Princess Sumaya University for Technology
King Hussein School for Computing Sciences
Cybersecurity Department

A Lightweight and Adaptive Logging Architecture

Prepared By:

Jana Abdeen 20210507
Diala Alazzeh 20210116
Eyad Alssouqi 20200528

Supervised By:

Prof.Mohamad Al Nabhan

Project Submitted in Partial Fulfillment of the Requirements for the Degree of Science in Cybersecurity

Submitted: May 2025

Declaration of Originality

This document has been written entirely by the undersigned team members of the project. The source of everyquoted text is clearly cited and there is no ambiguity in where the quoted text begins and ends. The source of any illustration, image or table that is not the work of the team members is also clearly cited. We are aware that using non-original text or material or paraphrasing or modifying it without proper citation is a violation of the university's regulations and is subject to legal actions.

Names and Signatures of team members:

Diala Alazzeh 20210116

Eyad Alssouqi 20200528

Jana Abdeen 20210507

Acknowledgements

We would like to convey our deepest appreciation to Dr.Mohamad Alnabhan, whose guidance, support and encouragement have played a key role in helping us successfully complete this project. We owe special thanks to all the doctors who have taught us over the past four years. Their dedication and commitment to teaching have shaped our understanding and skills, helping us grow both academically and personally. The time, effort, and care they invested in us have been a source of motivation and inspiration that will stay with us as we move forward in our careers.

Table of Contents

List of Figures

List of Tables

Abbreviations

MFA: Multi Factor Authentication
SOC :Security Operations Center
GUI: Graphical user interface
IT: Information Technology
CIA: Confidentiality, Integrity, Availability
IOPS: Input/output operations per second
RAM: Random Access Memory
CPU: Central processing unit
KPI: Key performance indicators
MVP: minimum viable product
JWT: JSON Web Token
SSO: Single Sign-On
IDS: Intrusion Detection System
DMZ: Demilitarized Zone

A Lightweight and Adaptive Logging Architecture

By
Diala Alazzeh

Jana Abdeen

Eyad Alssouqi

Supervisor
Prof. Mohammad Alnabhan

Abstract

Since the use of information has become more common nowadays,
the diversity of logs has increased more and more overtime, due to
that; the need for a lightweight and adaptive log management system
has become essential. Organizations need to collect, store, and analyze large volumes of log data, and keep them for suitable durations. Log management is the process of generating, transmitting, storing, analyzing, and disposing log data from network to databases.
Traditional logging architecture often creates huge amounts of
complex data that are hard to manage with limited security controls.
In addition, constant alerts, many of which are false alarms, lead to
"Alert fatigue," making it easy to miss real issues.
Our solution aims to offer an adaptive, lightweight logging architecture with customized KPIs that is scalable to meet the needs of any organization and minimizes data complexity, yet still maintains high security standards.
This approach will enable faster detection and response to cyber threats, reduce operational burden on SOC teams, and ensure organizations can effectively safeguard their environments with better performance.

Introduction

1.1 Overview

Organizations are facing increasing challenges with the rapid data growth, and maintaining data privacy regulations are creating pressure on IT spending. Most of data leaders are struggling with architectural complexity.[1] Logs are a critical part of troubleshooting. Developers need logs to help resolve issues and manage application workflow. But the sheer amount of log data created is expensive and hard to dissect. The reliability and usefulness of a logging solutions depends on two main functions: transferring data between applications in an arranged and real-time manner and applying abstraction that allows organizations to share data more easily. Reducing data complexity by extracting only the specific information relevant to an organization's needs from selected logs will improve performance and provide an optimized solution. Additionally, implementing encryption techniques will ensure the confidentiality, integrity, and availability (CIA) of the data. This project is personally meaningful to us as a group as it represents a big step toward building a career in cybersecurity. At the same time, the project is motivated by the growing demand for smart, lightweight, and adaptable logging systems, ones that can adjust to new threats in real time, keep data secure, and respect privacy.

1.2 Problem Statement

Problem Description

Managing log data can be both costly and complicated. As systems grow, so does the volume of logs generated, making it expensive to store and manage as data increases.[2]

1.One of the major problems regarding managing log data is the lack of important/crucial messages in log statements, which is reported in 35 (62.5%) studies. The most important decisions about logging are usually left to the programming stage that ends up with insufficient logging. [3]

2. Effective monitoring is crucial in real-time, as issues can arise unexpectedly. Without timely alerts, diagnosing and fixing the issue becomes much harder, potentially leading to longer downtimes and frustration.

3. Diversity of log messages is also an issue that has been talked about a lot, which has been mentioned in 9 (16.1%) studies. Because of the format, content, and the unstructured nature of log messages is generally dissimilar and different, especially with the increase of system complexity, which demands having more developers.

4.The content of logs may contain sensitive information that gets leaked due to security gaps in logging applications. Our project aims to address these challenges, making them more secure and scalable. With this system, logs will be centralized, making it faster and easier

to analyze them in real-time. [4]

Category	Issue	Primary studies	Percentage
Where & What	Performance overhead	[Chen 17a], [Chow 18], [Ding 15], [Fu 14], [Ghol 20], [Jia 18], [Kabi 16a], [Lal 16b], [Lal 17], [Lal 15], [Lal 16a], [Lal 16c], [Lal 19], [Li 20b], [Li 17a], [Li 17b], [Li 18a], [Li 20a], [Liu 20], [Liu 19], [Luo 18], [Marr 18], [Mizo 19], [Sain 16], [Shan 14], [Yao 18], [Yuan 12b], [Yuan 12a], [Zeng 19], [Zhao 17], [Zhu 15]	55.4%
What	Lacking crucial messages	[Cinq 09], [Cinq 10], [Cinq 20], [Cinq 12], [Fu 14], [Ghol 20], [Hass 18], [Jia 18], [Kubo 20], [Lal 16b], [Lal 17], [Lal 15], [Lal 16a], [Lal 16c], [Lal 19], [Li 20b], [Li 17a], [Li 17b], [Li 18a], [Li 18b], [Li 20a], [Liu 20], [Liu 19], [Luo 18], [Mizo 19], [Rong 18], [Sain 16], [Tova 13], [Yao 18], [Yuan 12b], [Yuan 12a], [Yuan 12c], [Zeng 19], [Zhao 17], [Zhi 19]	62.5%
	Redundant or useless messages	[Anu 19], [Cinq 09], [Cinq 12], [Ding 15], [Fu 14], [Hass 18], [Lal 17], [Lal 16a], [Lal 16c], [Li 20b], [Li 17a], [Li 18a], [Li 18b], [Li 20a], [Liu 20], [Liu 19], [Marr 18], [Sain 16], [Shan 14], [Tova 13], [Zeng 19], [Zhi 19], [Zhu 15]	41.1%
	Incorrect or ambiguous messages	[Chen 17a], [Cinq 09], [Cinq 10], [Cinq 12], [Hass 18], [He 18], [Kim 19], [Li 19], [Li 20a], [Pecc 15], [Pecc 12], [Tova 13], [Zhi 19]	23.2%
	Heterogeneity of the log messages	[Chen 20], [Cinq 09], [Ghol 20], [He 18], [Liu 20], [Marr 18], [Pecc 15], [Salf 04], [Tova 13]	16.1%
	Leakage of sensitive data	[Li 20a], [Zhi 20], [Zhou 20]	5.4%
How well	Maintenance barriers	[Chen 17b], [Chen 19], [Chen 17a], [Ghol 20], [Li 18b], [Pecc 15], [Yuan 12b], [Li 20a], [Kabi 16b], [Shan 14]	17.9%
	Difficulties in V&V of log statements	[Chen 19], [Rong 20]	3.6%

Figure 1:Distribution of Issues in Logging Practices

1.2.2 Project Outcomes

This project will deliver a flexible and secure logging system that includes:

- **Real-time log collection:** Logs from different sources will be gathered and processed instantly, so teams can respond quickly to any issues.
- **Customizable logs and KPI:** Security teams will be able to track logs and set up notifications to spot potential threats faster.
- **Simplicity and speed:** By using automated alerts, teams can quickly identify and address potential threats.
- **Cost-effectiveness:** By using open-source tools and scalable architecture, the system will offer a more affordable alternative to traditional, expensive logging solutions.
- **Scalability**: The architecture will be designed to handle increasing amounts of data as the organization grows, ensuring it can scale effectively without compromising performance.

1.2.3 Target Audience and Impact

The main users of this system will include:

1. Cybersecurity teams in high-stakes environments such as government agencies, financial institutions, and healthcare organizations.
2. Organizations with Limited Resources: Small and medium-sized businesses that lack the resources for complex and expensive logging systems can use your scalable and lightweight solution to enhance their cybersecurity without overwhelming their teams.
3. IT administrators and security engineers responsible for monitoring and analyzing logs.
4. Large Enterprises: Bigger organizations dealing with massive log volumes and complex environments.
5. Security Operations Center (SOC) teams who handle 24/7 monitoring security events and incidents data.

1.3 Project objectives

1. **Investigate existing logging systems:** First, we will explore and analyse the current logging systems used in critical sectors like banking and government.

2. **Design and implement an adaptive data model for logs:** Develop an efficient data model for managing logs, ensuring it can adapt to various organizational requirements while maintaining simplicity and security.
3. **Design and implement a distributed logging architecture:** Then, we'll create a distributed logging system capable of efficiently collecting and processing log data from multiple sources. The system will be designed for scalability and real-time monitoring, ensuring that logs are stored safely and retrieved quickly when needed.
4. **Implement Graphical User Interfaces for security engineers:** GUIs will assist security engineers to understand the current logging situation, system performance and receive real-time alerts.
5. **Evaluate and test the entire logging solution:** Finally, we'll test the complete system, evaluating its performance, security features, and reliability under different conditions. This will involve both stress testing and security audits to ensure the system meets the required standards and functions effectively in real-world situations.

1.4 Project Contribution

-Development of a New Data Model: The project introduces a new data model designed to improve the efficiency of log collection and analysis. This model simplifies the way logs are processed, making it faster and easier for SOC teams to identify and respond to potential threats.
-Implementation of a Lightweight Logging Architecture: Scalable and lightweight logging system will be designed and implemented to work effectively at different organizational levels. This architecture reduces data complexity, ensures high security, and adapts to the needs of both small and large organizations.

1.5 Outline of the report

Table 1 Outline

Chapter One: Introduction	This chapter provides an overview of logging architectures and explains the problem our project aims to solve, along with a review of related systems.
Chapter Two: Project Plan	This chapter describes our project plan, including the expected deliverables, tasks such as Analysis, Design and Implementation, roles and responsibilities, possible risks, overall costs, and the tools we used.
Chapter Three: Literature Review and Related Work	This chapter discusses the knowledge gaps in logging systems and detailed information about similar systems and compares our work with older designs.

Chapter Four: Requirements Specifications	This chapter describes the requirements of our project, including stakeholders and the needs of each one of them, the platform requirements, and other requirements of the system.
Chapter Five: System Design	This chapter provides a detailed presentation of the logical model design and physical model design of the solution.
Chapter Six: Implementation	This chapter describes the implementation details such as the general implementation description, pipeline implementation description, and any additional implementation details.
Chapter Seven: Testing	This chapter discusses the testing approach and the results of the testing and discussion of the results.
Chapter Eight: Conclusion and Future Work	Summary of key findings and accomplishments. Outline of future work including expanding datasets, enhancing user experience, and conducting user and industry feedback surveys. Reflection on the project's impact on cybersecurity and its potential for future developments.

Chapter 2

Project Plan

2.1Project Deliverables

Table 2 Deliverables

Deliverable Name	Description
Source Code	Source code for the adaptive data model, and lightweight logging architecture.
Documentation Files	Document visual and textual information, guides, steps and explanations. Documentation for all tools used, including their installation guides, setup, and any specific directions for these tools.
Test and Evaluation Reports	Detailed reports on the performance, scalability, and security testing of the logging architecture.
Datasets and Logs	Sample log datasets used for testing and demonstrating the system's functionality.
Final Project Report	A complete project report summarizing the research, design, implementation, testing, and evaluation phases.

2.2 Project Tasks

Table 3 Project Tasks

Task ID	Task Name and Description	Dependencies	Duration(in weeks)
T1	Analysis: Requirement Identification- Identifying and defining project requirements, including fact-finding methods, outputs, inputs, processes, performance, security levels, and scalability.	None	1
T2	Analysis: Requirement Modelling- Developing models representing the identified requirements, ensuring clarity and completeness.	T1	1
T3	Analysis: Development Strategy- Deciding on the hardware, software tools, and programming languages to be used.	T2	1
T4	Design: Reports Layout - Designing the layout and structure of project reports.	T3	1
T5	Design: User Interfaces - Creating the design for system user interfaces.	T4	2
T6	Design: Database Design - Planning and designing the database architecture and schema.	T5	2

T7	Implementation: Coding - Developing the codebase for the project.	T6	6
T8	Implementation: Testing -Strict testing of the developed system to ensure functionality and security.	T7	4
T9	Implementation: Documentation -Preparing comprehensive documentation, including a requirements document.	T8	3

Figure 2 represents the Pert chart of the Project Tasks to help visualize the tasks involved in a project and how they relate to each other. It also helps in managing task dependencies, showing which tasks must be completed before others can start.

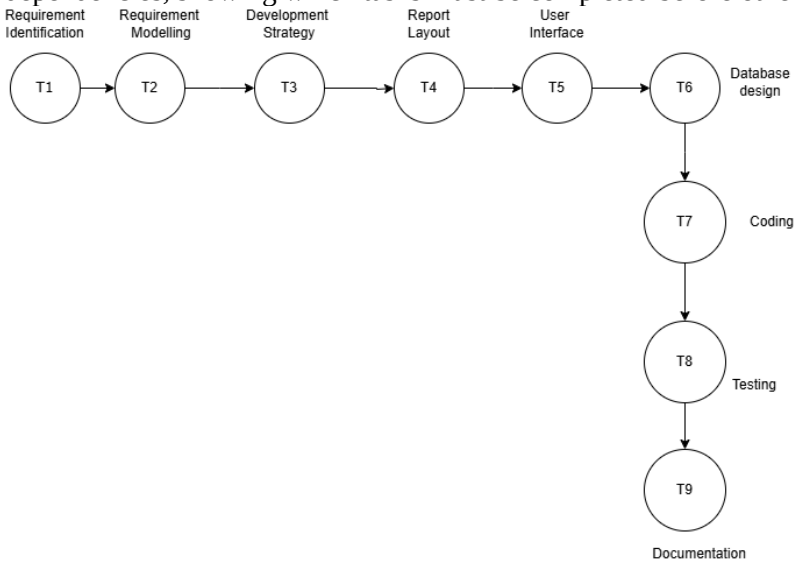


Figure 2 Pert Chart

Figure 3 represents the Gantt chart of the project which helps in displaying the start and end dates for each task and identifying the relationships between tasks, such as which tasks depend on the completion of others.

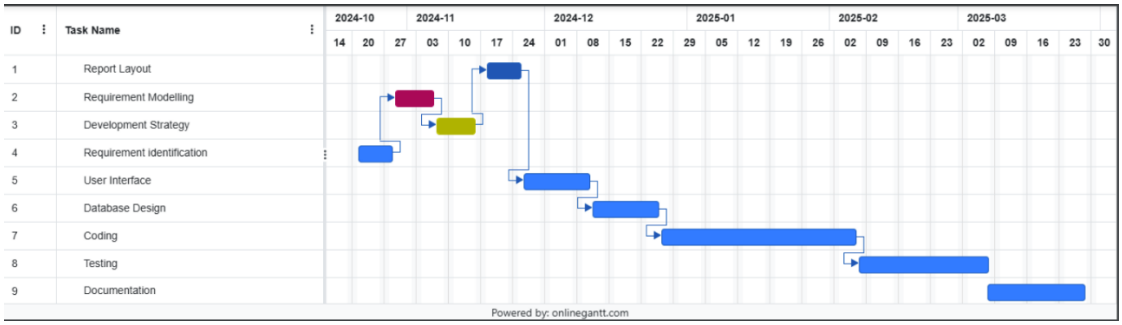


Figure 3 Gantt Chart

2.3 Roles and Responsibilities

Table 4 ResponsibilitiesTable 5 Risks

Member Name	Responsibilities
Diala Alazzeh	Target audience-Problem Statement-Project Outcomes-Project Tasks-Risk Assessment-Knowledge Gap-Nonfunctional Requirements-Other Requirements-use case diagram-sequence diagram-dashboard-architectural design-Documentation-Log extraction scripts-Future Work-Poster-PowerPoint-User Manual-Pipeline Implementation Description-Comparison between Structured and Raw logs
Jana Abdeen	Abstract-Target audience-Project objectives-Project contribution-Project Deliverables-Related Work-Stakeholders-Literature-Database design-logical design-risk assessment-Implementation-Testing-Code Documentation-General Implemenation Description
Eyad Alssouqi	Abstract-Project Importance and Motivation-Knowledge Gap-Platform Requirements-Functional Requirements-nonfunctional requirements-other requirements-document changes-Log extraction scripts-Video

2.4 Risk Assessment

The risk assessment table, Table 5, is considered a primary part to identify the project's risk management process. It consistently documents possible risks, alongside their attributes, and mitigation strategies. The table is integral for understanding, monitoring, and communicating the risk landscape throughout the project life cycle.

● Risk ID and Description: Each risk is given a unique identifier and a detailed description.
● Probability of Occurrence: This column gives an estimation of the probability of each risk occurring: low, medium, or high.
● Impact Level: The potential impact on the project if the risk materializes, rated as low, medium, or high.
● Mitigation Plan: Specific actions or plans to reduce or eliminate the risk.

Risk ID	Risk	Probability	Impact	Mitigation
R1	Lack of Data Sources	High	May result in inaccurate outputs and irrelevant system performance.	Use simulated or publicly available datasets, and ensure early communication with stakeholders to acquire real data promptly.

R2	Technical Complexity	Medium to High	Slows down development due to unpredicted challenges.	Break down complex tasks, and seek support from external resources like forums and documentation.
R3	Big Scope	High	Leads to resource exhaustion or missed deadlines.	Prioritize critical components, follow a modular development approach, and ensure tasks are realistic given the timeline.
R4	Time Constraints	High	Reduces overall quality or delays the project.	Create a detailed project timeline with buffer periods for unforeseen delays and focus on delivering the MVP.
R5	Skills and Capabilities	Medium to High	Affects system quality and ability to meet objectives.	Dedicate time for self-learning, seek mentorship, or distribute tasks according to team members' strengths and interests.

2.5 Cost Estimation

Table 6 Cost Estimation

Item	Cost (USD)	Details
Central Server	10$	Central point where all logs are aggregated and collected.
Monitoring Server	15$	Analyses logs received from logging server.
Backup Server	10$	Backs up logs to meet compliance or disaster recovery needs.
Log collectors	10$	Between logging server and application zone

2.6 Project Management Tools

While developing our project, “A Secure and Adaptive Logging Architecture”, we have successfully engaged a variety of project management tools enabling cooperation, task management and technical execution, these tools were extremely useful in ensuring the efficiency and success in our project. Below is a well-detailed full description of each tool and its application throughout our project:

Table 7 Tools

Tool	Description
Google Docs	We employed Google Docs for collaborative document creation and editing. Its real-time collaboration features enabled work by multiple

	team members on the same document simultaneously which facilitated a flexible approach to revise then finalize our project report.
Visual Paradigm	Used to design Gantt and PERT charts.
Google Drive	Used to share documents among team members.
Zoom	Zoom was primarily used as a communication tool, by conducting virtual meetings, discussions, and other sessions. It provided the feature of screen-sharing
ERD Plus	Used to design the ER diagram and the relational schema for the database.
Draw.io	For creating diagrams, flowcharts, and other visual aids, Draw.io was our tool of choice it facilitated the clear depiction of network diagrams, structural frameworks and flow charts. The visualizations that we created using Draw.io improved the clarity of the technical explanations and contributed to a more inclusive understanding of the project structure.

