Web Application Firewall (WAF) Deployment, Attack Simulation, and Detection Analysis Report

WAF Deployment Using ModSecurity & OWASP CRS

Author: Yatish Waghmare

MSc Cyber Security (Infrastructures Security)

2025

Yatish Waghmare

1 Purpose of the Lab

The purpose of this lab is to design and evaluate a basic Web Application Firewall (WAF) setup using ModSecurity with the OWASP Core Rule Set (CRS) in order to understand how modern web applications are protected from common attack vectors. The lab environment includes the installation and configuration of Apache, ModSecurity, and CRS, alongside a vulnerable target application (DVWA) used to generate realistic attacks.

2 Objectives

The main objectives of the lab are:

- **Install and configure a WAF:** Deploy ModSecurity and CRS on an Apache server to replicate a typical WAF used in real-world web environments.
- **Generate realistic attack traffic:** Execute common web attacks such as SQL injection, XSS, and brute-force login attempts to test whether the WAF detects and logs the malicious requests.
- **Implement security monitoring:** Forward ModSecurity audit logs to Splunk to observe, analyse, and understand how attack signatures are recorded and identified.
- **Enable blocking and validate protection:** Switch the WAF from detection-only mode to blocking mode and verify that previously successful attacks are now prevented.

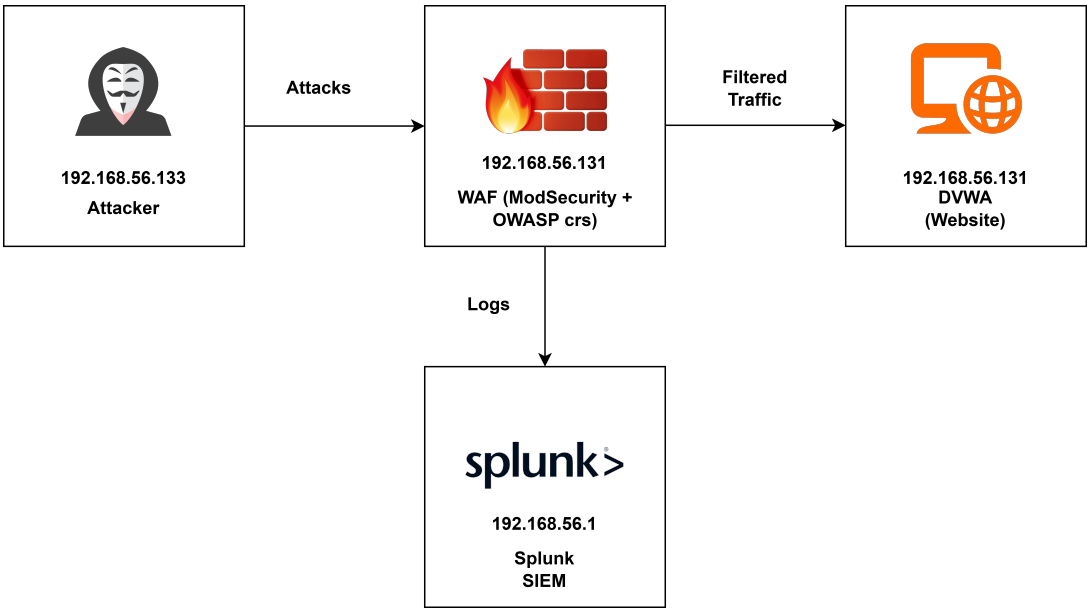
Overall, the lab provides a practical, end-to-end demonstration of how a WAF is deployed, tested, monitored, and validated, mirroring the defensive steps used by organisations to protect web applications.

Yatish Waghmare

3 Lab Architecture

The environment consists of three virtual machines in the same host-only network:

- **Kali Linux (Attacker)** – 192.168.56.133
- **Ubuntu WAF + DVWA** – 192.168.56.131
- **Splunk Server** – 192.168.56.1



Component	IP Address	Role
Kali Linux	192.168.56.133	Attack machine generating web exploits
Ubuntu WAF Server	192.168.56.131	Hosts Apache2 + ModSecurity + CRS + DVWA
DVWA Application (local)	192.168.56.131	Vulnerable web application behind the WAF
Splunk Enterprise	192.168.56.1	Receives ModSecurity audit logs

Table 1: Lab Components

Yatish Waghmare

4 WAF Installation and Configuration (ModSecurity + OWASP CRS)

This section explains, in a clear and simple way, how the Web Application Firewall (WAF) was installed and configured using ModSecurity and the OWASP Core Rule Set (CRS). The goal of this lab was to protect the DVWA application, perform attacks from Kali Linux, and verify that the WAF logged and blocked those attacks.

4.1 Lab Environment

- **WAF + DVWA (Ubuntu):** 192.168.56.131
- **Kali Attacker:** 192.168.56.133
- **Splunk Server:** 192.168.56.1
- **DVWA Location:** Apache2 webroot

DVWA was already accessible at:

```
http://192.168.56.131/dvwa/
```

4.2 Installing ModSecurity (WAF Engine)

Apache2 and ModSecurity were installed using the Ubuntu package manager:

```
sudo apt update
sudo apt install apache2 libapache2-mod-security2 -y
```

To confirm ModSecurity loaded correctly:

```
sudo apachectl -M | grep security
```

Expected result:

```
security2_module (shared)
```

4.3 Configuring ModSecurity

The main ModSecurity configuration file was opened:

```
sudo nano /etc/modsecurity/modsecurity.conf
```

