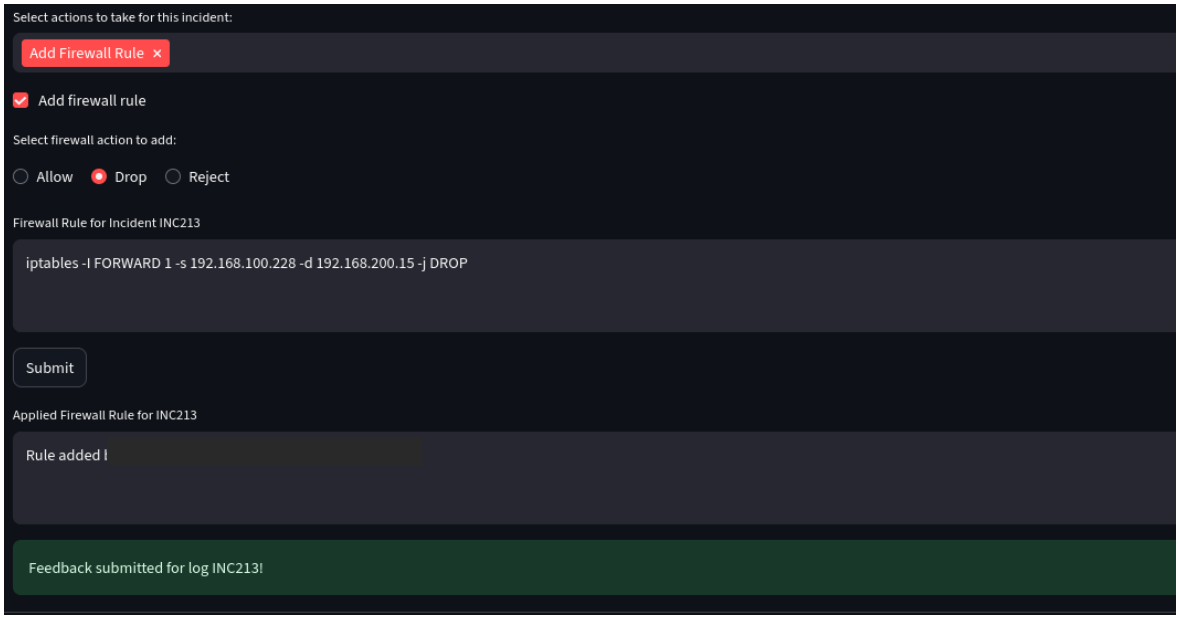

Figure 77 Mitigation Action - Adding Firewall rule

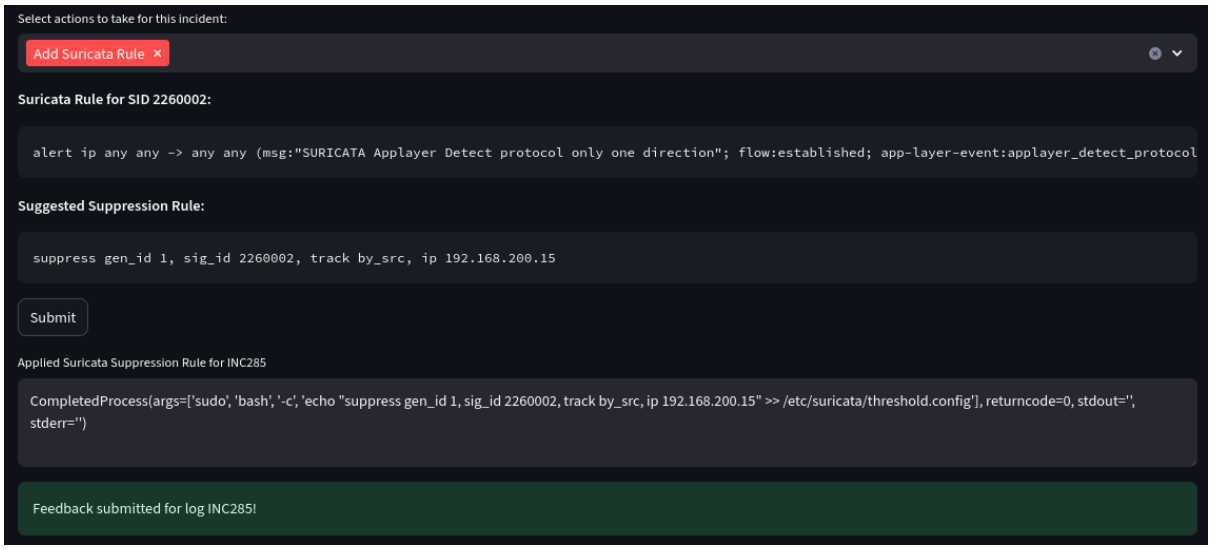

Figure 78 Mitigation Action- Suricata suppression Rule

The system includes a History section that displays all relevant historical data for reviewed alerts (Figure 79). This feature allows past alert records, including classification outcomes (TP, TN, FP, FN), associated logs, and any applied rules, to be easily accessed and reviewed. The History section enables tracking of trends, identification of recurring patterns, and assessment of previous mitigation actions. It also supports the validation of current predictions by referencing past incidents, ensuring more informed decision-making and improving the overall accuracy and efficiency of alert management.

