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.

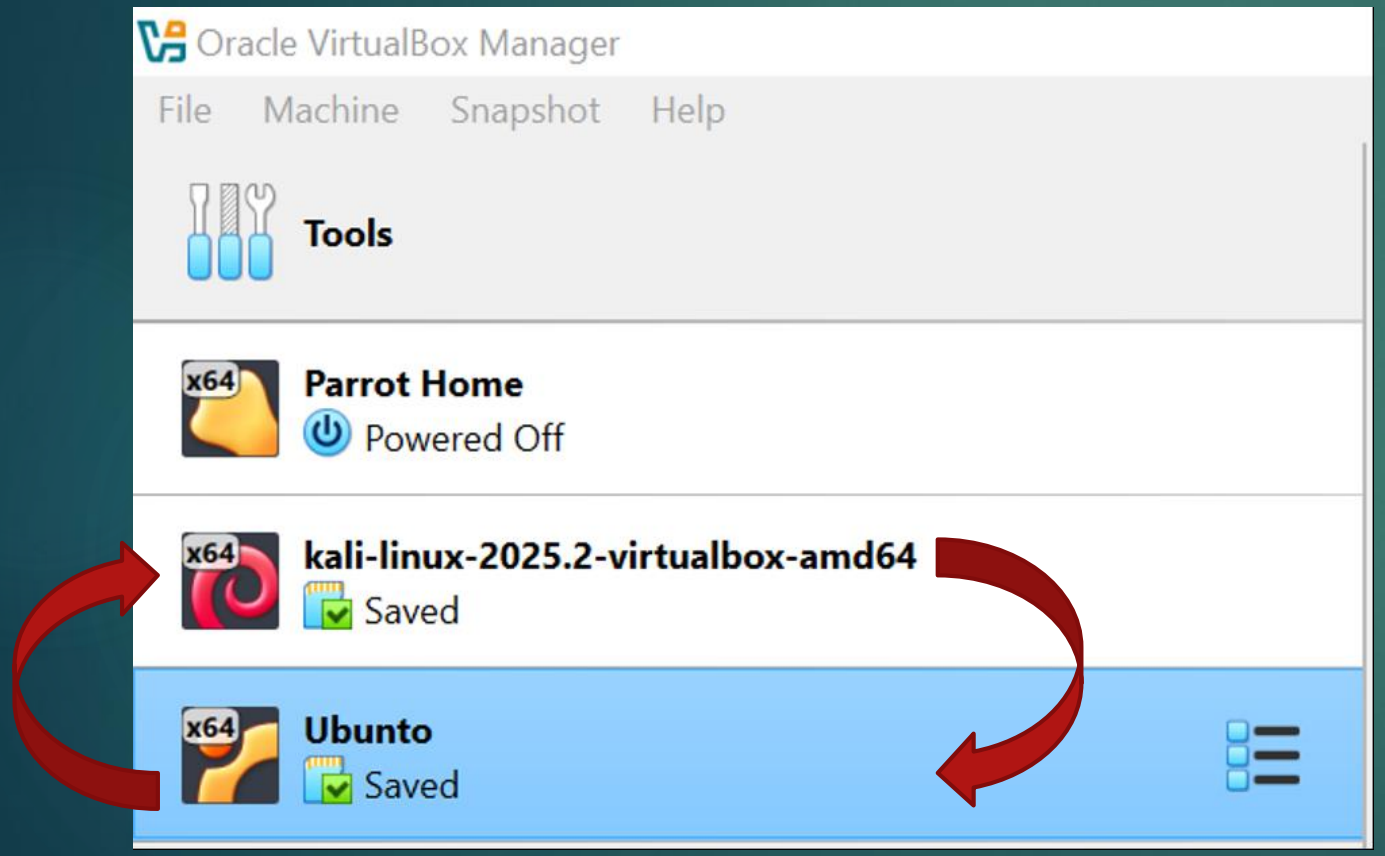
The Attack: Simulating a SQL injection attack on a vulnerable web app.

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.

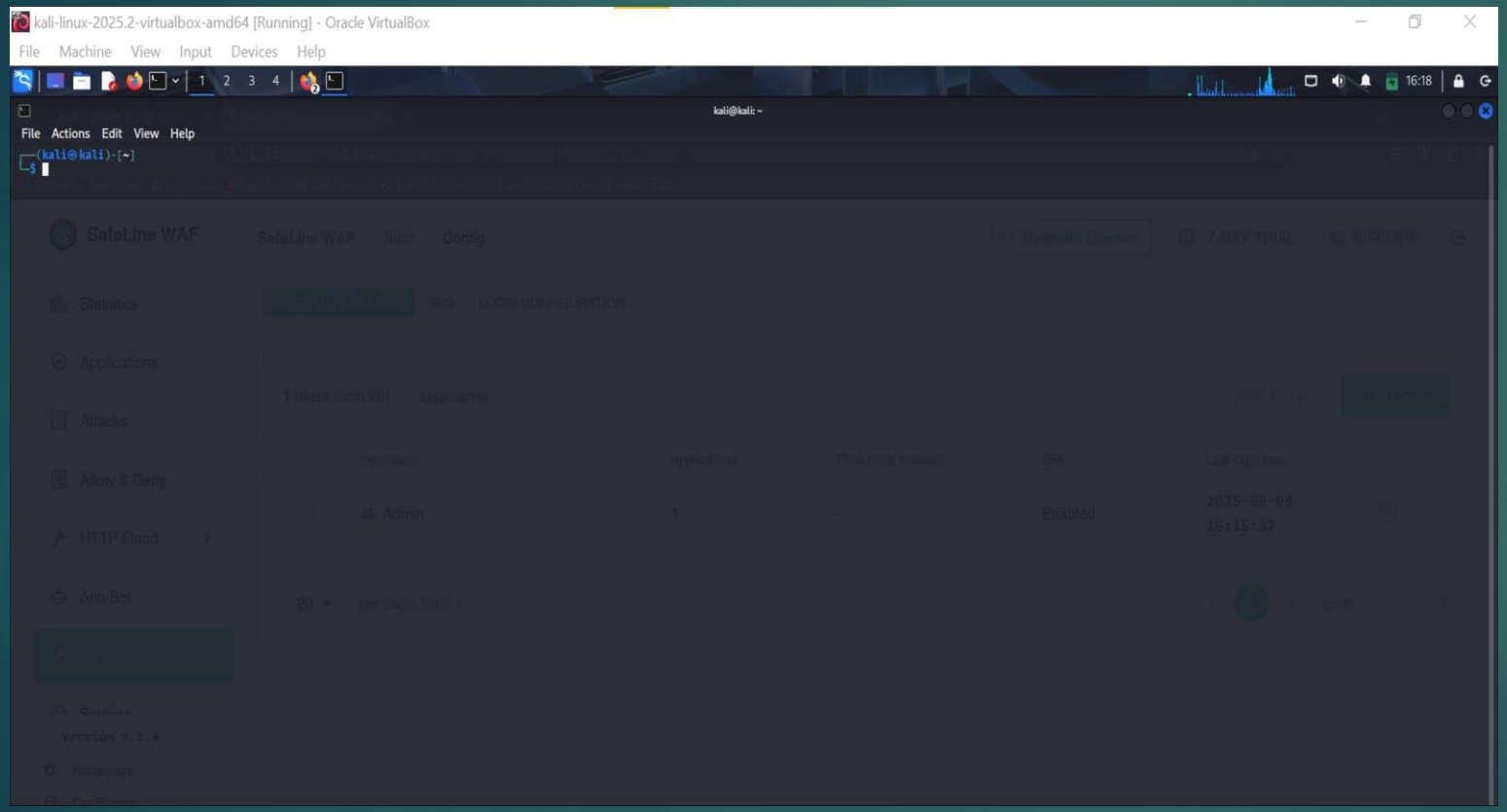
Lab Roles: A Controlled Battlefield



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)

