A Web Application Firewall Using Reflex Agents

Abbhinav Bharadwaj

November 2025

A

Project Report

on

A Web Application Firewall Using Reflex Agents

by

Abbhinav Bharadwaj (2300970100003)

Under the supervision of

Prof. Mukesh Kumar Singh

Computer Science and Engineering

Galgotia's College of Engineering & Technology

Greater Noida, Uttar Pradesh

India - 201306

Affiliated to

Dr. A.P.J. Abdul Kalam Technical University

Lucknow, Uttar Pradesh, India-226031

December 2025

GALGOTIA'S COLLEGE OF ENGINEERING & TECHNOLOGY

GREATER NOIDA, UTTAR PRADESH, INDIA - 201306

CERTIFICATE

This is to certify that the project report titled **"A Web Application Firewall Using Reflex Agents"** submitted by **Mr. Abbhinav Bharadwaj [2300970100003]** to Galgotia's College of Engineering & Technology, Greater Noida, Uttar Pradesh, affiliated to Dr. APJ Abdul Kalam Technical University Lucknow, U.P. in partial fulfillment for the award of Degree of Bachelor of Technology in Computer Science & Engineering is a bonafide record of the project work carried out by them under my supervision during the year 2025-26.

Prof. Mukesh Kumar Singh
Assistant Professor
Dept. of CSE

Dr. Pushpa Choudhary
Professor and Head
Dept. of CSE

GALGOTIA'S COLLEGE OF ENGINEERING & TECHNOLOGY
GREATER NOIDA, UTTAR PRADESH, INDIA - 201306

ACKNOWLEDGEMENT

I have taken my best efforts in this project. However, it would not be possible without the kind support and help of many individuals and organizations. I extend my sincere thanks to all of them.

I am highly indebted to **Prof. Mukesh Kumar Singh** for his guidance and constant supervision. Also, I am highly thankful to them for providing necessary information and support in completing the project.

I am extremely indebted to **Dr. Pushpa Choudhary**, HOD, Department of Computer Science and Engineering, GCET and **Ms. Lopamudra Mohanty**, Mini Project Coordinator, Department of Computer Science and Engineering, GCET for their valuable suggestions and constant support throughout my project tenure.

I also express my sincere thanks to all faculty and staff members of Department of Computer Science and Engineering, GCET for their support in completing this project on time.

I also express gratitude towards my parents for their kind co-operation and encouragement which helped me in completion of this project. Our thanks and appreciations also go to our friends in developing the project and all the people who have willingly helped me out with their abilities.

Abbhinav Bharadwaj
2300970100003

ABSTRACT

This project implements a Web Application Firewall by integrating reflex-agent–based decision making within a reverse proxy architecture. The system operates as middleware, inspecting incoming client requests and autonomously responding to potential threats in real time. Machine learning models are employed to monitor, detect, and act upon malicious patterns in HTTP payloads. Two common web vulnerabilities-SQL Injection[CWE-89] and Cross-Site Scripting (XSS)[CWE-79] are addressed using Support Vector Machine and XGBoost classifiers, respectively. Experimental evaluation demonstrates detection accuracies of 98.54% for SQL Injection and 96.15% for XSS, indicating the effectiveness of the proposed agentic framework in enhancing adaptive web application security.

KEYWORDS: *Reflex Agents, Reverse Proxy, SQL Injection, XSS, Support Vector Machine, XGBoost*

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AI	Artificial Intelligence
AJAX	Asynchronous Javascript and XML
API	Application Programming Interface
B2B	Business-to-Business
BiLSTM	Bidirectional Long Short-Term Memory
CCP	Coding Copilot (Coding Agent)
CNN	Convolutional Neural Network
CRS	Core Rule Set
CSS	Cascading Style Sheets
CVSS	Common Vulnerability Scoring System
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
CSP	Content Security Policy
CSRF	Cross-Site Request Forgery
DL	Deep Learning
DOM	Document Object Model
HTML	Hypertext Markup Langugae
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IT	Information Security
LLM	Large Language Model
ML	Machine Learning
OWASP	Open Worldwide Application Security Project
PoLP	Principle of Least Privilege
PDCGs	Prompt Driven Code Generators (Generative AI)
PHP	Hypertext Preprocessor
RF	Random Forest
SaaS	Software as a Service
SQL	Structured Query Language
SQLi	SQL Injection
SVM	Support Vector Machine
TF-IDF	Text Frequency - Inverse Document Frequency
WAF	Web Application Firewall
XGBoost	eXtreme Gradient Boosting
XML	Extensible Markup Language
XSS	Cross-Site Scripting

