Management and Analysis of Security Logs via a SIEM for a Web Server

Dardouri Meryem & Mossadeq Boutaina

Euromed University of Fez

January 20, 2025

Contents

1 Introduction

1.1 Context

In today's digital landscape, web servers are critical components of most organizations' IT infrastructure. However, they are also prime targets for cyberattacks, including SQL injections, cross-site scripting (XSS), brute force attacks, and unauthorized access attempts. To mitigate these risks, it is essential to implement robust security measures and continuously monitor server activity.

This project focuses on deploying a SIEM (Security Information and Event Management) solution using the ELK stack (Elasticsearch, Logstash, Kibana) to collect, analyze, and visualize logs from an Apache web server. The server is secured using SELinux to enforce strict access controls. The goal is to detect and respond to security incidents in real time.

1.2 Problem Statement

Web servers are vulnerable to a wide range of cyber threats. Without proper monitoring and analysis, security incidents can go unnoticed, leading to data breaches, service disruptions, and compliance violations.

1.3 Objectives

The primary objectives of this project are:

- Install and configure a secure Apache web server with SELinux.
- Deploy the ELK stack for centralized log management and analysis.
- Configure Filebeat and Auditbeat to collect logs and forward them to Logstash.
- Analyze logs in Kibana to identify suspicious activities.
- Set up alerts to notify administrators of potential security incidents using a Python script.
- Develop a process for responding to detected threats using a Python script.

2 Installation and Configuration

2.1 System Architecture

The following diagram illustrates the architecture of the SIEM solution:

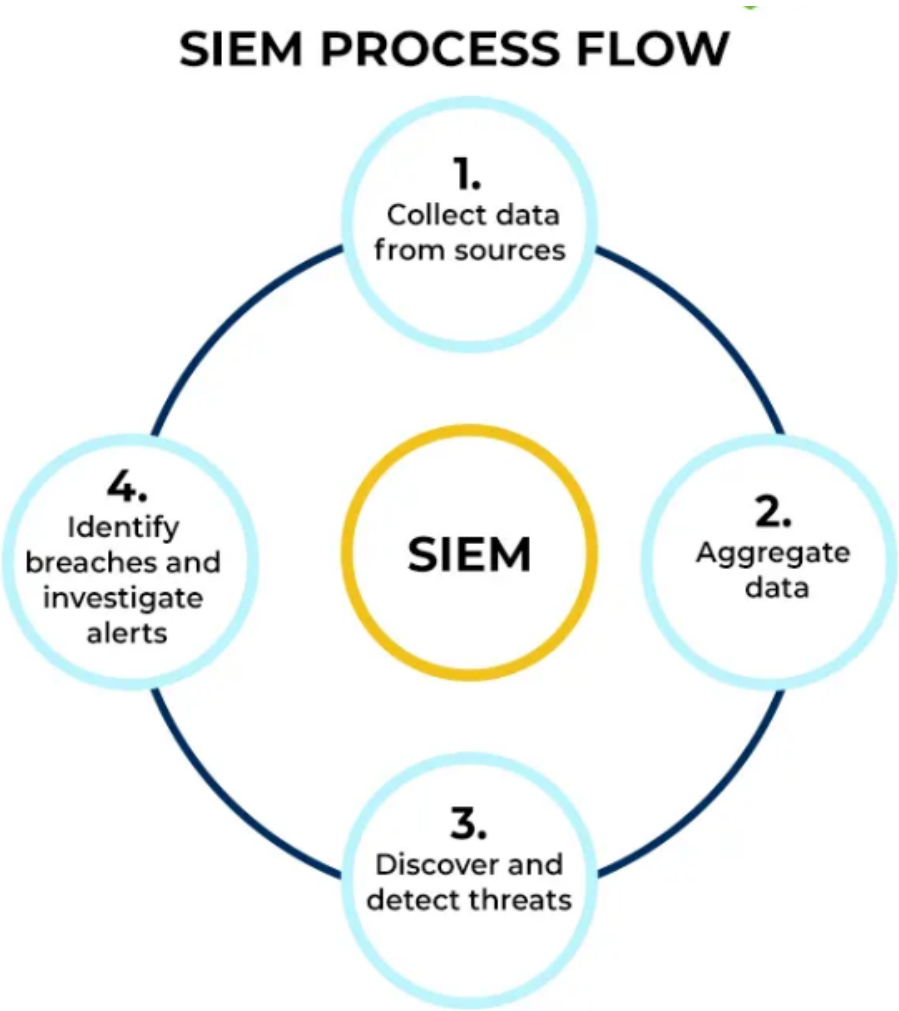


Figure 1: System Architecture of the SIEM Solution

2.2 Installation of the Apache Web Server

Apache is one of the most widely used web servers due to its flexibility, performance, and extensive documentation. Here's how to install and configure it:

```
sudo apt update
sudo apt install apache2 -y
sudo systemctl enable --now apache2
```

Listing 1: Installing Apache

2.2.1 Activation of Apache

After installation, verify that Apache is running:

```
sudo systemctl status apache2
```

Listing 2: Verifying Apache Status

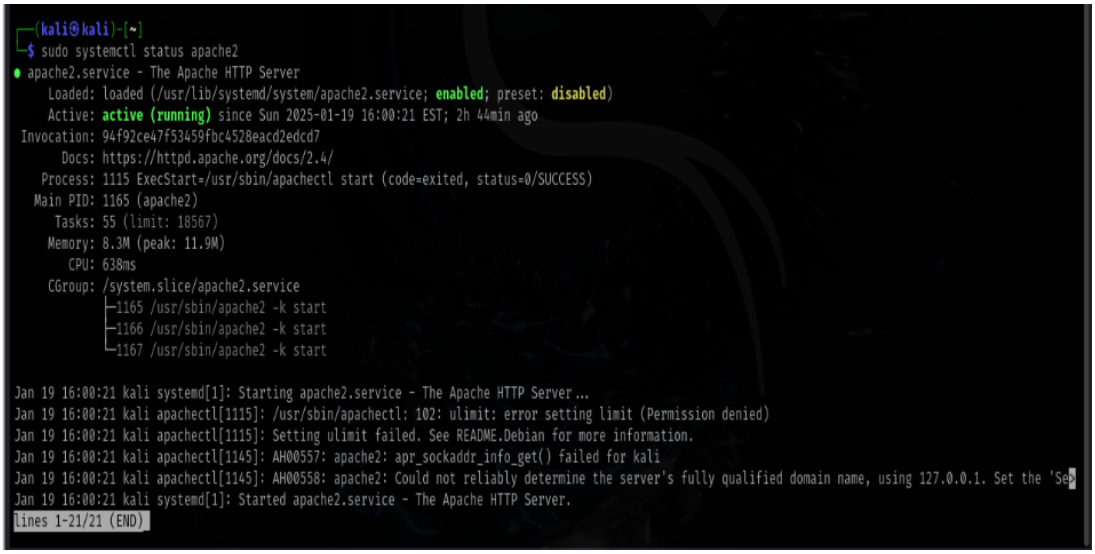


Figure 2: Apache Service Status

2.3 Configuration of SELinux for Apache

SELinux can significantly enhance the security of an Apache web server by restricting its actions. Here's how to configure SELinux:

```
sudo apt install selinux-basics selinux-policy-default auditd -y
sudo selinux-activate
sudo reboot
```

