ELK STACK DASHBOARD

Internship Project — Portfolio Version

DECEMBER 30, 2024

INAAM KABBARA
EPITA University – Ubility Company

Table of Contents

Abstract

This project focused on designing and implementing an automated threat detection and monitoring system using the ELK Stack (Elasticsearch, Logstash, and Kibana) combined with Python-based enrichment scripts. The primary objective was to streamline log analysis for security operations by automating the ingestion, categorization, and visualization of system and network logs. The solution integrated GeoIP enrichment to provide geographic insights, dynamic threat categorization through JSON-based rules, and real-time alerting via Slack webhooks. Evaluation on unseen log datasets demonstrated strong performance, achieving an accuracy of 73.68%, a recall of 85.71%, and an F1-score of 70.59%, confirming the system's reliability in detecting the majority of threats while maintaining a balance between true positives and false positives. By reducing manual workload and enabling faster identification of anomalies, the project significantly improved operational efficiency and delivered a practical, scalable solution tailored to the needs of Security Operations Centers.

Introduction

In modern cybersecurity operations, organizations generate vast volumes of log data from systems, applications, and network devices. Security Operations Centers (SOCs) rely on this data to detect, investigate, and respond to threats. However, manual log analysis is slow, error-prone, and often incapable of keeping pace with the complexity and scale of today's attacks. To overcome these challenges, the ELK Stack (Elasticsearch, Logstash, and Kibana) has become a widely adopted solution for centralized log management, real-time monitoring, and security analytics. Its scalability, flexibility, and visualization capabilities make it particularly suitable for building efficient and automated threat detection pipelines.

The goal of this internship project was to design and implement an enriched monitoring system that leverages the ELK Stack for automated log ingestion, categorization, and visualization, with additional processing through Python scripts. The system was designed to improve SOC efficiency by:

- Automating the collection and parsing of system logs.
- Dynamically categorizing threats using a JSON configuration file.
- Enriching log data with GeoIP information for contextual awareness.
- Providing real-time dashboards in Kibana to visualize threat activity.
- Sending automated alerts to Slack for high-severity incidents.

This report documents the full project lifecycle. It begins with the preparation of the environment and the installation of ELK components, followed by a description of the datasets and preprocessing techniques, including dynamic categorization and GeoIP enrichment. It then details the dashboard implementation, visualization features, and Slack integration for alerting. The evaluation section presents the system's performance using metrics such as accuracy, precision, recall, and F1-score, supported by visual results. Finally, the report discusses the challenges faced, the lessons learned, and the impact of the project within a SOC context.

By combining open-source technologies with automation, this project demonstrates how a well-engineered log analysis pipeline can reduce detection time, minimize manual workload, and strengthen the ability of SOC analysts to identify and respond to security incidents.

Methodology and Setup

The methodology for this project was carefully structured to ensure a systematic and reliable approach to building a complete log analysis and threat detection system using the ELK Stack. Each stage was carried out step by step, with iterative testing and adjustments to guarantee stability and accuracy. The work progressed from environment preparation and installation, through data acquisition and enrichment, to the creation of dashboards and alerting mechanisms.

1. Environment Preparation & Installation

The first stage focused on creating a stable and secure working environment capable of running the ELK Stack and supporting the additional processing requirements of Python scripts. The project was deployed on Ubuntu 20.04, a widely supported Linux distribution commonly used in cybersecurity research and enterprise SOC environments.

Key preparatory tasks included:

- System Updates: Ensuring that the operating system and dependencies were up to date to prevent compatibility issues.
- Java Installation: Installing and verifying the correct version of Java, a requirement for Elasticsearch.
- ELK Stack Components:
    - Elasticsearch was installed to serve as the main indexing and storage engine.
    - Logstash was configured to handle log parsing, transformation, and enrichment.
    - Kibana was deployed to provide a web-based interface for querying Elasticsearch and building dashboards.
- Python Environment: A Python virtual environment was created to isolate dependencies and ensure reproducibility. Libraries such as elasticsearch, geoip2, and requests were installed, enabling integration with Elasticsearch APIs, GeoIP enrichment, and Slack alerting.

This structured environment formed the backbone of the project and required significant effort in troubleshooting dependencies and ensuring seamless interaction between all components.

2. Data Sources

The system was designed to process multiple types of data in order to simulate real SOC conditions and provide meaningful insights.

- System Logs:
  Extracted from Linux system files (e.g., /var/log/auth.log), these logs included timestamps, source and destination IP addresses, usernames, and authentication messages. They provided the raw input for detecting anomalies such as brute-force attempts or unauthorized logins.
- Threat Categories (JSON-based rules):

A custom threat_categories.json file was created to define keywords, threat categories, and severity levels. For example, events containing “failed password” were categorized as brute-force attempts with high severity, while phishing-related keywords were flagged as medium severity. This design enabled flexible and dynamic categorization without modifying the core Python scripts.

- GeoIP Database:
  The MaxMind GeoLite2 database was integrated to enrich logs with geographic details. IP addresses were resolved into country, city, and approximate coordinates, allowing analysts to identify unusual access patterns such as repeated attempts from unexpected regions.

The combination of raw system logs, structured threat definitions, and contextual GeoIP data ensured that the pipeline was both technically comprehensive and practically useful for SOC workflows.

3. Preprocessing and Enrichment

Raw log data is often noisy and inconsistent, making preprocessing and enrichment essential for accurate threat detection.

- Standardization with Logstash Pipelines: Logstash parsed raw log entries into a consistent schema, extracting key fields such as timestamp, source IP, destination IP, and message for downstream processing.
- Dynamic Categorization: Python scripts matched log entries against the JSON-defined threat categories, flagging suspicious activity and assigning severity levels dynamically.
- GeoIP Enrichment: Each IP address was cross-referenced with the GeoLite2 database, adding geographic attributes such as country and city. This enrichment allowed for geographic mapping of threats and deeper contextual analysis.

