Web Application Firewall(WAF)

For Malicious URL Detection project

Submitted in partial fulfillment of the Internship Training at

Eagle-hitech softclou private limited

Submitted by:

Vimal Kumar P (510422149058)

Shamsudeen M S (510422149045)

Dhanush N (510422149010)

Akash S (510422149006)

Saran Raj P (510422149042)

Project Guide at Company:

Department of CSE(Cyber Security)

ARUNAI ENGINEERING COLLEGE (Autonomous institution)

Mathur,Tiruvannamalai.

Academic year(2022-2026)

1. Introduction

In today's digital era, the internet has become an essential part of daily life, enabling communication, commerce, education, and countless services. However, this increased connectivity has also led to the rise of cyber threats such as phishing attacks, malware distribution, and malicious URLs. These threats pose serious risks to personal data, organizational security, and national infrastructure. One of the most commonly exploited vectors for cyber-attacks is malicious URLs, which are specifically crafted to deceive users or compromise their systems.

To counter such threats, organizations implement various cybersecurity tools, among which the Web Application Firewall (WAF) plays a vital role. A WAF is a protective layer that filters, monitors, and blocks malicious traffic to and from a web application. It sits between the end-user and the web server, analyzing incoming HTTP/HTTPS requests to detect and mitigate suspicious behavior in real time.

This project, titled "Web Application Firewall (WAF) for Malicious URL Detection," was undertaken during our internship at [Company Name] with the objective of building a simplified, yet functional prototype of a WAF system that focuses on the detection of harmful URLs. This system uses pattern-matching techniques to identify potentially dangerous URLs based on common phishing and malware indicators.

This project not only deepens our understanding of web application security but also provides hands-on experience with real-world cybersecurity practices. Through this internship, we had the opportunity to explore key areas

of secure web development, HTTP request filtering, and log monitoring—skills that are critical in the ever-evolving field of cybersecurity.

2.Objectives:

Aim of the Project:

The primary aim of this project is to **design and develop a basic Web Application Firewall (WAF)** that can detect and block malicious URLs using pattern-matching techniques, helping to protect web applications and users from common cybersecurity threats like phishing, malware, and unauthorized access.

Goals:

1. **To understand the functioning of Web Application Firewalls (WAFs)** and their role in cybersecurity.
2. **To build a lightweight and functional WAF model** using Python and Flask, suitable for learning and demonstration purposes.
3. **To detect URLs with malicious intent** based on commonly known keywords and patterns using regular expressions.
4. **To provide real-time feedback to users** about whether a URL is safe or suspicious.
5. **To store logs and maintain a record** of scanned URLs for monitoring and analysis.
6. **To provide an admin module** for managing blocked URLs and resetting logs.

7. **To simulate the experience of working in a real-time security-based development project** during the internship.

Project Outcomes:

- A functional **Flask-based web application** that scans and detects malicious URLs using regex pattern matching.
- A **user interface** where users can input URLs and receive a safe/unsafe result.
- A **backend system** that logs results and maintains a list of blocked URLs.
- An **admin dashboard** with options to view blocked URLs and reset logs.
- Improved understanding of **cybersecurity threats**, web security practices, and Python-based development.
- Practical exposure to **client-server architecture, Flask web framework, and URL filtering logic**.
- Development of **teamwork, problem-solving, and technical documentation skills** as part of the internship.

3.Tools and Technologies Used:

During the development of the Web Application Firewall (WAF) project, a variety of tools, programming languages, frameworks, and technologies were utilized to ensure efficient and effective implementation. The following is a comprehensive list.

1) Programming Language:

Python

- Used for building the backend logic of the application.
- Enables easy integration with libraries and frameworks for web development and pattern matching (via re module).

2) Web Framework:

Flask (Python Web Framework)

- Lightweight and easy-to-use micro web framework.
- Used for building web routes, handling requests, and rendering templates.

3) Frontend Technologies:

HTML5

- Used for structuring the web pages and user input forms.

CSS3

- Used for styling the web interface to make it user-friendly and responsive.

4) Logging and File Handling:

Python's built-in datetime and file I/O features were used to:

- Create logs of scanned URLs.
- Record the results (safe/malicious) in a .txt log file.

