# POST-IMPLEMENTATION SECURITY REPORT

**Subject:** Deployment and Hardening of Web Application Firewall (WAF) on Legacy Infrastructure
**Date:** December 28, 2025
**Analyst:** Sai Avinash Sirasapalli

**Classification: INTERNAL TECHNICAL DOCUMENT**
**Intended Audience:** SOC Analysts, Infrastructure Security Team, Application Security Team

---

## 1. EXECUTIVE SUMMARY

This project involved a comprehensive security engineering lifecycle to protect a vulnerable legacy web application (DVWA) using a layered defense-in-depth strategy. By deploying **ModSecurity v2.9** and the **OWASP Core Rule Set (CRS) v3.x**, we established a preventive boundary at the HTTP layer. The project transitioned from a completely vulnerable baseline to a hardened state, achieving a **27% reduction in total vulnerabilities** and successfully implementing "Virtual Patching" for critical command injection vectors.

---

## 2. PROJECT METHODOLOGY

### Phase I: Research and Planning

**ModSecurity:** Selected for its robust HTTP inspection capabilities and deep integration with Apache.
**OWASP CRS:** Utilized as a standardized baseline to protect against the "OWASP Top 10" (SQLi, XSS, etc.) with minimal initial overhead.
**OWASP ZAP:** Chosen as the DAST (Dynamic Application Security Testing) tool to simulate real-world attack payloads and validate WAF efficacy.

### Phase II: Execution and Implementation

**Environment Setup:** Initialized **WSL2 (Ubuntu 22.04)** as the target host and **Kali Linux (VM)** as the attacker node.
**Stack Deployment:** Configured a LAMP stack (Apache, MariaDB, PHP) and deployed DVWA with security set to **"Low"** to maximize attack visibility.
**Baseline Scan:** Executed ZAP against the application in `DetectionOnly` mode to record 15 initial alerts.
**WAF Hardening:** Enabled `SecRuleEngine On` and implemented the Core Rule Set.
**Policy Tuning:** Authored custom rules to address protocol-level gaps and managed false positives encountered during administrative tasks.

## 3. THREAT MODEL AND ASSUMPTIONS

**Threat Actor:** Opportunistic attackers and automated bots looking for unpatched injection points or directory traversal vulnerabilities.
**Asset at Risk:** Sensitive application data and server-side execution environment.
**Attack Surface:** Exposed HTTP endpoints on the DVWA platform.
**Assumption:** The WAF is a compensating control; underlying code remains vulnerable, but the *exploitability* is reduced at the network edge.

---

## 4. SYSTEM ARCHITECTURE (LAB DESIGN)

**Target (Blue Team):** WSL2 hosting Apache. A network bridge was configured (listening on `0.0.0.0`) to allow the Kali VM to reach the host.
**Attacker (Red Team):** Kali Linux VM running ZAP Proxy for automated and manual exploitation.
**Telemetry:** Log traffic directed to `/var/log/apache2/modsec_audit.log` for granular event analysis.

---

## 5. QUANTITATIVE ANALYSIS & METRICS

| **Severity** | **Baseline (No WAF)** | **Stage 1 (CRS Active)** | **Stage 2 (Custom Rules)** | **Improvement** |
|---|---|---|---|---|
| **High** | 0* | 0 | 0 | - |
| **Medium** | 5 | 3 | **2** | **60% Reduction** |
| **Low** | 4 | 4 | **3** | **25% Reduction** |
| **Informational** | 6 | 6 | 6 | - |
| **TOTAL ALERTS** | **15** | **13** | **11** | **27% Total** |

**Note: High risks (SQLi/XSS) are present in the app but are dropped by the WAF before ZAP can confirm them, resulting in a "clean" scan result for confirmed vulnerabilities.*

---

# 6. OBSERVATIONS AND TECHNICAL ANALYSIS

## 6.1 Phase 1: CRS Effectiveness (Blocking Reconnaissance)

**Alerts Mitigated:** *Directory Browsing* and *Hidden File Found*.

**Mechanism:** The CRS identifies patterns in URI requests that target sensitive files (e.g., .git, .htaccess, config.inc).

**Analysis:** When ZAP attempted to crawl the directory structure, the CRS matched the request against **Rule 930110** (Sensitive Path Probing). The WAF intercepted the request and issued a **403 Forbidden**, preventing the attacker from mapping the server's internal file system.

## 6.2 Phase 2: Surgical Virtual Patching and Protocol Hardening

### A. Input Validation & Virtual Patching (Rules 10001, 10003, 10004)

**OS Reconnaissance (Rule 10001):** Specifically targeted the `whoami` string. By blocking this reconnaissance command, we prevent attackers from confirming successful command execution or identifying the web server's user privileges.

**Positive Security Model (Rule 10003):** Implemented strict type-enforcement on the `id` parameter.

**Logic:** `SecRule ARGS:id "!@rx ^\d+$" "id:10003,phase:2,deny,status:403..."`

**Analysis:** This rule adopts a "Positive Security" stance by allowing *only* numeric digits. This effectively immunizes the parameter against both SQL Injection (SQLi) and Cross-Site Scripting (XSS) by rejecting any non-numeric payload.

