ABSTRACT

The increasing number of cyberattacks on web applications highlights the need for advanced and intelligent security mechanisms. This project, titled **"ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall,"** focuses on identifying web vulnerabilities and implementing a strong security solution using machine learning. Initially, a vulnerable web application was developed without any firewall, making it susceptible to attacks such as those listed in the OWASP Top 10. This environment demonstrated how easily attackers can exploit insecure systems.

To address these vulnerabilities, a Web Application Firewall (WAF) was created, incorporating user behavior analysis, anomaly detection, and IP tracking**.** The firewall records user entry and exit times along with IP addresses**,** and all activities are logged into an SQLite database. A machine learning model was integrated to detect unusual behavior patterns, enhancing the system's ability to block potential threats automatically.

This project provides a comparative analysis between the vulnerable and secured web applications, proving the effectiveness of combining traditional security rules with intelligent, data-driven detection. The solution offers a scalable and proactive approach to web application security in real-world scenarios.

ACKNOWLEDGEMENT

We sincerely express our gratitude in depth to our esteemed Founder Chairman & Chancellor Dr. T. R. Paarivendhar, Thiru. Ravi Pachamuthu, Chairman SRM Group & Pro-Chancellor (Admin). Ms. R. Harini, Correspondent, SRM VEC and authorities of SRM Valliammai Engineering College for the patronage of our welfare rooted in the academic year.

We express our sincere gratitude and thankfulness to our respected Director Dr.B.Chidhambara Rajan, M.E., Ph.D., for his constant encouragement ,which has motivated us to strive towards excellence

We sincerely thank our Principal Dr. M. Murugan, M.E., Ph.D., for his constant encouragement, which has been our motivation to strive towards excellence.

We also thank our Vice Principal Dr. S. Visalakshi, M.E., Ph.D., for her constant motivation to strive towards excellence.

We extend our heartfelt gratitude towards our Head of the Department – Computer Application Dr. S. Parthasarathy, M.C.A., M.Tech., M.Phil., M.B.A., Ph.D., Professor for his unstinted support.

We are thankful to Project Coordinator Ms. P. PANDI DEEPA, M.E., Assistant Professor for providing us with the essential facilities.
We are grateful to our internal guide Mr. M. Asan Nainar MCA, M.Tech., Assistant Professor without his valuable guidance and encouragement, this major project would not have been success

We also like to thank all the teaching and non-teaching staff members of our department, for their support during the major project work.

CONTENTS

CHAPTER I

INTRODUCTION

In today's interconnected world, web applications are essential components of modern businesses, services, and communication. With increasing reliance on these systems, securing web applications from cyberattacks has never been more crucial. Hackers and malicious users continuously exploit vulnerabilities in web applications, causing severe damage to businesses, users, and society at large. Ensuring that these applications are protected requires advanced security mechanisms that can evolve to meet new and sophisticated threats.

Traditional security measures, such as firewalls and intrusion detection systems, rely on predefined rules to block known attacks. However, these systems are often limited when it comes to detecting new, unknown, or sophisticated threats. This has led to a growing need for smarter, more adaptable security solutions. The integration of **Machine Learning (ML)** into web application firewalls has emerged as a promising approach to addressing this gap, offering enhanced detection capabilities and providing more robust defenses against attacks.

This project, **"ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall,"** explores the use of machine learning algorithms to enhance the functionality of a web application firewall. The goal is to develop a comprehensive security solution that can not only protect against known vulnerabilities but also detect and prevent novel, emerging threats using intelligent anomaly detection.

1.1 About the Project

The increasing sophistication of cyberattacks on web applications has prompted the need for advanced security measures. A significant part of this project involves creating a **vulnerable web application** to simulate real-world security flaws and vulnerabilities. This web application serves as a demonstration of how easily attackers can exploit common weaknesses, such as those listed in the **OWASP Top 10**, including SQL Injection, Cross-Site Scripting (XSS), and Broken Authentication.

Once the vulnerable application is created, the project proceeds by building a **Web Application Firewall (WAF)** capable of monitoring and filtering traffic to prevent cyberattacks. The firewall protects the application from various attack vectors by inspecting incoming requests and blocking malicious traffic based on predefined rules and patterns. However, traditional WAFs often struggle with new or unknown attack types.

To address this limitation, the project integrates **machine learning algorithms** into the firewall. These algorithms will enable the WAF to learn normal user behaviors and detect anomalies that may indicate potential threats. The firewall will analyze user actions, including **entry and exit times**, **IP addresses**, and overall interactions. This information will be stored in an **SQLite database**, providing a comprehensive log of user activity for further analysis.

The key innovation of this project lies in the integration of **behavioral analysis** with traditional security measures. By using machine learning, the system is able to dynamically adjust to new attack patterns, improving detection accuracy and minimizing false positives. Additionally, the project compares the effectiveness of the vulnerable system with the enhanced WAF, highlighting the improvements in security and threat detection.

Ultimately, this project aims to demonstrate that the combination of machine learning-driven anomaly detection and traditional firewall protection can significantly enhance web application security. The results of this project will show how machine learning can be applied in real-world scenarios to protect web applications from both known and unknown threats.

1.2 Objective

The primary objective of this project, **“ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall,”** is to design and implement a robust security system that uses machine learning for **real-time threat detection** and **anomaly prevention** in web applications. The project aims to develop a web application firewall that can intelligently detect and mitigate cyberattacks based on user behavior, rather than relying solely on static, rule-based systems.

The specific objectives of the project are as follows:

- **Create a Vulnerable Web Application:**
  The first objective is to design a **vulnerable web application** that mimics real-world security flaws. By simulating vulnerabilities such as SQL Injection, XSS, and improper authentication, this application will provide an ideal testbed for understanding the behavior of attackers and the impact of various vulnerabilities on web application security.
- **Develop a Web Application Firewall (WAF):**
  A **custom-built WAF** will be developed to protect the vulnerable web application. The firewall will act as an intermediary between the client and server, inspecting and filtering incoming and outgoing traffic. The primary function of the firewall is to detect and block known attack patterns while allowing legitimate requests to pass through. This will be achieved by applying rule-based filtering and behavior analysis to monitor traffic.
- **Integrate Machine Learning for Anomaly Detection:**
  A key feature of this project is the integration of **machine learning** into the WAF. The machine learning model will be trained to detect anomalies in user behavior by analyzing patterns such as login frequency, request types, and access times. By learning from legitimate user interactions, the system will be able to identify and block suspicious activities indicative of a potential attack, even if the attack is novel and unknown.
- **Behavioral Logging and Analysis:**
  The WAF will log crucial user data, including **IP addresses, login times, and access patterns**, which will be stored in an **SQLite database** for analysis. This information will be used not only to detect anomalies but also for system auditing, helping administrators track and investigate security incidents.

The project ultimately seeks to demonstrate the superiority of combining **machine learning-driven anomaly detection** with traditional security mechanisms, creating a smarter, more adaptive, and more effective defense system for web applications.

1.3 Problem Statement

Web applications have become a fundamental part of modern businesses, but their security remains a significant concern. Despite the widespread use of traditional security measures such as firewalls and intrusion detection systems, these mechanisms are often insufficient in protecting against evolving threats. Many firewalls rely on **signature-based detection**, which can only block known attack patterns and fails to detect new, unknown, or sophisticated threats. This leaves web applications vulnerable to attacks that exploit previously unknown vulnerabilities.

Additionally, many existing **Web Application Firewalls (WAFs)** do not have the capability to **analyze user behavior**. Without understanding **normal user behavior**, it is difficult to detect unusual activity that may indicate a potential attack, such as session hijacking, credential stuffing, or other advanced persistent threats. Furthermore, many WAFs lack proper logging and **traceability**, making it difficult to audit, analyze, and understand security incidents after they occur.

The problems identified are as follows:

- **Limited Adaptability:**
  Traditional firewalls are static and based on predefined rules, making them ineffective against new or evolving attack techniques.
- **Lack of Behavioral Analysis:**
  Current WAFs often fail to monitor and analyze **user behavior**, which can provide crucial insights into potential security risks and early indicators of a breach.
- **Inefficient Detection of Unknown Attacks:**
  Signature-based firewalls struggle to identify attacks that do not match predefined patterns, leaving applications vulnerable to novel threats.
- **Insufficient Logging and Audit Capabilities:**
  Many systems lack robust logging mechanisms to track user activity, making it difficult to investigate attacks or learn from previous incidents.