1 INTRODUCTION

Web Application Security is among the fastest evolving domains in Computer Science. With the adaptation of new technologies built on top of each other, from PHP and AJAX with MySQL to Node and React with MongoDB, a thorough assessment of security reveals it did not decrease security risks, but made them complicated enough to be in the developer's oversight[1]. With the rise of modern AI agents in modern application development, for developing parts with Prompt Driven Code Generators (PDCGs) and applications as a whole with Coding Copilots (CCPs) (often referred to as *Vibe Coding*), it becomes important we reexamine how application security can be ensured[2]. Though Agents are on average better than human developers in writing secure code[3], their training data and method of operation can lead to inconsistencies in security performance[4].

1.1 BACKGROUND

Web Applications are centric to the modern digital landscape. With the surge in number of internet users, there is an increase in businesses preferring web based services over Desktop and Android Applications due to lesser development and maintainence cost. This correlates with an increase in website usage over app usage in recent years[5]. With web solutions being accessible from a variety of devices and technologies, comes the challenge of web vulnerabilities.

1.1.1 OWASP Top 10

The *Open Worldwide Application Security Project (OWASP)* is a leading cybersecurity organisation that handles several central security projects, including the *OWASP Top 10[6]*. OWASP Top 10 is a standard awareness document released every 5 years, representing a broad consensus of most critical security risks to web applications. OWASP also maintains the *Core Rule Set (CRS)*, a set of generic detection rules for protection from a wide range of attacks including the OWASP Top 10. It can be used with *ModSecurity*, an open source Web Application Firewall, or other supporting applications.

1.1.2 CWE

Common Weakness Enumeration, abbreviated as CWE, is a community maintained list of software and hardware weaknesses, often referred by the industry as a reliable resource. The CWE also maintains an yearly list of *Top 25 Most Dangerous Software Weaknesses[7]*.

1.1.3 CVE

Common Vulnerabilities and Exposures (CVEs) is a community driven catalog of publicly disclosed vulnerabilities[8]. It is more specific than the CWE index, hosting nearly 3,00,000 CVE Records open source.

1.1.4 CVSS

The *Common Vulnerability Scoring System (CVSS)* is an open framework that provides a standardized way to rate the severity of computer security vulnerabilities, on a scale from 0.0 to 10.0, with 10.0 being the most severe. It considers how a vulnerability may be exploited (called as an *Attack Vector*), how easily it is to exploit (called as *Attack Complexity*), privileges required to perform the attack, *et. cetera.*

1.2 EMERGING LANDSCAPE

Cybersecurity awareness and security conscious development in the industry has made several once prevalent vulnerabilities nearly extinct, such as *Path Traversal, Open Redirection, XML External Entities, Insecure Direct Object References, et. cetera.* However, building and maintaining secure systems remains a non trivial tasks today, with several dangerous vulnerabilities such as SQL Injection, Cross Site Scripting *XSS*, Cross Site Request Forgery *CSRF* and Remote Code Execution *RCE* among others surviving till date.

1.2.1 Increasing Application Complexity

Modern applications comprise of several systems working together, including multiple APIs, Microservices, Cache Systems, Distributed Servers, SaaS Services, *et. cetera* to provide cutting edge functionality and compete in meeting user requirements. This leads to an increase in attack surface, or potential points of attack.

1.2.2 Increasing Dependencies

Development teams often use open-source and proprietary dependencies to speed up their development and ship the product as fast as possible. When a security vulnerability is found in a library, it affects all the dependent applications[9].