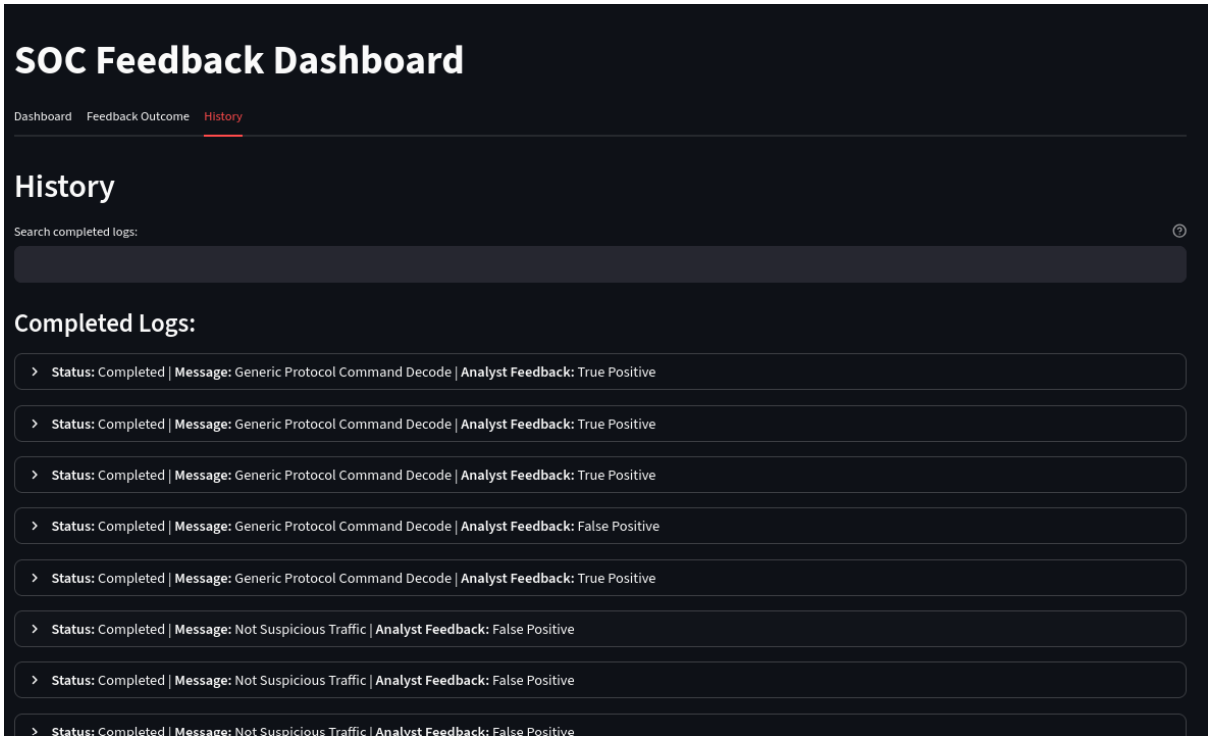

Figure 79 Historic Data

4.4 Performance Metrics

To evaluate the performance of the automation, its predictions were measured against the reference ground truth, representing the correct classification of each event. This comparison helps identify where the automation succeeds and where it makes errors, providing a clear measure of its effectiveness. Each prediction falls into one of four categories:

i. True Positive (TP) / True Negative (TN): The automation prediction matches the ground truth label.
ii. False Positive (FP) / False Negative (FN): The automation prediction does not match the ground truth label.

These categories are used to calculate key evaluation metrics in AI/ML:

1. **Precision:** The portion of correctly predicted positive instances out of all instances predicted as positive. It measures the reliability of positive predictions.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

2. **Recall:** Indicates the proportion of actual positive cases correctly identified by the model.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

3. **F1-score:** The mean of precision and recall, providing a balance between the two.

$$F1 - measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

4. **Accuracy:** The proportion of all correctly classified instances (both positive and negative) out of the total number of predictions.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

These metrics were calculated using scikit-learn, a widely used Python library for AI/ML, which provides functions for evaluating classification models. In this context, TP, TN, FP, and FN are simply measures of how well the automation agrees with the ground truth. They are not related to cybersecurity events or alerts but instead reflect the model's ability to predict outcomes accurately.

By evaluating performance in this structured manner, it becomes possible to quantify the strengths and weaknesses of the automation system, identify areas that may require improvement, and establish a transparent, reproducible method for assessing and reporting model performance.

Chapter 5: Testing and Evaluation

5.1 Test Plan and Objectives

The testing phase was designed around realistic attack scenarios and system behaviour under different conditions. The aim was to confirm that the entire alert classification and management workflow functioned as expected.

The test plan focused on the following objectives:

- **Attack scenario coverage:** Different types of attacks were generated to evaluate whether the system detected them and provided relevant alerts.
- **Prediction accuracy:** Each detected alert was checked to ensure that the prediction engine classified it correctly and provided a clear reasoning for its decision.
- **Feedback handling:** TP, TN, FP, and FN feedback, along with any associated comments, were verified to be stored correctly in the database and reflected in subsequent predictions.
- **Rule update verification:** Firewall and Suricata suppression rules were tested to confirm that new rules were applied in real time and recorded without errors.
- **UI usability and stability:** The interface was evaluated to ensure that it remained responsive, intuitive, and stable during normal use, under load, and when interacting with multiple alerts in succession.

These tests were run using a controlled environment where simulated network traffic, system logs, and manually injected events represented different attack types and operational scenarios.

5.2 Test Cases

The system was evaluated using multiple realistic network scenarios to generate True Positive (TP), False Positive (FP), and False Negative (FN) alerts. These scenarios were designed to test prediction accuracy, feedback handling, rule application, and system responsiveness under a variety of operational conditions. Each scenario was executed carefully to verify the system's classification accuracy and its ability to respond effectively to different types of network activity. TP scenarios represented confirmed malicious activity, such as brute force attacks, reconnaissance scans, and attempts to download malicious content, validating the system's ability to detect real threats. FN scenarios tested the system's capability to identify subtle or slow-moving attacks that could evade detection, such as low-rate brute force attempts or encrypted payloads. FP scenarios evaluated how the system handled benign or high-volume activities that could mistakenly trigger alerts, such as large file transfers, multiple concurrent connections, or routine DNS lookups. This comprehensive testing approach ensured that prediction, feedback incorporation, rule enforcement, and system stability were thoroughly assessed under conditions closely resembling real-world network environments, providing confidence in the overall robustness and reliability of the implemented solution.