The result of this stage was a set of enriched, structured logs ready for visualization and real-time monitoring.

4. Dashboard Implementation

The final stage of the methodology involved creating dashboards and alerting mechanisms to make the enriched data actionable for SOC analysts.

- Index Creation in Kibana: Data views were defined (e.g., syslog-*, threat-analysis) to organize ingested and enriched data.
- Dashboards and Visualizations: Multiple visualization types were implemented, including:
  - GeoIP Maps to display the geographic distribution of attack sources.
  - Bar Charts and Tables to categorize threats by type and severity.
  - Time-Series Graphs to show spikes in activity over time.
- Slack Alerts: Python scripts were configured to send webhook-based notifications directly to Slack channels for high-severity threats. This integration ensured immediate analyst awareness even without direct access to Kibana.

By combining visualization and alerting, the system transformed raw data into actionable intelligence, enabling faster detection and response.

Summary of Methodology

This methodology provided a complete, end-to-end pipeline: from preparing the environment and ingesting raw system logs, through categorization and enrichment, to delivering real-time dashboards and alerts. The work required persistence, troubleshooting, and repeated testing to overcome challenges such as dependency issues, false positives, and limited computing resources. Ultimately, this structured approach ensured the delivery of a reliable and scalable SOC-ready monitoring solution.

Results and Evaluation

The performance of the ELK Stack Dashboard was evaluated using both quantitative metrics and qualitative observations from the dashboards and alerting system. The evaluation focused on the system's ability to correctly categorize, enrich, and visualize threats in real time.

1. Performance Metrics

To assess the effectiveness of the categorization and enrichment pipeline, key metrics were calculated using a validation dataset that the system had not encountered during development.

Metric	Value	Interpretation
Accuracy	73.68%	The system correctly identified nearly three-quarters of all log events.
Precision	60.00%	Of all events flagged as threats, 60% were actual anomalies (40% false positives).
Recall	85.71%	The system successfully detected the majority of true threats, minimizing missed attacks.
F1-Score	70.59%	Balanced measure showing a good trade-off between precision and recall.

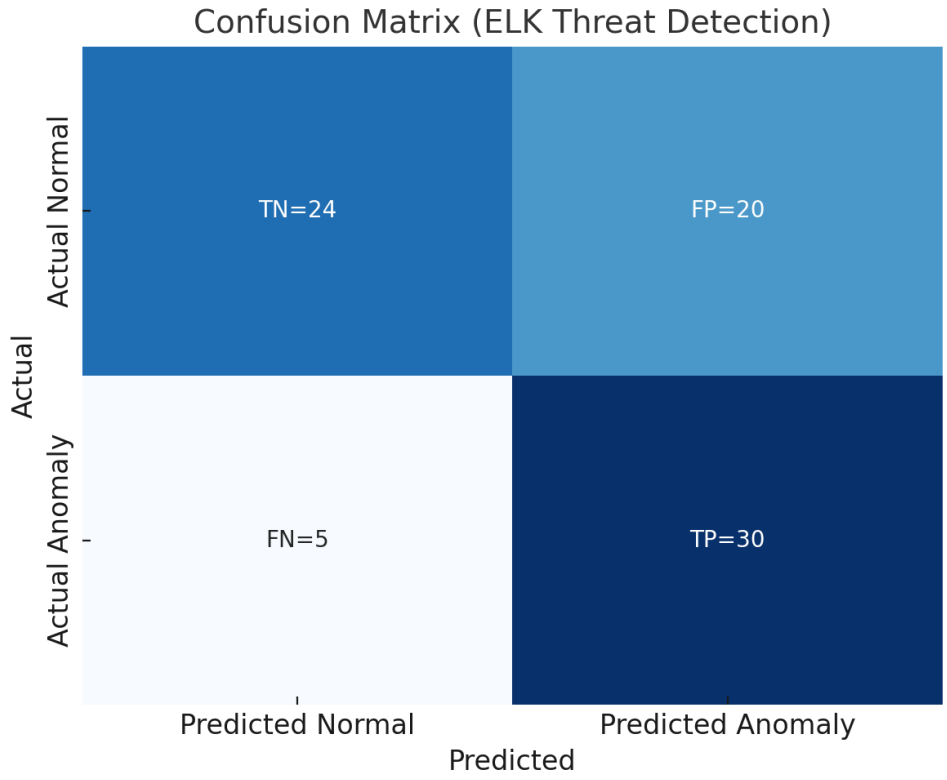
These results highlight the system's strengths in detecting threats (high recall) while also pointing to areas for improvement in reducing false positives (precision).

2. Visual Results

To complement the metrics, several visual outputs were generated:

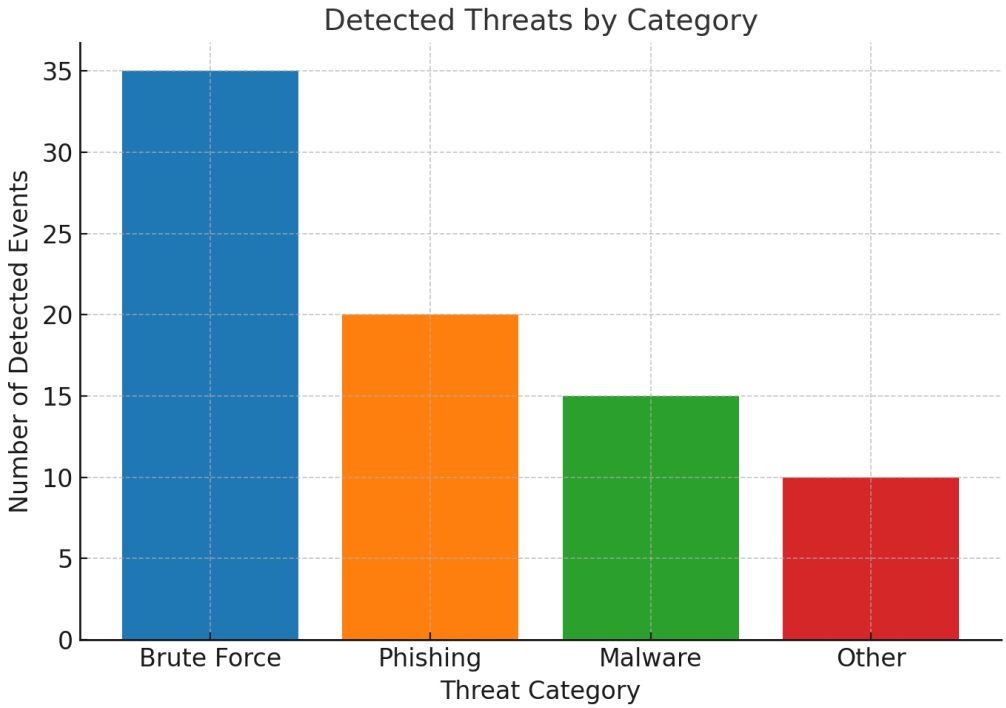
(a) Confusion Matrix

- A confusion matrix was plotted to illustrate the number of true positives, false positives, true negatives, and false negatives.
- This visualization clearly showed the high recall of the system, with most actual threats correctly identified.



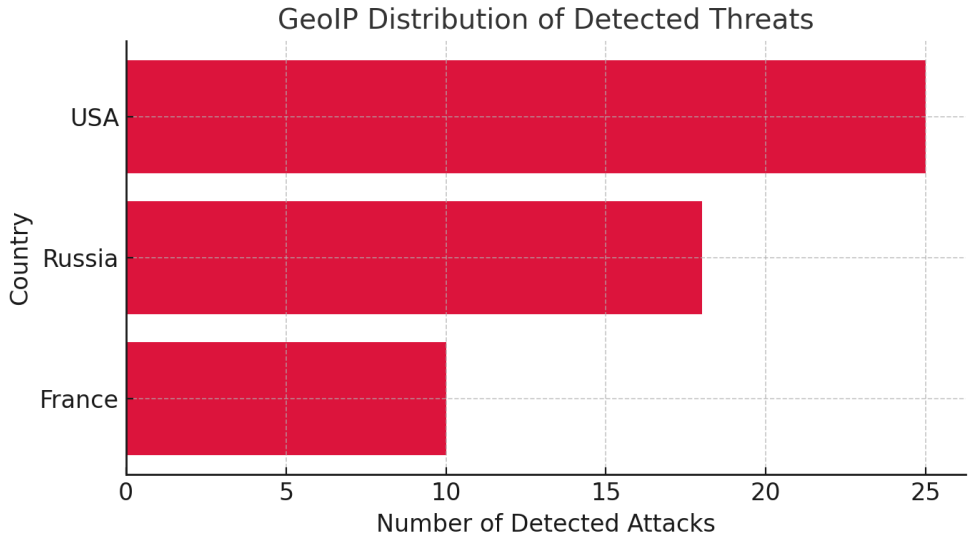
(b) Bar Chart of Threat Categories

- Detected threats were categorized into brute force, phishing, malware, and other defined categories.
- A bar chart summarized the frequency of each category, enabling quick identification of the most common attack types.



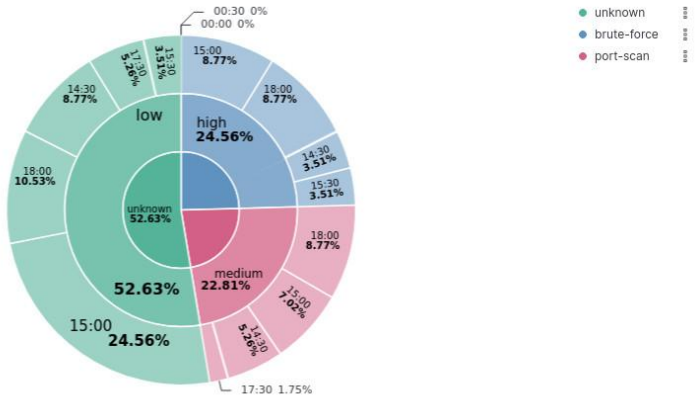
(c) GeoIP Visualization in Kibana

- A Kibana map displayed the geographic distribution of attack sources.
- This highlighted repeated activity from specific regions, offering valuable context for SOC analysts.



(d) Threat Categorization Dashboard

- Kibana dashboards presented categorized logs in real time.
- Filters allowed analysts to drill down by severity level, attack type, or source IP.



(e) Slack Alerts for High-Severity Threats

- Integration with Slack provided instant alerts for high-severity events.
- This ensured analysts were notified immediately, even without actively monitoring the dashboard.

Slack alert notification showing a high-severity SSH brute-force detection.

3. Interpretation

The combination of high recall and real-time visualizations demonstrated that the system was effective at catching threats while reducing the risk of missing critical anomalies. However, the precision metric indicated room for refinement, such as enhancing rule definitions or applying more advanced filtering techniques to minimize false positives.

Overall, the system delivered actionable insights, improved situational awareness, and validated the feasibility of using ELK Stack with enrichment techniques for SOC workflows.

Discussion and Analysis

The results of this project highlight both the strengths and limitations of the ELK Stack Dashboard as a tool for automated threat detection and monitoring.

