Internship Report (Project-2): Web Application Firewall (WAF) Rule Development and Evasion Testing

Author: Deep Patel

Project: Web Application Firewall (WAF) Rule Development and Evasion Testing Lab

---

1.0 Executive Summary

This report documents the successful completion of a four-week project focused on designing, implementing, and testing a custom ruleset for a Web Application Firewall (WAF). A comprehensive security lab was established using an Ubuntu Server, an Apache web server, the ModSecurity WAF, and the Damn Vulnerable Web Application (DVWA) to simulate a real-world application environment. The primary objective was to protect the web application from common vulnerabilities, specifically SQL Injection (SQLi) and Cross-Site Scripting (XSS). Custom rules were developed, and a red-teaming phase was conducted where evasion techniques like encoding and obfuscation were used to bypass the initial rules. This iterative process of testing and refinement led to a hardened, resilient WAF ruleset that significantly enhances the application's security posture against web-based attacks.

2.0 Lab Environment & Methodology

2.1 Lab Topology

A secure, isolated virtual lab was created to ensure all attack simulations were safely contained.

- **WAF / Web Server Platform:** An Ubuntu Server VM (IP: 192.168.80.10) hosting the Apache web server, the ModSecurity WAF, and the DVWA vulnerable application.
- **Attacker Machine:** A Kali Linux VM (IP: 192.168.80.20) containing the attack and evasion toolkit, including sqlmap, Burp Suite, and web browser developer tools.
- **Network:** A private virtual network (192.168.80.0/24) was configured to connect the two VMs.

2.2 Tools & Technologies

- **Web Application Firewall (WAF):** ModSecurity
- **Web Server:** Apache
- **Vulnerable Application:** DVWA (Damn Vulnerable Web App)
- **Operating System:** Ubuntu Server
- **Virtualization:** VMware Workstation
- **Attack & Testing Tools:** sqlmap, Burp Suite, OWASP ZAP, Web browser developer tools

2.3 Methodology

Each phase of the project followed a consistent and reproducible methodology:

1. **Setup:** The Ubuntu server, Apache, ModSecurity, and DVWA were configured to create a functional testing environment.
2. **Baseline Test:** An attack (e.g., SQLi, XSS) was launched against DVWA to confirm its vulnerability before any custom rules were applied.
3. **Rule Development:** A custom ModSecurity rule was written to detect and block a specific attack signature.
4. **Verification:** The attack was re-launched to confirm that the WAF rule successfully blocked the malicious request.
5. **Evasion & Hardening:** Advanced techniques (e.g., encoding, obfuscation) were used to attempt to bypass the rule. The rule was then refined and hardened based on the results of the evasion testing.

3.0 Week 1: Lab Setup and Baseline Verification

The initial phase focused on building the lab environment and confirming the WAF was operational. The Ubuntu VM was set up with Apache, and ModSecurity was installed and configured with the default OWASP Core Rule Set (CRS). DVWA was deployed on the web server. A baseline test was performed to confirm both the application's vulnerability and the WAF's default functionality.

- **Attack:** A basic SQL injection payload was sent to the DVWA SQL Injection page: ' or 1=1--
- **Result (WAF Disabled):** The application executed the query and returned data, confirming the vulnerability.
- **Result (OWASP CRS Enabled):** The WAF successfully blocked the request, generating an alert in the audit log and proving the baseline security was functional.

4.0 Week 2: Custom Rule Development for Basic Attacks

This phase involved writing specific custom rules to block basic SQL injection and XSS payloads.

4.1 Basic SQL Injection (SQLi) Detection

- **Strategy:** Create a simple rule to detect a common SQLi pattern (union select) often used to exfiltrate data.
- **Rule:**
- SecRule ARGS "@rx union\s+select" "id:1000010,phase:2,block,msg:'Custom Rule: Basic SQL Injection Detected'"
- **Attack:** A union-based SQL injection payload was submitted via a web browser: 1' union select 1,2--
- **Verification:** The request was blocked by ModSecurity, and the custom message "Custom Rule: Basic SQL Injection Detected" was logged, confirming the rule's effectiveness.

4.2 Basic Cross-Site Scripting (XSS) Detection

- **Strategy:** Create a rule to detect the presence of the common <script> tag in user input, which is a primary indicator of a stored or reflected XSS attack.
- **Rule:**
- SecRule ARGS "@rx <script>" "id:1000011,phase:2,block,msg:'Custom Rule: Basic XSS Detected'"
- **Attack:** A simple XSS payload was submitted: <script>alert('XSS')</script>

- **Verification:** The WAF blocked the payload, preventing the script from being stored or reflected. The audit log showed the custom alert, verifying the rule was working as intended.