Chapter 3

Literature Review and Related Work

3.1 Related Work

Reported issues about data complexity:
NetApp's 2024 Data Complexity Report Reveals AI's Make or Break Year Ahead
NetApp, the intelligent data infrastructure company, published a data complexity report that shows how organizations are facing complexity in managing and logging their data. This report provides a vision into how AI will affect organizations in 2025 and later, providing information that can help organizations in maximizing AI's potential while managing the complexities associated with this technology. [5]

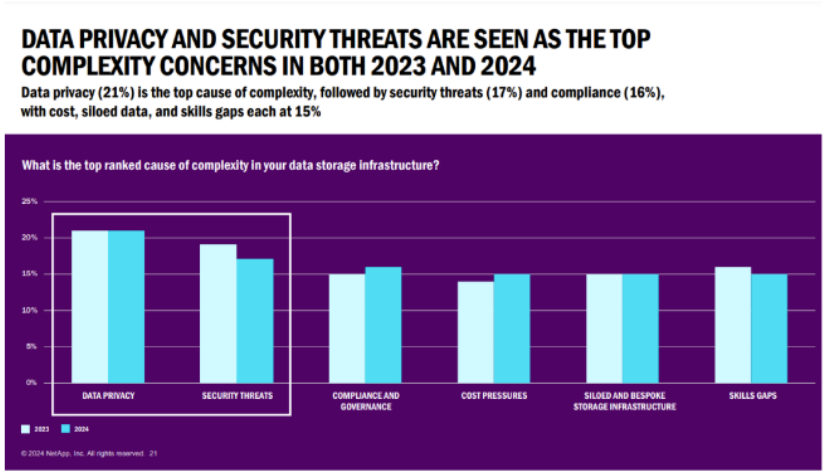


Figure 4 Complexity Concerns

Lightweight Models
Lightweight models are divided into two types. The first type is to reduce existing models, and the other type is to redesign a new logging model [6]. Designing a lightweight architecture involves creating models with less parameters. Below is an example of parsing log messages and log events.

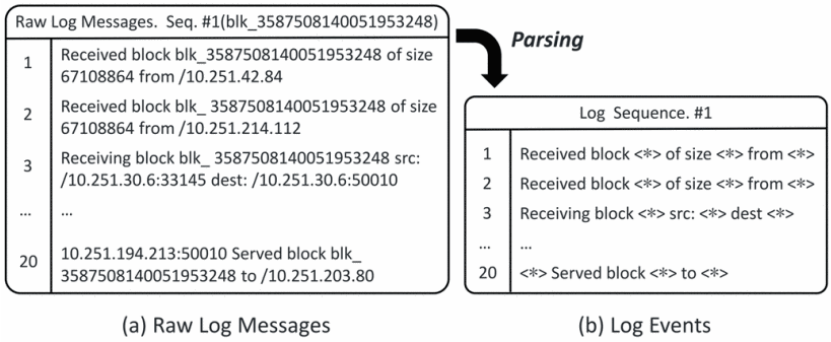


Figure 5 Example of raw logs,log events, and sequence[7]

To understand our solution more and how it works, it's important to discuss existing log management systems and tools to understand their strengths and limitations. To

understand our solution more and how it works, it's important to discuss existing log management systems and tools to understand their strengths and limitations.

1.Splunk

Splunk is a log management option designed to handle large and big data and analyzes them in real time. It has several characteristics including:

-Gathers data from different sources including logs.

-Searching features that lets users locate certain log records quickly and easily.

-Alerting functions that notify users of possible issues or threats. [8]

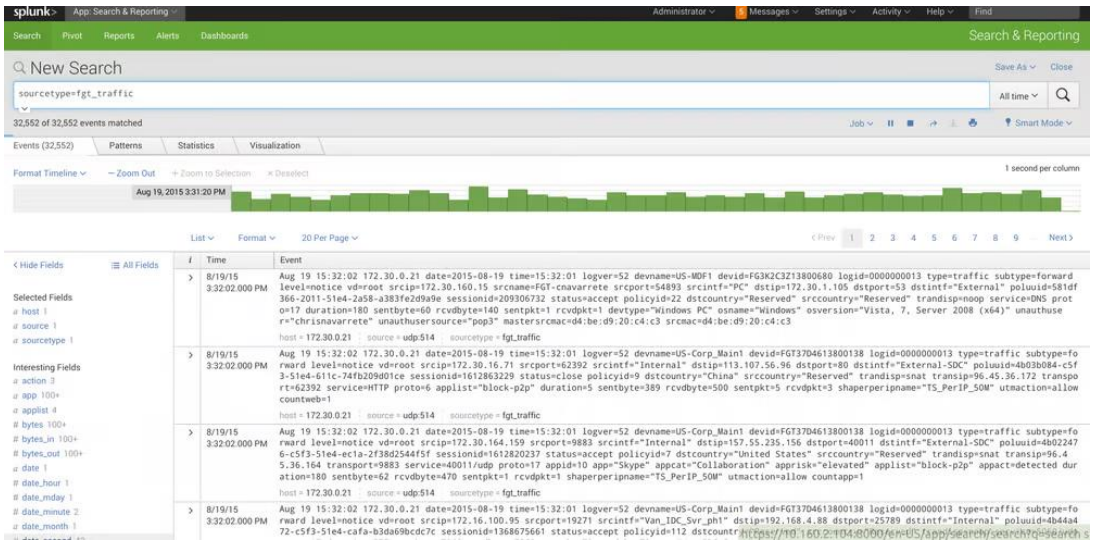


Figure 6 Splunk

Although it has a lot of good functionalities, Splunk can be complex and very hard to use for beginners. The cost of it depends on the size of data and it can be very costly for big organizations that have large data volumes.

2. Better Stack

Better Stack is another log monitoring tool that allows querying logs using SQL log management. It provides monitoring, alerting, log management and filtering of large datasets. [9]

Figure 7 Better Stack

Advantages:

-Speed: Efficient querying of large log datasets.
-Anomaly Detection: Supports setting alerts for unusual log patterns.
-Cost Efficiency: Designed to be resource-efficient, potentially reducing operational costs.
Limitations:
-Feature Set: May lack some advanced features present in more established platforms.
-Integration: Compatibility with existing systems and tools may require evaluation.
Our solution shares Better Stack's emphasis on speed and efficiency but aims to further enhance adaptability and user-friendliness.
3. Logstash
Logstash (part of the elastic stack) is designed to gather data from a variety of sources, apply transformations or filtering as needed, and forward the processed data to a destination, such as Elasticsearch or another storage system.[10]

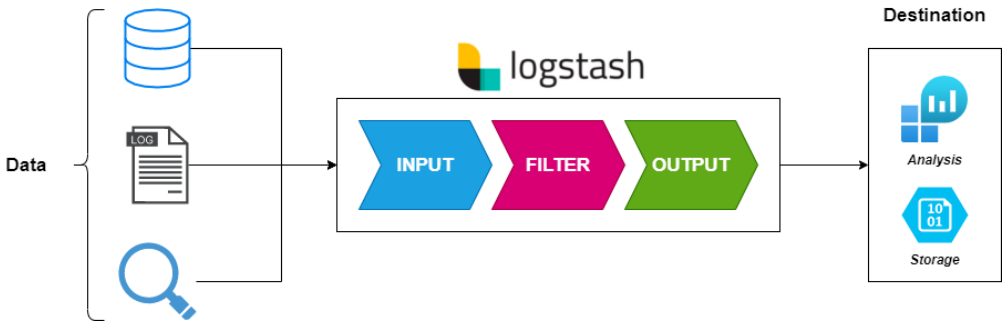


Figure 8 Logstach

Pricing:Logstash offers a free trial with multiple plans starting with $95/month all the way to $175/month.

4.LogicMonitor
LogicMonitor is a cloud-based system that integrates monitoring in hybrid settings and enables correlation of important log events on a centralized monitoring [10]

Figure 9 Logic Monitor

LogicMonitor decreases troubleshooting times and enables better workflow.In conclusion, while existing log management systems offer a range of features,they often come with complexities and resource demands that may not align with the needs of all organizations. Our lightweight logging architecture seeks to address these gaps by providing a scalable, efficient, and user-friendly solution tailored to diverse organizational requirements.

3.2 Knowledge Gap

Managing and analysing log data is a big challenge for organizations as the amount of data grows more and more every day, and while there are tools and systems that handle logs, there are still areas where we lack knowledge. These gaps make it harder to build systems that are efficient, and scalable enough to meet today's needs. In this section, we'll focus on the key knowledge gaps we've identified in log management and analysis.

1. Cross-Platform Log Collection

One of the main challenges we've identified is the lack of a unified way to collect logs across different environments, such as on-premises systems, cloud platforms, and IoT devices. Each of these environments produces logs in its own unique format, making it difficult to gather, standardize, and process them all in one place.

Tools like Fluentd offer some solutions. For example, Fluentd can process logs by adding context, reformatting them, and sending them to storage. It also supports a wide range of plugins to connect with various data sources and outputs, like log management systems or big data platforms. However, these tools often struggle with performance issues when handling large amounts of logs or working in complex environments.

2. Lightweight Log Processing Architectures

Processing large amounts of log data efficiently is a real challenge. There aren't enough studies comparing lightweight log processing systems in terms of how well they scale, how efficiently they use resources, or even how fast they can handle data. The increasing volume of logs being generated, it's important to focus on systems that can process this data quickly and without wasting resources. One good approach is using MapReduce, a framework that handles large-scale data tasks. It's good for things like indexing and aggregating logs, as these tasks fit naturally into its design.

3. User-Friendly Visualization and Dashboards

Visualization tools are important for reducing SOC team fatigue by presenting logs and alerts in an intuitive manner, but there is a limited focus exists on customizable dashboards that allow SOC teams to prioritize logs and alerts based on their specific needs or workflows.

Chapter 4

Requirements Specification

4.1 Stakeholders

The following table, Table 8, summarizes Stakeholder Classification and Needs. The table categorizes stakeholders and outlines their specific needs.

Table 8 Stakeholders

Stakeholder	Interaction with the system	Importance
SOC Teams	Ensuring computer systems and networks are secure and running smoothly as intended.	High
IT administrators	Managing, maintaining and configuring the logging architecture.	High
End-users	Indirectly generate events that are logged.	Low
Security Engineers	Use the system's graphical interface to visualize log data and generate reports for security audits and investigations.	High
Auditors	Review logs and system-generated reports.	Medium
Developers	Updating the code, fixing bugs, and introducing new features.	Medium
External vendors	Providing required tools or integrations to implement and maintain the system.	Low

4.2 Platform Requirements

The hardware and software requirements needed for our system are divided into the following:
Hardware:
-Database Server to Store processed logs and analysis.
-Authentication Server to validate requests.
-Monitoring Server to analyse logs and generate alerts.
-Logging Server to process logs.

Software:
-Operating System to host the environment for all the components we will be using.
-Log Collection to collect logs from different sources.
-Log Processing.
-Database to store logs.
-Dashboard GUI to visualize logs and alerts.

4.3 Functional Requirements

The following table, Table 9, presents the main functional requirements of the logging architecture from user authentication to alerting.

Table 9 Functional Requirements

Requirement	Input	Output	Processes	Constraints	Importance
User Authentication	SSO credentials	Access granted or denied...	Validate credentials Verify user roles Grant or deny or access	Authentication must be secure. Failures must result in appropriate error messages.	Essential
Log Collection	Various Data Sources	Raw Log Data	Data Collection from Different Sources	Collecting Logs from Web Servers, Network Devices	Essential
Log Formatting	Raw Log Data	Structured Logs	Normalizing and structuring logs for consistency	Formatting must ensure compatibility with downstream components.	Essential
Log Filtering and Sampling	Raw Log Data	Filtered Log Data	Filters logs to ensures relevance in the data collected	Limiting data collection to important events only.	Essential
Alerting	Processed Logs	Alert Notifications	Log analysis and alert generation	Sending Alerts for critical errors and major security events	Essential
Log Correlation	Logs from Multiple Sources	Correlated Logs	Correlating Logs Based on Event Patterns	Identifying and Correlating Related Events in Logs	Essential
Reporting	Processed Logs	Dashboard insights	Send logs to dashboards and GUIs for visualization	Dashboard must present logs and alerts in a user-friendly manner	Essential
Log Categorization	Structured Logs	Categorized Logs	Organize logs into categories (e.g., performance, security, operations)	Categorization must align with organizational priorities.	Essential
Log Processing	Raw or Categorized Logs	Processed Logs	Aggregate logs, normalize for consistency, and store in the database	Processing must prepare logs for analysis and alert generation	Essential

For the log collection
To collect logs from various sources and output raw log data, we need a robust and scalable solution. By that we mean the support for multiple data sources (e.g.,

systems logs, application logs), and ensure its capability to collect logs in their original format while also ensuring that no logs are missed during collection.
The log sources we chose to collect from are:
1) Application Logs
Application logs are important since that IT teams use them for various reasons such as investigating outages, or troubleshooting bugs and in general knowing the root cause of incidents. These logs include important information (e.g., user activities, error messages). Therefore, monitoring application logs helps in identifying crucial issues like unauthorized access attempts, and application
errors.
2) System Logs
System logs record events of the operating system. They contain details regarding user logins and logouts, system failures and other procedures of the system.
After determining the data sources to collect from, we deploy agents on each data source to collect logs and forward them to our database for storage. We can use Fluent Bit, Graylog Sidecar, or the built-in agent on Unix/Linux systems (Syslog).

4.4 Non-Functional Requirements

Table 10 Nonfunctional Requirements

Non-Functional Requirement	Example
Performance	The system’s response to users should be on time.
Storage Limits	The system should be able to store at least 6 months' worth of logs within.
Code Quality	The source code must align with the industry-standard coding standards and is subject to regular code reviews.
Documentation	Comprehensive documentation which includes installation manuals is provided.
Accessibility	The interface must align with specific standards to ensure ease of accessibility for users with special needs.
User Interface Ease-of-Use	An interface following best design practices to ensure ease of use for administrators.

4.5 Other Requirements

API Restrictions: The system is required to align with security guidelines and only use authorized APIs for data exchange. Any external API should follow industrial guidelines, and best practices, to provide authentication and smooth data transmission.
Data Transmission: The system mandatorily prioritizes the use of secure data transmission protocols like HTTPS, allowing communication of multiple components. Additionally, any protocol used for data exchange must be clearly cited along with its implementation through predefined standards.
Data Storage Formats: Any data stored inside the system (including logs and configuration files) is required to follow standardized guidelines, providing ease of fetching, retrieval, and analysis. Data storage formats are to be aligned, and compatible with industry-standard data storing, processing, and reporting tools.
Compliance with Regulations: The system should be fully compliant with regulations, avoiding any failures in following up with any ongoing updates in worldwide regulations.

Backup and Recovery: The system must put in force ordinary backup procedures to prevent information loss. A robust recuperation plan needs to be placed in the area, specifying techniques for restoring devices to a useful nation after a disruption.
User Authentication Standards: User authentication mechanisms must follow industrial requirements. Multi-element authentication is considered one of the better practices to achieve personal account security.
System Performance Monitoring: Continuous tracking and overseeing tools should be utilized to evaluate and compare the system's performance in real-time and overall performance thresholds and signals must be configured to notify administrators of any system failures. [11]

Chapter 5

System Design

5.1 Architectural Design

The following figure represents the Architectural design of our logging architecture. The logs from the DMZ Application Server and from the LAN zone are passed to a switch which forwards them to the logging server. After arriving at the logging server, they are moved to the monitoring server which checks the performance of the logs and ensures that logs are received whole and uncorrupted. The results of the monitoring server are passed to a dashboard that displays the results. The logging server is connected to a database which stores the structured logs and a file directory which stores the unstructured data. The database passes the logs to an IDS to detect any anomalies in the logs received.

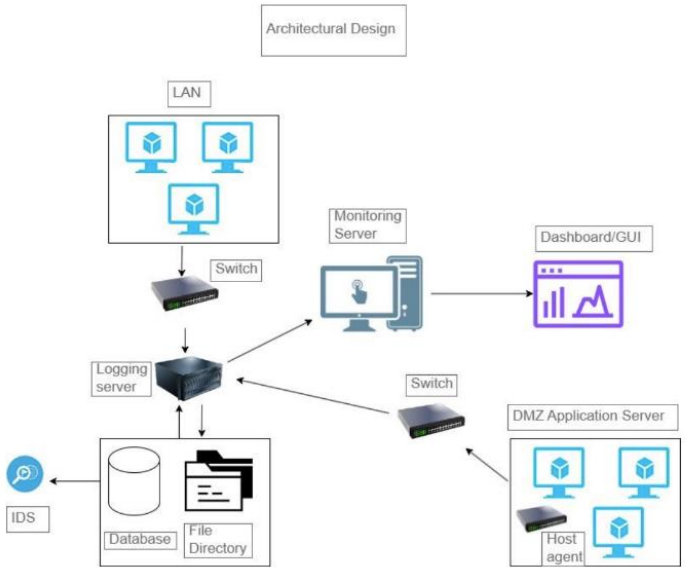


Figure 10 Architectural Design

5.2 Logical Model Design

The architecture represents a log collection, monitoring, and dashboarding system. In the Logical design, we'll describe the functionalities of every component of the architectural design.

1. LAN: The LAN connects various client machines to the logging and monitoring infrastructure. Client machines generate application logs, which are transmitted to the logging server via the LAN.

2.Host Agent: The host agent runs on each monitored machine (client). It collects logs, system metrics, and security events then sends this data to the Logging Server for storage and processing.
3. Switches: Ensures data flows correctly and isolates traffic for different components.

4. Logging Server: is a centralized log collection and storage that receives logs from client machines and host agents then stores logs in a structured format in the Database and File Directory. It passes relevant information to the Monitoring Server for analysis and alerting.
5. Database: Stores log information in an organized and query able format for analysis and supports indexing for fast search and retrieval of logs.
6. File Directory: Stores raw log files and acts as a backup and reference system for logs in case of database failure.
7. IDS: Monitors logs and network traffic to detect potential security threats or unauthorized activities.
8. Monitoring Server: Processes logs and alerts for real-time monitoring, ensures hat logs are received completely and uncorrupted and shows how many logs were generated from each device. It also Interfaces with the Dashboard/GUI to display monitoring results.38
9. Dashboard/GUI: It identifies if the alerts are critical or not and displays real- time data visualizations, charts, and summaries of collected logs and alerts.

10. DMZ Application Server : Hosts applications that process logs in the DMZ and receives specific log events forwarded from the monitoring servers.

5.2.2 Use case Diagram

The following figure represents the use case diagram which shows how a user interacts with the system. The figure includes three main stakeholders; network engineers, security engineers, and administrators.