One of the most significant findings is the system's high recall score of 85.7%, which demonstrates its effectiveness in detecting the majority of actual threats. In the context of a Security Operations Center (SOC), recall is particularly critical: missing a true threat (false negative) can have severe consequences. By capturing most anomalies, the system ensured that malicious activities such as brute-force attempts or lateral movement were far less likely to go unnoticed.

At the same time, the precision score of 60% indicates that the system produced a notable number of false positives. This means that some legitimate activities were incorrectly flagged as threats, which could increase the workload of SOC analysts by requiring them to review non-critical alerts. This trade-off is common in security monitoring, where reducing missed detections often comes at the expense of higher false alarms. Further refinement — such as better rule definitions, more advanced filtering logic, or the integration of machine learning models — could help reduce false positives and improve precision.

The accuracy (73.7%) and F1-score (70.6%) confirm that the system achieved a reasonable balance between detection capability and precision. When combined with enriched dashboards and Slack-based real-time alerts, these results demonstrate that the project successfully delivered a functional and practical monitoring tool that can support SOC operations in real-world environments.

During deployment and testing, several challenges were encountered and addressed through persistence and iterative troubleshooting:

- **False Positives:** Keyword-based categorization occasionally over-flagged benign events.
- **Resource Limitations:** Running Elasticsearch, Logstash, and Kibana simultaneously on limited hardware caused performance bottlenecks, requiring adjustments to maintain real-time log ingestion.
- **Configuration Complexity:** Ensuring consistent Logstash pipelines and maintaining compatibility between components demanded careful configuration and repeated testing.

Despite these challenges, the project demonstrated clear improvements compared to manual log analysis. By automating ingestion, enrichment, visualization, and alerting, the system reduced detection time, minimized human error, and improved overall SOC efficiency.

Conclusion

This internship project represented a significant milestone in my professional development, bringing together technical implementation, problem-solving, and applied cybersecurity practices into a cohesive system. The ELK Stack Dashboard that I designed and deployed successfully automated the end-to-end process of log ingestion, categorization, enrichment, and visualization. What once required time-consuming manual review was transformed into a streamlined, intelligent, and alert-driven monitoring solution aligned with the needs of a modern Security Operations Center (SOC).

The results demonstrated the system's ability to detect threats with a high recall rate of 85.7%, ensuring that the vast majority of anomalies were captured, while maintaining an overall accuracy of 73.7% and an F1-score of 70.6%. These figures validate the robustness of the pipeline and highlight its reliability in prioritizing detection over omission — a critical factor in SOC environments. The dashboards and Slack alerting mechanisms further underscored the system's practical value, offering enriched visualizations and real-time notifications that improved analyst efficiency and situational awareness.

Achieving these results was not without challenges. Throughout the internship, I had to address the complexity of configuring multiple ELK components, optimize pipelines under limited system resources, and reduce false positives caused by broad keyword-based matching. Overcoming these obstacles required persistence, iterative testing, and continuous refinement of the pipeline. These experiences not only improved the system's performance but also strengthened my own problem-solving skills and resilience as a cybersecurity professional.

Beyond the technical achievements, this project gave me hands-on mastery of SOC-focused workflows — from building ingestion pipelines to developing actionable dashboards and integrating automated alerts. It marked a turning point in my internship experience, moving me from theoretical understanding to delivering a fully functional solution with measurable operational impact.

In conclusion, the ELK Stack Dashboard project not only achieved its objectives but also demonstrated my ability to tackle complex challenges, integrate open-source technologies into a unified system, and deliver a solution that enhances cybersecurity monitoring. It stands as one of the most significant steps in my internship journey, reflecting both the technical accomplishments and the dedication required to transform an idea into a practical SOC-ready platform.

