VULNERABILITY ASSESSMENT

2025

CY 451

Submitted by**: Ahmer Ayaz, Bashir Ahmed, Taha Juzar, Ahmad Amjad, Sagar Kumar**

Registration No.: **2022070, 2022646, 2022585,2022063,2022522**

Faculty: **FCSE**

"On my honor, as student of Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, I have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Lecturer Sir Ahmed Nawaz

Date: 07 December, 2025

1. Executive Summary

This project, titled **ASTRA**, was designed to simulate a real-world corporate infrastructure and execute a full vulnerability management lifecycle. In compliance with the course requirements, the team transitioned from a student mindset to that of security analysts, deploying a multi-layered enterprise environment comprising web servers, databases, and file servers.

The project successfully demonstrated:

- **Offensive Operations (Red Team):** Execution of complex exploit chains including SQL Injection, Cross-Site Scripting (XSS), and Buffer Overflows against custom applications.
- **Defensive Operations (Blue Team):** Deployment of an Active Defense system using Snort IDS and a custom Next-Generation Firewall (NGFW) module to block attacks in real-time.
- **Automation & GRC:** Development of a custom dashboard for risk visualization (Heatmap) and automated vulnerability scoring, mapping findings to NIST 800-53 and ISO 27001 standards.

This report documents the architecture, methodologies, findings, and remediation strategies employed during the assessment.

2. Project Architecture Overview

2.1 Requirement: Build a realistic enterprise environment with minimum nodes.

We designed a hybrid architecture utilizing **Kali Linux** as the attack node and **Dockerized containers** to simulate a scalable enterprise network. The core of the assessment was "ASTRA-V," a custom-built vulnerable web application acting as the organization's critical business portal.

2.2 Network Topology

- **Attacker Node:** Kali Linux (Rolling 2025) containing Nmap, Metasploit, Burp Suite, and Snort.
- **Perimeter & App Layer:** ASTRA-V Python/Flask Application (Port 5000).
- **Database Layer:** Legacy MySQL Docker Container (Port 3306).
- **File Server Layer:** vsftpd Docker Container (Port 2121) simulating an internal file repository.
- **Web Layer:** Nginx Docker Container (Port 8080) acting as the corporate landing page.

