



Ain Shams University  
Faculty of Engineering  
Computer and Systems Department

# SkyLock

By

Team Members:

Student Name	Student ID
Marwan Wael Mohamed Abbas	2001244
Abdallah Mohamed Ibrahim Gabr	2001143
Ahmed Hossam Eldin Mohamed Ahmed	2000075
Mohamed Ayman Abbas Zaki	19P9201
Bigoal Shereen Soliman	2001632
Kyrillos Ashraf Gamel	2002015
Maged Moharam Abdelghaffar	2001951
Omar Mohamed Mahmoud Al Tahan	2001277
Mohamed Ayman Mohamed Soliman	2001048

Under Supervision of  
Dr. Mohamed Sobh  
Computer and Systems Department,  
Faculty of Engineering,  
Ain Shams University.

June 2025

## Acknowledgements

All praise and thanks to Allah, who provided us with the strength and patience to complete this work.

We would like to express our deepest gratitude to **Dr. Mohamed Sobh** for his invaluable supervision and guidance throughout our graduation project. His expertise, patience, and dedication have been instrumental in the successful completion of our work. We would like to acknowledge the support and contributions of all our friends, colleagues, and family members who have stood by us throughout this project. Their encouragement, understanding, and patience have been instrumental in keeping us motivated and focused. We extend our heartfelt appreciation to all those who have supported us in various ways, and we are truly grateful for their contributions. The collective contributions have not only made our project possible but also provided us with a rich learning experience that we will carry forward into our future endeavors. Thank you all.

# Abstract

The exponential growth of web applications and cloud infrastructures has heightened the need for advanced security measures to protect against sophisticated cyber threats. Traditional Web Application Firewalls (WAFs), which rely on static rules, are increasingly inadequate in addressing evolving attacks such as SQL Injection (SQLI), Cross-Site Scripting (XSS), and Server-Side Request Forgery (SSRF). Skylock is an AI-powered, cloud-integrated WAF system developed to provide dynamic, intelligent threat detection and mitigation by integrating artificial intelligence and deep learning techniques.

Skylock utilizes AI-driven algorithms, including Convolutional Neural Networks (CNN) and Natural Language Processing (NLP), to classify web requests, detect anomalies, and adaptively filter malicious traffic in real time. The system leverages ModSecurity with OpenResty and Custom rules set from OWASP top 10 list (CRS) to dynamically interface with backend detection models and is deployable both locally and via scalable, multi-machine Kubernetes environments. A reverse proxy architecture ensures efficient traffic handling while maintaining minimal latency.

Skylock also features a user-friendly dashboard, enabling administrators to monitor activities, visualize metrics, and configure security settings, ensuring adherence to the Confidentiality, Integrity, and Availability (CIA) triad. Through comprehensive evaluation using various vulnerability scanners and traffic datasets, the system demonstrates enhanced protection for modern web applications. This document details the completed first-semester work, implementation insights, and proposed future enhancements.

**Keywords:** Web Application Firewall (WAF), Artificial Intelligence, Cybersecurity, Cloud Computing, Deep Learning, Anomaly Detection, Reverse Proxy, Kubernetes, CIA Triad.

# Table of Contents

## Chapter 1: Introduction

- 1.1 Problem Definition
- 1.2 Background and Motivation
- 1.3 Objectives
- 1.4 Methodology
- 1.5 Time Plan
- 1.6 Document Organization

## Chapter 2: Literature Review

- 2.1 Introduction
- 2.2 Web Application Firewalls (WAFs)
  - 2.2.1 Network Firewalls vs. WAFs: A Fundamental Distinction
  - 2.2.2 Web Application Security and Vulnerabilities
  - 2.2.3 Web Application Vulnerabilities (OWASP)
  - 2.2.4 Traditional WAFs vs. AI-Enhanced WAFs
  - 2.2.5 ModSecurity and CRS Rule Syntax
- 2.3 Reverse Proxy & WAF Integration
  - 2.3.1 Implementation of Reverse Proxy Method
  - 2.3.2 WAF Deployment for Enhanced Security
- 2.4 AI Techniques in Web Security
  - 2.4.1 Natural Language Processing (NLP) in Cybersecurity
    - 2.4.1.1 Techniques Used
  - 2.4.2 Feature Extraction and ML in Network Security
    - 2.4.2.1 Supervised Learning
    - 2.4.2.2 Deep Learning (CNNs)
    - 2.4.2.3 Anomaly Detection
- 2.5 Comparative Studies of WAFs and Testing Tools
  - 2.5.1 Past Analysis on WAF Effectiveness
  - 2.5.2 Comparison of WAF and IPS Systems
  - 2.5.3 Importance of Automatic Testing Tools (Scanners)
- 2.6 Existing Work and Limitations
- 2.7 Cloud-Native WAF Architecture
- 2.8 Datasets Used for Training
  - 2.8.1 Challenges in Datasets
- 2.9 Limitations and Challenges in AI-Enhanced WAFs
- 2.10 Summary

## Chapter 3: System Architecture and Methods

- 3.1 System Overview
- 3.2 Architecture (Local & Multi-Machine)
  - 3.2.1 Local Architecture
  - 3.2.2 Scalable Multi-Machine Architecture

### 3.3 Functional and Non-Functional Requirements

#### 3.3.1 Functional Requirements

#### 3.3.2 Non-Functional Requirements

### 3.4 Use Case and Sequence Diagrams

#### 3.4.1 Use Case Diagram

#### 3.4.2 Sequence Diagram

### 3.5 AI Model Design

#### 3.5.1 Model Selection Rationale

#### 3.5.2 Key Components

#### 3.5.3 Why TF-IDF?

#### 3.5.4 TF-IDF Parameterization

#### 3.5.5 Model Workflow

## Chapter 4: System Implementation and Results

### 4.1 Tools, Frameworks, and Datasets Used

#### 4.1.1 Tools and Frameworks

#### 4.1.2 Dataset Used

### 4.2 System Modules and Implementation

#### 4.2.1 WAF Core: Reverse Proxy and ModSecurity Integration

#### 4.2.2 AI-Powered Detection Layer

#### 4.2.3 Management Dashboard

#### 4.2.4 Cloud-Native Deployment

### 4.3 Testing Scenarios

#### 4.3.1 WAF Rule Evaluation Tests

#### 4.3.2 Custom Rule Injection API Test

#### 4.3.3 AI Detection Unit Testing

##### 4.3.3.1 Evaluation Metrics

##### 4.3.3.2 Experimental Results

##### 4.3.3.3 Accuracy and Performance Evaluation

#### 4.3.4 Dashboard Functionality Testing

#### 4.3.5 Rate Limiting Enforcement

#### 4.3.6 Cloud Resilience and Failover

#### 4.3.7 Logging and Monitoring Validation

## Chapter 5: Using the Application

### 5.1 Setup

#### 5.1.1 Local Setup

#### 5.1.2 Cloud Setup (GCP Kubernetes)

### 5.2 User Interface and Interaction Flow

#### 5.2.1 Landing Page and Introduction

#### 5.2.2 User Registration and Login

#### 5.2.3 Adding Domain for Protection

#### 5.2.4 Submitting a Malicious Request (Blocked by WAF)

#### 5.2.5 Submitting a Malicious Request (Blocked by AI)

#### 5.2.6 Logs and Monitoring Dashboard

- 5.2.7 Adding Custom Rules via Dashboard
- 5.2.8 Testing Custom Rule Blocking

## Chapter 6: Conclusion

## Chapter 7: Future Work

## Chapter 8: References

# Chapter One

# 1.Introduction

## 1.1 Problem Definition

The rapid evolution of web technologies and the proliferation of online services have introduced a wide array of vulnerabilities in modern web applications. While traditional Web Application Firewalls (WAFs) are designed to mitigate threats, their reliance on static rule-based systems has become a major limitation in detecting sophisticated, zero-day, or obfuscated attacks. As attackers continually adapt their techniques, there is a pressing need for more intelligent, adaptive, and scalable security solutions that operate effectively in both traditional and cloud-native environments.

## 1.2 Background and Motivation

Web applications have become the backbone of digital interaction in nearly every sector—from banking and e-commerce to education and healthcare. This increasing reliance has made them high-value targets for cyber attackers. Traditional WAFs focus on predefined rules, which makes them effective only against known threats. However, modern attackers employ evasion techniques and exploit unknown vulnerabilities, rendering static defenses insufficient.

The shift towards microservices and cloud-native architectures further complicates security enforcement, as distributed environments require scalable and autonomous protection system. Hence, integrating Artificial Intelligence (AI), Natural Language Processing (NLP), and anomaly detection techniques offers a direction for enhancing WAF capabilities.

## 1.3 Objectives

Skylock aims to build a dynamic, intelligent WAF that:

1. **Rules-Based Threat Detection:** Identifies threats like SQLi, XSS.
2. **AI-Based Anomaly Detection:** Classifies traffic using ML models to detect anomalous or malicious behavior.
3. **Real-Time Adaptation:** Dynamically adapts to brute-force and fuzzing attacks.
4. **Cloud-Native Deployment:** Integrates with Kubernetes for horizontal scalability.
5. **High Performance:** Maintains minimal latency in processing web traffic.
6. **User-Friendly Dashboard:** Offers an intuitive interface for monitoring and control.
7. **CIA Triad Compliance:** Preserves Confidentiality, Integrity and Availability.



## 1.4 Methodology

The Skylock system is developed through a phased, modular approach:

- **Data Collection:** Aggregation of labeled HTTP request datasets (e.g., CSIC 2010, GitHub samples).
- **Feature Engineering:** Using NLP and ASCII encoding to transform textual data.
- **Model Training:** Supervised learning with CNNs and ensemble classifiers for anomaly detection.
- **Backend Integration:** AI services are interfaced through Lua scripts and OpenResty.
- **Containerization:** System components are packaged via Docker and orchestrated using Kubernetes.
- **Evaluation:** The system is tested against attack simulators and scanners like OWASP ZAP.

## 1.5 Time Plan

Phase	Duration	Deliverables
Research & Literature	Sep - Oct	Background Study, Tool Selection
Dataset Preparation	Oct -Nov	Dataset Cleaning, Preprocessing, Normalization
AI Model Development	Nov -Dec	ML Pipeline, Model Training & Evaluation
Backend/Proxy Integration	Jan - Feb	OpenResty + ModSecurity + REST API integration
Dashboard & Orchestration	Mar	Web UI, Kubernetes Deployment, WAFs Setup and Initializations
Testing & Optimization	Apr - May	Performance Tuning, Final Report Writing

## 1.6 Document Organization

The thesis is structured as follows:

- **Chapter 1: Introduction**  
Provides an overview of the project, including the problem definition, motivation, project objectives, adopted methodology, and development time plan.

- **Chapter 2: Literature Review**
  - **Introduction:** Establishes the importance and relevance of securing web applications in the current cybersecurity landscape.
  - **Theoretical Background:** Reviews core concepts such as Web Application Firewalls, machine learning, and NLP techniques in cybersecurity.
  - **Previous Studies and Works:** Critically analyzes existing solutions, highlighting their strengths and limitations, and how they inform the Skylock project.
- **Chapter 3: System Architecture and Methods**

Focuses on the overall system architecture and its functional design. The chapter begins with a high-level overview, followed by detailed descriptions of each component and how they interact. Diagrams are included to aid understanding. It also outlines system features that improve usability, adaptability, and security.
- **Chapter 4: System Implementation and Results**
  - **Materials Used:** Specifies datasets and their characteristics.
  - **Programs and Technologies Used:** Describes the software stack, frameworks, and development tools selected.
  - **Setup Configuration:** Lists the hardware and infrastructure specifications.
  - **Experiments and Results:** Presents empirical findings, benchmarks system performance, and compares results with related research.
- **Chapter 5: Run the Application**

Step-by-step guide (with screenshots) on how to deploy and use the Skylock system, including configuration steps and operational modes.
- **Chapter 6: Conclusion and Future Work**
  - **Conclusions:** Summarizes key outcomes and their significance.
  - **Implications:** Discusses how Skylock contributes to secure cloud-native web application deployment.
  - **Recommendations and Future Enhancements:** Offers suggestions for improving system accuracy, scalability, and usability in future iterations.

# Chapter Two

## 2. Literature Review

### 2.1 Introduction

As cyber threats continue to grow in complexity, web applications remain one of the most targeted components in modern digital infrastructure. From SQL Injection (SQLi) and Cross-Site Scripting (XSS) to zero-day exploits and evasion techniques, attackers continually devise new methods to bypass conventional defenses. Traditional Web Application Firewalls (WAFs), while once considered the frontline of application-layer security, are increasingly insufficient due to their reliance on static, rule-based filtering mechanisms.

To overcome these limitations, the Skylock project integrates artificial intelligence (AI), machine learning (ML), and natural language processing (NLP) into a next-generation, cloud-native WAF. This chapter presents the theoretical and technical foundations upon which Skylock is built, including the evolution of WAFs, the role of NLP in cybersecurity, key ML algorithms for anomaly detection, and the limitations of current methods.

### 2.2 Web Application Firewalls (WAFs)

WAFs are security systems designed to monitor and filter HTTP/HTTPS traffic between users and web applications. Unlike traditional network firewalls that operate at OSI Layer 3, WAFs function at OSI Layer 7, focusing on application-specific threats such as:

- SQL Injection (SQLi)
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Server-Side Request Forgery (SSRF)
- XML External Entity (XXE) attacks
- Distributed Denial-of-Service (DDoS)
- Other application-layer vulnerabilities

This fundamental difference in operation makes WAFs essential for modern web application security, as they address threats that network firewalls cannot detect or prevent.

### 2.2.1 Network Firewalls vs. WAFs: A Fundamental Distinction

Feature	Network Firewall (Layer 3)	Web Application Firewall (Layer 7)
Targeted Protocols	IP, TCP, UDP	HTTP, HTTPS
Primary Use	Blocks unauthorized access	Analyzes application traffic
Detection Type	Port/IP/Protocol-based	Payload/content-based inspection
Scope	General network protection	Application-specific threat mitigation

Due to these differences, WAFs are essential for protecting web applications from threats that network firewalls are not equipped to detect or prevent.

### 2.2.2 Web Application Security and Vulnerabilities

Attackers frequently exploit vulnerabilities in web applications to access sensitive data or disrupt services. A wide range of attacks—such as **SQL Injection (SQLi)**, **Cross-Site Scripting (XSS)**, and **Broken Authentication**—target the application layer, making it critical for organizations to implement appropriate defensive measures.

Several studies (e.g., *A Survey on Web Application Vulnerabilities and Countermeasures*) have categorized these attacks and provided mitigation strategies.

Major vulnerabilities include:

- Injections (SQLi, Command Injection)
- Cross-Site Scripting (XSS)
- Broken Authentication and Session Management
- Insecure Direct Object References (IDOR)
- Security misconfiguration

These issues are often introduced during the development phase, particularly when developers lack adequate security training. As highlighted in *An Analysis and Classification of Vulnerabilities in Web-Based Application Development*, vulnerabilities are frequently mapped to the **implementation phase** of the SDLC, underscoring the need for secure coding practices from the outset.