Listing 3: Enabling SELinux

2.3.1 Verification of SELinux Status

After enabling SELinux, verify its status:

```
sestatus
```

Listing 4: Verifying SELinux Status

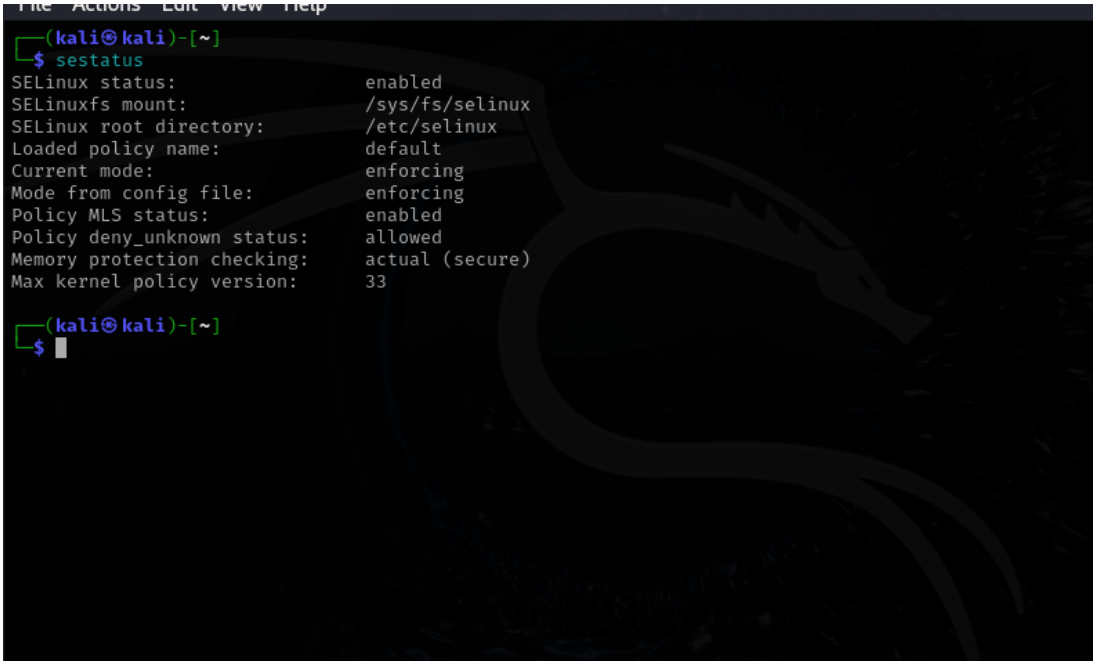


Figure 3: SELinux Status

2.4 Installation of the ELK Stack

The ELK stack can be installed directly on a Linux system. Below are the steps for installation.

2.4.1 Install Elasticsearch

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt
    -key add -
sudo apt-get install apt-transport-https
echo "deb␣https://artifacts.elastic.co/packages/8.x/apt␣stable␣main" |
    sudo tee -a /etc/apt/sources.list.d/elastic-8.x.list
sudo apt-get update && sudo apt-get install elasticsearch
sudo systemctl enable elasticsearch
sudo systemctl start elasticsearch
```

Listing 5: Installing Elasticsearch

2.4.2 Verification of Elasticsearch

After installation, verify that Elasticsearch is running:

```
curl -X GET "localhost:9200"
```

Listing 6: Verifying Elasticsearch Status

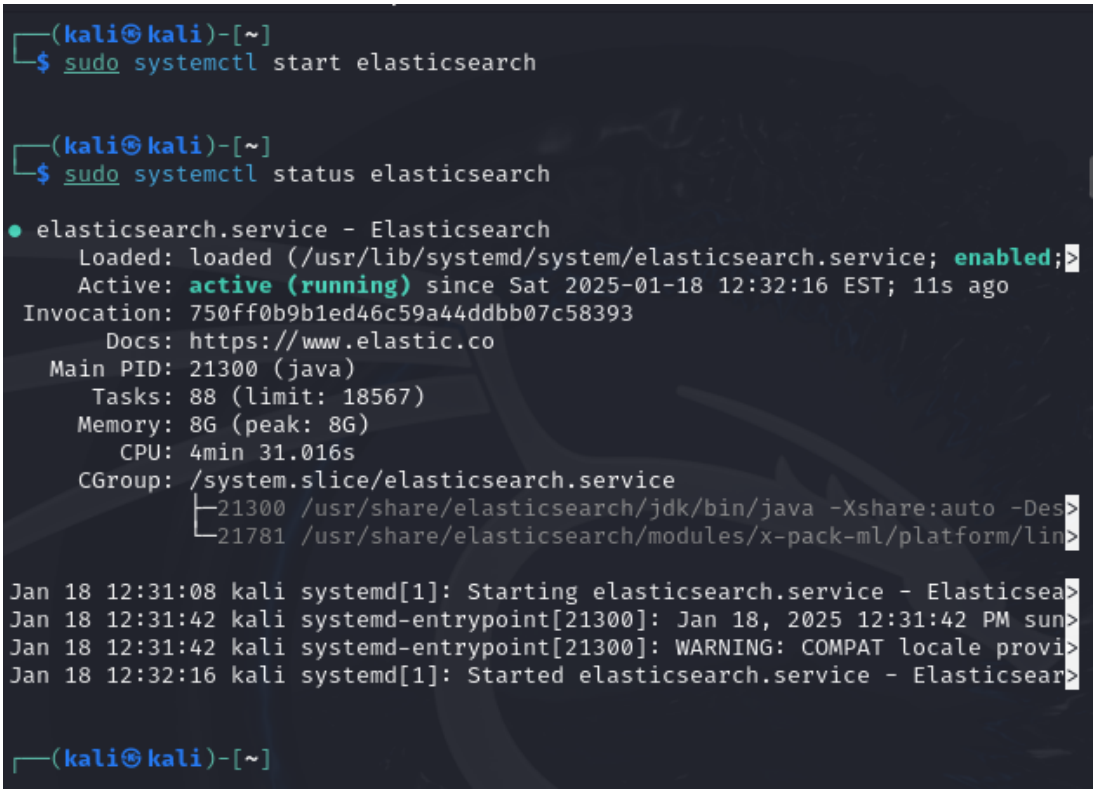


Figure 4: Elasticsearch Status

2.4.3 Install Logstash

```
sudo apt-get install logstash
sudo systemctl enable logstash
sudo systemctl start logstash
```

Listing 7: Installing Logstash

2.4.4 Verification of Logstash

After installation, verify that Logstash is running:

```
sudo systemctl status logstash
```

Listing 8: Verifying Logstash Status

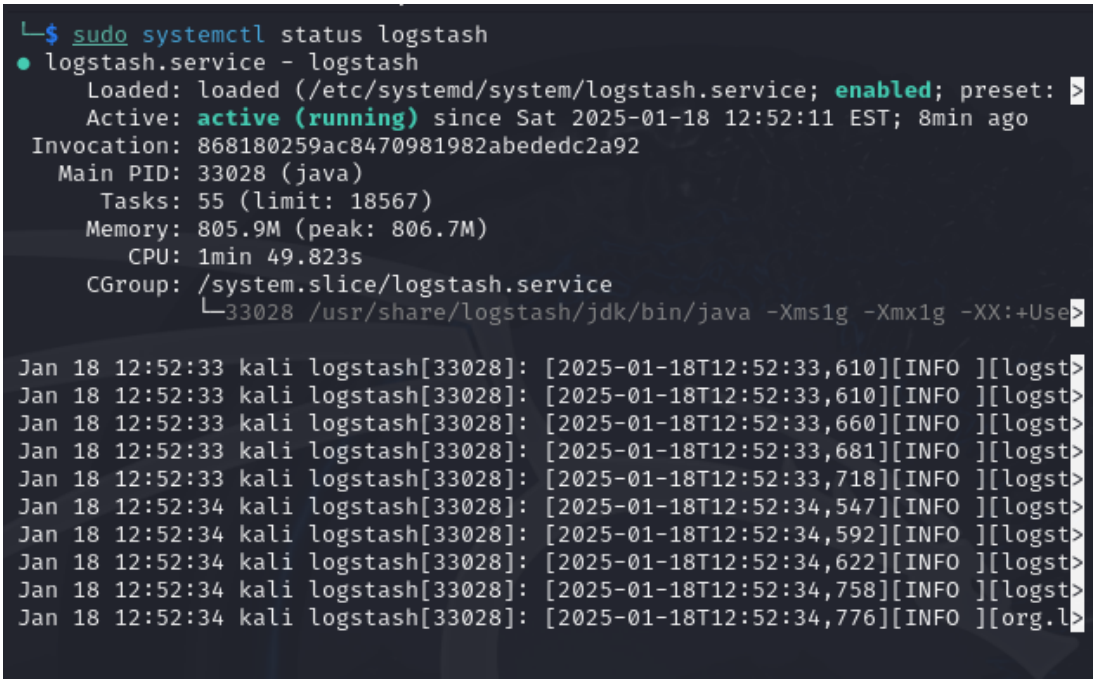


Figure 5: Logstash Status

2.4.5 Install Kibana

```
sudo apt-get install kibana
sudo systemctl enable kibana
sudo systemctl start kibana
```

Listing 9: Installing Kibana

2.4.6 Verification of Kibana

After installation, verify that Kibana is running:

```
sudo systemctl status kibana
```

Listing 10: Verifying Kibana Status

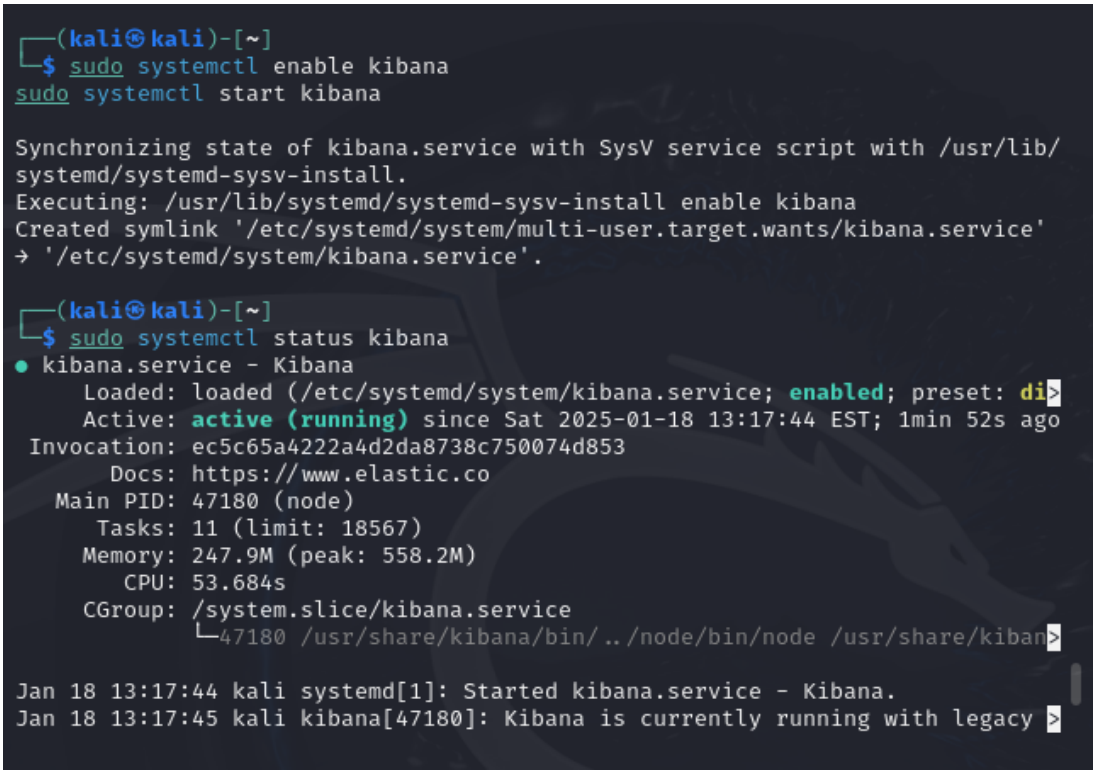


Figure 6: Kibana Status

2.5 Installation of Filebeat and Auditbeat

Filebeat and Auditbeat are lightweight log shippers that collect logs and send them to Logstash or Elasticsearch. Here's how to install and configure them:

2.5.1 Install Filebeat

```
sudo apt-get install filebeat
sudo systemctl enable filebeat
sudo systemctl start filebeat
```

Listing 11: Installing Filebeat

2.5.2 Install Auditbeat

```
sudo apt-get install auditbeat
sudo systemctl enable auditbeat
sudo systemctl start auditbeat
```

Listing 12: Installing Auditbeat

2.5.3 Verification of Filebeat and Auditbeat

After installation, verify that Filebeat and Auditbeat are running:

```
sudo systemctl status filebeat
sudo systemctl status auditbeat
```

Listing 13: Verifying Filebeat and Auditbeat Status

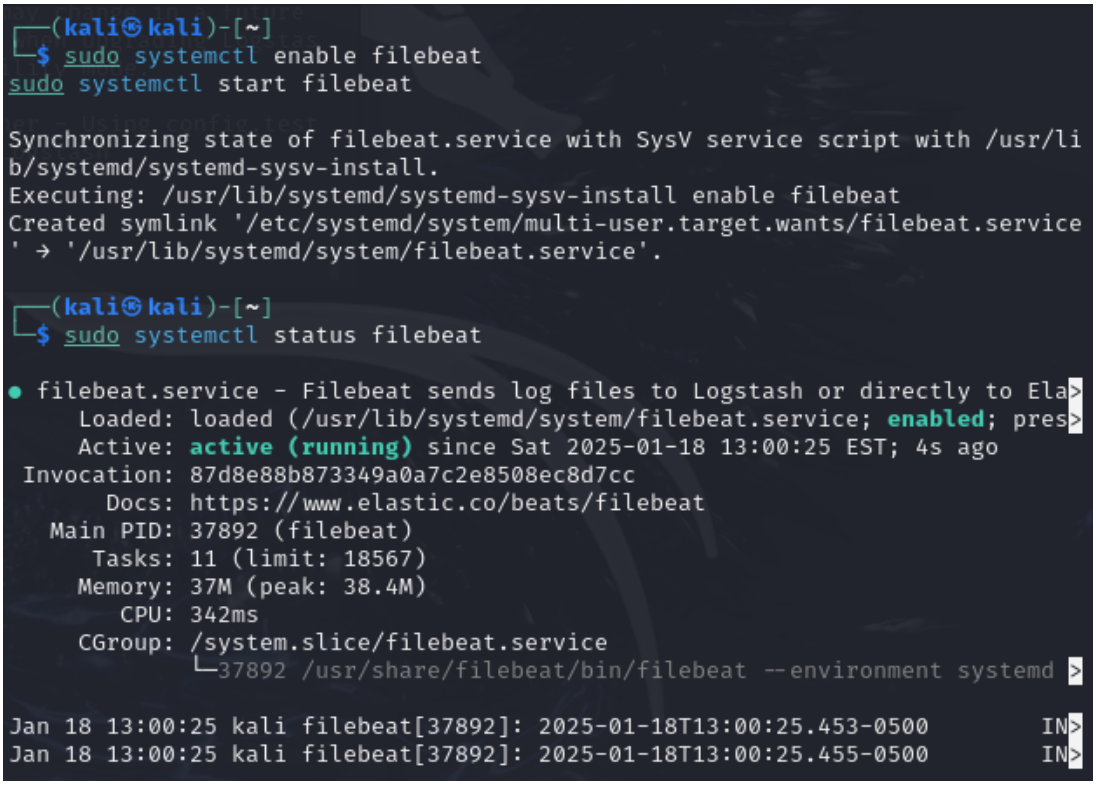


Figure 7: Filebeat Status

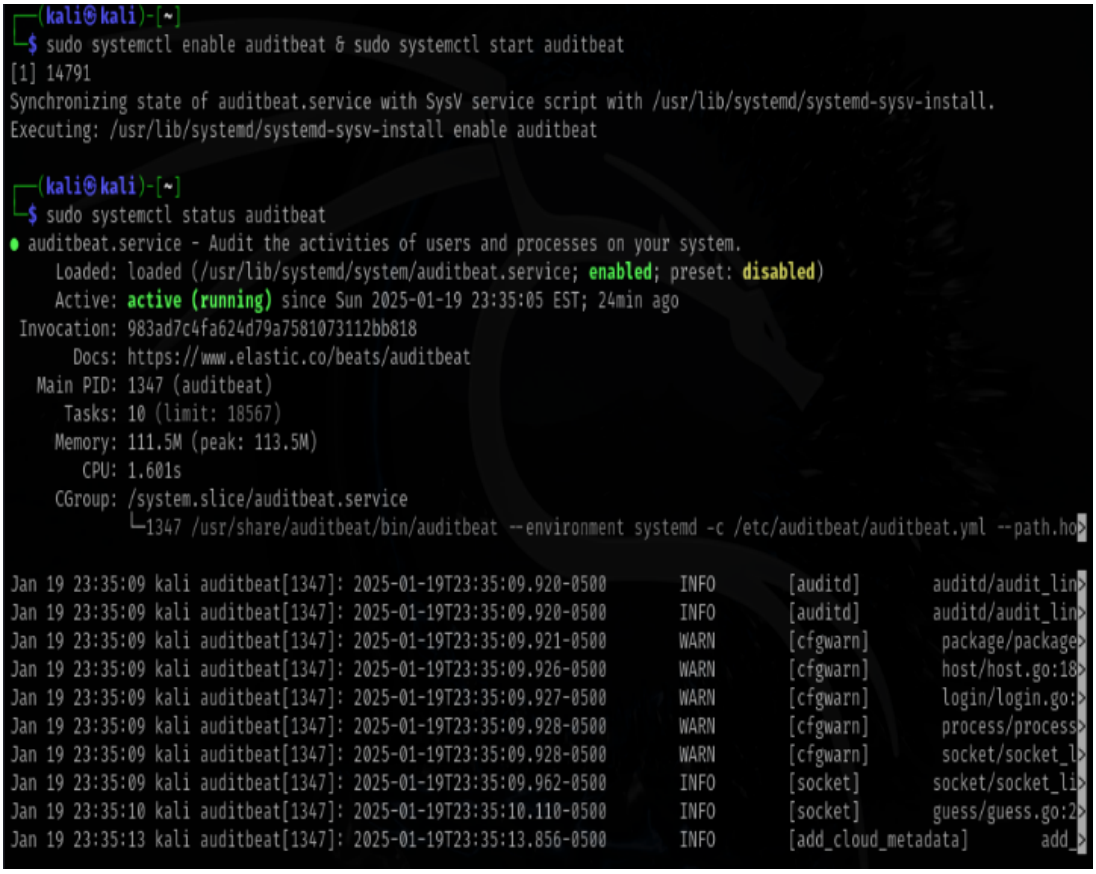


Figure 8: Auditbeat Status

2.6 Configuration of Filebeat and Auditbeat

Filebeat and Auditbeat need to be configured to collect logs and forward them to Logstash or Elasticsearch.

2.6.1 Filebeat Configuration