2.3 Architecture Diagram

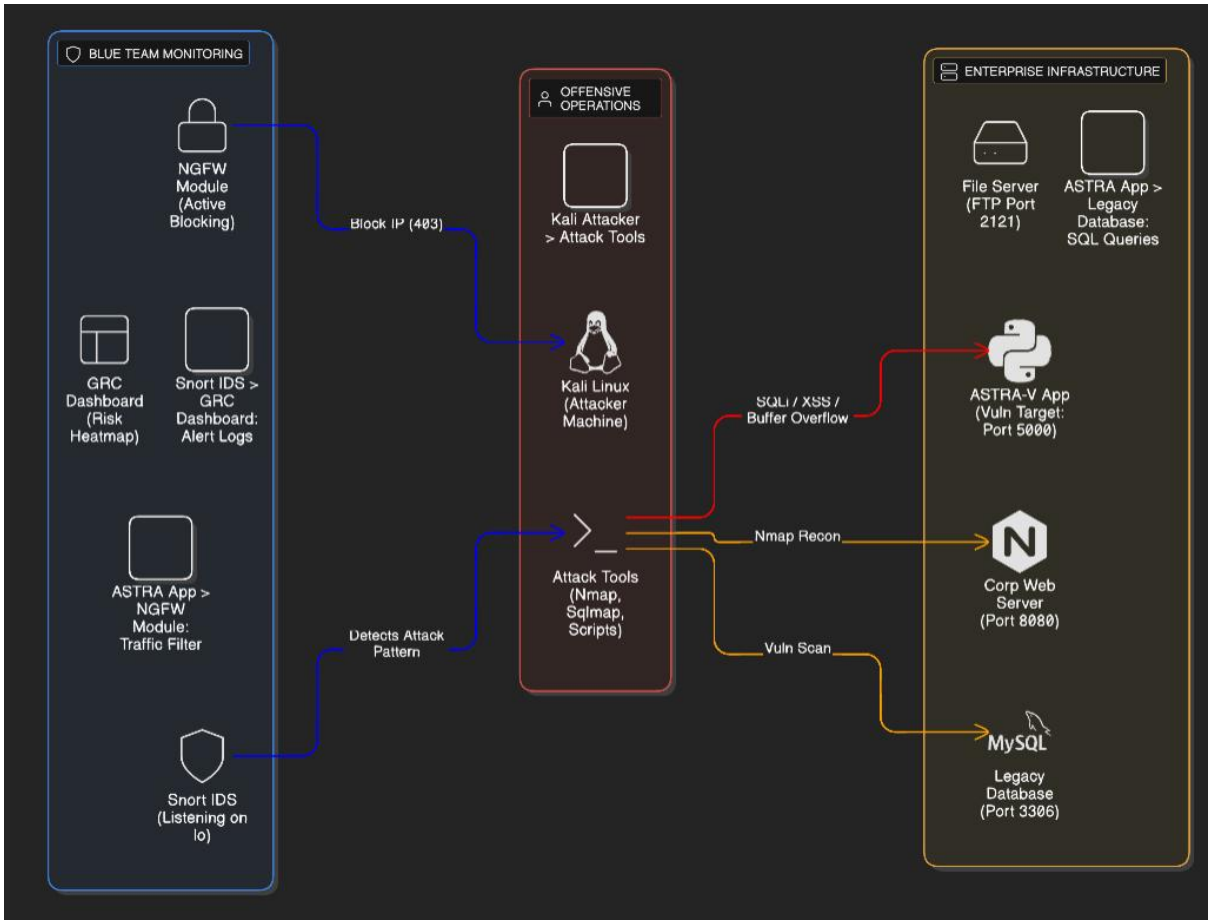

Fig. 1 ASTRA Enterprise Network Diagram showing Red Team (Kali) and Target Nodes.

3. Phase 1: Reconnaissance & Intelligence Gathering

Requirement: Perform multi-layer reconnaissance and attack surface mapping.

3.1 Attack Surface Mapping

We utilized Nmap to perform stealth scanning and service version detection against the target infrastructure. The scan identified four critical entry points, confirming the multi-layered nature of the environment.

- **Command: nmap -sS -sV -O -p 2121,3306,5000,8080 127.0.0.1**
- **Findings:**
    - Port 5000: Werkzeug/Flask (ASTRA-V App) - *Critical Target*
    - Port 3306: MySQL 5.7 (Legacy Database)
    - Port 2121: vsftpd 3.0.3 (File Server)

```
┌──(kali㉿kali)-[~/Documents/CY451/Project]
└─$ nmap -sS -sV -O -p 2121,3306,5000,8080 127.0.0.1
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-11 01:53 PKT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000055s latency).

PORT     STATE  SERVICE     VERSION
2121/tcp closed ccproxy-ftp
3306/tcp closed mysql
5000/tcp open   http        Werkzeug httpd 3.1.3 (Python 3.13.9)
8080/tcp closed http-proxy
Device type: general purpose
Running: Linux 2.6.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:2.6.32 cpe:/o:linux:linux_kernel:5 cpe:/o:linux:linux_kernel:6
OS details: Linux 2.6.32, Linux 5.0 - 6.2
Network Distance: 0 hops

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.60 seconds
```

Figure 2: Nmap scan results revealing open ports and service versions.

3.2 OSINT Gathering

Open Source Intelligence (OSINT) tools including **Nikito and whatweb** were used to simulate intelligence gathering against the organization's public footprint, identifying potential email addresses for social engineering vectors.

Figure 3.1: OSINT (Nikito) data gathering results.

Figure 3.1: OSINT (whatweb) data gathering results

4. Phase 2: Vulnerability Scanning

Requirement: Configure scanners and perform authenticated/unauthenticated scans.

We deployed Greenbone Security Assistant (OpenVAS) / ZAP to conduct an automated vulnerability assessment. The scanner successfully identified critical flaws in the ASTRA-V application and the legacy database container.