This project addresses these challenges by integrating **machine learning-based behavioral analysis** into a WAF to detect and mitigate attacks in real time. By providing a **comparative analysis** between a vulnerable web application and a secured version, the project demonstrates the value of combining traditional security mechanisms with modern, intelligent threat detection systems

CHAPTER II

II. SYSTEM ANALYSIS

System analysis is a crucial step in understanding the current landscape of web application security, identifying its flaws, and proposing a more effective solution. This chapter provides an analysis of the existing web application security systems, identifies their limitations, and presents the **Proposed System** — an intelligent **ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall**. Furthermore, it examines the advantages of the proposed system and explores its feasibility.

2.1 Existing System

The existing system for web application security largely relies on traditional **Web Application Firewalls (WAFs)** and intrusion detection systems (IDS), which primarily focus on blocking known attack patterns. These systems typically operate using signature-based detection mechanisms, where they compare incoming traffic against predefined attack signatures to identify malicious requests. In addition, many web applications utilize basic security measures such as **input validation**, **rate limiting**, and **secure communication protocols** (e.g., HTTPS), which aim to mitigate specific vulnerabilities.

Some of the most widely used systems include:

- **Traditional Web Application Firewalls (WAFs):**
  These firewalls inspect HTTP requests and responses, filtering traffic to block malicious requests based on predefined rules. They detect known attacks, such as **SQL Injection** and **Cross-Site Scripting (XSS)**, by comparing the incoming request against a database of signatures or rules. While effective against certain attack patterns, traditional WAFs often fall short when it comes to new or sophisticated attack methods.
- **Intrusion Detection Systems (IDS):**
  IDS systems monitor network traffic and server logs to identify suspicious behavior or unauthorized access attempts. While IDS can alert administrators about potential security breaches, they lack the ability to prevent attacks in real-time.
- **Basic Input Validation and Authentication Mechanisms:**
  Many web applications still rely on simple input validation techniques (e.g., rejecting invalid characters in form fields) and basic authentication measures (e.g., password-based authentication) to secure user interactions. While these measures are necessary, they are not sufficient in preventing more advanced attacks or malicious user behavior.
- **Static Security Testing Tools:**
  Tools like vulnerability scanners and static analysis tools are used to test the web application for known vulnerabilities, such as outdated libraries or unpatched security flaws. However, they typically only identify issues at the time of testing, and any new vulnerabilities discovered later may remain unaddressed.

These systems, while valuable in providing foundational security, are often reactive in nature, relying on predefined rules or signatures. They cannot dynamically adapt to new threats or detect abnormal user behavior in real-time.

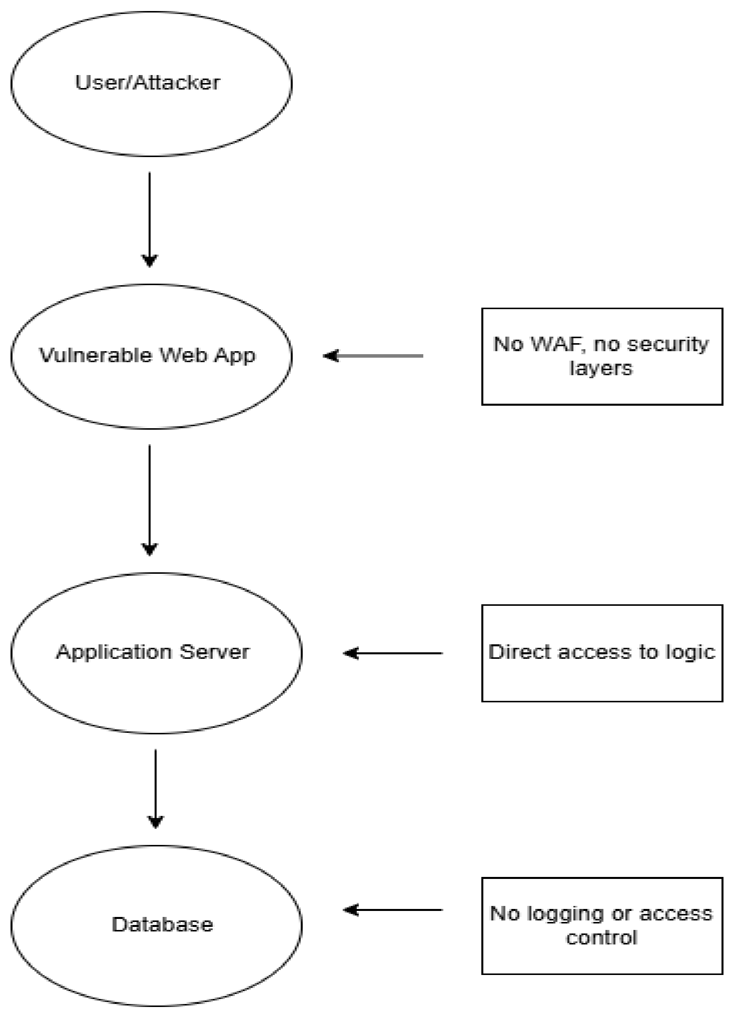


Figure 2.1 Vulnerable Web Application

2.2 Disadvantages of Existing System

While traditional web security mechanisms like WAFs and IDS offer basic protections, they suffer from several key limitations that make them insufficient for modern web applications:

- **Limited Detection of Unknown Attacks:**
  Signature-based systems can only detect **known attack patterns** and **known vulnerabilities**. When new attack vectors are discovered, these systems may be ineffective unless updated with new signatures. This leaves the application exposed to zero-day attacks and previously unknown exploits.
- **Failure to Analyze User Behavior:**
  Traditional security systems do not monitor **user behavior** on the application. Attackers often rely on **anomalous** or **non-standard user interactions** to exploit vulnerabilities. For example, credential stuffing attacks or brute-force login attempts can easily bypass basic security mechanisms. Without the ability to

analyze user actions, such as login times or IP addresses, malicious behavior may go unnoticed.

- **High Rate of False Positives:**
  Many traditional firewalls rely on simple pattern matching, which can result in **false positives** (legitimate requests being flagged as malicious). This creates additional workload for security teams and may lead to decreased productivity, as important legitimate traffic may be incorrectly blocked.
- **No Real-Time Adaptive Learning:**
  Current systems lack the ability to **learn and adapt** to new threats. They do not incorporate **machine learning** or **artificial intelligence (AI)** to dynamically adjust to evolving threats based on data patterns. As a result, traditional systems can only block known threats and cannot identify novel attack methods that deviate from established signatures.
- **Limited Logging and Auditing Capabilities:**
  Existing systems often fail to provide comprehensive **logging and traceability** of user actions. Without sufficient logging of user behavior and system interactions, it is difficult for administrators to perform **post-attack analysis** or determine the extent of a breach. The absence of detailed logs hinders the ability to investigate incidents thoroughly.
- **Performance Overhead:**
  Some traditional security systems can introduce significant performance overhead, especially when processing high volumes of traffic. This can degrade the user experience and make the web application slower, which is especially critical for real-time applications like e-commerce or banking platforms.

2.3 Proposed System

The proposed system, **ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall**, seeks to address the limitations of traditional security mechanisms by integrating **machine learning** for **behavioral analysis** and **anomaly detection**. The proposed system aims to not only protect against known attacks but also dynamically learn from user behavior, adapt to evolving threats, and identify suspicious activities that may indicate potential security breaches.

Key Features of the Proposed System:

- **Behavioral Analysis with Machine Learning:**
  Unlike traditional systems, the proposed firewall utilizes **machine learning** to detect anomalies in user behavior. By tracking user **entry and exit times**, **IP addresses**, **request patterns**, and other behavioral data, the system can identify **deviations** that may indicate an attack. For example, if a user’s behavior suddenly changes — such as rapid login attempts or accessing sensitive areas of the application outside normal usage patterns — the system can flag this as a potential threat.
- **Real-Time Threat Detection:**
  The system continuously monitors and analyzes traffic in real-time. If it detects an anomaly or suspicious activity based on **predefined rules** and **machine learning insights**, it can block malicious requests or alert administrators. This real-time detection capability significantly reduces the chances of successful attacks.

- **Integration with SQLite Database for Logging:**
  All user activity, including **entry/exit times**, **IP addresses**, and interactions with the web application, will be logged into an **SQLite database**. This database provides a valuable resource for post-incident analysis, allowing administrators to investigate suspicious activities and trace the actions of potential attackers.
- **Dynamic and Flexible Rule-Based System:**
  The system still incorporates **rule-based protection** to address well-known attack patterns (e.g., SQL Injection or XSS). However, by combining these static rules with machine learning-driven dynamic detection, the system can adapt and respond to new types of attacks.