**XSS Virtual Patch (Rule 10004):** Targeted the `<script>` tag within the `name` field.

**Security Value:** This acts as a surgical intervention for the Reflected XSS vulnerability identified by ZAP. It drops the request at the WAF layer before the script can be echoed back to the user's browser.

### B. Policy Tuning & Availability (Rule 10002)

**Logic:** `SecRule REQUEST_URI "@contains setup.php" "id:10002...ctl:ruleRemoveById=942100"`

**Analysis:** Aggressive SQLi signatures in the CRS often generate "False Positives" during administrative tasks. Rule 10002 demonstrates **Strategic Tuning**—it disables a specific conflicting rule *only* for the setup page, ensuring security doesn't break business-critical functionality.

**C. HTTP Response Header Hardening** To address the "Medium" severity alerts found in the ZAP baseline, the following security headers were enforced via the custom configuration:

**X-Frame-Options ("DENY"):** Prevents the application from being loaded in an `<iframe>`, neutralizing **Clickjacking (UI Redressing)** attacks.

**X-Content-Type-Options ("nosniff"):** Disables "MIME-sniffing," forcing the browser to strictly follow the declared Content-Type. This prevents **MIME-type Confusion** attacks where an attacker tries to execute a text file as a script.

**X-XSS-Protection ("1; mode=block"):**
**Logic:** `Header set X-XSS-Protection "1; mode=block"`
**Analysis:** This activates the browser's built-in XSS filter. The `mode=block` setting ensures that if a reflected XSS attack is detected, the browser will refuse to render the entire page rather than just sanitizing the script, providing a much higher level of safety for the end-user.
.

---

## 7. CHALLENGES FACED (STAR ANALOGY)

**Situation:** Upon deploying the `RESPONSE-999-CUSTOM.conf` file, the Apache2 service failed to start, reporting a critical **AH00526 syntax error**. Additionally, during initial testing, custom rules appeared to be ignored in favor of the default OWASP CRS signatures.

**Task:** Restore service availability and ensure that custom "Virtual Patches" were being correctly prioritized and logged by the WAF engine.

**Action:**
1. **Dependency Resolution:** Used `sudo apache2ctl configtest` to isolate the error. Identified that Windows-style line endings (**CRLF**) from the initial rule creation were causing "Invalid Command" errors in the Linux environment; utilized `sed -i 's/\r//' [filename]` to sanitize the configuration file.
2. **Rule Isolation & Validation:** Observed that the OWASP CRS (900-series) was "shadowing" custom rules. To verify the efficacy of Rule 10001 and 10004, the corresponding CRS configuration files were temporarily renamed/isolated. This allowed for the definitive capture of custom Rule IDs in the `modsec_audit.log`.
3. **Service Restoration:** Performed a systematic restart of the Apache service (`systemctl restart apache2`) after each configuration change to verify the integrity of the rule-loading order.

**Result:** Service was restored with **"Syntax OK."** The "Chain of Evidence" was successfully established, with logs definitively proving that custom rules were providing high-fidelity protection alongside the CRS.

---

## 8. GAP ANALYSIS: REMAINING RISKS & MITIGATION

| **Remaining Alert** | **Risk** | **Mitigation Strategy** |
|---|---|---|
| **CSP Header Not Set** | Med | **Implementation:** Define a "Content Security Policy" header. This prevents unauthorized scripts from executing, neutralizing XSS even if the WAF is bypassed. |
| **HTTP Only Site** | Med | **Implementation:** Upgrade to HTTPS (TLS 1.3). This prevents Man-in-the-Middle (MITM) attacks that bypass WAF inspection via unencrypted traffic. |
| **Cookie w/o SameSite** | Low | **Implementation:** Update php.ini to set session.cookie_samesite=Lax. This prevents CSRF attacks by ensuring cookies are only sent to the original site. |

## 9. FURTHER IMPROVEMENTS & ROADMAP

**SIEM Forwarding:** Integrate modsec_audit.log with a platform like **Wazuh** or **ELK Stack**. This allows us to correlate WAF blocks with other system telemetry, such as failed SSH login attempts.
**Zero Trust Architecture:** Transition from a "Detection" mindset to a "Whitelisting" mindset for critical endpoints like login.php, only allowing specific character sets.
**CI/CD Integration:** Integrate ZAP into the development pipeline so that every time the app code changes, a new scan validates that the WAF rules still provide adequate coverage.

## 10. MITRE ATT&CK MAPPING (APPENDIX)

| **Attack Technique** | **MITRE ID** | **Mitigation Method** |
|---|---|---|
| **Exploit Public-Facing App** | T1190 | ModSecurity/CRS Request Inspection |
| **Directory/File Discovery** | T1083 | CRS Reconnaissance Signatures |
| **OS Command Injection** | T1059 | Custom Virtual Patch (Rule 10001) |
| **MIME-Type Abuse** | T1036 | X-Content-Type-Options Hardening |
| **Clickjacking** | T1189 | X-Frame-Options Header Enforcement |