1.2.3 Rise of Coding Agents

With Generative AI advancements, LLMs such as ChatGPT, Gemini, Claude and Coding Agents such as Cursor, Warp, Claude Code allow users to write code with the highest level of abstraction. Since the training data can not be guaranteed to be secure, specially in large amounts, they are prone to writing insecure code[2]. Since AI Systems are non-deterministic and prompt-dependent, application security becomes a trade of luck and attention[10].

1.3 MODERN APPROACHES TO APPLICATION SECURITY

Most if not all, of IT and Software firms are now Cybersecurity aware. Security is a core component of any modern software product.

1.3.1 Cybersecurity Departments

Leading software and IT firms have dedicated Cybersecurity departments, that are responsible for secure operations and actively handling security crises.

1.3.2 Security Aware Development Teams

Developers are now security aware, and learn to follow safe code practices as a non-trivial part of their career. There exist journals for information regarding the trends in security to keep developers up to date.

1.3.3 Crowdsourcing

There are Hacker communities that proactively discover vulnerabilities in applications, and report them for merit and monetary compensations. They are often referred to as *White Hat Hackers* or *Bug Bounty Hunters*.

1.3.4 Security Outsourcing

Many firms outsource some or all of their security components, either to product based services such as a **Web Application Firewall** or to B2B SaaS products, such as Cloudflare, Orca, CrowdStrike, *et. cetera.*

1.4 WEB APPLICATION FIREWALLS

A traditional Web Application Firewall is a reverse proxy that intercepts incoming traffic to validate and sanitize or block the user request. Most of the application vulnerabilities can be catered by this approach.

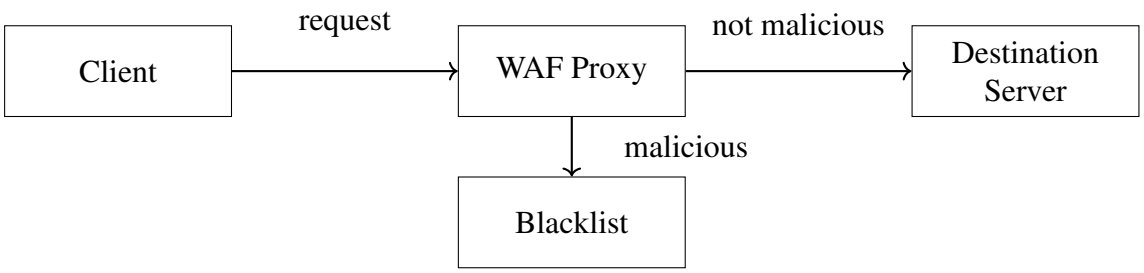

Figure 1.1: Workflow of a Web Application Firewall

1.4.1 Signature Based WAFs

They use specific pre-defined attack patterns to block known threats. Incoming traffic is matched against a vast set of known signatures.

1.4.2 Rule Based WAFs

They put reliance on strict rule sets, such as automatons or regular expressions to detect malicious user requests. One of the reputed rule sets is the OWASP CRS (Core Rule Set). The **Azure Web Application Firewall** incorporates the OWASP CRS.

1.4.3 WAF Agents

A part of active research, where the WAF has the ability to predict, act and adapt to cater to application security tasks.

1.5 WEB VULNERABILITIES

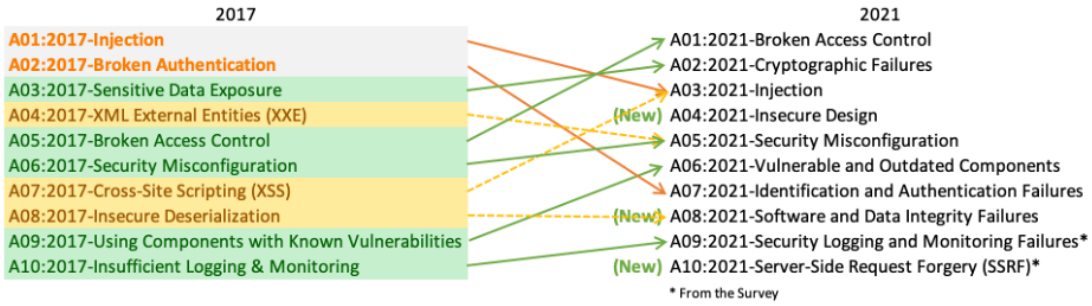

Figure 1.2: OWASP Top 10 2021 Comparison from OWASP[6]