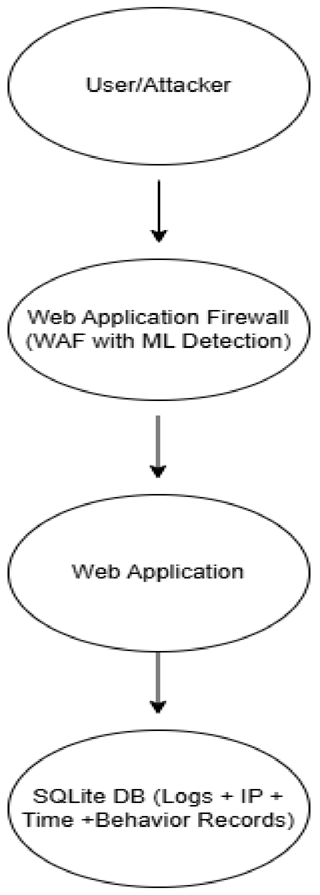

User/Attacker
Web Application Firewall (WAF with ML Detection)
Web Application
SQLite DB (Logs + IP + Time +Behavior Records)

Figure 2.2 ML-Enhanced WAF

2.4 Advantages of Proposed System

The **ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall** offers several advantages over traditional systems:

- **Enhanced Detection of Unknown and Novel Attacks:**
  The use of machine learning enables the system to detect new attack patterns that have not been encountered before, providing protection against **zero-day** attacks and new attack vectors that traditional signature-based systems cannot handle.
- **Behavioral Profiling and Anomaly Detection:**
  By analyzing **user behavior**, the system can identify abnormal patterns and

detect attacks such as **credential stuffing**, **brute-force login attempts**, and **session hijacking**, which traditional systems often miss..

- **Real-Time Threat Mitigation:**
  Unlike traditional systems that often operate on pre-configured rules, the proposed system offers **real-time detection** and **prevention**, enabling immediate response to emerging threats.
- **Comprehensive Logging and Auditing:**
  The logging mechanism provides detailed data for **incident analysis**, helping administrators understand the nature of the threat and respond effectively. This is crucial for forensic analysis and for improving future threat detection strategies.

2.5 Feasibility Studies

Before developing and implementing the **ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall**, it is essential to assess the **feasibility** of the proposed system in terms of **technical, operational, and economic** aspects.

1.Technical Feasibility:

The integration of machine learning into web application security is **technically feasible**. Modern machine learning algorithms, such as **supervised learning** and **anomaly detection models**, can be trained on historical traffic data to identify abnormal behavior. Additionally, incorporating the solution with **existing web servers** and **databases** like SQLite for logging is both feasible and efficient. With the rapid advancement of ML frameworks like TensorFlow, Scikit-learn, and PyTorch, developing the machine learning model is well within current technological capabilities.

2.Operational Feasibility:

The operational feasibility of the system is high. Integrating the **WAF** with machine learning does not require significant changes to the existing web application architecture. The solution is designed to work seamlessly with existing web servers and databases. Additionally, the system provides administrators with easy-to-understand reports and alerts, ensuring smooth integration with the organization's current security monitoring processes.

3.Economic Feasibility:

The cost of developing and deploying the system is relatively low compared to traditional, commercially available WAFs. The machine learning-based system can run on standard hardware or cloud platforms, making it a cost-effective solution for small to medium-sized businesses. Furthermore, the increased **detection accuracy** and **real-time protection** reduce the costs associated with data breaches and downtime, making the system a good investment.

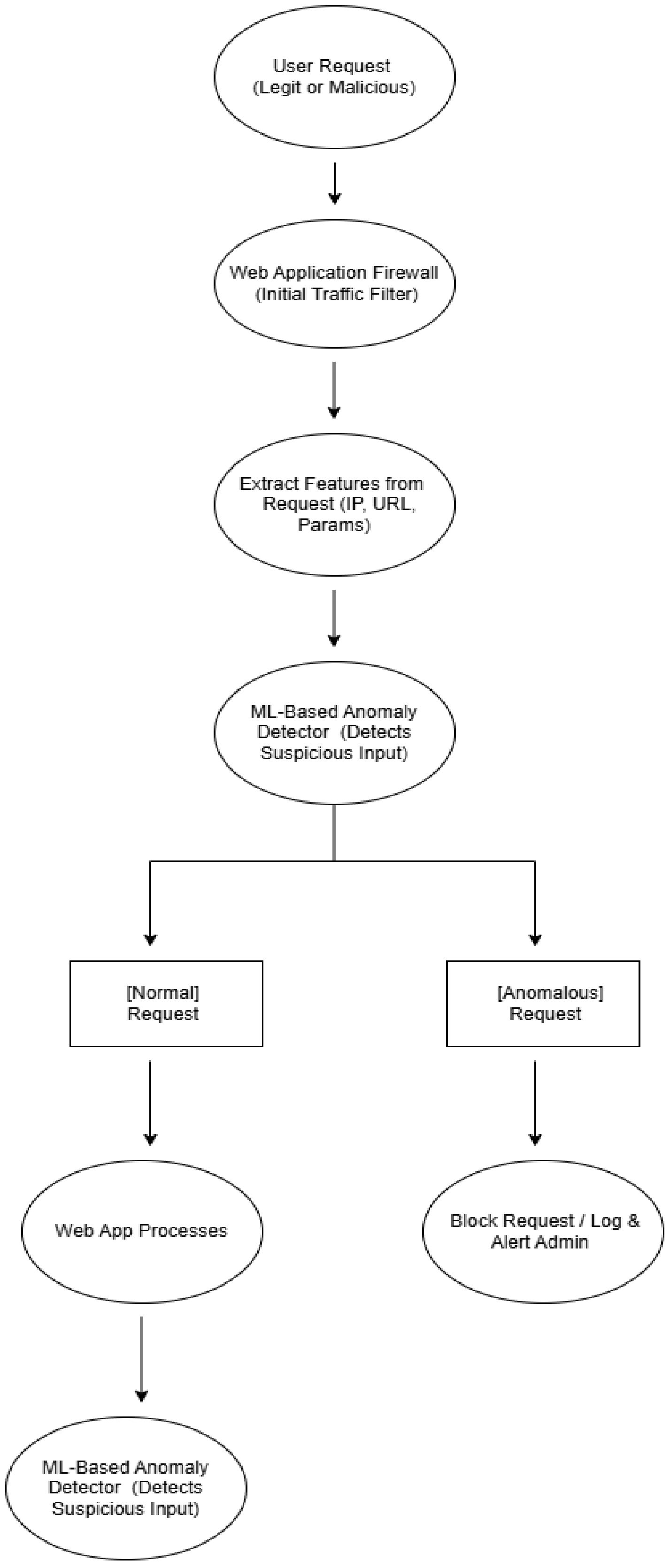

User Request (Legit or Malicious)
Web Application Firewall (Initial Traffic Filter)
Extract Features from Request (IP, URL, Params)
ML-Based Anomaly Detector (Detects Suspicious Input)
[Normal] Request
[Anomalous] Request
Web App Processes
Block Request / Log & Alert Admin
ML-Based Anomaly Detector (Detects Suspicious Input)

Figure 2.3 System Workflow for ML-Enhanced WAF

CHAPTER III

SYSTEM SPECIFICATION

3.1 Hardware Requirements

To ensure the proper functioning of the ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall system, certain minimum and recommended hardware specifications are required. These specifications ensure smooth operation, especially during machine learning model training and real-time traffic monitoring.

- **Minimum Requirements:**

  Processor: Intel Core i5 (8th Gen) or AMD Ryzen 5

  RAM: 8 GB

  Storage: 256 GB SSD

  Network: Ethernet or Wi-Fi adapter

  Display: 1080p monitor (for GUI testing and monitoring)

  Others: Keyboard, mouse, and webcam (optional)

- **Recommended Requirements:**

  Processor: Intel Core i7 or AMD Ryzen 7 (Multi-core)

  RAM: 16 GB or higher

  Storage: 512 GB SSD or higher

  Network: High-speed broadband connection

  GPU: NVIDIA GTX 1650 or above (for faster ML inference)

  Others: Uninterruptible Power Supply (UPS), external storage for logs

3.2 Software Requirements

The software stack includes essential tools, programming environments, libraries, and operating systems used to develop, test, and deploy the firewall and web application.

- **Operating System:**

  Ubuntu 20.04 LTS / Windows 10 or 11 (Developer Mode enabled)

- **Development Tools & Libraries:**

  Programming Language: Python 3.10+

  IDE: Visual Studio Code / PyCharm

  Web Technologies: HTML, CSS, JavaScript, Flask (for web server)

  Database: SQLite (Lightweight embedded database)

  Machine Learning Libraries:

  Scikit-learn

  Pandas, NumPy

  TensorFlow (optional, for advanced ML models)

  Security Tools:

  OWASP ZAP (for vulnerability testing)

  Burp Suite (optional)

  Version Control: Git & GitHub

  Virtual Environment: pipenv or venv

3.3 Project Description

This project aims to enhance web application security using a machine learning-driven Web Application Firewall (WAF). The core idea is to build an intelligent system capable of detecting and mitigating cyberattacks by analyzing user behavior and identifying anomalies.

