WAF HOMELAB

PROJECT

Author: viitheone

Project Type: Cybersecurity Homelab

Environment: Virtualized Local Infrastructure

1. Introduction

Modern web applications are constantly exposed to a wide range of attacks such as SQL Injection, Cross-Site Scripting (XSS), and denial-of-service attempts. Traditional perimeter security controls are often insufficient to protect application-layer traffic, making **Web Application Firewalls (WAFs)** a critical defensive component.

This project documents the design and implementation of a **Web Application Firewall home lab**, built to simulate real-world attack and defense scenarios in a controlled, isolated environment. The lab demonstrates how a WAF deployed as a **reverse proxy with TLS termination** can inspect, detect, and mitigate malicious web traffic before it reaches a vulnerable backend application.

The lab environment was intentionally designed to balance **realism** and **resource efficiency**, making it suitable for hands-on learning while preserving industry-aligned architecture principles.

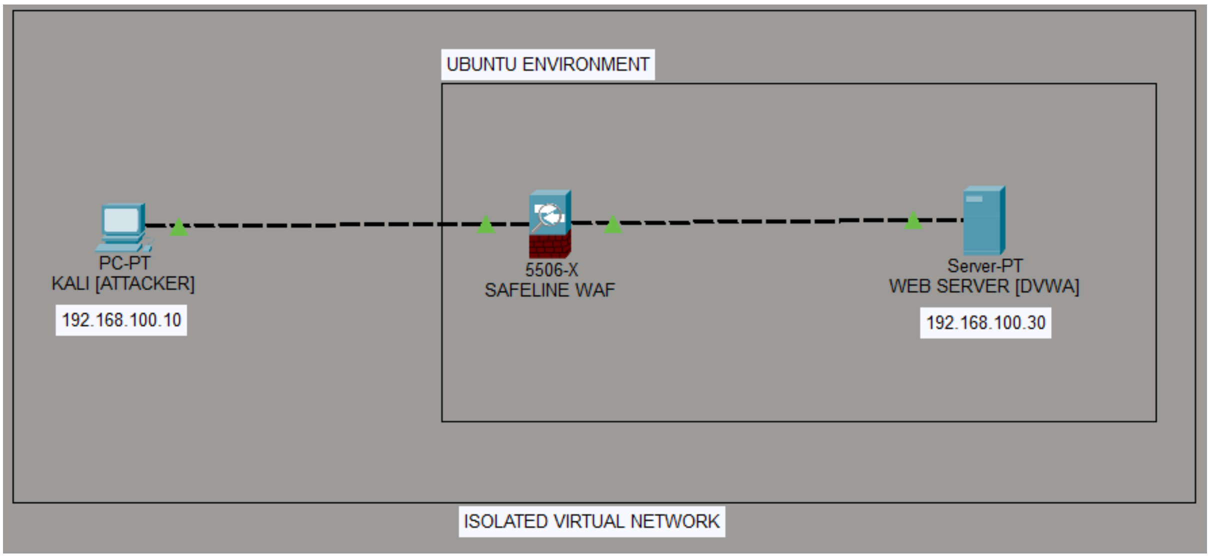
2. Lab Objectives

The primary objectives of this lab are:

- To design a multi-tier web security architecture.
- To deploy a vulnerable web application for controlled testing.
- To implement a WAF as a reverse proxy.
- To terminate TLS at the WAF layer for traffic inspection.
- To simulate common web attacks from an attacker machine.
- To observe and analyze WAF detection and mitigation behavior.
- To understand the security benefits of backend isolation.

3. Network Architecture

The lab follows a **three-tier architecture** commonly found in production environments.



Architectural Design Rationale

- **Reverse Proxy Placement:**

  The WAF is positioned between the client and backend server to enforce inspection and filtering of all inbound traffic.

- **TLS Termination at the WAF:**

  Encrypted HTTPS traffic is decrypted at the WAF, allowing inspection of payloads that would otherwise be opaque.

- **Backend Isolation:**

  The vulnerable application is not directly exposed to the client and only accepts traffic originating from the WAF.

- **Port Separation:**

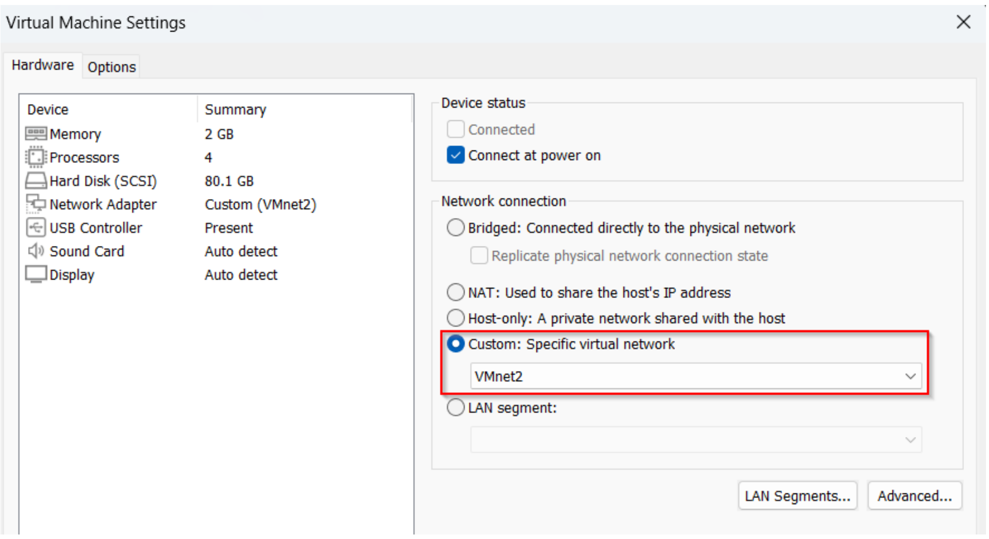
  Public-facing traffic uses port 443, while backend communication occurs over port 8080.