```
filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /var/log/apache2/access.log
      - /var/log/apache2/error.log
output.logstash:
  hosts: ["localhost:5044"]
```

Listing 14: Filebeat Configuration

2.6.2 Auditbeat Configuration

```
auditbeat.modules:
  - module: auditd
    audit_rule_files: [ '${path.config}/audit.rules.d/*.conf' ]
```

```
output.logstash:
  hosts: ["localhost:5044"]
```

Listing 15: Auditbeat Configuration

2.7 Configuration of Logstash

Logstash processes logs before sending them to Elasticsearch. Here's a sample configuration:

NB: you can find the entire configuration file below in the apendencies

```
input {
  beats {
    port => 5044
  }
}
filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }
}
output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "apache-logs-%{+YYYY.MM.dd}"
  }
}
```

Listing 16: Logstash Configuration

3 Log Analysis

3.1 Visualization in Kibana

Kibana provides a user-friendly interface for exploring and analyzing logs. Here's how to get started:

- Access Kibana at `http://localhost:5601`.
- Create an index pattern for Apache logs (e.g., 'apache-logs-*').
- Use the Discover tab to search and filter logs.
- Create visualizations (e.g., bar charts, pie charts) to identify trends and anomalies.

4 Error Handling

During the installation and configuration process, several errors were encountered. Below are the most common issues and their solutions.

4.1 Apache Fails to Start Due to Port Conflict

4.1.1 Error

Apache fails to start because another service is using port 80.

4.1.2 Solution

Change the Apache port to an available port (e.g., 8080):

```
sudo nano /etc/apache2/ports.conf
# Change "Listen 80" to "Listen 8080"
sudo systemctl restart apache2
```

Listing 17: Changing Apache Port

4.2 Firefox and Elasticsearch Not Working After SELinux Activation

4.2.1 Error

After enabling SELinux, Firefox and Elasticsearch fail to start due to restrictive policies.

4.2.2 Solution

Create and apply SELinux policies to allow Firefox and Elasticsearch to function:

```
sudo audit2allow -a -M firefox_policy
sudo semodule -i firefox_policy.pp

sudo audit2allow -a -M elasticsearch_policy
sudo semodule -i elasticsearch_policy.pp
```

Listing 18: Creating SELinux Policies

4.3 Alerts and Incident Response Not Functioning in Kibana

4.3.1 Error

Alerts and incident response features in Kibana are not working as expected.

4.3.2 Solution

Implement a Python script to handle alerts and incident response:

```
import smtplib
from email.mime.text import MIMEText
import subprocess

def send_alert(email, subject, message):
    msg = MIMEText(message)
    msg['Subject'] = subject
    msg['From'] = 'noreply@example.com'
```

```
    msg['To'] = email

    with smtplib.SMTP('localhost') as server:
        server.sendmail('noreply@example.com', [email], msg.as_string())

def block_ip(ip_address):
    subprocess.run(['iptables', '-A', 'INPUT', '-s', ip_address, '-j', '
        DROP'])

# Example usage
send_alert('admin@example.com', 'Security␣Alert', 'Suspicious␣activity␣
    detected!')
block_ip('192.168.1.100')
```

Listing 19: Python Script for Alerts and Incident Response

5 Results

5.1 Apache Logs

The following figure shows an example of Apache logs visualized in Kibana:

Figure 9: Example of Apache Logs

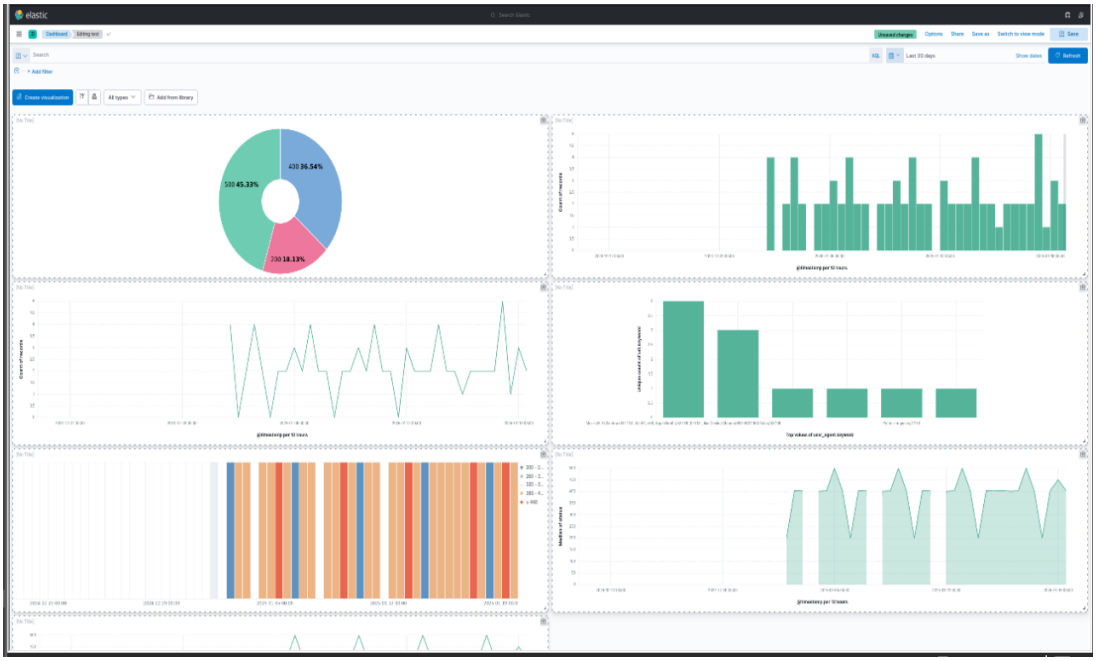
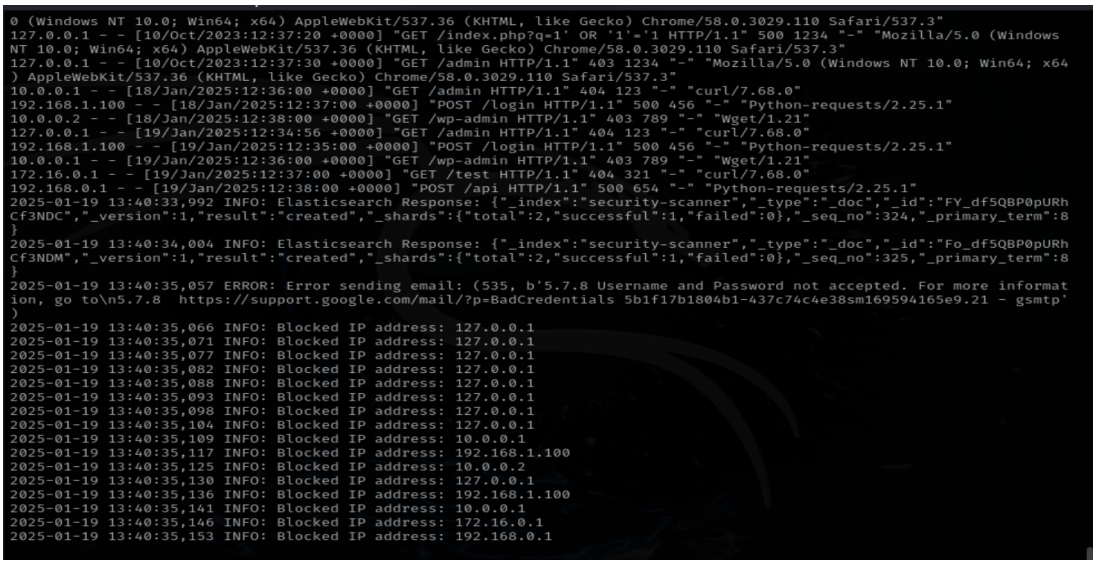


Figure 10: Apache Logs in Kibana

5.2 Security Alerts

The following figure shows an example of security alerts generated by the python script:



```
0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3"
127.0.0.1 - - [10/Oct/2023:12:37:20 +0000] "GET /index.php?q=1' OR '1'='1 HTTP/1.1" 500 1234 "-" "Mozilla/5.0 (Windows
NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3"
127.0.0.1 - - [10/Oct/2023:12:37:30 +0000] "GET /admin HTTP/1.1" 403 1234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64
) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3"
10.0.0.1 - - [18/Jan/2025:12:36:00 +0000] "GET /admin HTTP/1.1" 404 123 "-" "curl/7.68.0"
192.168.1.100 - - [18/Jan/2025:12:37:00 +0000] "POST /login HTTP/1.1" 500 456 "-" "Python-requests/2.25.1"
10.0.0.2 - - [18/Jan/2025:12:38:00 +0000] "GET /wp-admin HTTP/1.1" 403 789 "-" "Wget/1.21"
127.0.0.1 - - [19/Jan/2025:12:34:56 +0000] "GET /admin HTTP/1.1" 404 123 "-" "curl/7.68.0"
192.168.1.100 - - [19/Jan/2025:12:35:00 +0000] "POST /login HTTP/1.1" 500 456 "-" "Python-requests/2.25.1"
10.0.0.1 - - [19/Jan/2025:12:36:00 +0000] "GET /wp-admin HTTP/1.1" 403 789 "-" "Wget/1.21"
172.16.0.1 - - [19/Jan/2025:12:37:00 +0000] "GET /test HTTP/1.1" 404 321 "-" "curl/7.68.0"
192.168.0.1 - - [19/Jan/2025:12:38:00 +0000] "POST /api HTTP/1.1" 500 654 "-" "Python-requests/2.25.1"
2025-01-19 13:40:33,992 INFO: Elasticsearch Response: {"_index":"security-scanner","_type":"_doc","_id":"FY_df5QBP0pURh
Cf3NDC","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":324,"_primary_term":8
}
2025-01-19 13:40:34,004 INFO: Elasticsearch Response: {"_index":"security-scanner","_type":"_doc","_id":"Fo_df5QBP0pURh
Cf3NDM","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":325,"_primary_term":8
}
2025-01-19 13:40:35,057 ERROR: Error sending email: (535, b'5.7.8 Username and Password not accepted. For more informat
ion, go to\n5.7.8  https://support.google.com/mail/?p=BadCredentials 5b1f17b1804b1-437c74c4e38sm169594165e9.21 - gsmtp'
)
2025-01-19 13:40:35,066 INFO: Blocked IP address: 127.0.0.1
2025-01-19 13:40:35,071 INFO: Blocked IP address: 127.0.0.1
2025-01-19 13:40:35,077 INFO: Blocked IP address: 127.0.0.1
2025-01-19 13:40:35,082 INFO: Blocked IP address: 127.0.0.1
2025-01-19 13:40:35,088 INFO: Blocked IP address: 127.0.0.1
2025-01-19 13:40:35,093 INFO: Blocked IP address: 127.0.0.1
2025-01-19 13:40:35,098 INFO: Blocked IP address: 127.0.0.1
2025-01-19 13:40:35,104 INFO: Blocked IP address: 127.0.0.1
2025-01-19 13:40:35,109 INFO: Blocked IP address: 10.0.0.1
2025-01-19 13:40:35,117 INFO: Blocked IP address: 192.168.1.100
2025-01-19 13:40:35,125 INFO: Blocked IP address: 10.0.0.2
2025-01-19 13:40:35,130 INFO: Blocked IP address: 127.0.0.1
2025-01-19 13:40:35,136 INFO: Blocked IP address: 192.168.1.100
2025-01-19 13:40:35,141 INFO: Blocked IP address: 10.0.0.1
2025-01-19 13:40:35,146 INFO: Blocked IP address: 172.16.0.1
2025-01-19 13:40:35,153 INFO: Blocked IP address: 192.168.0.1
```

Figure 11: Example of Security Alerts

5.3 Unrealized Objectives

Two objectives were not fully realized in this project:

- Set up alerts to notify administrators of potential security incidents: - Alerts were implemented using a Python script instead of Kibana. - The script monitors logs

and sends email notifications when suspicious activities are detected.

- Development of a process for responding to detected threats: - A response process was implemented using a Python script. - The script automatically blocks IP addresses associated with suspicious activities using ‘iptables‘.

6 Future Improvements

While the current implementation provides a robust solution for log management and security monitoring, there are several areas for improvement:

6.1 Integration with Other Tools

- **Suricata or Snort**: Integrate with network intrusion detection systems (NIDS) like Suricata or Snort to monitor network traffic for malicious activity.
- **Wazuh**: We attempted to integrate Wazuh for endpoint security monitoring. However, due to conflicts with Kibana, we were unable to proceed with its implementation. Below are the steps we followed for installation and configuration:

```
# Add the Wazuh repository
curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | sudo apt
    -key add -
echo "deb␣https://packages.wazuh.com/4.x/apt/␣stable␣main" |
    sudo tee /etc/apt/sources.list.d/wazuh.list
sudo apt update

# Install Wazuh manager
sudo apt install wazuh-manager
sudo systemctl enable wazuh-manager
sudo systemctl start wazuh-manager