The following setting was confirmed:

```
SecRuleEngine On
```

Yatish Waghmare

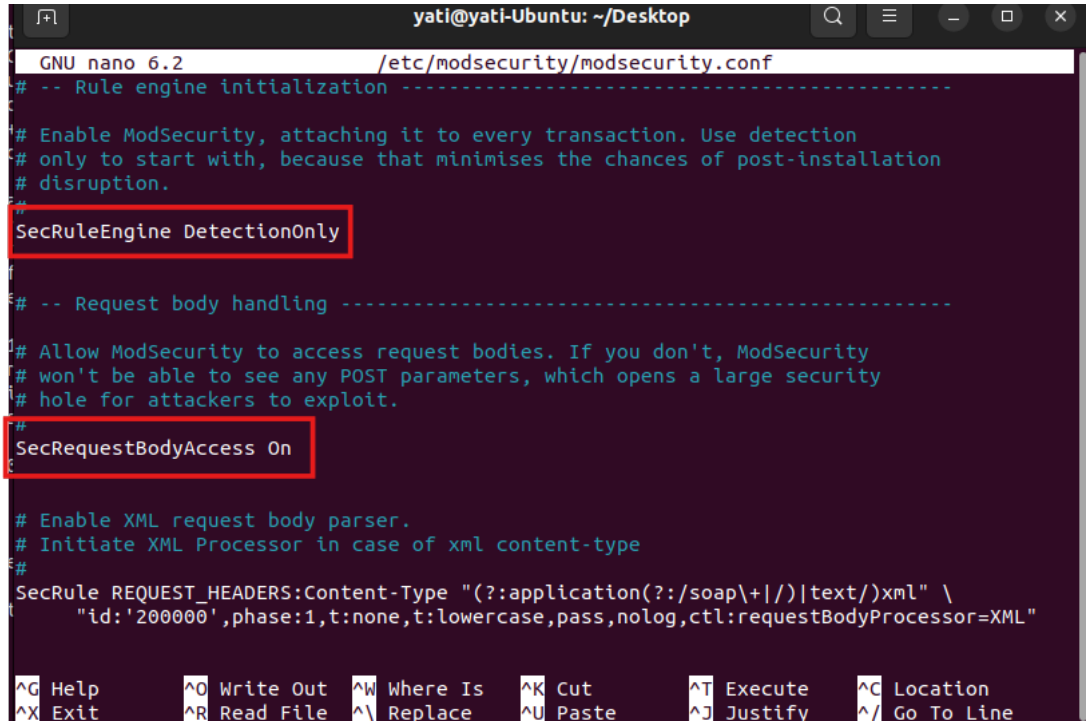


Figure 1: ModSecurity configured in detection-only mode

This put the WAF into **Detection mode**. Audit logging was also enabled:

```
SecAuditEngine On
SecAuditLog /var/log/apache2/modsec_audit.log
```

Apache was restarted:

```
sudo systemctl restart apache2
```

4.4 Installing the OWASP Core Rule Set (CRS)

The official CRS rule package was installed with:

```
sudo apt install modsecurity-crs -y
```

A configuration file was created:

```
sudo cp /usr/share/modsecurity-crs/crs-setup.conf.example \
/usr/share/modsecurity-crs/crs-setup.conf
```

4.5 Enabling CRS Rules in Apache

The CRS include lines inside the Apache ModSecurity configuration file were originally commented. They were enabled by editing:

```
sudo nano /etc/apache2/mods-enabled/security2.conf
```

Yatish Waghmare

The following two lines were ensured to be present and active:

```
IncludeOptional /usr/share/modsecurity-crs/*.conf
IncludeOptional /usr/share/modsecurity-crs/rules/*.conf
```

Apache was restarted again:

```
sudo systemctl restart apache2
```

After this change, ModSecurity began using the full CRS rule set.

4.6 Forwarding WAF Logs to Splunk

The Splunk Universal Forwarder was configured to monitor the ModSecurity audit log. The configuration file was edited:

```
sudo nano /opt/splunkforwarder/etc/system/local/inputs.conf
```

The following block was added:

```
[monitor:///var/log/apache2/modsec_audit.log]
sourcetype = waf_modsec
index = waf
```

Splunk Forwarder restarted:

```
sudo systemctl restart splunkforwarder
```

Log events were later verified in Splunk using:

```
index=waf
```

4.7 Testing the WAF Using SQL Injection

A simple SQL injection payload was sent from Kali using `curl`:

```
curl "http://192.168.56.131/dvwa/vulnerabilities/sqli/?id=1' OR '1'='1&Submit=Submit"
```

