



From Attack to Defense: Building a Cybersecurity Home Lab

**A Hands-On Journey Through Offensive and Defensive
Security**

Project Overview

Objective:

To demonstrate the end-to-end process of building a cybersecurity home lab, from setting up vulnerable services to implementing and testing defensive security controls.

Key Sections:

Lab Architecture: Overview of the attacker ([Kali](#)) and target ([Ubuntu](#)) machines.

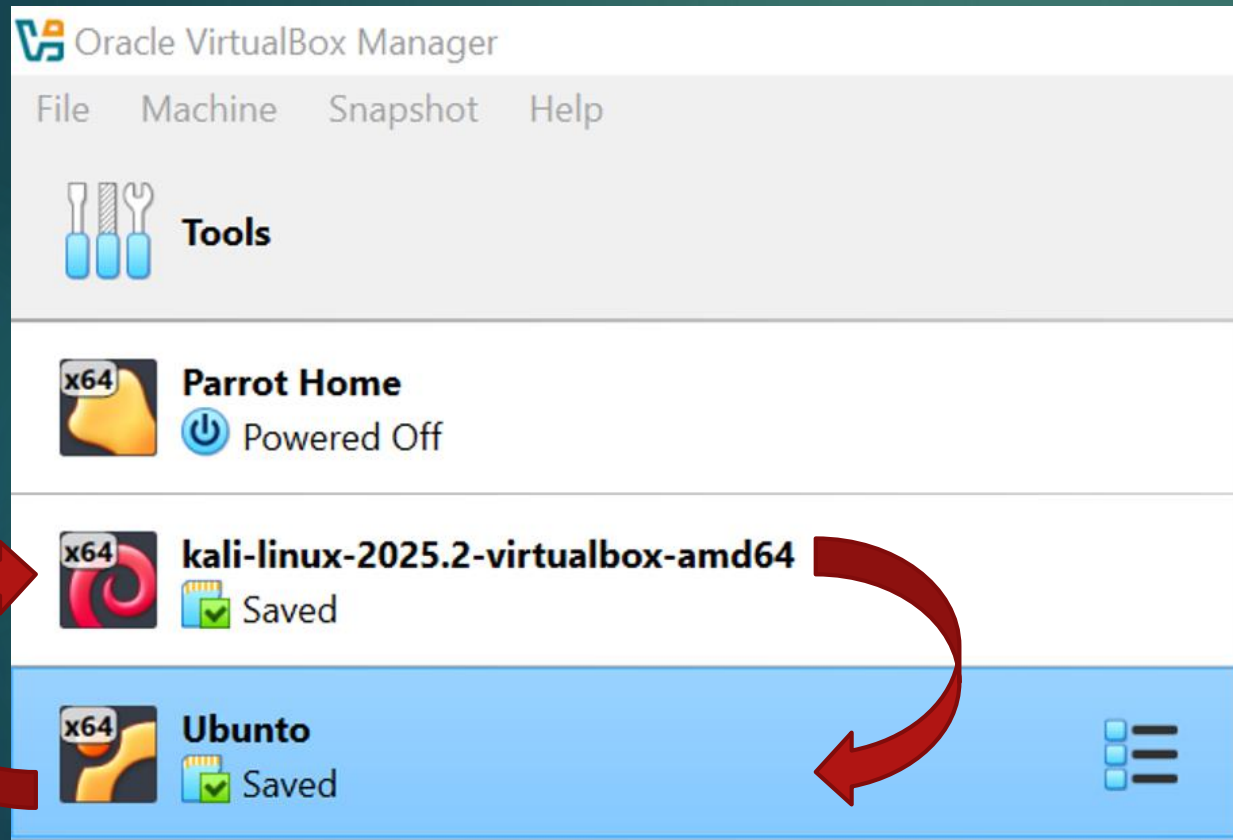
The Attack: Simulating a SQL injection attack on a vulnerable web app.

The Defense: Implementing a Web Application Firewall ([WAF](#)) for protection.

Validation: Testing and confirming the WAF successfully blocks attacks.

Conclusion: Key learnings and future enhancements.