5)Development Environment:

Visual Studio Code (VS Code)

- Main code editor used for writing and debugging the project.

Command Line Interface (CMD / Terminal)

- Used to run the Flask server and manage project files.

6) Version Control (Optional):

Git & GitHub

- Used to track changes, collaborate as a team, and store the project in a shared repository (if applicable during the internship).

8) Hosting & Deployment (Optional for Advanced Phase):

Localhost (Development Server)

- Flask was run in debug mode on localhost for testing and demonstration purposes.

4.Working Principle:

The Web Application Firewall (WAF) project operates as a **URL scanner and filtering system,** designed to detect and block potentially **malicious or phishing links** before they reach end users. The core working principle is based on **pattern matching, blacklisting techniques,** and **request inspection** to ensure web security.

1) User Input (URL Submission)

- The user accesses a web interface developed using **HTML/CSS**.
- They enter the URL they wish to scan into the form field.
- Upon submission, the data is sent to the backend server using **HTTP POST**.

2) Pattern Matching Engine

- The URL is processed in the backend by a **Python Flask server**.
- A function named is_suspicious(url) checks the submitted URL against a **predefined list of malicious patterns** using **regular expressions (Regex)**.
- These patterns include keywords and fragments such as:
    - phishing, bit.ly, login-stealer, password-reset, .exe, update-info, etc.
- If any of the patterns are matched, the URL is flagged as **malicious**.

3) Decision Logic

- Based on the result from pattern matching:
    - If the URL is **safe**, the response Safe URL is returned to the user.
    - If the URL is **malicious**, the response Malicious URL Detected! is displayed.
- The decision is made purely on string pattern checks and does not involve accessing or loading the actual website, making it fast and secure.

4) Logging and Blocking

- Every scanned URL is recorded in a **log file** (waf_log.txt) along with its status and timestamp.
- If the URL is malicious, it is also added to a **blocklist** (blocked_urls[]) for administrative viewing.
- Admin users can:
    - View blocked URLs via the /admin/blocked_ips route.
    - Reset the logs and blocklist via the /admin/reset_logs route.

5) Output Display

- The frontend displays the result (safe or malicious) on the same page using Flask's **render_template** functionality.
- Optionally, the system can be extended to redirect users or alert security teams.

5. Code Implementation:

Python (flask):

```
from flask import Flask, request, render_template, jsonify
import re
import os
import datetime

app = Flask(_name_)

LOG_FILE = "waf_log.txt"
blocked_urls = []

# Check if URL is suspicious (simple pattern matching for demo)
def is_suspicious(url):
    patterns = [r"free[-.]gift", r"malware", r"phishing", r"@", r"bit.ly", r"login[-.]stealer", r"\\.exe", r"\\.zip", r"unlock[-.]code", r"login[-.]form", r"secure[-.]login", r"bank[-.]account", r"credit[-.]card", r"password[-.]reset", r"verify[-.]account", r"update[-.]info", r"click[-.]here", r"urgent[-.]action", r"confirm[-.]details", r"account[-.]verification", r"security[-.]alert", r"alert[-.]notification",
```

```
r"claim[-.]reward", r"win[-.]prize", r"limited[-.]offer", r"exclusive[-.]deal",
r"urgent[-.]message", r"invoice[-.]payment", r"tax[-.]refund", r"shipping[-
.]confirmation", r"order[-.]status", r"subscription[-.]renewal", r"account[-
.]suspension", r"service[-.]interruption", r"technical[-.]support", r"customer[-
.]service", r"feedback[-.]survey", r"product[-.]review", r"recovery[-.]link",
r"reset[-.]password", r"account[-.]recovery", r"security[-.]question", r"identity[-
.]verification", r"login[-.]attempt", r"unauthorized[-.]access", r"data[-.]breach",
r"phishing[-.]attempt", r"scam[-.]alert", r"pass",

        r"authentication", r"support", r"paypal", r"webmail", r"banking",
r"login", r"redirect",  r"account", r"secure", r"verify", r"update", r"confirm",
r"alert", r"notification", r"reward", r"prize", r"offer", r"deal", r"message",
r"inbox", r"inbox[-_.]message", r"setup", r"hack"]
  for pattern in patterns:
    if re.search(pattern, url, re.IGNORECASE):
      return True
  return False


def log_url(url, result):
  with open(LOG_FILE, "a") as f:
    f.write(f"{datetime.datetime.now()} - {url} - {result}\n")


@app.route("/", methods=["GET", "POST"])
def index():
  result = ""
  if request.method == "POST":
    url = request.form.get("url")
    if is_suspicious(url):
      result = "❌ Malicious URL Detected!"
```

```
            blocked_urls.append(url)
            log_url(url, "Blocked")
        else:
            result = " ✅ Safe URL"
            log_url(url, "Safe")
    return render_template("index.html", result=result)

@app.route("/admin/blocked_ips")
def view_blocked():
    return jsonify(blocked_urls)

@app.route("/admin/reset_logs")
def reset_logs():
    open(LOG_FILE, "w").close()
    blocked_urls.clear()
    return " ✅ Logs and blocked URLs have been reset."

if _name_ == "_main_":
    app.run(debug=True)
```