Activity	Source → Target	Description	Source IP Address
SSH brute force	Kali → Ubuntu	Repeated login attempts on SSH to test brute force detection	192.168.100.228
			192.168.100.150
FTP brute force	Kali → Ubuntu	Repeated login attempts on FTP to trigger brute force alerts	192.168.100.248
			192.168.100.138
Nmap scan	Kali → Ubuntu	TCP SYN scan on ports 21 and 22 to simulate reconnaissance	192.168.100.128
Malicious download	Windows → Ubuntu	Access to a known malicious URL to trigger a download-related alert	192.168.200.12

Table 1 True Positive Scenarios

Activity	Source → Target	Description	Source IP Address
Slow SSH brute force	Kali → Ubuntu	Brute force attack at very low rate (1 attempt per minute) to test evasion	192.168.100.223
			192.168.100.233
Encrypted SSH payload	Kali → Ubuntu	Malicious SSH packets with encryption to test detection evasion	192.168.100.100

Table 2 False Negative Scenarios

Activity	Source → Target	Description	Source IP Address
Large FTP transfer	Windows → Ubuntu	Upload of a large file (>1KB) to FTP server	192.168.200.12
Multiple SSH sessions	Windows → Ubuntu	Multiple concurrent SSH sessions to simulate heavy but benign activity	192.168.200.12
Continuous SSH data transfer	Windows → Ubuntu	Large file transfer via SCP	192.168.200.12
ICMP flood	Windows → Ubuntu	Continuous ping to simulate network stress without malicious intent	192.168.200.12
Multiple SSH login failures	Windows → Ubuntu	Repeated failed login attempts that are legitimate testing attempts	192.168.200.12
File transfer via SCP	Windows → Ubuntu	Transfer of a large file from Windows to Ubuntu for testing	192.168.200.12
DNS lookup	Windows → Ubuntu	Query for a non-existent domain to check alert response	192.168.200.12

Table 3 False Positive Scenarios

5.3 System Observations and Limitations

During the testing phase, the system demonstrated robust performance across multiple scenarios. Alerts were successfully detected and classified for True Positive, False Positive, and False Negative cases, indicating that the prediction engine and alert correlation functioned as expected. Feedback input, including TP, TN, FP, and FN selections along with associated comments, was accurately recorded in the database and reflected in subsequent predictions. Rule updates for both Suricata suppression and firewall rules were applied in real time and consistently logged in the backend (Figure 80 and 81). The user interface remained stable and responsive, even when handling multiple alerts simultaneously, and integration between prediction, feedback, rule enforcement, and database modules was seamless. Testing was conducted in a controlled lab environment with a limited network scale, and the virtual machine used had constrained resources, making it challenging to process large volumes of log data efficiently. Extreme network loads were not fully stress tested.

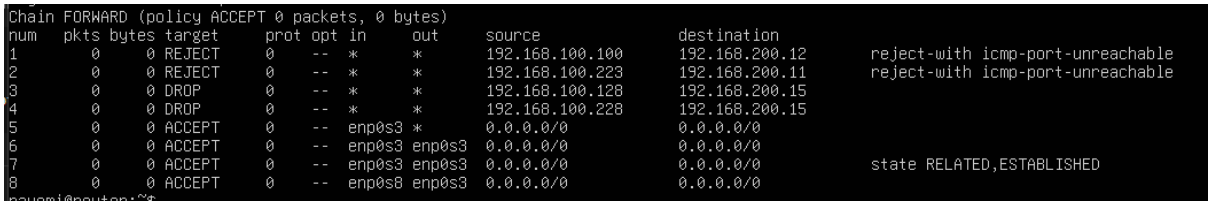

Figure 80 Firewall rule updated

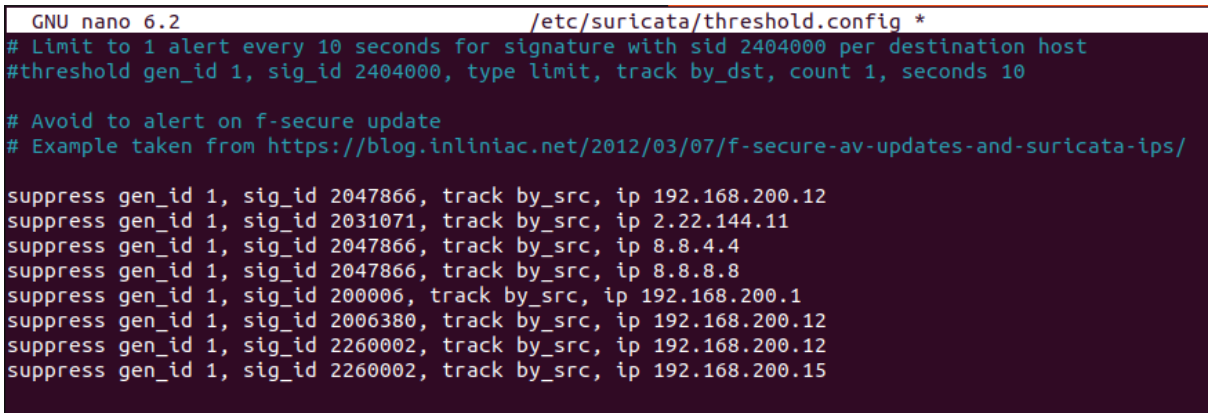

Figure 81 Suricata rule updated

Chapter 6: Results and Analysis

6.1 Evaluation of Rule Effectiveness on Alert Reduction

The results of the evaluation are illustrated in Figure 82, which tracks the distribution of True Positives (TPs), False Positives (FPs), and False Negatives (FNs) across time. The objective of this evaluation was to assess whether the applied rule modifications improved detection accuracy while reducing unnecessary alerts.