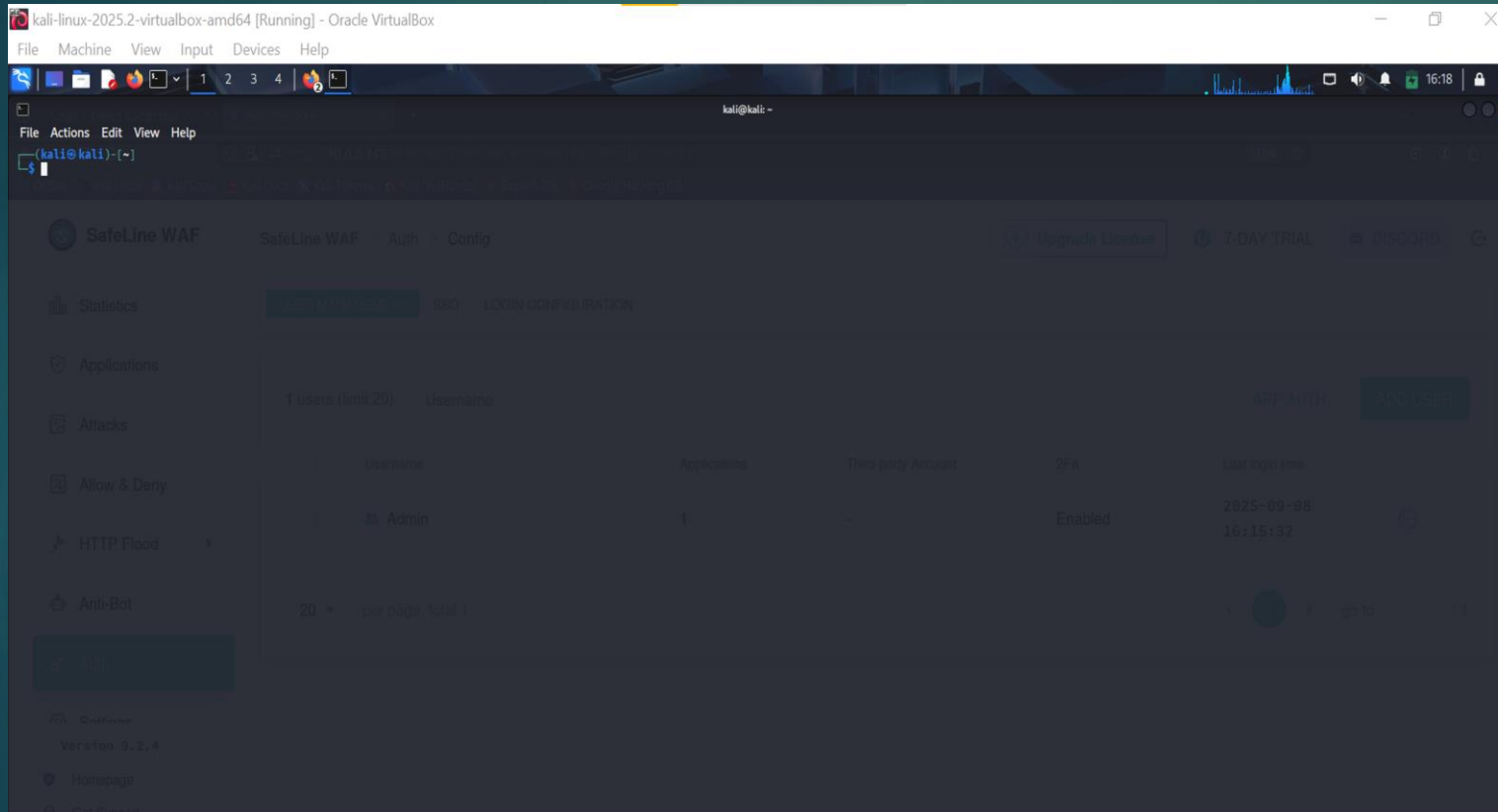
Lab Roles: A Controlled Battlefield



KALI LINUX FUNCTIONED AS THE ATTACKER, UTILIZING ITS OFFENSIVE TOOLKIT TO FIND WEAKNESS.

THE UBUNTU SERVER ACTED AS THE TARGET, HOSTING VULNERABLE SERVICES LIKE DVWA AND THE WAF FOR DEFENSE PRACTICE.

Kali Linux: The Attack Platform



Role: Offensive Attacker

OS: Pre-built Kali Linux ISO

Key Tools:

Web Browser (for manual SQL injection)

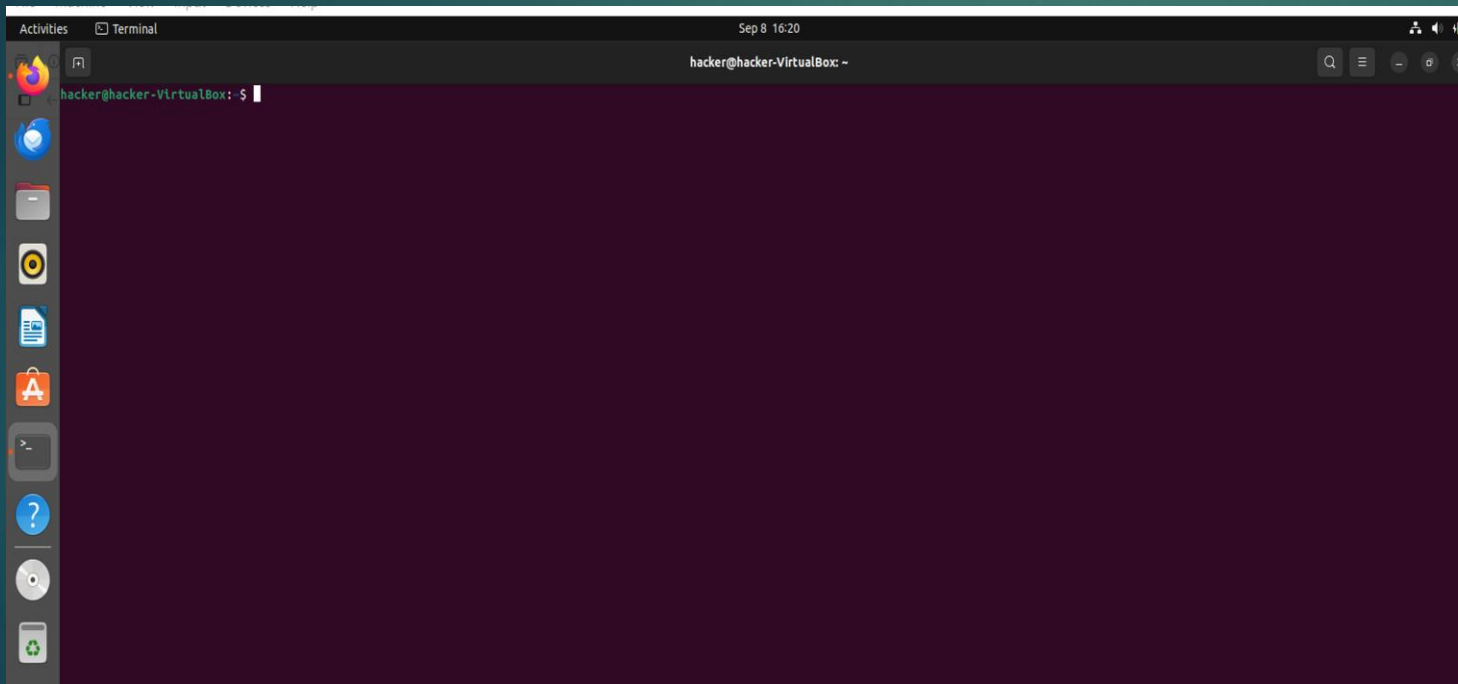
Built-in command line tools

Configuration:

Bridged Networking

Local hosts file entry for dvwa.local

Ubuntu Server: The Target Platform



Role: The Target / Defense

Installed:

LAMP Stack (Apache,
MySQL, PHP)

DVWA (Vulnerable Web App)

SafeLine WAF (Web
Application Firewall)

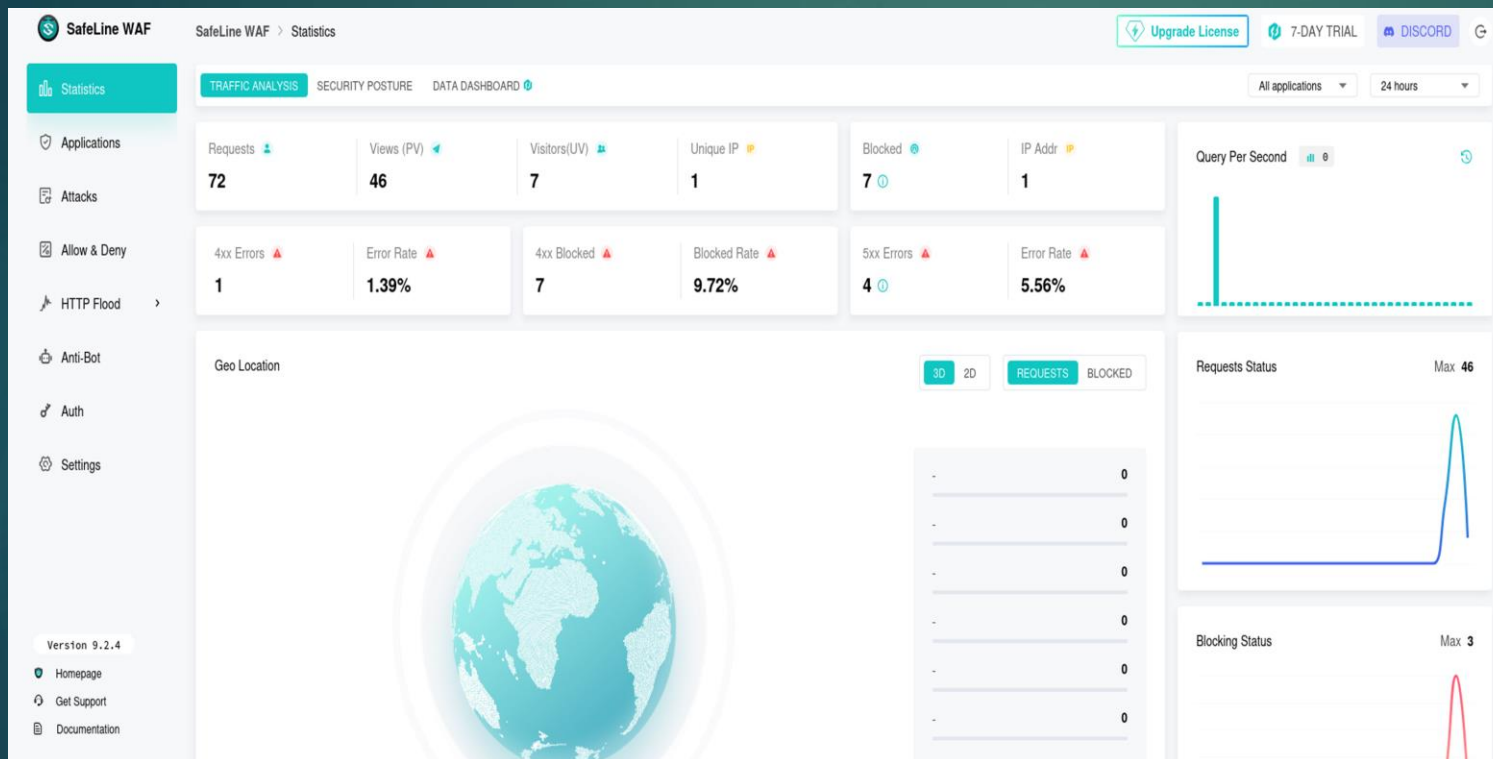
Key Config:

Apache on Port 8080

Local DNS (dvwa.local)

WAF protecting DVWA

Defensive Layer: Web Application Firewall (WAF)