4. Network Design & Segmentation

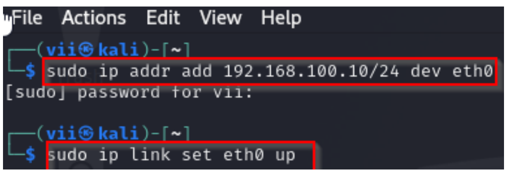
The lab is deployed within an **isolated virtual network** using VMware Workstation to prevent unintended exposure to the host system or external networks.

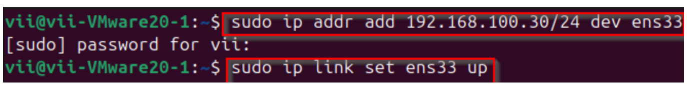
Network Characteristics

- Internal virtual network (VMnet2)



- Static IP addressing for predictability





- No direct internet exposure for backend services
- Controlled east-west traffic flow

Device Roles

Machine	**Role**
Kali Linux	Attacker / Client
Safeline WAF	Reverse Proxy & Security Inspection
Ubuntu Server	DVWA Backend

Logical separation is shown in diagrams for architectural clarity, even though services may be co-located in lab environments due to resource constraints.

5. Backend Application Setup (DVWA)

5.1 Application Choice

The **Damn Vulnerable Web Application (DVWA)** was selected as the backend application due to its intentionally insecure design, which allows controlled testing of common web vulnerabilities.

DVWA simulates real-world coding flaws and provides adjustable security levels, making it suitable for demonstrating WAF effectiveness.

5.2 LAMP Stack Overview

The backend application is hosted on a **LAMP stack**, consisting of:

- **Linux** – Operating system
- **Apache** – Web server
- **MySQL/MariaDB** – Database backend
- **PHP** – Server-side scripting language

This stack enables dynamic content generation and database-driven vulnerabilities required for realistic attack simulation.

5.3 Backend Port Configuration

By default, Apache listens on port 80. In this lab, Apache was configured to listen on **port 8080** instead.

Reasoning:

- Prevents accidental direct access from clients
- Enforces all traffic to flow through the WAF

- Mimics backend service isolation found in real environments

6. TLS / SSL Certificate Generation

To support HTTPS, a **self-signed TLS certificate** was generated and deployed on the WAF.

6.1 Why TLS Is Required

TLS provides:

- Encryption of client-server communication
- Integrity protection against tampering
- Server authentication (to the extent allowed by self-signed certificates)

Although self-signed certificates are not trusted by default, they are suitable for lab environments and allow full encryption and inspection workflows.

6.2 Certificate Generation Process

The certificate generation followed a standard Public Key Infrastructure (PKI) flow:

1. **Private Key Generation**

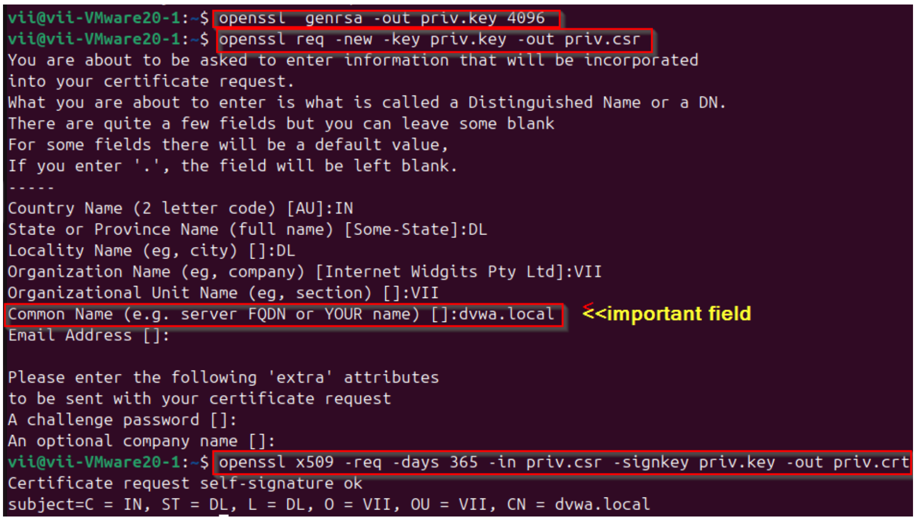
   A 4096-bit RSA private key was generated to ensure strong cryptographic security.