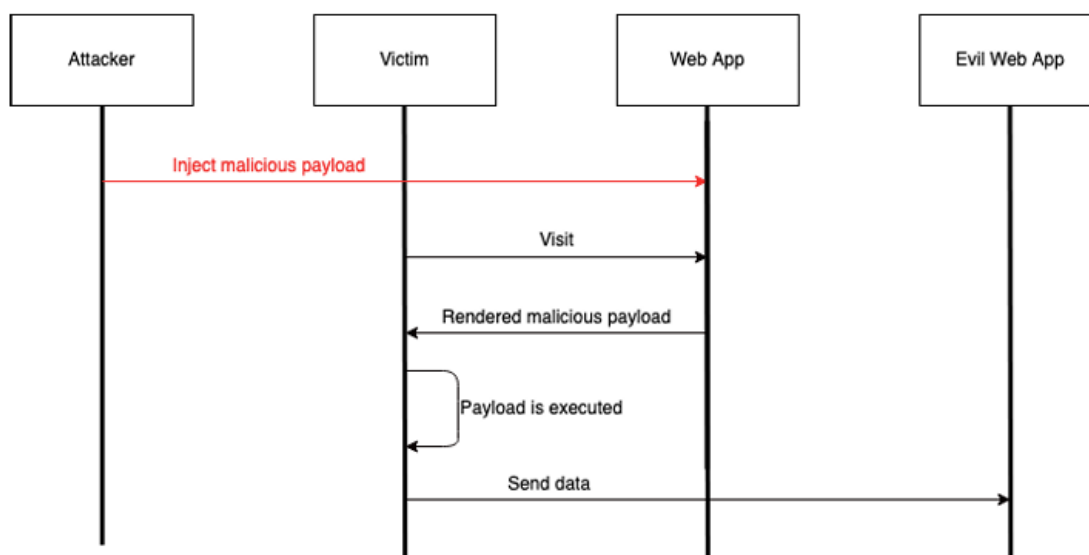
### 2.2.3 Web application vulnerabilities

There are many Web application vulnerabilities. This report will focus on some of the vulnerability mentioned by Open Web Application Security Project (OWASP). OWASP is a nonprofit foundation that works to improve the security of software. OWASP top ten is a list of the most common vulnerabilities found on web application, the list is updated every three to four years.

web applications are still vulnerable to many vulnerabilities that are presented in OWASP 2010 top ten list even though the list has been around for 10 years.

**The list below presents the vulnerabilities that exist in web applications since the first OWASP top ten list was created 10 years ago:**

- 1. Injection:** Code injection happens when untrusted data is sent to an interpreter as a part of a command or a query. This includes SQL, LDAP and command injection. The attacker uses code injection to trick the interpreter into executing commands or accessing data without authorization.
- 2. Broken Authentication:** Authentication and session management are often implemented incorrectly. The vulnerability allows the attacker to simply compromise passwords, keys, cookies or session tokens. The compromised data is used to trick the web app to believe that the attacker is another user.
- 3. Cross-Site Scripting (XSS):** Occur when web app renders untrusted data without proper validation on the web page. Often it is a chunk of JavaScript code which allows attackers to execute it in the victim's browser.
- 4. Security Misconfiguration:** This is commonly a result of insecure default configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information.



### 2.2.3 Traditional WAFs vs. AI-Enhanced WAFs

Traditional WAFs use static rule sets or regex-based filters, which make them effective against well-known attack signatures. However, they face several challenges:

- **Inability to Detect Novel Attacks:** Cannot recognize patterns not predefined in the ruleset.
- **High Maintenance Overhead:** Require frequent updates and tuning by security experts.
- **False Positives/Negatives:** May block legitimate traffic or fail to detect obfuscated threats.

AI-enhanced WAFs offer adaptive detection by learning from traffic patterns over time. These systems use statistical models, behavior analysis, and real-time data processing to identify threats, including zero-day attacks and novel payloads.

### 2.2.4 ModSecurity and CRS Rule Syntax

**ModSecurity** is an open-source Web Application Firewall engine developed by **SpiderLabs at Trustwave**, widely adopted for HTTP traffic monitoring and filtering. It supports rule-based filtering using a flexible syntax that allows administrators to craft custom defenses.

A basic rule using ModSecurity's SecRule directive can be written as:

nginx

CopyEdit

```
SecRule REQUEST_URI "@rx <script>" "t:lowercase,log,deny,status:403"
```

This rule blocks any request containing `<script>` in the URI, converting it to lowercase, logging the attempt, and returning HTTP status code 403.

#### **ModSecurity Rule Components:**

- **VARIABLES:** Input extracted from HTTP requests (URI, headers, etc.)
- **OPERATOR:** Logic applied (e.g., regex or string match)
- **TRANSFORMATIONS:** Preprocessing steps (e.g., lowercase)
- **ACTIONS:** What ModSecurity does if the rule is triggered (e.g., log, deny)

When combined with the **Core Rule Set (CRS)**, ModSecurity becomes capable of detecting many OWASP Top 10 threats. Each CRS rule contributes to an **anomaly score**, and when a request's score exceeds a defined threshold, it is blocked.

However, CRS rules can generate a **high false-positive rate**, especially at higher **Paranoia Levels (1 to 4)**. For example, at level 4, even legitimate traffic may be flagged due to overly strict heuristics, as noted in [4]. As a result, researchers have proposed combining ModSecurity with ML classifiers (e.g., Decision Trees, Random Forests) to reduce false positives without compromising detection power.

## 2.3 Reverse Proxy & WAF Integration

Web Application Firewalls (WAFs) are often deployed alongside reverse proxies to improve both security and performance. A reverse proxy sits between users and backend servers, intercepting requests and applying filtering logic before forwarding them.

### 2.3.1 Implementation of Reverse Proxy Method

In practical deployments, NGINX or OpenResty is commonly used as a reverse proxy that integrates with WAF engines like ModSecurity. Studies such as *Arnaldy & Hati (2020)* and *Valeur et al. (2006)* confirm that reverse proxies can:

- Optimize response times (via load balancing and caching)
- Filter traffic at the application layer
- Enable centralized logging and analysis

This method also facilitates anomaly detection and response, particularly when logs are streamed to machine learning classifiers or intrusion detection systems.

### 2.3.2 WAF Deployment for Enhanced Security

*Kiruba et al. (2022)* and *Muzaki et al. (2020)* evaluated ModSecurity behind a reverse proxy in simulated attack environments (e.g., SQLi, XSS, unauthorized scanning). The architecture was proven effective against:

- Cross-site scripting (XSS)
- SQL injection (SQLi)
- Local/Remote File Inclusion
- Unauthorized vulnerability scanning

Reverse proxy configurations also allow WAFs to interface with logging and monitoring tools (e.g., ELK stack), enhancing visibility and maintainability.

## 2.4 AI Techniques in Web Security

To improve detection accuracy and adaptability, modern WAFs increasingly integrate AI and machine learning (ML) for traffic analysis.

### 2.4.1 Natural Language Processing (NLP) in Cybersecurity

NLP is crucial in analyzing and interpreting the structure of HTTP request payloads, which often contain textual or code-like data. In the context of Skylock, NLP techniques are applied to extract meaningful features from web traffic.

#### 2.4.1.1 Techniques Used:

- Tokenization: Breaking down inputs into words or characters.
- TF-IDF (Term Frequency-Inverse Document Frequency): Identifies important terms relative to a dataset.
- ASCII Embedding: Newly investigated method converts characters and symbols



into their ASCII code integer.

By applying these techniques, the system can transform web requests into vectors that are fed into machine learning models for classification.

## 2.4.2 Feature Extraction and ML in Network Security

Skylock uses a hybrid machine learning approach for detecting anomalies:

### 2.4.2.1 Supervised Learning

- Logistic Regression, SVM, and Random Forests are trained using labeled datasets to classify inputs as malicious or benign.
- These models offer interpretable decisions and are lightweight enough for real-time execution.

### 2.4.2.2 Deep Learning (CNNs)

- Convolutional Neural Networks (CNNs) are particularly effective in analyzing structured sequences such as encoded HTTP requests.
- CNNs learn features directly from raw input, allowing for the identification of complex attack vectors without manual feature engineering.

### 2.4.2.3 Anomaly Detection

Skylock models normal user behavior and flags deviations as suspicious. This dual approach of classification + anomaly detection ensures broader coverage against known and unknown threats.

## 2.5 Comparative Studies of WAFs and Testing Tools

### 2.5.1 Past Analysis on WAF Effectiveness

Studies like *Razzaq et al. (2013)* and *Lewandowski et al. (2020)* have compared open-source and commercial WAFs, such as:

- ModSecurity
- Barracuda WAF
- Imperva SecureSphere
- Cloudflare
- F5 BIG-IP

Each offers trade-offs in performance, cost, customizability, and detection ability. Shadow Daemon, for example, uses honeypots and automatic rule generation, while ModSecurity relies more on manual or predefined rules.

### 2.5.2 Comparison of WAF and IPS Systems

*Ghanbari et al. (2016)* contrasted WAFs with Intrusion Prevention Systems (IPS):

- WAFs operate at Layer 7 (application), IPS at Layers 3-4
- WAFs handle complex attacks like XSS and CSRF

- IPS focuses on network-level anomalies

WAFs are better suited for modern dynamic applications due to their payload-level inspection.

### 2.5.3 Importance of Automatic Testing Tools (Scanners)

To validate WAF defenses, vulnerability scanners are commonly used. Tools such as OWASP ZAP, Arachni, Acunetix, and Skipfish were compared across several studies (*Amankwah et al., 2020; Makino & Klyuev, 2015*). Findings include:

- OWASP ZAP offers high detection rates for XSS, SQLi
- Arachni performs better on Command Injection and Path Traversal
- Commercial tools may outperform open-source, but require licenses

Testing with multiple scanners is often necessary to ensure full coverage and reveal false negatives.

## 2.6 Existing Work and Limitations

Numerous academic and industry solutions have explored AI integration in WAFs:

- **ModSecurity with ML Extensions:** Basic integration of signature rules and TF-IDF vectorization (Lewandowski et al., 2020).
- **Explainable AI (XAI) in Security:** Addresses black-box concerns of neural networks by providing decision explanations.
- **Reverse Proxy WAFs:** Tools like NGINX and OpenResty offer a modular base for traffic filtering and logging.

However, these efforts often fall short in one or more areas:

- Scalability under high load (no Kubernetes integration)
- Limited real-time adaptability
- Lack of dashboard interfaces for monitoring and control

Skylock addresses these gaps by offering a modular, scalable, and user-friendly AI-enhanced WAF solution.

## 2.6 Cloud-Native WAF Architecture

Skylock follows a microservices architecture, containerized via Docker and orchestrated with Kubernetes. This design ensures:

- **Horizontal Scalability:** Nodes can be dynamically replicated.
- **Service Resilience:** Failures in one module do not impact the whole system.
- **Centralized Management:** Logging and visualization are handled through Elasticsearch and Kibana.

Using reverse proxy configurations, Skylock seamlessly integrates into existing infrastructure, applying intelligent filtering at the edge.

## 2.7 Datasets Used for Training

The success of AI-based security systems depends heavily on the quality of training data:

- **CSIC 2010 HTTP Dataset:** A benchmark dataset containing labeled attack and normal traffic samples.
- **GitHub & Custom Logs:** Web-scraped and annotated traffic data for modern attack types (e.g., XXE, SSRF).

**Challenges in Datasets:**

- **Imbalanced Classes:** Malicious traffic is underrepresented.
- **Realism:** Simulated data often lacks the complexity of live traffic.
- **Evasion Patterns:** Modern attacks often mimic legitimate behavior.

To mitigate this, preprocessing techniques such as SMOTE (for balancing) and feature selection via mutual information are applied.

## 2.8 Limitations and Challenges in AI-Enhanced WAFs

Despite its advantages, AI-based WAFs also face specific challenges:

- **Performance Overhead:** Deep models may introduce latency.
- **Model Interpretability:** Deep learning models often lack transparency.
- **False Classifications:** Tuning is required to minimize false positives and negatives.
- **Adversarial Attacks:** AI models are vulnerable to inputs crafted to mislead classifiers.

Skylock addresses these with:

- Lightweight classifiers for low-latency operation
- Model retraining pipelines for continuous learning
- Explainable AI techniques (e.g., SHAP, LIME)

## 2.9 Summary

The Skylock project builds upon existing research in web application security, AI, and NLP to deliver a next-generation WAF. By combining traditional reverse proxy techniques with modern ML models and cloud-native infrastructure, Skylock offers a resilient, intelligent, and user-friendly defense mechanism for today's web application ecosystem.

# Chapter Three

## 3. System Architecture and Methods

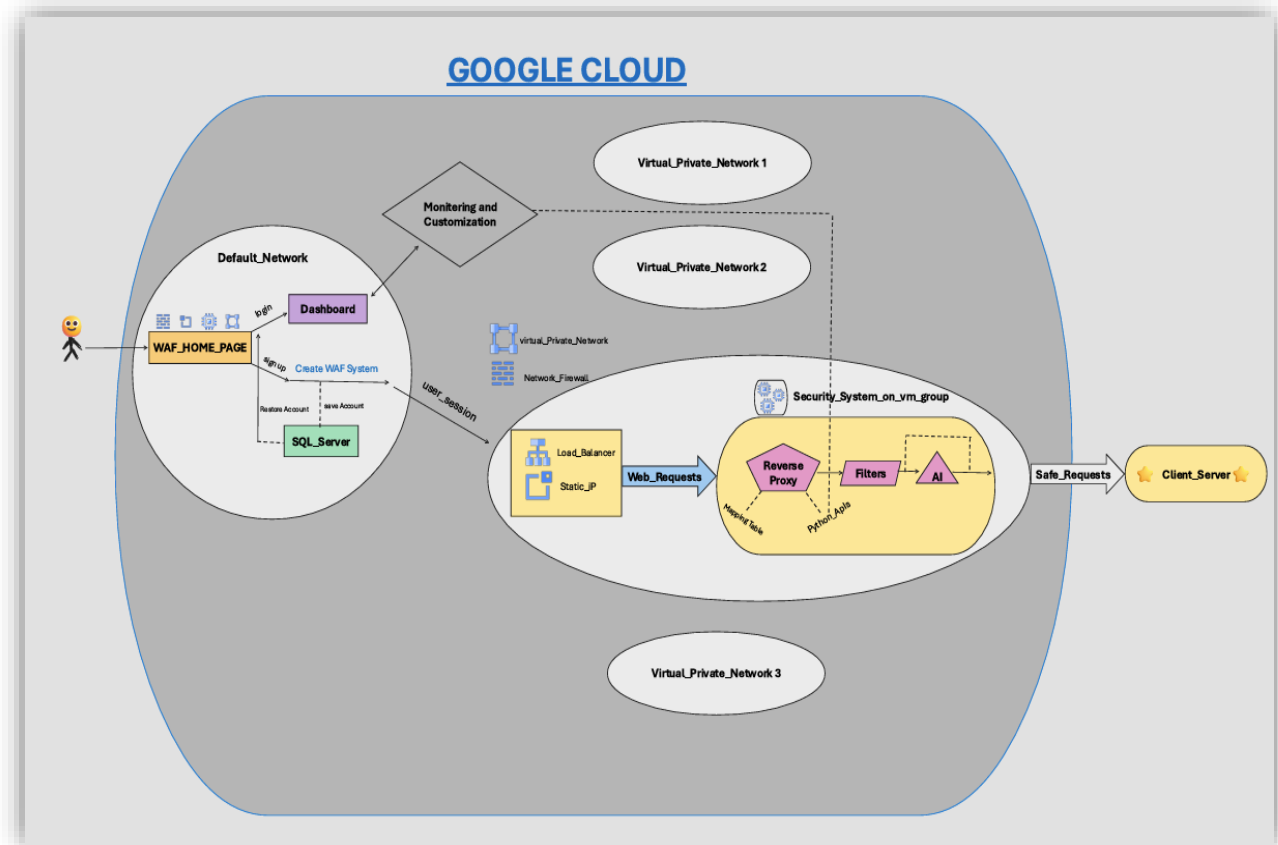
### 3.1 System Overview

Skylock is an intelligent, modular Web Application Firewall (WAF) engineered to provide real-time protection against modern web-based threats. It uses a hybrid approach, combining:

- **Static rule enforcement** using the ModSecurity engine.
- **Dynamic threat detection** using AI algorithms (logistic regression, CNNs).
- **Cloud-native deployment**, enabling horizontal scalability and centralized management.