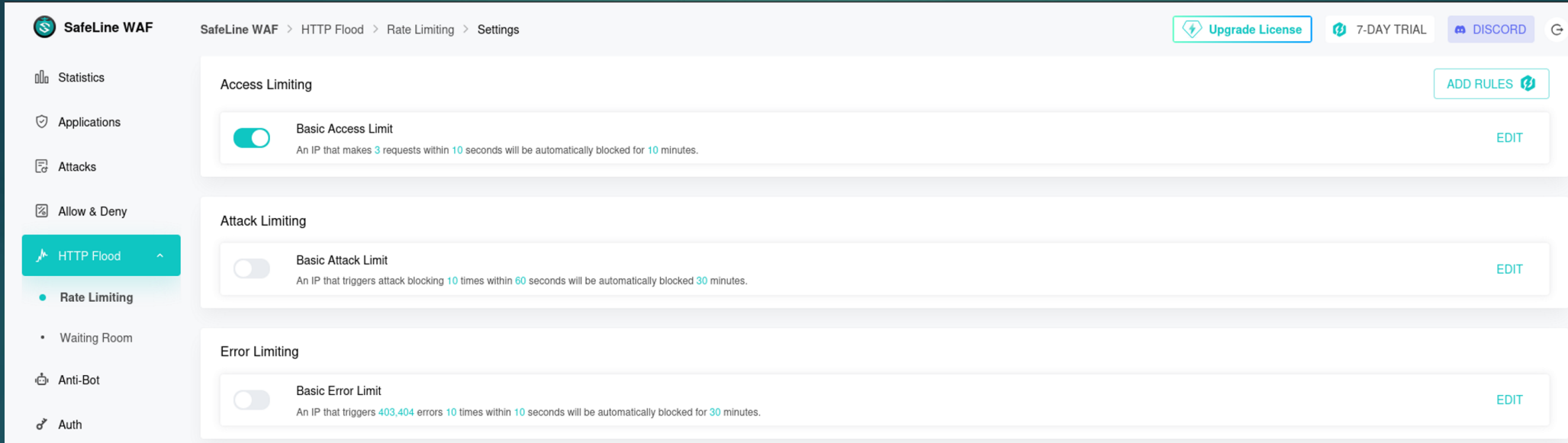


Tool: SafeLine WAF

Purpose: To monitor traffic and block malicious requests before they reach the web server.

Outcome: Successfully mitigated SQL injection attacks, as confirmed by dashboard logs.

Configuring HTTP Flood Rules in the WAF



The screenshot displays the SafeLine WAF configuration interface. The top navigation bar includes the SafeLine WAF logo, a breadcrumb trail (SafeLine WAF > HTTP Flood > Rate Limiting > Settings), and links for Upgrade License, 7-DAY TRIAL, and DISCORD. The left sidebar contains a menu with options: Statistics, Applications, Attacks, Allow & Deny, HTTP Flood (selected), Rate Limiting, Waiting Room, Anti-Bot, and Auth. The main content area is titled 'Settings' and contains three sections: Access Limiting, Attack Limiting, and Error Limiting. Each section has a toggle switch and an 'EDIT' button. The 'Basic Access Limit' toggle is turned on, with a description: 'An IP that makes 3 requests within 10 seconds will be automatically blocked for 10 minutes.' The 'Basic Attack Limit' toggle is turned off, with a description: 'An IP that triggers attack blocking 10 times within 60 seconds will be automatically blocked 30 minutes.' The 'Basic Error Limit' toggle is turned off, with a description: 'An IP that triggers 403,404 errors 10 times within 10 seconds will be automatically blocked for 30 minutes.'

SafeLine WAF

SafeLine WAF > HTTP Flood > Rate Limiting > Settings

Upgrade License 7-DAY TRIAL DISCORD

Statistics Applications Attacks Allow & Deny HTTP Flood Rate Limiting Waiting Room Anti-Bot Auth

Access Limiting

ADD RULES

Basic Access Limit

An IP that makes 3 requests within 10 seconds will be automatically blocked for 10 minutes.

EDIT

Attack Limiting

Basic Attack Limit

An IP that triggers attack blocking 10 times within 60 seconds will be automatically blocked 30 minutes.

EDIT

Error Limiting

Basic Error Limit

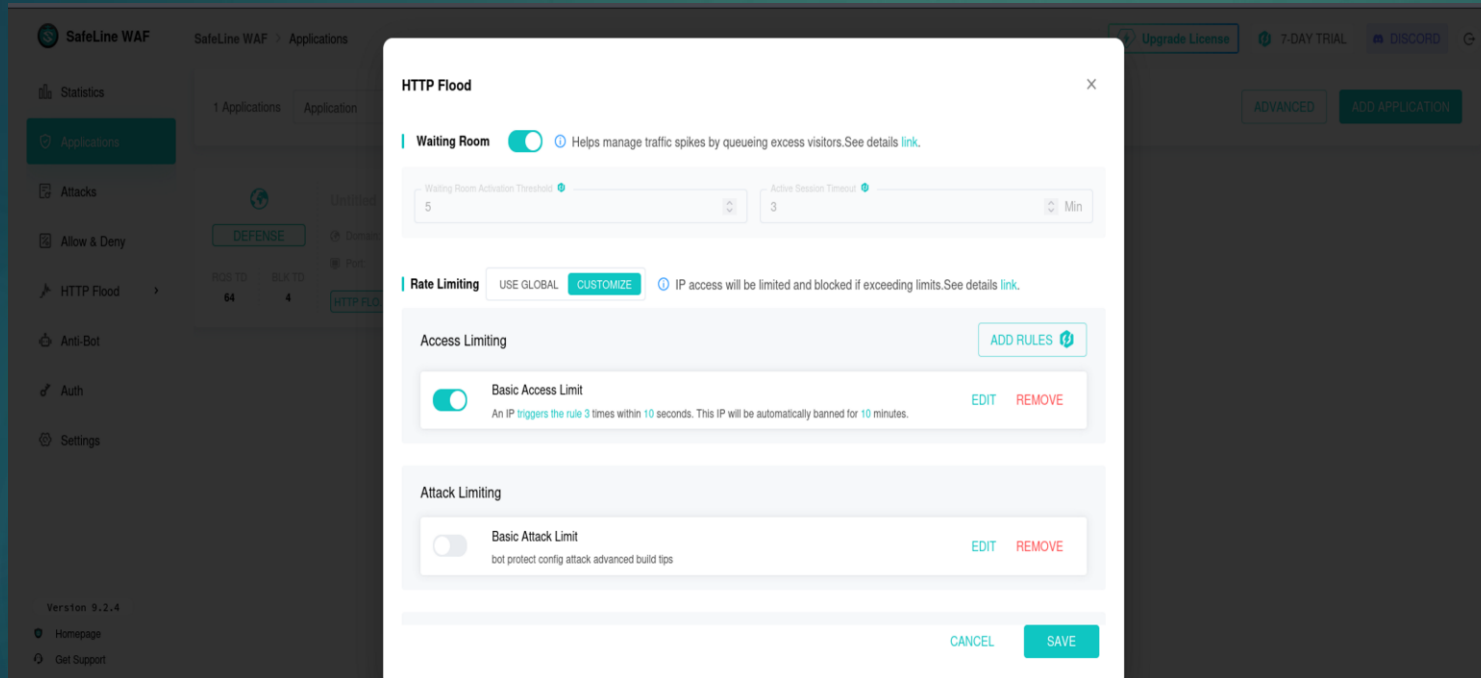
An IP that triggers 403,404 errors 10 times within 10 seconds will be automatically blocked for 30 minutes.