# Install Wazuh agent
sudo apt install wazuh-agent
sudo systemctl enable wazuh-agent
sudo systemctl start wazuh-agent
```

Listing 20: Installing Wazuh

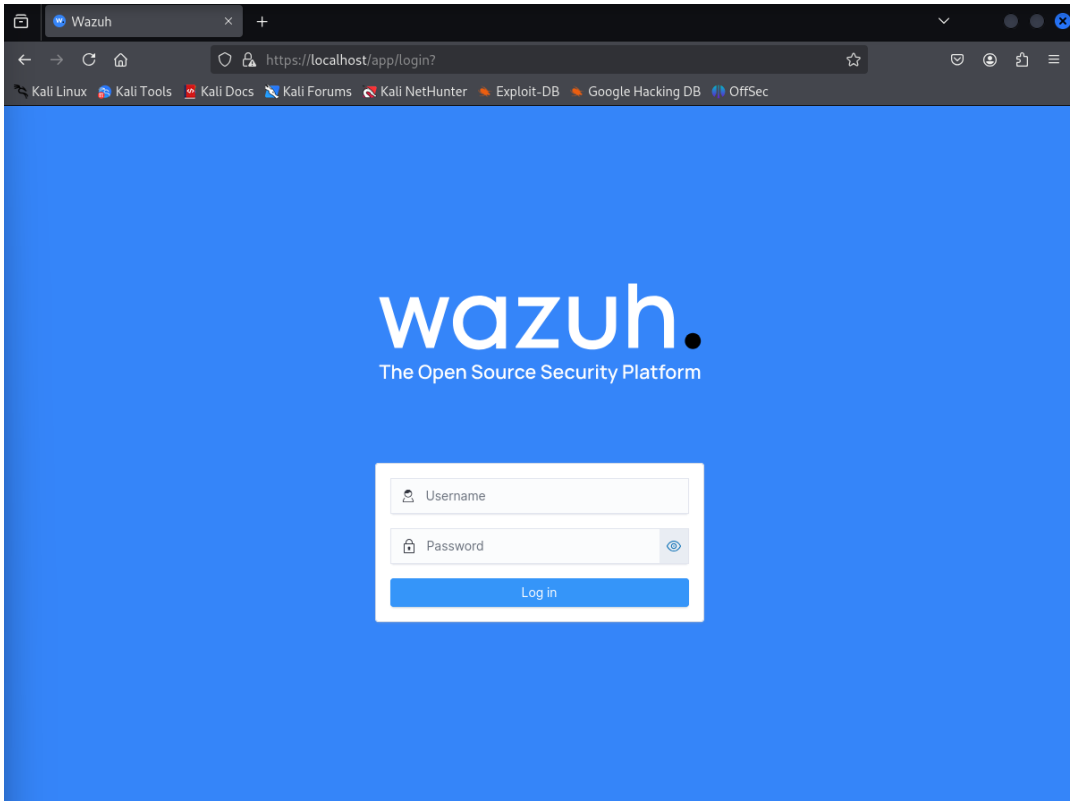


Figure 12: Wazuh Login

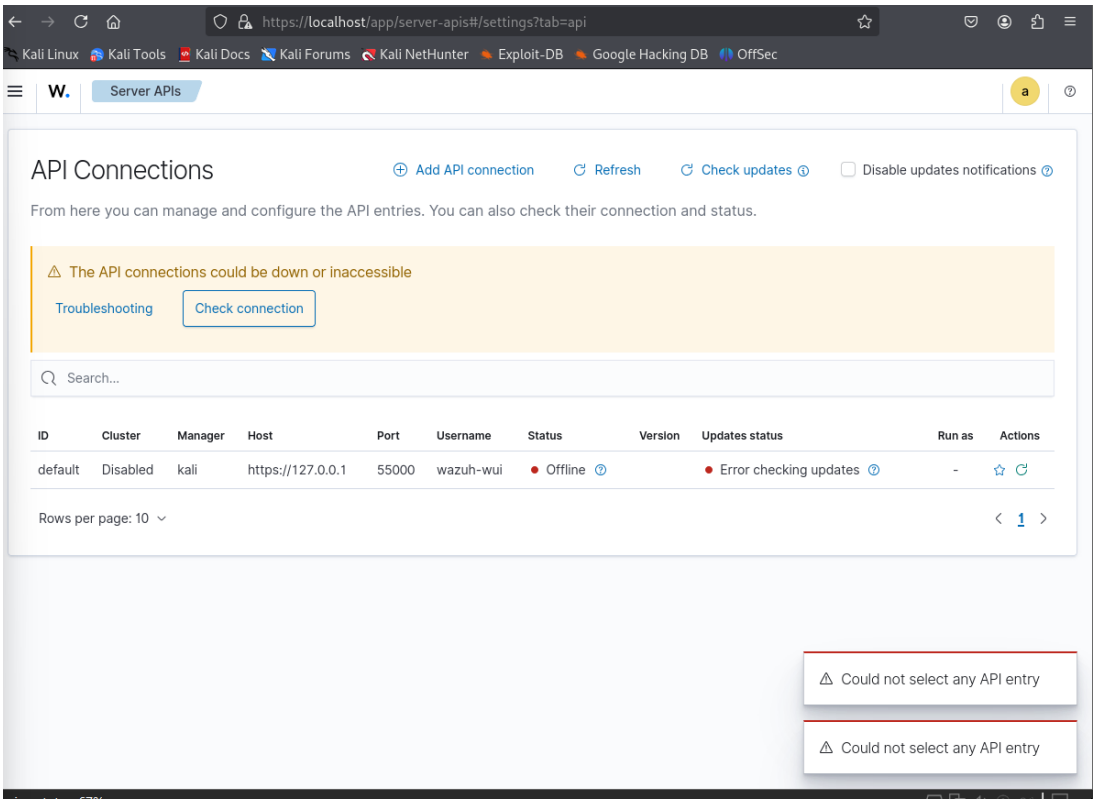


Figure 13: Wazuh interface

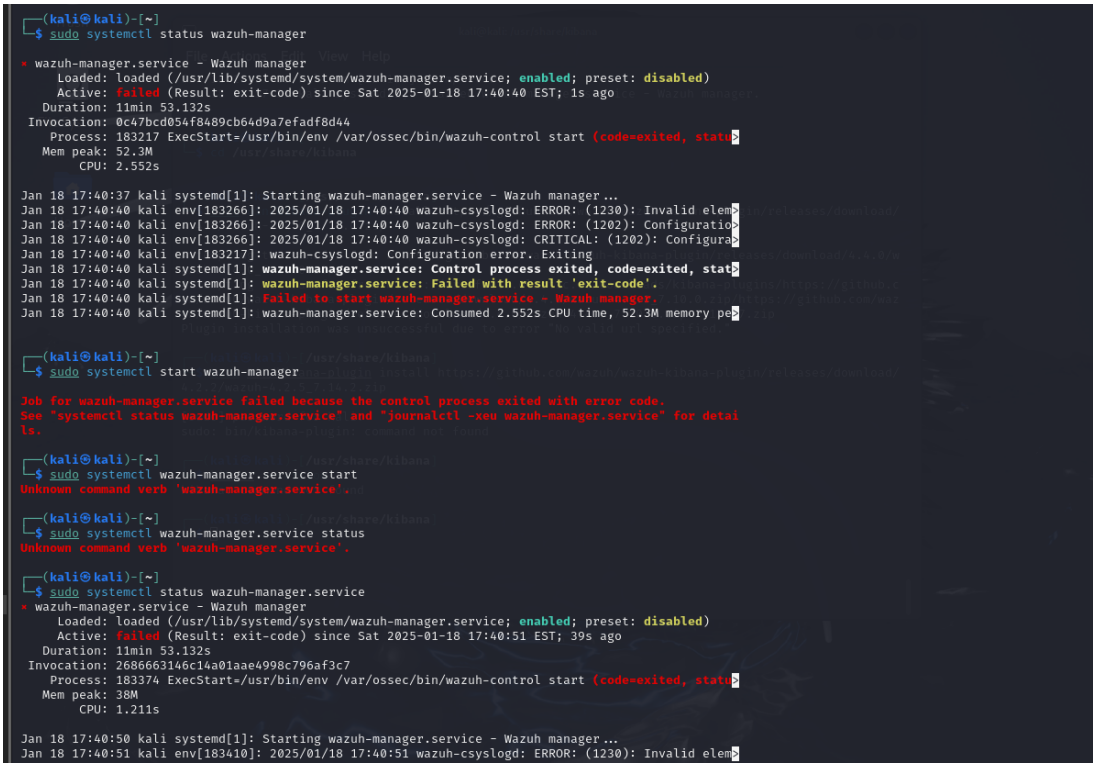


Figure 14: Wazuh not working errors

- **OpenVAS**: We also experimented with OpenVAS for vulnerability scanning. However, due to some errors, we did not continue with its deployment. Below are the steps we followed for installation and configuration:

```
# Install OpenVAS
sudo apt update
sudo apt install openvas

# Set up OpenVAS
sudo gvm-setup
sudo gvm-start
```

Listing 21: Installing OpenVAS

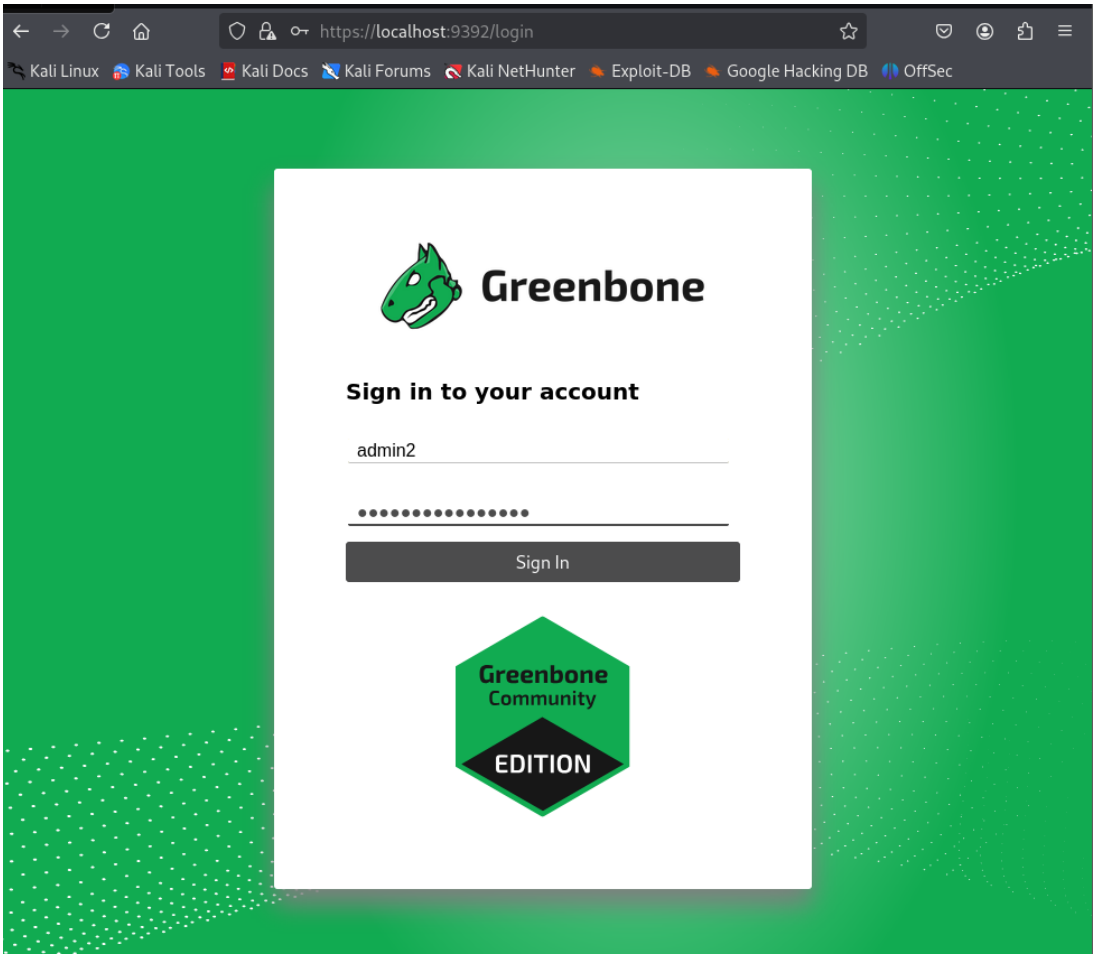


Figure 15: OpenVAS Login

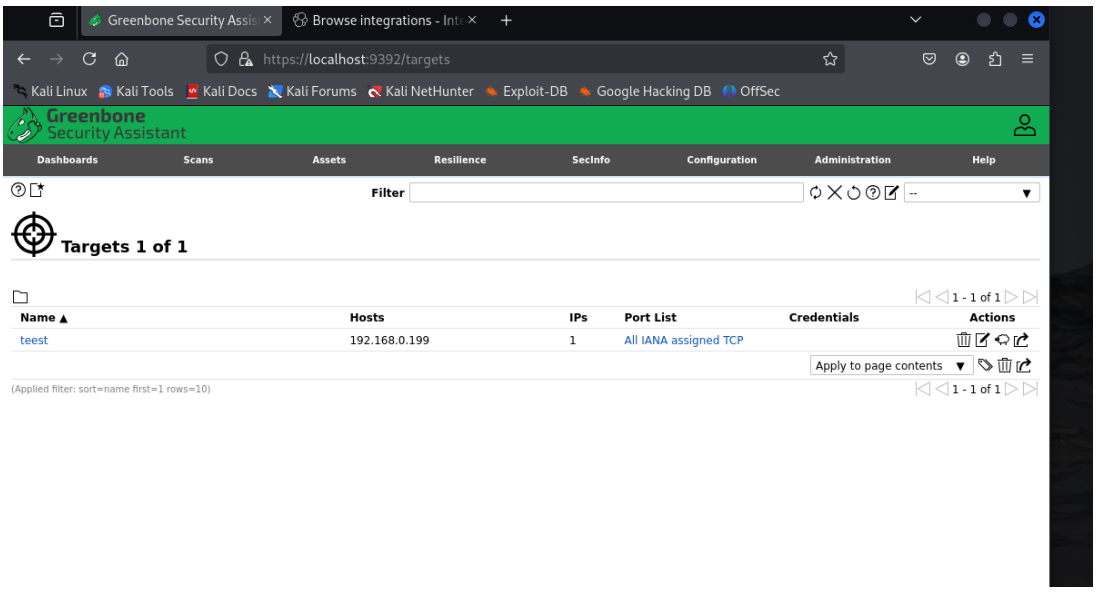


Figure 16: Create target to scan

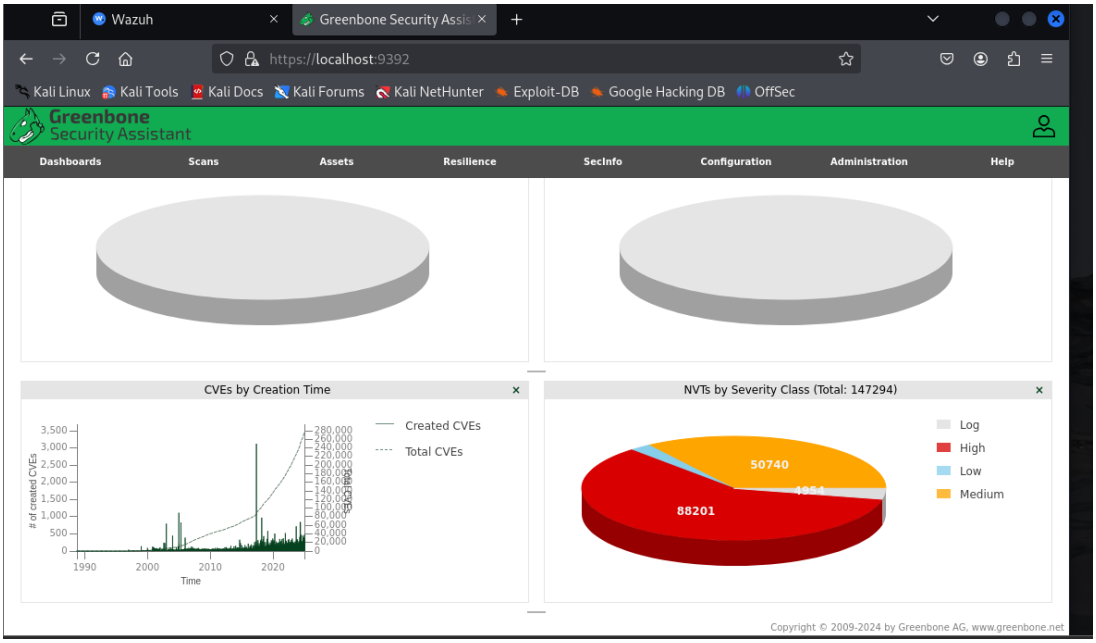


Figure 17: OpenVas interface results

6.2 Advanced Analytics

- Implement machine learning models to detect anomalies automatically.
- Use Elastic Machine Learning to identify unusual patterns in log data.

6.3 Scalability

- Deploy Elasticsearch in a clustered environment for high availability and scalability.
- Use Kafka as a message broker to handle high volumes of log data.

6.4 Automation

- Automate incident response using **Elastic Security Playbooks**.
- Integrate with Slack or Microsoft Teams for real-time notifications.

7 Conclusion

This project underscores the critical importance of centralized log management and real-time security monitoring in safeguarding web servers against a wide array of cyber threats. In today's digital landscape, where web servers are prime targets for attacks such as SQL injections, cross-site scripting (XSS), brute force attempts, and unauthorized access, implementing robust security measures is no longer optional but a necessity. By deploying the ELK stack (Elasticsearch, Logstash, and Kibana) alongside SELinux, this project has demonstrated an effective approach to enhancing the security posture of web servers.

The ELK stack serves as a powerful tool for collecting, processing, and analyzing logs in real time. Elasticsearch provides a scalable and efficient search engine for storing logs, Logstash enables the parsing and transformation of log data, and Kibana offers an intuitive interface for visualizing and analyzing security events. This centralized approach not only simplifies log management but also enables organizations to detect anomalies, identify potential threats, and respond to incidents promptly.

The integration of SELinux further strengthens the security framework by enforcing mandatory access controls and restricting the actions of the Apache web server. This ensures that even if an attacker gains access to the system, their ability to cause harm is significantly limited. The combination of the ELK stack and SELinux provides a multi-layered defense mechanism, enhancing both detection and prevention capabilities.

In conclusion, this project has successfully demonstrated the value of leveraging modern tools like the ELK stack and SELinux to enhance web server security. By combining centralized log management, real-time monitoring, and proactive incident response, organizations can significantly reduce their vulnerability to cyber threats. Future advancements in analytics, automation, and integration will further solidify the system's effectiveness, paving the way for a more secure digital environment.

8 References

- Elasticsearch Documentation: `https://www.elastic.co/guide/index.html`
- Apache HTTP Server Documentation: `https://httpd.apache.org/docs/`
- SELinux Wiki: `https://selinuxproject.org/`

9 Appendices

9.1 Full Configuration Files

9.1.1 Logstash Configuration

```
input {
  beats {
    port => 5044
  }
}

filter {
  if [fields][type] == "apache" {
    grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
    date {
      match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss␣Z" ]
    }

    # Detect SQL Injection
    if [message] =~ /.*(UNION SELECT|DROP TABLE|1=1).*/ {
```

```
      mutate { add_tag => ["sql_injection_attempt"] }
    }

    # Detect Unauthorized File Access
    if [request] =~ /.*(\/etc\/passwd|\.htaccess).*/ {
      mutate { add_tag => ["unauthorized_file_access"] }
    }

    # Detect Brute Force Attacks
    if [response] == "401" or [response] == "403" {
      aggregate {
        task_id => "%{clientip}"
        code => "map['count'] ||= 0; map['count'] += 1"
        map_action => "create"
        timeout => 60
      }
      if [aggregate][count] > 5 {
        mutate { add_tag => ["brute_force_attempt"] }
      }
    }
  }
}

output {
  elasticsearch {
    hosts => ["localhost:9200"]
    index => "apache-logs-%{+YYYY.MM.dd}"
  }
  stdout { codec => rubydebug }
}
```

Listing 22: logstash.conf

9.1.2 Filebeat Configuration