Key Phases of the Project:

1.Development of Vulnerable Web App:

A basic web application was created intentionally with common security flaws (e.g., SQL injection, XSS, CSRF).

This app was exposed to attacks to simulate real-world vulnerability scenarios.

2.Attack Simulation & Vulnerability Analysis:

Tools like OWASP ZAP were used to launch simulated attacks.

Weaknesses were identified in line with the OWASP Top 10.

3.WAF Implementation with ML:

A WAF was developed to intercept and analyze incoming traffic.

The system logs user IP addresses, login/logout timestamps, and URL access patterns.

4.Machine Learning Integration:

A supervised model was trained using historical traffic data to identify abnormal usage patterns.

Features such as request frequency, time intervals, and IP reputation were fed into the model.

5.Real-Time Detection & Response:

The ML model flags suspicious behavior and blocks access automatically.

Logs are continuously written into SQLite for future analysis and auditing.

3.4 Modules

The system is divided into several interconnected modules, each responsible for a specific function:

Module 1: Vulnerable Web Application

Description: A vulnerable web application is a website or web-based software that contains security flaws or weaknesses. These vulnerabilities can be exploited by attackers to gain unauthorized access, steal data, or perform malicious actions. Common issues include SQL injection, XSS, and insecure authentication.

Features: Login page, data input forms, admin access.

Purpose: Demonstrates how web apps are exploited without protection.

Module 2: Web Application Firewall (WAF)

Description: A Web Application Firewall (WAF) is a security system that monitors and filters HTTP traffic between a web application and the internet. It helps protect against common attacks like SQL injection, cross-site scripting (XSS), and other web exploits.

Functions:

Detect SQL injection, XSS, and other threats.

Enforce security rules (IP blacklisting, rate limiting).

Module 3: Behavior & Anomaly Detection Engine

Description:A Behavior & Anomaly Detection Engine monitors system or user behavior to identify unusual patterns. It helps detect potential security threats by flagging deviations from normal activity.

Components:

Feature extraction: Time-based patterns, IP reputation.

Model: Trained ML classifier (e.g., Random Forest, SVM).

Output: Flagging or blocking of anomalous requests.

Module 4: User Activity Logger

Description: A User Activity Logger records and tracks actions performed by users within a system or application. It helps in auditing, monitoring, and identifying suspicious or unauthorized activities.

Database: SQLite

Logged Data:

Entry and exit timestamps

IP addresses

Accessed URLs

Module 5: Comparative Analysis Interface

Description: A Comparative Analysis Interface allows users to compare data sets, features, or performance metrics side by side. It helps in making informed decisions by highlighting differences and similarities clearly.

Purpose:

Visualize attack frequency

View ML detection logs

Evaluate performance (accuracy, precision, etc.)

CHAPTER IV

SYSTEM DESIGN

4.1 Architecture Design

The architecture of the "ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall" is built on a **multi-layered architecture** consisting of client, application, security, machine learning, and data layers. The client layer interacts with the web interface built using HTML and Flask, which processes requests through the application logic. Every incoming request is filtered by the Web Application Firewall (WAF), which applies both rule-based and ML-based filtering.

The **ML layer** detects anomalous patterns by evaluating user behavior, such as frequency of requests, irregular access times, or unknown IP origins. Upon detection of abnormal behavior, requests are either logged, flagged, or blocked. All logs are recorded in an **SQLite database** for further analysis. The layered design promotes modularity, making the system scalable and maintainable.

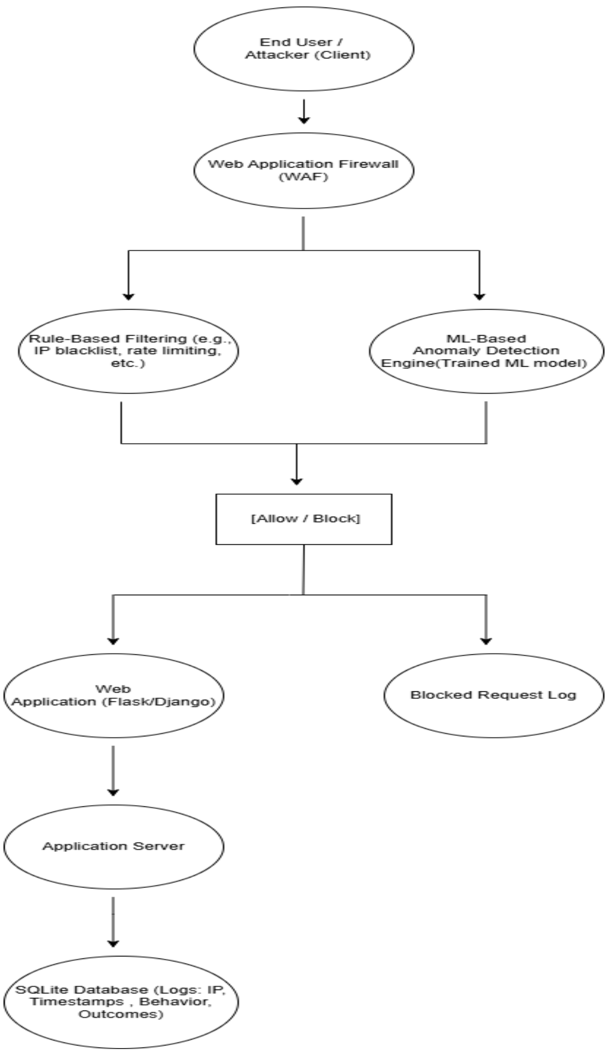

End User / Attacker (Client)
Web Application Firewall (WAF)
Rule-Based Filtering (e.g., IP blacklist, rate limiting, etc.)
ML-Based Anomaly Detection Engine(Trained ML model)
[Allow / Block]
Web Application (Flask/Django)
Blocked Request Log
Application Server
SQLite Database (Logs: IP, Timestamps , Behavior, Outcomes)

Figure 4.1 System Architecture

4.2 Structural Design

The structural design follows a **modular pattern**, breaking down the project into key components:

Web Application Module: Handles UI, routing, and user interaction.

WAF Module: Contains filters, IP tracking, and access control logic.

ML Module: Incorporates the machine learning model for behavior analysis.

Logging Module: Responsible for recording events into the SQLite database.

Each module is loosely coupled and interacts through well-defined APIs. This structural separation ensures that the WAF and ML components can be updated or replaced without affecting the core application. This also facilitates easy debugging, testing, and upgrades.