EDIT

To mitigate DDoS risk:

- i. Navigated to HTTP Flood settings.
- ii. Enabled the protection toggle.
- iii. Configured access limit thresholds.

Implementing HTTP Flood Rules in the WAF




Action Taken:

Created a new rule: "Basic Access Limit"

Set Threshold: Block an IP address that exceeds 3 requests within a 10-second duration.

Set Action: Block the offending IP address for 10 minutes.

Security Rule Stress Test



The screenshot shows the DVWA homepage. The left sidebar contains a menu with the following items: Home (highlighted with a red arrow), Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), and XSS (Stored). The main content area has a header with the DVWA logo and a welcome message. Below the welcome message, there are sections for 'General Instructions' and a 'WARNING!' section.

Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficulty**, with a simple straightforward interface.

General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerabilities** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

WARNING!

Stress-testing the rule by rapidly requesting the home page to verify the WAF enforces the access limit.

HTTP Flood Mitigation: Successful



Access Forbidden

Blocked for Access Too Fast



Security Detection Powered By SafeLine WAF



Analyzing the HTTP Flood Attack in WAF Logs

HTTP Flood > Rate Limiting

Upgrade License 7-DAY TRIAL DISCORD

Limiting

the rate of requests from a single source, by defends against DDoS attacks, crawlers and bots.

SHOW THIS AGAIN

* No rate limit * With rate limit

Application	Detail	Blocked	Start At	
Untitled dvwa.local	Reason: 3 Reqs within 10 seconds, Basic Access Limit was triggered Action: Block 10 minutes	1	2025-09-06 18:05:59	Unblock
Untitled dvwa.local	Reason: 3 Reqs within 10 seconds, Basic Access Limit was triggered Action: Block 10 minutes	3	2025-09-06 17:28:35	Unblocked
Untitled dvwa.local	Reason: 3 Reqs within 10 seconds, Basic Access Limit was triggered Action: Block 1 minutes	0	2025-09-06 17:21:48	Unblocked

The lab successfully demonstrated a full cyber kill chain:

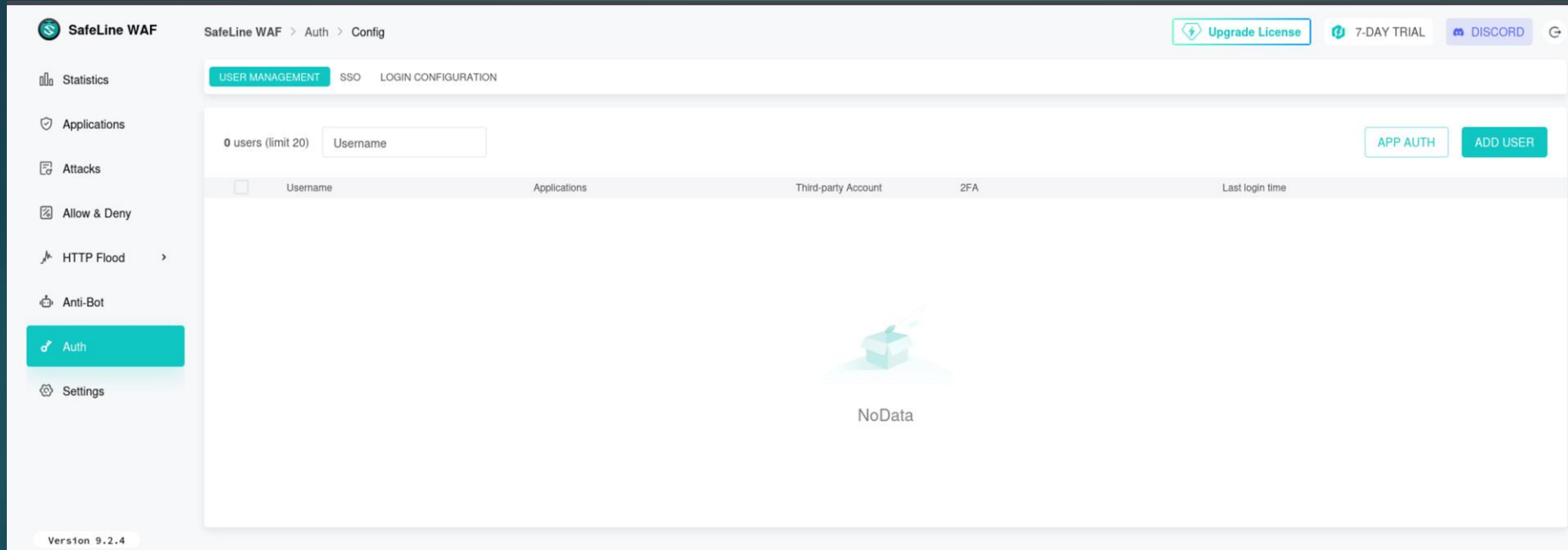
Attack: Simulated HTTP Flood from Kali.