Figure 1.2 shows the change of trends in vulnerabilities from 2017 to 2021, where several vulnerabilities changed positions, and three rose to the Top 10 list. It is noted that XSS and SQL Injection, the primary targets of this project, are listed at 3^{rd} rank.

Rank	ID	Name	Score	CVEs in KEV	Rank Change vs. 2024
1	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	60.38	7	0
2	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	28.72	4	+1
3	CWE-352	Cross-Site Request Forgery (CSRF)	13.64	0	+1
4	CWE-862	Missing Authorization	13.28	0	+5
5	CWE-787	Out-of-bounds Write	12.68	12	-3

Figure 1.3: Top 5 from CWE Top 25, 2025

CWE Top 25, 2025 as shown in figure 1.3, lists XSS at 1^{st} rank and SQL Injection at 2^{nd} rank. This highlights the importance and severity of the *age-old* attack vectors in the modern landscape. Hence, We approach these vulnerabilities first in this project.

2 LITERATURE REVIEW

2.1 INTRODUCTION

SQL Injections and XSS are severe vulnerabilities, and pose a major threat to any organization. Several approaches have been taken to mitigate the two attack vectors, by several direct and indirect means.

2.1.1 The state of SQL Injections

A survey done by **Aikido**[11] in 2024 notes:

- 6.7% of all vulnerabilities found in open-source projects were SQLi.
- 10% of all vulnerabilities found in proprietary projects were SQLi.
- Over 20% of closed source projects scanned are vulnerable to SQL injection when they first start using security tooling.
- For organizations vulnerable to SQL injection, the average number of SQL injection sites is nearly 30 separate locations in the code.

OWASP Top 10 2025 notes for *A05_2025 Injection*[12]:

- XSS (High frequency, Low impact) had $>$ 30k CVEs.
- SQLi (Low frequency, High impact) had $>$ 14k CVEs.
- The draft also notes that, due to higher CVEs of XSS than other injections, Injection is brought down to 5^{th} on the list.

In December 2023, a hacker group was reported to have stolen 2 million email addresses and other personal information from atleast 65 websites, mainly relying on SQL Injection attack vectors[13].

2.1.2 What are SQL Injections

SQL Injection is an attack vector where a threat actor can interfere with the queries being made to the application database, execute arbitrary queries, and obtain their output directly or indirectly. Consider the following insecure code:

```
String query="select \* from accounts
where custId='"+req.getParameter("id")+"'";
```

Now if the attacker modifies the id parameter or value in the JSON object, as:

```
' and id is 'victimId'; --
```

this payload effectively allows the threat actor to access data not to his belonging.

2.1.3 What is XSS

Cross Site Scripting (XSS) or *Improper Neutralization of Input During Web Page Generation* is an injection attack where the threat actor can inject malicious scripts into an otherwise benign and trusted website. XSS can be of several types, such as DOM XSS, Blind XSS, Stored XSS, Reflected XSS and Self XSS. Examples of XSS payloads are:

```
<img src=x onerror=alert(document.cookie)>
"onclick=prompt(8)>"@x.y
&lt;A HREF=\"http&#58;//google&#46;com/\"&gt;XSS&lt;/A&gt;
```

2.2 CONTEMPORARY APPROACHES TO MITIGATE INJECTIONS

Several steps are recommended to mitigate SQL Injections:

1. Use Parameterized Queries.
2. Ensure Input Validation and Sanitization.
3. Apply Principle of Least Privilege (PoLP), provide only the minimum necessary permissions for functions.
4. Use a Web Application Firewall

For XSS Mitigation:

1. Context Aware Output Encoding/Decoding
2. Validate and Sanitize Input
3. Implement Content Security Policy (CSP)
4. Use a Web Application Firewall

2.3 WEB APPLICATION FIREWALLS

Firewalls such as ModSecurity rely on OWASP CRS, a very successful rule set which helps in mitigating common attack vectors. They are based on strict signatures and patterns that are blacklisted or whitelisted.