Appendices

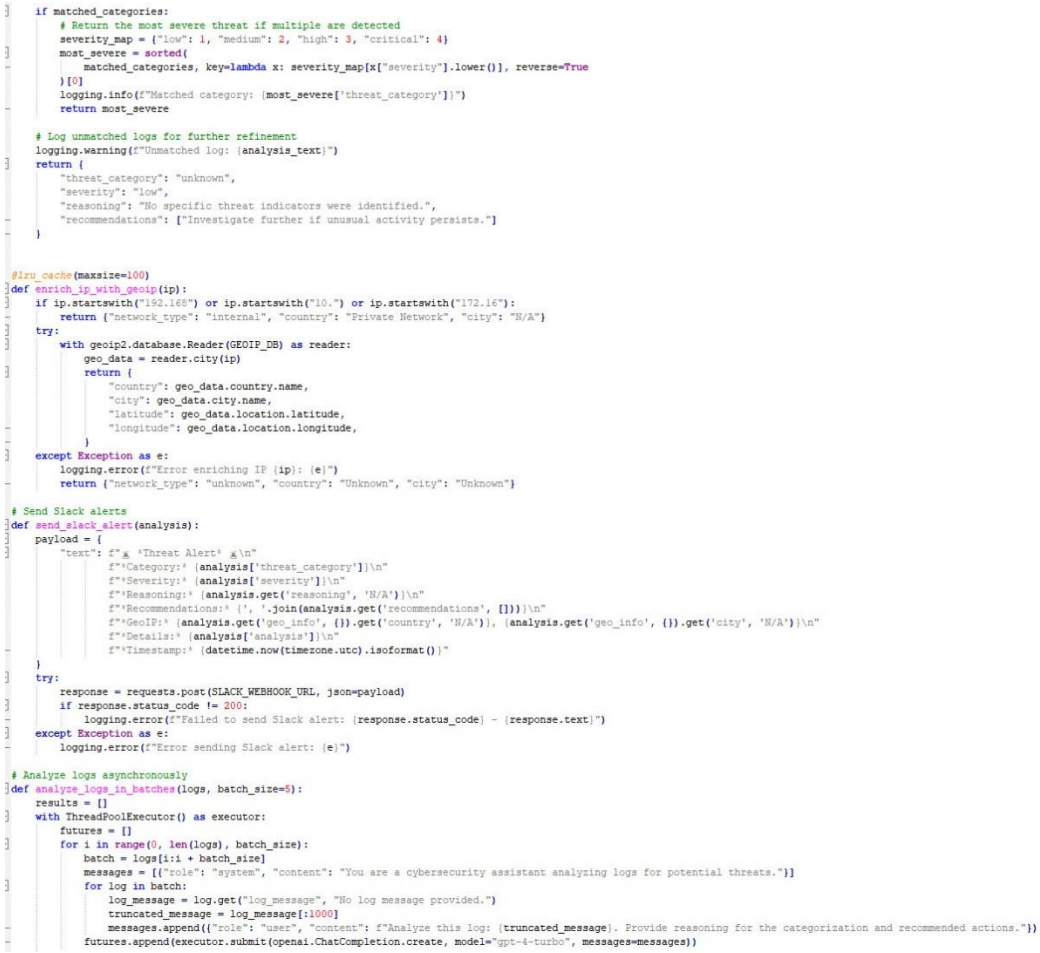
Appendix A – Python Scripts

To support reproducibility and clarity, the main scripts developed during this project are included below. These scripts demonstrate the automation steps for log ingestion, categorization, enrichment, and alerting.

- **Script Part 1:** Initial log fetching and preprocessing from Elasticsearch indices.
- **Script Part 2:** Dynamic categorization of logs using the threat_categories.json configuration file.
- **Script Part 3:** GeoIP enrichment of IP addresses and integration with Slack webhooks for high-severity alerts.

```python
import os
import openai
from elasticsearch import Elasticsearch, helpers
from dotenv import load_dotenv
from datetime import datetime, timezone
import requests
import logging
import re
import geoip2.database
from concurrent.futures import ThreadPoolExecutor
from functools import lru_cache
import json

# Configure logging
logging.basicConfig(
    filename="llm_analyzer.log",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s"
)

# Load environment variables
load_dotenv(dotenv_path="api.env")
openai.api_key = os.getenv("OPENAI_API_KEY")

# Connect to Elasticsearch
es = Elasticsearch([{'host': 'localhost', 'port': 9200, 'scheme': 'http'}])

# Slack webhook for alerts
SLACK_WEBHOOK_URL = "https://hooks.slack.com/services/your/slack/webhook"

# GeoIP database
GEOIP_DB = "GeoLite2-City.mmdb"

# Load threat categories from JSON
def load_threat_categories(file_path="threat_categories.json"):
    try:
        with open(file_path, "r") as f:
            return json.load(f)
    except Exception as e:
        logging.error(f"Error loading threat categories: {e}")
        return {}

THREAT_CATEGORIES = load_threat_categories()

# Fetch logs from Elasticsearch
def fetch_logs(index="syslog-*", size=10):
    try:
        response = es.search(
            index=index,
            body={
                "size": size,
                "query": {"match_all": {}},
                "sort": [{"@timestamp": {"order": "desc"}}]
            }
        )
        logs = [hit["_source"] for hit in response['hits']['hits']]
        logging.info(f"Fetched {len(logs)} logs.")
        return logs
    except Exception as e:
        logging.error(f"Error fetching logs from {index}: {e}")
        return []

# Categorize and explain threats dynamically
# Categorize and explain threats dynamically
def categorize_analysis(analysis_text, log_data):
    matched_categories = []
    for category, details in THREAT_CATEGORIES.items():
        for keyword in details["keywords"]:
            if keyword.lower() in analysis_text.lower():
                matched_categories.append({
                    "threat_category": category,
                    "severity": details["severity"],
                    "reasoning": f"Identified as {category} based on the presence of keyword: {keyword}.",
                    "recommendations": details["recommendations"]
                })
```

The script (part 1)



```
    if matched_categories:
        # Return the most severe threat if multiple are detected
        severity_map = {"low": 1, "medium": 2, "high": 3, "critical": 4}
        most_severe = sorted(
            matched_categories, key=lambda x: severity_map[x["severity"].lower()], reverse=True
        )[0]
        logging.info(f"Matched category: {most_severe['threat_category']}")
        return most_severe

    # Log unmatched logs for further refinement
    logging.warning(f"Unmatched log: {analysis_text}")
    return {
        "threat_category": "unknown",
        "severity": "low",
        "reasoning": "No specific threat indicators were identified.",
        "recommendations": ["Investigate further if unusual activity persists."]
    }


@lru_cache(maxsize=100)
def enrich_ip_with_geoip(ip):
    if ip.startswith("192.168") or ip.startswith("10.") or ip.startswith("172.16"):
        return {"network_type": "internal", "country": "Private Network", "city": "N/A"}
    try:
        with geoip2.database.Reader(GEOIP_DB) as reader:
            geo_data = reader.city(ip)
            return {
                "country": geo_data.country.name,
                "city": geo_data.city.name,
                "latitude": geo_data.location.latitude,
                "longitude": geo_data.location.longitude,
            }
    except Exception as e:
        logging.error(f"Error enriching IP {ip}: {e}")
        return {"network_type": "unknown", "country": "Unknown", "city": "Unknown"}

# Send Slack alerts
def send_slack_alert(analysis):
    payload = {
        "text": f"⚠ *Threat Alert* ⚠\n"
                f"*Category:* {analysis['threat_category']}\n"
                f"*Severity:* {analysis['severity']}\n"
                f"*Reasoning:* {analysis.get('reasoning', 'N/A')}\n"
                f"*Recommendations:* {', '.join(analysis.get('recommendations', []))}\n"
                f"*GeoIP:* {analysis.get('geo_info', {}).get('country', 'N/A')}, {analysis.get('geo_info', {}).get('city', 'N/A')}\n"
                f"*Details:* {analysis['analysis']}\n"
                f"*Timestamp:* {datetime.now(timezone.utc).isoformat()}"
    }
    try:
        response = requests.post(SLACK_WEBHOOK_URL, json=payload)
        if response.status_code != 200:
            logging.error(f"Failed to send Slack alert: {response.status_code} - {response.text}")
    except Exception as e:
        logging.error(f"Error sending Slack alert: {e}")

# Analyze logs asynchronously
def analyze_logs_in_batches(logs, batch_size=5):
    results = []
    with ThreadPoolExecutor() as executor:
        futures = []
        for i in range(0, len(logs), batch_size):
            batch = logs[i:i + batch_size]
            messages = [{"role": "system", "content": "You are a cybersecurity assistant analyzing logs for potential threats."}]
            for log in batch:
                log_message = log.get("log_message", "No log message provided.")
                truncated_message = log_message[:1000]
                messages.append({"role": "user", "content": f"Analyze this log: {truncated_message}. Provide reasoning for the categorization and recommended actions."})
            futures.append(executor.submit(openai.ChatCompletion.create, model="gpt-4-turbo", messages=messages))
```