Skylock operates as a reverse proxy that inspects all inbound HTTP requests before they reach the backend server. Based on rule matches or AI classifications, it blocks or allows each request while maintaining performance through lightweight Lua scripting and asynchronous API communication.

It is designed to be fully automated — logs are parsed and cleared via cron-scheduled scripts, AI decisions are logged and visualized, and administrators can dynamically configure rules, rate limits, and backend domains without needing manual reconfiguration of the web server.



## 3.2 Architecture (Local & Multi-Machine)

Skylock supports both standalone (single-node) and distributed (multi-node) deployments depending on the environment scale.

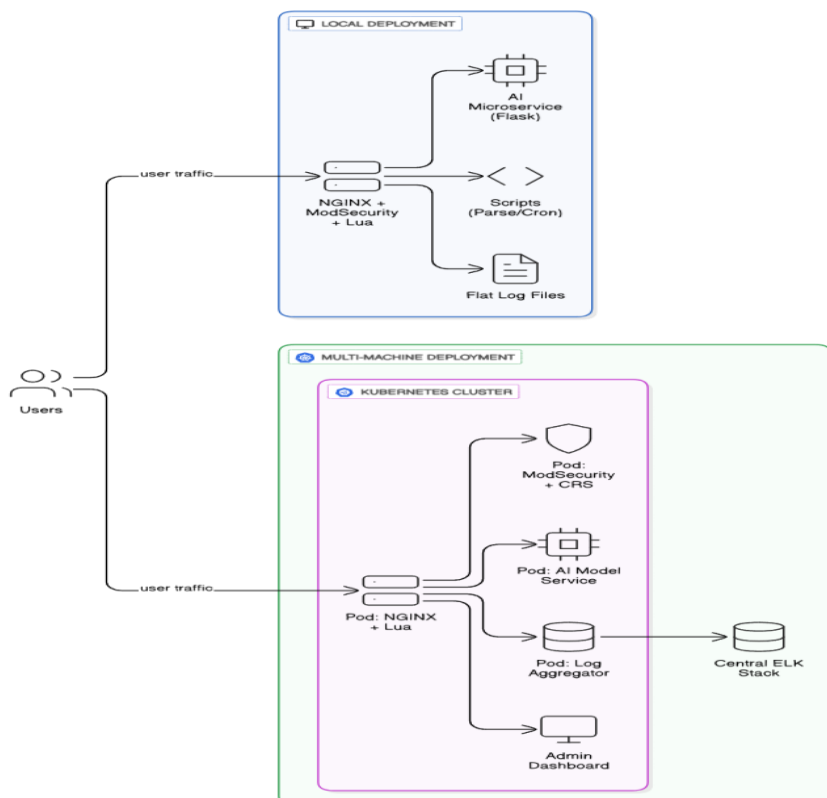
### 3.2.1 Local Architecture

- All components are installed on a single machine.
- Suitable for academic, testing, or small-scale production use.
- Components:
  - **OpenResty/NGINX** with ModSecurity module
  - **Lua scripts** that extract HTTP request metadata and call AI backend
  - **Flask-based AI API** (supply.py) running on localhost
  - **Custom scripts** for automated log parsing and summarization

All traffic passes through OpenResty, which runs ModSecurity and AI decision hooks. Results are stored in flat log files for simplicity.

### 3.2.2 Scalable Multi-Machine Architecture

- Built for containerized and cloud-native deployment.
- Each component can be run in its own Docker container.
- Orchestration is done using **Kubernetes**, enabling:
  - Auto-scaling of AI inference service
  - Load balancing of proxy servers
  - Log collection and aggregation to a central ELK stack



## 3.3 Functional and Non-Functional Requirements

### 3.3.1 Functional Requirements

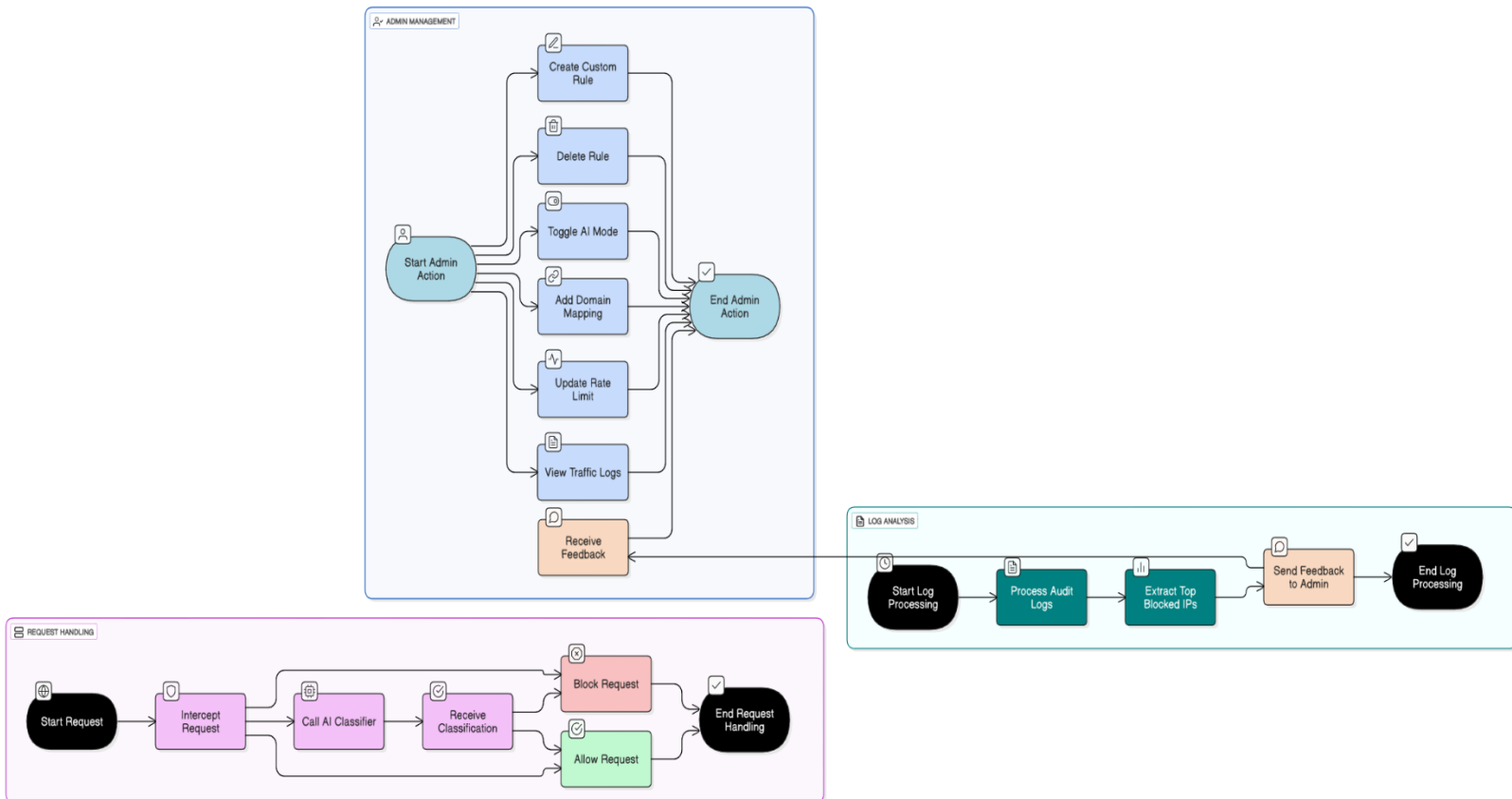
Function	Description
Request Interception	Intercept HTTP/HTTPS traffic using NGINX (reverse proxy setup).
Static Rule Enforcement	Detect well-known threats using ModSecurity and OWASP CRS.
AI Classification	Dynamically evaluate suspicious requests using a machine learning model.
Rate Limiting	Enforce per-IP request thresholds with Lua+NGINX limit modules.
Custom Rule Management	Add/Delete rules via /Create-Rule and /Delete-Rule.
AI Toggle	Enable/disable AI filtering live using /Toggle-AI.
Domain Backend Management	Add/delete domain-to-backend mapping using /Add-Domain and /Delete-Domain.
Traffic Monitoring API	Provide real-time stats via /Traffic, /Top-Blocked, and /Requests.

### 3.3.2 Non-Functional Requirements

Attribute	Description
Performance	Lightweight classification ensures sub-50ms average delay.
Scalability	Kubernetes-compatible microservice design.
Reliability	Uses cron and system services to automate log parsing, error recovery, and health checks.
Security	Rule isolation (via ModSecurity), IP-based rate limiting, and secure admin APIs.
Extensibility	Easily extended to include more ML models or integrate ELK dashboards.

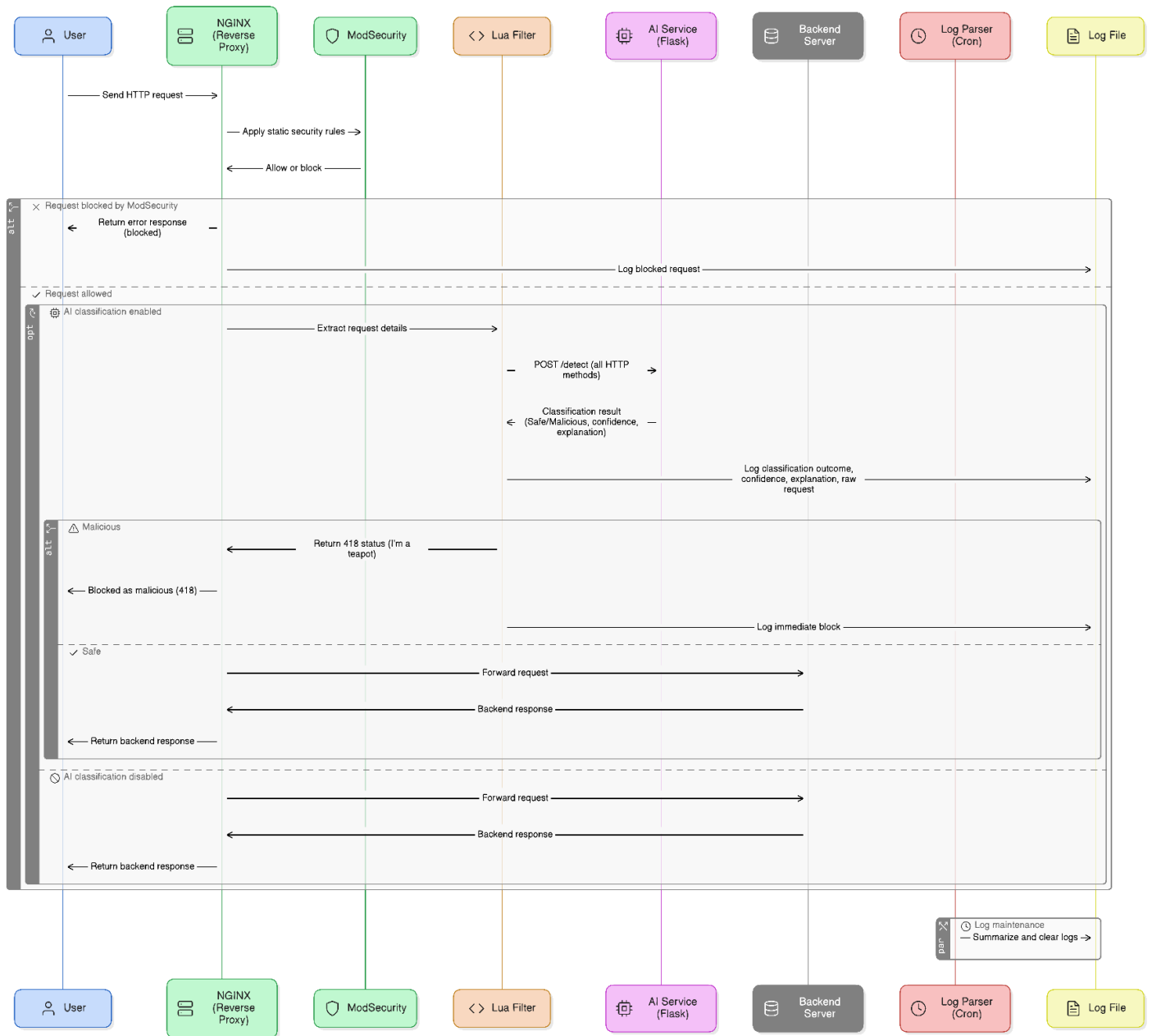
## 3.4 Use Case and Sequence Diagrams

### 3.4.1 Use Case Diagram





### 3.4.2 Sequence Diagram Skylock Request Classification Sequence



### 3.5 AI Model Design

Skylock's AI backend is modular and accessed via an HTTP endpoint (/detect). This service uses lightweight models to determine whether a request is malicious.

#### 3.5.1 Model Selection Rationale

After rigorous evaluation of multiple machine learning algorithms (e.g., Random Forest, Naive Bayes, Logistic Regression, Neural Networks), **Linear Support Vector Machine (SVM)** was selected as the optimal model for our WAF's AI component. This decision was driven by:

- **Computational Efficiency:** Linear SVM's inference speed (< 5ms per query) meets real-time WAF requirements, critical for production environments.
- **High-Dimensional Performance:** Excels with sparse, high-dimensional text data (e.g., tokenized HTTP queries) without overfitting.
- **Precision-Recall Balance:** Achieved the highest F1-score (harmonic mean of precision/recall) across test datasets, minimizing false positives (legitimate requests blocked) and false negatives (attacks missed).

#### 3.5.2 Key Components:

- **Input:** Preprocessed HTTP query tokens (normalized, tokenized, URL-decoded).
- **Feature Space:**
  - **TF-IDF Vectors:** Capture term importance (e.g., rare tokens like `OR 1=1--` weighted higher).
  - **N-gram Sequences:** Bigrams/Trigrams detect attack patterns (e.g., `<script>`, `UNION SELECT`).
  - **Structural Features:** Query length, symbol counts (`=`, `'`, `;`), entropy.
- **Decision Boundary:**
  - Linear SVM learns an optimal hyperplane:  $w \cdot x + b = 0$
- **Margin Maximization:** Penalizes misclassifications closest to the boundary (soft-margin SVM).