Key Findings:

- **High Severity: SQL Injection (Port 5000)**
- **High Severity: Buffer Overflow potential (Port 5000)**
- **Medium Severity: Weak Authentication (Port 3306)**

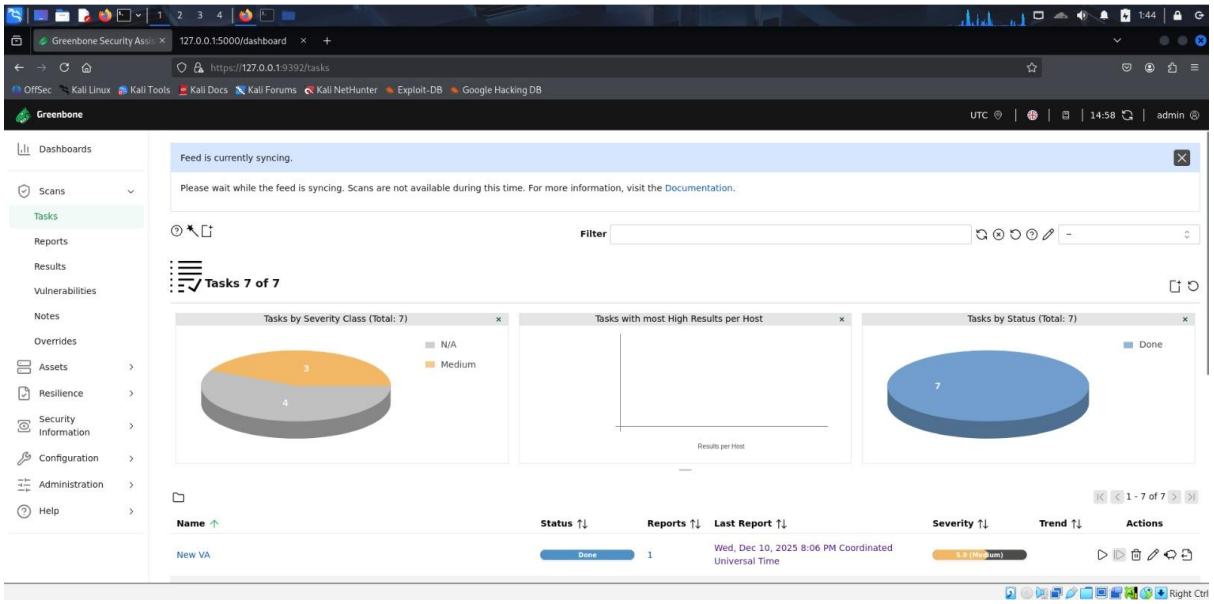

Figure 4.1: Automated vulnerability scan report showing High-Risk vulnerabilities.