Frontend Integration (HTML, CSS etc,.)

Include the HTML template (index.html) used to submit the URL and display the result.to stylish the page using the css (style.css).

6. Testing and Results

1)Testing Strategy

To ensure that the Web Application Firewall (WAF) functions correctly, a series of manual and automated tests were conducted. These tests aimed to verify the following:

- The correct detection of malicious or suspicious URLs.
- The proper logging of all scanned URLs and their statuses.
- The user interface's ability to provide clear feedback (Safe or Malicious).
- Admin routes such as viewing blocked URLs and resetting logs.

Testing was performed using both:

- **Safe URLs**: such as https://www.google.com, https://www.wikipedia.org.
- **Suspicious URLs**: such as http://free-gift.com, https://bit.ly/stealer, http://malware-site.exe.

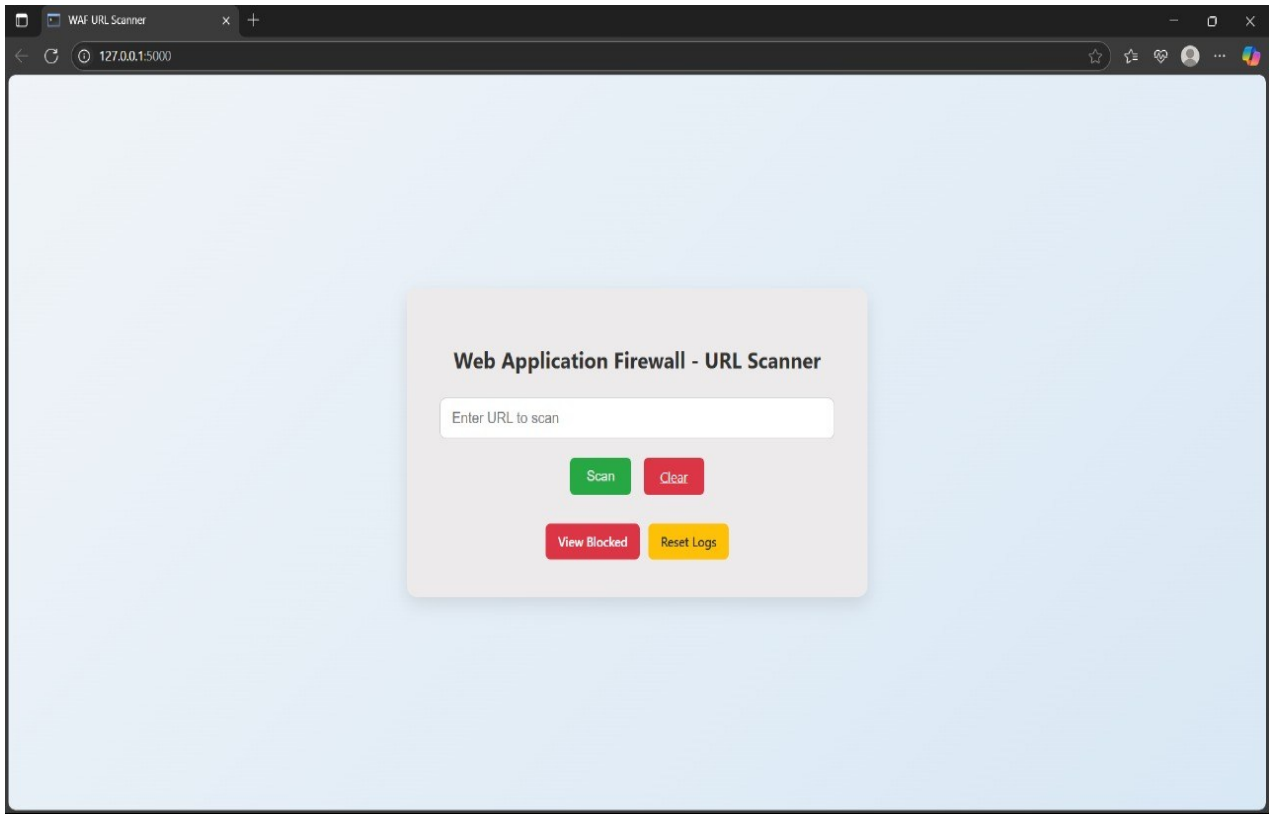
2)Test Cases and Results

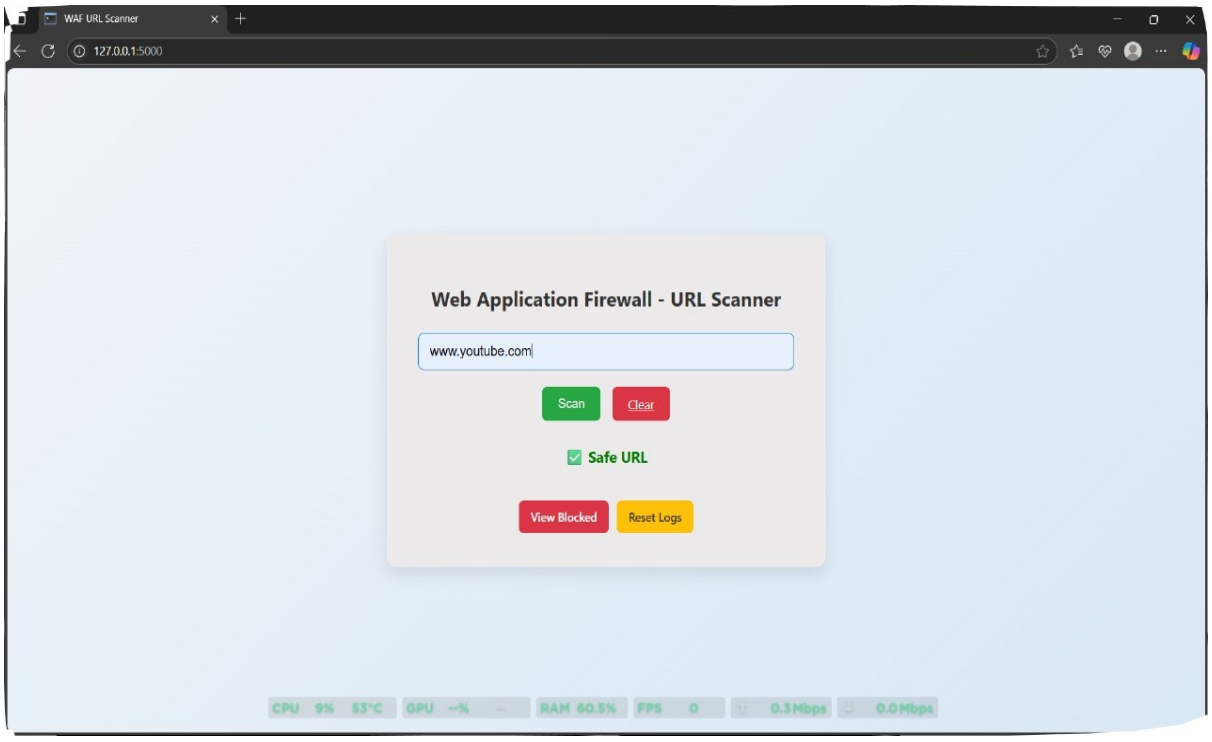
Test Case	Input URL	Expected Result	Actual Result	Status
TC1	https://www.google.com	✅ Safe URL	✅ Safe URL	Passed
TC2	http://bit.ly/phishing	❌ Malicious	❌ Malicious	Passed