2.3.1 Negative Security Model

Blocks known malicious patterns, signatures and known attack vectors. It **remains vulnerable to new or polymorphic attacks.** In case of a **0-day vulnerability, the rule sets are hard to modify.**

2.3.2 Positive Security Model

Allows only expected or ”good” incoming traffic. Highest security, but requires high maintainence.

2.4 MACHINE LEARNING APPROACHES

Machine Learning and Deep Learning approaches have been explored for mitigation of injection attack vectors in recent years. Habib(2024)[14] surveyed several developments in this domain and noted the following performance metrics across several approaches:

Table 2.1: Comparison of AI-based SQL Injection Detection Techniques (Reproduced from [14])

Algorithms	**Dataset**	**Evaluation Metrics**
CNN-BiLSTM	Different Websites	A: 98%
NB Algorithm	User URL access log data from ISP	A: 93.3%
SVM		A: 96.4%
SVM + SMO		A: 95.67%
SVM + PSO		A: 91.57%
MLP + LSTM		A: 99.67%
NB	Open source datasets	Not clear
SVM		
K-NN		
CNN	Dataset include generic, blind, error, union based SQLI	A: 94.84%, Precision: 85.67%, R: 96.56%
NB		Not mentioned
CNN		
SVM		
KNN		
RF	7576 malicious SQL queries from public repositories	A: 99.8%
Tensor Flow (BTC)		A: 99.6%
Adaboost classifier		A: 99.5%
DT		A: 99.5%
SGD Classifier		A: 99.6%
Deep ANN		A: 99.4%

Continued on next page

Algorithms	Dataset	Evaluation Metrics
Tensor Flow (LC)		A: 99.8%
LSTM	SQL injection statements taken from Kaggle	A: 62.32%, P: 66.23%, R: 65.16%, F1: 64.23%
KNN		A: 82.69%, P: 71.51%, R: 88.56%, F1: 79.13%
DT		A: 92.33%, P: 89.58%, R: 89.74%, F1: 89.66%
SQLNN		A: 96.16%, P: 97.28%, R: 92.23%, F1: 94.68%
AdaBoost	Not mentioned	Not clear
CNN	30,919 SQL injection queries from various websites	A: 96%, F1: 49%
RNN Auto encoder		A: 94%, F1: 92%
ANN		A: 94%, F1: 92%
NB		A: 82%, F1: 80%
SVM		A: 75%, F1: 49%
RF		A: 92%, F1: 89%
LR		A: 93%, F1: 90%
DT		A: 90%, F1: 87%
DT	OWASP dataset	A: 98%, TN: 97%, P: 98%, R: 97%, AUC: 98.2%
SVM (linear)	SQL injections using SQLmap, Wireshark on various websites	A: 91%, P: 91%, R: 90%, F1: 91.5%
KNN		A: 90%, P: 91%, R: 90.6%, F1: 90.7%
NB		A: 91.7%, P: 91%, R: 92%, F1: 91%
DT		A: 93%, P: 92.5%, R: 93%, F1: 93%
RF		A: 95%, P: 95%, R: 96.5%, F1: 95.5%
Our Method		A: 94%, P: 95%, R: 93%, F1: 94%
Naïve Bayes	Text file with 5234 SQL injections from GitHub	A: 97.5%, P: 91.2%, R: 1.0, F1: 95.4%
SVM		A: 82.6%, P: 1.0, R: 31.6%, F1: 48%
KNN		A: 86.5%, P: 79.8%, R: 80.1%, F1: 75.2%
DT		A: 84.1%, P: 61.7%, R: 1.0, F1: 76.3%
Markov Decision Processes (MDPs)	1000 SQL environments	Not clear
SVM	Novel dataset	A: 76.3%, P: 84.9%, R: 84.8%, F1: 91.1%, AUC: 83.3%
KNN		A: 87.6%, P: 84.8%, R: 96.7%, F1: 90.4%, AUC: 96.6%
Neural Network		A: 97.6%, P: 98.7%, R: 97.4%, F1: 98%, AUC: 98.9%
Multilayer Perceptron		A: 97.6%, P: 98.7%, R: 97.4%, F1: 98%, AUC: 98.9%