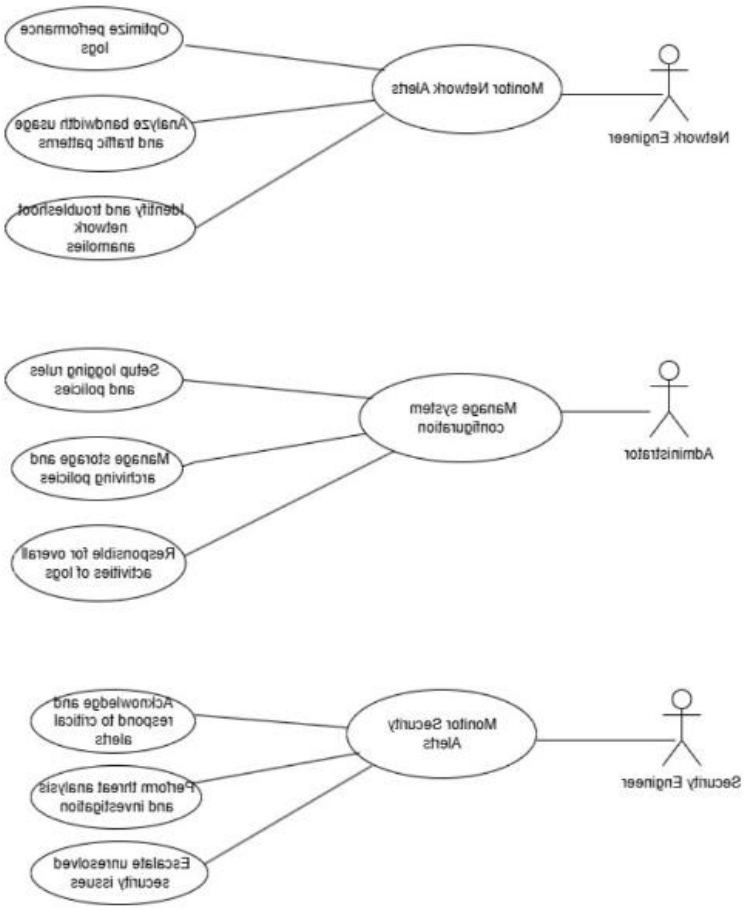


Figure 11 Use Case Diagram

5.2.3 Sequence Diagram

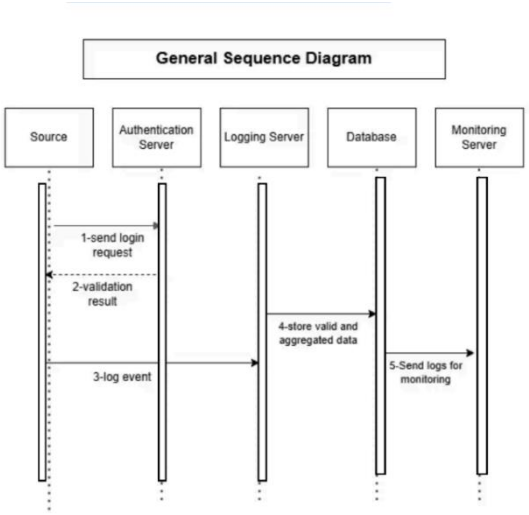


Figure 12 Sequence Diagram

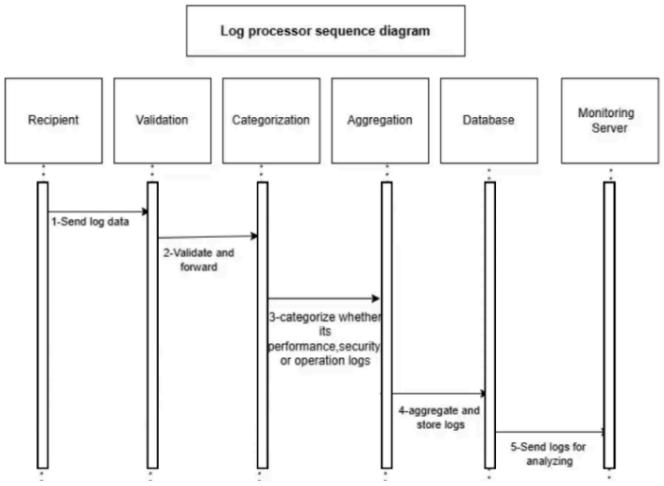


Figure 13 Log Processing Sequence Diagram

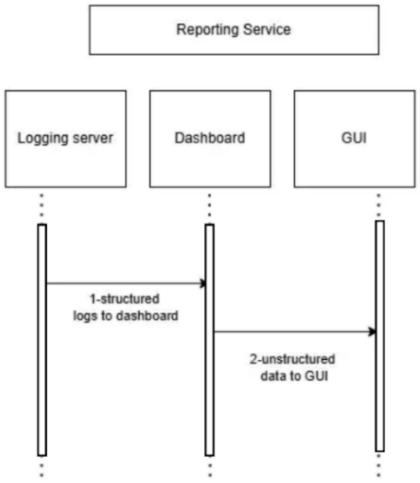


Figure 14 Reporting Service Sequence Diagram

Physical Model Design

Database Design

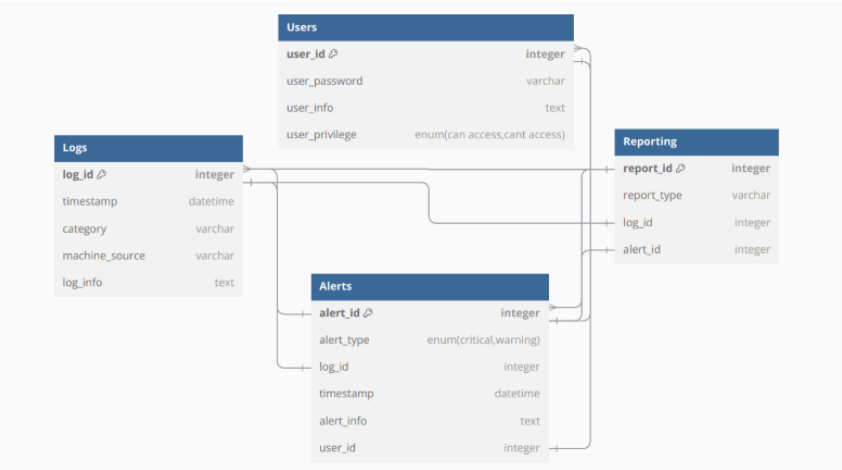


Figure 15 Database Design

Dashboard/GUI

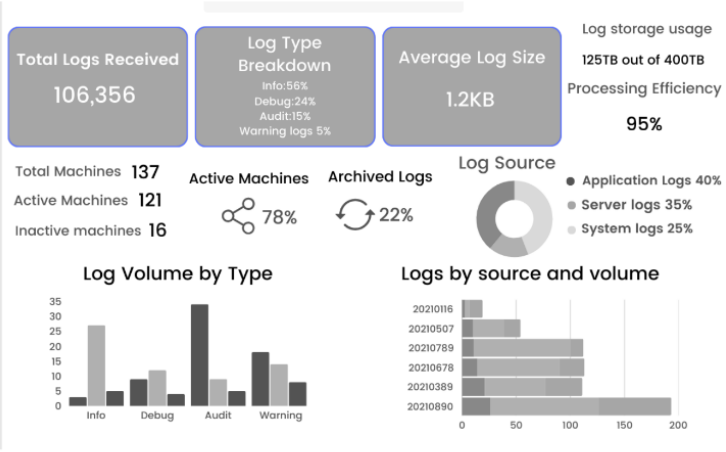


Figure 16 Dashboard

Chapter 6

Implementation:

6.1 General Implementation Description

The system is designed as a centralized log collection and IDS with instant visualization. It combines multiple tools to collect, detect, forward, store and help visualize system, kernel, and authentication logs and alerts related to security.
The implementation prioritizes lightweight design for effective operation across different servers.
Programming Languages, Tools, and APIs Used:
Python was the main programming language for writing log extraction, forwarding, and API handling scripts.
Flask (Python) was used to create at REST API (log_receiver.py) on the central server. It receives incoming logs via HTTP POST requests from the decentralized servers.
Suricata, an IDS and was chosen for its stability and clean alert logging format (fast.log).
InfluxDB used as a database to store collected logs and alerts. It was chosen for its compatibility with Grafana.
Grafana provides a real-time visualization dashboard. It is connected to Influx DB and uses Influx QL queries to display logs, alerts, and customized panels.
Command-line tools like Hydra (for SSH brute-force), Nmap (for port scanning), Hping3 (for DoS simulation), and curl (for web requests) were used to simulate multiple attacks.
Bash was used for starting services, checking interfaces, and managing permissions.
Codebase Overview
Number of Python scripts: 6 core scripts
-extract_authlogs.py, extract_syslogs.py, extract_kernellogs.py – for log collection
-send_authlogs.py – sends logs to the central server as JSON
-log_receiver.py – Flask-based API on central server to receive logs
-forward_suricata_alerts.py – monitors Suricata alerts and forwards them to the central server

API endpoints: One POST endpoint (`/receive_logs`) used by all client scripts to submit data.

Database interactions: All data is written to Influx DB and visualized in Grafana using Influx QL queries. Database interaction occurs via Grafana's query interface.
Grafana dashboards: Several panels were implemented like time-series graphs, data tables, and alert counters, all coming from Influx DB queries.
Coding Conventions
snake_case for variables and functions
space indentation
Logical structure with functions and reusable components
Log messages and JSON data are formatted across scripts to ensure clean integration with InfluxDB and Grafana

Filenames clearly reflect script function (e.g., `forward_suricata_alerts.py`)
The implementation reflects a clear separation of concerns: log collection, attack detection, data transport, storage, and visualization are handled by independent, well-named modules.

6.2 Pipeline Implementation Description

The pipeline we used follows a flexible and structured way to gather, transfer, detect, store, and visualize logs related to security events instantly.
It contains four servers -three for log generation and one central server for grouping, detecting, and visualizing logs connected with an IDS, forming an ideal lightweight security monitoring system.

Main Stages of the Pipeline:

- **Log Collection**: Every server of the system, kernel, and authentication runs a script to extract attributes from raw logs and formats and stores them in a structured JSON format.
- **Log Transmission**: Python requests library is used by client-side to send the JSON data using HTTP POST to the central server which it makes it network based and avoids file transfer and dependency in shared storage.
- **Central Server**: A lightweight API (log_receiver.py) receives incoming logs using the /receive logs POST endpoint. Logs are validated and directly sent to the database for storage in (log_data) which contains metadata like timestamp and type of log. The central server acts as the recipient for logs collected.
- **Intrusion Detection System Integration:** Suricata runs on the authentication server to track traffic and generate alerts.
- **Python Scripts**:(forward_suricate_alerts.py) tails this log file and forwards the alerts to the central server using the same POST endpoint. Alerts are tagged with their special source field and stored in the database with other logs.
- **Visualization**: Grafana accesses the database content using InfluxQL. Different panels are used to visualize log contents,alerts, and IDS specific messages. Grafana also supports filtering alerts based on log content.

Tools and Technologies Used:

- **Grafana**: used for visualizing real-time log alerts and data.
- **Suricata**: used for monitoring traffic and generates alerts in a structured format on authentication server.
- **Flask**: used for allowing the central server API to receive log data.
- **InfluxDB**: database used to store alerts and logs in time-series format.
- **Python**: used for writing the scripts of log processing, forwarding, and API interactions.

6.3 Additional Implementation Details

Data Pipeline Structure (Code References & Description)
The architecture of the system follows a modular data pipeline:
1. Log Extraction Scripts (on edge servers)

Python scripts (extract_authlogs.py, extract_syslogs.py, extract_kernellogs.py) were written to extract logs from system files and write structured JSON objects into authlogs_output.json, kernel_logs_output.json, and syslogs_output.json.

Extract_authlogs.py script :

```
import json
import os
import re
import time

# Define the output JSON file with a timestamp-based name
output_file = f"authlogs_output.json"

# Path to authentication logs
auth_log_path = "/var/log/auth.log"

# Check if the authentication log file exists
if not os.path.isfile(auth_log_path):
    print("Authentication log file not found!")
    exit(1)

logs = []

# Regex pattern to extract IP addresses
ip_pattern = re.compile(r'\b(?:\d{1,3}\.){3}\d{1,3}\b')

# Function to calculate anomaly score (basic heuristic)
def calculate_anomaly_score(log_message, contains_special_chars):
    score = len(log_message) / 100  # Normalize length (longer logs might be suspicious)
    if contains_special_chars:
        score += 1  # Increase score if special characters exist
    return round(score, 2)  # Keep it to 2 decimal places

# Read the authentication log file line by line
with open(auth_log_path, "r") as file:
    for line in file:
        parts = line.strip().split()
        if len(parts) >= 5:  # Ensure there's enough data in the log entry
            timestamp = " ".join(parts[:3])  # First three fields as timestamp
            log_message = " ".join(parts[4:])  # Everything after the hostname is the log message

            # Extract source IP if present
            source_ip = None
            match = ip_pattern.search(line)
            if match:
```

Figure 17 authlogs.py 1

```
                source_ip = match.group(0)

            # Check for special characters
            # Improved regex: Exclude common log symbols like ':', '[', ']', '(', ')'
            contains_special_chars = bool(re.search(r'[{}<>$&^*@!]', log_message))

            # Calculate log length
            log_length = len(log_message)

            # Compute anomaly score
            anomaly_score = calculate_anomaly_score(log_message, contains_special_chars)

            # Store extracted data
            logs.append({
                "timestamp": timestamp,
                "source_ip": source_ip if source_ip else "N/A",
                "log_message": log_message,
                "contains_special_chars": contains_special_chars,
                "log_length": log_length,
                "anomaly_score": anomaly_score
            })

# Write the extracted logs to a JSON file
with open(output_file, "w") as json_file:
    json.dump(logs, json_file, indent=4)

print(f"Logs have been saved to {output_file}")
```

Figure 18 authlogs.py 2

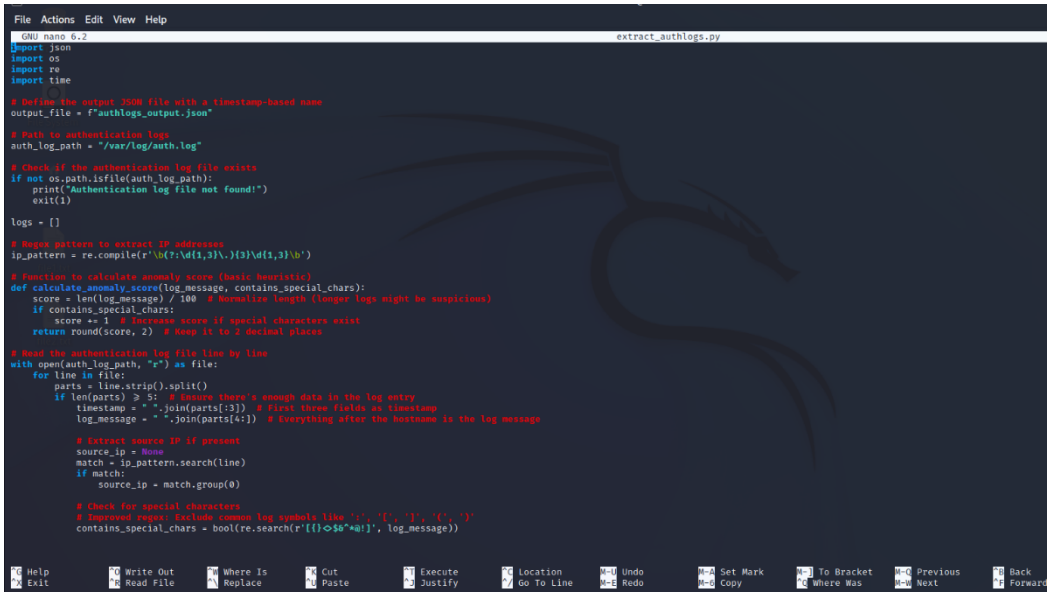


Figure 19 Authlogs.py script

Figure 20 authentication logs output json format

Extract_syslogs.py script:

```
Extract_syslogs.py script:
import json
import os
import re
import time
import hashlib

# Define the output JSON file with a timestamp-based name
output_file = f"syslogs_output.json"

# Path to system logs (change to the correct log file, e.g., /var/log/syslog)
sys_log_path = "/var/log/syslog"  # Or you could use "/var/log/messages"

# Check if the system log file exists
if not os.path.isfile(sys_log_path):
    print("System log file not found!")
    exit(1)

logs = []

# Regex pattern to extract IP addresses (for source IP extraction)
ip_pattern = re.compile(r'\b(?:\d{1,3}\.){3}\d{1,3}\b')

# Function to calculate anomaly score (basic heuristic)
def calculate_anomaly_score(log_message, contains_special_chars):
    score = len(log_message) / 100  # Normalize length (longer logs might be suspicious)
    if contains_special_chars:
        score += 1  # Increase score if special characters exist
    return round(score, 2)  # Keep it to 2 decimal places

# Function to compute log hash (SHA-256)
def compute_log_hash(log_message):
    return hashlib.sha256(log_message.encode('utf-8')).hexdigest()

# Read the system log file line by line
with open(sys_log_path, "r") as file:
    for line in file:
        parts = line.strip().split()
        if len(parts) >= 5:  # Ensure there's enough data in the log entry
            timestamp = " ".join(parts[:3])  # First three fields as timestamp
```

Figure 21 syslogs.py 1

```
# Read the system log file line by line
with open(sys_log_path, "r") as file:
    for line in file:
        parts = line.strip().split()
        if len(parts) >= 5:  # Ensure there's enough data in the log entry
            timestamp = " ".join(parts[:3])  # First three fields as timestamp
            log_message = " ".join(parts[4:])  # Everything after the hostname is the log message

            # Extract source IP if present
            source_ip = None
            match = ip_pattern.search(line)
            if match:
                source_ip = match.group(0)

# Check for special characters in the log message
            contains_special_chars = bool(re.search(r'[{}<>$&^*@!]', log_message))

            # Calculate log length
            log_length = len(log_message)

            # Compute anomaly score
            anomaly_score = calculate_anomaly_score(log_message, contains_special_chars)

            # Compute log hash (SHA-256)
            log_hash = compute_log_hash(log_message)

            # Store extracted data
            logs.append({
                "timestamp": timestamp,
                "source_ip": source_ip if source_ip else "N/A",
                "log_hash": log_hash,
                "log_length": log_length,
                "anomaly_score": anomaly_score,
                "log_message": log_message
            })

# Write the extracted logs to a JSON file
with open(output_file, "w") as json_file:
    json.dump(logs, json_file, indent=4)
```

Figure 22 syslogs.py 2

Figure 23 Extract_syslogs.py script

Extract_kernellogs.py script:

```
import json
import os
import re
import time

# Define the output JSON file with a timestamp-based name
output_file = f"kernel_logs_output.json"

# Path to kernel log
kern_log_path = "/var/log/kern.log"

# Check if the kernel log file exists
if not os.path.isfile(kern_log_path):
    print("Kernel log file not found!")
    exit(1)

logs = []

# Regex pattern to extract error codes from kernel logs (assuming the pattern in logs)
error_code_pattern = re.compile(r"0x[0-9A-Fa-f]+")  # Example pattern for hexadecimal error codes

# Function to extract error codes from messages
def extract_error_code(message):
    match = error_code_pattern.search(message)
    return match.group(0) if match else None

# Function to extract system call failures (basic heuristic)
def extract_system_call_failures(line):
    if "syscall" in line.lower() and "failed" in line.lower():
        return True
    return False

# Read the kernel log file line by line
with open(kern_log_path, "r") as file:
    for line in file:
        # Extract timestamp (assuming format: "MMM DD HH:MM:SS")
        parts = line.split(" ", 3)  # Split into 4 parts: month, day, time, and message
        if len(parts) < 4:
            continue  # Skip lines without a valid timestamp
```

Figure 24 kernellogs.py 1

```
        timestamp = " ".join(parts[:3])  # Join the month, day, and time as the timestamp
        kernel_message = parts[3].strip()  # The rest is the kernel message

        # Extract error code
        error_code = extract_error_code(kernel_message)

        # Check if there are system call failures
# Check if there are system call failures
        system_call_failure = extract_system_call_failures(kernel_message)

        # Store extracted data
        logs.append({
            "timestamp": timestamp,
            "kernel_message": kernel_message,
            "error_code": error_code if error_code else "N/A",
            "system_call_failure": system_call_failure
        })

# Write the extracted logs to a JSON file
with open(output_file, "w") as json_file:
    json.dump(logs, json_file, indent=4)

print(f"Kernel logs have been saved to {output_file}")
```