```
filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /var/log/apache2/access.log
      - /var/log/apache2/error.log
    fields:
      type: apache

output.logstash:
  hosts: ["localhost:5044"]
```

Listing 23: filebeat.yml

9.1.3 Auditbeat Configuration

```
auditbeat.modules:
- module: auditd
```

```
  audit_rule_files: [ '${path.config}/audit.rules.d/*.conf' ]
  audit_rules: |
    -a always,exit -F arch=b64 -S execve -k process_execution

output.logstash:
  hosts: ["localhost:5044"]
```

Listing 24: auditbeat.yml

9.2 Python Script used to scan and generate alerts block malicious ips

```
import requests
import os
import logging
import smtplib
import subprocess
import re
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from datetime import datetime, timezone

# Set up logging
logging.basicConfig(filename='/home/kali/sec_scan.log', level=logging.
    INFO, format='%(asctime)s␣%(levelname)s:␣%(message)s')

# Email configuration
EMAIL_HOST = "smtp.gmail.com"  # Gmail SMTP server
EMAIL_PORT = 587  # Gmail SMTP port
EMAIL_USER = os.getenv("EMAIL_USER") # Gmail address
EMAIL_PASSWORD = os.getenv("EMAIL_PASSWORD")  #Gmail app password
EMAIL_RECEIVER = "Email"  # Receiver's email address

if not EMAIL_USER or not EMAIL_PASSWORD:
    raise ValueError("Email␣credentials␣are␣not␣set␣in␣environment␣
        variables.")

def send_email(subject, message):
    try:
        # Create the email
        msg = MIMEMultipart()
        msg["From"] = EMAIL_USER
        msg["To"] = EMAIL_RECEIVER
        msg["Subject"] = subject
        msg.attach(MIMEText(message, "plain"))

        # Connect to the SMTP server
        server = smtplib.SMTP(EMAIL_HOST, EMAIL_PORT)
        server.starttls()  # Enable TLS encryption
        server.login(EMAIL_USER, EMAIL_PASSWORD)  # Log in to the SMTP
            server
        server.sendmail(EMAIL_USER, EMAIL_RECEIVER, msg.as_string())  #
            Send the email
        server.quit()  # Disconnect from the server
```

```
        logging.info("Email␣alert␣sent␣successfully.")
    except Exception as e:
        logging.error(f"Error␣sending␣email:␣{e}")

def run_command(command):
    """Run a shell command and return its output."""
    try:
        result = subprocess.run(command, shell=True, capture_output=True
            , text=True)
        if result.returncode == 0:
            return result.stdout.strip()
        else:
            logging.error(f"Command␣failed:␣{command}\nError:␣{result.
                stderr}")
            return None
    except Exception as e:
        logging.error(f"Error␣running␣command␣{command}:␣{e}")
        return None

def scan_apache_server():
    """Scan the local Apache2 server and gather information."""
    try:
        # Check SELinux status
        selinux_status = run_command("sestatus")
        logging.info(f"SELinux␣Status:\n{selinux_status}")

        # Check Apache status
        apache_status = run_command("systemctl␣status␣apache2")
        logging.info(f"Apache␣Status:\n{apache_status}")

        # Check if mod_security is enabled
        mod_security_status = run_command("apachectl␣-M␣2>/dev/null␣|␣
            grep␣security")
        logging.info(f"mod_security␣Status:␣{'enabled'␣if␣
            mod_security_status␣else␣'disabled'}")

        # Check if mod_evasive is enabled
        mod_evasive_status = run_command("apachectl␣-M␣2>/dev/null␣|␣
            grep␣evasive")
        logging.info(f"mod_evasive␣Status:␣{'enabled'␣if␣
            mod_evasive_status␣else␣'disabled'}")

        # Analyze Apache access logs for bad traffic
        bad_traffic = run_command("tail␣-n␣100␣/var/log/apache2/access.
            log␣|␣grep␣-E␣'404|500|403|SQL␣injection|XSS|brute␣force|
            suspicious'")
        logging.info(f"Bad␣Traffic␣Detected:\n{bad_traffic}")

        return selinux_status, apache_status, mod_security_status,
            mod_evasive_status, bad_traffic
    except Exception as e:
        logging.error(f"Error␣during␣Apache␣server␣scan:␣{e}")
        return None, None, None, None, None

def send_to_elasticsearch(data):
```

```
    """Send data to Elasticsearch."""
    url = "http://localhost:9200/security-scanner/_doc"
    headers = {"Content-Type": "application/json"}
    try:
        response = requests.post(url, json=data, headers=headers)
        response.raise_for_status()  # Raise an exception for HTTP
            errors
        logging.info(f"Elasticsearch␣Response:␣{response.text}")
    except requests.exceptions.RequestException as e:
        logging.error(f"Error␣sending␣data␣to␣Elasticsearch:␣{e}")

def block_ip(ip_address):
    """Block an IP address using iptables."""
    try:
        # Block the IP address
        command = f"iptables␣-A␣INPUT␣-s␣{ip_address}␣-j␣DROP"
        result = subprocess.run(command, shell=True, capture_output=True
            , text=True)
        if result.returncode == 0:
            logging.info(f"Blocked␣IP␣address:␣{ip_address}")
        else:
            logging.error(f"Failed␣to␣block␣IP␣address␣{ip_address}:␣{
                result.stderr}")
    except Exception as e:
        logging.error(f"Error␣blocking␣IP␣address␣{ip_address}:␣{e}")

def quarantine_process(process_id):
    """Quarantine (kill) a suspicious process."""
    try:
        # Kill the process
        command = f"kill␣-9␣{process_id}"
        result = subprocess.run(command, shell=True, capture_output=True
            , text=True)
        if result.returncode == 0:
            logging.info(f"Quarantined␣process:␣{process_id}")
        else:
            logging.error(f"Failed␣to␣quarantine␣process␣{process_id}:␣{
                result.stderr}")
    except Exception as e:
        logging.error(f"Error␣quarantining␣process␣{process_id}:␣{e}")

def generate_alert(message):
    """Generate an alert and send it to Elasticsearch and via email."""
    # Send alert to Elasticsearch
    alert_data = {
        "timestamp": datetime.now(timezone.utc).isoformat(),
        "alert_type": "security_alert",
        "message": message
    }
    send_to_elasticsearch(alert_data)

    # Send alert via email
    email_subject = "Security␣Alert:␣Bad␣Traffic␣Detected"
    email_message = f"Security␣Alert:\n\n{message}"
    send_email(email_subject, email_message)
```

```
    # Block IP addresses and quarantine processes if bad traffic is
        detected
    if "Bad␣traffic␣detected" in message:
        # Extract IP addresses from the bad traffic logs
        ip_addresses = re.findall(r"\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}",
            message)
        for ip in ip_addresses:
            block_ip(ip)

        # Extract process IDs from the bad traffic logs (if applicable)
        process_ids = re.findall(r"PID:␣(\d+)", message)
        for pid in process_ids:
            quarantine_process(pid)

if __name__ == "__main__":
    # Perform Apache server scan
    selinux_status, apache_status, mod_security_status,
        mod_evasive_status, bad_traffic = scan_apache_server()

    # Prepare data for Elasticsearch
    if selinux_status and apache_status:
        data = {
            "timestamp": datetime.now(timezone.utc).isoformat(),
            "scan_type": "security_scan",
            "status": "completed",
            "details": {
                "selinux_status": "enabled" if "enabled" in
                    selinux_status.lower() else "disabled",
                "apache_status": "active" if "active" in apache_status.
                    lower() else "inactive",
                "mod_security_status": "enabled" if mod_security_status
                    else "disabled",
                "mod_evasive_status": "enabled" if mod_evasive_status
                    else "disabled",
                "bad_traffic_detected": bool(bad_traffic.strip()) if
                    bad_traffic else False  # Handle None case
            }
        }

        # Send data to Elasticsearch
        send_to_elasticsearch(data)

        # Generate alerts if bad traffic is detected
        if bad_traffic and bad_traffic.strip():  # Check if bad_traffic
            is not None and not empty
            alert_message = f"Bad␣traffic␣detected␣on␣Apache␣server:\n{
                bad_traffic}"
            generate_alert(alert_message)
    else:
        logging.error("Apache␣server␣scan␣failed.␣No␣data␣sent␣to␣
            Elasticsearch.")
```

Listing 25: Python Script for Scanning and generating Alerts and Incident Response