The script (part 2)

```
        for future in futures:
            try:
                response = future.result()
                if 'choices' in response and response['choices']:
                    analysis = response['choices'][0]['message']['content']
                    parsed_log = {"ip": "127.0.0.1"}  # Example IP for enrichment
                    geo_info = enrich_ip_with_geoip(parsed_log.get("ip", ""))
                    categorized_result = categorize_analysis(analysis, parsed_log)
                    results.append({
                        "analysis": analysis,
                        "geo_info": geo_info,
                        **categorized_result
                    })
            except Exception as e:
                logging.error(f"Error analyzing logs: {e}")
                results.append({
                    "analysis": "Analysis failed.",
                    "threat_category": "unknown",
                    "severity": "low"
                })
    return results

# Push analyses to Elasticsearch
def push_analysis_to_elasticsearch_bulk(analyses, index="threat-analysis-new"):
    actions = [
        {
            "_index": index,
            "_source": {
                "analysis": result["analysis"],
                "threat_category": result["threat_category"],
                "severity": result["severity"],
                "reasoning": result.get("reasoning", ""),
                "recommendations": result.get("recommendations", []),
                "geo_info": result.get("geo_info", {}),
                "@timestamp": datetime.now(timezone.utc).isoformat()
            }
        }
        for result in analyses
    ]
    try:
        helpers.bulk(es, actions)
        logging.info(f"Successfully pushed {len(actions)} analyses to index: {index}")
    except Exception as e:
        logging.error(f"Error pushing analyses to Elasticsearch index {index}: {e}")

# Main function
if __name__ == "__main__":
    logs = fetch_logs(index="syslog-*", size=20)
    if logs:
        analyses = analyze_logs_in_batches(logs)
        push_analysis_to_elasticsearch_bulk(analyses)
    else:
        logging.info("No logs found.")
```

The script (part 3)

Appendix B – JSON Threat Categories Configuration

The JSON configuration file was used to define keywords, map them to threat categories, and assign severity levels. An example entry is shown below:

```
{
    "brute-force": {
        "keywords": ["brute force", "multiple failed login", "password guessing"],
        "severity": "high",
        "recommendations": ["Block IP", "Enable account lockout", "Implement MFA"]
    },
    "port-scan": {
        "keywords": ["port scan", "reconnaissance", "probing multiple ports", "nmap scan"],
        "severity": "medium",
        "recommendations": ["Update firewall rules", "Block IP", "Monitor traffic"]
    },
    "sql-injection": {
        "keywords": ["sql injection", "database error", "sql syntax", "1=1", "SELECT", "UNION ALL SELECT"],
        "severity": "critical",
        "recommendations": ["Sanitize inputs", "Use prepared statements", "Enable WAF"]
    },
    "xss": {
        "keywords": ["xss", "cross-site scripting", "<script>", "javascript:", "alert("],
        "severity": "critical",
        "recommendations": ["Validate inputs", "Encode outputs", "Use CSP headers"]
    },
    "malware": {
        "keywords": ["malware", "trojan", "ransomware", "virus", "malicious file", "backdoor"],
        "severity": "critical",
        "recommendations": ["Quarantine affected systems", "Update antivirus", "Investigate source"]
    },
    "privilege-escalation": {
        "keywords": ["privilege escalation", "sudo exploit", "elevated rights", "kernel exploit"],
        "severity": "critical",
        "recommendations": ["Patch systems", "Restrict sudo access", "Enable alerts for privilege escalation attempts"]
    },
    "denial-of-service": {
        "keywords": ["denial of service", "dos attack", "service unavailable", "flooding"],
        "severity": "critical",
        "recommendations": ["Implement rate limiting", "Use a WAF", "Monitor server resources"]
    },
    "phishing": {
        "keywords": ["phishing", "fake login page", "social engineering", "credential harvesting"],
        "severity": "high",
        "recommendations": ["Educate users", "Enable email filters", "Block malicious URLs"]
    },
    "ransomware": {
        "keywords": ["ransom note", "encryption detected", "data exfiltration", "ransom demand"],
        "severity": "critical",
        "recommendations": ["Disconnect the affected system", "Restore backups", "Investigate network activity"]
    }
}
```