Defense: WAF detected the anomaly.

Result: Attack was automatically mitigated by IP block.

This validates the entire defensive setup.

Adding Users to the WAF Authentication Policy




Action Taken:

- I. Navigated to the Authentication (**Auth**) section in the SafeLine WAF dashboard.
- II. Selected the "**ADD USERS**" management tab.
- III. Created authorized user accounts to define who has access to the protected application.


Adding a New Authorized User

Add user ×

Username *
Admin

Password *
QPnfwQV Random password 

Email

 +1 |

☐ Force Enable TOTP


CANCEL CREATE

Process:


Entered a new username and password.


Finalized by clicking 'Create.'


User Account Created


 **SafeLine WAF**


SafeLine WAF > Auth > Config


[Upgrade License](#) [7-DAY TRIAL](#) [DISCORD](#) 


 Statistics


 Applications


 Attacks

 Allow & Deny

 HTTP Flood >

 Anti-Bot



 Auth

 Settings

USER MANAGEMENT SSO LOGIN CONFIGURATION

1 users (limit 20)

[APP AUTH](#) [ADD USER](#)

<input type="checkbox"/>	Username	Applications	Third-party Account	2FA	Last login time	
<input type="checkbox"/>	 Admin	0	-	Close	-	

20 per page, total 1

< 1 > go to / 1

Access Control: Before and After WAF Authentication



Username

Password

Login

[Damn Vulnerable Web Application \(DVWA\)](#)

Sign In - Password

Sign In



Security Detection Powered By SafeLine WAF

Reviewing Authentication Logs

The screenshot shows the SafeLine WAF dashboard with the 'Auth' section selected. A sidebar on the left contains navigation links: Statistics, Applications, Attacks, Allow & Deny, HTTP Flood, Anti-Bot, Auth (highlighted), and Settings. The main content area has a header 'SafeLine WAF > Auth' and a message: 'Users need to authenticate their identities when accessing applications with this feature. Unauthorized users will be denied.' Below this is a 'DONT SHOW THIS AGAIN' link and a '* No Auth' note. A filter bar includes 'Account Name', 'Auth Results' (dropdown), 'Source IP', 'Application', and a 'MORE' button. The table below has columns: Account Name, Application, Login Method, and Auth Results. It lists four login attempts for 'dvwa.local': one successful login for 'Admin' and three failed attempts for 'Admin' and 'admin'. The footer shows 'Version 9.2.4' and links for 'Homepage', 'Get Support', and 'Documentation'.

Account Name	Application	Login Method	Auth Results
Admin	Unlabeled * dvwa.local	Account password	SUCCESS
Admin	Unlabeled * dvwa.local	Account password	FAIL
admin	Unlabeled * dvwa.local	Account password	FAIL
admin	Unlabeled * dvwa.local	Account password	FAIL

Key Activities:

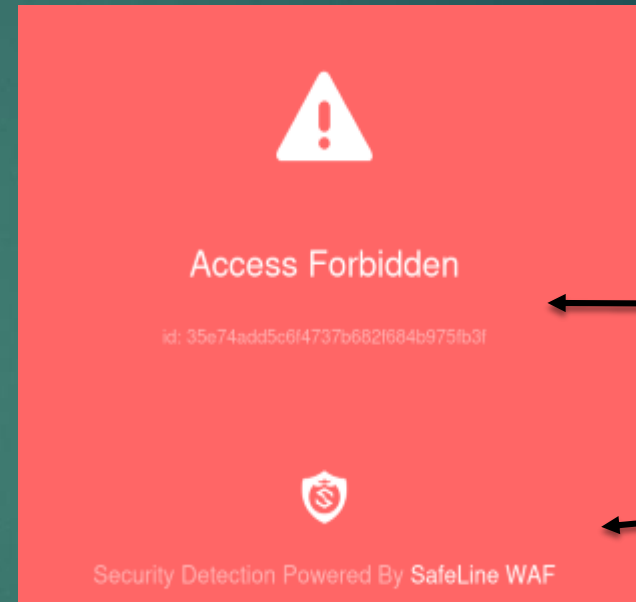
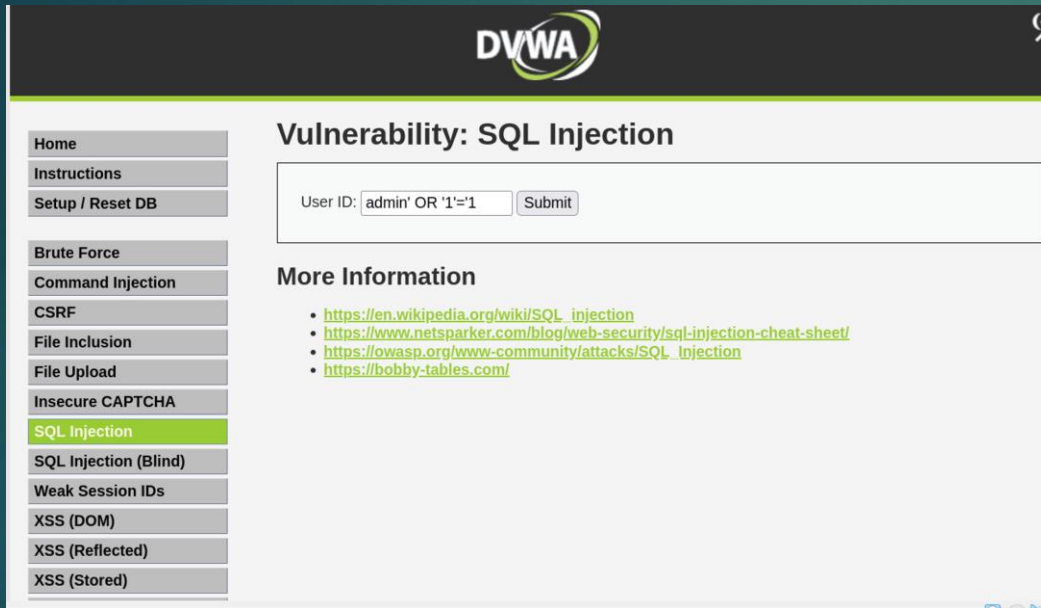
Located the authentication section in the WAF dashboard.