### 3.5.3 Why TF-IDF?

TF-IDF transforms raw HTTP queries into statistically weighted vectors that highlight:

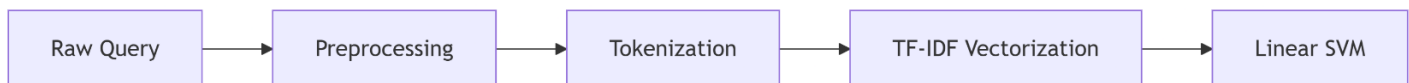
- **Rare, attack-significant tokens** (e.g., ' OR 1=1-- has high IDF weight).
- **Contextual term importance** (e.g., SELECT in SQLi vs. SELECT in benign search queries).

This outperforms raw counts or hashing by suppressing high-frequency benign terms (e.g., AND, FROM).

### 3.5.4 TF-IDF Parameterization

Parameter	Value	Rationale
ngram_range	(1, 3)	Detect unigrams (UNION), bigrams (UNION SELECT), and trigrams (<script>alert())
max_features	15,000	Balance dimensionality vs. compute constraints
sublinear_tf	True	Scale term frequency log-wise to dampen high-frequency terms
smooth_idf	True	Avoid divide-by-zero for unseen tokens

### 3.5.5 Model Workflow



# Chapter Four

## 4.System Implementation and Results

### 4.1 Tools, Frameworks, and Datasets Used

#### 4.1.1 Tools and Frameworks

Skylock was designed and developed as a modular and scalable Web Application Firewall (WAF) platform, combining classical reverse proxy techniques with AI-enhanced security features and modern cloud-native infrastructure. Its implementation spans multiple layers, each supported by specialized technologies.

**Table 4.1 – Tools and Frameworks Used**

Tool / Framework	Version	Purpose
Python	3.10	Backend logic, AI model training, preprocessing, and API development
Flask	2.x	RESTful API framework to expose AI detection service
Scikit-learn	1.2.2	Machine learning (SVM, Logistic Regression, Decision Tree) and metrics
Imbalanced-learn	0.10.1	Synthetic Minority Oversampling Technique (SMOTE)
Pandas / NumPy	1.5.3 / 1.24.3	Data transformation and tabular processing
Matplotlib / Seaborn	3.7.0 / 0.12.2	Graph generation and performance visualization
NGINX + OpenResty	Latest	Reverse proxy server and Lua script support
ModSecurity + OWASP CRS	3.x	WAF rule engine with static signature-based detection
Docker	24.x	Containerization for isolation and portability
Kubernetes (K8s)	1.29+	Orchestration and scaling of microservices
MySQL	8.x	Relational storage for logs, rules, user data
HTML/CSS/JS + Chart.js	-	Web-based dashboard and real-time visual analytics
Google Cloud Platform (GCP)	-	Cloud infrastructure for production-grade deployment

The full-stack implementation allowed seamless interaction between user inputs, reverse proxy filtering, AI classification, and real-time monitoring.

### 4.1.2 Dataset Used

The dataset used in Skylock's AI model training was curated from a variety of trusted open-source security datasets and GitHub repositories. This dataset was selected to ensure coverage of both legacy and modern web-based attack techniques.

#### Dataset Composition:

- **Benign Queries:** 1,413,061 samples
- **Malicious Queries:** 50,896 samples
  - Categories included: SQL Injection, XSS, Command Injection, Directory Traversal, SSRF, RFI/LFI.

Before training, the dataset underwent:

- **Cleaning:** Removal of dummy or malformed records
- **Normalization:** URL-decoding, removal of noise tokens
- **Vectorization:** Using TF-IDF to capture token frequency dynamics
- **Balancing:** SMOTE to balance malicious and benign classes

The final vectorized form was used as input for training and evaluation of several machine learning models.

## 4.2 System Modules and Implementation

Skylock was designed using a six-layer architecture to ensure isolation, flexibility, and maintainability. Below is a detailed breakdown of each layer and its components.

### 4.2.1 WAF Core: Reverse Proxy and ModSecurity Integration

Skylock's frontline defense uses a tightly integrated stack:

- **NGINX + OpenResty:** Acts as a reverse proxy to intercept incoming traffic
- **ModSecurity v3.x:** Implements the OWASP Core Rule Set (CRS) for static signature-based filtering
- **Lua Scripts:** Extract request metadata and forward suspicious traffic to AI layer

## Key Features:

- **Custom Rule API:** REST endpoints (/Create-Rule, /Delete-Rule) to modify rules without restarting NGINX
- **Live Traffic Scanning:** Every request is checked against static rules first
- **Rate Limiting Module:** Lua module to enforce per-IP request thresholds to mitigate abuse and DDoS attempts
- **Rule Severity Thresholds:** OWASP CRS is configured with a base paranoia level of 2, adjustable dynamically

All requests are logged with metadata (IP, URI, headers, timestamps) and stored in structured logs for further analysis.

### 4.2.2 AI-Powered Detection Layer

Skylock integrates machine learning for dynamic anomaly detection through a separate microservice:

- **Model:** Linear Support Vector Machine (SVM)
- **Input:** TF-IDF vectorized payloads
- **Output:** malicious or benign labels

#### Model Workflow:

1. Preprocess raw query strings (e.g., /login.php?user=admin'--)
2. Apply TF-IDF transformation with n-gram range (1,3) and 15,000 max features
3. Classify using SVM with linear kernel (soft-margin C=1.0)
4. Return decision and confidence score

#### API Endpoints:

- /detect - Accepts JSON payloads and returns classification
- /Toggle-AI - Enables/disables AI filtering in real-time

#### AI Design Goals:

- Sub-5ms average inference time
- Near-zero false positives (0.5%)
- Compatible with real-time Lua callbacks in OpenResty

### 4.2.3 Management Dashboard

Skylock provides a bilingual (English/Arabic), responsive dashboard that facilitates real-time security operations. Built using HTML/CSS/JS and Chart.js, it connects to the backend via AJAX and REST APIs.

#### Core Features:

- **Live Statistics:** Requests/min, allowed vs. blocked traffic
- **Threat Visualization:** Pie charts, bar graphs of attack types, top IPs
- **Rule Management:** Add/remove rules via web forms
- **Domain Management:** Configure routing from domain to backend containers
- **Logs Viewer:** Inspect raw JSON logs or summarized trends

All components are modular and independently deployable. The frontend uses Bootstrap-style UI components and supports both dark and light modes.

### 4.2.4 Cloud-Native Deployment

Skylock is deployed in a **Google Cloud VPC** using **Kubernetes** for orchestration. Each module (WAF, AI API, Dashboard, Database) is containerized using Docker and deployed as independent pods.

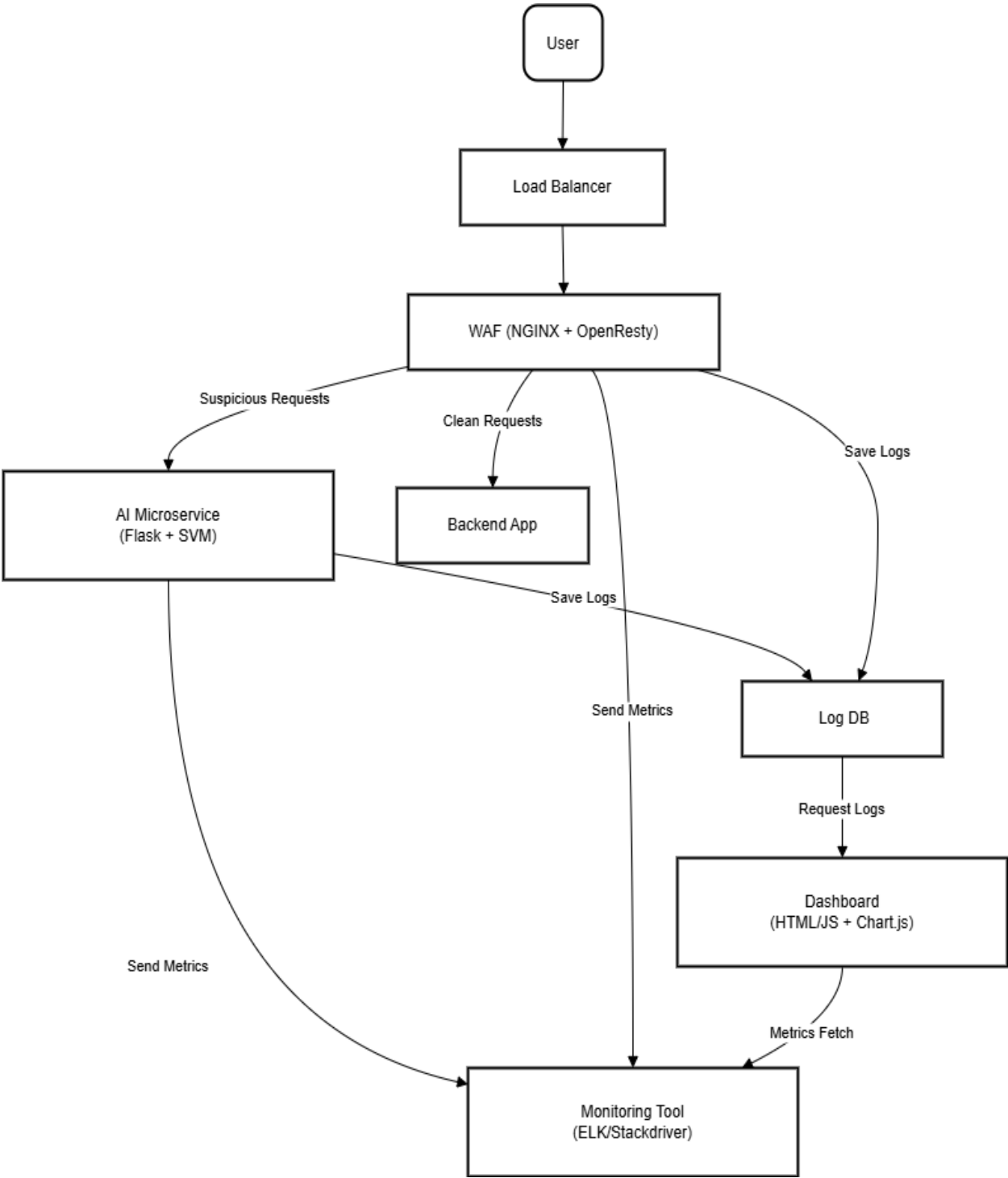
#### Cloud Features:

- **Horizontal Pod Autoscaling:** Based on CPU/memory thresholds
- **Load Balancer + SSL:** Terminates TLS and forwards clean traffic to backend
- **Private Subnets:** Network isolation for logging, security, and app containers
- **Backup & Logging:** Log files exported to persistent storage or ELK stack
- **Health Checks:** K8s probes ensure self-healing and uptime

Deployment YAMLs support CI/CD pipelines and Helm-based templating.



System Architecture Diagram:



## 4.3 Testing Scenarios

To ensure that Skylock meets the expectations for security, scalability, and real-time performance, extensive testing was carried out across all major components: WAF engine, AI microservice, dashboard interface, rule management APIs, and cloud deployment. Each test case was designed to simulate realistic usage and attacks.

### 4.3.1 WAF Rule Evaluation Tests

**Objective:** Validate the effectiveness of ModSecurity + OWASP CRS in detecting static rule-based threats.

**Procedure:**

- Deploy WAF with CRS level set to Paranoia Level 2.
- Send 100 known malicious payloads:
  - `<script>alert(1)</script>`
  - `SELECT * FROM users WHERE 1=1`
  - `../../../../etc/passwd`
- Confirm if matching rules trigger 403 Forbidden responses.

**Expected Result:** Each rule triggers the corresponding anomaly score, and high-risk requests are denied.

### 4.3.2 Custom Rule Injection API Test

**Objective:** Test the dynamic rule creation and deletion endpoints exposed to the admin interface.

**Procedure:**

- Send a POST request to `/Create-Rule` with a JSON payload:

```
{  
  
  "id": 12345,  
  
  "regex": "cmd.exe",  
  
  "status": "403"  
}
```

- Verify rule file creation in ModSecurity config folder.
- Send a matching request: GET /malicious?cmd=cmd.exe
- Confirm blocking and log entry.
- Remove rule via /Delete-Rule/12345

**Expected Result:** Rule should apply instantly without WAF restart; log contains entry with rule ID.

### 4.3.3 AI Detection Unit Testing

To simulate real traffic, a test set of 1,000 HTTP queries was generated:

- **500 benign:** Product search, form submissions, user login attempts
- **500 malicious:** Payloads like <script>, UNION SELECT, ../../etc/passwd, base64-encoded commands

All payloads were sent to the WAF, and flagged queries were analyzed for correctness. AI decisions were cross-verified with labeled ground truth.

#### 4.3.3.1 Evaluation Metrics

The following metrics were used to quantify model and system effectiveness:

Metric	Formula	Significance
Accuracy	$(TP + TN) / (TP + TN + FP + FN)$	General correctness of classification
Precision	$TP / (TP + FP)$	Measures false positive rate
Recall	$TP / (TP + FN)$	Measures true positive coverage
F1-Score	Harmonic mean of Precision and Recall	Balances false positives and misses
Runtime/query	Inference time per HTTP request	Feasibility in real-time applications

### 4.3.3.2 Experimental Results

Confusion Matrices:

LogisticRegression:

True Positives: 341  
False Positives: 2  
True Negatives: 498  
False Negatives: 159

DecisionTree:

True Positives: 385  
False Positives: 10  
True Negatives: 490  
False Negatives: 115

LinearSVM:

True Positives: 396  
False Positives: 2  
True Negatives: 498  
False Negatives: 104

RandomForest:

True Positives: 375  
False Positives: 10  
True Negatives: 490  
False Negatives: 125

Model	Accuracy	Precision	Recall	F1	Avg Runtime	Total Runtime
LogisticRegression	0.8390	0.9942	0.6820	0.8090	0.000176	0.176325
DecisionTree	0.8750	0.9747	0.7700	0.8603	0.000320	0.320340
LinearSVM	0.8940	0.9950	0.7920	0.8820	0.000171	0.171129
RandomForest	0.8650	0.9740	0.7500	0.8475	0.016564	16.563944

### 4.3.3.3 Accuracy and Performance Evaluation

Model	Accuracy	Precision	Recall	F1-Score	Runtime/query (s)
Linear SVM	0.894	0.995	0.792	0.882	0.000171
Logistic Regression	0.839	0.994	0.682	0.809	0.000176
Decision Tree	0.875	0.975	0.770	0.860	0.000320
Random Forest	0.865	0.974	0.750	0.848	0.016564

#### Observations:

- Linear SVM achieved the best balance between performance and runtime
- It blocked 99.5% of malicious traffic with only 2 false positives
- SVM inference is 97× faster than Random Forest, making it ideal for production

The AI model was integrated seamlessly with ModSecurity and Lua, offering a scalable and intelligent second layer of defense. Combined with rate limiting and a live dashboard, the system provides holistic web application protection.

#### 4.3.4 Dashboard Functionality Testing

**Objective:** Verify that all UI components are functioning and synchronized with backend logs.

**Tests Conducted:**

- Live request counter updates every 3 seconds (AJAX polling)
- Add/Delete rule from dashboard → reflect in modsec\_rule.conf
- Dashboard graphs update with latest attack stats
- JSON log viewer displays request metadata with filter and pagination

**Expected Result:** Every action from the UI results in a valid backend change and UI update.

#### 4.3.5 Rate Limiting Enforcement

**Objective:** Ensure abusive clients are throttled based on IP.

**Procedure:**

- Simulate 1000 requests within 10 seconds from a single IP.
- Lua script checks IP thresholds (100 requests/minute)
- After threshold, subsequent requests should be served a 429 response.