5.0 Week 3: Evasion Techniques and Advanced Rule Hardening

This phase focused on acting as an attacker to bypass the simple rules created in Week 2 and then hardening those rules to counter common evasion techniques.

5.1 SQLi Evasion (Comment and Case Obfuscation)

- **Evasion Technique:** Attackers often use SQL comments (/**/) and mixed case to break up and obfuscate keywords, which can bypass simple regular expressions.
- **Attack:** The following payload was used to bypass the basic rule: 1' uNiOn/**/sElEcT 1,2--
- **Result:** The simple rule from Week 2 failed to detect this payload, allowing the attack to proceed.
- **Rule Hardening:** The rule was improved by adding transformation functions (t:) to first convert the input to lowercase and then remove SQL comments before the regex check.
- SecRule ARGS "@rx union\s+select" "id:1000012,phase:2,block,t:lowercase,t:removeComments,msg:'Hardened Rule: SQLi with Obfuscation Detected'"
- **Verification:** The obfuscated attack was re-launched, and the hardened rule successfully normalized the payload, detected the malicious pattern, and blocked the request.

5.2 XSS Evasion (Payload Encoding)

- **Evasion Technique:** Attackers frequently use URL or HTML entity encoding to hide malicious characters from WAFs.
- **Attack:** A URL-encoded version of the XSS payload was submitted: %3Cscript%3Ealert('XSS')%3C/script%3E
- **Result:** The basic <script> rule failed because it was looking for literal characters, not their encoded equivalents.

- **Rule Hardening:** A URL decoding transformation function was added to the rule to ensure encoded payloads are analyzed correctly.
- SecRule ARGS "@rx <script>" "id:1000013,phase:2,block,t:lowercase,t:urlDecode,msg:'Hardened Rule: XSS Detected after URL Decode'"
- **Verification:** The encoded XSS attack was launched again. The hardened rule first decoded the input, then identified the <script> tag and blocked the attempt.

6.0 Week 4: Final Rule Refinement and Penetration Testing

The final phase was dedicated to refining the custom ruleset based on all previous evasion tests and conducting a final round of penetration testing to verify its overall resilience.

6.1 Final Hardening: Chaining Rules for Higher Fidelity

- **Strategy:** To reduce false positives, multiple rules can be chained together. An alert is only triggered if all conditions in the chain are met. For SQLi, we can chain a rule that looks for SQL keywords with another that looks for characters that end a statement.
- **Hardened Rule:**
- SecRule ARGS "@rx (select|union|insert|update|delete)" "id:1000014,phase:2,block,t:lowercase,t:removeComments,msg:'Hardened Rule: Multi-Stage SQLi Attack Detected',chain"
- SecRule ARGS "@rx (--|;|\#)" "t:lowercase"
- **Result:** This chained rule is more precise. It only blocks requests that contain both a common SQL command *and* a statement terminator, making it more resilient and less likely to block legitimate traffic.

6.2 Final Verification with Automated Tooling

- **Test:** The automated SQL injection tool sqlmap was run against the DVWA instance protected by the final, hardened WAF ruleset.
- **Attack Command:**

Bash

sqlmap -u "http://192.168.80.10/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="<session_cookie>" --batch --level=5 --risk=3

- **Result:** sqlmap was unable to find any SQL injection vulnerabilities. The ModSecurity audit logs showed hundreds of blocked requests originating from the sqlmap scanner, with alerts triggered by the custom hardened rules. This confirmed that the final ruleset successfully defended the application against an advanced, automated attack simulation.

7.0 Conclusion & Key Learnings

This project successfully demonstrated the end-to-end process of developing, testing, and hardening a custom WAF ruleset. A functional security lab was built to test custom rules against common and evasive SQLi and XSS attacks. The iterative red-teaming process proved crucial in identifying weaknesses in initial rules and building a more robust and resilient final configuration.

The key learnings from this project include:

- **The Attacker's Mindset:** Actively attempting to bypass one's own defenses is one of the most effective ways to discover security gaps and build stronger rules.
- **The Power of Transformations:** Simple signature matching is insufficient. Using ModSecurity's transformation functions is essential for detecting obfuscated and encoded attack payloads.
- **Importance of Log Analysis:** Analyzing WAF logs is critical for understanding how attackers are attempting to bypass rules and provides the necessary feedback to refine and harden the security posture.
- **Iterative Hardening:** Security is not a one-time setup. The process of testing, detecting evasions, and refining rules is a continuous cycle required to stay ahead of evolving attack techniques.