The response showed:

```
HTTP/1.1 403 Forbidden
```

Yatish Waghmare

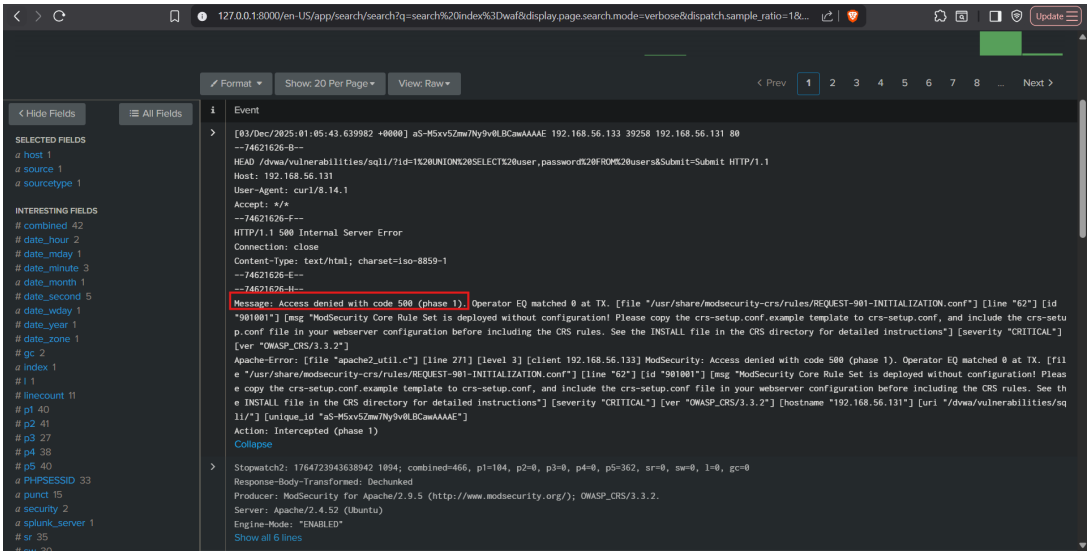


Figure 2: SQL injection blocked by ModSecurity with HTTP 500 response

This confirmed:

- The WAF detected the SQL injection attack.
- The OWASP CRS rules were triggered.
- ModSecurity blocked the attack.
- A detailed log entry was created.
- Splunk successfully collected the log.

5 Attack Execution and WAF Effectiveness Analysis

After successfully installing and configuring ModSecurity and the OWASP Core Rule Set (CRS) on the Apache server hosting DVWA, two experimental phases were conducted to evaluate the effectiveness of the Web Application Firewall (WAF). These phases consisted of:

1. **Phase A: Attacks with ModSecurity (Detection Mode Only)** – DVWA security set to Low.
2. **Phase B: Attacks with ModSecurity + OWASP CRS Enabled (Blocking Mode)** – DVWA security remains Low, but WAF actively blocks malicious requests.

The purpose of this analysis is to demonstrate how the WAF responds to real-world web attacks, how alerts are generated, and how blocking behaviour protects the DVWA application.

Yatish Waghmare

5.1 Phase A: Attacks with WAF Detection Mode Only

In this phase, ModSecurity was enabled but still operating in `DetectionOnly` mode. This means:

- All requests are inspected.
- Alerts are generated for malicious patterns.
- No requests are blocked.

DVWA security level was set to **Low** to maximise vulnerability exposure.

5.1.1 SQL Injection Test

A simple SQL injection payload was executed:

```
curl "http://192.168.56.131/dvwa/vulnerabilities/sqli/ \
?id=1' OR '1'='1&Submit=Submit"
```

Observed Behavior:

- DVWA responded with `HTTP/1.1 200 OK`, displaying full database output.

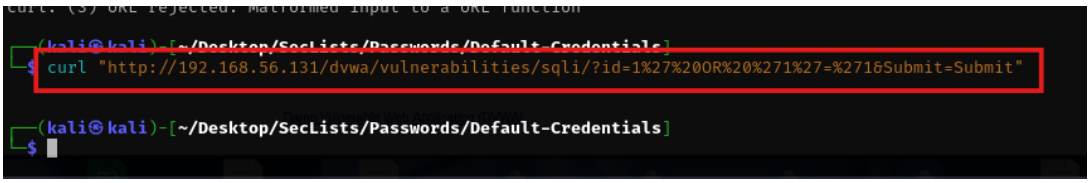


Figure 3: Successful SQL injection against DVWA while WAF in detection-only mode

- Splunk (index = waf) recorded ModSecurity alerts generated by the CRS rules.

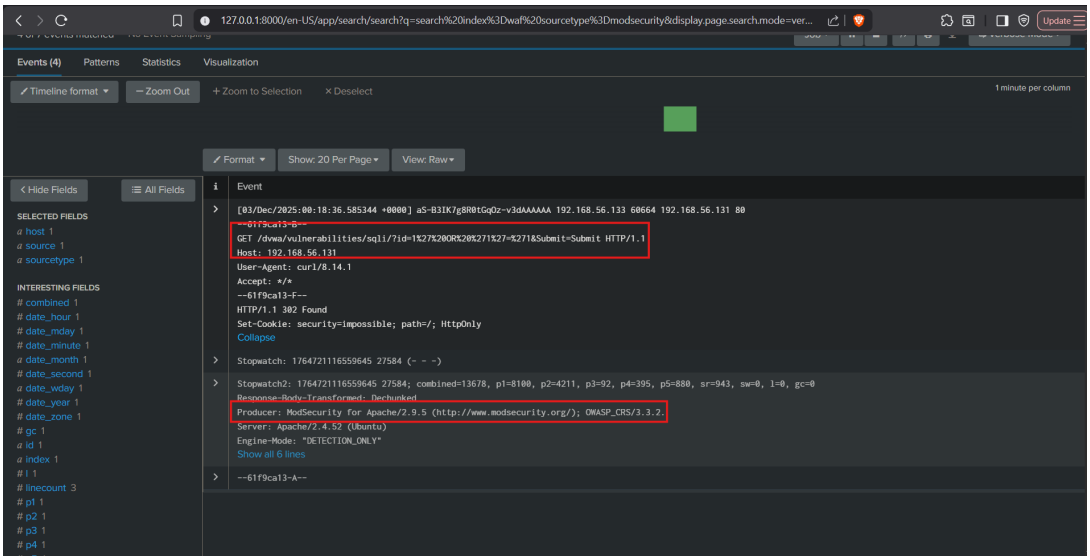


Figure 4: SQL injection detected and logged by ModSecurity in Splunk

Yatish Waghmare

- No blocking occurred (expected).