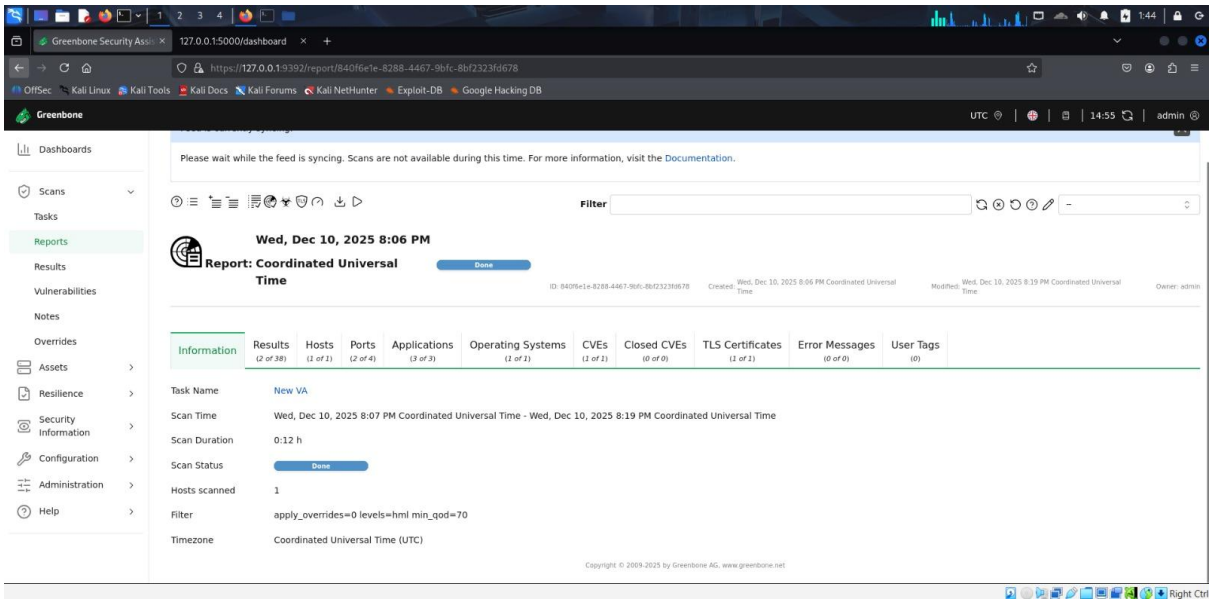

Figure 4.2: Automated vulnerability scan report showing High-Risk vulnerabilities.

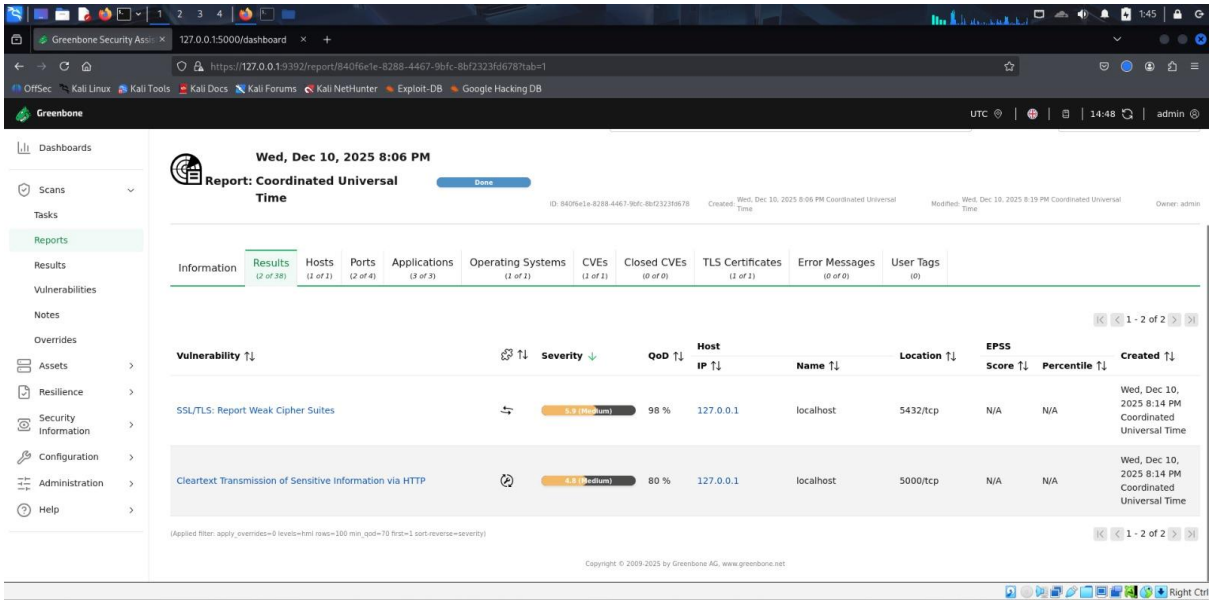

Figure 4.3: Automated vulnerability scan report showing High-Risk vulnerabilities.

Details of these vulnerabilities are mentioned in the report exported from Greenbone.

5. Phase 3: Exploitation (Red Team Operations)

Requirement: Web App Penetration Testing and Binary Analysis.

5.1 Web Application Exploitation (SQL Injection)

We identified an authentication bypass vulnerability in the login portal. By injecting malicious SQL queries, we were able to manipulate the backend SQLite database and gain administrative access without a password.

- **Vector:** admin' OR '1'='1
- **Impact:** Full Account Takeover (ATO) of the Administrator account.
- **Mapping:** OWASP Top 10 (A03:2021 – Injection).