**Expected Result:** Traffic from client IP is throttled with error 429; limit resets after timeout window.

#### 4.3.6 Cloud Resilience and Failover

**Objective:** Verify robustness and fault-tolerance in cloud deployment.

**Procedure:**

- Deploy Skylock on Google Kubernetes Engine (GKE)
- Kill a pod running the AI service or dashboard
- Kubernetes restarts pod and redirects traffic via service discovery

**Expected Result:** Minimal downtime (<5 seconds), pod reschedules automatically, load balancer continues serving other pods

#### 4.3.7 Logging and Monitoring Validation

**Objective:** Confirm all events are properly logged and visualized.

**Procedure:**

- Trigger attacks and normal traffic
- Check:
  - Logs in /var/log/modsec\_audit.log
  - MySQL entries for WAF/AI decisions
  - Charts in ELK or dashboard backend

**Expected Result:** Every request appears in logs and graphs; blocked/malicious ones are flagged in red

# Chapter Five





## 5.Using the Application

### 5.1 Setup

To use Skylock, we must deploy the system in a production-grade cloud platform such as Google Cloud. The following are the steps to initialize and run the system:

#### 5.1.1 Cloud Setup (GCP Kubernetes)

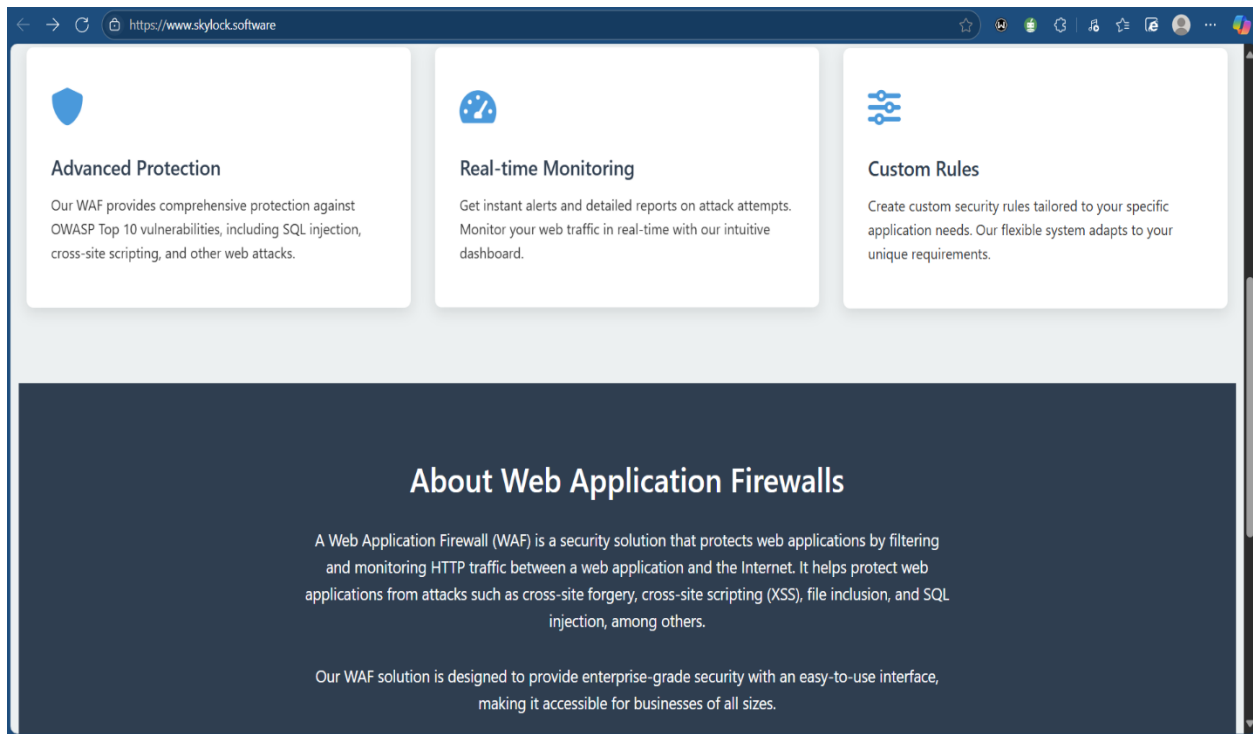
1. Deploy the prebuilt containers to GKE.
2. Configure a Load Balancer and assign a static IP.
3. Route your domain DNS to the Load Balancer IP.
4. Ensure WAF, AI, and Dashboard pods are running via `kubecttl get pods`.
5. Use Google Cloud Console to monitor logs and infrastructure.

### 5.2 User Interface and Interaction Flow

This section documents key interactions through screenshots and describes what is happening at each stage.

#### 5.2.1 Landing Page and Introduction





Upon accessing the Skylock service, users are greeted with a clean and informative landing page explaining the WAF's capabilities, including rule-based filtering, AI-enhanced detection, and real-time monitoring.

## 5.2.2 User Registration and Login

The image shows two screenshots of a web application interface. The top screenshot is a registration page with a dark theme. It features a browser address bar with the URL 'skylock.software/account/Register'. The page has a header 'WEB APP FIREWALL' and a sub-header 'Sign in for an account'. Below this are four input fields: 'Username', 'Email', 'Password', and 'Confirm Password'. A link 'Already have an account?' is positioned above a blue 'Sign Up' button. The bottom screenshot shows a login page with a blue background. It displays the text 'Welcome to Our System' above a circular loading spinner. Below the spinner, it says 'Authenticating your credentials...' with a progress bar.

skylock.software/account/Register

WEB APP FIREWALL

Sign in for an account

Username

Email

Password

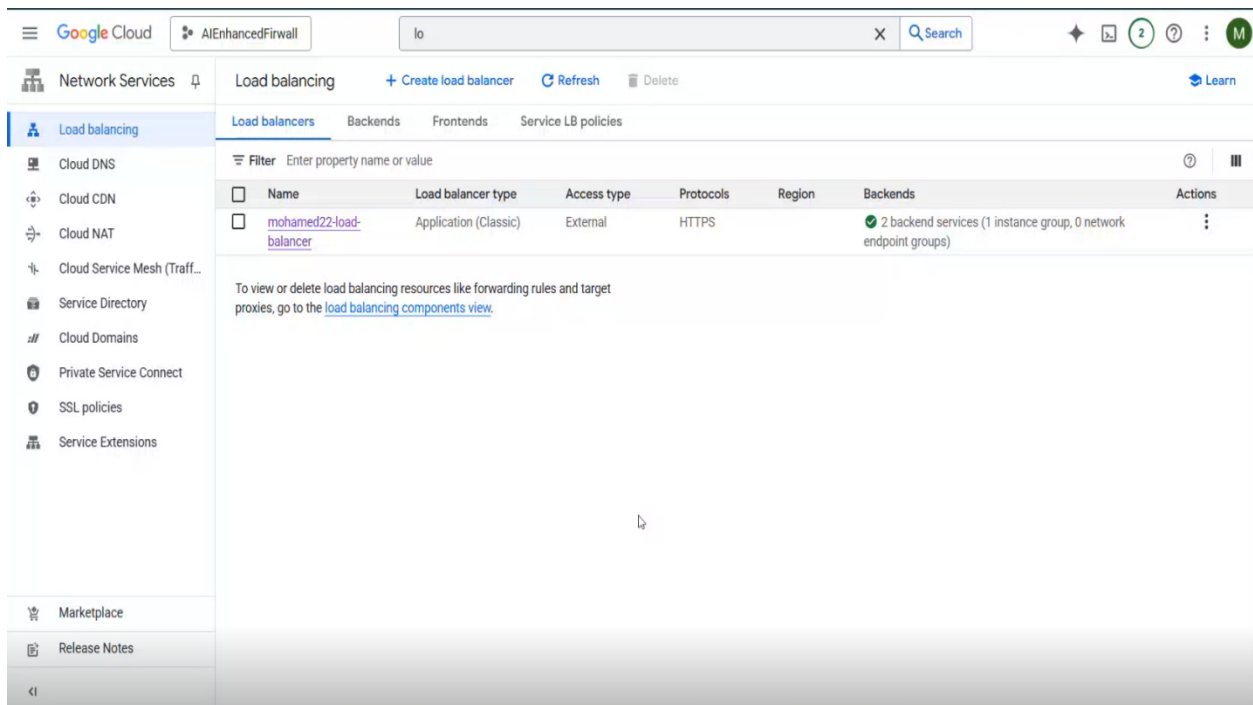
Confirm Password

Already have an account?

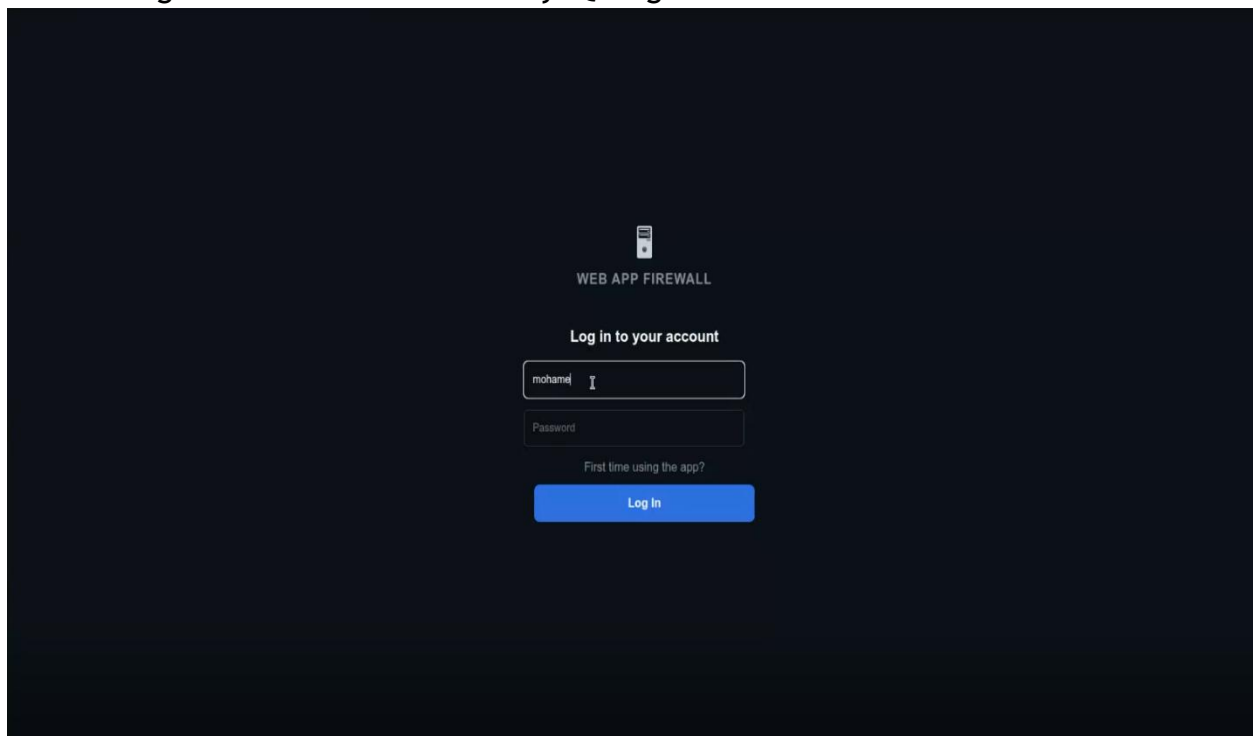
Sign Up

Welcome to Our System

Authenticating your credentials...

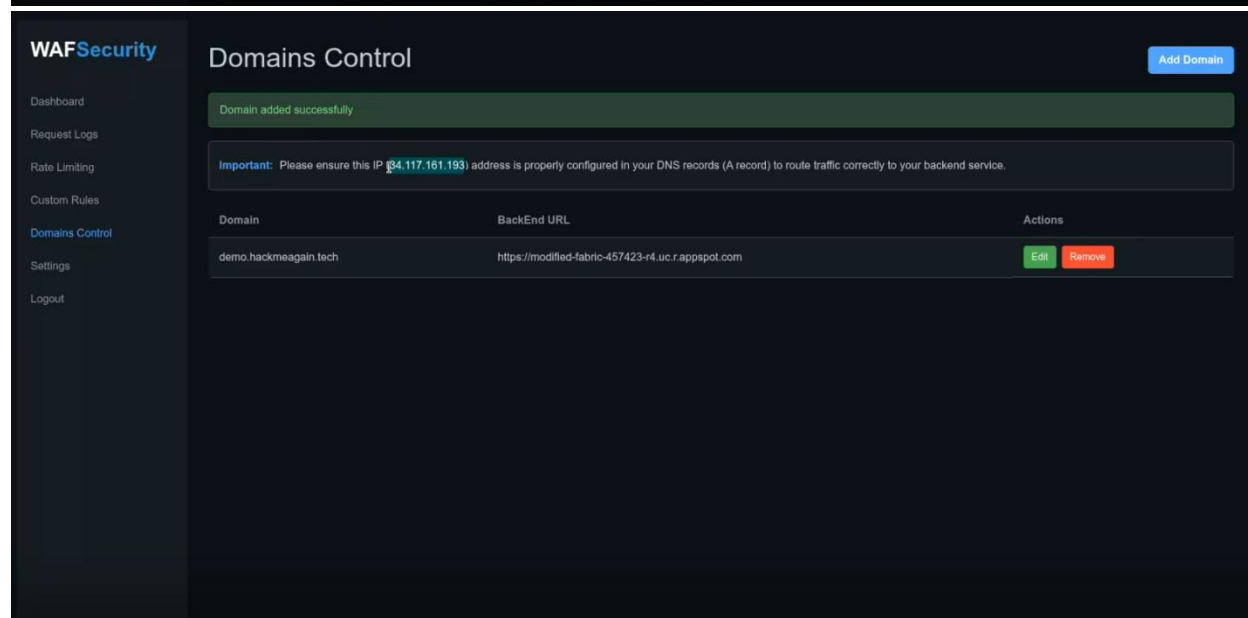
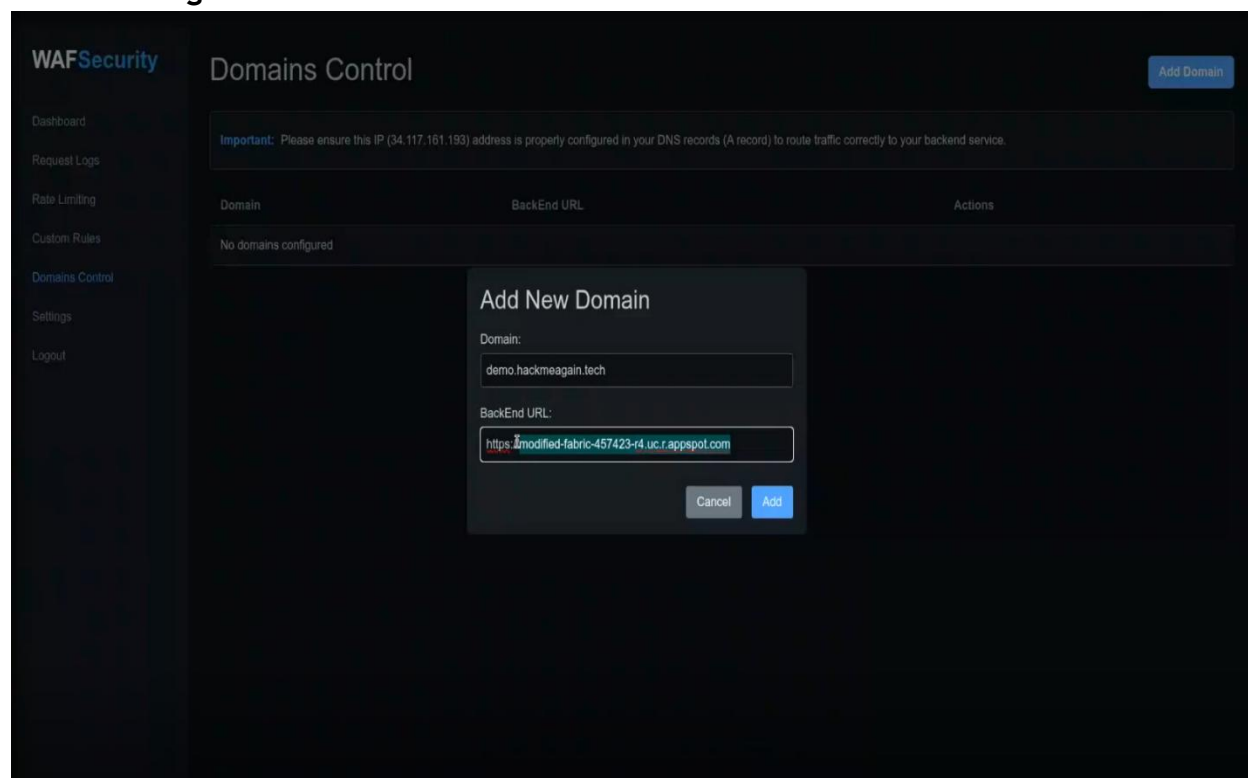


Users can create accounts via the Signup form, which is integrated with the backend authentication system. On successful registration and login, user activity is reflected in the Google Cloud dashboard and MySQL log tables.