Continued on next page

Algorithms	Dataset	Evaluation Metrics
DT		A: 89.4%, P: 96.3%, R: 85.6%, F1: 90.6%, AUC: 94.6%
Random Forest		A: 83.3%, P: 89.6%, R: 87.5%, F1: 96.4%, AUC: 97.4%
Progressive Neural Network	62.2KB SQL query, Open source dataset	A: 97.897%

Habib(2024) noted the following across the approaches:

1. 90% of the papers preferred supervised learning, and remaining worked with unsupervised learning.
2. DL Models demonstrate high accuracy in various detection tasks.
3. They require large sets of labeled data, which is challenging for SQL Injection detection.
4. They are prone to overfitting, specially in case of noise or imbalance in the dataset.
5. ML Models, on the other hand require lesser labelled data and are more robust to noise.

Devi et al.(2025) investigated the use of reflex agents based on **Bidirectional Long Short Term Memory (BiLSTM)** for SQL Injection, XSS and XML Attacks. They noted that BiLSTM outperformed traditional classifiers such as SVM and Decision Trees[15].

2.5 CHALLENGES IN REFLEX AGENT APPROACH

2.5.1 What is a Reflex Agent

Russel & Norvig define a simple reflex agent as:

```
"Rule-based reasoning to map from percepts to optimal action;
each rule handles a collection of perceived states."
```

The Simple Reflex Agent is a stateless agent which perceives from the surrounding and acts in a deterministic manner. The architecture in this project, and as investigated by Devi et al.(2025)[15] can be proven to be based on reflex agent architecture. The reverse proxy can be called as a simple reflex agent, as:

1. Percept: Incoming user request
2. Condition: Classification result
3. Action: Allow or Block

2.6 CHALLENGES WITH DEEP LEARNING MODELS AS CONDITIONS

It is observed that Deep Learning Models outperform SVMs, Decision Trees and other ML based classifiers. But there are non-trivial challenges in using Deep Learner Models as conditions:

1. Deep Learning Models tend to overfit to noise, and imbalances in the data. SQL Injection datasets and XSS datasets have a lot of noise, making the training tricky.
2. They require larger amounts of data to understand the patterns, which may not be suitable.
3. CNN-BiLSTM has **300-600 times** more latency than Random Forest, XGBoost and Decision Trees[16]. This **reduces the server performance by more than 300 times.**

It is obvious that the trade for increase in accuracy takes a *severe* toll on WAF performance.

2.7 ACCURACY-PERFORMANCE TRADEOFF

2.7.1 SQL Injection Detection Model

Since SQL databases themselves are high latency, the model to be used must be low latency to avoid becoming a bottleneck in the system. This project hence utilizes the *Random Forest Classifier* on an open dataset, vectorized with *Text Frequency-Inverse Document Frequency (TF-IDF)*.

2.7.2 XSS Detection Model

XSS involves multiple languages simultaneously - HTML, CSS, Javascript, DOM event model, with virtually no limit on symbols and patterns involved. This makes the data very high dimensional - suitable for a classifier such as the **Support Vector Machine** on vectorized data such as with TF-IDF.