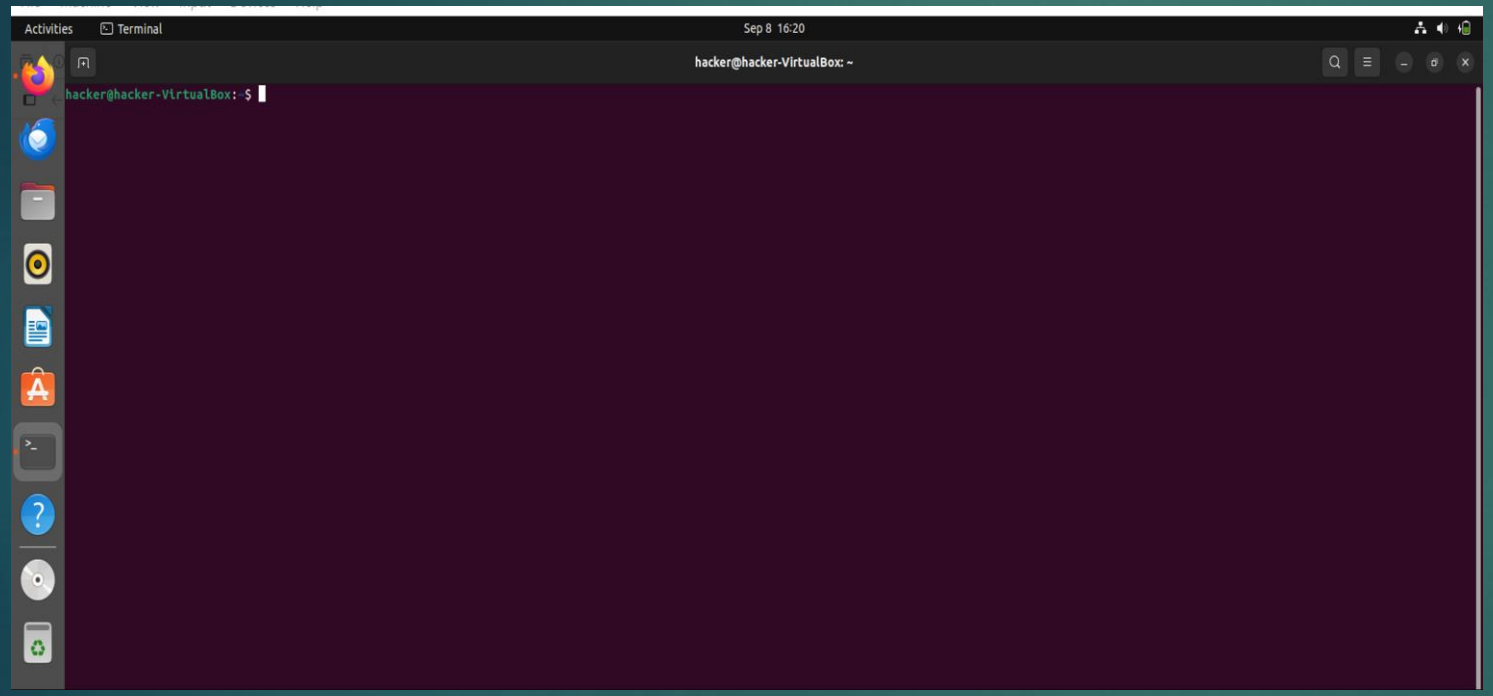
Built-in command line tools

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)

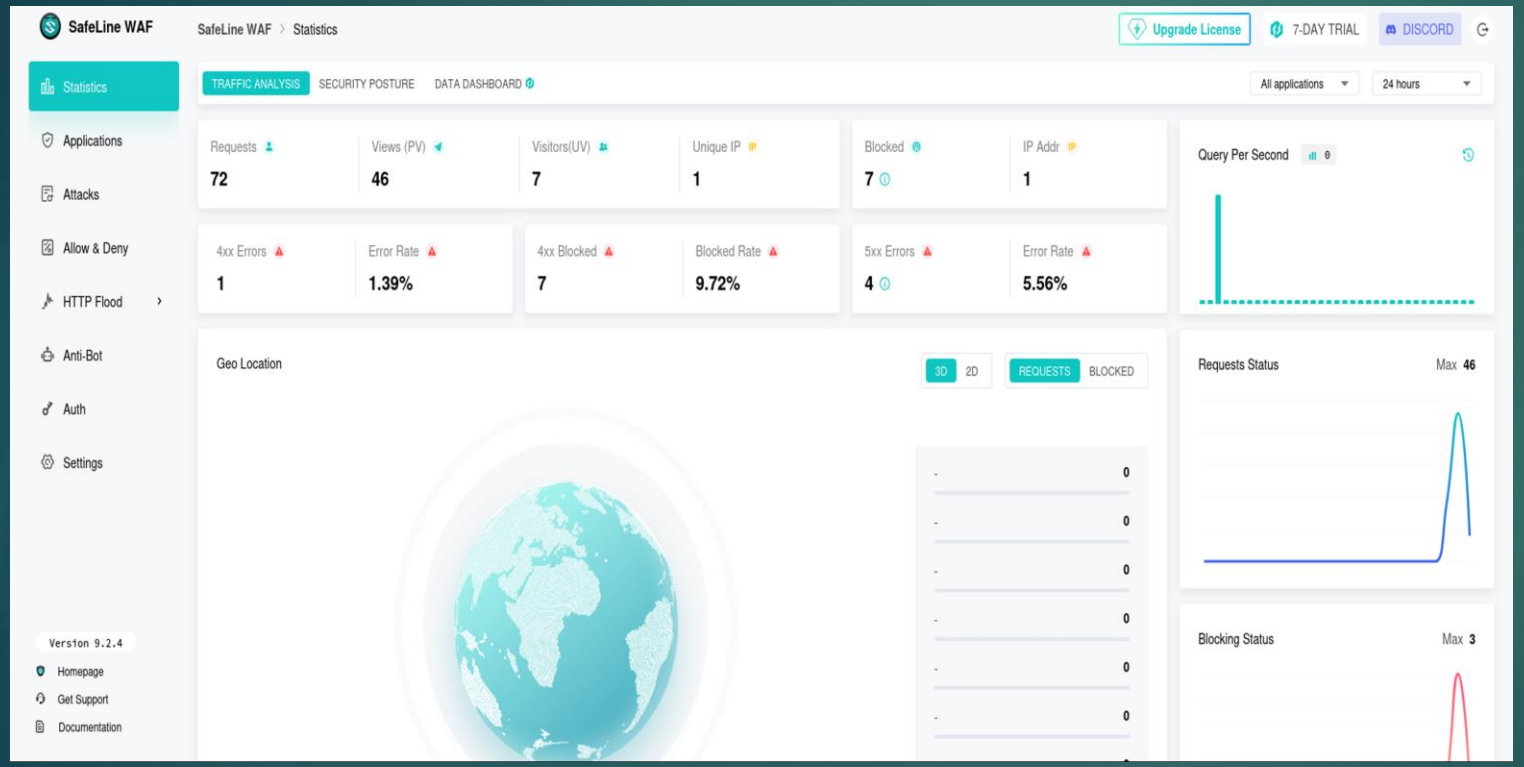
Key Config:

Apache on Port 8080

Local DNS (dvwa.local)

WAF protecting DVWA

Defensive Layer: Web Application Firewall (WAF)