Filtered logs to view login events and access attempts.

Identified successful user logins and any failed brute-force attempts.

Confirmed that only authorized users gained access to the application.

SQL Injection Defense in Action

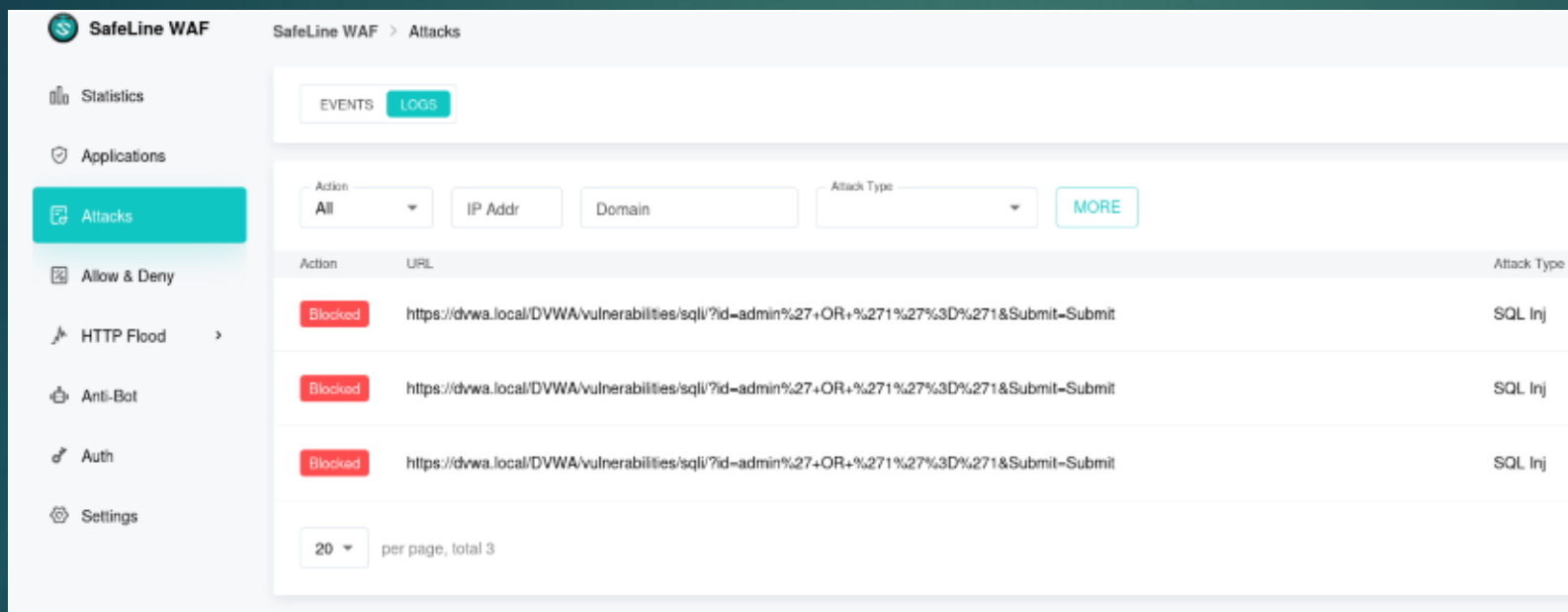


Attack Simulated: A standard SQL injection payload (admin' OR '1'='1) was sent to the login form.

Attack Blocked: The WAF identified the malicious input in real-time.

Request Denied: The attack was stopped before it could reach the web server and database.

Monitoring WAF SQLi Detection Events



The screenshot displays the SafeLine WAF 'Attacks' page. On the left is a sidebar with navigation links: Statistics, Applications, Attacks (highlighted), Allow & Deny, HTTP Flood, Anti-Bot, Auth, and Settings. The main content area has tabs for 'EVENTS' and 'LOGS'. Below these are filters for Action (set to 'All'), IP Addr, Domain, and Attack Type, with a 'MORE' button. A table lists three blocked SQLi attacks. Each row shows a red 'Blocked' status, the full URL, and the attack type 'SQL Inj'. At the bottom, a pagination control shows '20' items per page and a total of 3 items.