Figure 25 kernellogs.py 1

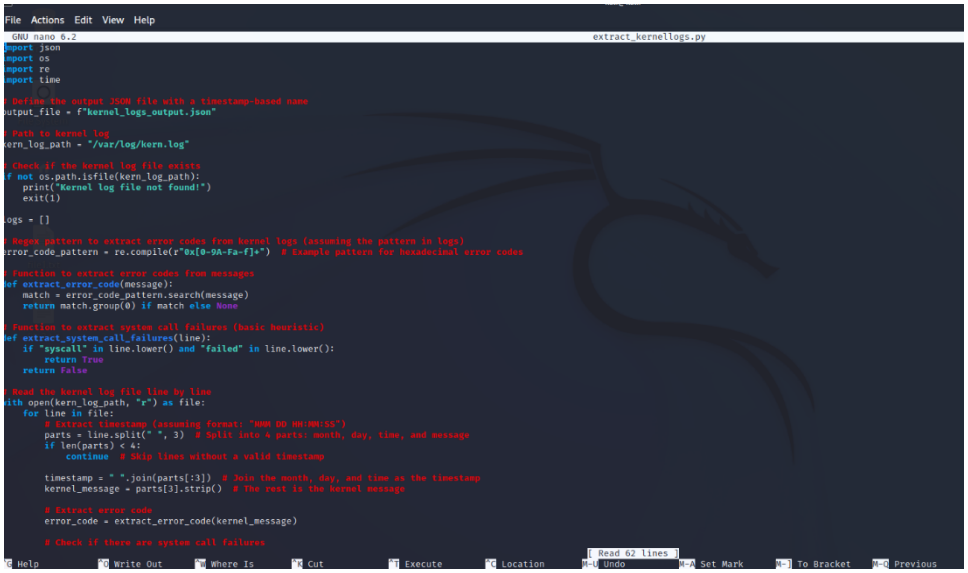


Figure 26 Extract_kernellogs.py script

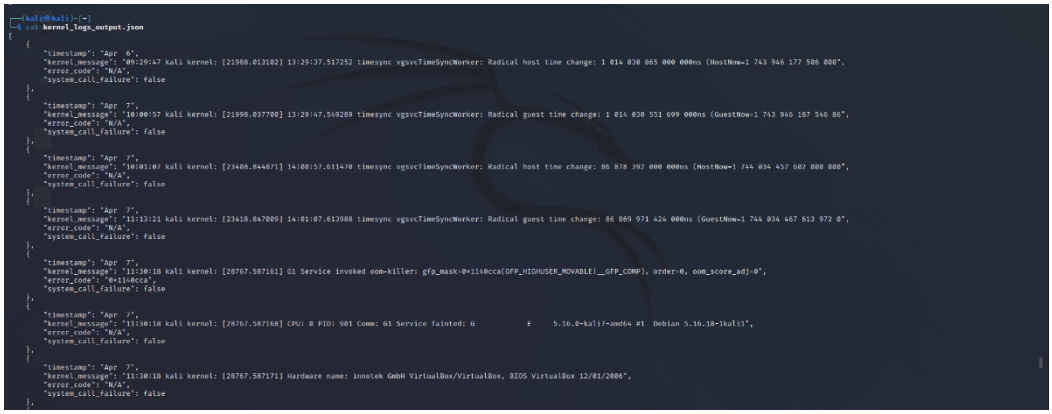


Figure 27 Extracted Kernel Logs

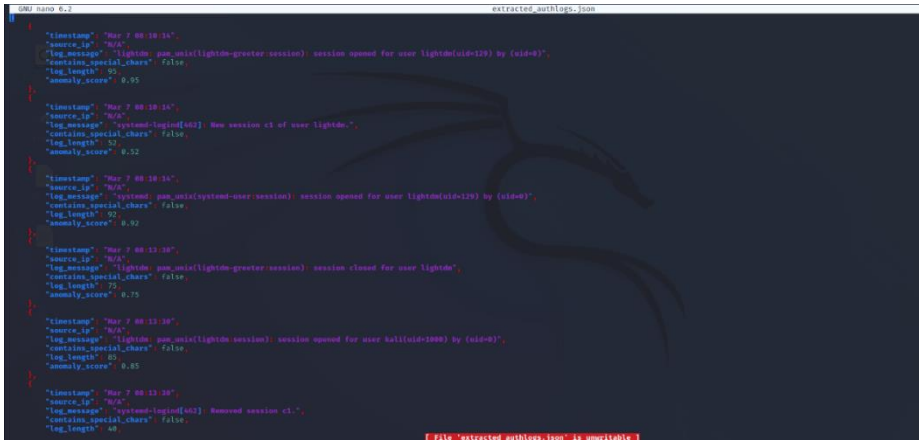


Figure 28 Extracted Kernel logs 2

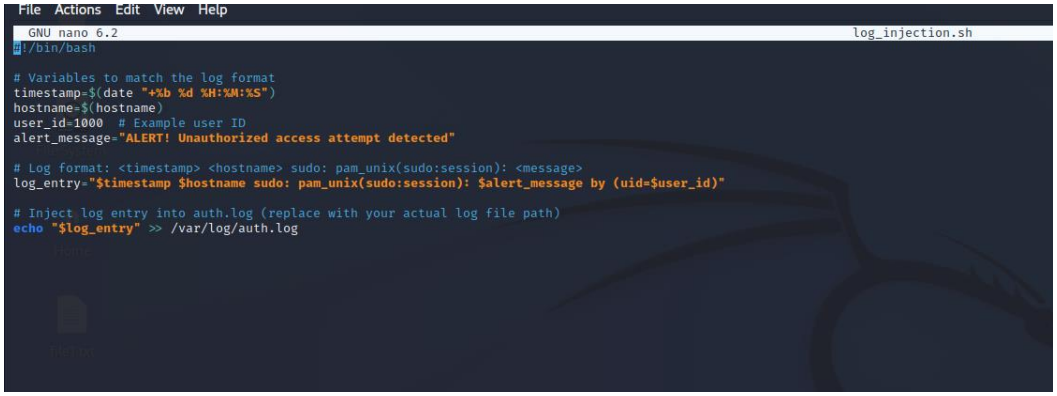


Figure 29 Simulating a log injection on autehentication

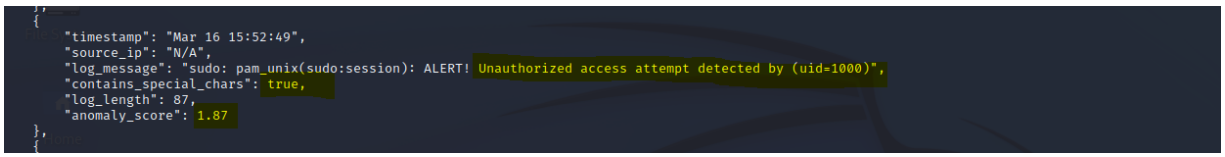


Figure 30 Alert showing

2. Purchasing 4 servers (authentication, system, kernel, centralized)

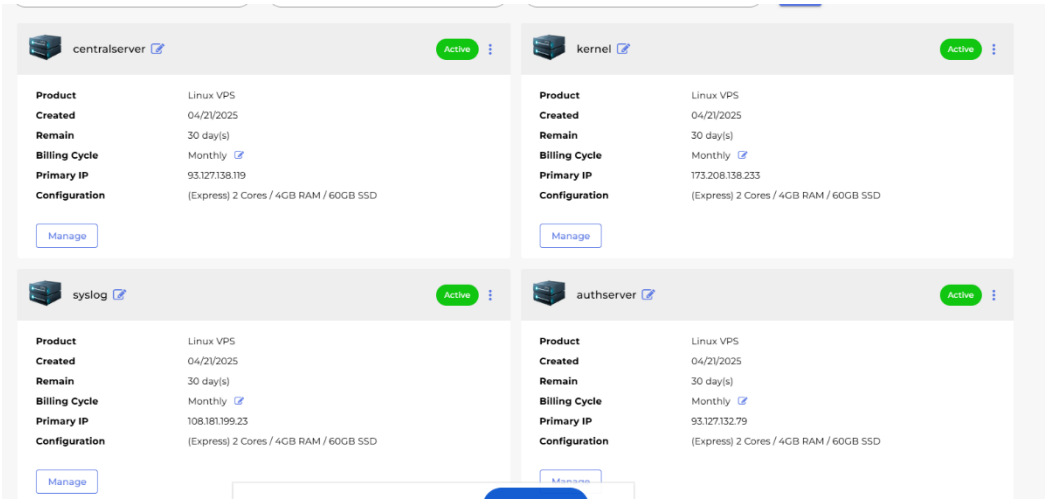


Figure 31 purchased servers

Accessing authentication server (first server) using ssh administrator@93.127.132.79:

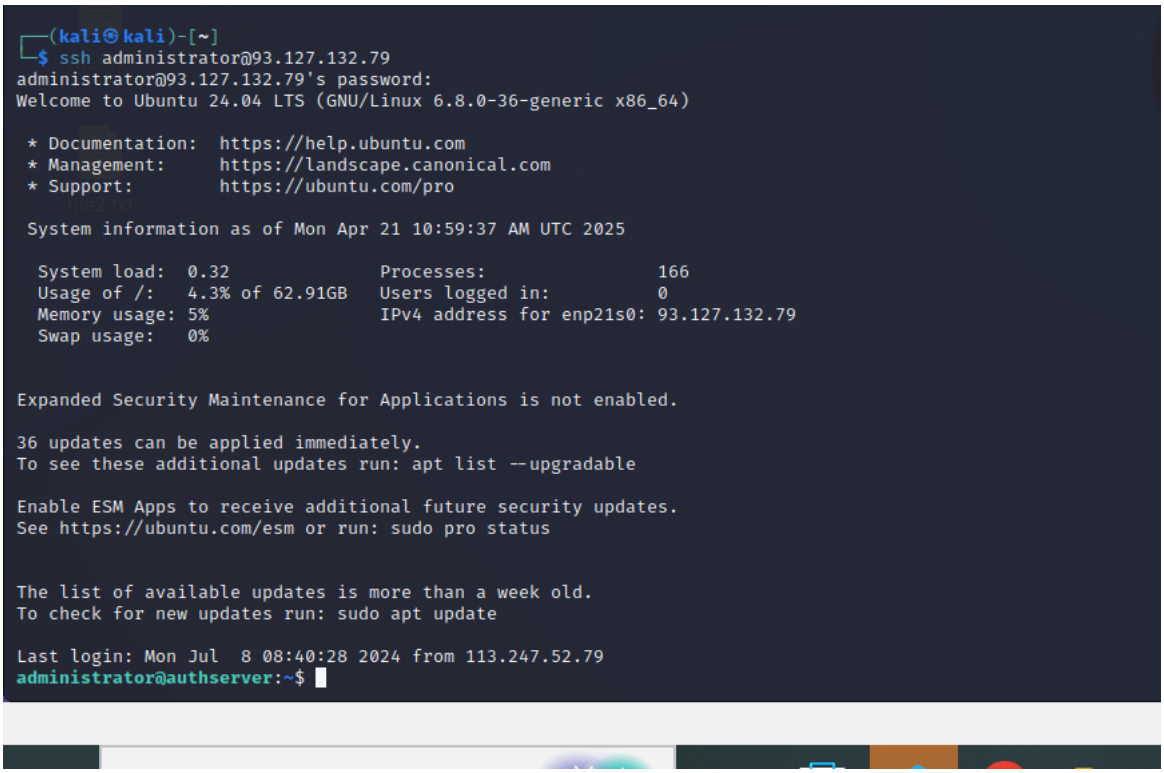


Figure 32 accessing authentication server

Copying the extract_authlogs.py to the authentication server using scp:

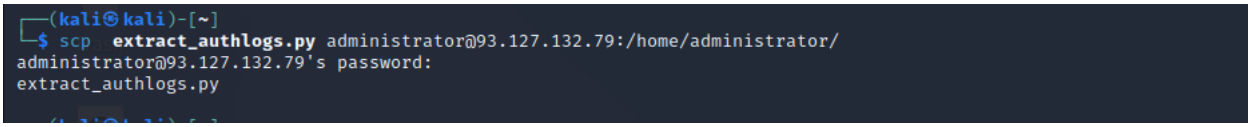


Figure 33 Using scp

Accessing system server (second server) using the same commands (ssh):

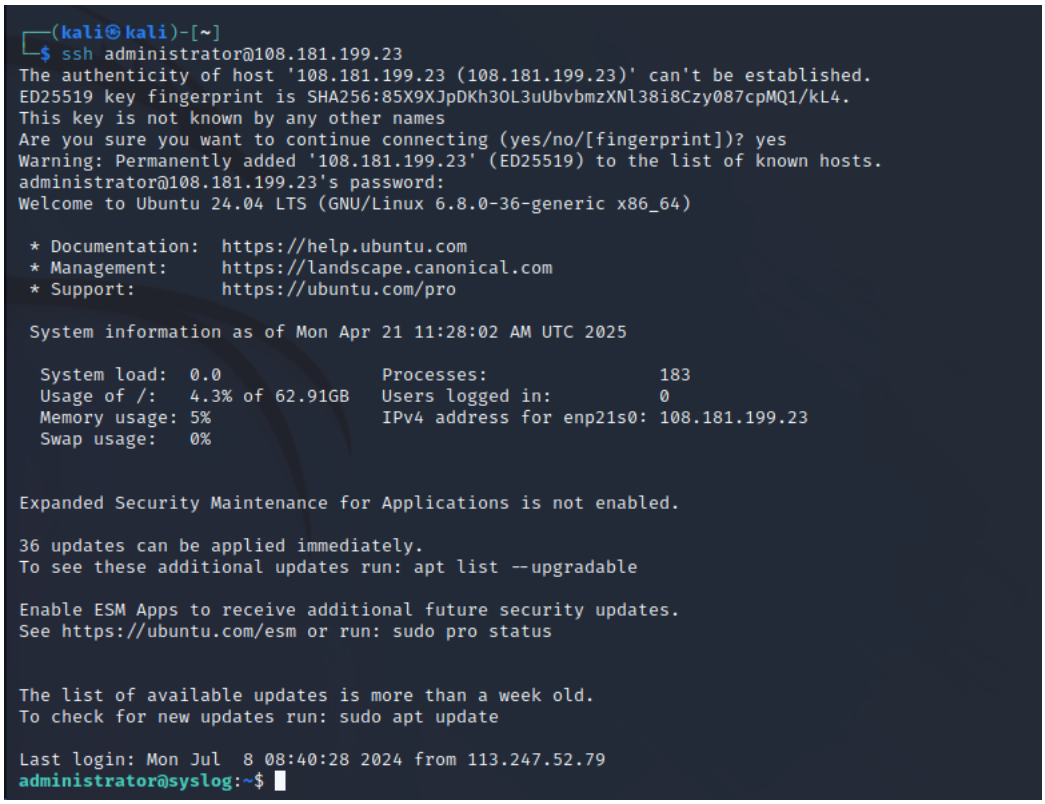


Figure 34 Accessing System Server

Copying extract_syslogs.py to the system server :

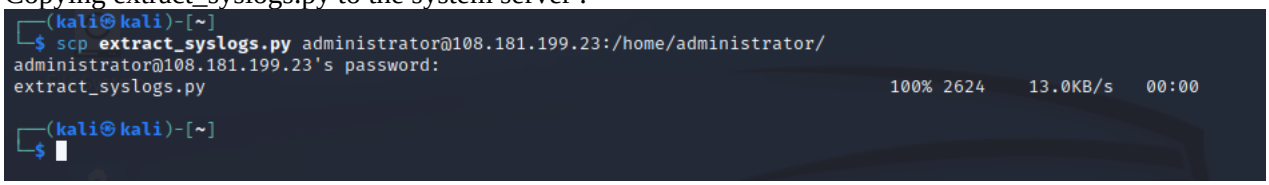


Figure 35 Using scp

Accessing kernel server (third server) :

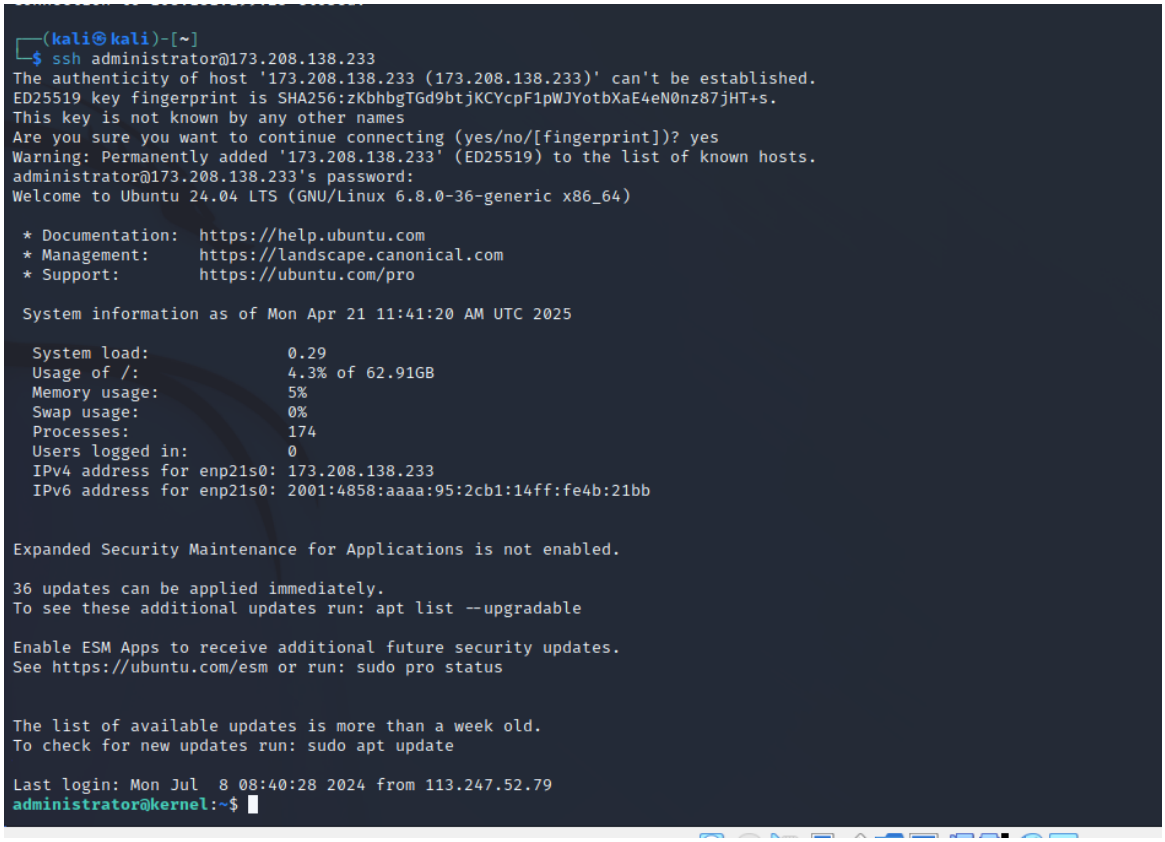


Figure 36 Accessing kernel server

Copying extract_kernellogs.py into the kernel server:

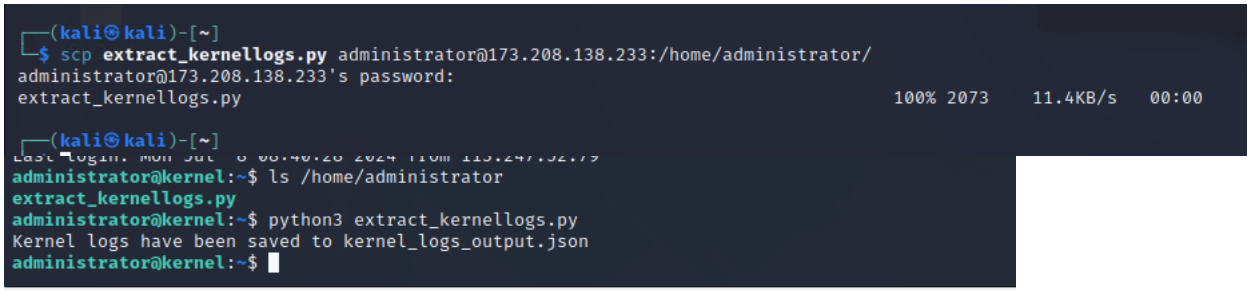


Figure 37 Using scp to copy scripts to server

Acessing the central server (4th server) :