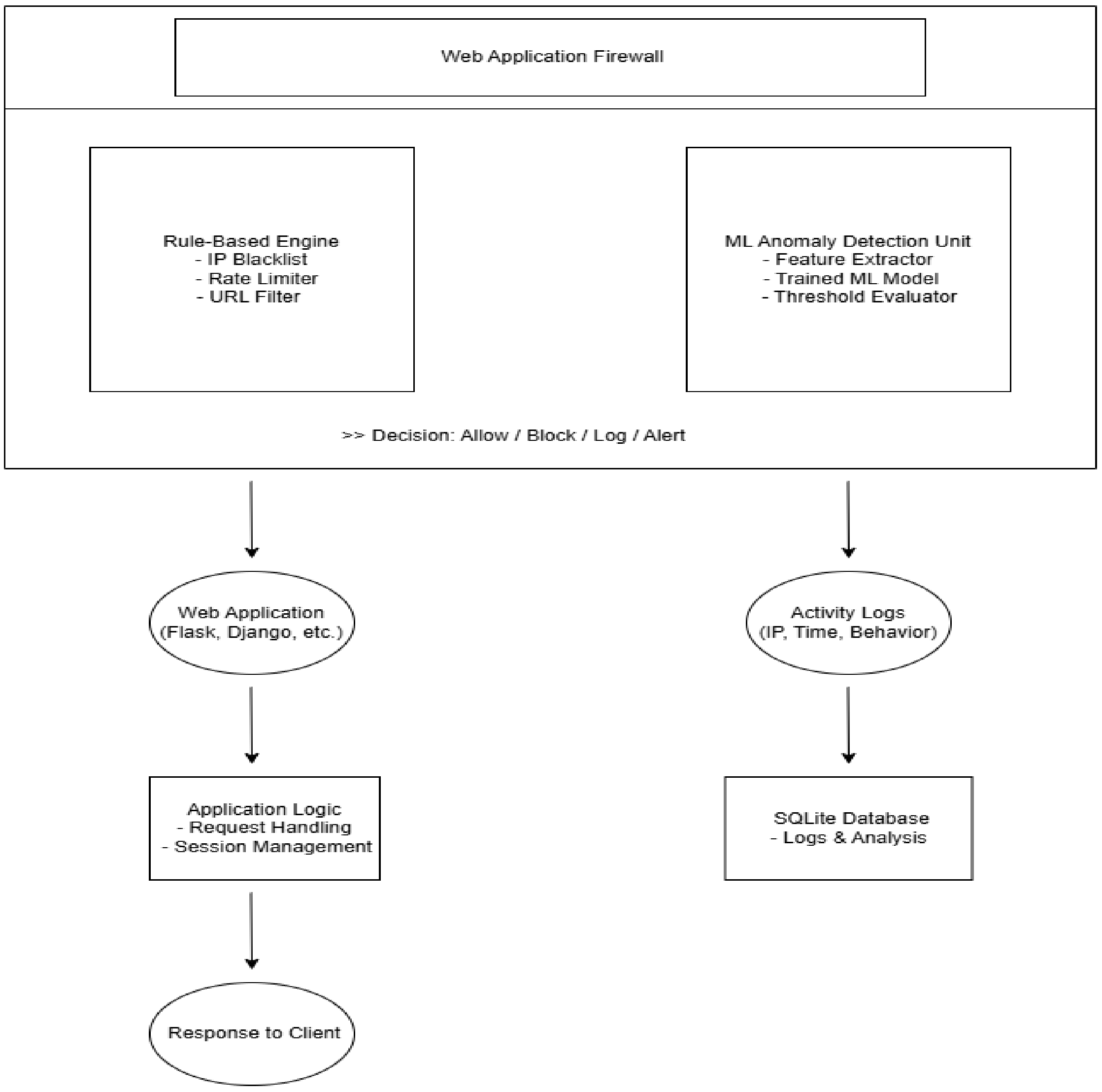


Figure 4.2 Structural Design

4.3 Behavioural Design

Behavioral design captures **system dynamics** and how components interact in response to external events. The key behavior is how the firewall processes requests and applies security logic.

Normal User Flow:

User logs in → Accesses pages → Logs out

Session data logged: IP, time, URLs accessed

Abnormal Flow:

User exceeds request threshold or uses a blacklisted IP

WAF raises a flag

ML module evaluates behavior

If anomaly is confirmed → User is blocked

This flow is represented using **sequence diagrams** and **state transition diagrams** to ensure all user and system interactions are understood and optimized for threat mitigation.

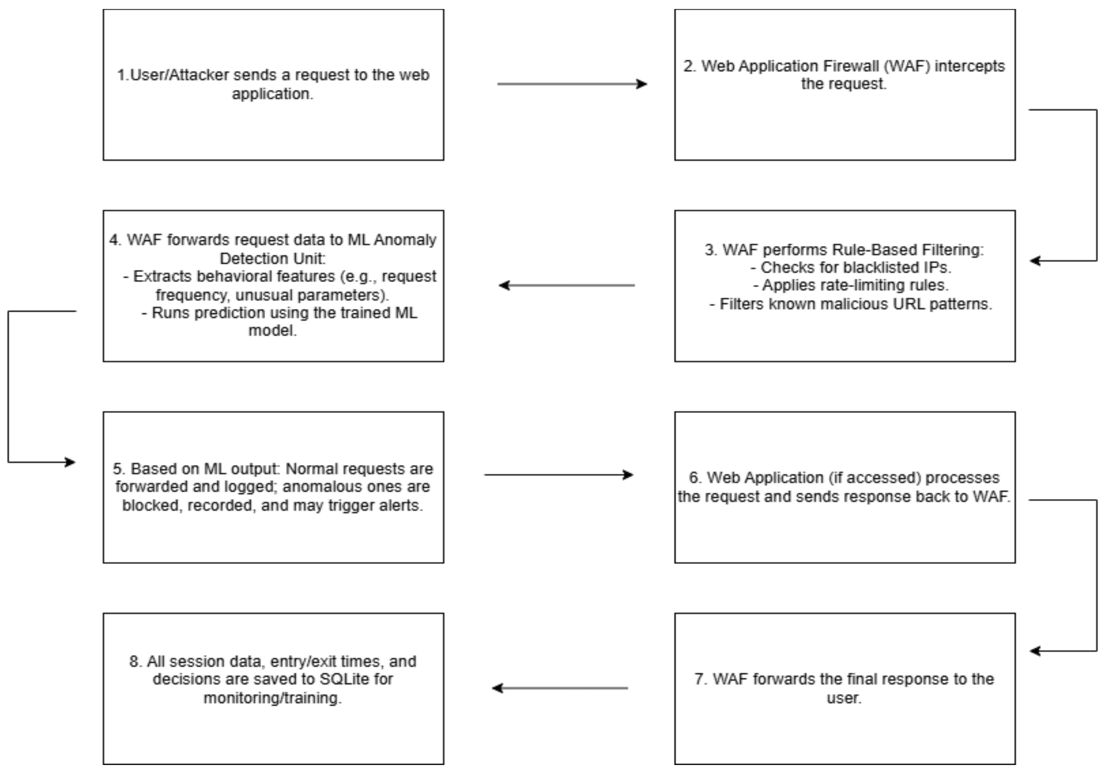

1.User/Attacker sends a request to the web application.
2. Web Application Firewall (WAF) intercepts the request.
3. WAF performs Rule-Based Filtering: - Checks for blacklisted IPs. - Applies rate-limiting rules. - Filters known malicious URL patterns.
4. WAF forwards request data to ML Anomaly Detection Unit: - Extracts behavioral features (e.g., request frequency, unusual parameters). - Runs prediction using the trained ML model.
5. Based on ML output: Normal requests are forwarded and logged; anomalous ones are blocked, recorded, and may trigger alerts.
6. Web Application (if accessed) processes the request and sends response back to WAF.
7. WAF forwards the final response to the user.
8. All session data, entry/exit times, and decisions are saved to SQLite for monitoring/training.

Figure 4.2 Behavioural Design

4.4 Table Design

The system relies on an **SQLite database** to store log and detection data. Two core tables are used:

Table: user_logs

Field	**Type**	**Description**
id	INTEGER	Primary key
user_id	TEXT	Session or login ID
ip_address	TEXT	User IP address
login_time	TEXT	Login timestamp
logout_time	TEXT	Logout timestamp
pages_accessed	TEXT	List of visited URLs

Table: anomaly_records

Field	**Type**	**Description**
id	INTEGER	Primary key
ip_address	TEXT	IP triggering anomaly
detection_time	TEXT	Timestamp of detection
behavior_type	TEXT	Type of anomaly
action_taken	TEXT	Blocked/Flagged

This structured logging helps in retrospective audits and model retraining.

4.5 User Interface Design

The user interface (UI) is designed to be **minimal, responsive, and functional**. Built using HTML, CSS, and Bootstrap, it includes:

User Portal: Login, logout, and web service access

Admin Panel: View logs, suspicious activities, block/unblock IPs

Real-time Alerts: Displays flags triggered by the ML engine

Navigation Bar: Links to different sections like Home, Dashboard, Reports

The UI ensures both usability and security by avoiding exposure of internal system logic or sensitive actions to unauthorized users.