The total number of alerts steadily decreased as the system continuously applied Suricata suppression and firewall rules. True positives also dropped slightly, showing that duplicate alerts were being filtered out while new or ongoing threats continued to be detected. At the same time, false positives fell significantly, reflecting improved handling of benign activity, and false negatives remained very low, meaning important malicious events were rarely missed. The initial sharp decline in alerts demonstrates how quickly recurring threats were mitigated, while the later stabilization shows the system's ability to adapt to ongoing network activity. Overall, these results highlight that real-time rule enforcement, combined with historical feedback, effectively reduces unnecessary alerts and keeps operations running efficiently.

Figure 82 Analysis of Alert

6.2 Prediction Accuracy

The performance of the Automation in detecting events was evaluated by comparing its predictions against the ground truth provided by the analyst. Metrics were calculated separately for each category to understand how well the Automation performs in identifying specific types of events. Across all tasks, the system achieved an overall prediction accuracy of 92.76%, indicating a high level of agreement with the analyst.

This means that, in most cases, the automation correctly identified events and non-events, effectively supporting the analyst in distinguishing between real threats and benign activity. The strong overall accuracy reflects that the system is reliable and consistent

In practice, this level of accuracy helps reduce the analyst's workload by minimizing false alerts while ensuring that most critical events are correctly detected.

```
False Negative
Precision: 60.00%
Recall: 75.00%
F1: 66.67%
Number of test examples: 8

True Negative
Precision: 82.35%
Recall: 82.35%
F1: 82.35%
Number of test examples: 34

True Positive
Precision: 97.78%
Recall: 84.62%
F1: 90.72%
Number of test examples: 52

False Positive
Precision: 96.21%
Recall: 100.00%
F1: 98.07%
Number of test examples: 127

Overall Prediction Accuracy: 92.76%
```

Figure 83 System Prediction Accuracy

Chapter 7: Conclusion and Future Work

7.1 Summary of Key Finding

This project explored the development of an automated alert classification and management system for a Security Operations Center (SOC), combining Suricata intrusion detection, system and application logs, and MISP threat intelligence. The work began with understanding how alerts could be effectively classified and prioritized, ensuring that critical threats were detected while reducing unnecessary noise.

The framework was designed with both automation and human oversight in mind. Python scripts processed historical data to predict alert outcomes, while feedback mechanisms allowed the system to learn from past decisions and adapt to evolving network activity. Real-time rule enforcement through Suricata suppression rules and firewall policies provided an additional layer of control, filtering repeated alerts and allowing analysts to focus on actionable incidents.

Testing under realistic network conditions confirmed that the system could handle high volumes of alerts without compromising stability. Alerts were accurately classified, redundant notifications were minimized, and situational awareness was strengthened. With a prediction accuracy of **92.76%**, the framework demonstrated its effectiveness in balancing automation, responsiveness, and operational reliability.

Through this project, it became clear that integrating alert classification, rule management, and historical feedback creates a more resilient and efficient SOC environment. The system not only streamlines operations but also empowers decision-making, ensuring that security teams can respond quickly and confidently to emerging threats, ultimately contributing to a safer and more controlled digital landscape.

7.2 Future Work

The current framework successfully reduces false positives and improves alert handling using a rule-based correlation engine, but there are several ways it could be strengthened. A logical next step is to integrate statistical or machine-learning techniques alongside the existing rules. Rather than replacing the rule-based system, these methods could help identify subtle patterns or unusual activity that static rules might miss, improving detection and allowing the system to adapt to new threats.

Another promising direction is to expand log collection to include IoT devices and cloud-based systems, which are increasingly part of modern networks. Incorporating these additional sources would provide a more complete view of network activity, helping detect threats that span traditional endpoints, cloud services, and connected devices.

Careful implementation would preserve transparency, so that alerts remain understandable while detection capabilities evolve. Over time, combining rule-based logic with adaptive techniques could make the framework more resilient and effective, even as networks grow more complex and distributed.

References

Aktar, M.N., Akhand, M.A.H., Uddin, M.Z., Siddique, A.Y. and Islam, S.M.R., 2024. Enhancing false positive alert detection in security information and event management system using recurrent neural network. In: *2024 27th International Conference on Computer and Information Technology (ICCIT)*, Cox's Bazar, Bangladesh, 2024, pp. 2794-2799. doi: 10.1109/ICCIT64611.2024.11022066. Available at: https://ieeexplore.ieee.org/document/11022066

docs.suricata.io. (n.d.). *2. Quickstart guide — Suricata 8.0.0-dev documentation*. [online] Available at: https://docs.suricata.io/en/latest/quickstart.html

Elastic.co. (2025). *Download Winlogbeat | Ship Windows Event Logs | Elastic*. [online] Available at: https://www.elastic.co/downloads/beats/winlogbeat.

Elastic.co. (2025). *Download Filebeat • Lightweight Log Analysis*. [online] Available at: https://www.elastic.co/downloads/beats/filebeat.

Fuentes, J., Ortega-Fernandez, I., Villanueva, N.M. and Sestelo, M. (2025). *Cybersecurity threat detection based on a UEBA framework using Deep Autoencoders*. [online] arXiv.org. Available at: https://arxiv.org/abs/2505.11542

Ghubaish, A., Yang, Z. and Jain, R. (2024). HDRL-IDS: A Hybrid Deep Reinforcement Learning Intrusion Detection System for Enhancing the Security of Medical Applications in 5G Networks. [online] pp.1–6. doi:https://doi.org/10.1109/smartnets61466.2024.10577692.