Example of the threat_categories.json

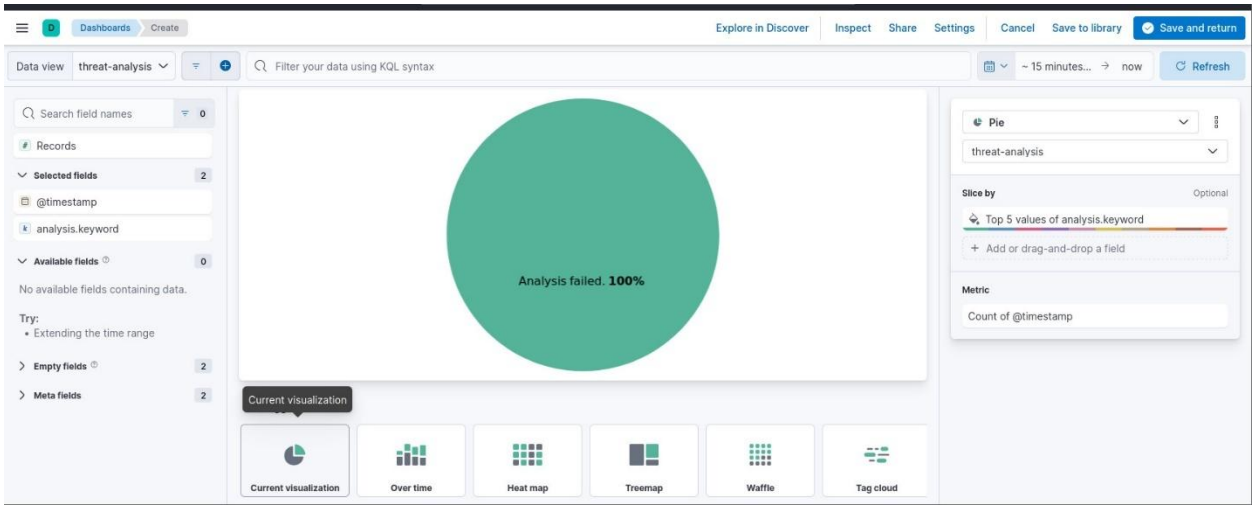
This dynamic configuration approach allowed easy updates to threat categories without modifying the core scripts.

Appendix C – Additional Screenshots

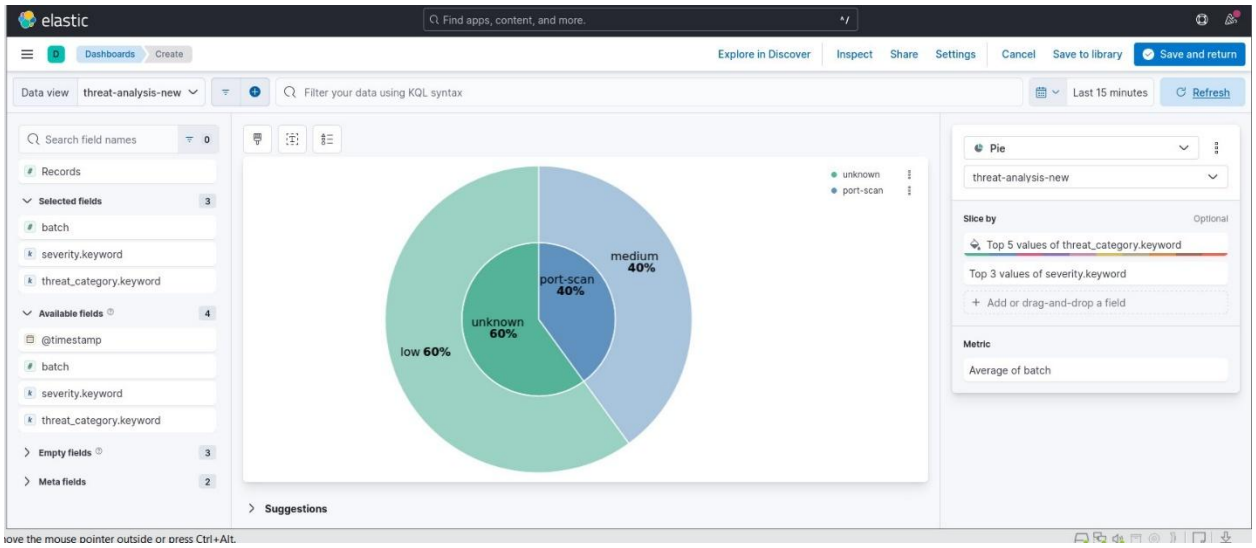
The following screenshots illustrate the outputs and dashboards created during the project:

1. **GeoIP Dashboard** – Map-based visualization of detected threats by geographic source.
2. **Threat Categorization Dashboard** – Real-time categorization of log events by type and severity.
3. **Slack Alerts** – Examples of automated notifications for high-severity incidents.

