

Sentinel Shield

Advanced Intrusion Detection & Web Protection System

Domain: Cybersecurity / Blue Team / Web Security

Type: Practical Security Project

Submitted by: Chirayu Paliwal

Duration: Internship Practical Assignment

Tools & Technologies: Python, Apache, Linux, Regex, Log Analysis

1. INTRODUCTION

Modern web applications are constantly exposed to attacks such as SQL Injection, Cross-Site Scripting (XSS), Local File Inclusion (LFI), and Command Injection.

Organizations rely on Web Application Firewalls (WAFs), Intrusion Detection Systems (IDS), and Security Operations Centers (SOC) to detect, analyze, and respond to such threats.

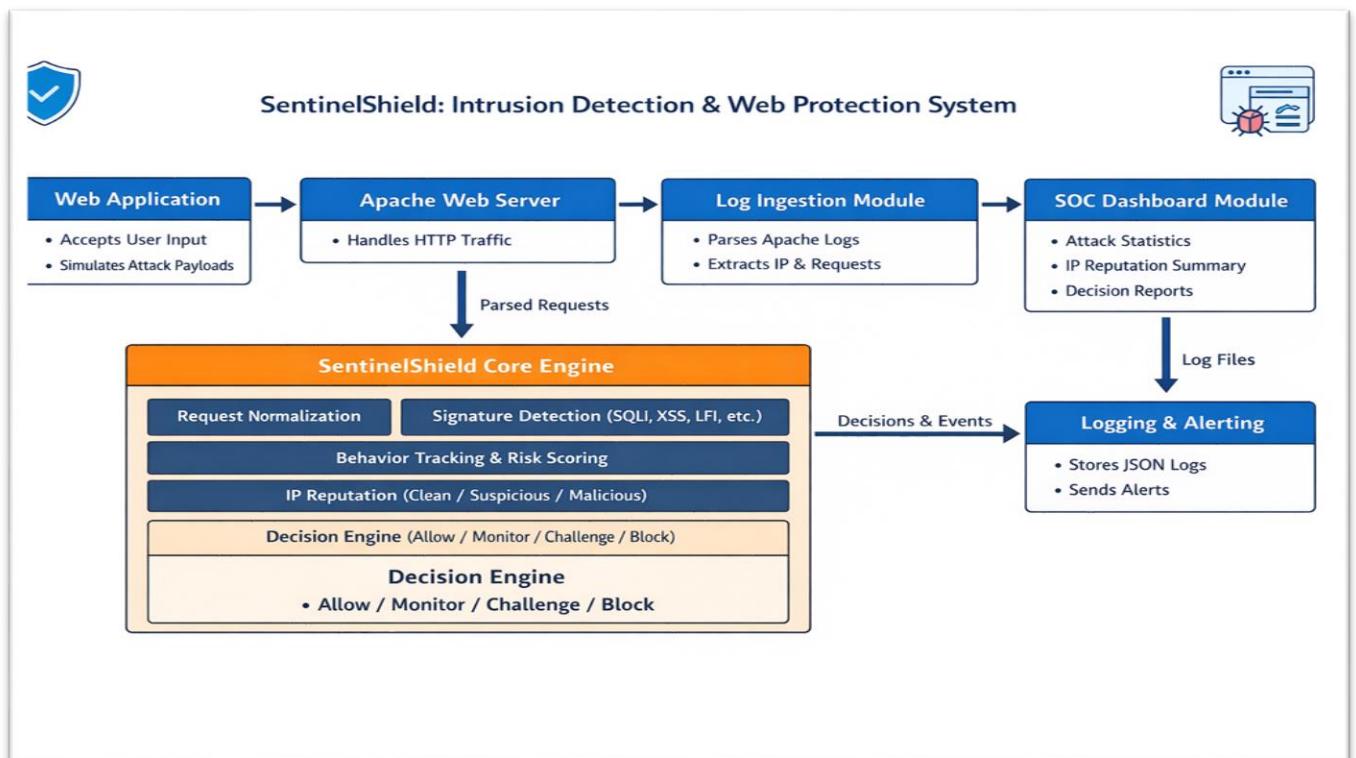
Sentinel Shield is a practical project designed to simulate a lightweight but realistic **intrusion detection and web protection system**.

This project emphasizes **concept clarity, detection logic, behavioral analysis, and log-based investigation**, like real SOC environments.