Hussain, S., He, J., Zhu, N., Mughal, F.R., Hussain, M.I., Algarni, A.D., Ahmad, S., Zarie, M.M. and Ateya, A.A. (2024). An Adaptive Intrusion Detection System for WSN using Reinforcement Learning and Deep Classification. *Arabian Journal for Science and Engineering*. doi:https://doi.org/10.1007/s13369-024-09769-x

Ishaque, M., Johar, M., Khatibi, A. and Yamin, M. (2024). Dynamic Adaptive Intrusion Detection System Using Hybrid Reinforcement Learning. *Lecture notes in networks and systems*, pp.245–253. doi:https://doi.org/10.1007/978-3-031-55911-2_23

Iyer, K.I. (2024). AI and Human Analysts: The Ultimate Synergy for Threat Defense. *International Journal of Innovative Research in Computer and Communication Engineering*, 12(04). doi:https://doi.org/10.15680/ijircce.2024.1204372

Kataria, A. (2023). An ML-Based Intrusion Detection System Design and Evaluation for Enhanced Cybersecurity. pp.1036–1040. doi:https://doi.org/10.1109/iccsai59793.2023.10421690

Kikissagbe, B.R. and Adda, M. (2024). Machine Learning-Based Intrusion Detection Methods in IoT Systems: A Comprehensive Review. *Electronics*, 13(18), p.3601. doi:https://doi.org/10.3390/electronics13183601

Sashwin, K.S., N.K., Priyadharshini, S.P., M.S. and Saranya, 2024. *Analysis, trends, and utilization of Security Information and Event Management (SIEM) in critical infrastructures*. In: 2022 8th International Conference on Advanced Computing and Communication Systems

(ICACCS). [online] IEEE, pp.1980–1984. Available at: https://doi.org/10.1109/icaccs60874.2024.10717237

Sheeraz, M. *et al.* (2024) 'Revolutionizing SIEM Security: an innovative correlation engine design for Multi-Layered attack detection,' *Sensors*, 24(15), p. 4901. https://doi.org/10.3390/s24154901.

Shetty, P. (2024) 'AI and Security, From an Information Security and Risk Manager Standpoint', *IEEE Access*, 12, pp. 77468–77474. doi: 10.1109/ACCESS.2024.3408144. Available at: https://ieeexplore.ieee.org/document/10542982

Siddiqui, M.A., Stokes, J.W., Seifert, C., Argyle, E., McCann, R., Neil, J. and Carroll, J. (2019). *Detecting Cyber Attacks Using Anomaly Detection with Explanations and Expert Feedback*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICASSP.2019.8683212.

Soltani, M., Khashayar Khajavi, Mahdi Jafari Siavoshani and Amir Hossein Jahangir (2024). A multi-agent adaptive deep learning framework for online intrusion detection. *Cybersecurity*, 7(1). doi:https://doi.org/10.1186/s42400-023-00199-0.

Tariq, S., Chhetri, M. B., Nepal, S., & Paris, C. (2025). Alert Fatigue in Security Operations Centres: Research Challenges and Opportunities. *ACM Computing Surveys*, 57(9), Article 224, 1–38. https://doi.org/10.1145/3723158

Appendix A : Sample Logs

Suricata logs

Suricata acts as a network intrusion detection and prevention system. It monitors traffic in real time and creates detailed logs whenever it sees network activity or detects something suspicious. These logs are written in JSON format, with each entry describing a single network event. They include details such as the time of the event, the source and destination IP addresses, port numbers, the network protocol, the type of alert, and the signature that triggered it. By default, Suricata saves the logs in the **/var/log/suricata/** directory. The two primary log files are **fast.log**, which contains concise, human-readable alerts, and **eve.json**, which provides detailed JSON events that can be easily parsed and analysed. These logs serve as the primary information for detecting and investigating suspicious or malicious network activity.