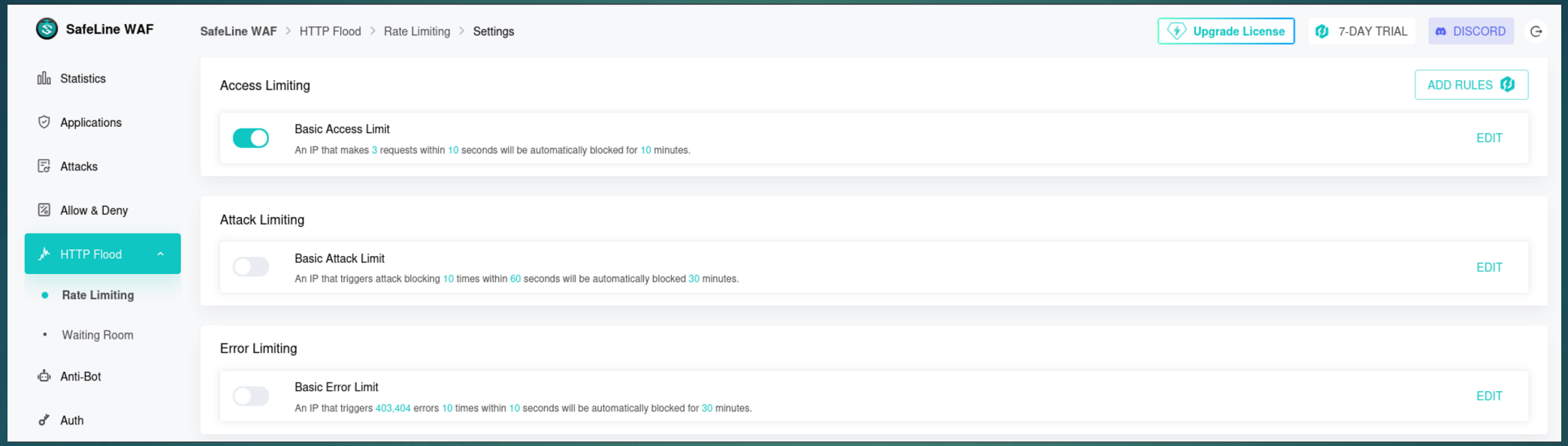


Tool: SafeLine WAF

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



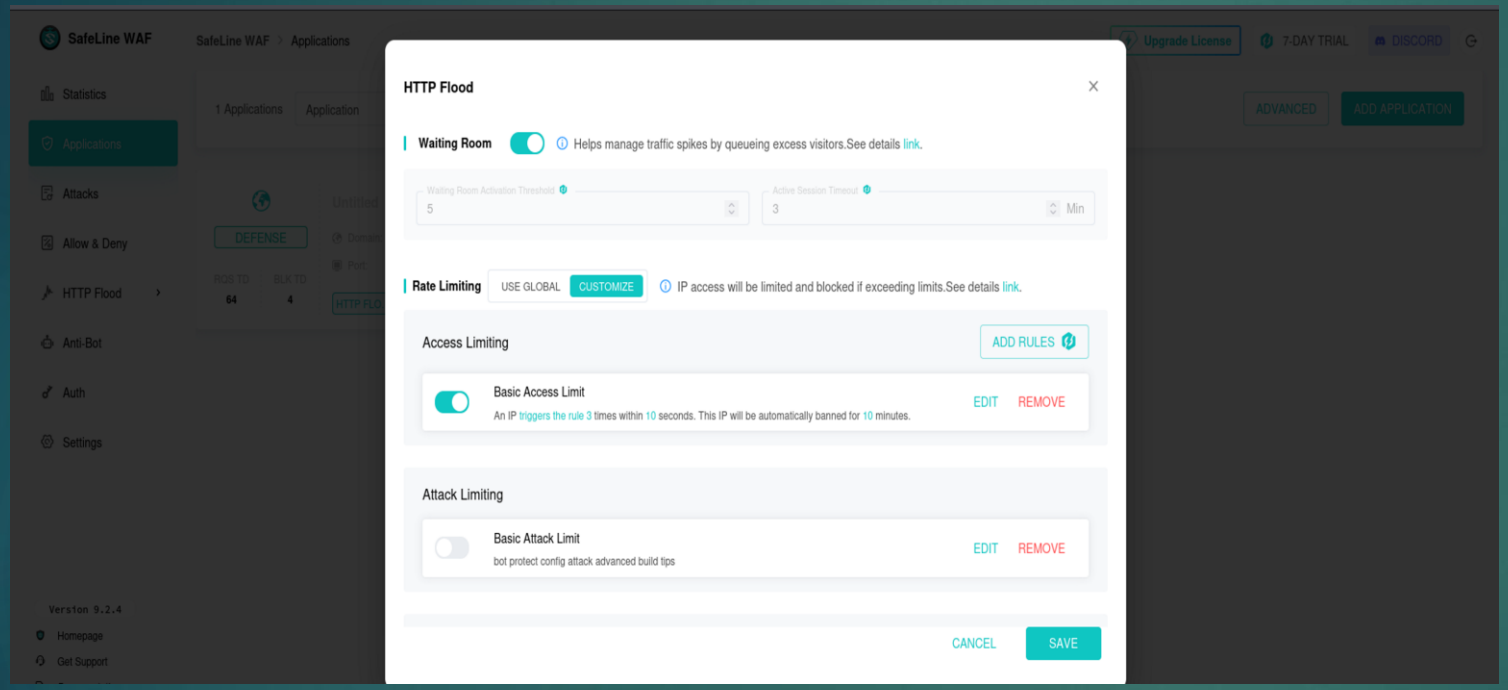
To mitigate DDoS risk:

i. Navigated to HTTP Flood settings.

ii. Enabled the protection toggle.

iii. Configured access limit thresholds.

Implementing HTTP Flood Rules in the WAF



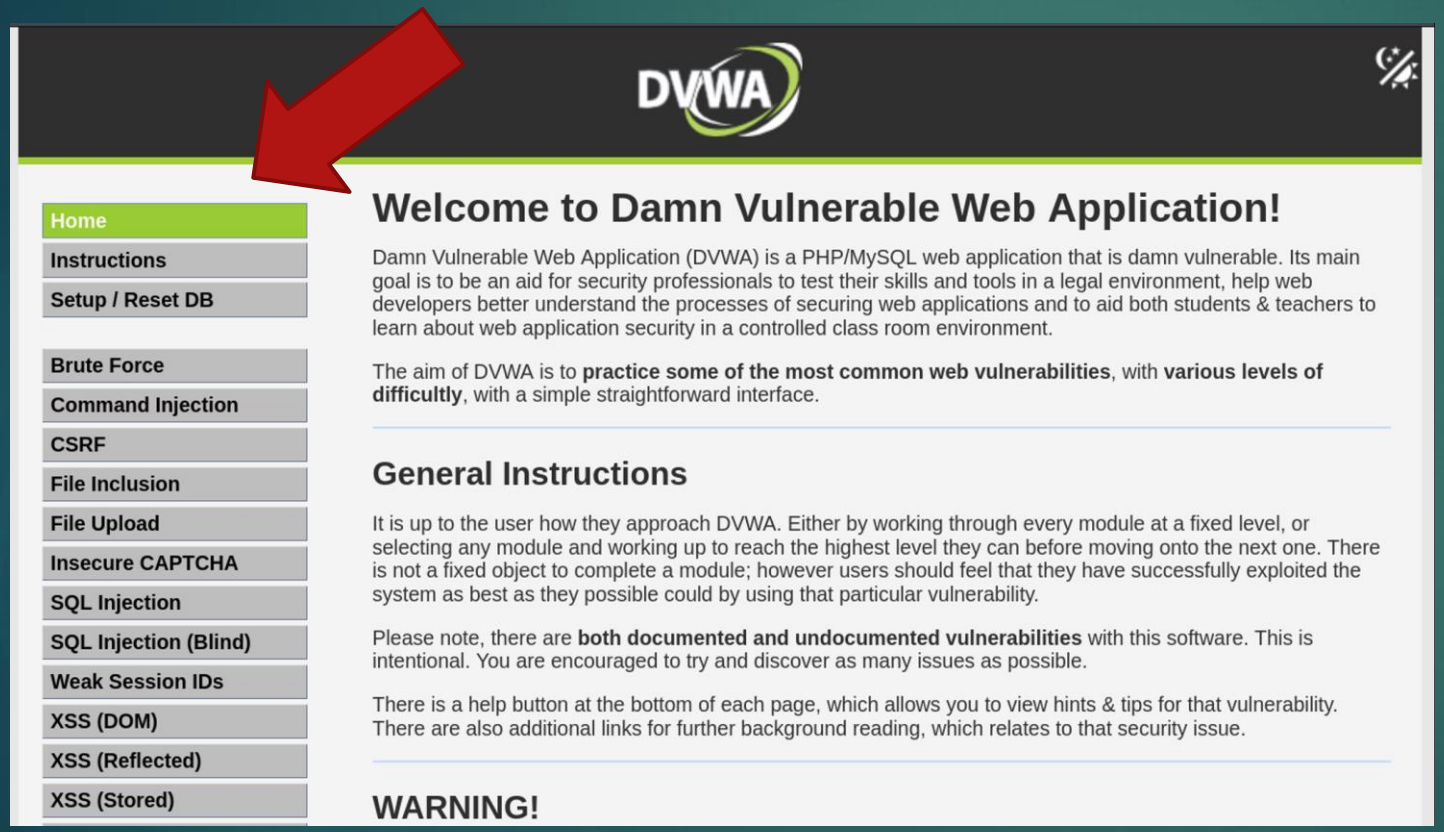
Action Taken:

Created a new rule: "Basic Access Limit"

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

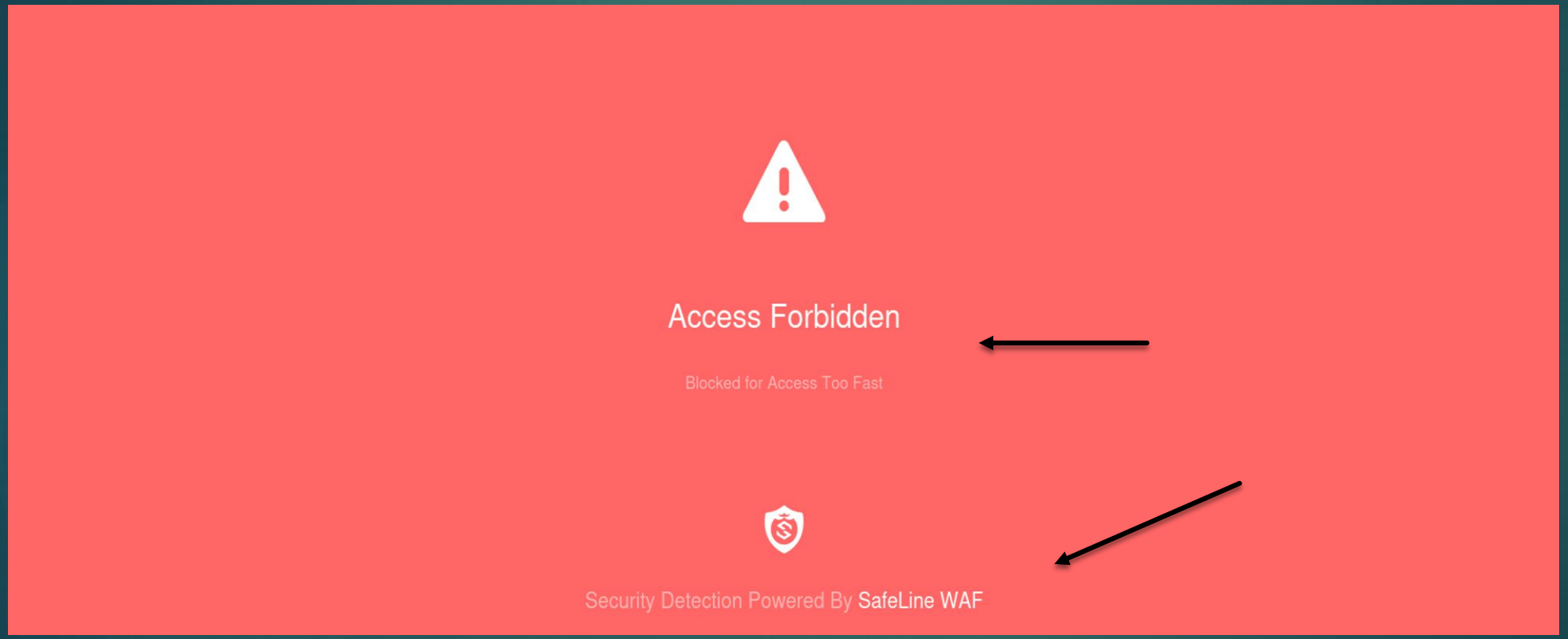
Set Action: Block the offending IP address for 10 minutes.

Security Rule Stress Test

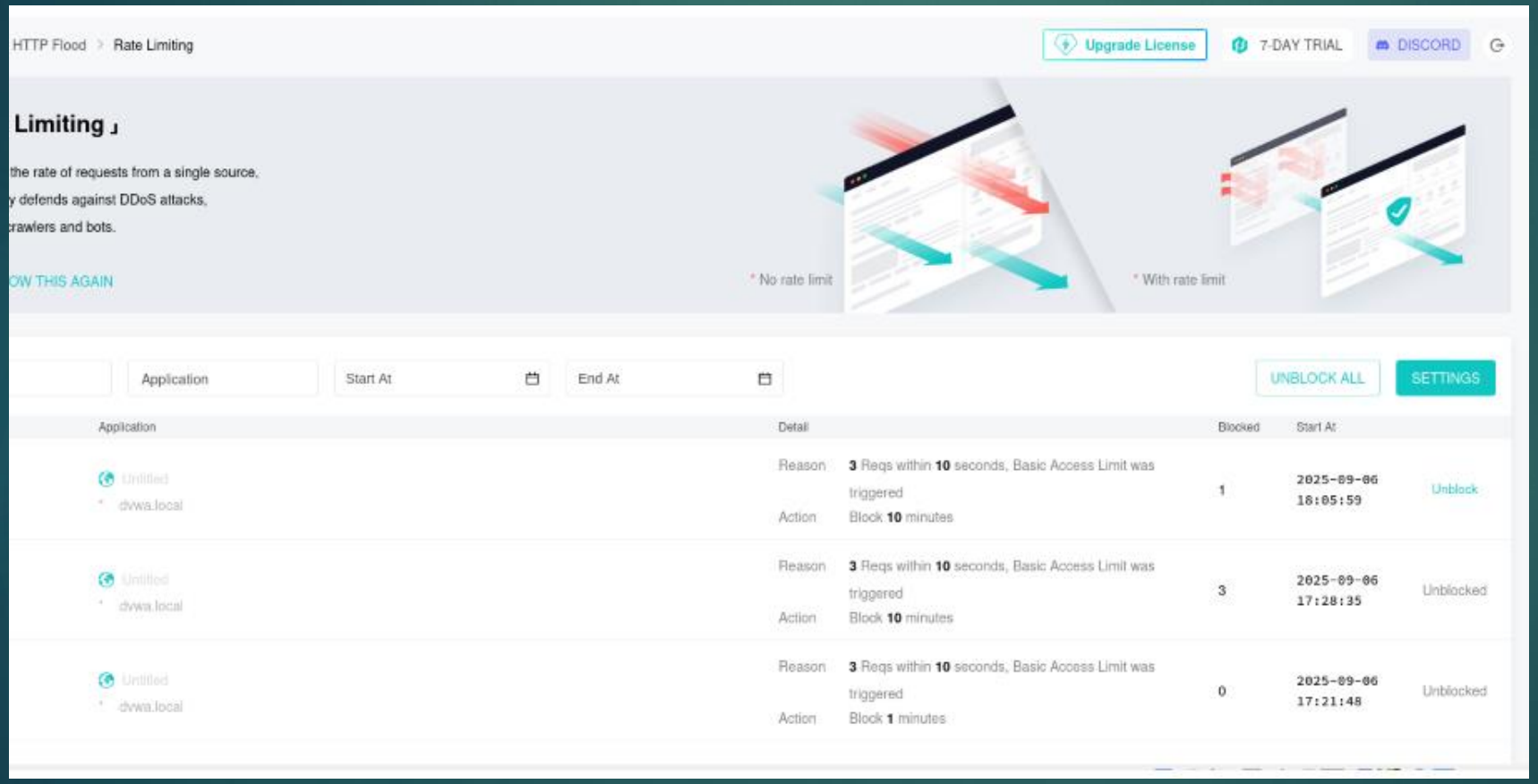


Stress-testing the rule by rapidly requesting the home page to verify the WAF enforces the access limit.