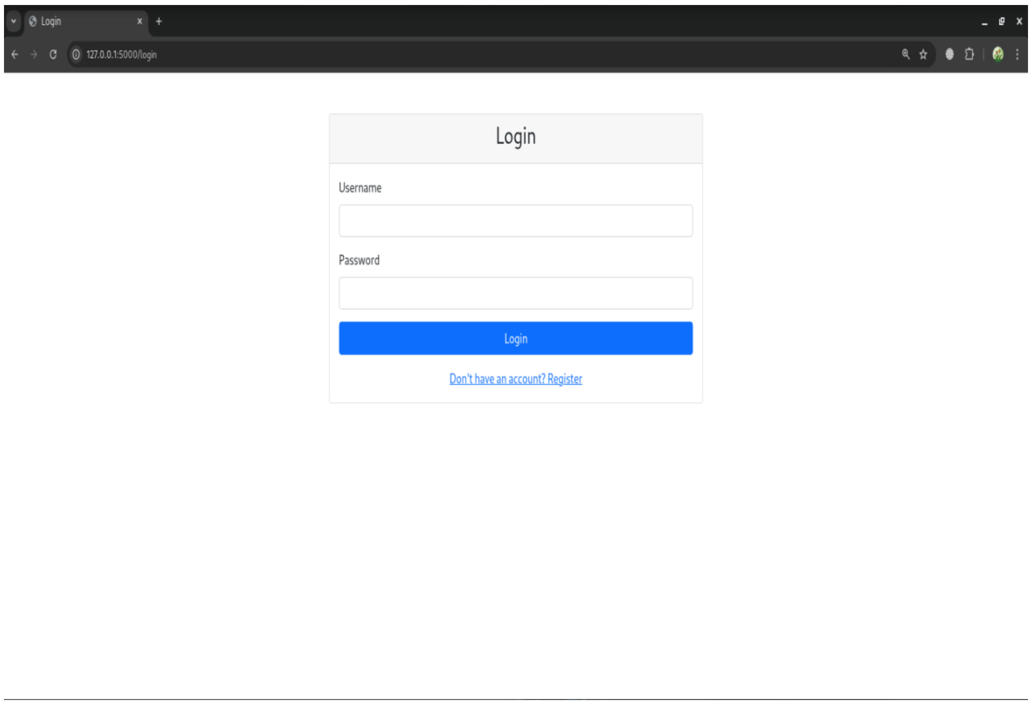
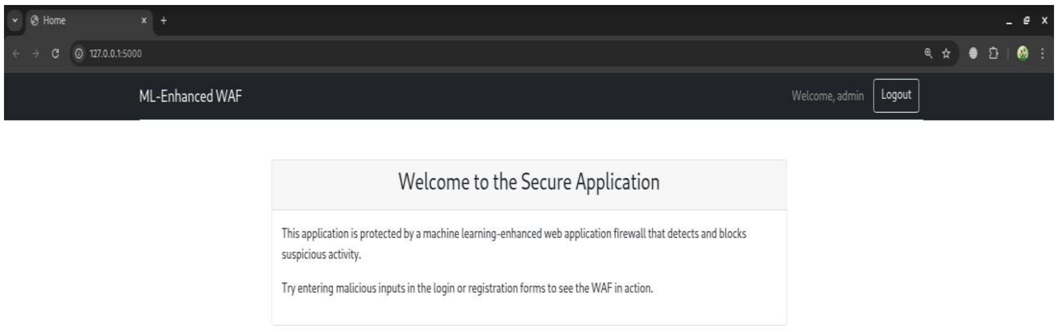


Figure 4.3User Interface Design



Welcome to the Secure Application

This application is protected by a machine learning-enhanced web application firewall that detects and blocks suspicious activity.

Try entering malicious inputs in the login or registration forms to see the WAF in action.

Figure 4.4 User Interface Design

4.6 Deployment Design

The deployment involves a **local server setup** using Flask and SQLite for initial testing and can scale to cloud platforms like AWS or Heroku.

Deployment Steps:

Install dependencies using pip

Launch Flask server on a local port (e.g., http://127.0.0.1:5000)

Connect the ML model and database

Enable logging and WAF filters

Monitor server logs for anomalies

For production, environment variables are secured, HTTPS is enabled, and web traffic is routed through reverse proxies like Nginx.

4.7 Navigation Design

Navigation follows a **logical and user-friendly layout**, ensuring easy movement between system modules.

Navigation Structure:

Home: Overview and project introduction

Login Page: Entry point for users

Dashboard: Displays active sessions, IP logs, alerts

Report Section: Visualizations of threats and anomalies

Admin Tools: Manage blocked IPs, export logs, retrain ML model

Breadcrumb navigation and consistent headers/footers maintain clarity across pages.

4.8 Code Design

Code design emphasizes **readability, modularity, and reusability** using Python and Flask as the main stack. Key principles include:

Modular Functions: Each function handles a specific task (e.g., log request, predict anomaly)

Separation of Concerns: Business logic, ML logic, and UI rendering are separated

Reusable Components: ML prediction, IP filtering, and logging are reusable

Documentation: Inline comments and function docstrings explain the purpose

Security Practices: Input sanitization, SQL query protection, session validation

```
# app/main.py

from flask import Flask, request, redirect, url_for, session, flash, render_template

import os

from datetime import datetime

from werkzeug.security import generate_password_hash, check_password_hash

from config import OWASPConfig, MODEL_PATH

from app.security.detector import AttackDetector

from app.security.logger import log_access, log_anomaly

from app.db.db_utils import (

    insert_access_log,

    insert_anomaly_log,

    create_user_session,

    end_user_session,

    get_user_sessions

)


app = Flask(__name__)

app.secret_key = 'your-secret-key-here'  # Change this in production!


# Initialize with default admin user
```

```
users = {
    'admin': {
        'password': generate_password_hash('SecureAdminPassword123!'),  # Change
this!
        'login_attempts': 0,
        'is_admin': True
    }
}


detector = AttackDetector()


@app.route('/')
def home():
    if 'username' in session:
        user_info = users.get(session['username'], {})
        return render_template('index.html',
                              username=session['username'],
                              is_admin=user_info.get('is_admin', False))
    return redirect(url_for('login'))


@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        ip = request.remote_addr
        username = request.form.get('username')
        password = request.form.get('password')

        user_agent = request.headers.get('User-Agent')


        log_access(ip, username, '/login', 'attempt', user_agent)


        # Check password requirements

        if len(password) < OWASPConfig.PASSWORD_MIN_LENGTH:

            flash('Password does not meet minimum length requirements', 'danger')

            return redirect(url_for('login'))


        # Check brute force protection

        if detector.detect_brute_force(ip, username):

            log_anomaly(ip, username, '/login', 'brute_force', 'Repeated login attempts')

            flash(f'Too many login attempts. Please try again after 
{OWASPConfig.LOCKOUT_TIME_MINUTES} minutes.', 'danger')

            return redirect(url_for('login'))


        # Authentication logic

        if username in users and check_password_hash(users[username]['password'], 
password):

            session['username'] = username

            users[username]['login_attempts'] = 0  # Reset on successful login


            # Create session record

            create_user_session(username, ip)
```

```
            log_access(ip, username, '/login', 'success', user_agent)

            return redirect(url_for('home'))

        else:

            # Track failed attempts

            if username in users:

                users[username]['login_attempts'] += 1

            log_access(ip, username, '/login', 'failed', user_agent)

            flash('Invalid username or password', 'danger')



    return render_template('login.html')



@app.route('/register', methods=['GET', 'POST'])

def register():

    if request.method == 'POST':

        ip = request.remote_addr

        username = request.form.get('username')

        password = request.form.get('password')

        user_agent = request.headers.get('User-Agent')



        # Password complexity check

        if len(password) < OWASPConfig.PASSWORD_MIN_LENGTH:

            flash(f'Password must be at least
{OWASPConfig.PASSWORD_MIN_LENGTH} characters', 'danger')

            return redirect(url_for('register'))
        if username not in users:
            users[username] = {
                'password': generate_password_hash(password),
                'login_attempts': 0,
                'is_admin': False
            }
            log_access(ip, username, '/register', 'success', user_agent)
            flash('Registration successful! Please login.', 'success')
            return redirect(url_for('login'))
        else:
            log_access(ip, username, '/register', 'failed', user_agent)
            flash('Username already exists', 'danger')

    return render_template('register.html')


@app.route('/logout')
def logout():
    if 'username' in session:
        username = session['username']
        ip = request.remote_addr
        user_agent = request.headers.get('User-Agent')

        # End the user session
        end_user_session(username)
        log_access(ip, username, '/logout', 'success', user_agent)
        session.pop('username', None)
    return redirect(url_for('login'))


@app.route('/session-history')
def session_history():
    if 'username' not in session or not users.get(session['username'], {}).get('is_admin'):
        flash('Admin access required', 'danger')
        return redirect(url_for('login'))


    username_filter = request.args.get('username')
    sessions = get_user_sessions(username_filter)
    return render_template('session_history.html', sessions=sessions)


if __name__ == '__main__':
    from app.db.db_utils import init_db
    # Initialize database and create tables
    os.makedirs('logs', exist_ok=True)
    init_db()


    # Apply security headers
    @app.after_request
    def apply_security_headers(response):
        for header, value in OWASPConfig.SECURITY_HEADERS.items():
            response.headers[header] = value
        return response


    app.run(debug=True, host='0.0.0.0', port=5000)
```

CHAPTER V

SYSTEM IMPLEMENTATION

5.1 Framework of Demo Implementation

The demo implementation of the ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall uses a lightweight, modular framework designed to simulate real-world web application threats and responses. The core of the implementation is built using the Flask web framework in Python, enabling rapid development and flexible routing for both user and admin interfaces.

The system includes a custom Web Application Firewall (WAF) that intercepts incoming HTTP requests. This firewall applies rule-based filtering (e.g., blocking known malicious IPs) and sends behavior metrics to a machine learning model, developed using Scikit-learn, to detect anomalies like abnormal access frequency or suspicious activity patterns.