Action	URL	Attack Type
Blocked	https://dvwa.local/DVWA/vulnerabilities/sql/?id=admin%27+OR+%271%27%3D%271&Submit=Submit	SQL Inj
Blocked	https://dvwa.local/DVWA/vulnerabilities/sql/?id=admin%27+OR+%271%27%3D%271&Submit=Submit	SQL Inj
Blocked	https://dvwa.local/DVWA/vulnerabilities/sql/?id=admin%27+OR+%271%27%3D%271&Submit=Submit	SQL Inj

Key Findings in the Logs:

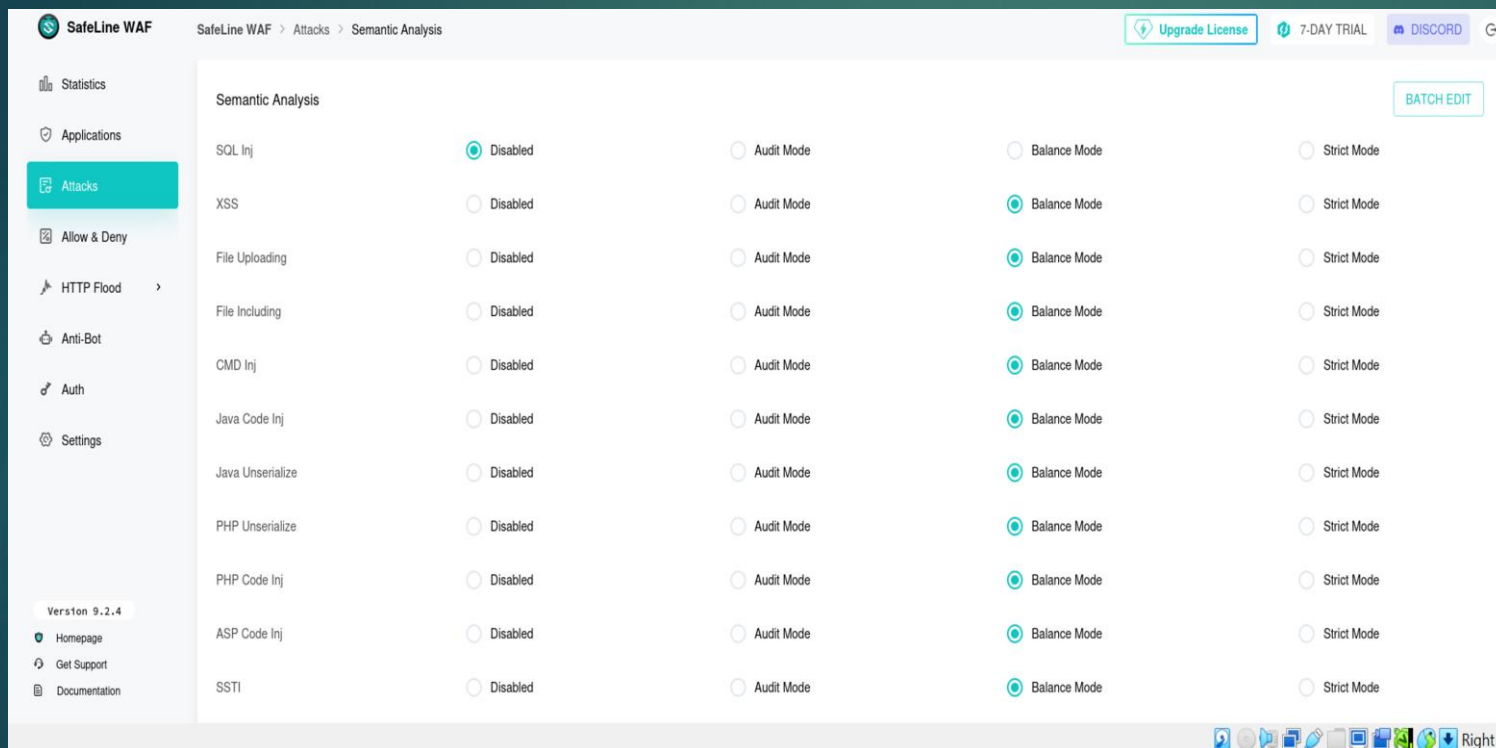
Threat Identified: Logs show repeated SQLi payloads (e.g., admin' OR '1'='1) detected.

Action Taken: The WAF automatically blocked the requests.

Source IP: The logs recorded the attacker's IP address for further analysis.

Timestamp: Each event was logged with date and time for precise auditing.

WAF Configuration – SQL Injection Disabled



SQL Injection protection is turned off.

Other attack vectors (XSS, File Uploading, CMD Injection, etc.) are in Balance Mode.

This creates a high-risk gap in application security.

Action Required: Enable at least Balance Mode for SQL Injection.

Impact of Disabling SQL Injection Protection

DVWA

Vulnerability: SQL Injection

User ID:

ID: admin' OR '1'='1
First name: admin
Surname: admin

ID: admin' OR '1'='1
First name: Gordon
Surname: Brown

ID: admin' OR '1'='1
First name: Hack
Surname: Me

ID: admin' OR '1'='1
First name: Pablo
Surname: Picasso

ID: admin' OR '1'='1
First name: Bob
Surname: Smith

SQL Injection left unprotected allows attackers to bypass login.

Query injection (' OR '1'='1) reveals all records.

Demonstration: DVWA shows exposed user accounts.

Real-world risk: Data breach, privilege escalation, compliance violations.

Solution: Enable SQLi protection in WAF (Balance or Strict Mode).



Thank You

Let's Connect & Explore Further

Lab Documentation:

GitHub: <https://github.com/Eyaan-123>