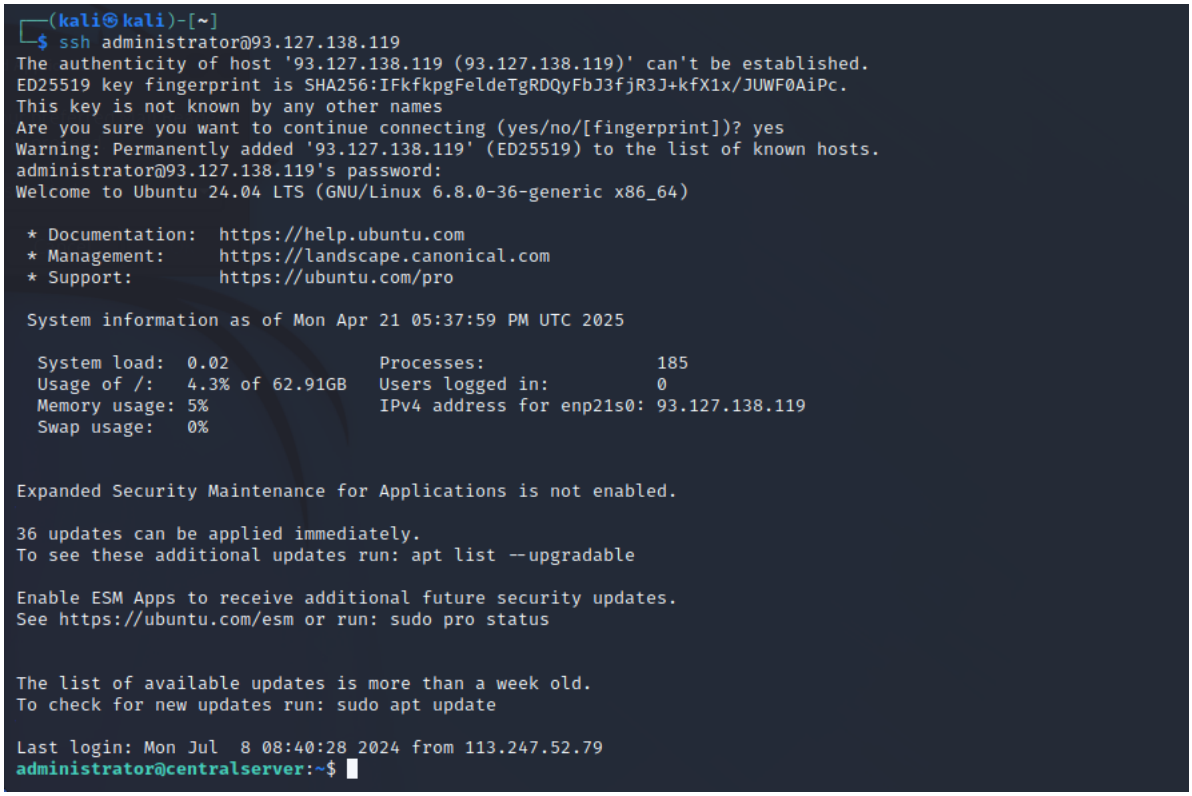


Figure 38 Accessing central server

Installing flask API and connecting influxdb to the central server :

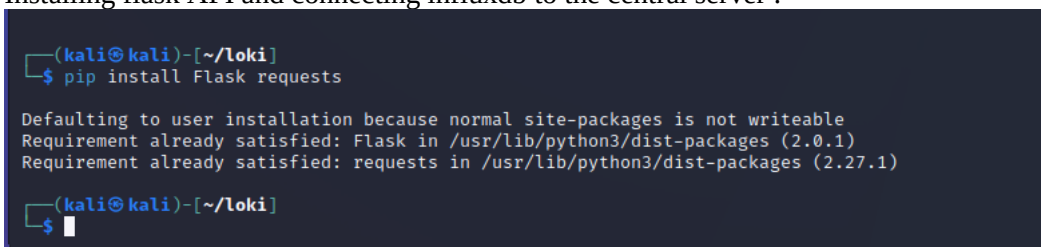


Figure 39 Installing Flask

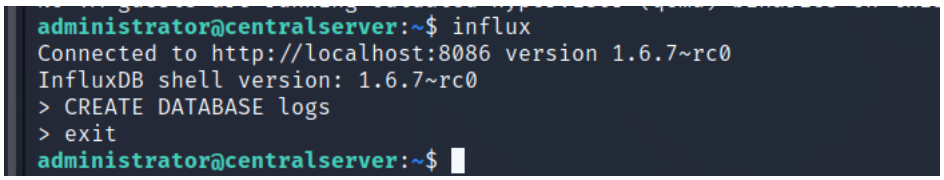


Figure 40 Connecting InfluxDb to central server

3.Log Transmission to Central Server: Each edge server includes a Python sender script (e.g. `send_authlogs.py`) that uses HTTP POST (via the `requests` module) to send logs to the Flask API (`log_receiver.py`) on the central server.

The central server hosts log_receiver.py, a lightweight Flask API that receives logs at /receive_logs, validates them, and stores them into InfluxDB with metadata tags (log type, timestamp, source).

Log_reciever script for the central server:

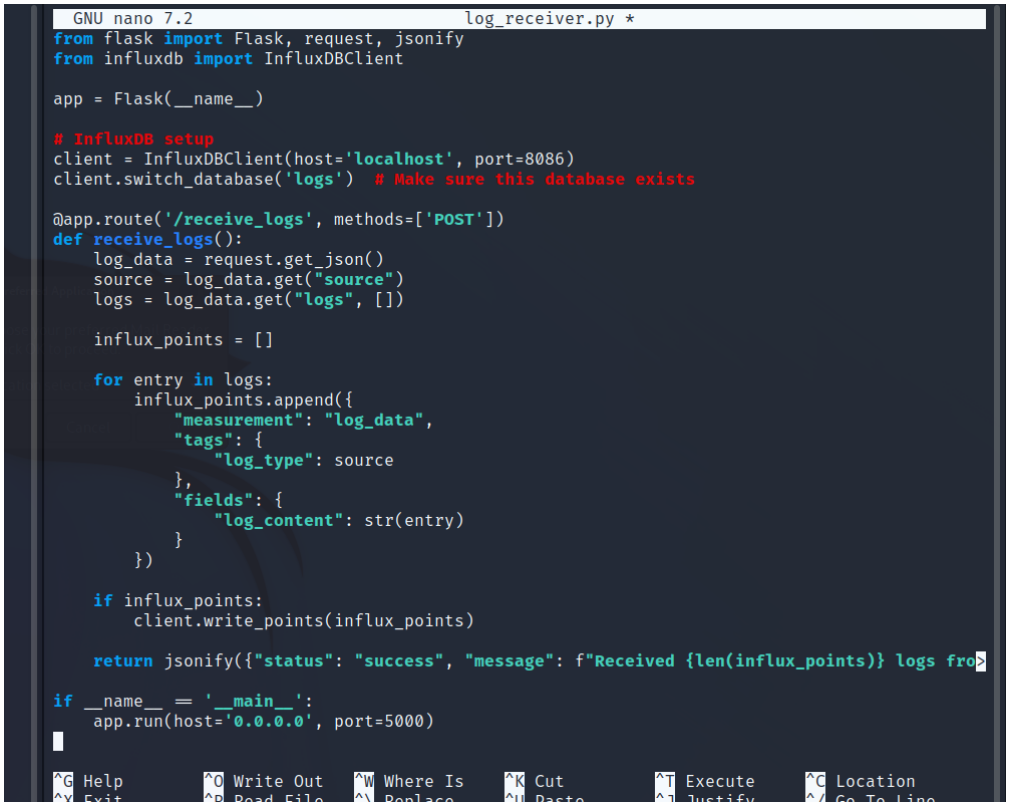


Figure 41 Log receiver script

Script for sending the logs to the central server :

Figure 42 Sending Logs to Central Server

Sending authentication logs to the central server :

Figure 43 Sending logs to central server

Success of sending :

Figure 44 Succesfully sending logs

Sending kernel logs to the central server (using the same script):

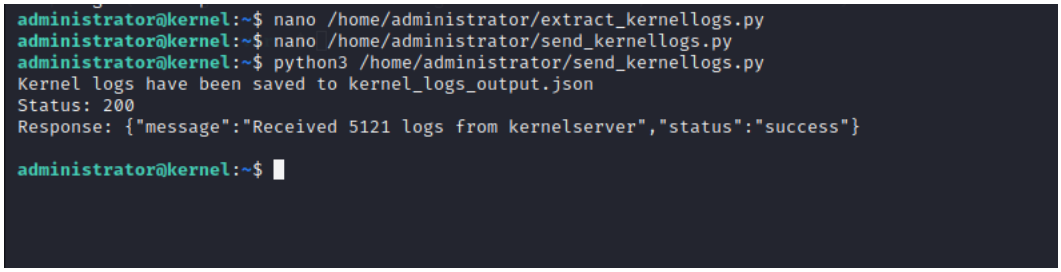


Figure 45 Sending kernel logs to central server

Sending system logs to the central server (using the same script):

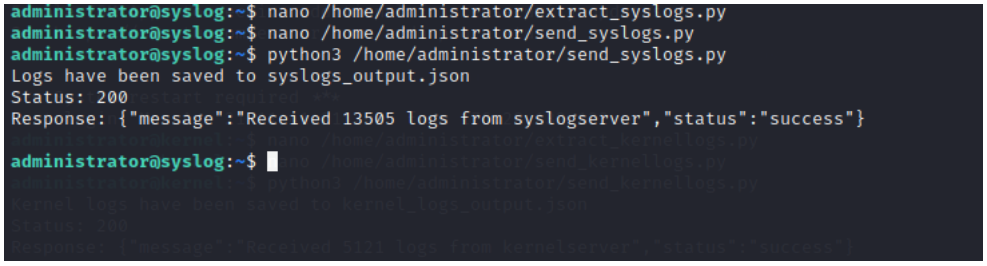


Figure 46 Sending system logs to central server

4. Intrusion Detection and Alert Forwarding:

Suricata runs on the authserver and generates alerts (e.g., brute-force, port scan, DoS) in `fast.log`. These are monitored by `forward_suricata_alerts.py`, which sends real-time alerts to the central server API in the same structured format.

Forwarding suricata alerts script :

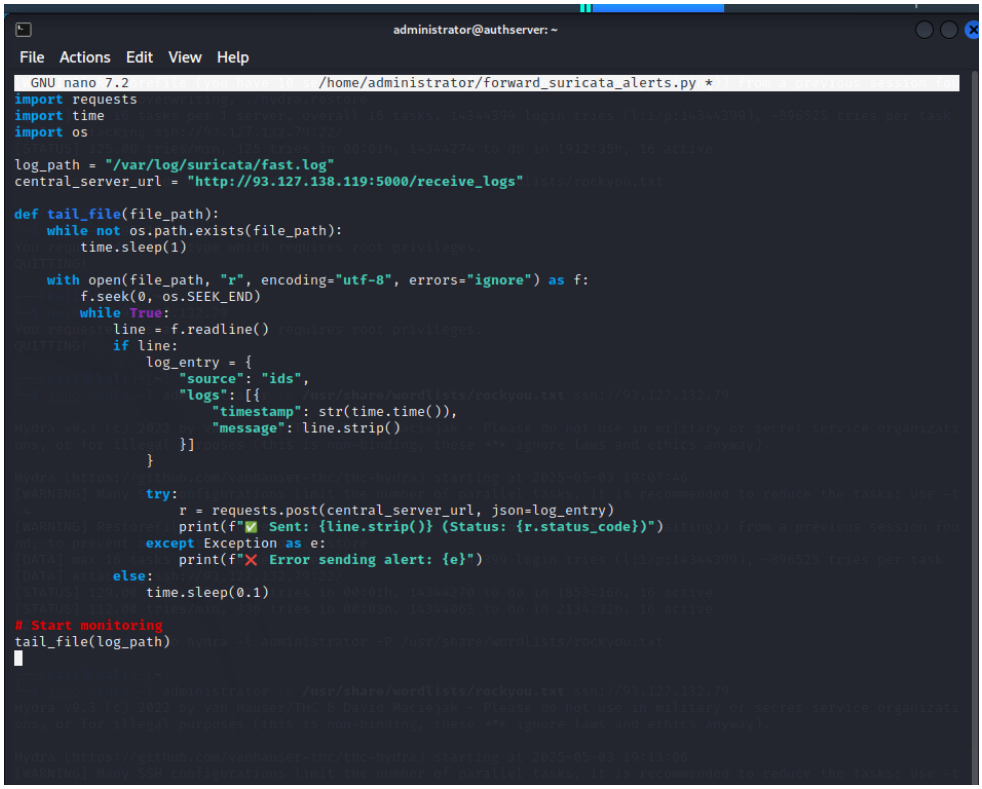


Figure 47 Forwarding Suricata Alerts

Running the script successfully :

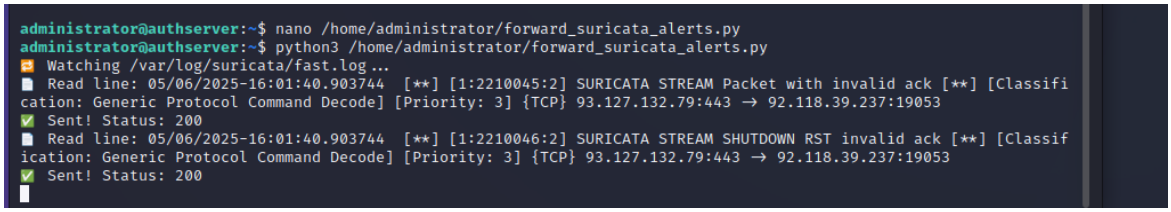


Figure 48 Suricata scripts running successfully

5. Visualization in Grafana

Grafana queries InfluxDB using InfluxQL and visualizes all logs and alerts on live dashboards, including total alert counts, alert types, time-series views, and log detail tables.

Grafana installation (on central server):

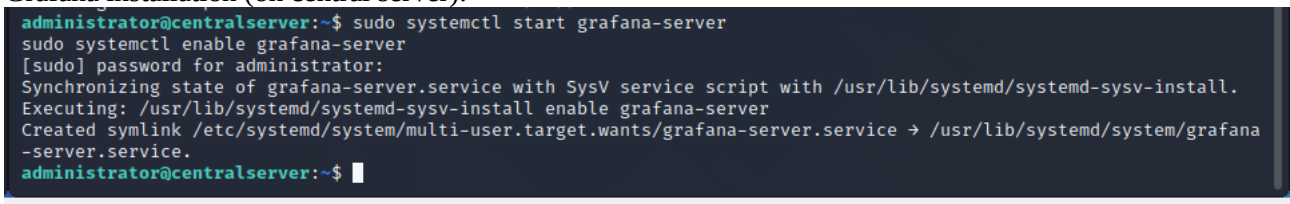


Figure 49 Installing Grafana

Grafana running and being active :

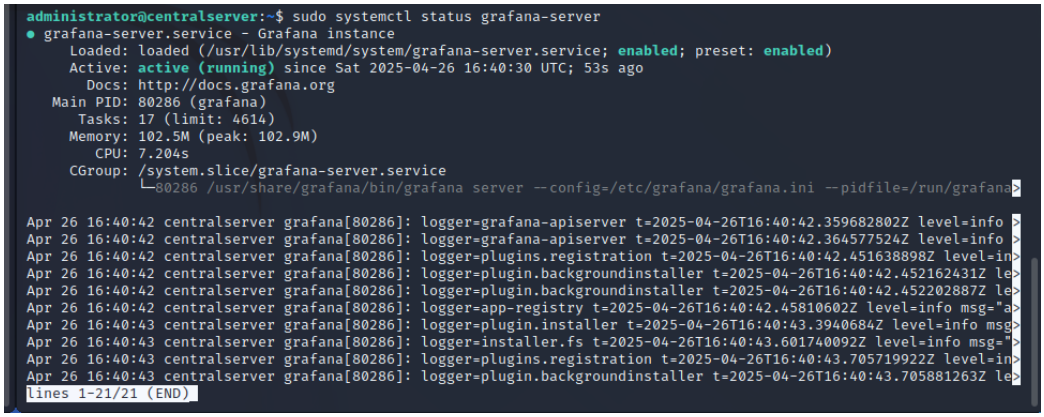


Figure 50 Grafana running

Successfully logging into grafana using ://93.127.138.119:3000/login on kali's firefox and adding panels:
Data source working on grafana:

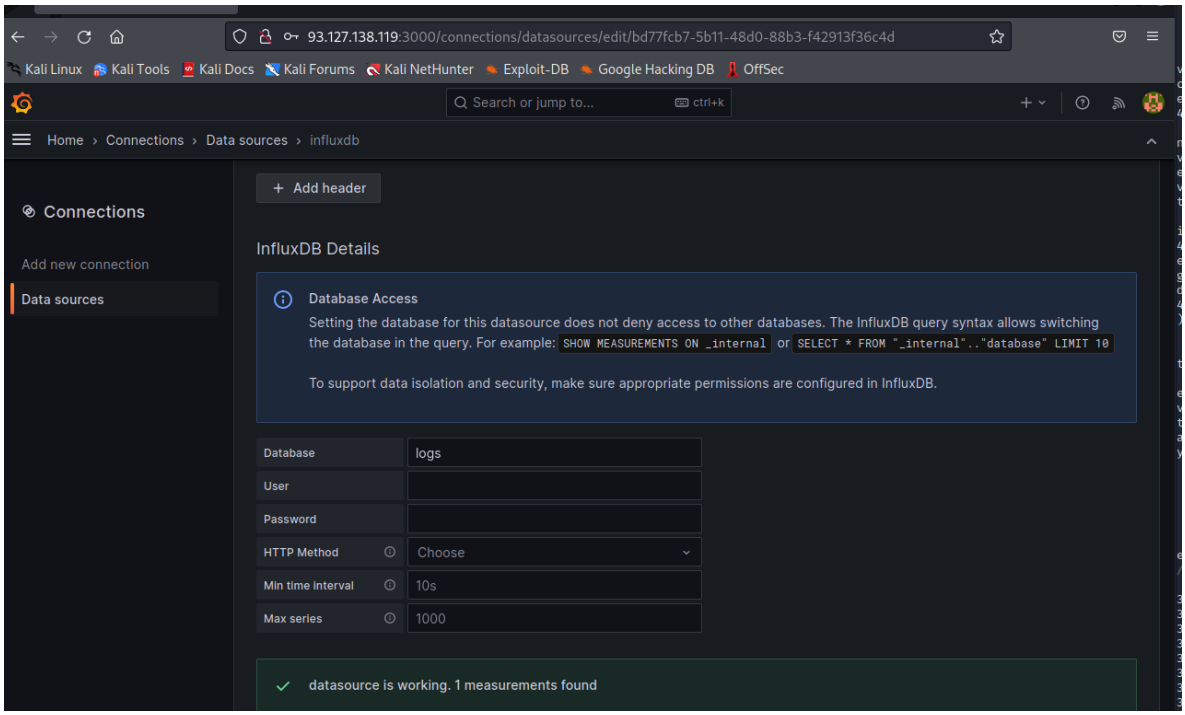


Figure 51 Data source on Grafana

Panels added :

Table 11 Panels

Panel Title	Type	InfluxDB Query
IDS Alerts Over Time	Bar Chart	`SELECT count("log_content") FROM "log_data" WHERE "log_type" = 'ids' AND $timeFilter GROUP BY time(5m)`
IDS Alert Frequency	Time Series	`SELECT count("log_content") FROM "log_data" WHERE "log_type" = 'ids' AND $timeFilter GROUP BY time(5m)`

Total IDS Alerts (Last 6h)	Stat	`SELECT count("log_content") FROM "log_data" WHERE "log_type" = 'ids' AND $timeFilter`
Snort IDS Alerts	Logs	`SELECT "log_content" FROM "log_data" WHERE "log_type" = 'ids' AND $timeFilter`
Logs per Hour	Time Series	`SELECT count("log_content") FROM "log_data" WHERE $timeFilter GROUP BY time(1h)`
Log Type Distribution	Pie Chart	`SELECT count("log_content") FROM "log_data" WHERE $timeFilter GROUP BY "log_type"`
Syslogserver logs	Table	`SELECT "log_content" FROM "log_data" WHERE ("log_type" = 'syslogserver') AND $timeFilter`
Kernelserver logs	Table	`SELECT "log_content" FROM "log_data" WHERE ("log_type" = 'kernelserver') AND $timeFilter`
Authentication Logs	Table	`SELECT "log_content" FROM "log_data" WHERE ("log_type" = 'authserver') AND $timeFilter`
All Logs	Table	`SELECT "log_content" FROM "log_data" WHERE $timeFilter`