HTTP Flood Mitigation: Successful



Analyzing the HTTP Flood Attack in WAF Logs



The **lab successfully** demonstrated a full cyber kill chain:

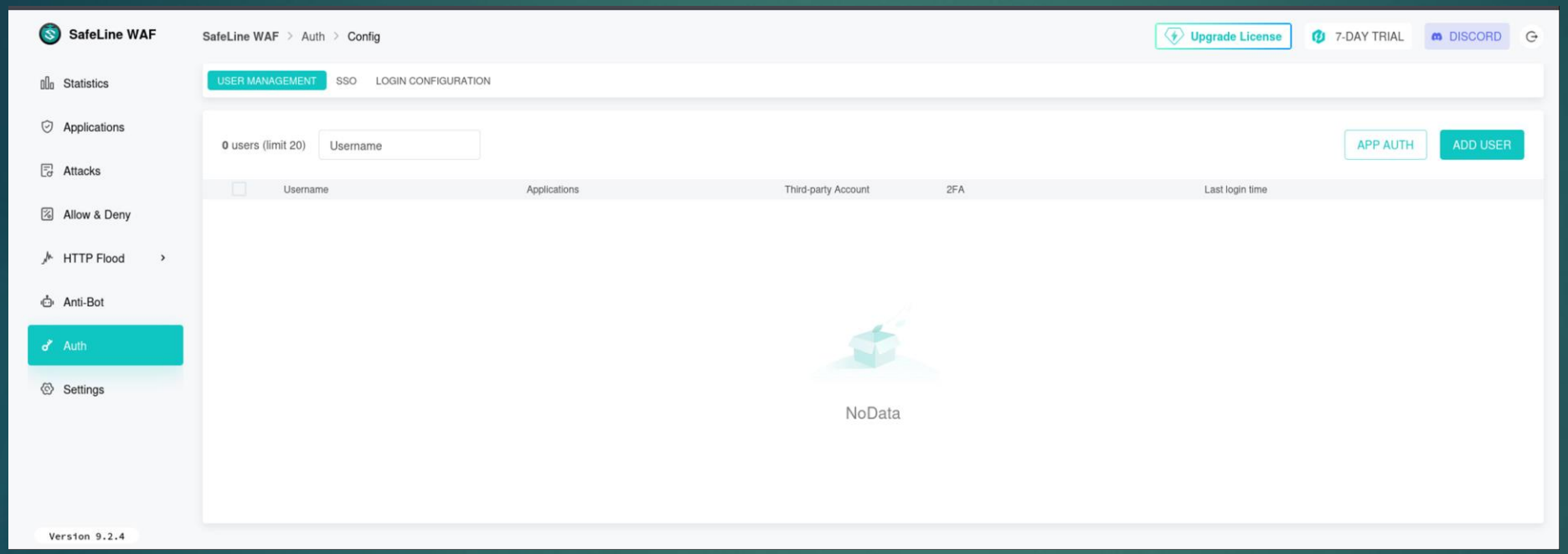
Attack: Simulated HTTP Flood from Kali.

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



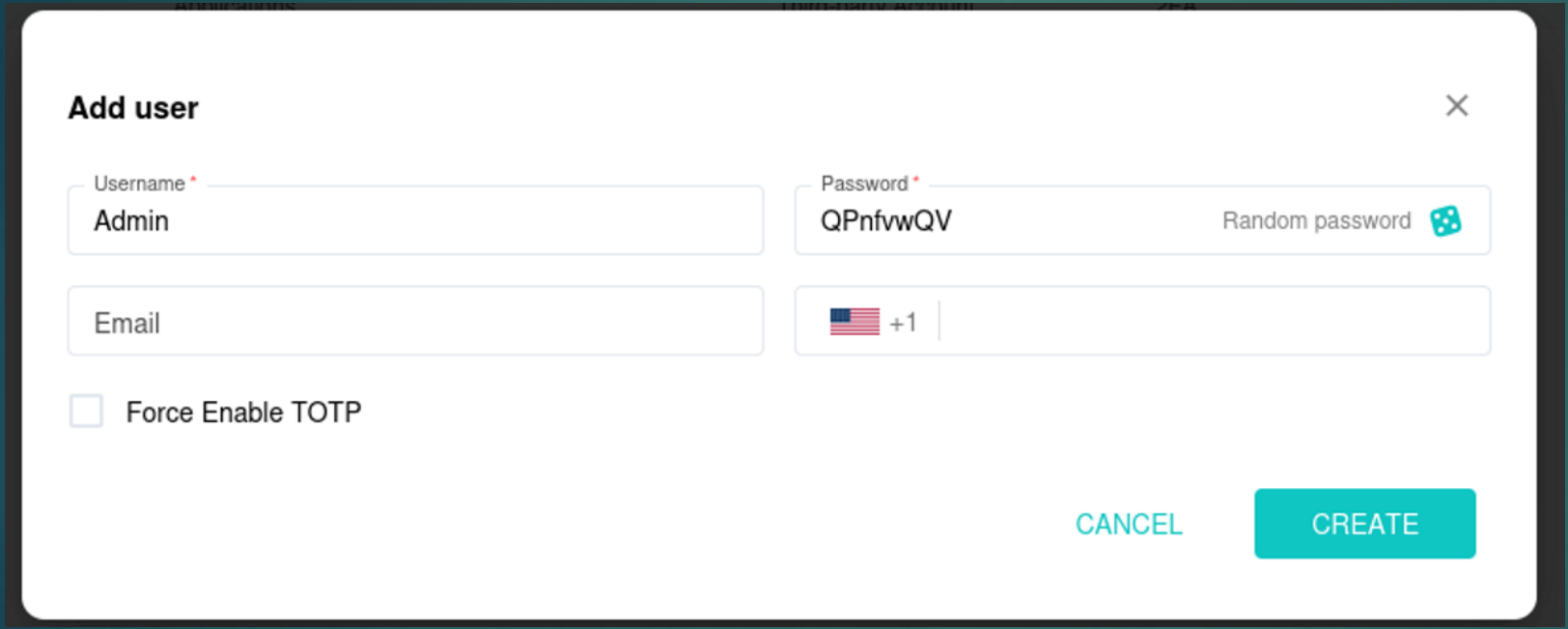
Action Taken:

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

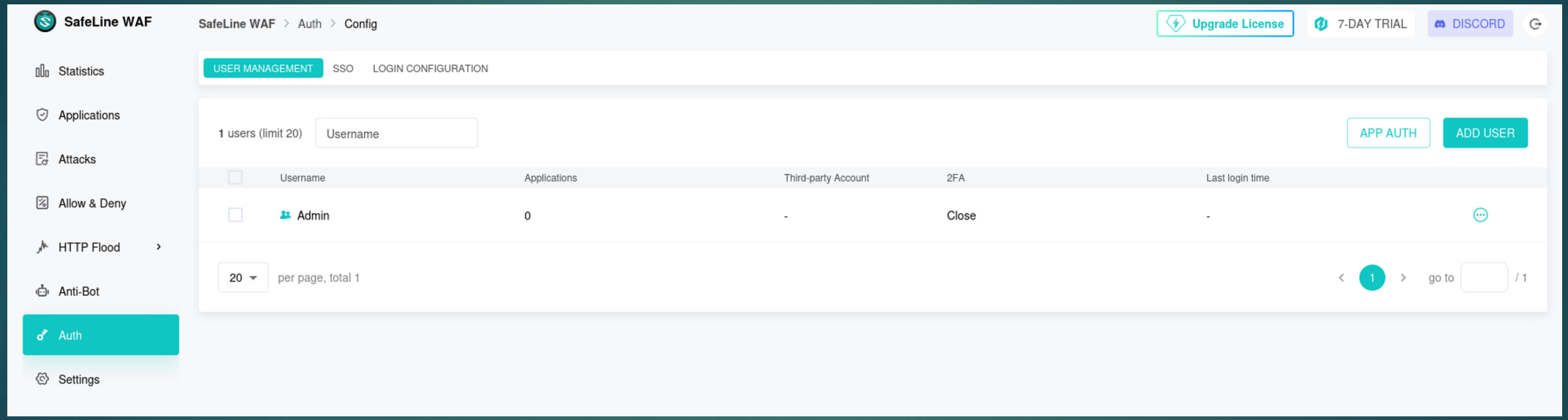


Process:

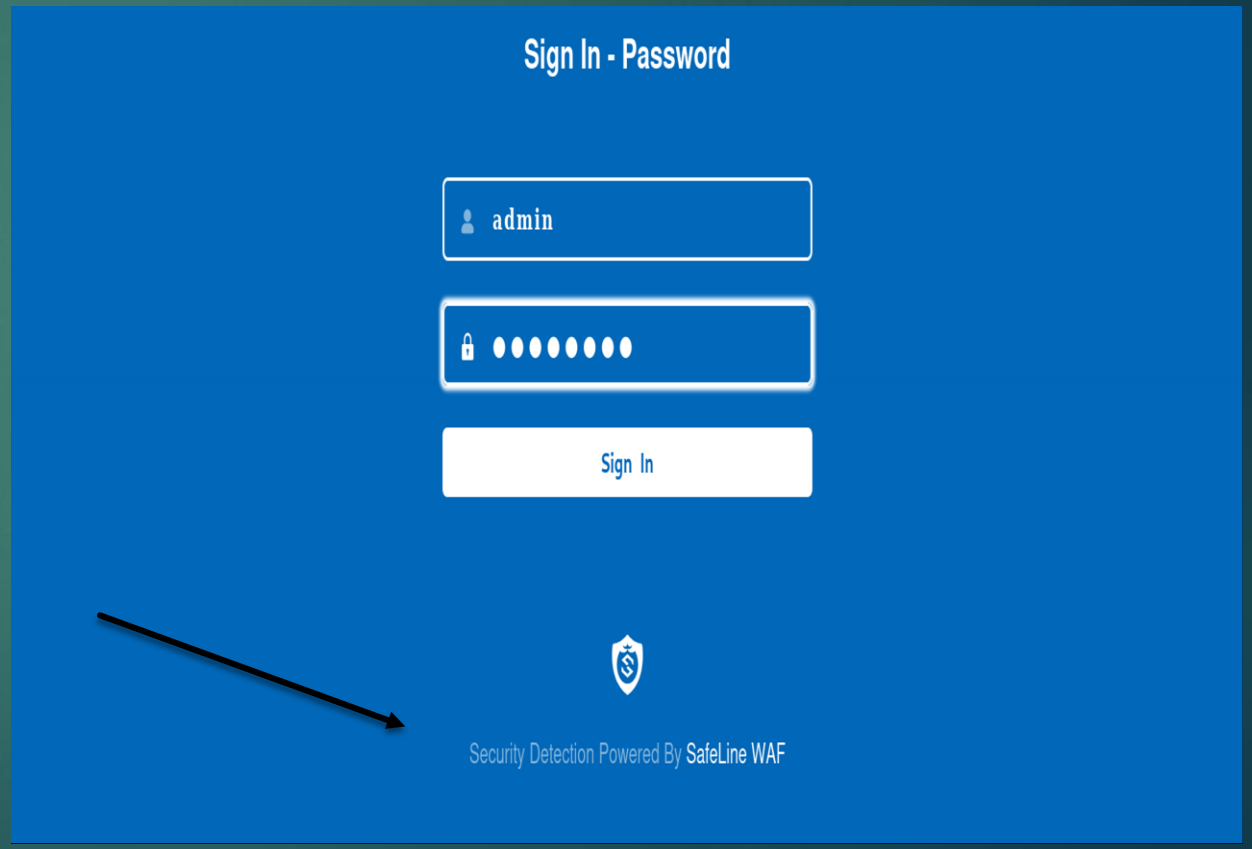
Entered a new username and password.

Finalized by clicking 'Create.'

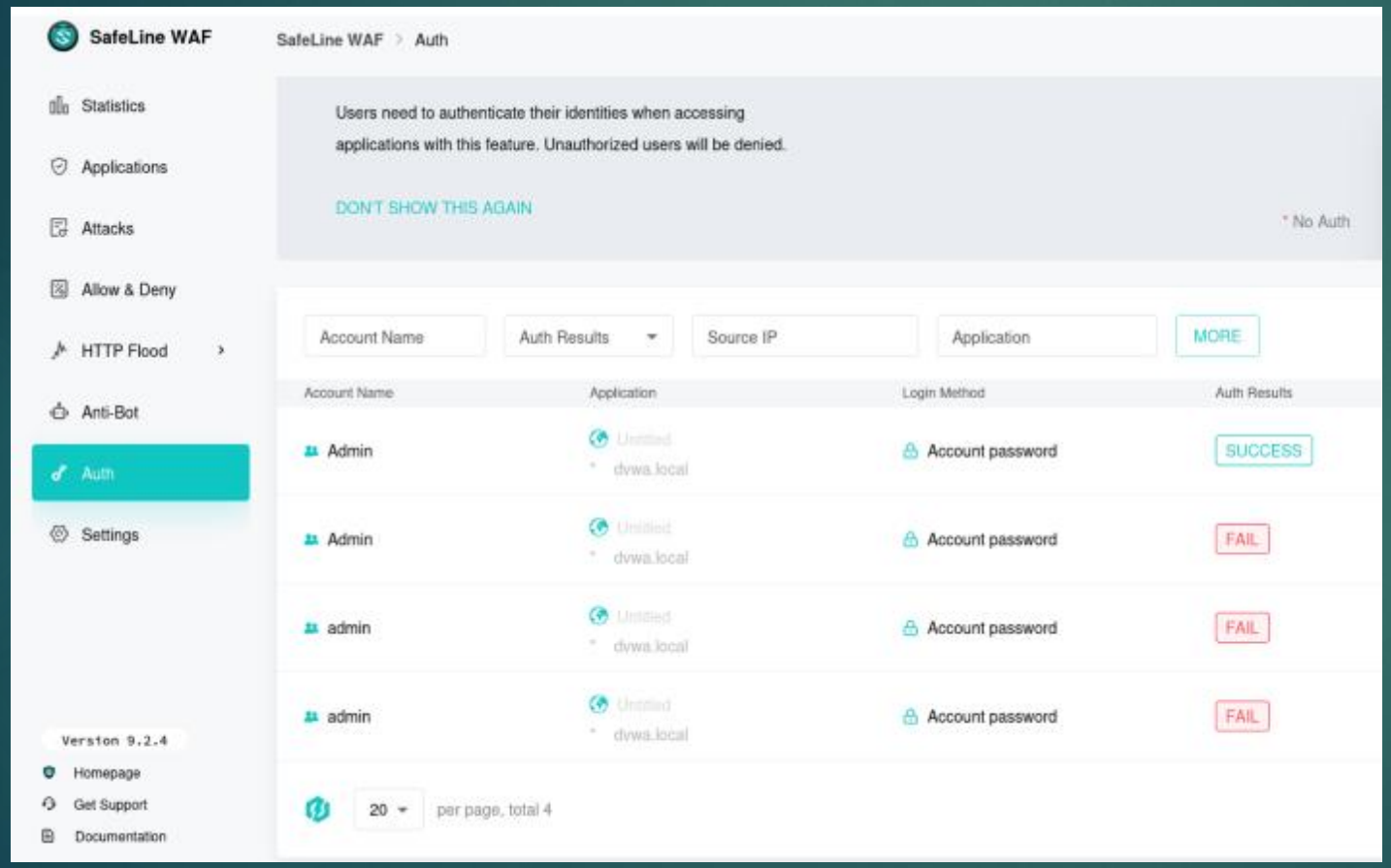
User Account Created



Access Control: Before and After WAF Authentication



Reviewing Authentication Logs



Key Activities:

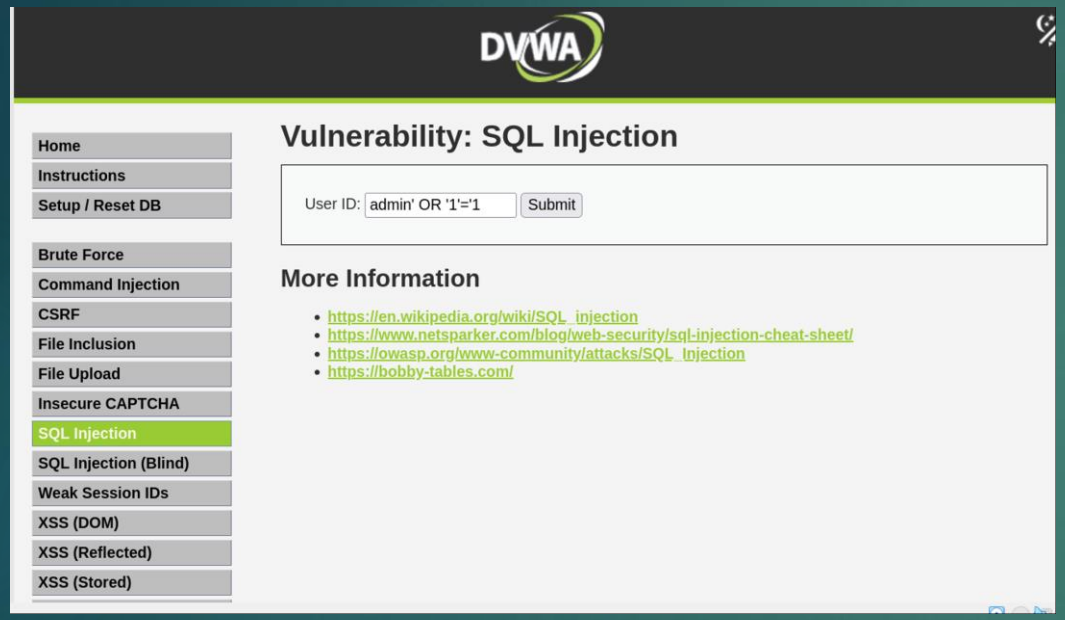
Located the authentication section in the WAF dashboard.

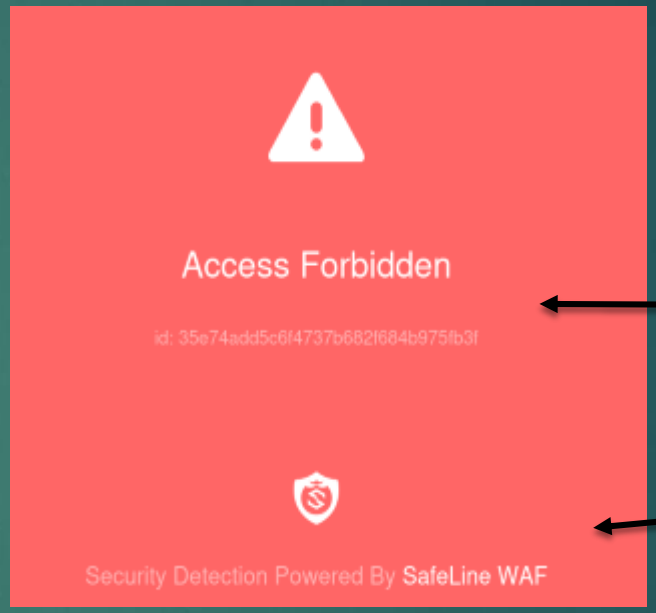
Filtered logs to view login events and access attempts.

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



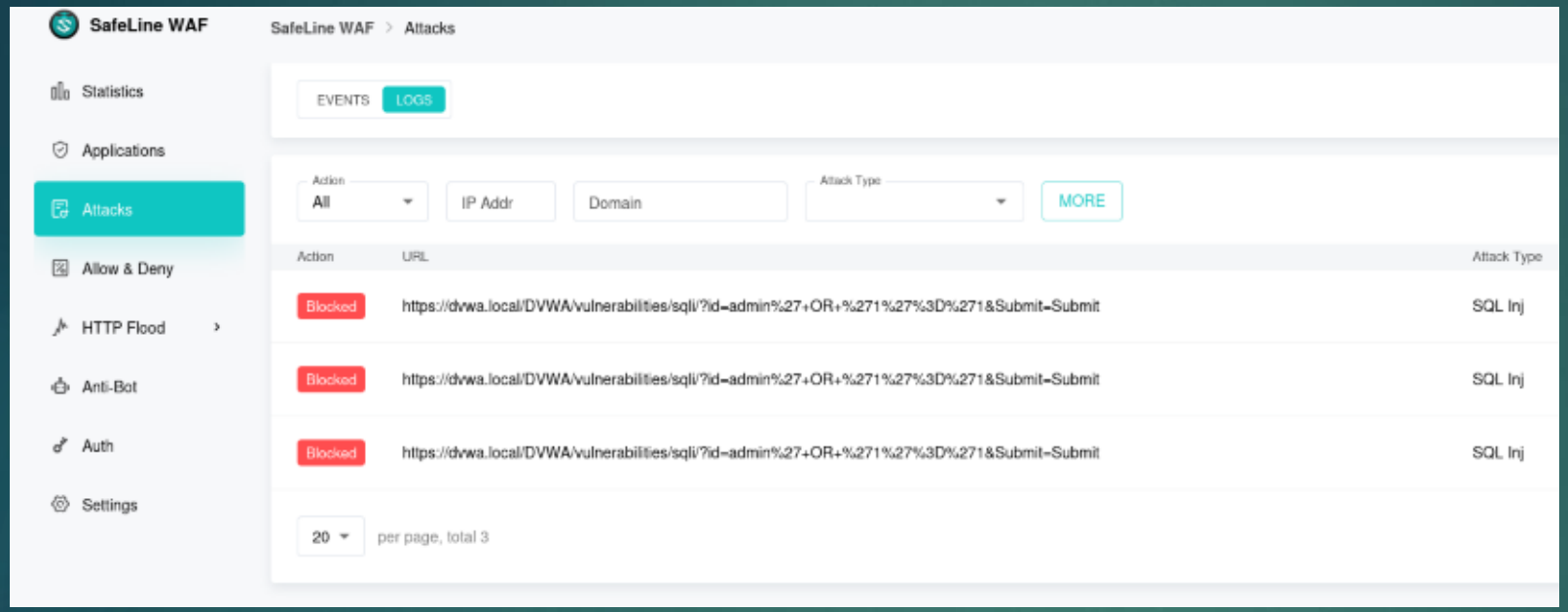


Attack Simulated: A standard SQL injection payload (admin' OR '1'='1) was sent to the login form.