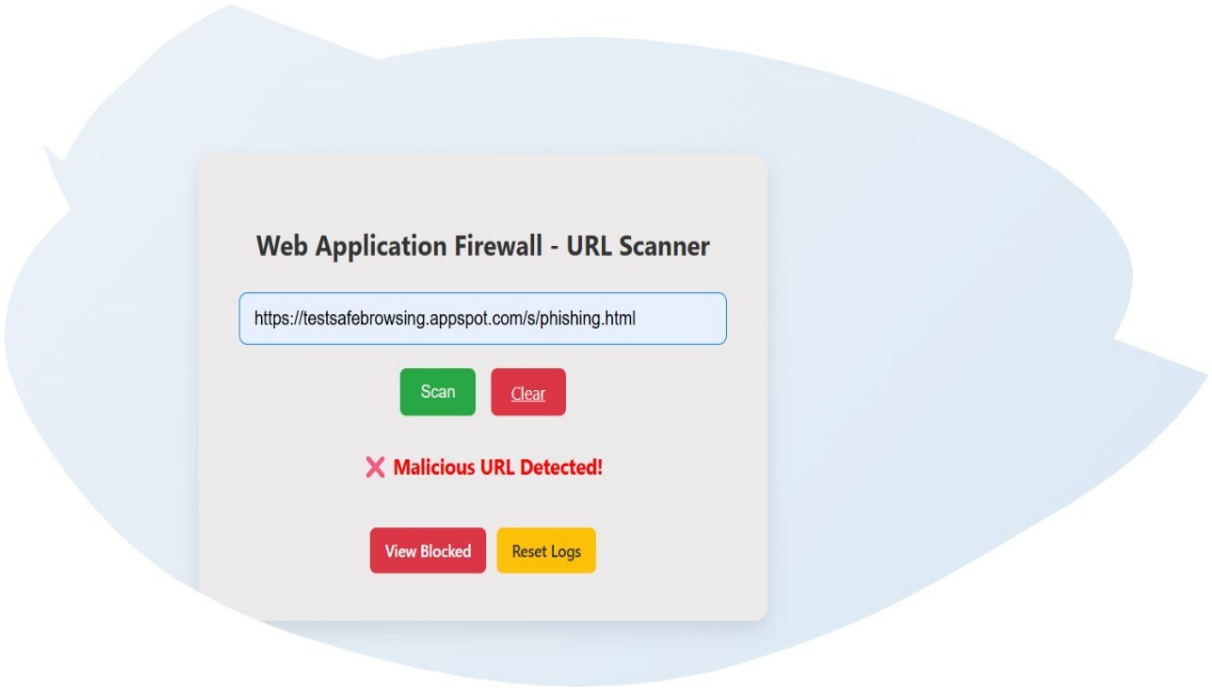
Test Case	Input URL	Expected Result	Actual Result	Status
		URL Detected!	URL Detected!	
TC3	http://secure-login.bank.com	❌ Malicious URL Detected!	❌ Malicious URL Detected!	Passed
TC5	https://example.com/download.zip	❌ Malicious URL Detected!	❌ Malicious URL Detected!	Passed
TC6	https://educational-site.org	✅ Safe URL	✅ Safe URL	Passed

---

3)Output Screenshots:


WAF URL Scanner
127.0.0.1:5000
Web Application Firewall - URL Scanner
Enter URL to scan
Scan
Clear
View Blocked
Reset Logs


WAF URL Scanner
127.0.0.1:5000
Web Application Firewall - URL Scanner
www.youtube.com
Scan
Clear
Safe URL
View Blocked
Reset Logs



4)Summary of Testing:

- The application correctly identifies suspicious and harmful URLs based on predefined regex patterns.
- All inputs are accurately logged with timestamps.
- The user interface responds instantly with appropriate messages.
- Admin functions work as expected and allow log management and blocked URL monitoring.

7.Challenges Faced:

During the development of the Web Application Firewall (WAF) project, several technical and practical challenges were encountered. These

obstacles required thorough analysis, problem-solving, and collaboration among team members to ensure successful implementation.