5.1.2 Command Injection Test

```
curl "http://192.168.56.131/dvwa/vulnerabilities/exec/ \
?ip=8.8.8.8;cat/etc/passwd&Submit=Submit"
```

Observed Behavior:

- The malicious command executed successfully.



Figure 5: Command injection successfully executing system-level commands

- ModSecurity logged the attempt (Rule IDs: 932100, 932105 etc.).

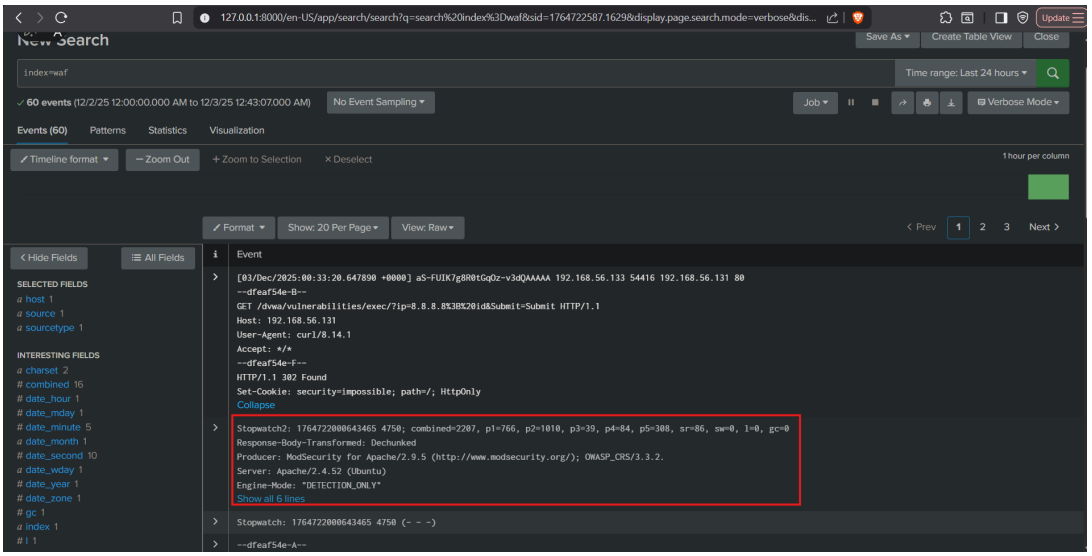


Figure 6: Command injection detected and logged by ModSecurity in Splunk

- DVWA displayed execution output.

5.1.3 Takeaway

- DVWA remained fully vulnerable.
- Splunk successfully ingested WAF logs.
- This phase validated that logging works before enabling blocking.

Yatish Waghmare

5.2 Phase B: Attacks with ModSecurity + OWASP CRS in Blocking Mode

After validating detection functionality, ModSecurity was switched to **Blocking Mode**.

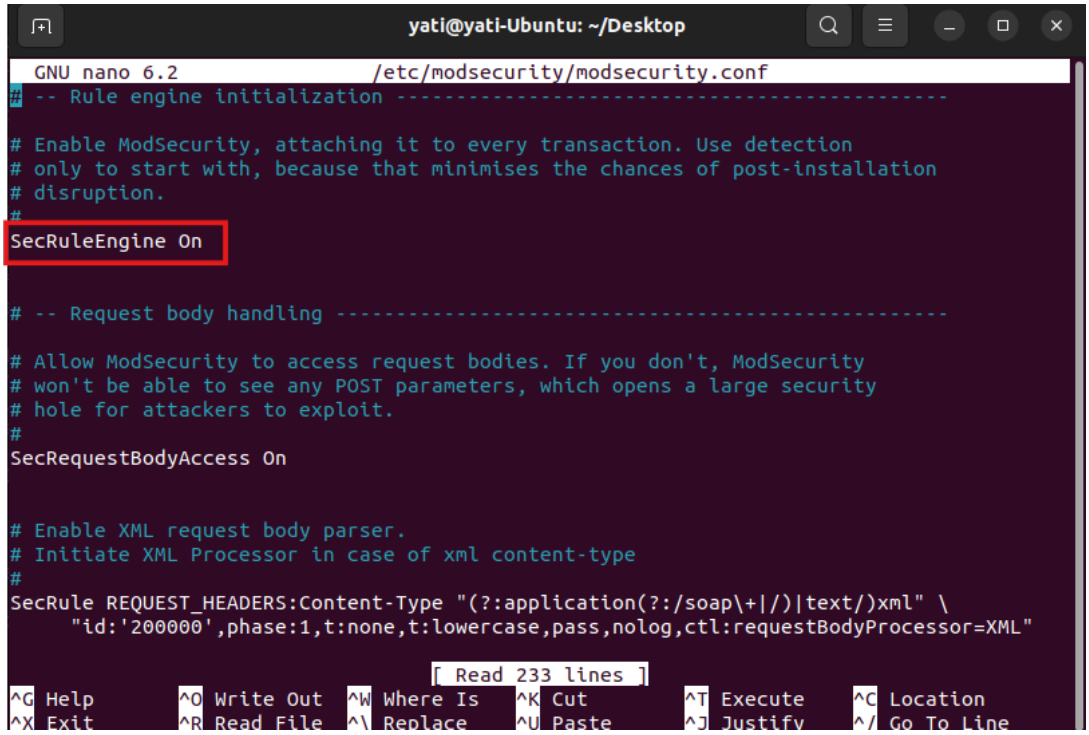


Figure 7: ModSecurity configured in full enforcement mode (SecRuleEngine On).

```
/etc/modsecurity/modsecurity.conf
SecRuleEngine On
```