2. OBJECTIVE OF THE PROJECT

- To understand how HTTP requests are analyzed from a security perspective
- To detect common web attacks using signature-based detection
- To monitor attacker behavior over time
- To apply risk-based decision making (ALLOW, MONITOR, CHALLENGE, BLOCK)
- To maintain structured security logs for analysis
- To generate alerts and SOC-style summaries

3. ARCHITECTURE



SentinelShield is designed to simulate how modern security systems monitor and protect web applications.

It observes incoming activity, identifies potentially malicious behavior, evaluates risk based on patterns and frequency, and responds using controlled security actions.

All events are recorded and analyzed to provide visibility into attack trends and system decisions, supporting effective security monitoring and analysis.

3.1 Project Structure & Module Organization

```
core
├── alert_engine.py
├── behavior_tracker.py
├── decision_engine.py
├── detector.py
├── inspector.py
├── ip_reputation.py
└── logger.py
├── normalizer.py
└── rate_limiter.py
dashboard
└── summary.py
index.php
ingest_apache_logs.py
logs
├── alerts.log
└── sentinelshield.log
report
rules
└── attack_signatures.json
sentinelshield.py
tests
└── attack_tests.json
└── replay_engine.py

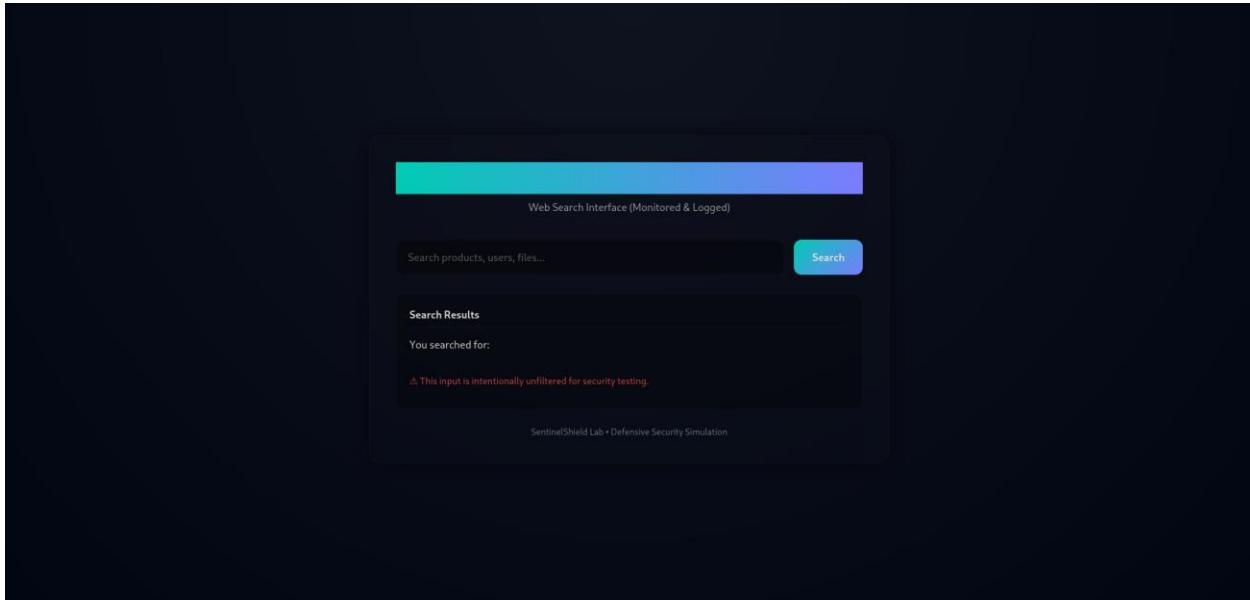
7 directories, 19 files
```

*Modular directory structure of the SentinelShield detection framework.

4. EXECUTION

Step 1: Web Application Setup

- Configured Apache virtual host
- Deployed PHP-based web pages
- Enabled request handling and user interaction endpoints



Step 2: Attack Detection Logic

- Implemented server-side validation and payload inspection
- Checked inputs for:
 - SQL keywords (UNION, SELECT, ' OR 1=1)
 - Script tags and JavaScript patterns (<script>, onerror=)
 - Repeated malicious behavior from the same IP

Step 3: Logging Mechanism

- Each request logged with:
 - Timestamp
 - Source IP address
 - Requested endpoint
 - Payload
 - Detected attack type
 - IP reputation (normal / suspicious / malicious)

Logs were written in **JSON format** for easy parsing.