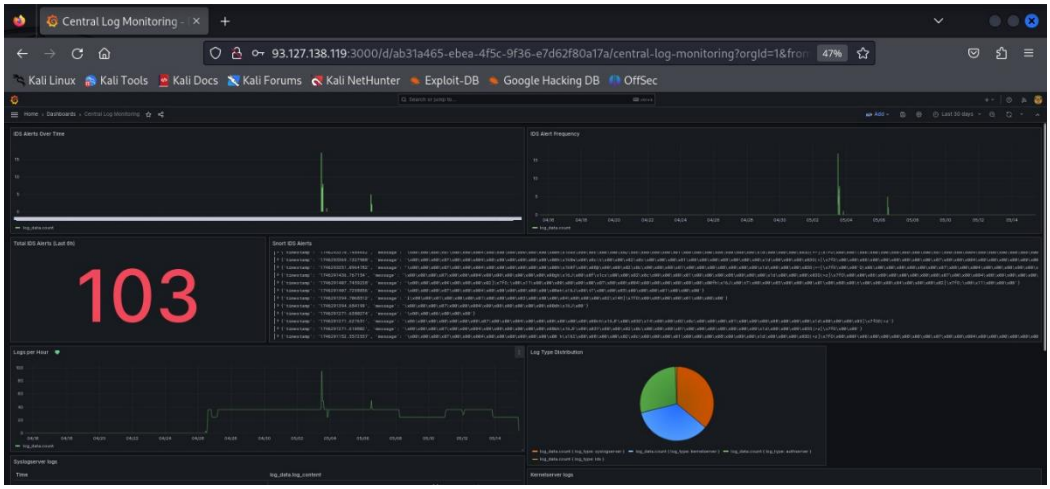


Figure 52 Grafana 1

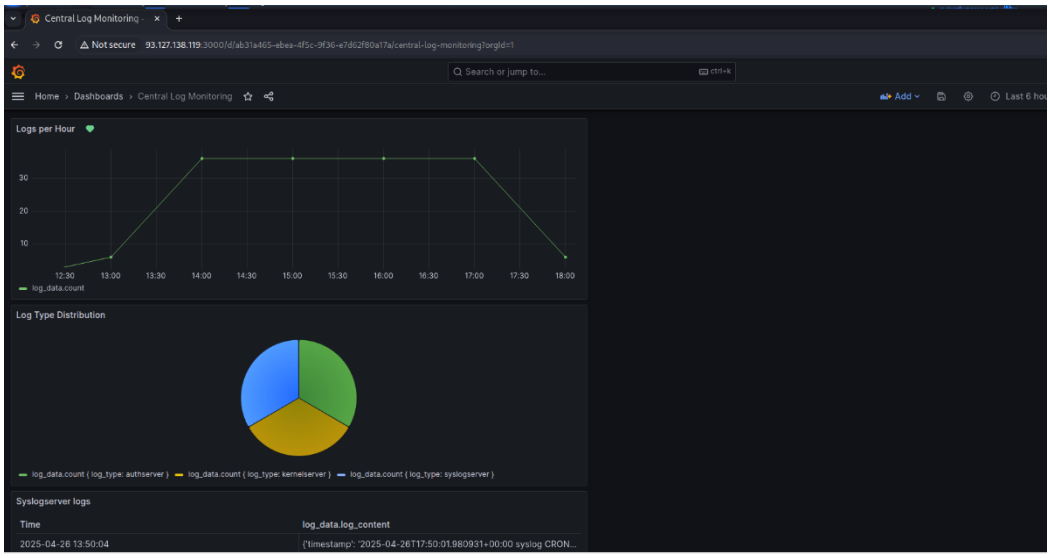
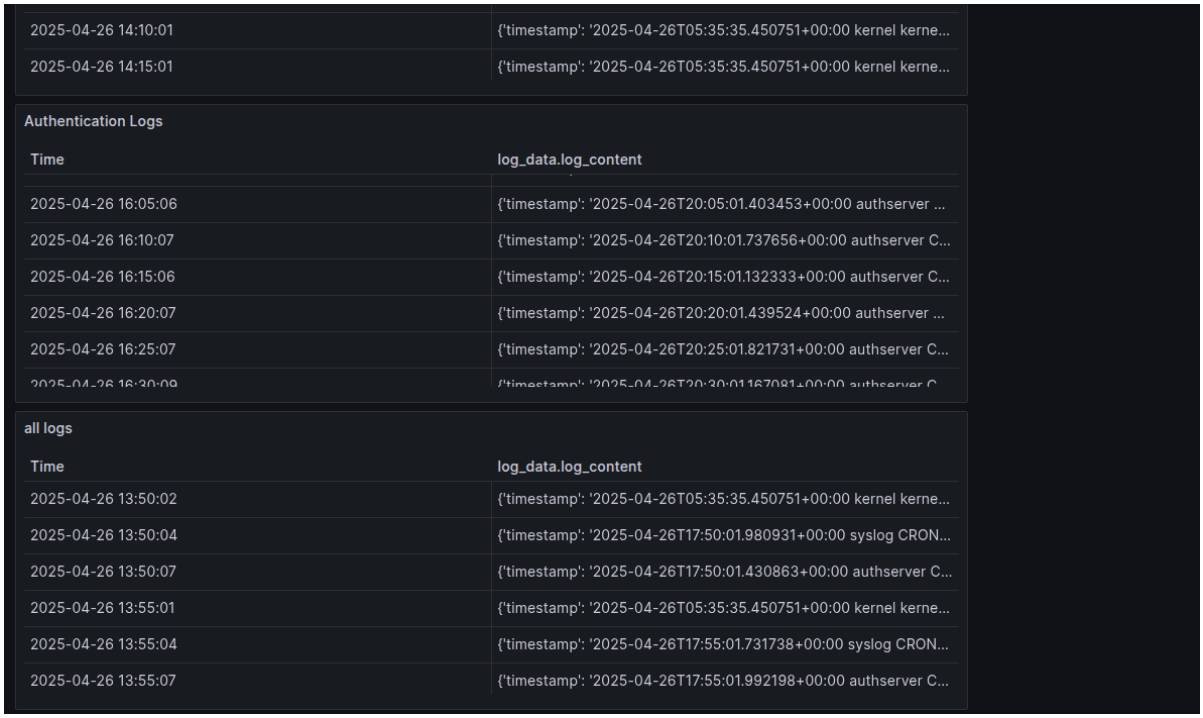


Figure 53 Grafana 2



2025-04-26 14:10:01	{'timestamp': '2025-04-26T05:35:35.450751+00:00 kernel kerne...
2025-04-26 14:15:01	{'timestamp': '2025-04-26T05:35:35.450751+00:00 kernel kerne...

Authentication Logs

Time	log_data.log_content
2025-04-26 16:05:06	{'timestamp': '2025-04-26T20:05:01.403453+00:00 authserver ...
2025-04-26 16:10:07	{'timestamp': '2025-04-26T20:10:01.737656+00:00 authserver C...
2025-04-26 16:15:06	{'timestamp': '2025-04-26T20:15:01.132333+00:00 authserver C...
2025-04-26 16:20:07	{'timestamp': '2025-04-26T20:20:01.439524+00:00 authserver ...
2025-04-26 16:25:07	{'timestamp': '2025-04-26T20:25:01.821731+00:00 authserver C...

all logs

Time	log_data.log_content
2025-04-26 13:50:02	{'timestamp': '2025-04-26T05:35:35.450751+00:00 kernel kerne...
2025-04-26 13:50:04	{'timestamp': '2025-04-26T17:50:01.980931+00:00 syslog CRON...
2025-04-26 13:50:07	{'timestamp': '2025-04-26T17:50:01.430863+00:00 authserver C...
2025-04-26 13:55:01	{'timestamp': '2025-04-26T05:35:35.450751+00:00 kernel kerne...
2025-04-26 13:55:04	{'timestamp': '2025-04-26T17:55:01.731738+00:00 syslog CRON...
2025-04-26 13:55:07	{'timestamp': '2025-04-26T17:55:01.992198+00:00 authserver C...

Figure 54 Grafana 3

Ids query (influxdb):

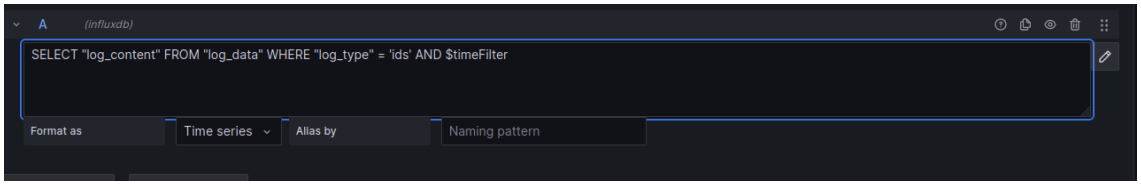


Figure 55 IDS query

6.4.2 Key Techniques and Algorithms

Real-Time Log Forwarding Algorithm (Python)
One major component of the system is the log tailing + forwarding loop used in forward_suricata_alerts.py. This allows direct detection and move of alerts from the IDS to the central server.
Efficient (uses seek() and readline() methods)
No dependency on log agents or external tools
Easy to integrate with existing Flask API
Pseudo-code:
Open the alert log file
Seek to the end
Loop:
  Read a new line
  If new data:
    Convert to JSON
    Send to /receive_logs API
  Else:
    Wait briefly

Table 12 Requirements

Requirement	Input	Output	Processes	Constraints	Importance	Implemented?
Authentication	Single sign-on credentials	Access given/denied	Validate credentials and roles	Must be secure; show correct errors	Essential	Yes
Log collection	Various data sources	Raw log data	Extract logs from multiple servers	Must collect from kernel, auth, syslog	Essential	Yes
Log Formatting	Raw log data	Structured logs	Normalize and structure logs	Must be compatible with InfluxDB & Grafana	Essential	Yes
Log Filtering and Sampling	Raw log data	Filtered logs	Filter based on importance	Only important logs forwarded	Essential	Yes
Alerting	Processed logs (Suricata)	Alert notifications	Trigger alerts on suspicious activity	Must detect port scans, brute force, log injection	Essential	Yes
Log correlation	Multiple log sources	Correlated logs	Combine related log events	Must reveal multi-step or linked behaviors	Essential	Yes
Reporting	Processed Logs	Dashboard insights	Send data to Grafana Organize logs by purpose or source	Dashboards must be clear and real-time	Essential	Yes
Log Categorization	Structured logs	Labeled log types	Organize logs by purpose or source	Must recognize syslog, auth, kernel, IDS	Essential	Yes
Log processing	Raw/categorized logs	Clean logs for DB	Normalize, aggregate, and prepare logs	Must be optimized for InfluxDB use	Essential	Yes

Chapter 7

Testing

7.1 Testing Approach

To test the effectiveness of the logging steps and IDS, we made different experiments. These tests focused on validating each component of the system by doing simulated attacks.
Validation Criteria:
1-If the IDS detects and forwards alerts based on a specific pattern.
2- If logs are correctly visualized in Grafana.
3-Time between the generation of the logs and visualization.
4-The servers correctly sending the logs to central server.
Each stage was tested alone before system integration:

Table 13 Pipeline

Stage	Test Performed	Result
Log Collection (scripts)	Logs correctly extracted into structured JSON	Successful
Log Transmission	POST request delivers logs to central Flask API	Successful
Log Reception & Storage	Central API writes logs into InfluxDB	Successful
Alert Forwarding (Suricata)	Alerts from Suricata are forwarded to the central API	Successful
Grafana Visualization	InfluxDB data shown in panels	Successful

```
administrator@authserver:~$ python3 /home/administrator/send_authlogs.py
Logs have been saved to authlogs_output.json
Status: 200
Response: {"message":"Received 56827 logs from authserver","status":"success"}
```

Figure 56 Sending authlogs

```
administrator@authserver:~$ nano /home/administrator/forward_suricata_alerts.py
administrator@authserver:~$ python3 /home/administrator/forward_suricata_alerts.py
🔁 Watching /var/log/suricata/fast.log ...
📄 Read line: 05/06/2025-16:01:40.903744  [**] [1:2210045:2] SURICATA STREAM Packet with invalid ack [**] [Classifi
cation: Generic Protocol Command Decode] [Priority: 3] {TCP} 93.127.132.79:443 → 92.118.39.237:19053
✅ Sent! Status: 200
📄 Read line: 05/06/2025-16:01:40.903744  [**] [1:2210046:2] SURICATA STREAM SHUTDOWN RST invalid ack [**] [Classif
ication: Generic Protocol Command Decode] [Priority: 3] {TCP} 93.127.132.79:443 → 92.118.39.237:19053
✅ Sent! Status: 200
```

Figure 57 Suricata Sending Alerts

Attack Simulation:
Multiple attacks were done from Kali against the authserver while Suricata tracked the network. These attacks were done to trigger alerts and test log volume detection capabilities.

Table 14 Attacks done

Attack Type	Tool	Target Port	Expected Outcome	Alerted?
SSH Brute Force	Hydra	22	Detect multiple failed login attempts	Yes

Port Scan	Nmap	All	Detect scan activity	Yes
DoS Flood	hping3	80	Detect abnormal SYN flood	Yes
Log Injection	echo	N/A	Detect suspicious log entries	Yes

Suricata sending alerts successfully:

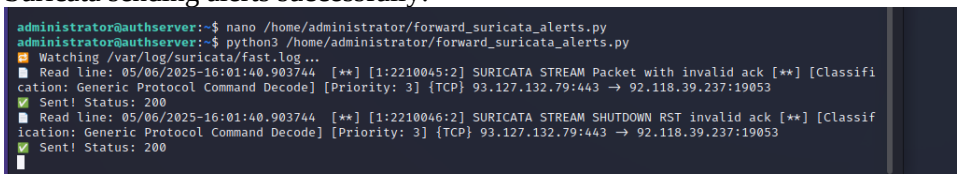


Figure 58 Suricata Reading Alerts Successfully

Brute force attack on authentication server :

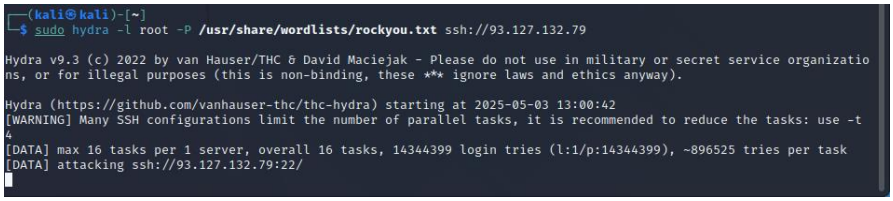


Figure 59 Brute Force Attack 1

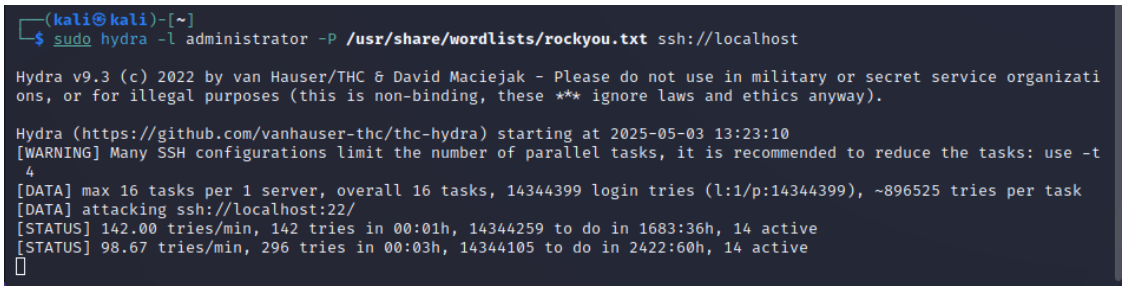


Figure 60 Brute force attack 2

nmap port scans :

Figure 61 Port scan attack 1

Figure 62 Port scan attack 2

Nmap success :

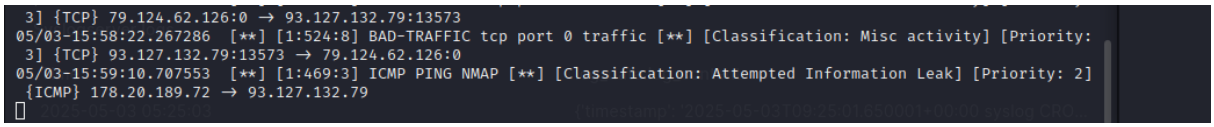


Figure 63 Successful port scan attack

Dos attack:

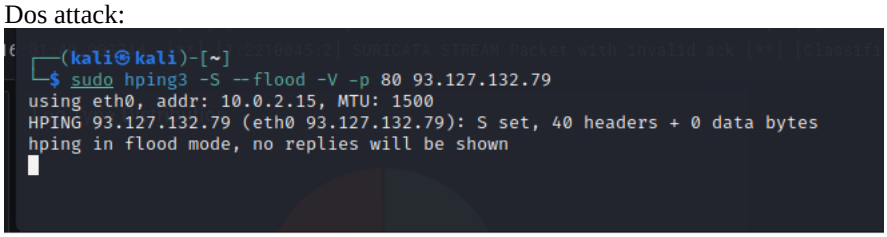


Figure 64 DOS Attack

Automated log injection attack :

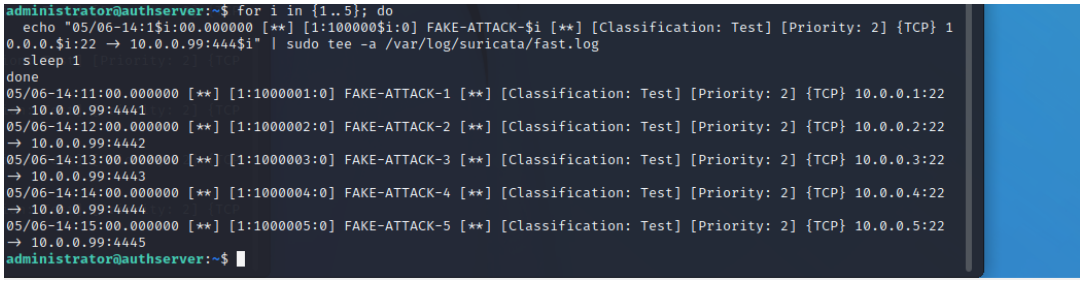


Figure 65 Log Injection Automated

Autmoted log injection running in the background :

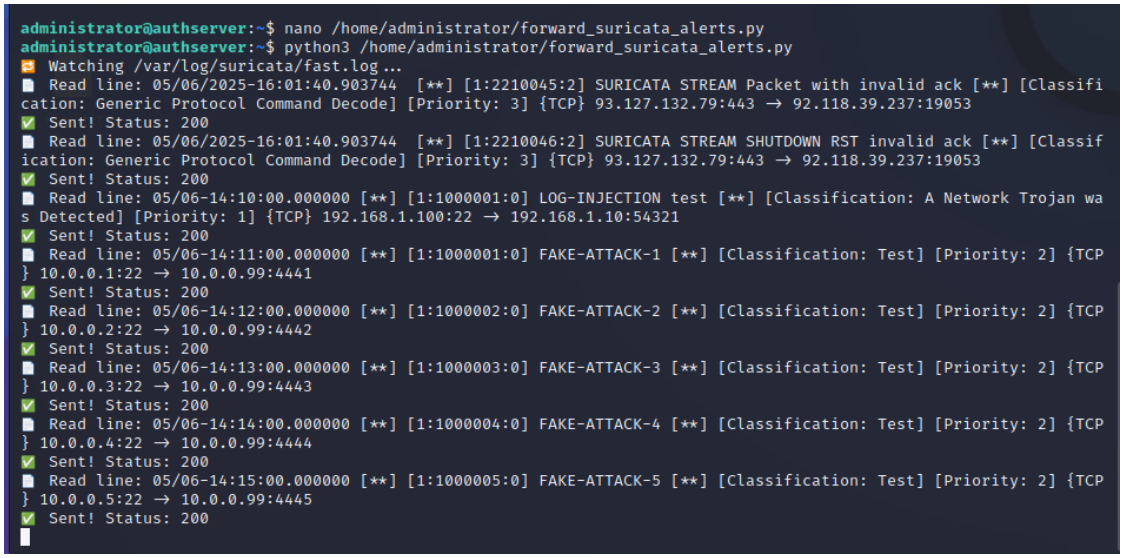


Figure 66 Alerts showing 1

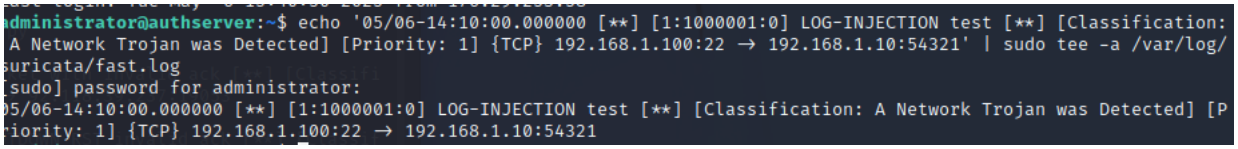


Figure 67 Alerts showing 2

7.2 Testing Results

The Grafana dashboard panels were used to show how many alerts were triggered during attacks. The panels included:

Table 15 Testing Results

Panel Title	Result during tests
IDS Alerts Over Time	Sudden increase in alert bars during attacks
IDS Alert Frequency	Showed sudden activity during tests
Snort IDS Alerts (Log view)	Showed raw alert messages
Log Type Distribution	Showed mix of log types
All Logs (Table)	Showed logs from all three servers

Figure 68 Grafana Alerts 1

Figure 69 Grafana Alerts 2

Figure 70 Grafana Alerts 3

Testing:
Took care of different logs from three servers.
Showed IDS alerts in real-time.
Forwarded logs and active alerts through the same structure.
Done log injection without breaking visualization or data consistency.
Insights:
Suricata proved more stable than Snort in logging and alert file.
Feedback from Grafana allowed easier debugging and verification.