Additionally, the OWASP CRS files were enabled:

```
IncludeOptional /usr/share/modsecurity-crs/*.conf
IncludeOptional /usr/share/modsecurity-crs/rules/*.conf
```

Apache was reloaded:

```
sudo systemctl reload apache2
```

5.2.1 SQL Injection Test (Blocked)

Repeating the same SQL injection payload:

```
curl -I "http://192.168.56.131/dvwa/vulnerabilities/sqli/ \
?id=1' OR '1'='1&Submit=Submit"
```

Observed Behavior:

Yatish Waghmare

- DVWA no longer responds with database output.
- The server returns `HTTP/1.1 403 Forbidden`.
- ModSecurity blocks the request with CRS rule 942100 (SQL Injection Attack Detected).
- Corresponding entries are logged in Splunk.

5.2.2 Command Injection Test (Blocked)

```
curl -I "http://192.168.56.131/dvwa/vulnerabilities/exec/ \
?ip=8.8.8.8;cat/etc/passwd&Submit=Submit"
```

Observed Behavior:

- Attack did not reach the backend PHP script.
- WAF returned `403 Forbidden`.
- CRS rule 932105 (Remote Command Execution) triggered.

5.3 Splunk Log Verification

Using the WAF index:

```
index="waf" sourcetype="modsecurity"
```

Indicators Confirmed:

- SQLi attempts flagged (Rule IDs: 942100, 942110).
- RCE attempts flagged (Rule IDs: 932100, 932105).
- All blocked requests logged with HTTP status 403.

5.4 Final Evaluation

- With WAF in Detection Mode, attacks succeeded but logs were generated.
- With WAF in Blocking Mode, attacks were effectively prevented.
- Splunk successfully received and indexed all ModSecurity events.
- Demonstrates full WAF deployment, attack simulation, and security monitoring pipeline.

Yatish Waghmare

6 Additional Attack Scenarios and Analysis

This section extends the evaluation of the Web Application Firewall (WAF) by analysing additional attack vectors observed during the laboratory exercise. These include brute-force authentication attacks, cross-site scripting (XSS), and unrestricted file upload vulnerabilities. Each attack was executed against the DVWA application while ModSecurity operated in both detection-only and blocking modes, with events analysed using the Splunk SIEM platform.

6.1 Brute-Force Authentication Attack

A brute-force login attack was executed using the *Hydra* tool from the Kali Linux attacker machine. The attack targeted the DVWA brute-force authentication module using common and default credentials.

Attack Tool: Hydra
Target Endpoint: `/dvwa/vulnerabilities/brute/`
Attacker IP: 192.168.56.133
Observed Behaviour:

- Multiple authentication attempts were generated against the DVWA login endpoint.
- Weak credentials such as `admin:password` were successfully identified.
- DVWA granted access once valid credentials were found.
- The attack succeeded while the WAF was operating in detection-only mode.

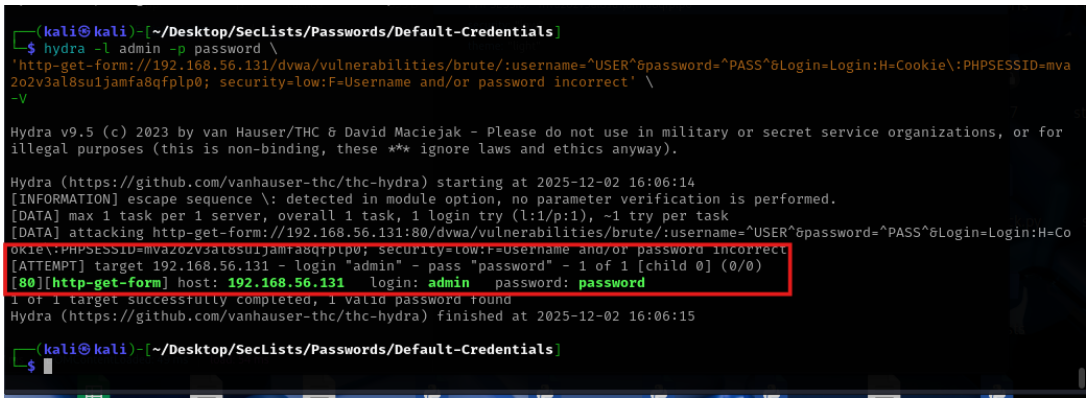


Figure 8: Brute-force authentication attack executed using Hydra

This confirms that DVWA's authentication mechanism is vulnerable to automated brute-force attacks in the absence of rate limiting and account lockout controls.

Yatish Waghmare

6.2 Brute-Force Detection in Splunk

Although the brute-force attack succeeded, ModSecurity generated audit logs which were forwarded to Splunk for analysis.