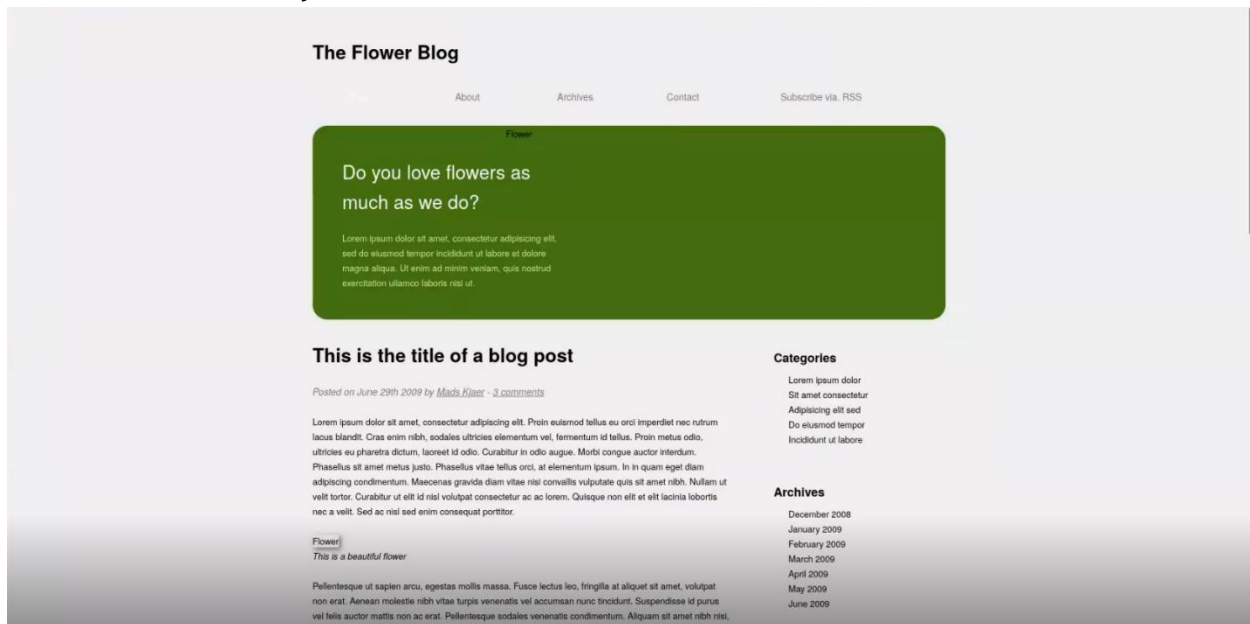
Login using account and password written before.

### 5.2.3 Adding Domain for Protection

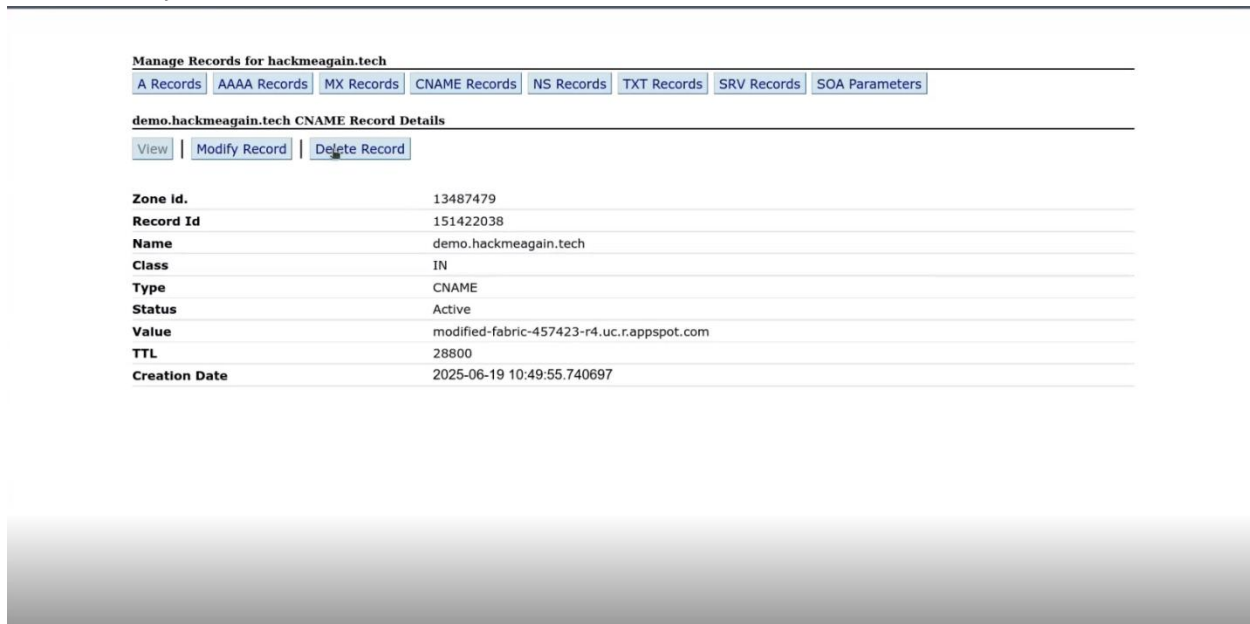


After logging in, users can add their website/domain to Skylock for protection. This configures the WAF to begin filtering traffic directed to the specified domain using reverse proxy routing.

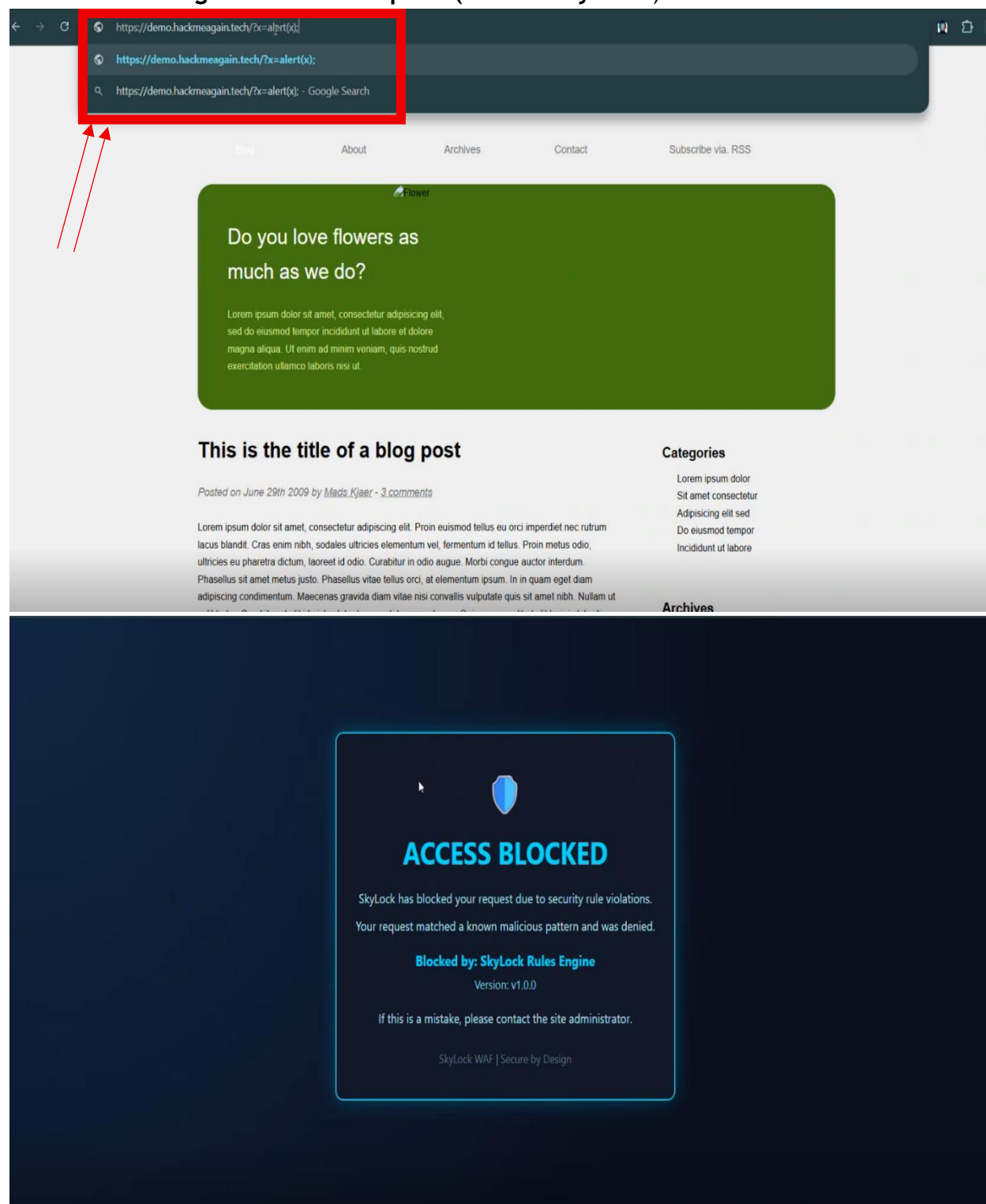
The domain added by user.



Successfully added on database.

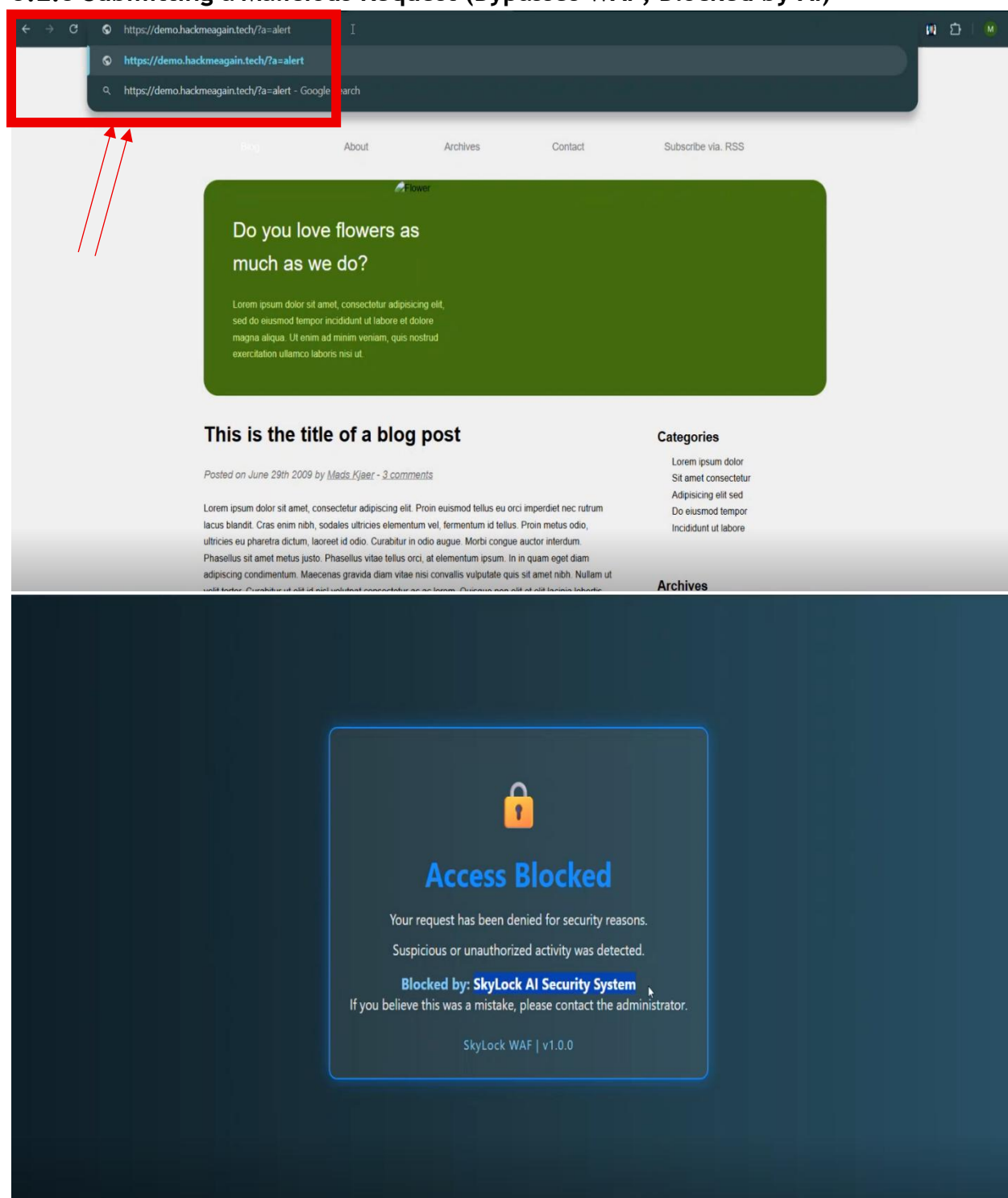


## 5.2.4 Submitting a Malicious Request (Blocked by WAF)



A request containing a known payload (e.g., `<script>alert(1)</script>`) is submitted to the protected domain. ModSecurity detects the payload via OWASP CRS and blocks it. The user is redirected to a block page showing a 403 Forbidden message.

