Web Application Firewall (WAF) Using Python and Flask

A Comprehensive Project Report

Table of Contents

---

1. Abstract

Web applications are frequently targeted by cyber threats such as **SQL Injection, Cross-Site Scripting (XSS), and Denial of Service (DoS) attacks**. A **Web Application Firewall (WAF)** is essential to protect web applications by filtering and monitoring HTTP traffic.

This project develops a **lightweight WAF using Python and Flask**, capable of **detecting and preventing malicious attacks**. The WAF employs **regular expressions (Regex) for attack detection**, **IP blocking mechanisms**, and **rate-limiting techniques** to prevent abuse. The solution can be deployed **locally or on cloud platforms like Heroku**, making it scalable and effective in real-world scenarios.

---

2. Introduction

2.1 Background

Web applications form the backbone of modern digital services. However, their **increased reliance on user input and database interactions** makes them vulnerable to cyber threats. Attackers exploit these vulnerabilities to steal data, disrupt services, or gain unauthorized access.

A Web Application Firewall (WAF) provides **an additional layer of security** by monitoring and filtering HTTP traffic. Unlike traditional firewalls, which **operate at the network level**, a WAF focuses on **application-layer security**, blocking **SQL Injection, XSS, and DoS attacks** before they reach the web application.

2.2 Objectives

The primary objectives of this project are:

1. **Developing a Python-based WAF** to enhance web security.
2. **Detecting and blocking common web attacks** like SQL Injection, XSS, and IP-based threats.
3. **Implementing logging and monitoring** to track potential threats.
4. **Deploying the WAF on a cloud platform** for practical application.

2.3 Scope of the Project

This project focuses on securing **Flask-based web applications** from common security threats. While the WAF provides fundamental protection, **it does not replace commercial-grade WAFs** with advanced AI-based detection systems.

---

3. Literature Review

3.1 Web Security Threats

Web applications are vulnerable to various attacks:

- **SQL Injection**: Attackers inject malicious SQL queries to access or modify databases.
- **XSS (Cross-Site Scripting)**: Inserting malicious scripts into web pages to steal session data.
- **DoS (Denial of Service) Attacks**: Overloading a server to disrupt services.

3.2 Existing Security Solutions

Traditional security solutions include:

- **Network Firewalls**: Protect against unauthorized access but do not filter HTTP requests.
- **Intrusion Detection Systems (IDS)**: Detect attacks but do not block them automatically.
- **Commercial WAFs**: Costly and complex to configure for small applications.

3.3 Importance of a WAF

A WAF is **essential for web security**, as it operates at the application layer, filtering **malicious HTTP requests** in real time.

---

4. Methodology

4.1 System Architecture

The **WAF intercepts HTTP requests** before they reach the Flask web application, analyzing request data for **SQL Injection, XSS, and IP-based threats**.

4.2 Technologies Used

- **Programming Language**: Python
- **Framework**: Flask
- **Security Techniques**: Regex filtering, IP blocking, rate limiting
- **Deployment**: Heroku

4.3 Implementation Steps

1. **Set up a Flask-based web application.**
2. **Develop security filters** for detecting malicious requests.
3. **Integrate the WAF with the Flask app.**
4. **Test the firewall with simulated attacks.**
5. **Deploy the application.**

---

5. Implementation

5.1 Setting Up the Development Environment

Install dependencies:

pip install Flask flask-limiter

5.2 Writing the WAF Logic

waf.py (Security Mechanisms)

```
import re

SQL_INJECTION_PATTERN = r"(\bselect\b|\binsert\b|\bunion\b|\b--\b|\b;\b|\bdrop\b)"
XSS_PATTERN = r"<script.*?>.*?</script>"
BLOCKED_IPS = ["192.168.1.100"]

def detect_sql_injection(input_data):
    return bool(re.search(SQL_INJECTION_PATTERN, input_data, re.IGNORECASE))

def detect_xss(input_data):
    return bool(re.search(XSS_PATTERN, input_data, re.IGNORECASE))

def block_ip(ip):
    return ip in BLOCKED_IPS
```

app.py (Main Web Application)

```
from flask import Flask, request, abort
from waf import detect_sql_injection, detect_xss, block_ip
from flask_limiter import Limiter

app = Flask(__name__)
limiter = Limiter(app, key_func=lambda: request.remote_addr)

@app.before_request
def check_security():
    user_ip = request.remote_addr
    if block_ip(user_ip):
        abort(403, "Blocked IP")
    for key in request.form.keys():
        if detect_sql_injection(request.form[key]) or detect_xss(request.form[key]):
            abort(400, "Malicious request detected")

@app.route('/')
def home():
    return "Welcome to the secure web app!"

if __name__ == '__main__':
    app.run(debug=True)
```

5.3 Testing the WAF

Simulated **SQL Injection test**:

curl -X POST -d "user_input=' OR 1=1 --" http://127.0.0.1:5000/

---

6. Deployment

6.1 Hosting on Local Server

Run Flask:

python app.py

6.2 Deploying on Heroku

1. **Create a Procfile**:
2. web: gunicorn app:app
3. **Push to Heroku**:
4. git init
5. git add .
6. git commit -m "Deploying WAF"
7. heroku create my-waf-app
8. git push heroku main

---

7. Results and Discussion

7.1 Performance Analysis

- Successfully detected and blocked **SQL Injection and XSS attacks**.
- **Rate-limiting** prevented DoS-style attacks.

7.2 Limitations

- Limited to **basic regex-based filtering**.
- Can be bypassed by **advanced attack techniques**.

7.3 Future Enhancements

- **Machine Learning-based threat detection**.
- **Integration with cloud-based monitoring services**.

---

8. Conclusion

The **Flask-based WAF** effectively protects web applications from SQL Injection and XSS attacks. **Deploying the WAF on Heroku** enables scalable and real-world implementation.

---

9. References

1. OWASP Web Security Guide
2. Flask Documentation
3. Heroku Deployment Guide