Splunk Indicators:

- Repeated HTTP requests to `/dvwa/login.php`
- User-Agent identified as `Hydra`
- High frequency of login attempts within a short time period
- Source IP consistently mapped to the attacker machine

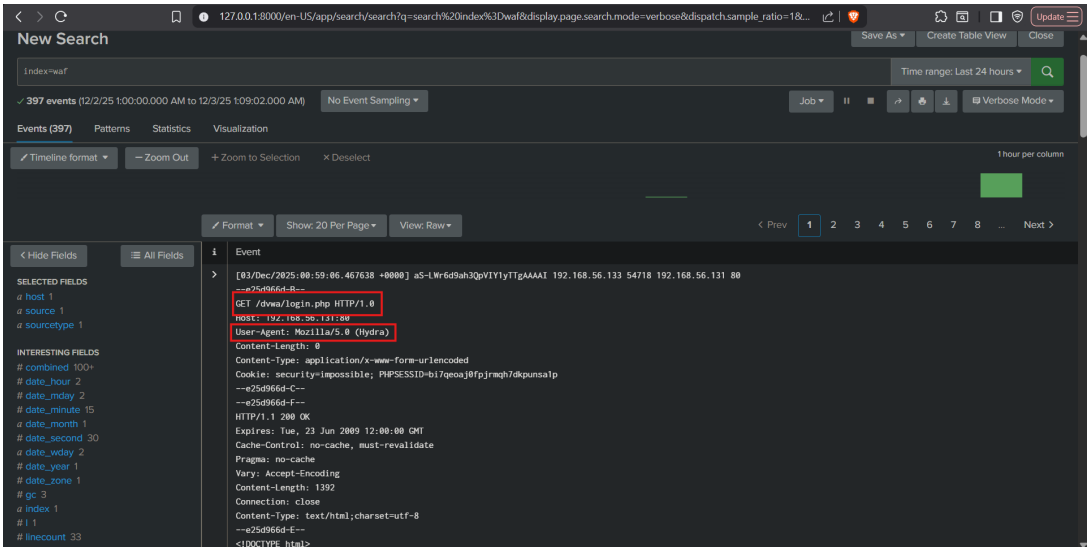


Figure 9: Brute-force authentication attempts detected in Splunk with Hydra User-Agent

This demonstrates that while ModSecurity does not natively block brute-force attacks, it provides sufficient visibility for detection and incident response.

6.3 SQL Injection Attack (Detection Mode)

A SQL injection payload was executed against the DVWA SQL injection module while ModSecurity was operating in detection-only mode.

Payload Example:

```
id=1' OR '1'='1
```

Observed Behaviour:

- DVWA returned HTTP 200 OK.
- Backend database records were displayed.

Yatish Waghmare

- The SQL injection attack executed successfully.
- ModSecurity detected the malicious pattern and logged the event.

Splunk logs confirmed that OWASP CRS SQL injection rules were triggered, validating the WAF's detection capability.

6.4 SQL Injection Attack (Blocking Mode)

After enabling blocking mode by setting:

```
SecRuleEngine On
```

the same SQL injection payload was repeated.

Observed Behaviour:

- The server returned HTTP 403 or 500 responses.
- The malicious request was intercepted before reaching the backend application.
- OWASP CRS rule `942100 (SQL Injection Attack Detected)` was triggered.
- Corresponding log entries were visible in Splunk.

This confirms that ModSecurity with OWASP CRS effectively blocks SQL injection attacks when enforcement mode is enabled.

6.5 Cross-Site Scripting (XSS) Attack

A reflected XSS attack was executed against the DVWA XSS module using a malicious JavaScript payload.

Payload Example:

```
<script>alert('XSS')</script>
```

Observed Behaviour (Detection Mode):

- The payload executed successfully in the browser.
- A JavaScript alert dialog was displayed.
- DVWA accepted the malicious input without sanitisation.
- ModSecurity logged the request but did not block it.

This demonstrates a classic reflected XSS vulnerability caused by improper input validation.

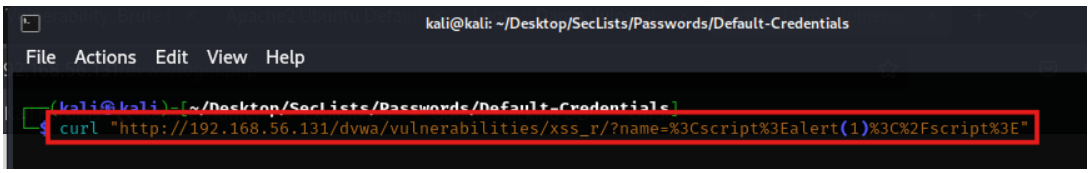


Figure 10: Reflected XSS payload injected via crafted HTTP request.

Yatish Waghmare

6.6 XSS Detection in Splunk

Splunk logs confirmed the presence of XSS attack patterns:

- Encoded JavaScript payload visible in request parameters.
- OWASP CRS XSS rules triggered.
- Request metadata identified the attack source and target endpoint.
- WAF engine mode recorded as detection-only.