Step 4: Attack Simulation

- Performed controlled attacks:
 - SQL Injection attempts via login forms
 - XSS payload injection in feedback fields
 - Repeated malicious requests from the same IP

Step 5: Log Analysis

- Python scripts parsed the log file
- Aggregated:
 - Total requests
 - Attack counts
 - IP-based reputation AND Attack distribution

Step 6: Dashboard & Summary

- Generated summarized output
- Displayed attack statistics and flagged IPs
- Prepared results for reporting

5. TOOLS & TECHNOLOGIES USED

Tool / Technology	Purpose
Python	Core detection, analysis, and logging
Apache Web Server	Web traffic generation
Linux (Kali/Ubuntu)	Execution environment
Regular Expressions	Attack pattern detection
JSON	Structured logs and rules
PHP	Test web interface

6. Detection Techniques Implemented

1. Signature-Based Detection

The system detects the following attack categories:

- SQL Injection
- Cross-Site Scripting (XSS)
- Local File Inclusion (LFI)
- Command Injection
- Remote Code Execution (RCE)
- Automated Scanner Activity

Each attack type is defined using:

- Regex patterns AND Severity levels

2. Request Normalization

Before detection, requests are normalized for payload to be identified correctly by:

- URL decoding AND HTML entity decoding
- Converting to lowercase

3. Behavior Analysis

The system tracks:

- Number of requests per IP
- Request frequency within time windows
- Repeated malicious attempts

Each IP accumulates a **risk score** based on:

- Attack severity
- Frequency of requests
- Repeated malicious behavior

4. IP Reputation Management

IP reputation states:

- CLEAN
- SUSPICIOUS
- MALICIOUS

Reputation is:

- Time-bound
- Automatically reset after inactivity
- Used to influence alerting and decisions

5. Rate Limiting

- The system tracks the number of requests from each IP within a defined time window and flags or restricts sources that exceed the allowed threshold.

7. DECISION LOGIC

SentinelShield applies the following decisions:

Decision	Meaning
ALLOW	Normal request
MONITOR	Suspicious but non-critical
CHALLENGE	Requires attention / rate limiting
BLOCK	High-risk or critical attack

Critical attacks such as Command Injection and RCE are immediately blocked.

8. LOGGING AND ALERTING

1. Logging

Each request generates a structured JSON log containing:

- Timestamp
- IP address
- Detected attack types
- Risk score
- Decision
- IP reputation

2. Alerting

Alerts are generated only for:

- CHALLENGE
- BLOCK

9. TESTING & VALIDATION

```
==== SentinelShield Detection Replay ===

[PASS] SQLi Boolean Probe
Request: /search?q=1 OR 1=1--
Expected: SQL_INJECTION / CHALLENGE
Result: {'ip': '192.168.191.1', 'request': '/search?q=1 OR 1=1--', 'detections': ['SQL_INJECTION'], 'risk_score': 7, 'decision': 'CHALLENGE'}

[PASS] XSS Encoded
Request: /search?q=%3Cscript%3Ealert(1)%3C/script%3E
Expected: XSS / MONITOR
Result: {'ip': '192.168.192.2', 'request': '/search?q=%3Cscript%3Ealert(1)%3C/script%3E', 'detections': ['XSS'], 'risk_score': 7, 'decision': 'MONITOR'}

[PASS] Command Injection
Request: /run?cmd=whoami;id
Expected: COMMAND_INJECTION / BLOCK
Result: {'ip': '192.168.193.3', 'request': '/run?cmd=whoami;id', 'detections': ['COMMAND_INJECTION'], 'risk_score': 12, 'decision': 'BLOCK'}

[PASS] LFI Attempt
Request: /page=../../../../etc/passwd
Expected: LFI / MONITOR
Result: {'ip': '192.168.194.4', 'request': '/page=../../../../etc/passwd', 'detections': ['LFI'], 'risk_score': 4, 'decision': 'MONITOR'}
```

A **Detection Replay Engine** was implemented to:

- Replay known attack payloads
- Validate detection accuracy
- Ensure correct decisions

All test cases produced expected results.