2. **Certificate Signing Request (CSR)**

   The CSR defines the server identity and binds it to the public key.

3. **Self-Signing**

   The CSR was signed using the private key to generate a self-signed certificate.



NOTE: When you generate the CSR, if you left Common Name blank, that means your certificate has no proper hostname bound to it. That can cause browser warnings beyond normal self-signed warning.

This process mirrors how certificates are issued in production, except that a trusted Certificate Authority is not involved.

6.3 TLS Termination Point

TLS is terminated at the **WAF**, not at the backend.

Client → Encrypted HTTPS → WAF → Decrypted HTTP → Backend

This design allows the WAF to:

- Inspect decrypted payloads
- Detect malicious patterns
- Enforce security policies

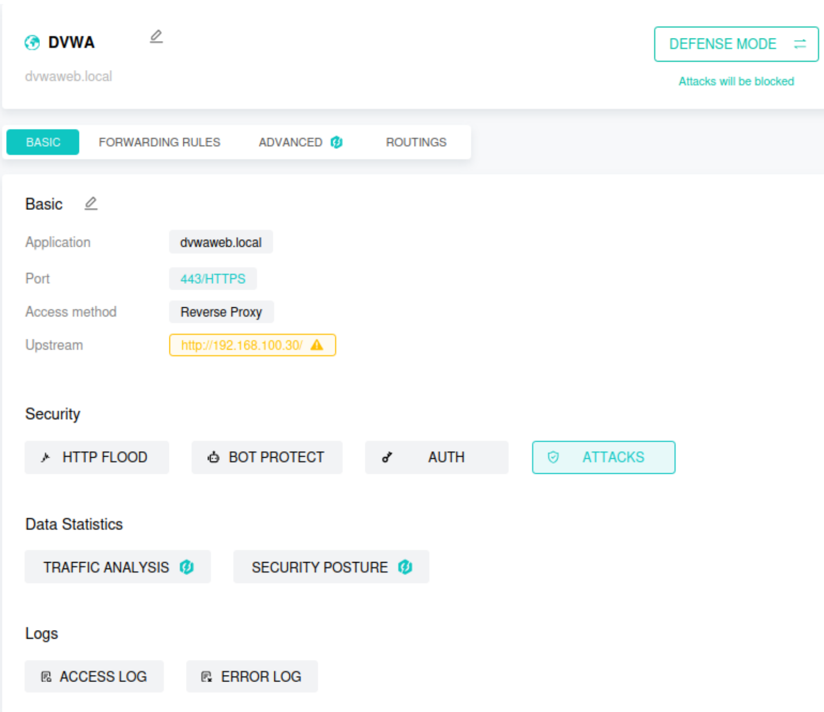
7. WAF Deployment & Reverse Proxy Configuration

The WAF is configured to operate as a **reverse proxy**, acting as the single entry point for all web traffic.

Responsibilities of the WAF

- Accept HTTPS traffic from clients
- Perform TLS decryption
- Inspect application-layer requests
- Forward legitimate traffic to backend
- Block or drop malicious requests
- Log security events and access data

The backend application trusts only the WAF as a traffic source.



8. Attack Simulation & Testing

8.1 Attacker Environment

Kali Linux is used as the attacker machine, simulating a malicious client attempting to exploit vulnerabilities in the web application.

8.2 SQL Injection Testing

SQL injection payloads were executed against DVWA, such as:

admin' OR '1'='1

Observed Behavior:

- Payload intercepted at the WAF
- Malicious pattern detected
- Request blocked before reaching backend
- Event logged for analysis

8.3 Additional Tests

- HTTP flood testing
- Custom IP deny rules
- Manual rule enforcement

These tests validate the WAF's ability to mitigate both application-layer and volumetric threats.

9. Observations & Analysis

- TLS termination enabled deep inspection of encrypted traffic
- Backend isolation significantly reduced attack surface
- WAF logging provided visibility into attack attempts
- Signature-based detection was effective but may produce false positives

10. Limitations

- Self-signed certificates are not trusted by browsers
- Lab traffic volume is lower than real-world environments
- Advanced evasion techniques were not fully explored

11. Future Enhancements

- Integrate ModSecurity + OWASP CRS
- Centralize logs into a SIEM
- Implement anomaly-based detection
- Use a trusted CA for TLS
- Add rate limiting and behavioral analysis

12. Conclusion

This lab successfully demonstrates how a Web Application Firewall can be deployed as a reverse proxy to protect vulnerable web applications. By combining TLS termination, backend isolation, and traffic inspection, the lab reflects real-world defensive architectures and provides hands-on experience in modern web security design.

Tools

- VMware workstation.
- Kali Linux
- Ubuntu Linux
- SafeLine Web Application Firewall
- Damn Vulnerable Web Application (DVWA)

References

- https://youtu.be/N0dEC1nuWCQ
- DVWA GitHub Repository: https://github.com/digininja/DVWA