User sessions, including IP addresses and access logs, are stored in a local SQLite database, which enables real-time logging and monitoring. The front-end interface is built with HTML/CSS for simplicity, featuring user and admin views to track activities and manage threat responses.

This framework successfully demonstrates a hybrid security approach, combining static filtering and dynamic, intelligent detection to prevent cyberattacks in real-time.

CHAPTER VI

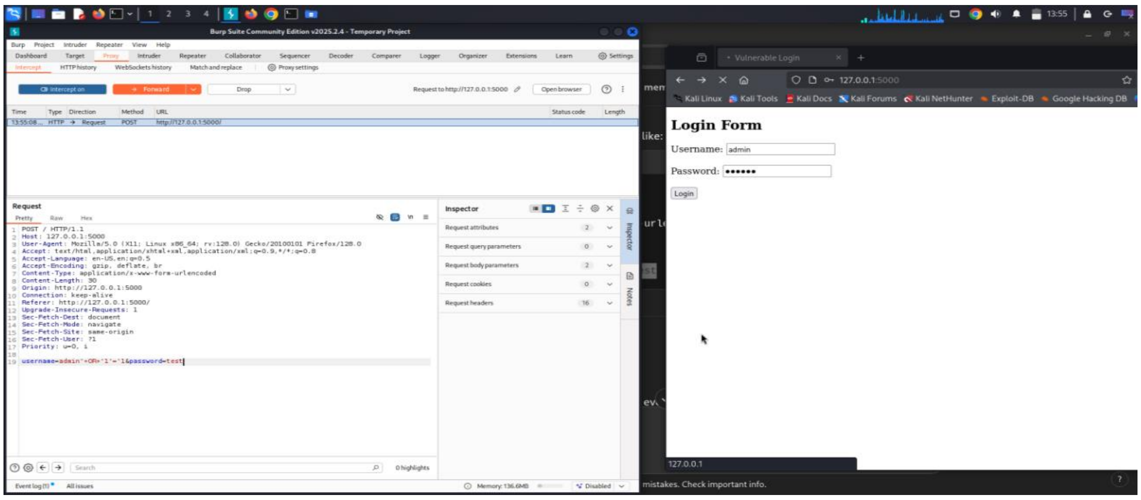
TEST PLAN

6.1 Test Cases and Test Reports

To validate the functionality and security effectiveness of the proposed system, several test cases were designed targeting key components—user authentication, request monitoring, anomaly detection, and IP blocking. The objective was to ensure that the WAF and ML model respond correctly to both normal and malicious behaviors.

Table: Test Cases and Test Reports

Test Case	Expected Result	Status
Normal user login	Access granted, session logged	Passed
Excessive requests from IP	Detected as anomaly, IP flagged	Passed
SQL injection attempt	Blocked by WAF rules	Passed
Unknown IP access	Behavior analyzed, blocked if anomalous	Passed
Admin views flagged logs	All logs correctly displayed	Passed

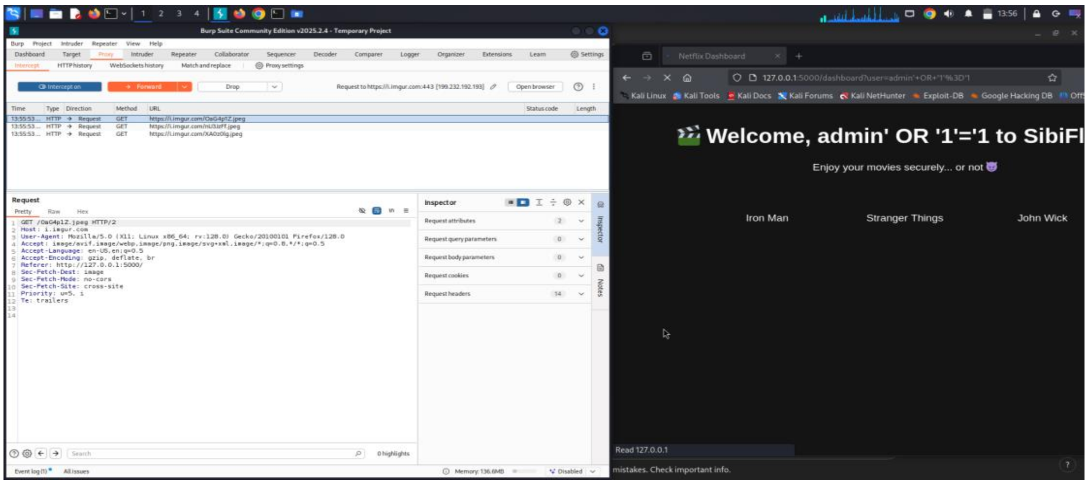


```
Time        Type  Direction  Method  URL
13:55:08 ... HTTP  → Request  POST    http://127.0.0.1:5000/
```

```
POST / HTTP/1.1
Host: 127.0.0.1:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Origin: http://127.0.0.1:5000
Connection: keep-alive
Referer: http://127.0.0.1:5000/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i

username=admin'+OR+'1'='1&password=test
```

Figure 6.1 Vulnerable Web Application

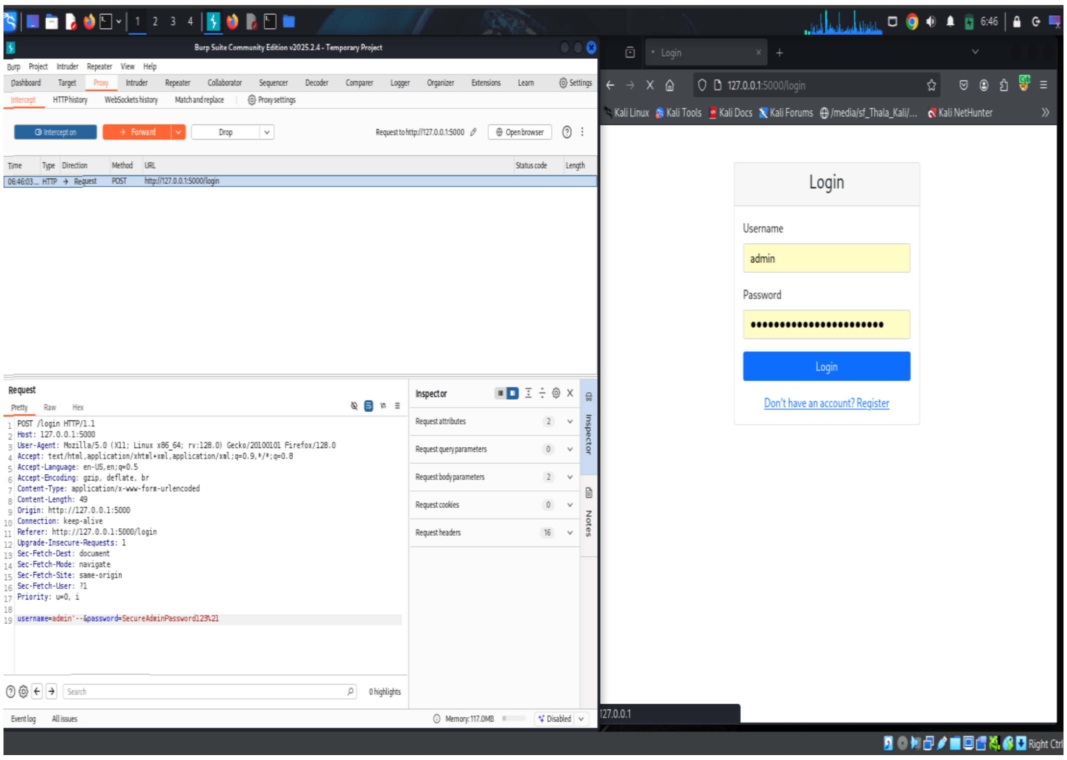


```
GET /OaG4p1Z.jpeg HTTP/2
Host: i.imgur.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://127.0.0.1:5000/
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: cross-site
Priority: u=5, i
Te: trailers
```

Welcome, admin' OR '1'='1 to SibiFl

Enjoy your movies securely... or not

Figure 6.2 Vulnerable Web Application



```
POST /login HTTP/1.1
Host: 127.0.0.1:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 49
Origin: http://127.0.0.1:5000
Connection: keep-alive
Referer: http://127.0.0.1:5000/login
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i

username=admin'--&password=SecureAdminPassword123%21
```