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



Key Findings in the Logs:

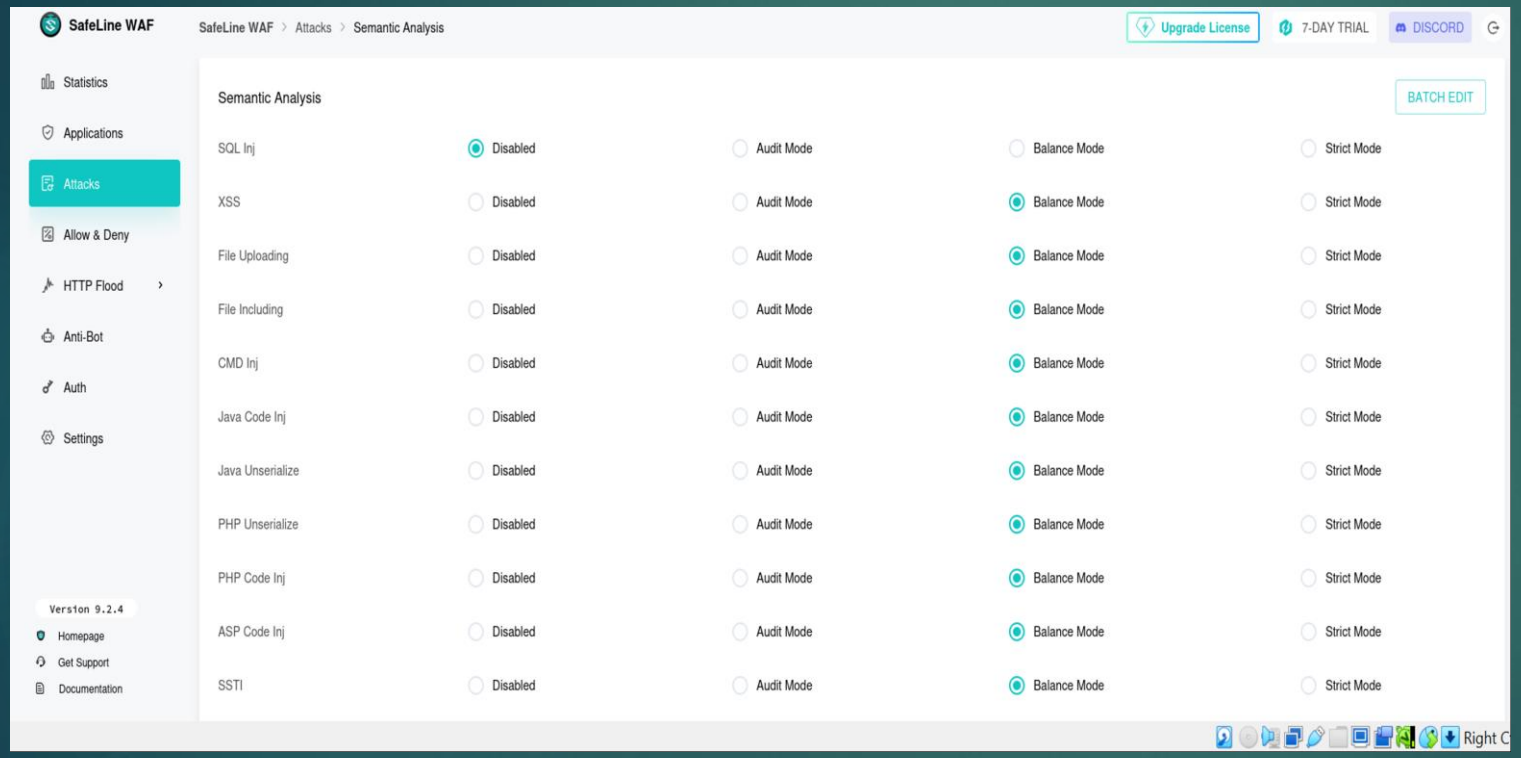
Threat Identified: Logs show repeated SQLi payloads (e.g., admin' OR '1'='1) detected.

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



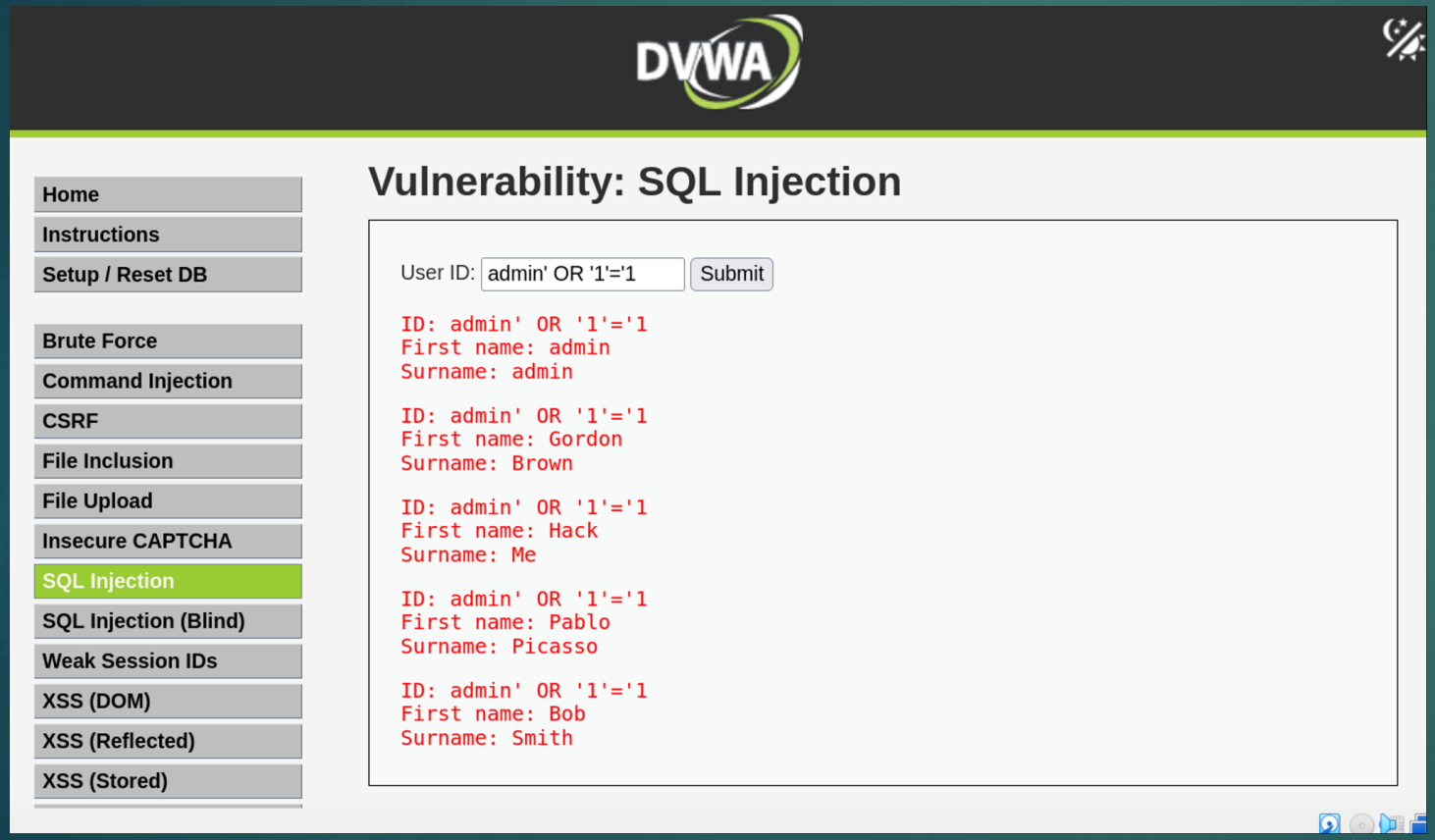
SQL Injection protection is turned off.

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



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