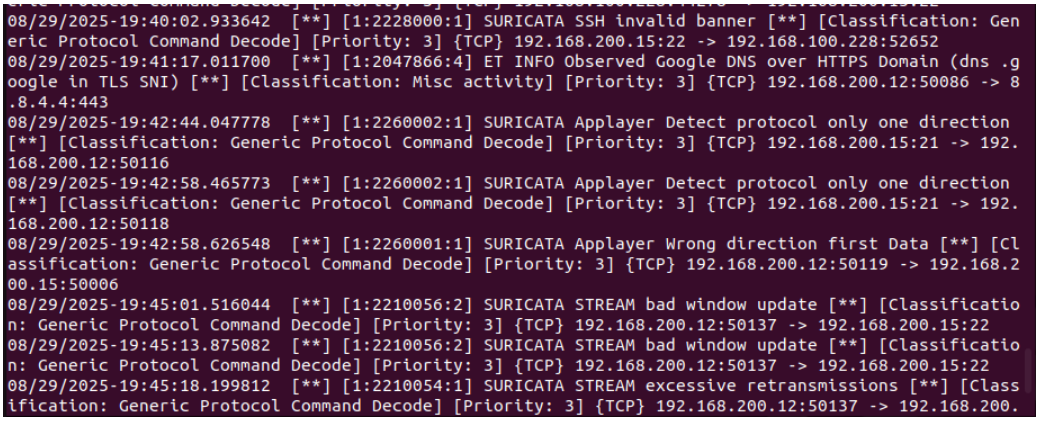

Figure 84 Examples of Suricata fast.log

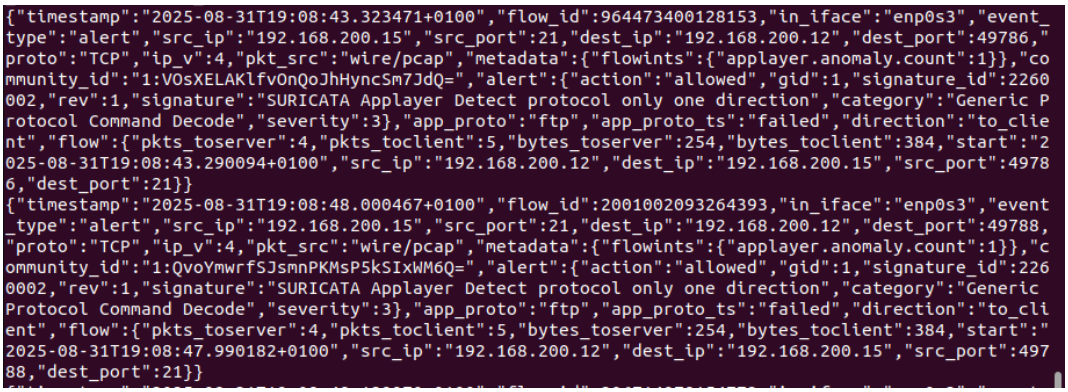

Figure 85 Examples of Suricata eve.json

System Logs

System logs capture the operational state of the systems like MISP and Ubuntu server VMs. It captures kernel messages, service status changes, hardware events, and software errors. They are stored in plain text using the traditional syslog format, which contains timestamp, hostname, process name, process identifier, and the message content. All system activity and events are saved in the file **/var/log/syslog**. This baseline information is essential for monitoring host stability and provides supporting evidence during incident triage when correlating alerts with underlying system behaviour.

```
Sep  4 21:22:37 victim systemd[1]: Starting vsftpd FTP server...
Sep  4 21:22:37 victim systemd[1]: Started LSB: automatic crash report generation.
Sep  4 21:22:37 victim systemd[1]: Finished Save/Restore Sound Card State.
Sep  4 21:22:37 victim systemd[1]: grub-common.service: Deactivated successfully.
Sep  4 21:22:37 victim systemd[1]: Finished Record successful boot for GRUB.
Sep  4 21:22:37 victim systemd[1]: Finished OpenVPN service.
Sep  4 21:22:37 victim systemd[1]: Finished Permit User Sessions.
Sep  4 21:22:37 victim systemd[1]: Started vsftpd FTP server.
Sep  4 21:22:37 victim ModemManager[675]: <info>  ModemManager (version 1.20.0) starting in system bus...
Sep  4 21:22:37 victim systemd[1]: Started User Login Management.
Sep  4 21:22:38 victim systemd[1]: Reached target Sound Card.
Sep  4 21:22:38 victim systemd[1]: Starting GRUB failed boot detection...
Sep  4 21:22:38 victim udisksd[640]: Failed to load the 'mdraid' libblockdev plugin
Sep  4 21:22:38 victim accounts-daemon[593]: started daemon version 22.07.5
Sep  4 21:22:38 victim systemd[1]: Starting Hold until boot process finishes up...
Sep  4 21:22:38 victim systemd[1]: Started Unattended Upgrades Shutdown.
Sep  4 21:22:38 victim systemd[1]: gpu-manager.service: Deactivated successfully.
Sep  4 21:22:38 victim systemd[1]: Finished Detect the available GPUs and deal with any system changes.
Sep  4 21:22:38 victim systemd[1]: Started Accounts Service.
Sep  4 21:22:39 victim systemd[1]: grub-initrd-fallback.service: Deactivated successfully.
Sep  4 21:22:39 victim systemd[1]: Finished GRUB failed boot detection.
```

Figure 86 Examples of Syslog Messages

Authentication Logs

Authentication logs maintain a record of all login activities and authentication-related events on Linux systems. They capture successful and failed login attempts, SSH session initiations, sudo command usage, and account management actions. These logs are written in plain text and usually include a timestamp, hostname, the responsible service such as sshd, vsftpd or sudo, and the outcome of the authentication attempt. All this information is recorded in the authentication log, which is stored at **/var/log/auth.log**. These records play a crucial role in identifying brute-force attacks, privilege escalation, and lateral movement across the network.

```
Aug 29 19:38:57 victim sshd[3576]: Connection closed by invalid user user 192.168.100.228 port 44250 [preauth]
Aug 29 19:38:57 victim sshd[3576]: PAM 1 more authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=192.
168.100.228
Aug 29 19:38:57 victim sshd[3569]: Failed password for invalid user user from 192.168.100.228 port 52634 ssh2
Aug 29 19:38:57 victim sshd[3570]: Failed password for invalid user user from 192.168.100.228 port 52636 ssh2
Aug 29 19:38:57 victim sshd[3589]: Connection closed by invalid user user 192.168.100.228 port 44316 [preauth]
Aug 29 19:38:57 victim sshd[3589]: PAM 1 more authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=192.
168.100.228
Aug 29 19:38:58 victim sshd[3570]: Connection closed by invalid user user 192.168.100.228 port 52636 [preauth]
Aug 29 19:38:58 victim sshd[3570]: PAM 2 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=192
.168.100.228
Aug 29 19:38:58 victim sshd[3569]: Connection closed by invalid user user 192.168.100.228 port 52634 [preauth]
Aug 29 19:38:58 victim sshd[3569]: PAM 2 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=192
.168.100.228
Aug 29 19:43:32 victim sshd[756]: exited MaxStartups throttling after 00:05:03, 7 connections dropped
Aug 29 19:43:35 victim sshd[3620]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
rhost=192.168.200.12  user=nayomi
Aug 29 19:43:37 victim sshd[3620]: Failed password for nayomi from 192.168.200.12 port 50137 ssh2
Aug 29 19:43:42 victim sshd[3620]: Accepted password for nayomi from 192.168.200.12 port 50137 ssh2
Aug 29 19:43:42 victim sshd[3620]: pam_unix(sshd:session): session opened for user nayomi(uid=1000) by (uid=0)
Aug 29 19:43:42 victim systemd-logind[638]: New session 4 of user nayomi.
Aug 29 19:43:42 victim systemd: pam_unix(systemd-user:session): session opened for user nayomi(uid=1000) by (uid=0)
```