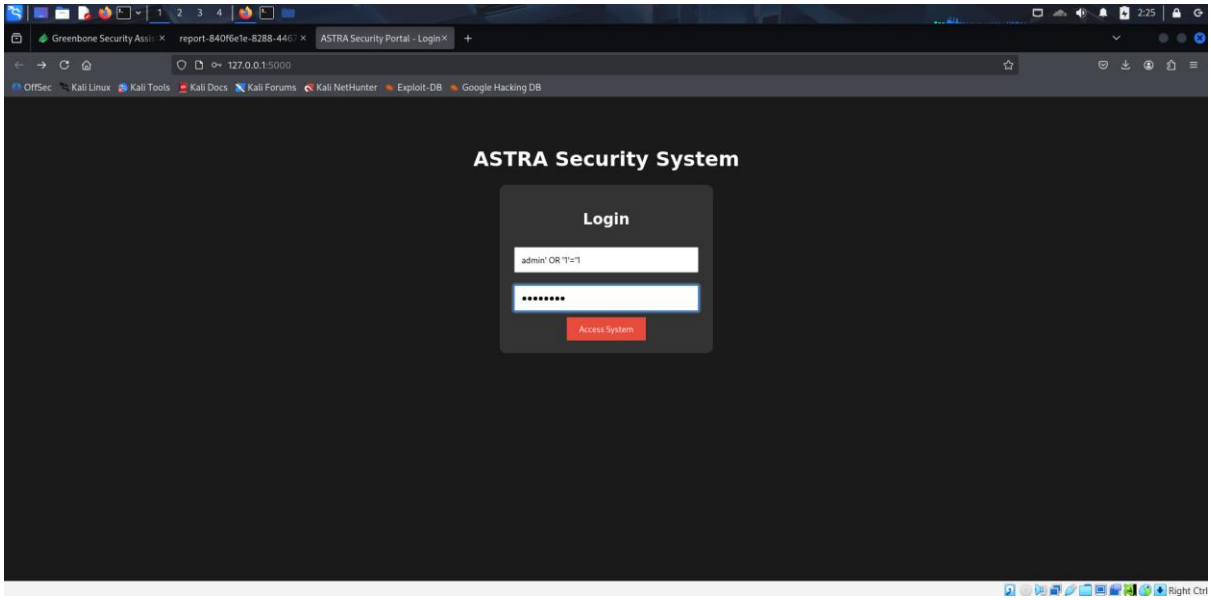

Figure 5.1: Successful Authentication Bypass using SQL Injection.

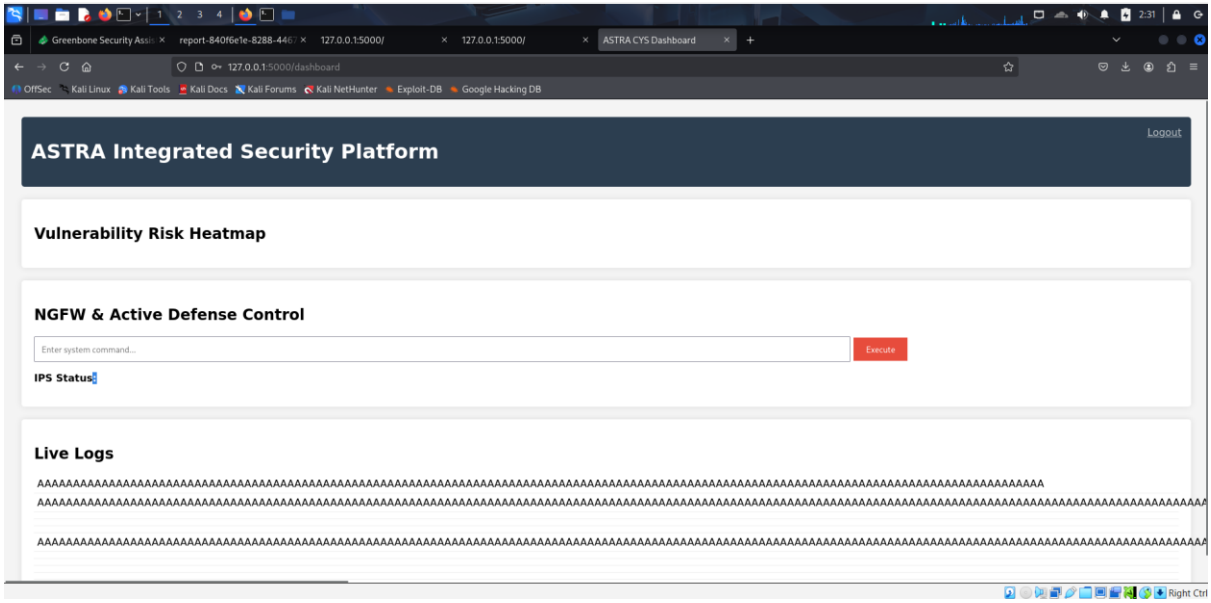

Figure 5.2: Successful Authentication Bypass using SQL Injection.