1)Integration with Frontend and Backend:

- **Challenge:** Connecting the Python (Flask) backend with the HTML/CSS frontend and ensuring smooth data flow and response rendering.
- **Solution:** Used Flask's render_template and request.form functions effectively to pass user input and display results dynamically.

2)Pattern Detection Accuracy:

- **Challenge:** Designing effective and comprehensive regular expressions (regex) to detect a wide range of malicious URL patterns without generating false positives.
- **Solution:** Studied multiple sources including known phishing patterns and integrated a broad regex list to improve accuracy.

3)Log Management:

- **Challenge:** Maintaining a readable and consistent format for log files, and handling file operations safely without data loss.
- **Solution:** Implemented log appending and provided an admin route to reset logs securely using file handling in Python.

4)Handling Dynamic URLs:

- **Challenge:** Some malicious URLs use dynamic or shortened formats (e.g., bit.ly) that are harder to detect.
- **Solution:** Added support to scan shortened links and included commonly used redirect domains in the suspicious patterns list.

5)Admin Panel Functionalities:

- **Challenge:** Ensuring that the admin functionalities such as viewing blocked URLs and clearing logs were secure and functioned without errors.
- **Solution:** Created separate routes with @app.route and handled global state using Python data structures.

6)Deployment Issues:

- **Challenge:** Running the Flask application on different development machines with varying environments and dependencies.
- **Solution:** Used a virtual environment (venv) and documented the required packages in a requirements.txt file to standardize setup.

7)Testing with Real-World URLs:

- **Challenge:** Testing with real malicious URLs was risky and potentially unsafe.
- **Solution:** Simulated phishing URLs and used harmless placeholder examples to safely test detection patterns without exposing systems to actual threats.

8.Future Enhancements:

As technology evolves and cyber threats grow in complexity, the Web Application Firewall (WAF) system can be enhanced in various ways to improve

its efficiency, scalability, and security. The following enhancements are proposed for future development:

- **Machine Learning Integration**
- **Real-time Threat Intelligence API**
- **User Authentication and Role-Based Access**
- **Email or SMS Alerts**
- **URL Redirect and Behavior Analysis**
- **Dashboard for Analytics**
- **Multi-language and Multi-device Support**
- **Cloud Deployment**

9. Features:

The Web Application Firewall (WAF) project includes several key features that contribute to enhancing web application security by detecting and preventing malicious URLs.

Key Features:

- 🔍 **URL Scanner**
  Allows users to input any URL and check whether it is malicious or safe.
- 🛑 **Suspicious Pattern Detection**
  Utilizes regular expressions to identify common phishing and malware indicators.
- 📜 **Logging System**
  Automatically logs every scanned URL with timestamp and status (safe or blocked).

- 🚫 **Blocked URL Management**
  Maintains a list of URLs that were flagged as suspicious or dangerous.
- ♻️ **Reset Function**
  Admin interface to reset logs and blocked URLs for fresh monitoring.
- 🧠 **Simple Heuristic Engine**
  Matches against a broad set of keywords and structures known to appear in unsafe links.
- ⚡ **Fast and Lightweight**
  Built using Python and Flask for rapid responses and ease of deployment.
- 🔐 **Admin Panel Routes**
  Routes to view blocked URLs and reset logs are secured to prevent unauthorized manipulation.

10. Conclusion:

The **Web Application Firewall (WAF) project** developed during this internship demonstrates an effective and lightweight approach to identifying and blocking potentially malicious URLs. With the increasing sophistication of **phishing, malware, and social engineering attacks on web applications**, implementing a firewall mechanism that can filter out suspicious links at the application level is vital.

This project used **Flask** for the backend, along with regular expressions for pattern-based threat detection, providing a practical solution that can be deployed in real-time environments. The logging mechanism, admin controls, and ability to reset blocked lists make it not only functional but also manageable from a security operations perspective.

Development and testing phases, the project proved to be reliable in detecting a wide variety of **suspicious URL patterns.** While it may not replace advanced, enterprise-level WAF systems, it serves as a strong foundational tool suitable for small to medium web applications or as a teaching tool in cybersecurity education.

Learnings from this project have enhanced our understanding **of cybersecurity threats, real-world firewall mechanisms, and full-stack web application development**. With further improvements and enhancements, this system can evolve into a more intelligent and scalable defense mechanism against web-based threats.