Figure 87 Examples of Authentication log Message

VSFTPD Logs

The VSFTPD service generates logs that capture FTP server activity, including connection attempts, login successes and failures, and file transfer operations such as uploads and downloads. The logs are stored in plain text, with each entry typically containing a timestamp, the FTP command executed, the client IP address, the username involved, the status of the session, filename, file size etc. This information is recorded in **/var/log/vsftpd.log**, providing clear visibility into FTP usage and help in identifying unauthorized file transfers or compromised credentials targeting the service

```
Fri Aug 29 19:34:39 2025 [pid 3378] CONNECT: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:39 2025 [pid 3381] CONNECT: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:39 2025 [pid 3382] CONNECT: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:40 2025 [pid 3384] CONNECT: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:40 2025 [pid 3371] [user] FAIL LOGIN: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:40 2025 [pid 3369] [user] FAIL LOGIN: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:40 2025 [pid 3375] [user] FAIL LOGIN: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:40 2025 [pid 3373] [user] FAIL LOGIN: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:40 2025 [pid 3367] [user] FAIL LOGIN: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:42 2025 [pid 3379] [user] FAIL LOGIN: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:42 2025 [pid 3380] [user] FAIL LOGIN: Client "::ffff:192.168.100.248"
Fri Aug 29 19:34:42 2025 [pid 3377] [user] FAIL LOGIN: Client "::ffff:192.168.100.248"
Fri Aug 29 19:42:37 2025 [pid 3593] CONNECT: Client "::ffff:192.168.200.12"
Fri Aug 29 19:42:37 2025 [pid 3592] [nayomi] OK LOGIN: Client "::ffff:192.168.200.12"
Fri Aug 29 19:42:46 2025 [pid 3597] [nayomi] OK DELETE: Client "::ffff:192.168.200.12", "/test1GB.bin"
Fri Aug 29 19:42:51 2025 [pid 3613] CONNECT: Client "::ffff:192.168.200.12"
Fri Aug 29 19:42:51 2025 [pid 3612] [nayomi] OK LOGIN: Client "::ffff:192.168.200.12"
Fri Aug 29 19:42:51 2025 [pid 3614] [nayomi] OK UPLOAD: Client "::ffff:192.168.200.12", "/testfile.txt.txt", 0.00Kbyt
e/sec
Sun Aug 31 18:07:14 2025 [pid 3674] CONNECT: Client "::ffff:192.168.200.12"
Sun Aug 31 18:07:15 2025 [pid 3673] [nayomi] OK LOGIN: Client "::ffff:192.168.200.12"
Sun Aug 31 18:07:21 2025 [pid 3677] CONNECT: Client "::ffff:192.168.200.12"
Sun Aug 31 18:07:21 2025 [pid 3676] [nayomi] OK LOGIN: Client "::ffff:192.168.200.12"
```

Figure 88 Examples of FTP log Messages

Windows Logs

Windows event logs capture system, application, and security-related events on Windows VM. Windows events are stored in a binary format known as EVTX, which is structured and viewable through the Event Viewer application or exportable in XML or JSON for analysis. They provide important information for comprehensive correlation and threat detection across the environment.

Figure 89 Examples of Windows Log Messages

Appendix B : Source Code Snippets

This appendix provides code snippets used in the project, highlighting key functions and processes that were essential for the automated alert classification and management framework.

1.MISP Threat Intelligence API Call

This snippet demonstrates the setup required to connect to a MISP instance using the ExpandedPyMISP library. It includes suppression of warnings, configuration for pandas display, and initialization of the MISP object for further API interactions such as searching for or posting threat intelligence events.

```
from pymisp import ExpandedPyMISP
import re


MISP_URL = "https://misp.local/"
API_KEY = "Y8gKoub27x31uA3Bcqm16tMPipw3rkhnAa1JyY83"
VERIFY_SSL = False
misp = ExpandedPyMISP( *args: MISP_URL, API_KEY, VERIFY_SSL)
```

Figure 90 MISP AIP call code

2.Log Co-relation

This snippet shows how logs are correlated within the system. It checks whether log entries match a specific time window and source-destination IP pair. Matching entries are tagged with a new incident ID, and relevant event details are extracted for further analysis and correlation with other data sources.

```
# --- AUTHLOG ---
in_authlog_mask = (filtered_authlog_df['time_range'] == tr) & (filtered_authlog_df['source_dest_ip'] == sd_ip)
in_authlog = bool(in_authlog_mask.any())
auth_data = {}
if in_authlog:
    filtered_authlog_df.loc[in_authlog_mask, 'incident_id'] = new_incident_id
    auth_rows = filtered_authlog_df.loc[in_authlog_mask]
    if not auth_rows.empty:
        auth_match = auth_rows.iloc[0].to_dict()
        auth_data = {
            'event_dataset_auth': auth_match.get('event_dataset'),
            'message_auth': auth_match.get('message'),
            'process_name_auth': auth_match.get('process_name'),
            'event_outcome_auth': auth_match.get('event_outcome'),
            'event_category_auth': auth_match.get('event_category'),
            'event_action_auth': auth_match.get('event_action')
        }
```