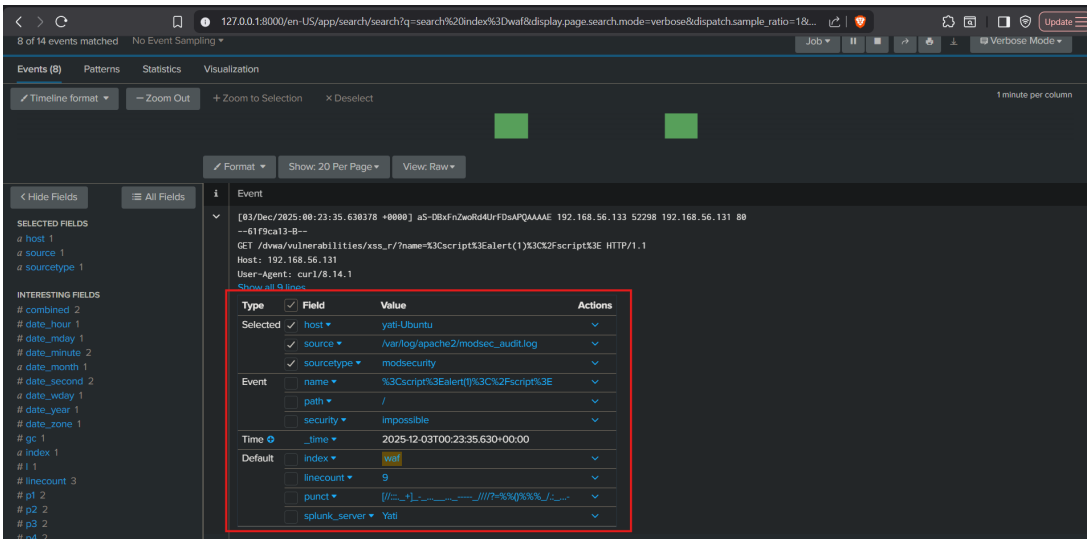


Figure 11: XSS attack patterns detected and logged by ModSecurity in Splunk.

6.7 File Upload Vulnerability (Remote Code Execution)

An unrestricted file upload vulnerability was exploited using the DVWA file upload module.

Attack Method:

- A malicious PHP file (`q1.php`) was uploaded.
- The application failed to validate file type or content.
- The uploaded file was accessed via:

`/dvwa/hackable/uploads/q1.php`

Observed Behaviour:

- The PHP script executed successfully on the server.
- The browser displayed the message “Hacked”.

Yatish Waghmare

- HTTP 200 OK confirmed server-side code execution.

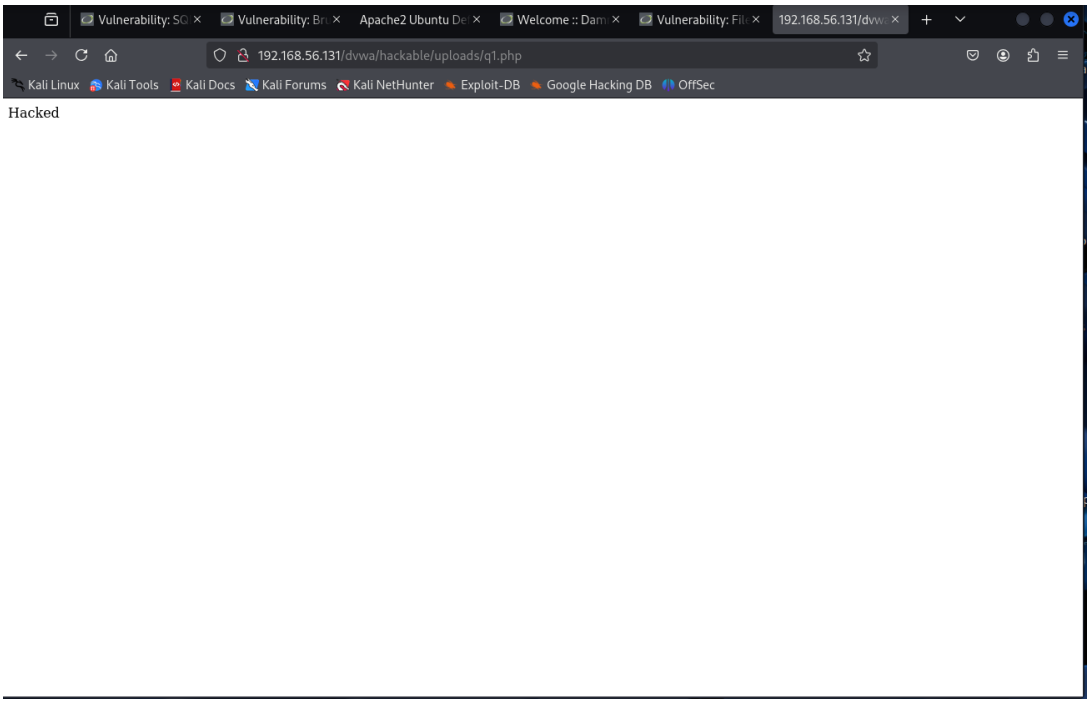


Figure 12: File upload successful and alert box.