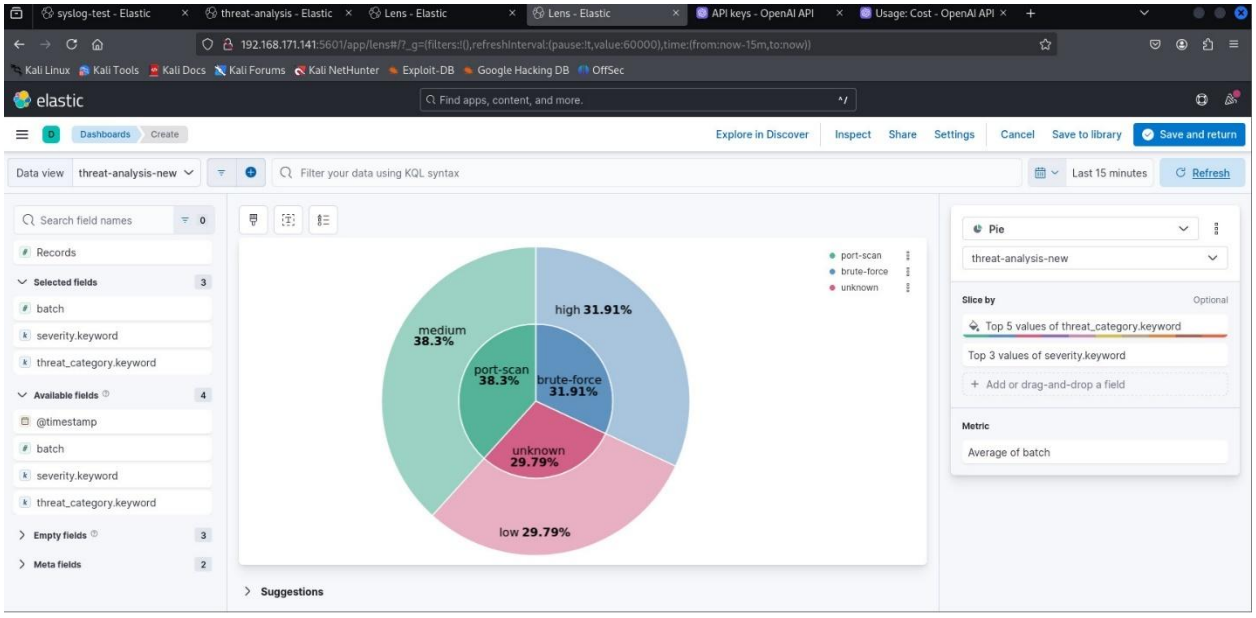
4. **Iterative Testing Snapshots** – Progression of system accuracy and refinement during the internship.



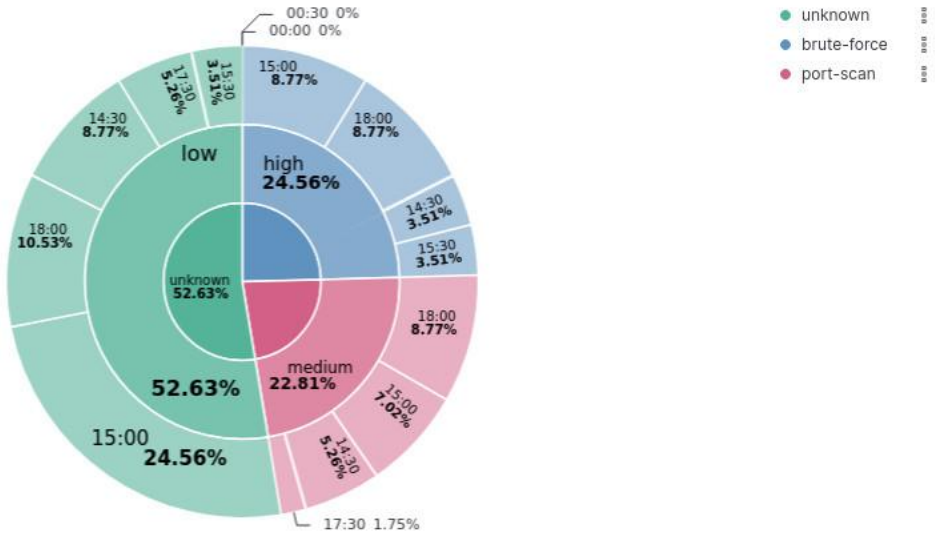
First attempt



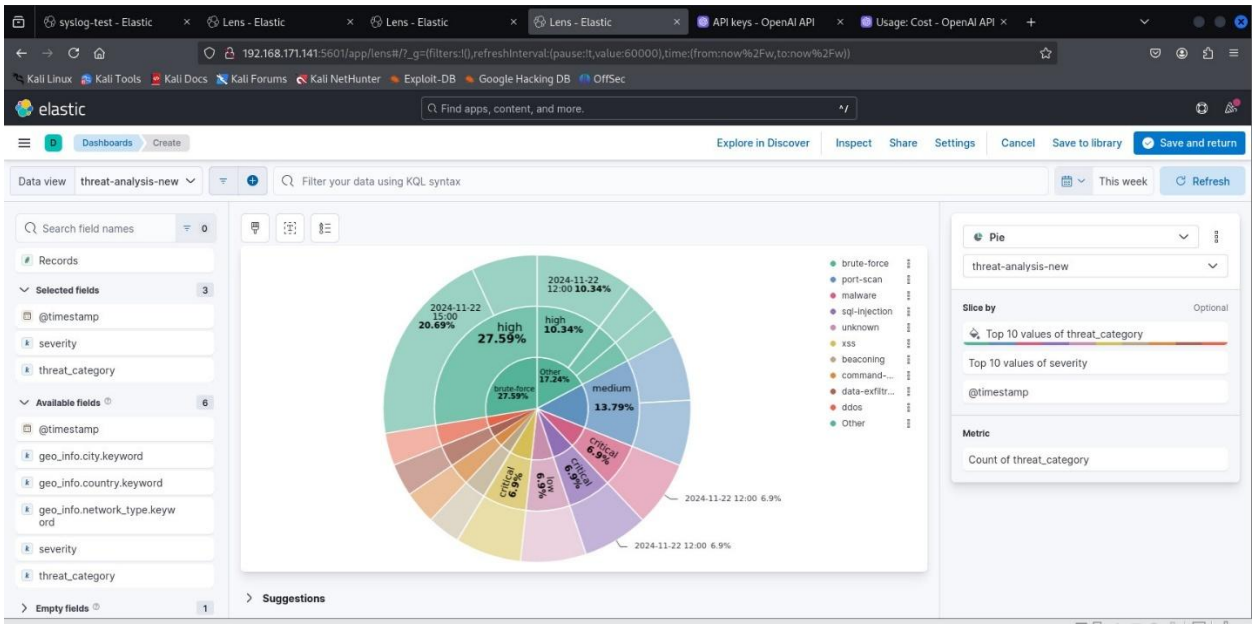
Start to give results



Improving the results



Testing during the time



Trying different types of threats