Figure 91 Co-relation logic

3.Prediction Confidence

This snippet calculates the prediction confidence for a given alert or incident based on feedback counts. It identifies the dominant feedback type, computes its proportion, and assigns a confidence label (High, Medium, or Low). This helps prioritize alerts and guide analysts in focusing on the most reliable predictions.

```python
def get_prediction_confidence(feedback_counts, incident_ids):

    total_feedback = sum(feedback_counts.values())
    if total_feedback == 0:
        return None, 0, "N/A", []

    # calculate proportions
    proportions = {fb: c / total_feedback for fb, c in feedback_counts.items()}

    # pick dominant feedback (highest proportion)
    dominant_feedback, max_prop = max(proportions.items(), key=lambda x: x[1])
    dominant_percent = round(max_prop * 100)

    # confidence range
    if dominant_percent >= 80:
        confidence_label = "High"
    elif dominant_percent >= 60:
        confidence_label = "Medium"
    else:
        confidence_label = "Low"

    dominant_count = feedback_counts[dominant_feedback]
    dominant_incidents = incident_ids[:dominant_count]

    return dominant_feedback, dominant_percent, confidence_label, dominant_incidents
```

Figure 92 Prediction Confidence

4.Alert Prediction with Historical Feedback

For each matched alert, historical feedback is checked to refine the prediction. The system calculates the dominant feedback type, assigns a confidence label, and records the reasoning behind the prediction, referencing past incidents that informed the decision. This process ensures that alert predictions are data-driven, context-aware, and traceable.

```
tp_mask = (df['has_ioc'] == True) & (df['in_suricata'] == True)

df.loc[tp_mask, 'initial_prediction'] = 'True Positive'
for idx in df[tp_mask].index:
    comments = []
    rule_id = df.at[idx, 'rule_id_suri']
    src_ip = df.at[idx, 'source_ip']
    dst_ip = df.at[idx, 'destination_ip']

    # ---Historical feedback check ---
    try:
        hist_row = feedback_group.loc[(rule_id,src_ip, dst_ip)]
        feedback_counts = hist_row['feedback_counts']
        incident_ids = hist_row['incident_ids']

        dominant_feedback, dominant_percent, confidence_label, dominant_incidents = get_prediction_confidence(feedback_counts, incident_ids)

        if dominant_feedback:
            # filter incident_ids for only those that match the dominant feedback
            dominant_count = feedback_counts[dominant_feedback]
            dominant_incidents = incident_ids[:dominant_count]

            df.at[idx, 'initial_prediction'] = dominant_feedback
            df.at[idx, 'prediction_confidence'] = confidence_label
            df.at[idx, 'prediction_reason'] = (
                f"This incident has been classified as {dominant_feedback} "
                f"with {dominant_percent}% confidence ({confidence_label}). "
                f"Decision is based on prior analyst feedback from incidents: "
                f"{', '.join(map(str, dominant_incidents))}."
            )
            continue
    except KeyError:
        pass
```

Figure 93 Alert Prediction with Historical Feedback

5. Auth log-Based Alert Evaluation

This snippet checks whether an alert is present in the authentication logs. It counts the number of authentication attempts. The reasoning for the prediction is recorded in the comments for traceability and analysis.

```
# --- authlog ---
if df.at[idx, 'in_authlog']:
    attempts = extract_auth_attempts(df.at[idx, 'message_auth'])
    if attempts and attempts > 5:
        reason = f"{attempts} authentication attempts detected in authlog"
        prediction = "True Positive"
    else:
        reason = "low number of authentication attempts detected in Authlog"
    comments.append(reason)
```

Figure 94 Auth log-Based Alert Evaluation

6. VSFTPD-Based Alert Evaluation

This snippet evaluates alerts based on VSFTPD (FTP) logs. It extracts the number of login attempts and the amount of data transferred in FTP sessions.

```
# --- vsftpd ---
if df.at[idx, 'in_vsftpd']:
    ftp_activity = extract_ftp_activity(df.at[idx, 'message_vsftpd'])
    ftp_attempts = ftp_activity.get("login_attempts")
    ftp_bytes = ftp_activity.get("bytes_transferred")

    # Thresholds
    LOGIN_THRESHOLD = 5
    BYTES_THRESHOLD = 20_000_000  #  (~20 MB)

    reason_parts = []
    # --- Check login attempts ---
    if ftp_attempts is not None and ftp_attempts > LOGIN_THRESHOLD:
        reason_parts.append(f"{ftp_attempts} authentication attempts detected in FTP")
        prediction = "True Positive"

    # --- Check bytes transferred ---
    if ftp_bytes is not None and ftp_bytes > BYTES_THRESHOLD:
        reason_parts.append(f"{ftp_bytes} bytes transferred in FTP session")
        prediction = "True Positive"

    if not reason_parts:
        reason_parts.append("Low number of FTP attempts and small/no file transfer detected")

    reason = " and ".join(reason_parts)
    df.at[idx, 'initial_prediction'] = prediction
    df.at[idx, 'prediction_reason'] = reason

    comments.append(reason)
```

Figure 95 VSFTPD-Based Alert Evaluation

7. Syslog-Based Alert Evaluation

This snippet evaluates alerts using system logs. It counts suspicious events and checks if the total or repeated occurrences exceed predefined thresholds. If thresholds are met, the alert is classified as a True Positive.

```
# --- syslog ---
if df.at[idx, 'in_syslog']:
    syslog_counts = extract_syslog_attempts(df.at[idx, 'message_sys'])
    total_suspicious = sum(syslog_counts.values()) if syslog_counts else 0
    if total_suspicious > 10 or (syslog_counts and any(v > 5 for v in syslog_counts.values())):
        reason = "high/repeated syslog events: " + ", ".join(
            f"{k.upper()}={v}" for k, v in syslog_counts.items() if v > 0
        )
        prediction = "True Positive"
    else:
        reason = "low number of syslog events"
    comments.append(reason)
```

Figure 96 Syslog-Based Alert Evaluation