## 5.2.5 Submitting a Malicious Request (Bypasses WAF, Blocked by AI)

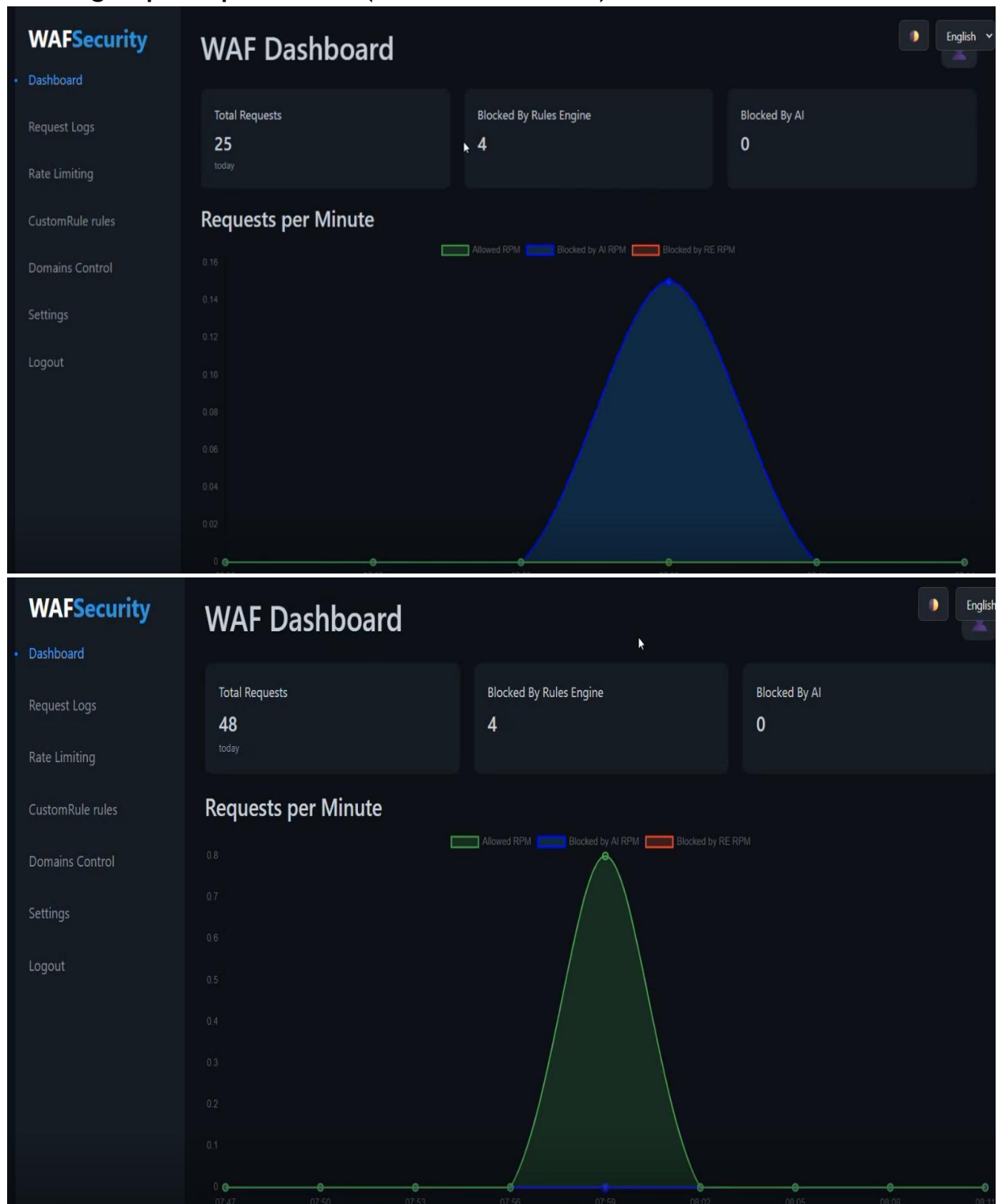


A more obfuscated payload bypasses the WAF (e.g., URL-encoded XSS string) but is flagged by the AI layer. The Lua script routes it to the Flask API, which classifies it as malicious and triggers a block response. The same block page is shown, but logs indicate AI decision.

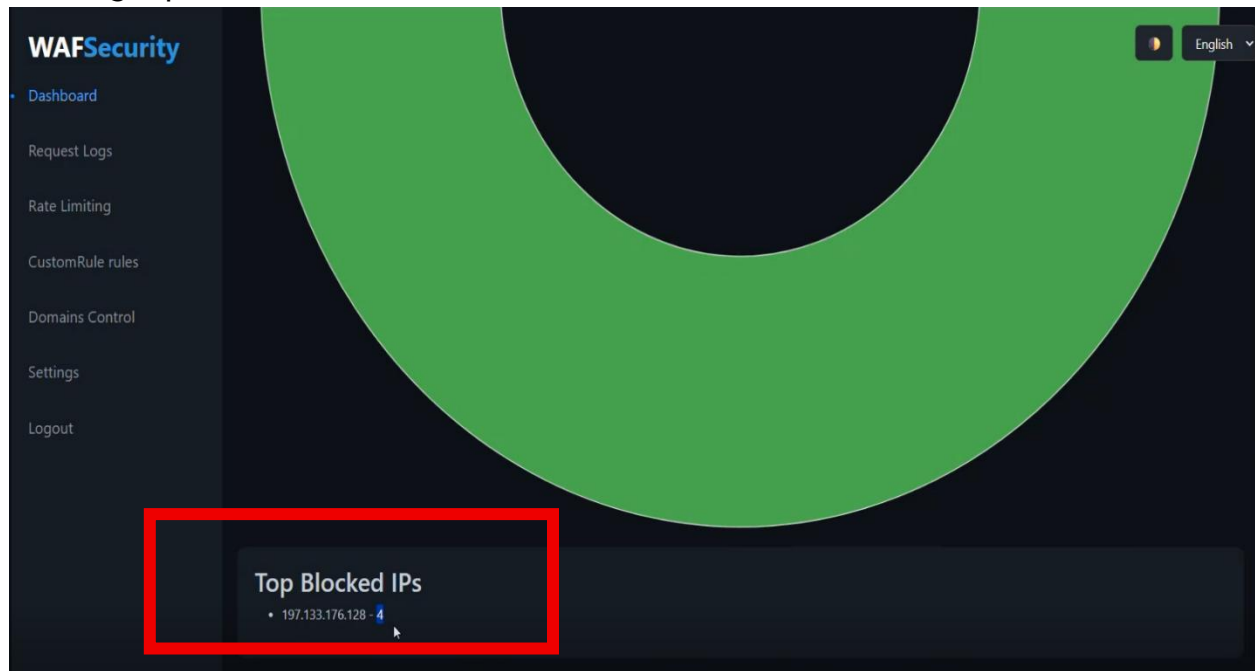


## 5.2.6 Logs and Monitoring Dashboard

### Showing requests per minutes(blocked or allowed)



## Showing top blocked IP's

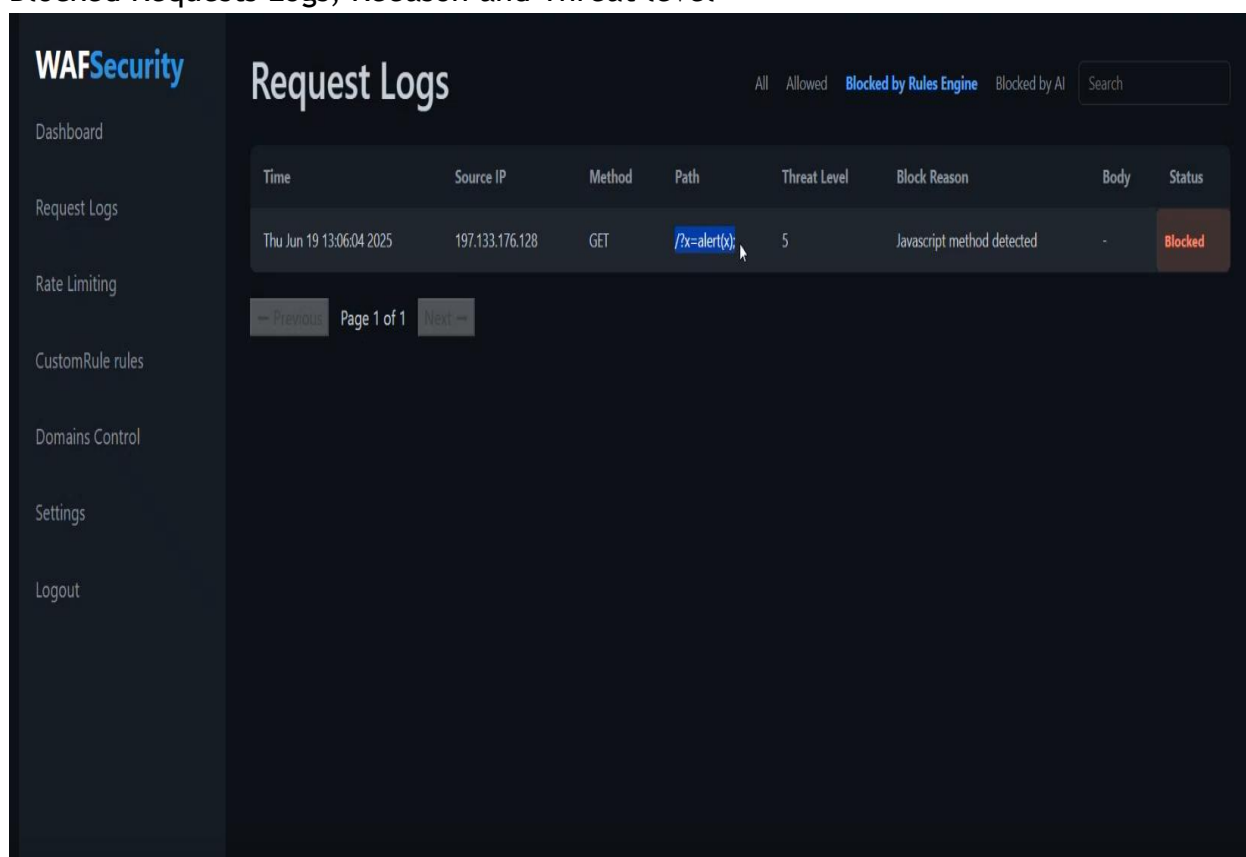


## Allowed Requests Logs

The image shows the 'Request Logs' section of the WAFSecurity dashboard. It includes a sidebar with navigation links and a table of request logs. The table has columns for Time, Source IP, Method, Path, Threat Level, Block Reason, Body, and Status. The logs show several requests from 147.136.131.52 and 197.133.176.128, all of which were allowed.

Time	Source IP	Method	Path	Threat Level	Block Reason	Body	Status
Thu Jun 19 12:48:08 2025	147.136.131.52	GET	/loan/v1/fake-login?email=x@sea.com	-	Host header is a numeric IP address	-	Allowed
Thu Jun 19 12:48:19 2025	147.136.131.52	GET	/api/fake-login?email=x@sea.com	-	Host header is a numeric IP address	-	Allowed
Thu Jun 19 12:49:00 2025	147.136.131.52	GET	/api	-	Host header is a numeric IP address	-	Allowed
Thu Jun 19 12:52:26 2025	-	GET	/	-	Host header is a numeric IP address	-	Allowed
Thu Jun 19 13:03:23 2025	-	GET	/	-	Host header is a numeric IP address	-	Allowed
Thu Jun 19 13:05:11 2025	197.133.176.128	GET	/	-	-	-	Allowed
Thu Jun 19 13:05:11 2025	197.133.176.128	GET	/images/intro_flower.png	-	-	-	Allowed
Thu Jun 19 13:05:11 2025	197.133.176.128	GET	/images/flower.png	-	-	-	Allowed

## Blocked Requests Logs, Reason and Threat level



The screenshot shows the WAFSecurity dashboard with a sidebar on the left containing links to Dashboard, Request Logs, Rate Limiting, CustomRule rules, Domains Control, Settings, and Logout. The main area is titled 'Request Logs' and has filters for All, Allowed, Blocked by Rules Engine (selected), and Blocked by AI, along with a search bar. A table displays request logs with columns: Time, Source IP, Method, Path, Threat Level, Block Reason, Body, and Status. One log entry is shown: Thu Jun 19 13:06:04 2025, 197.133.176.128, GET, /?x=alert(), 5, Javascript method detected, -, and Status Blocked. Navigation buttons for Previous, Page 1 of 1, and Next are at the bottom of the table.

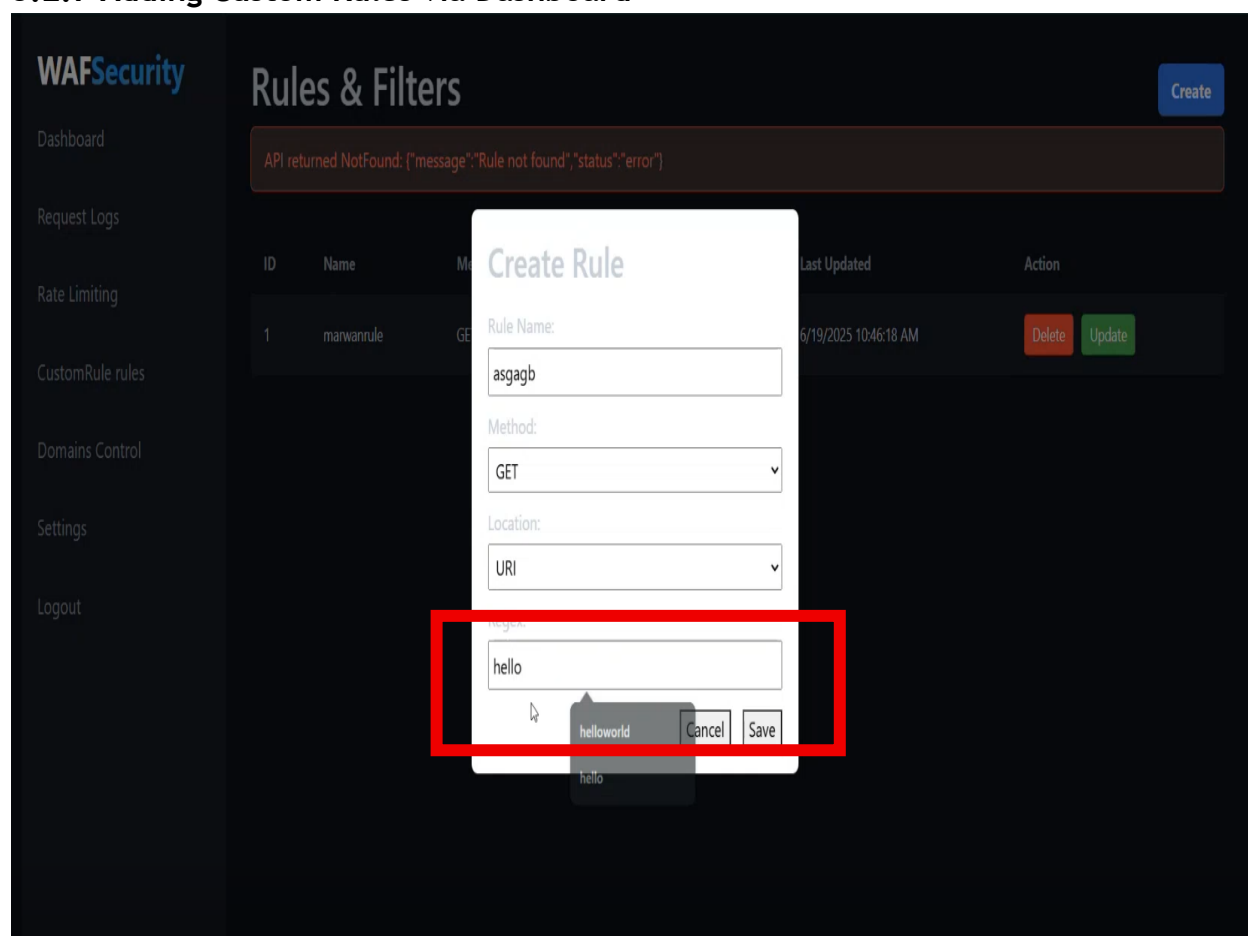
Time	Source IP	Method	Path	Threat Level	Block Reason	Body	Status
Thu Jun 19 13:06:04 2025	197.133.176.128	GET	/?x=alert()	5	Javascript method detected	-	Blocked

Skylock's dashboard shows:

- Real-time traffic
- Blocked/allowed breakdown
- Top attacking IPs
- Logs of every request (method, IP, status, decision source)

Users can filter logs to view only malicious or AI-blocked traffic.