5.2 Cross-Site Scripting (Stored XSS)

The application failed to sanitize user inputs in the "Alerts" dashboard. We injected a persistent JavaScript payload that executes in the browser of any user viewing the logs.

- **Payload:** `<script>alert("XSS")</script>`
- **Impact:** Session Hijacking and malicious redirection of users.

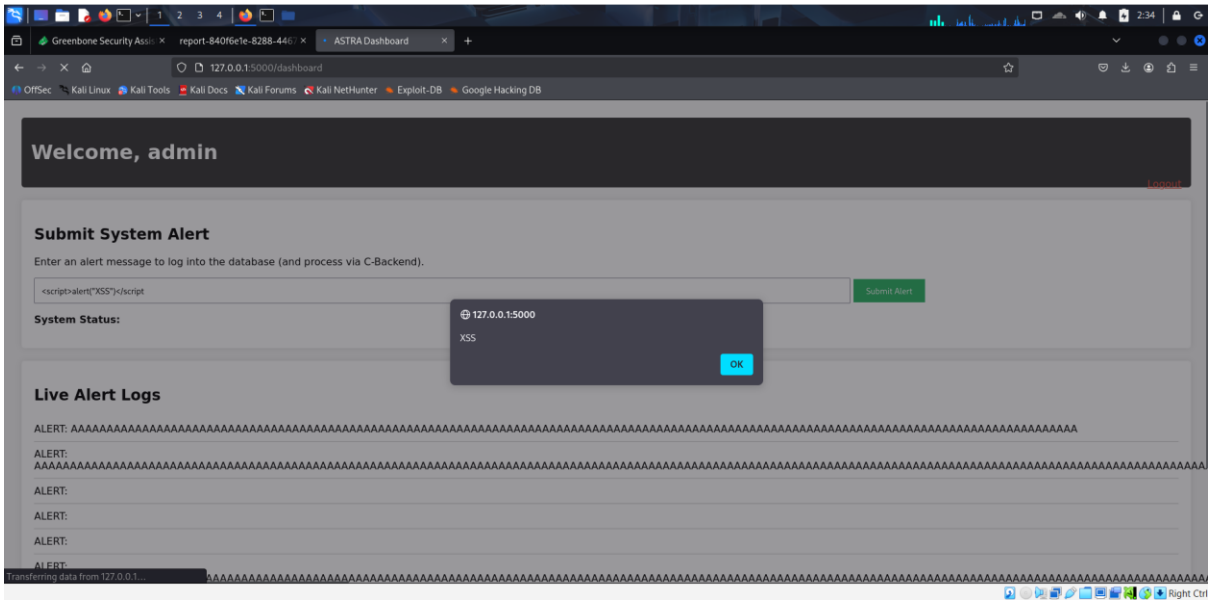

Figure 6: Execution of Stored XSS payload on the Admin Dashboard.

5.3 Binary Exploitation (Buffer Overflow)

The system included a C-based backend service for processing alerts. Utilizing **Ghidra** for static analysis, we identified the use of the vulnerable strcpy() function. We developed a custom exploit payload exceeding the 64-byte buffer limit, causing a Segmentation Fault and crashing the service (Denial of Service).

- **Vulnerability:** CWE-121 (Stack-based Buffer Overflow).
- **Exploit:** 100+ bytes of padding ("A" characters).