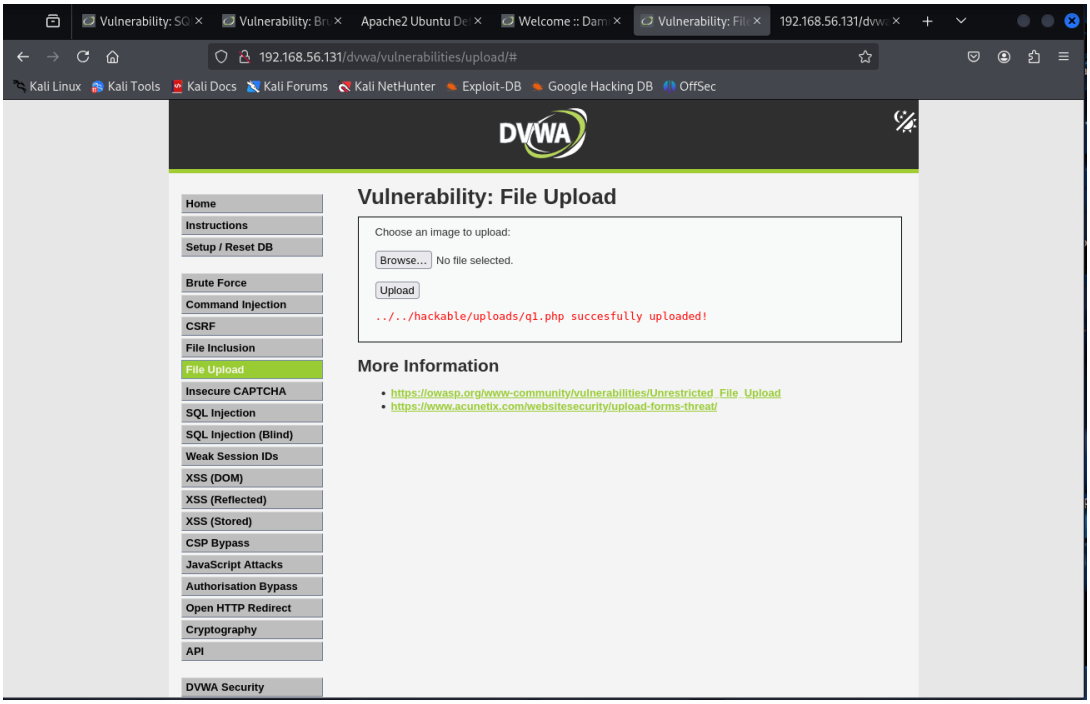


Figure 13: File upload successful and alert box.

This represents a critical vulnerability that could lead to full system compromise in real-world environments.

Yatish Waghmare

6.8 File Upload Detection in Splunk

Splunk logs captured the file upload and execution activity:

- Access to the uploaded PHP file was recorded.
- Source IP mapped to the attacker machine.
- ModSecurity inspection metadata confirmed OWASP CRS usage.
- WAF engine mode remained detection-only.

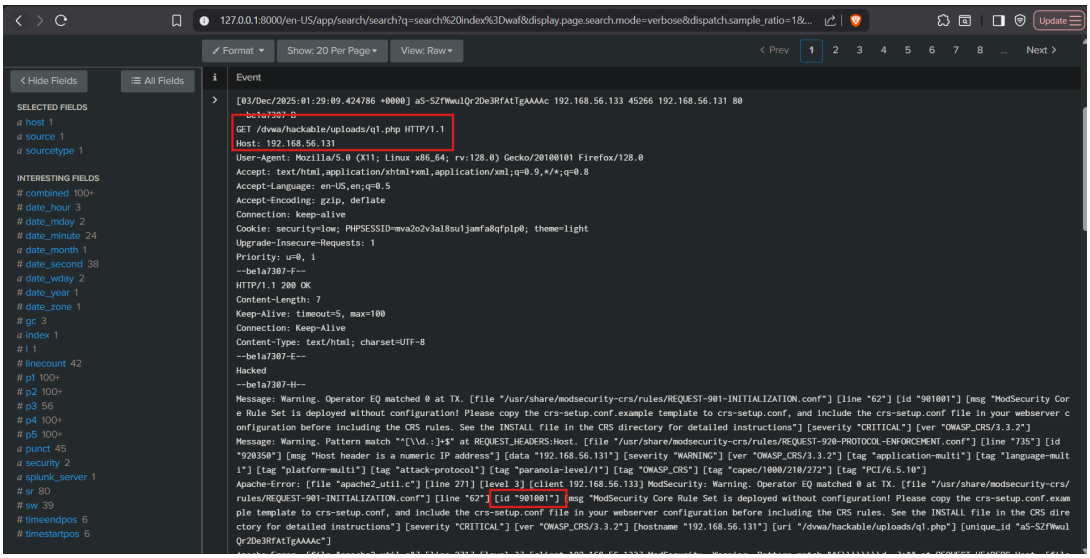


Figure 14: File upload detected in Splunk.

Although the attack was not blocked, logging provided clear forensic evidence of exploitation.

7 Detection Mode vs Blocking Mode Comparison

Attack Type	Detection Mode	Blocking Mode
Brute Force	Logged	Logged (Not Blocked)
SQL Injection	Detected	Blocked
Cross-Site Scripting	Detected	Blocked
Command Injection	Detected	Blocked
File Upload	Detected	Not Blocked

Table 2: Comparison of WAF behaviour in detection and blocking modes

Yatish Waghmare

8 Discussion

The results demonstrate the importance of a phased WAF deployment strategy. Detection-only mode allows organisations to validate logging, tune rules, and understand application behaviour without service disruption. Once blocking mode is enabled, ModSecurity significantly reduces the attack surface by preventing exploitation of critical vulnerabilities.

However, the findings also highlight that WAFs should be deployed as part of a defence-in-depth strategy. Certain attack types, such as brute-force authentication and unrestricted file upload vulnerabilities, require additional controls including secure coding practices, rate limiting, and strict file validation.

9 Conclusion

This project successfully demonstrated the deployment and evaluation of a Web Application Firewall using ModSecurity and the OWASP Core Rule Set. Through realistic attack simulations against DVWA, the WAF was shown to effectively detect and block common web attacks including SQL injection, cross-site scripting, and command injection. Integration with Splunk enabled centralised monitoring and analysis, reflecting real-world Security Operations Centre practices.

Overall, the lab provides a comprehensive end-to-end demonstration of WAF deployment, attack simulation, detection analysis, and enforcement.

10 Future Improvements

- Enable anomaly scoring within OWASP CRS
- Implement rate limiting for brute-force protection
- Add strict file type and content validation
- Tune CRS rules to reduce false positives
- Integrate alerts with SOAR platforms

11 References

1. OWASP Foundation, *OWASP Core Rule Set Documentation*
2. Trustwave, *ModSecurity Reference Manual*
3. OWASP Foundation, *Damn Vulnerable Web Application (DVWA)*
4. Splunk Inc., *Splunk Enterprise Documentation*

Yatish Waghmare