7.3.1 Comparison Between Traditional and Structured Logging

1-Storage Efficiency

Authentication Raw Logs:

```
┌──(dalia㉿kali)-[~]
└─$ cp /var/log/auth.log ~/raw_logs.txt

┌──(dalia㉿kali)-[~]
└─$ du -h ~/raw_logs.txt
88K     /home/dalia/raw_logs.txt
```

Authentication Structured Logs;

```
┌──(dalia㉿kali)-[~]
└─$ du -h authlogs_output.json
72K     authlogs_output.json
```

System Raw Logs:

```
┌──(dalia㉿kali)-[~]
└─$ du -h raw_system_logs.txt
22M     raw_system_logs.txt
```

Figure 64 Calculations

System Structured Logs:

```
┌──(dalia㉿kali)-[~]
└─$ du -h syslogs_output.json
16M     syslogs_output.json
```

Kernel Raw logs:

```
┌──(dalia㉿kali)-[~]
└─$ du -h raw_kernel_logs.txt
108K    raw_kernel_logs.txt
```

Structured Kernel Logs:

```
┌──(dalia㉿kali)-[~]
└─$ du -h kernel_logs_output.json
4.0K    kernel_logs_output.json
```

Compression Ratio Formula=Raw log size ÷ Structured Log Size

1-Authentication Logs

Raw=88KB
Structured=72KB
88 ÷72=1.222
18% storage reduction

2-System Logs
Raw=22MB
Structured=18MB
22÷18=1.222
18% storage reduction

3-Kernel Logs

Raw=108KB
Structured=4.0KB
108÷4=27
96.3 storage reduction

2-Data Usability

Raw logs : low and needs manual filtering
Structured logs: high and ready to use

3-Attack Detection

Raw logs: manual using tools like grep and search
Structured logs: automatic pattern detection, can point suspicious activity.

4-Processing time
Although raw logs processed faster, they require manual filtering unlike the structured logs that support direct filtering and integration, which saves time.

```
┌──(dalia㉿kali)-[~]
└─$ nano structures_log_reader.py

┌──(dalia㉿kali)-[~]
└─$ python3 structures_log_reader.py
Auth Logs (Structured) Processing Time: 0.1386 seconds
Kernel Logs (Structured) Processing Time: 0.0493 seconds
System Logs (Structured) Processing Time: 0.7836 seconds

Kernel Logs (Raw) Processing Time: 0.0002 seconds
System Logs (Raw) Processing Time: 0.0225 seconds
```

Chapter 8

Conclusions and Future Work

8.1 Conclusions

As we wrap up our intensive work and journey at Princess Sumaya University for Technology it is time to reflect on our journey and impact we have made.

Project Overview and Success: Our project titled “A Lightweight and Adaptive Logging Architecture” main’s goal was to enhance the logging barriers related to the collection, storage, and managing of logs. It successfully enhanced
the process of storing logs making them more lightweight and still capable of real-time detection. The system we developed made the log management process easier across multiple servers by using adaptable python scripts, Influx DB, and Suricata. The system has real-time detection and visualization in Grafana, and it was validated by simulated attacks like log injection, brute force login attempts, and port scanning.

Approaches and Methods: The balance between simplicity and capability is what made the project stand out. Instead of collecting huge number of raw logs, our system focused on collecting specific attributes from specific logs that help in detecting various attacks. We also used lightweight REST API using Flask. Choosing Grafana and Influx DB helped in providing high performance and visualization without heavy querying.

Focus on education and ethics: Simulated attacks were done in a controlled environment and all configurations prioritized integrity. We took care of how improper logging, weak detection mechanisms, or lack of monitoring could be exploited, and how lightweight and ethical logging architectures can serve as first lines of defense in any organization.

Looking Ahead: **A**t the end, we agree that the challenges in the cybersecurity world are ongoing but through our work we believe that the project has laid a strong foundation for future research as we ensure that our contribution to cybersecurity will proceed to evolve and stay relevant and up to date to the market trends.

Focus on Education and Ethics: the project has placed a strong emphasis on making work accessible for academic purposes and aligning to ethical standards including developing a user-friendly interface.

8.2 Future work

1.Customization Options: The system offers customization features allowing the users to edit it and customize alert options.
2.Implementing a User Feedback System: The user feedback system will operate as an integrated part of the system to encourage users to participate in the enhancement of the system.
3.Conducting Targeted Surveys: surveys targeting users and industry experts for the purpose of alignment with the cybersecurity model with realistic needs and feedback will be regularly conducted.
4.Continuous Adaptation to Emerging Threats: A planned key strategy is to create a framework for our models to provide adaptability to new cyber threats through integration of AI for ongoing learning and strategy modification.
5.Broader attack Detection: We will expand the system to detect additional attack types that involves updating the data model and integrating new log sources.
6.Integration with SIEM Platforms
Enhance the system's ability to detect alerts by integrating it with Security Information and Event Management tools like Splunk for advanced threat intelligence feeds.
7.Mobile Dashboard Interface
Develop a mobile-friendly application so that SOC teams can monitor alerts and metrics on the go.
8.User Role Management and Access Control
Implement a role-based access control system for the graphical user interface and API, ensuring different users can only access authorized sections or data.

References

[1]Ahlawat, P. et al. (2024) A new architecture to manage data costs and complexity, BCG Global. Available at:https://www.bcg.com/publications/2023/new-data-architectures-can-help-manage-data-costs-and-complexity (Accessed: 04 January 2025).

[2]Berman, D. (2019, January 24). The Challenge of Log Management in Modern IT Environments. Logz.io. https://logz.io/blog/modern-log-management-challenges/

[3]S. Gu, G. Rong, H. Zhang and H. Shen, "Logging Practices in Software Engineering: A Systematic Mapping Study," in IEEE Transactions on Software Engineering, vol. 49, no. 2, pp. 902-923, 1 Feb. 2023, doi: 10.1109/TSE.2022.3166924. keywords: {Software systems;Codes;Systematics;Task analysis;Runtime;Industries;Maintenance engineering;Logging practices;log;systematic mapping;empirical study},

[4] Nearform. (n.d.-b). The Cost of Logging in 2022 | Nearform. https://www.nearform.com/insights/the-cost-of-logging-in-2022/

[5] Editorial Team. (2024, December 24). insideAI News. https://insideainews.com/2024/12/26/netapps-2024-data-complexity-report-reveals-ais-make-or-break-year-ahead/

[6]. Y. Li, J. Liu and L. Wang, "Lightweight network research based on deep learning: A review", 2018 37th Chinese Control Conference (CCC), pp. 9021-9026, 2018.

[7] Exabeam. (2024, October 8). Top 6 Log Management Tools in 2024 and How to Choose | Exabeam. https://www.exabeam.com/explainers/log-management/top-6-log-management-tools-and-how-to-choose

[8] Dhivya, K., Kumar, P.P., Saravanan, D. and Pajany, M., 2018. Evaluation of web security mechanisms using vulnerability & Sql attack injection. International Journal of Pure and Applied Mathematics, 119(14), pp.989-996.

[9]Wickramasinghe, S. (n.d.). Logstash 101: Using Logstash in a Data Processing Pipeline. BMC Blogs. https://www.bmc.com/blogs/logstash-using-data-pipeline/

[10] LogicMonitor. (2024, September 11). Managed Service Provider (MSP) IT Monitoring Solutions | LogicMonitor. https://www.logicmonitor.com/msp

[11] Mathijssen, M., Overeem, M., & Jansen, S. (2020). Identification of Practices and Capabilities in API Management: A Systematic Literature Review. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2006.10481

[12] Y. Sun, Y. Chen, H. Zhao and S. Peng, "Design and Development of a Log Management System Based on Cloud Native Architecture," 2023 9th International Conference on Systems and Informatics (ICSAI), Changsha, China, 2023, pp. 1-6, doi: 10.1109/ICSAI61474.2023.10423328.
keywords: {Cloud computing;System performance;Soft sensors;Service computing;Computer architecture;Production;Anomaly detection;Cloud Native;Log System;Anomaly Detection}, https://ieeexplore-ieee-org.psut.idm.oclc.org/document/10423328

[13]Y. Wang, "Design of Visual Log Analysis System," 2023 IEEE International Conference on Sensors, Electronics and Computer Engineering (ICSECE), Jinzhou, China, 2023, pp. 1649-1652, doi: 10.1109/ICSECE58870.2023.10263397.keywords: {Visualization;System performance;Operating systems;Data processing;Software systems;User experience;Sensor systems;log analysis;log parsing;anomaly detection;visualization}, https://ieeexplore-ieee-org.psut.idm.oclc.org/document/10263397

[14] *J. Muan "Centralized logging architecture*. IBM" retreived May 2025. (n.d.). https://www.ibm.com/docs/en/cics-tx/10.1.0?topic=stack-centralized-logging-architecture

[15] Jamie Riedesel, Software Telemetry: Reliable logging and monitoring , Manning, 2021. keywords: {software;telemetry;operations;devops;security;cloud;business;metadata;logging;records;sampling}, https://ieeexplore-ieee-org.psut.idm.oclc.org/document/10280575

[16]S. Locke, H. Li, T. -H. P. Chen, W. Shang and W. Liu, "LogAssist: Assisting Log Analysis Through Log Summarization," in *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3227-3241, 1 Sept. 2022, doi: 10.1109/TSE.2021.3083715. keywords: {Task analysis;Runtime;Tools;Testing;Faces;Anomaly detection;Software systems;Log analysis;log compression;n-gram modeling;log abstraction;workflow characterization;log reduction}, https://ieeexplore-ieee-org.psut.idm.oclc.org/document/9442364

[17]Y. Wang, "Design of Visual Log Analysis System," *2023 IEEE International Conference on Sensors, Electronics and Computer Engineering (ICSECE)*, Jinzhou, China, 2023, pp. 1649-1652, doi: 10.1109/ICSECE58870.2023.10263397. keywords: {Visualization;System performance;Operating systems;Data processing;Software systems;User experience;Sensor systems;log analysis;log parsing;anomaly detection;visualization},
https://ieeexplore-ieee-org.psut.idm.oclc.org/document/10263397