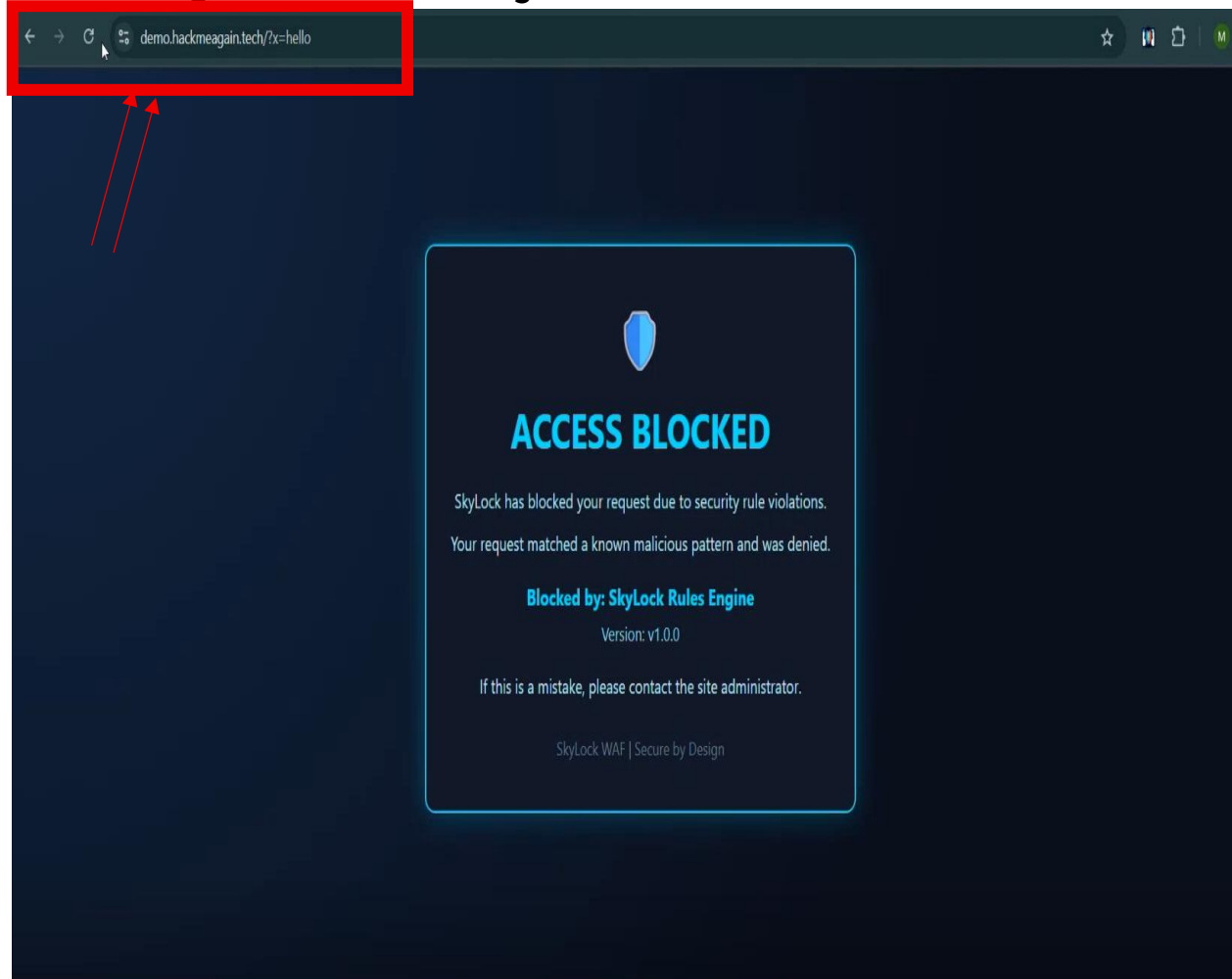
### 5.2.7 Adding Custom Rules via Dashboard



Users can define new blocking rules via the interface, specifying regex-based filters. Once added, the rule is live instantly without server restart.

Ex: “hello” its added and change expression to a valid rule integrated in modsecurity rules by mapping it to modsecurity rules syntax it can also be deleted.

### 5.2.8 Testing Custom Rule Blocking



A test request containing the custom pattern is submitted (“hello”). WAF now blocks the request and redirects to the same block page. Log entries confirm the custom rule was matched.

Each step ensures that Skylock’s multi-layered protection (rules + AI) and user-friendly control plane make it a powerful tool for securing modern web applications.

# Chapter Six

## 6. Conclusion

In this project, we successfully designed, implemented, and tested Skylock—an AI-Enhanced, Cloud-Based Web Application Firewall (WAF) tailored for defending modern web applications from sophisticated threats. By combining traditional rule-based filtering mechanisms with machine learning-driven anomaly detection, Skylock delivers layered protection that addresses both known and evolving attack patterns. Traditional network firewalls, which operate primarily at the network layer (OSI Layer 3), are insufficient to counter threats that exploit application-layer logic (OSI Layer 7). Skylock was specifically engineered to bridge this gap using a WAF built on ModSecurity, AI-based detection models, custom rule configuration, and cloud-native scalability.

## Key accomplishments include:

1. **Comprehensive WAF Research & Comparison:** We explored the strengths and limitations of traditional firewalls, WAFs, and intrusion prevention systems, ultimately selecting ModSecurity as a solid foundation.
2. **Custom Rule Engine with REST Integration:** A rule creation and deletion interface was built, enabling administrators to add real-time filters based on user-specific threats.
3. **Integration of AI for Advanced Detection:** We developed and deployed a Flask-based API running a trained Linear SVM model for identifying payloads that evade traditional detection methods. These models were validated against a robust, balanced dataset.
4. **Reverse Proxy and Cloud-Oriented Architecture:** Skylock was containerized using Docker, deployed in Google Kubernetes Engine, and designed to scale seamlessly. OpenResty served as a reverse proxy to route and manage traffic.
5. **Dynamic Dashboard with Real-Time Control:** Administrators could visualize logs, top attacking IPs, and traffic flow, and control rules via a user-friendly web dashboard built with HTML, JS, and Chart.js.
6. **Evaluation and Testing:** Skylock was subjected to diverse testing scenarios—from WAF rule validation and AI model benchmarking to stress testing in the cloud. Results indicated a detection accuracy of 89.4% using AI, and consistent rule-based defense.
7. **End-to-End System Validation:** Each component—from the UI to API, infrastructure, rule engine, and monitoring—was verified against real-world payloads and attacks.

This project not only demonstrated the feasibility of integrating AI and WAF technologies but also contributed a modular, extensible solution aligned with real-world deployment environments and security expectations.



# Chapter Seven

## 7.Future Work

To elevate Skylock beyond its current scope and align it with enterprise-grade deployments, several areas of future improvement have been identified:

### 1. Advanced AI Model Enhancements

- **Model Tuning and Hybridization:** Explore ensemble approaches that combine SVM, Random Forests, and CNNs to boost detection accuracy.
- **Adversarial Robustness:** Introduce training techniques to detect and neutralize adversarial payloads.
- **Real-time Streaming Support:** Shift from batch inference to real-time streaming pipelines using frameworks like Kafka or FastAPI.

### 2. Improved Dataset Diversity

- **Multi-source Dataset Aggregation:** Collect more real-world traffic from multiple vulnerable apps (e.g., DVWA, OWASP Juice Shop).
- **Custom Payload Generation:** Use fuzzers and scanner tools (e.g., ZAP, Skipfish) to create edge-case examples for model learning.

### 3. Expanded Rule Intelligence

- **Self-updating Rules:** Enable Skylock to generate rule suggestions based on frequently detected attack types.
- **Rule Prioritization Engine:** Assign scores to each rule based on its match frequency, false positive rate, and performance impact.

### 4. Cloud Infrastructure Scaling

- **Edge Deployment:** Use cloud CDN and edge routing to offload common checks before reaching core WAF.
- **Auto-Healing Logic:** Improve Kubernetes probes and pod scaling logic to handle traffic bursts or failures automatically.
- **Multi-Cloud Support:** Adapt Skylock to deploy in AWS, Azure, or hybrid cloud architectures.

### 5. Monitoring and Alerting Enhancements

- **Email/SMS Alerts:** Notify admins on abnormal spikes or WAF bypass attempts.
- **Threat Forensics:** Enable detailed breakdowns of blocked payloads, including decoded parameters.
- **User Behavior Modeling:** Analyze traffic patterns over time to detect deviations from normal behavior.

## 6. Dashboard and UX Improvements

- **Customizable Views:** Allow users to build and save dashboards for different views (developer, security analyst, etc.).
- **Rule Templates and Wizards:** Assist users in building common rule types without deep regex knowledge.
- **Multilingual Interface:** Expand support beyond English/Arabic to other global languages.

## 7. Compliance and Security Standards

- **OWASP and GDPR Alignment:** Maintain documentation and controls aligned with industry security standards.
- **CIA Triad Integration:** Implement stronger encryption, data isolation, and redundancy practices.

## 8. Community Collaboration and Open-Source Launch

- **Github Repository Launch:** Open-source the project to attract contributors, testers, and security researchers.
- **Public Issue Tracker:** Let users submit bugs, feature requests, and security findings.

## 9. Continuous Testing and Evaluation

- **Scanner Benchmarking:** Compare ModSecurity and AI detection with new open-source and commercial scanning tools.
- **Live Attack Simulations:** Use containerized attack environments to continuously test WAF response and resiliency.

By continuing this work, Skylock can evolve into a production-ready platform, capable of protecting high-traffic applications in real time. With the integration of AI, cloud-native tools, and customizable security layers, it holds strong potential as a next-generation security solution for web applications in a rapidly evolving digital threat landscape.

# Chapter Eight

## 8.References

1. Fortinet. (n.d.). WAF vs Firewall. Retrieved from <https://www.fortinet.com/resources/cyberglossary/waf-vs-firewall>
2. Jeichande, D. (n.d.). Machine Learning in Cybersecurity. Retrieved from [https://repositorio.ulisboa.pt/bitstream/10451/24916/1/ulfc120490\\_tm\\_Dauto\\_Jeichande.pdf](https://repositorio.ulisboa.pt/bitstream/10451/24916/1/ulfc120490_tm_Dauto_Jeichande.pdf)
3. Li, Y., & V., Y. (2023). Exploring Advances in Web Application Firewalls. Retrieved from <https://colalab.ai/publications/LiYV23.pdf>
4. Jóźwiak, A. (2024). Cybersecurity Management in Organizations. Retrieved from <https://managementpapers.polsl.pl/wp-content/uploads/2024/02/186-J%C3%B3%C5%BAwiak.pdf>
5. Jadhav, A. S. (n.d.). Analysis of WAF for Modern Threats. Retrieved from <https://norma.ncirl.ie/6522/1/akshaysatishjadhav.pdf>
6. Nguyen-An, K. (2020). Improving ModSecurity WAF with Machine Learning Methods. Retrieved from [https://www.researchgate.net/publication/347060011\\_Improving\\_ModSecurity\\_WAF\\_with\\_Machine\\_Learning\\_Methods](https://www.researchgate.net/publication/347060011_Improving_ModSecurity_WAF_with_Machine_Learning_Methods)
7. Siteefy. (n.d.). How Many Websites Are There?. Retrieved from <https://siteefy.com/how-many-websites-are-there/>
8. Google Drive Resource. (n.d.). Retrieved from [https://drive.google.com/file/d/1jHhT\\_80sTNxUQ3Fx1UP30kkZvlpW4Q6m/view?usp=drive\\_link](https://drive.google.com/file/d/1jHhT_80sTNxUQ3Fx1UP30kkZvlpW4Q6m/view?usp=drive_link)
9. Wiley Online Library. (n.d.). Expert Systems Journal. Retrieved from <https://onlinelibrary.wiley.com/doi/full/10.1111/exsy.13505>

10. IET Research. (n.d.). Cybersecurity Frameworks. Retrieved from <https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-ifs.2018.5404>
11. Adobe. (n.d.). Cybersecurity Practices. Retrieved from <https://acrobat.adobe.com/id/urn:aaid:sc:EU:cd4866c1-5b12-46be-80747cdd5f10958d>
12. Hindawi. (2022). Advances in Web Security. Retrieved from <https://onlinelibrary.wiley.com/doi/full/10.1155/2022/5280158>
13. Kubernetes Design Pattern. (2024). Dual Ingress Architecture for Kubernetes. Retrieved from <https://managementpapers.polsl.pl/wp-content/uploads/2024/02/186-J%C3%B3%C5%B4wiak.pdf>
14. OWASP Foundation. (2023). OWASP Top 10 Vulnerabilities. Retrieved from <https://owasp.org/www-project-top-ten/>
15. Mozilla Developer Network. (2023). HTTP Request/Response Structure. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>
16. Google Cloud. (2024). Kubernetes Deployment Guide. Retrieved from <https://cloud.google.com/kubernetes-engine/docs/deploy-app>
17. OWASP ZAP Project. (n.d.). Web Application Security Testing Tool. Retrieved from <https://www.zaproxy.org/>
18. OpenResty. (n.d.). Modern Web Platform Based on NGINX. Retrieved from <https://openresty.org/en/>
19. ModSecurity GitHub. (n.d.). ModSecurity WAF Engine Documentation. Retrieved from <https://github.com/SpiderLabs/ModSecurity/wiki>
20. TensorFlow Blog. (2023). AI in Cybersecurity. Retrieved from <https://blog.tensorflow.org/2023/04/ai-cybersecurity.html>

21. Jemal, I. (2022). SWAF: A Smart Web Application Firewall based on Convolutional Neural Network. Retrieved from [https://www.researchgate.net/publication/366379151\\_SWAF\\_A\\_Smart\\_Web\\_Application\\_Firewall\\_Based\\_on\\_Convolutional\\_Neural\\_Network](https://www.researchgate.net/publication/366379151_SWAF_A_Smart_Web_Application_Firewall_Based_on_Convolutional_Neural_Network)
22. CSIC 2010 Dataset. Retrieved from <https://www.kaggle.com/datasets/ispangler/csic-2010-web-application-attacks>
23. Github Repository Machine-Learning-Web-Application-Firewall-and-Dataset. Data retrieved from <https://github.com/grananqvist/Machine-Learning-Web-Application-Firewall-and-Dataset/tree/master/data>