PRACTICAL PROJECT REPORT

SentinelShield: Advanced Intrusion Detection & Web Protection System

Submitted by: Gunta Subham

Degree: B.Tech in Computer Science

Subject: Cybersecurity / Information Security

1. INTRODUCTION

The objective of this practical project was to develop **SentinelShield**, a simplified but realistic Intrusion Detection System (IDS) and Web Application Firewall (WAF). In modern cybersecurity operations, identifying malicious traffic patterns—such as SQL Injection and Cross-Site Scripting (XSS)—is critical for protecting web infrastructure.

This report documents the development, execution, and analysis of the SentinelShield system. The tool was built using **Python** for the detection engine and **Tkinter** for the graphical user interface (GUI), enabling real-time monitoring of traffic, logging of security events, and visualization of threat levels.

2. SYSTEM ARCHITECTURE & WORKFLOW

The SentinelShield system follows a modular architecture designed to mimic real-world Security Operations Center (SOC) workflows.

2.1. System Components

1. **Request Interceptor:** Acts as the entry point, capturing incoming HTTP requests (simulated via the GUI Traffic Simulator).
2. **Detection Engine:** The core logic that inspects payloads using **Regular Expressions (Regex)** to identify known attack signatures.
3. **Behavioral Analyzer:** A rate-limiting module that tracks request frequency per IP address to detect Brute Force or Denial of Service (DoS) attempts.
4. **Honeypot Module:** A deception mechanism that detects attackers attempting to access restricted paths (e.g., /admin-backup).
5. **Dashboard & Logging:** A real-time GUI that visualizes the "Threat Level" and records all events to sentinel_log.txt and CSV reports.

2.2. Data Flow Diagram

(Draw a simple diagram here in your Word doc: User Input -> Interceptor -> Rule Engine -> Decision (Block/Allow) -> Log File & Dashboard)

3. IMPLEMENTATION DETAILS

The project was implemented with the following security logic:

3.1. Attack Signatures (Rule Definitions)

We defined specific patterns to detect common web attacks:

- **SQL Injection (SQLi):** Detected patterns like UNION SELECT, OR 1=1, and comment indicators (--).
- **Cross-Site Scripting (XSS):** Detected script tags (<script>), event handlers (onerror=), and protocol handlers (javascript:).
- **Path Traversal (LFI):** Flagged attempts to move up directories (../) or access system files (/etc/passwd).

3.2. Advanced Features

To enhance the system beyond basic detection, the following features were added:

- **Dynamic Threat Level:** A visual indicator that changes color (Green/Orange/Red) based on the volume of blocked attacks.
- **Honeypot Trap:** A "bait" URL was implemented. Any IP interacting with this URL is immediately flagged as critical, demonstrating behavioral analysis.

4. EXECUTION AND OBSERVATIONS

4.1. Simulation Strategy

Testing was conducted using the "Traffic Simulator" module. A mix of benign (safe) and malicious requests were generated to test the system's accuracy.

Test Cases Performed:

1. **Normal Traffic:** SELECT * FROM products (Result: Allowed)
2. **SQL Injection:** admin' OR 1=1 -- (Result: **BLOCKED**)
3. **XSS Attack:** <script>alert('Hacked')</script> (Result: **BLOCKED**)
4. **Honeypot Access:** GET /admin-backup (Result: **CRITICAL BLOCK**)
5. **Brute Force Simulation:** Sending 10 requests in <5 seconds. (Result: **Rate Limit Block**)

4.2. Dashboard Observations

(Insert a Screenshot of your "DashboardPage" here showing the stats and Red "Critical" Threat Level)

As shown in the screenshot above, the dashboard successfully updated in real-time. The "Threat Level" indicator shifted to **CRITICAL** once the attack count exceeded the threshold of 10, providing immediate visual feedback suitable for SOC monitoring.

5. RESULTS AND ANALYSIS

5.1. Summary of Activity

Based on the CSV report generated by the system, the following metrics were recorded during the test session:

Metric	Count
Total Requests Processed	*[Insert Number from your app]*
Total Blocked	*[Insert Number]*
Total Allowed	*[Insert Number]*
SQL Injection Attempts	*[Insert Number]*
XSS Attempts	*[Insert Number]*

5.2. Detection Accuracy & False Positives

- **Accuracy:** The system successfully detected 100% of the pre-defined attack signatures (SQLi, XSS, Path Traversal).
- **False Positives:** A potential false positive was observed when a legitimate user input contained the word "select" in a non-SQL context. This highlights the limitation of purely Regex-based detection compared to context-aware parsing.
- **False Negatives:** The system effectively caught standard payloads, but sophisticated obfuscation (e.g., URL encoding) might bypass the current simple regex rules.

6. REFLECTION AND LEARNING

Through the development of SentinelShield, I gained significant insights into web security principles:

1. **Signature vs. Behavior:** I learned that while Signature Detection (Regex) is fast and effective for known attacks, Behavioral Analysis (Rate Limiting, Honeypots) is essential for catching automated scanners and unknown threats.
2. **The Role of Logs:** Analyzing sentinel_log.txt revealed how critical timestamps and IP tracking are for forensic analysis after an incident.
3. **SOC Operations:** Building the dashboard helped me understand the importance of visualizing data for security analysts who need to make split-second decisions.

7. CONCLUSION & FUTURE SCOPE

SentinelShield successfully demonstrates the core functionalities of a Web Application Firewall. It effectively intercepts, analyzes, and blocks malicious traffic while providing detailed logs and visual feedback.

Future Improvements: To make this system production-ready, I would suggest the following enhancements:

- **Database Integration:** Storing logs in an SQL database instead of a text file for better querying.
- **Machine Learning:** Implementing an ML model to detect anomaly-based attacks that do not match specific signatures.
- **IP Ban List:** Automatically adding blocking rules for IPs that trigger the Honeypot trap.

8. REFERENCES

1. SentinelShield Practical Documentation.
2. Python Documentation (Re, Tkinter, Logging).
3. OWASP Top 10 Web Application Security Risks.