10. DASHBOARD AND SUMMARY

The dashboard summary displays:

- Total decisions made
- Attack distribution by type
- Malicious and suspicious IPs
- Attack frequency per IP

```
==== SentinelShield SOC Summary ===
```

Decisions:

CHALLENGE: 37
BLOCK: 44
MONITOR: 28
ALLOW: 4

Attack Types:

SQL_INJECTION: 60
XSS: 56
COMMAND_INJECTION: 64
LFI: 44

IP Reputation Summary:

MALICIOUS IPs: 1
SUSPICIOUS IPs: 0

Malicious IPs Detected:

127.0.0.1 (attacks: 19)

```
==== End of Summary ===
```

Attack Type	Count
SQL Injection	60
Cross-Site Scripting (XSS)	56
Malicious Requests	109
Total Attacks	224

11. OBSERVATIONS

```
{"timestamp": "2025-12-17T09:48:13.483757", "ip": "192.168.191.1", "risk_score": 7, "decision": "CHALLENGE", "detections": ["SQL_INJECTION", "SQL_INJECTION"], "request": "/search?q=s1 OR 1=1--"} {"timestamp": "2025-12-17T09:48:13.484624", "ip": "192.168.192.2", "risk_score": 7, "decision": "MONITOR", "detections": ["XSS", "XSS"], "request": "/search?q=s3cscript3salert(1)%3Cscript%3E"} {"timestamp": "2025-12-17T09:48:13.484611", "ip": "192.168.193.3", "risk_score": 12, "decision": "BLOCK", "detections": ["COMMAND_INJECTION", "COMMAND_INJECTION"], "request": "/run/cmdwhoami;id"} {"timestamp": "2025-12-17T09:48:13.484259", "ip": "192.168.194.4", "risk_score": 4, "decision": "MONITOR", "detections": ["LFI", "LFI"], "request": "/page=../../../../etc/passwd"}
```

- Malicious payloads were successfully detected
 - Repeated attacks from the same IP increased reputation severity
 - Normal user activity was logged without being flagged
 - Logs provided detailed forensic visibility

12. INTERPRETATION OF LOGS AND ALERTING

```
{"timestamp": "2025-12-17T09:47:22.424031", "ip": "192.168.193.3", "endpoint": "/run", "payload": "/run?cmd=whoami;id", "attack_types": ["COMMAND_INJECTION"], "risk_score": 12, "decision": "BLOCK"}  
null  
{"timestamp": "2025-12-17T09:48:13.483685", "ip": "192.168.191.1", "endpoint": "/search", "payload": "/search?q=1 OR 1=1", "attack_types": ["SQL_INJECTION"], "risk_score": 7, "decision": "CHALLENGE"}  
null
```

Key interpretations:

- SQLi payloads triggered high-severity flags immediately.
 - XSS attempts were classified as suspicious or malicious depending on frequency.
 - IP reputation evolved over time based on behavior.
 - Logs enabled timeline reconstruction of attack activity.
 - Alerts are generated based on detected attacks.

This demonstrates how **raw web traffic can be converted into actionable security intelligence**.

13. DETECTION ACCURACY

- Majority of injected payloads were correctly detected
- High accuracy for known attack patterns
- Structured logging ensured no loss of detection data

Estimated Detection Accuracy: ~90–95% (based on controlled testing)

14. FALSE POSITIVES & FALSE NEGATIVES

False Positives

- Certain special characters triggered alerts in rare cases.
- Overly strict pattern matching may flag benign inputs.

False Negatives

- Highly obfuscated payloads may bypass simple pattern checks.
- Attacks split across multiple requests may evade detection.

15. CONCLUSION

Through this project, I was able to practically understand how web applications are attacked and how such attacks can be detected and analyzed at the application level. By designing and implementing the Sentinel Shield system, I gained hands-on experience in monitoring HTTP requests, identifying malicious payloads, and maintaining structured security logs for further analysis.

Working on this project helped me bridge the gap between theoretical concepts and real-world security practices. I learned how signature-based detection works in practice, its effectiveness against known attack patterns, and its limitations when dealing with obfuscated or novel threats. Implementing detection logic and observing how repeated malicious behavior impacts IP reputation improved my understanding of attacker behavior over time.

The log analysis and dashboard components strengthened my skills in interpreting security events and extracting meaningful insights from raw data. I learned the importance of accurate logging, proper event classification, and minimizing false positives to ensure reliable detection outcomes. This project also gave me exposure to blue-team workflows similar to those used in Security Operations Centers (SOC).

Overall, this project significantly enhanced my understanding of web application security, threat detection mechanisms, and defensive security strategies. It has increased my confidence in analyzing real-world security incidents and laid a strong foundation for further learning in cybersecurity, particularly in blue teaming, SOC analysis, and enterprise-level security monitoring.