Appendices

Appendix A

A.1 Users' Manual

1-Purchase the servers needed for log collection and one central server

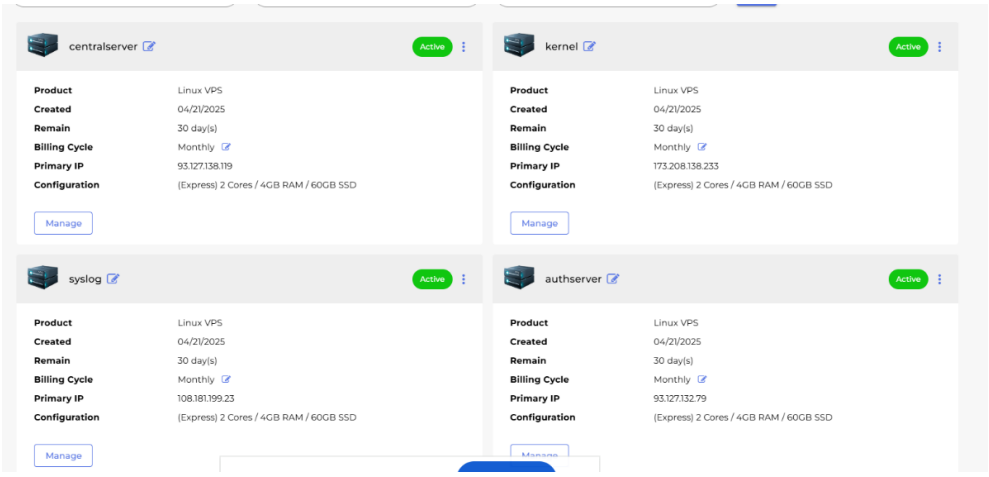


Figure 71 Servers purchased

2- Run a script to extract specific logs decided

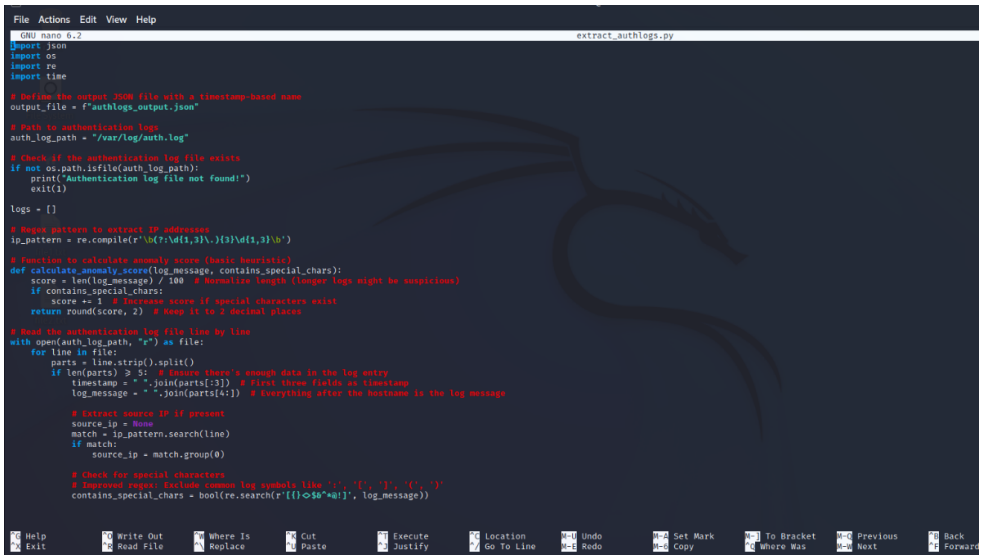


Figure 72 Authentication extraction script

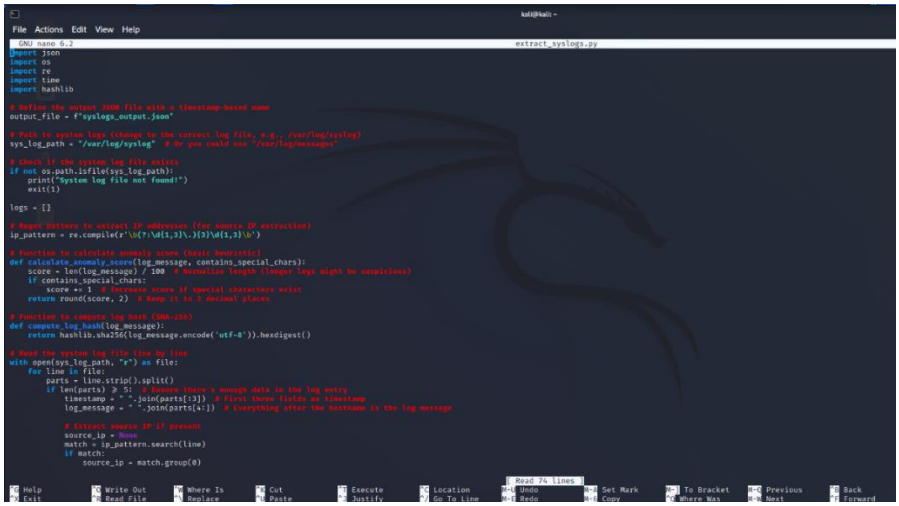


Figure 73 System Logs Extraction

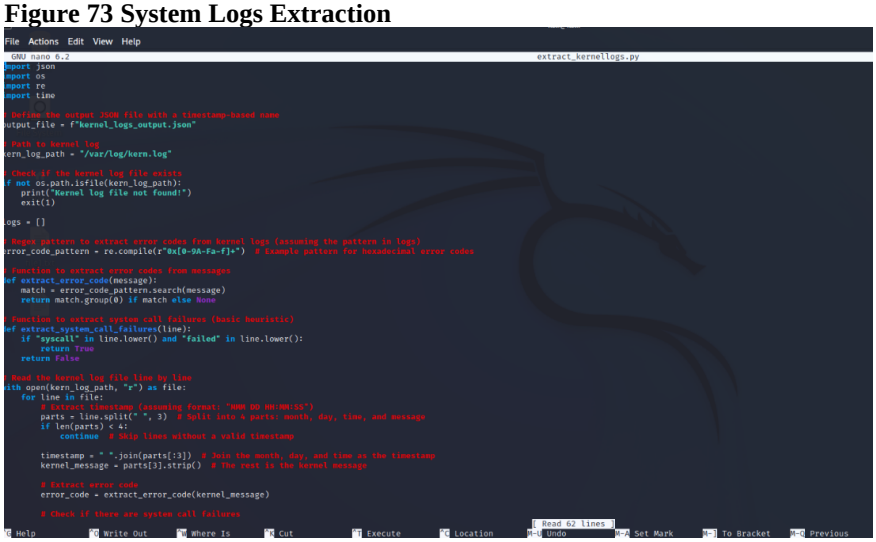


Figure 74 Kernel Logs Extraction

3-Accessing all servers

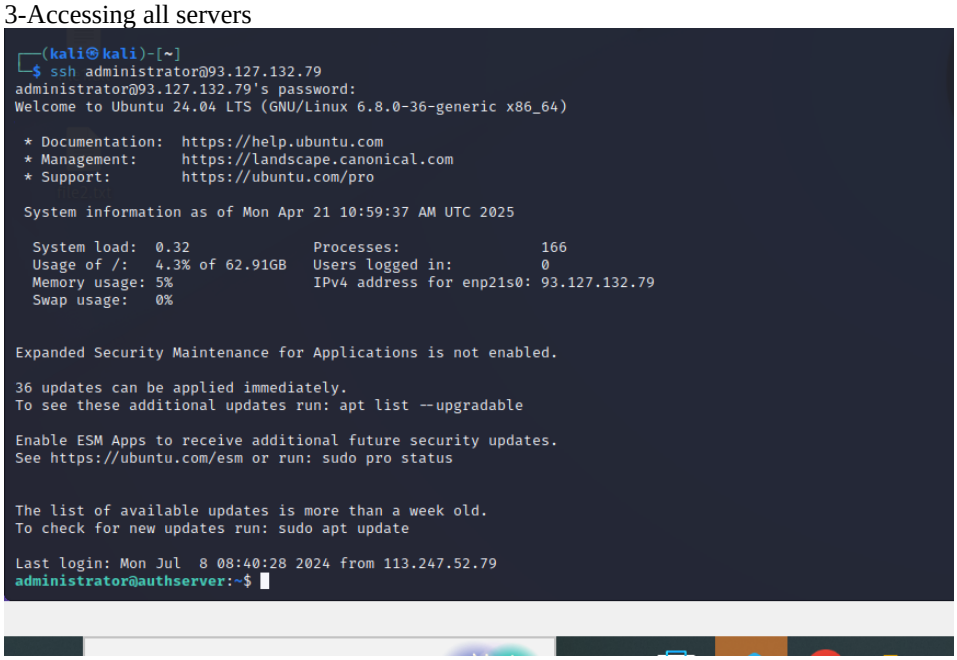


Figure 75 Accessing server1

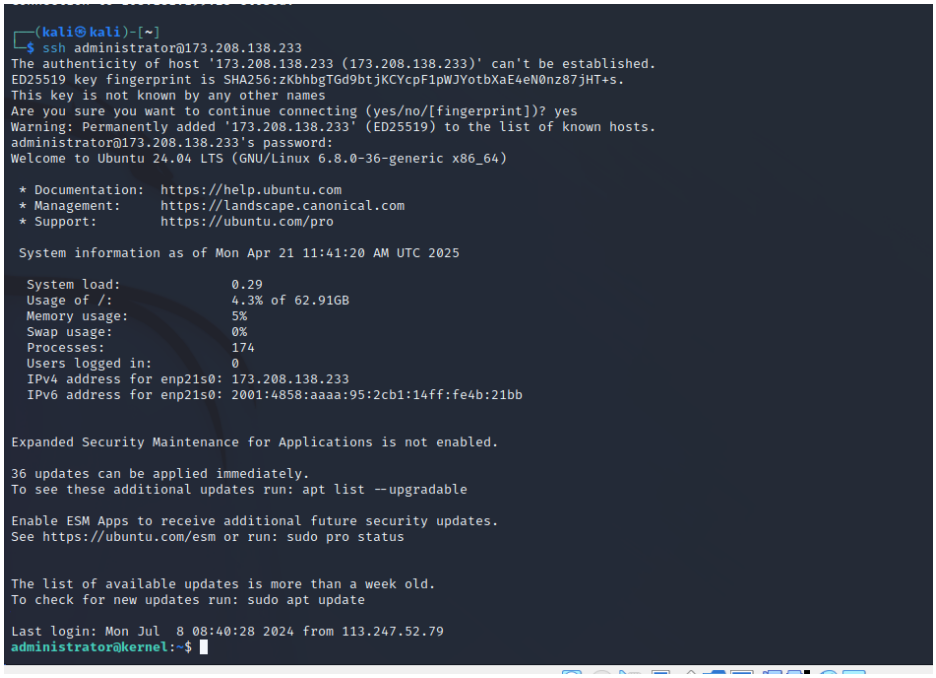


Figure 76 Accessing server2

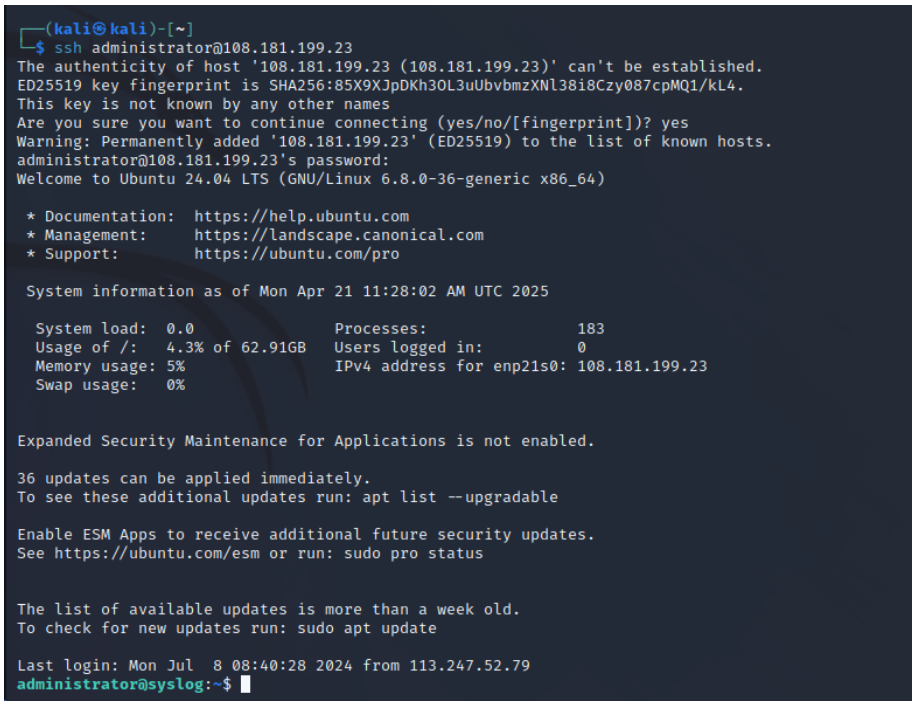


Figure 77 Accessing server3

4-Move the logs extracted to their specific server using SCP

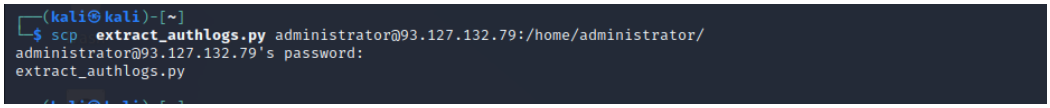


Figure 78 Using scp on authlogs.py

```
┌──(kali㉿kali)-[~]
└─$ scp extract_syslogs.py administrator@108.181.199.23:/home/administrator/
administrator@108.181.199.23's password:
extract_syslogs.py                                            100% 2624    13.0KB/s   00:00

┌──(kali㉿kali)-[~]
└─$
```

Figure 79 Using scp on syslogs.py

```
┌──(kali㉿kali)-[~]
└─$ scp extract_kernellogs.py administrator@173.208.138.233:/home/administrator/
administrator@173.208.138.233's password:
extract_kernellogs.py                                         100% 2073    11.4KB/s   00:00

┌──(kali㉿kali)-[~]
```

Figure 80 Using scp on kernellogs.py

5-Installing Flask and connecting Influx database to the central server

```
┌──(kali㉿kali)-[~/loki]
└─$ pip install Flask requests

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: Flask in /usr/lib/python3/dist-packages (2.0.1)
Requirement already satisfied: requests in /usr/lib/python3/dist-packages (2.27.1)

┌──(kali㉿kali)-[~/loki]
└─$
```

Figure 81 Flask Install

```
administrator@centralserver:~$ influx
Connected to http://localhost:8086 version 1.6.7~rc0
InfluxDB shell version: 1.6.7~rc0
> CREATE DATABASE logs
> exit
administrator@centralserver:~$
```

Figure 82 InfluxDB

6-Write a script to send logs to the central server for storage

```
 GNU nano 7.2                        log_receiver.py *
from flask import Flask, request, jsonify
from influxdb import InfluxDBClient

app = Flask(__name__)

# InfluxDB setup
client = InfluxDBClient(host='localhost', port=8086)
client.switch_database('logs')  # Make sure this database exists

@app.route('/receive_logs', methods=['POST'])
def receive_logs():
    log_data = request.get_json()
    source = log_data.get("source")
    logs = log_data.get("logs", [])

    influx_points = []

    for entry in logs:
        influx_points.append({
            "measurement": "log_data",
            "tags": {
                "log_type": source
            },
            "fields": {
                "log_content": str(entry)
            }
        })

    if influx_points:
        client.write_points(influx_points)

    return jsonify({"status": "success", "message": f"Received {len(influx_points)} logs fro

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
```

Figure 83 Log sending script

```
administrator@kernel:~$ nano /home/administrator/extract_kernellogs.py
administrator@kernel:~$ nano /home/administrator/send_kernellogs.py
administrator@kernel:~$ python3 /home/administrator/send_kernellogs.py
Kernel logs have been saved to kernel_logs_output.json
Status: 200
Response: {"message":"Received 5121 logs from kernelserver","status":"success"}

administrator@kernel:~$
```

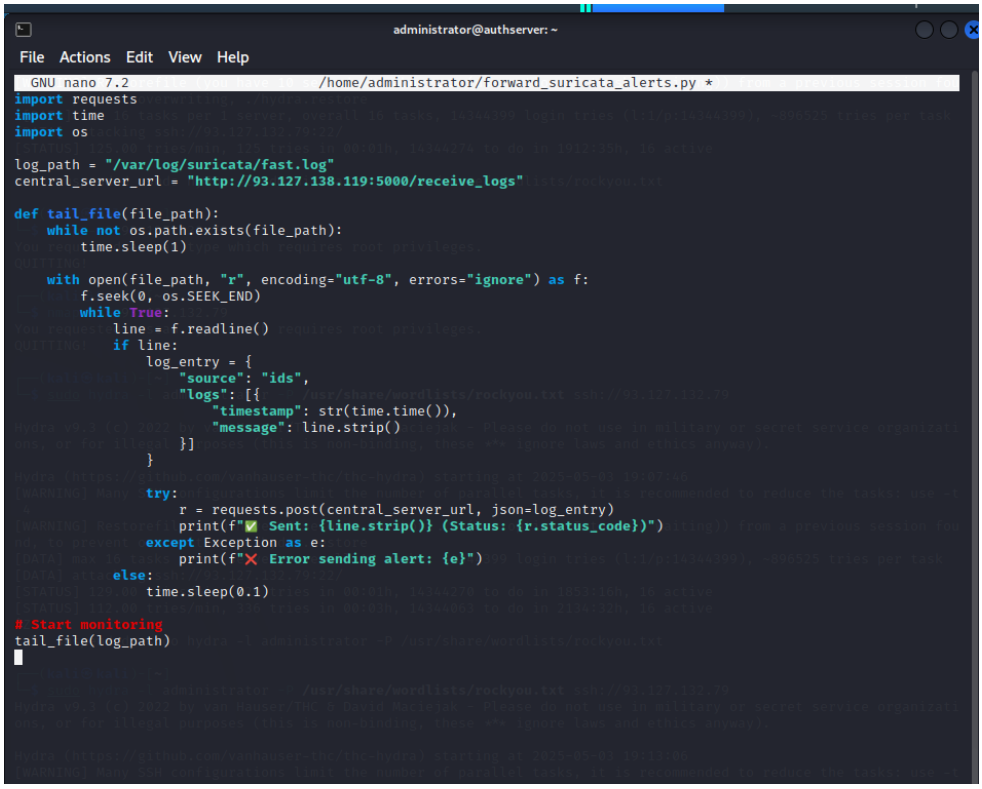
Figure 84 Receiving kernel logs

```
administrator@syslog:~$ nano /home/administrator/extract_syslogs.py
administrator@syslog:~$ nano /home/administrator/send_syslogs.py
administrator@syslog:~$ python3 /home/administrator/send_syslogs.py
Logs have been saved to syslogs_output.json
Status: 200
Response: {"message":"Received 13505 logs from syslogserver","status":"success"}

administrator@syslog:~$
```

Figure 85 Receiving system logs

6-Set suricata IDS to forward alerts to Grafana



7-Install Grafana on central server and access it using firefox to add panels and visualize

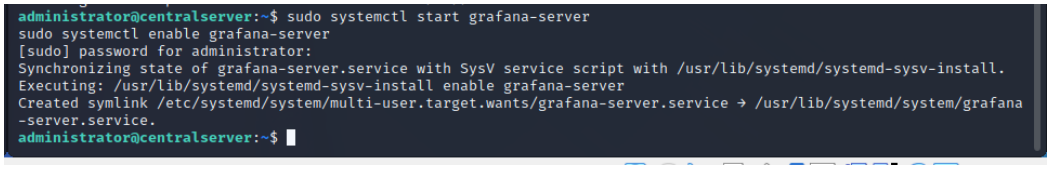


Figure 86 Grafana Installation

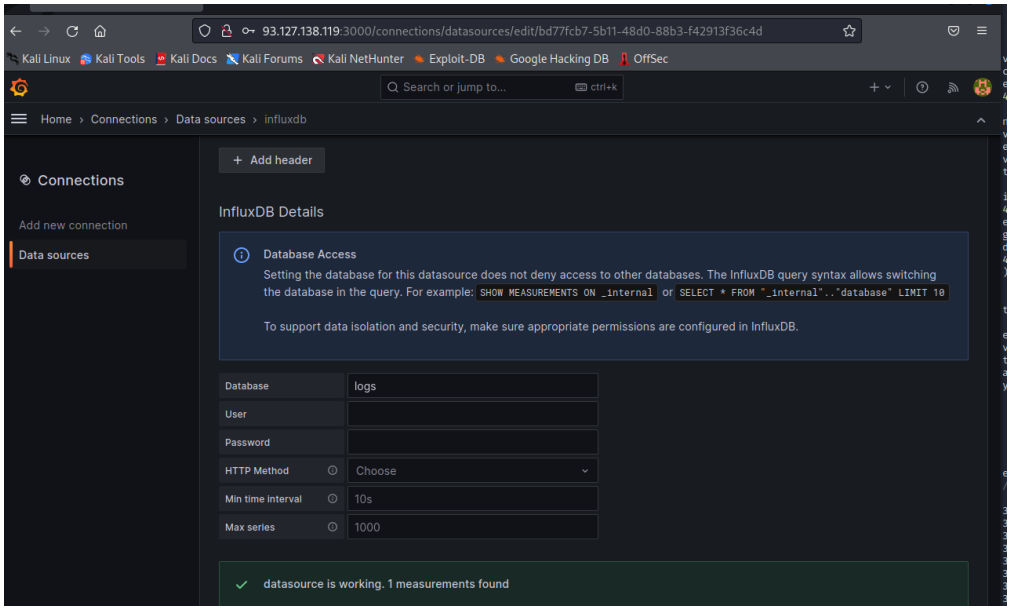


Figure 87 Accessing Grafana

Appendix B

B.1 Document Changes

1. **IDS Tool Defined**

- **Change**: In GP1 document we discussed integrating an Intrusion Detection System (IDS) as part of the system, but we did not specify which tool would be used. In GP2, **Suricata** was selected and integrated as the IDS component.
- **Reason**: Suricata was chosen for its structured alert format, performance, and compatibility with the logging structure, supporting the goal of creating an adaptive and lightweight system.

2. **Tools for Storage and Visualization**

- **Change**: In GP2 we decided to use **InfluxDB** to store logs and alerts as time-stamped data and **Grafana** to visualize them in real time.

- **Reason**: These tools were chosen because they handle data efficiently and work well together. Grafana makes it easy to create dashboards that show alert counts and log activity which helps SOC teams quickly understand what's happening across the network.

3. **Log Transmission Method**

- **Change**: In GP2 we introduced a clear method for sending logs. Using a lightweight Flask API to transmit log data as JSON over HTTP from each server to a central server.

- **Reason**: This wasn't specified in GP1 document. Adding this method made the system more practical by allowing log transfer without needing shared storage or manual handling. It also keeps the overall design simple and easy to scale.

Appendix C

C.1 Code Documentation

```
# ===== 1. extract_authlogs.py =====

import json
import os
import re
import time

output_file = f"authlogs_output.json"
auth_log_path = "/var/log/auth.log"

if not os.path.isfile(auth_log_path):
    print("Authentication log file not found!")
    exit(1)

logs = []
ip_pattern = re.compile(r'\b(?:\d{1,3}\.){3}\d{1,3}\b')

def calculate_anomaly_score(log_message, contains_special_chars):
    score = len(log_message) / 100
    if contains_special_chars:
        score += 1
    return round(score, 2)

with open(auth_log_path, "r") as file:
    for line in file:
        parts = line.strip().split()
        if len(parts) >= 5:
            timestamp = " ".join(parts[:3])
            log_message = " ".join(parts[4:])
            source_ip = None
            match = ip_pattern.search(line)
            if match:
                source_ip = match.group(0)
            contains_special_chars = bool(re.search(r'[{}<>$&^*@!]',
log_message))
            log_length = len(log_message)
            anomaly_score = calculate_anomaly_score(log_message,
contains_special_chars)
            logs.append({
                "timestamp": timestamp,
                "source_ip": source_ip if source_ip else "N/A",
                "log_message": log_message,
                "contains_special_chars": contains_special_chars,
                "log_length": log_length,
                "anomaly_score": anomaly_score
            })
```

```
with open(output_file, "w") as json_file:
    json.dump(logs, json_file, indent=4)

print(f"Logs have been saved to {output_file}")


# ===== 2. extract_syslogs.py =====

import json
import os
import re
import time
import hashlib

output_file = f"syslogs_output.json"
sys_log_path = "/var/log/syslog"

if not os.path.isfile(sys_log_path):
    print("System log file not found!")
    exit(1)

logs = []
ip_pattern = re.compile(r'\b(?:\d{1,3}\.){3}\d{1,3}\b')

def calculate_anomaly_score(log_message, contains_special_chars):
    score = len(log_message) / 100
    if contains_special_chars:
        score += 1
    return round(score, 2)

def compute_log_hash(log_message):
    return hashlib.sha256(log_message.encode('utf-8')).hexdigest()

with open(sys_log_path, "r") as file:
    for line in file:
        parts = line.strip().split()
        if len(parts) >= 5:
            timestamp = " ".join(parts[:3])
            log_message = " ".join(parts[4:])
            source_ip = None
            match = ip_pattern.search(line)
            if match:
                source_ip = match.group(0)
            contains_special_chars = bool(re.search(r'[{}<>$&^*@!]',
log_message))
            log_length = len(log_message)
            anomaly_score = calculate_anomaly_score(log_message,
contains_special_chars)
            log_hash = compute_log_hash(log_message)
```

```
            logs.append({
                "timestamp": timestamp,
                "source_ip": source_ip if source_ip else "N/A",
                "log_hash": log_hash,
                "log_length": log_length,
                "anomaly_score": anomaly_score,
                "log_message": log_message
            })

with open(output_file, "w") as json_file:
    json.dump(logs, json_file, indent=4)

print(f"Logs have been saved to {output_file}")


# ===== 3. extract_kernellogs.py =====

import json
import os
import re
import time

output_file = f"kernel_logs_output.json"
kern_log_path = "/var/log/kern.log"

if not os.path.isfile(kern_log_path):
    print("Kernel log file not found!")
    exit(1)

logs = []
error_code_pattern = re.compile(r"0x[0-9A-Fa-f]+")

def extract_error_code(message):
    match = error_code_pattern.search(message)
    return match.group(0) if match else None

def extract_system_call_failures(line):
    return "syscall" in line.lower() and "failed" in line.lower()

with open(kern_log_path, "r") as file:
    for line in file:
        parts = line.split(" ", 3)
        if len(parts) < 4:
            continue
        timestamp = " ".join(parts[:3])
        kernel_message = parts[3].strip()
        error_code = extract_error_code(kernel_message)
        system_call_failure = extract_system_call_failures(kernel_message)
        logs.append({
            "timestamp": timestamp,
```

```
            "kernel_message": kernel_message,
            "error_code": error_code if error_code else "N/A",
            "system_call_failure": system_call_failure
        })

with open(output_file, "w") as json_file:
    json.dump(logs, json_file, indent=4)

print(f"Kernel logs have been saved to {output_file}")


# ===== 4. send_authlogs.py =====

import requests, json, subprocess

subprocess.run(["python3", "extract_authlogs.py"])

with open("authlogs_output.json", "r") as file:
    logs = json.load(file)

response = requests.post(
    "http://93.127.138.119:5000/receive_logs",
    json={"source": "authserver", "logs": logs}
)

print("Status:", response.status_code)
print("Response:", response.text)


# ===== 5. log_receiver.py =====

from flask import Flask, request, jsonify
from influxdb import InfluxDBClient

app = Flask(__name__)
client = InfluxDBClient(host='localhost', port=8086)
client.switch_database('logs')

@app.route('/receive_logs', methods=['POST'])
def receive_logs():
    log_data = request.get_json()
    source = log_data.get("source")
    logs = log_data.get("logs", [])
    influx_points = []
    for entry in logs:
        influx_points.append({
            "measurement": "log_data",
            "tags": {"log_type": source},
            "fields": {"log_content": str(entry)}
        })
```

```
    if influx_points:
        client.write_points(influx_points)
    return jsonify({"status": "success", "message": f"Received
{len(influx_points)} logs from {source}"}), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

 # ===== 6. forward_suricata_alerts.py =====

```
import requests
import time
import os

log_path = "/var/log/suricata/fast.log"
central_server_url = "http://93.127.138.119:5000/receive_logs"

def tail_file(file_path):
    print(f"🔁 Watching {file_path}...")
    while not os.path.exists(file_path):
        print("⏳ Waiting for log file...")
        time.sleep(1)

    with open(file_path, "r", encoding="utf-8", errors="ignore") as f:
        f.seek(0, os.SEEK_END)
        while True:
            line = f.readline()
            if line:
                print(f"📄 Read line: {line.strip()}")
                log_entry = {
                    "source": "ids",
                    "logs": [{
                        "timestamp": str(time.time()),
                        "message": line.strip()
                    }]
                }
                try:
                    r = requests.post(central_server_url, json=log_entry)
                    print(f"✅ Sent! Status: {r.status_code}")
                except Exception as e:
                    print(f"❌ Failed to send: {e}")
            else:
                time.sleep(0.5)

tail_file(log_path)
```

 # ===== 7. start_log_receiver.sh =====

```
#!/bin/bash
source /home/administrator/venv/bin/activate
python /home/administrator/log_receiver
```