Login

Username: admin

Password: ••••••••••••••••••••••

Figure 6.3 ML-Enhanced WAF

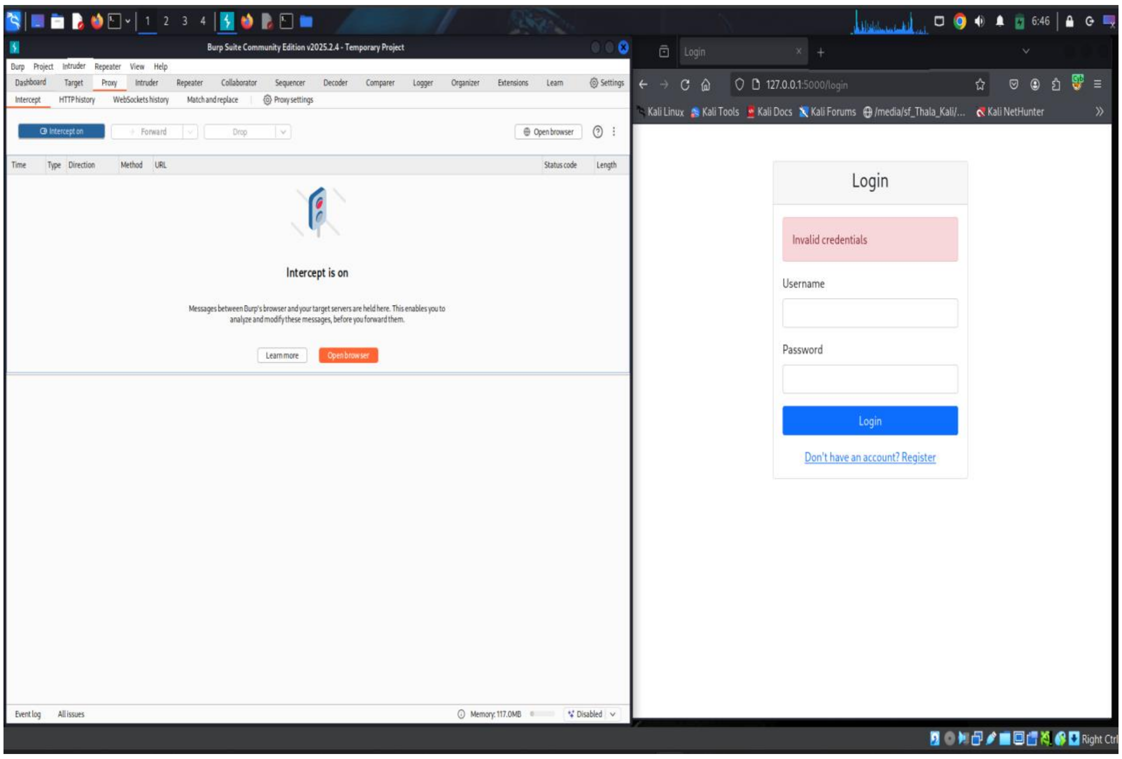


Figure 6.4 ML-Enhanced WAF

CHAPTER VII

CONCLUSION AND FUTURE ENHANCEMENT

7.1 Conclusion

This project successfully demonstrates the design and implementation of a machine learning–enhanced Web Application Firewall (WAF) that provides proactive protection against cyber threats. By combining rule-based filtering with behavior-based anomaly detection, the system addresses common vulnerabilities highlighted in the OWASP Top 10. Through comprehensive logging, IP tracking, and real-time ML analysis, the firewall intelligently identifies and blocks suspicious activity. The comparative analysis between vulnerable and protected setups clearly validates the system's effectiveness. Overall, the project offers a scalable and intelligent solution for improving the security posture of web applications.

7.2 Future Enhancement inckude

While the current system performs effectively in detecting and blocking threats, future improvements can significantly enhance its scalability and adaptability. Potential enhancements include:

- **Integration with cloud platforms** for large-scale deployment and real-time threat updates.
- **Advanced ML models**, such as deep learning for more accurate behavioral analysis.
- **User role-based access control** and authentication features to tighten security.
- **Visual dashboards** using tools like Grafana for real-time monitoring and reporting.
- **Automated retraining** of the ML model using newly logged data for continuous learning.

These enhancements would elevate the system from a functional prototype to an enterprise-grade web security solution.

REFERENCE

1. Deep Learning for Anomaly Detection in Log Data: A Survey This survey explores deep learning techniques for analyzing log data to detect anomalies, highlighting the challenges and advancements in the field.

2. Federated Learning for Intrusion Detection System: Concepts, Challenges, and Future Directions Discusses the application of federated learning in intrusion detection systems, emphasizing privacy-preserving decentralized model training.

3. Adaptively Detecting Malicious Queries in Web Attacks Proposes AMODS, an adaptive system that periodically updates detection models to identify new web attacks, enhancing WAF capabilities.

4. A Survey on Anomaly Detection for Technical Systems using LSTM Networks Reviews the use of Long Short-Term Memory (LSTM) networks for anomaly detection in technical systems, applicable to WAFs for detecting temporal anomalies.

5. Leveraging Deep Neural Networks for Anomaly-Based Web Application Firewall Introduces a WAF model utilizing deep neural networks for anomaly detection, demonstrating improved accuracy in identifying web attacks.

6. Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks Explores the integration of deep learning techniques in WAFs, focusing on the detection of various web attacks and performance evaluation.

7. Artificial Intelligence Web Application Firewall for Advanced Detection of Web Injection Attacks Presents an AI-driven WAF that employs machine learning algorithms to detect advanced web injection attacks, enhancing security measures.

8. Comparative Analysis of Anomaly Detection Approaches in Firewall Logs Analyzes different anomaly detection methods in firewall logs, integrating lightweight synthesis of security logs and artificially generated attack detection.

9. Anomaly-Based Web Attack Detection Proposes a deep learning approach for detecting anomalous web requests, utilizing Recurrent Neural Networks (RNNs) to learn normal request patterns.

10. Machine Learning and Natural Language Processing Based Web Application Firewall for Mitigating Cyber Attacks in Cloud Develops a WAF that combines machine learning and natural language processing to mitigate cyber attacks in cloud environments.

11. Deep Learning for Vulnerability and Attack Detection on Web Applications: A

Systematic Literature Review Conducts a systematic review of deep learning-based web attack detection methods, categorizing studies and identifying research gaps.

12. Anomaly Detection Using Generative Adversarial Networks on Firewall Log Message DataInvestigates the use of Generative Adversarial Networks (GANs) for anomaly detection in firewall log data, proposing a novel approach for identifying security threats.

13. Ontology for Attack Detection: An Intelligent Approach to Web Application Security Proposes an ontology-based approach for attack detection in web applications, enhancing the intelligence of WAFs.

14. Anomaly Detection Method to Detect Web Attacks Using Stacked Auto-Encoder Introduces a method employing stacked auto-encoders for detecting web attacks, demonstrating its effectiveness in anomaly detection.

15. Extreme Learning Machines for Web Layer Anomaly Detection Explores the application of extreme learning machines in detecting anomalies at the web layer, offering insights into their effectiveness.

16. Detection of Web-Based Attacks Through Markovian Protocol Parsing Discusses the detection of web-based attacks using Markovian protocol parsing, contributing to the understanding of attack patterns.

17. Detection and Mitigation of Web Services Attacks Using Markov Model Investigates the use of Markov models in detecting and mitigating web services attacks, providing a statistical approach to security.

18. An On-Line Learning Statistical Model to Detect Malicious Web Requests Proposes an online learning statistical model for detecting malicious web requests, enhancing real-time threat detection.

19. Anomaly Detection Method Based on Multi-Models to Detect Web Attacks Develops a multi-model-based anomaly detection method for web attacks, improving detection accuracy.

20. Semantic Security Against Web Application Attacks Explores semantic security measures to protect web applications from attacks, focusing on the understanding of attack semantics

21.Artificial Intelligence-Based Web Application Firewall for Advanced Detection of Web Injection Attacks Presents an AI-based WAF for advanced detection of web injection attacks, emphasizing the importance of adaptive learning.

22.ARCADE: Adversarially Regularized Autoencoder Focus: Detects anomalies in network traffic using an autoencoder. Use in WAF: Fast, unsupervised detection of unusual web requests

23. Anomal-E: GNN-based Intrusion Detection Focus: Uses Graph Neural Networks for detecting intrusions.

Use in WAF: Captures complex behavior patterns in traffic.

24. DeepAID: Explainable Deep Anomaly Detection Focus: Improves transparency of anomaly detection models. Use in WAF: Helps understand why traffic is flagged.

25. Outlier Exposure for Deep Anomaly Detection Focus: Trains models using outliers from other domains.Use in WAF: Better detection of novel, unknown attacks.

26. Auto-Threshold Deep SVDD (ATDSVDD) Focus: Deep learning + automatic threshold tuning. Use in WAF: Reduces false positives in anomaly detection