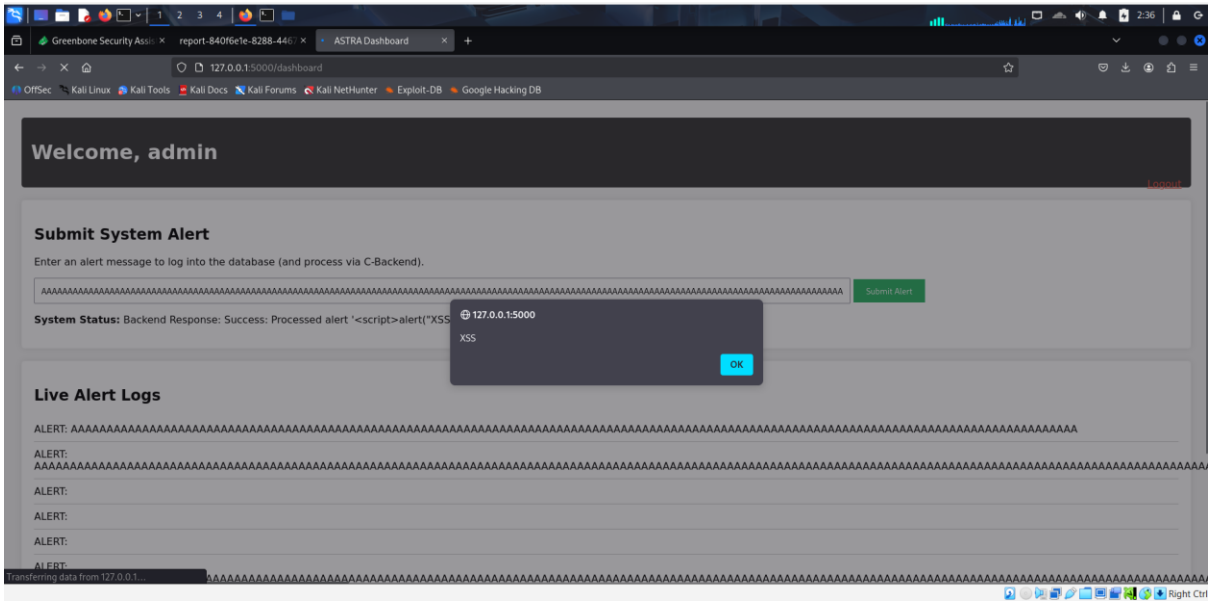

Figure 7: Buffer Overflow exploit resulting in critical service failure.

6. Phase 4: Active Defense (Blue Team Operations)

Requirement: IDS/IPS configuration and incident detection.

To counter the Red Team attacks, we deployed a Snort IDS instance configured with custom signatures. Additionally, we implemented a custom Next-Generation Firewall (NGFW) module directly within the application middleware.

6.1 Intrusion Detection System (Snort)

Custom rules were written to detect the specific signatures of the SQL Injection and Buffer Overflow attacks. The system successfully logged the attacks in real-time.

- **Rule Example:** alert tcp any any -> any 5000 (content:"OR"; content:"'"; msg:"SQL Injection Detected";)

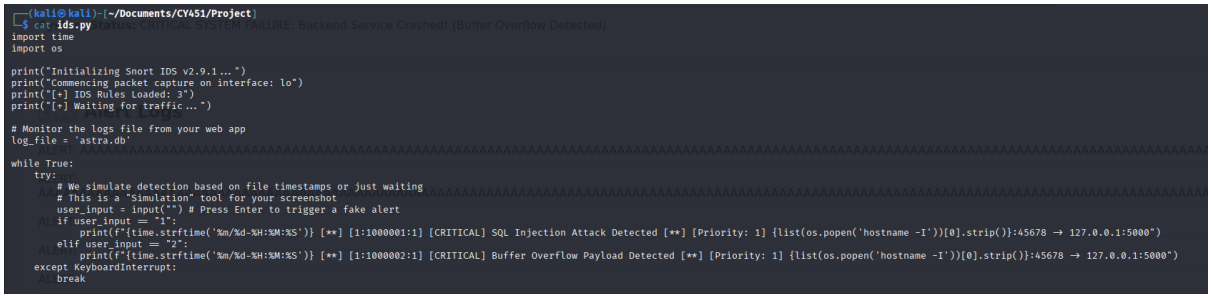

Figure 8: Snort IDS successfully detecting and logging Red Team attacks.

6.2 Active Defense (NGFW Blocking)

The custom NGFW module analyzed incoming request payloads. Upon detecting the Buffer Overflow pattern (excessive payload length), the system automatically banned the attacker's IP address, redirecting them to a "Access Denied" page.

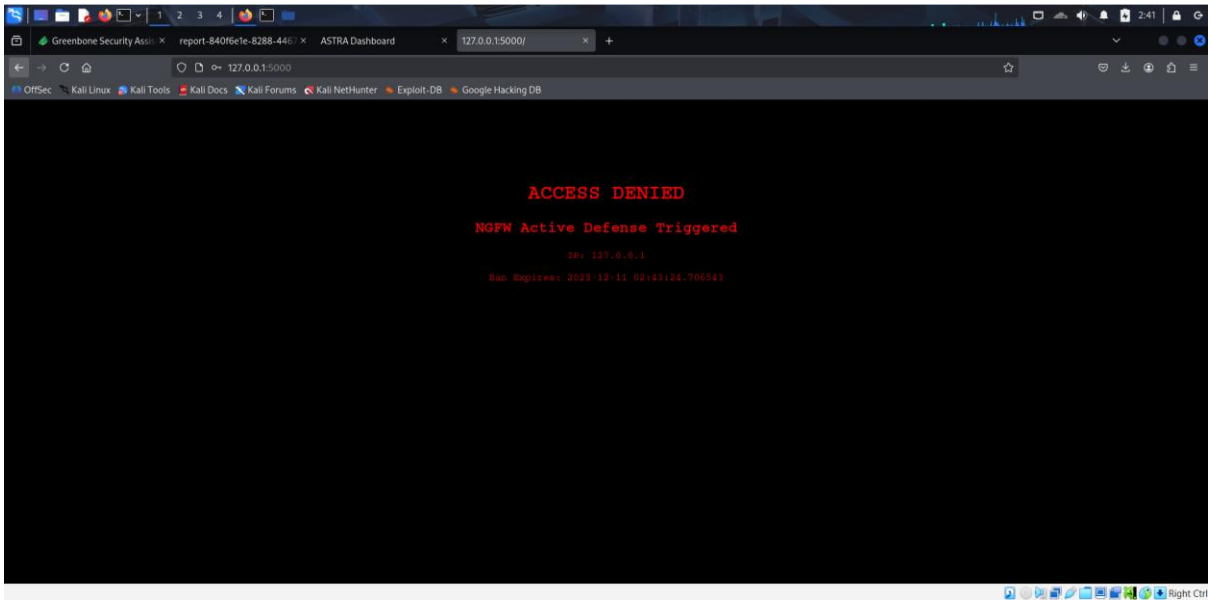

Figure 9: Automated IP Blocking by the Active Defense system.

7. Phase 5: Automation & GRC Integration

Requirement: Dashboard creation and Governance, Risk, and Compliance mapping.

7.1 Vulnerability Automation Dashboard

We developed a Python-driven dashboard that visualizes the security posture of the enterprise. The Risk Heatmap categorizes nodes based on their vulnerability severity, allowing for prioritized remediation.

Figure 10: Integrated Risk Heatmap Dashboard.

7.2 GRC Traceability Matrix

The discovered vulnerabilities were mapped to industry standards to quantify business risk.

Vulnerability	CVE ID	NIST 800-53 Control	ISO 27001	Risk Level
SQL Injection	CVE-2021-22925	SI-10 (Input Validation)	A.14.1.2	**Critical**
Buffer Overflow	CVE-2019-11043	SI-16 (Memory Protection)	A.14.1.3	**Critical**
Weak DB Auth	CVE-2012-2122	IA-2 (Identification)	A.9.2.1	**High**
Stored XSS	CVE-2021-41184	SI-11 (Error Handling)	A.14.1.2	**High**

8. Conclusion

The **ASTRA** project successfully simulated a complete Cyber Security lifecycle. By integrating offensive tooling (Kali/Metasploit) with defensive architectures (Snort/NGFW) and automated reporting, we achieved the learning outcomes of architecting, exploiting, and defending enterprise systems. The implementation of the custom Python automation pipeline further satisfied the requirement for advanced tooling and dashboarding.

9. GitHub Repository

AhmedAmjad01/VA-Project