2.8 RANDOM FOREST

Random decision forest is an ensemble learning method for supervised learning. It involves training a multitude of decision trees on the data. In classification tasks, the output of the Random Forest is the class selected by most trees. Ensemble Learners like Random Forests and XGBoost

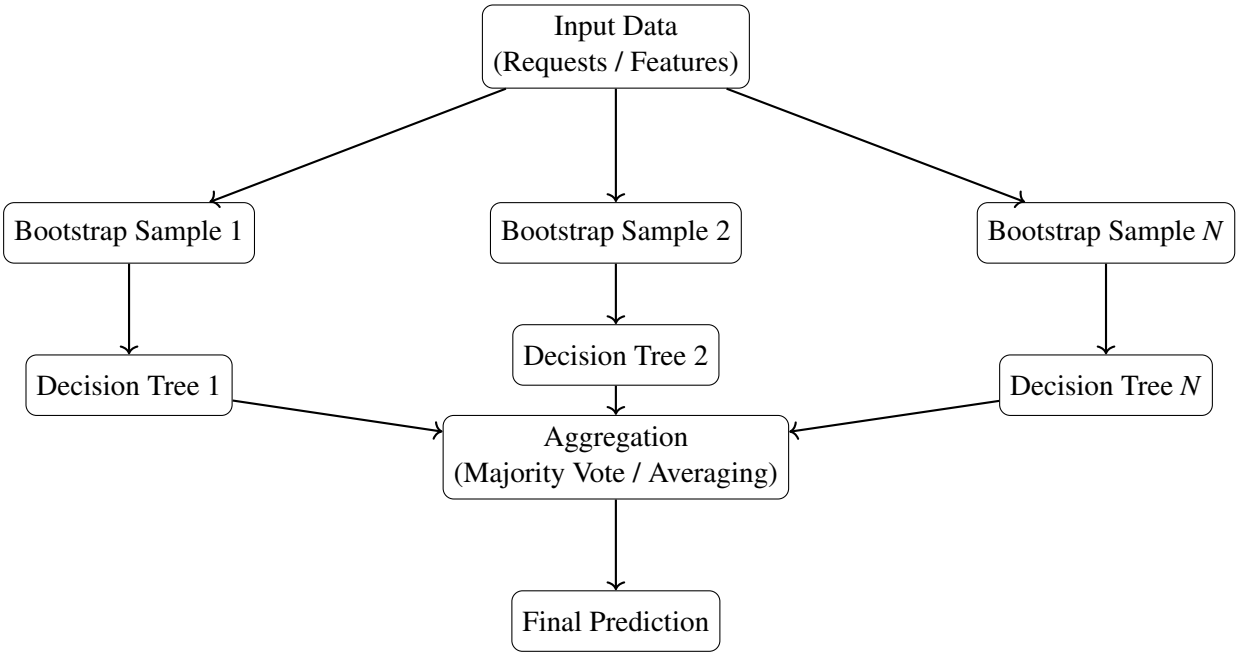

Figure 2.1: Architecture of a Random Forest Classifier

are widely used for classification tasks. The architecture of Random Forests is visualized as a parallel set of trees, where each tree is a decision tree.

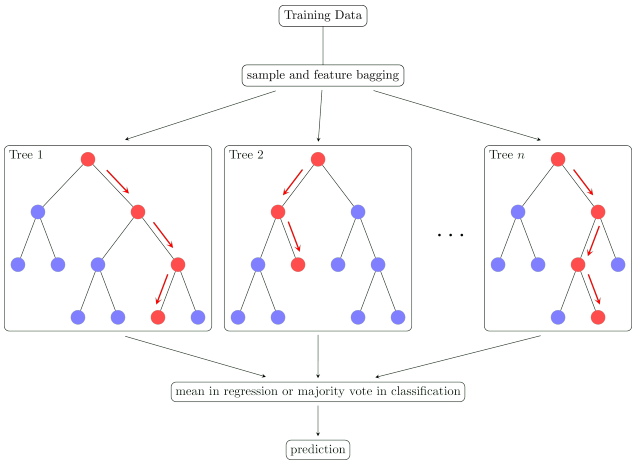

Figure 2.2: Random Forest Illustration (reproduced from Janosh[17])

2.9 SUPPORT VECTOR MACHINE

A Support Vector Machine finds the most optimal **hyperplane** which acts as the decision boundary between the classes with maximal margin from support vectors. In general, using a *linear kernel function*, if there are p features in the data, the hyperplane is $p-1$ dimensional.

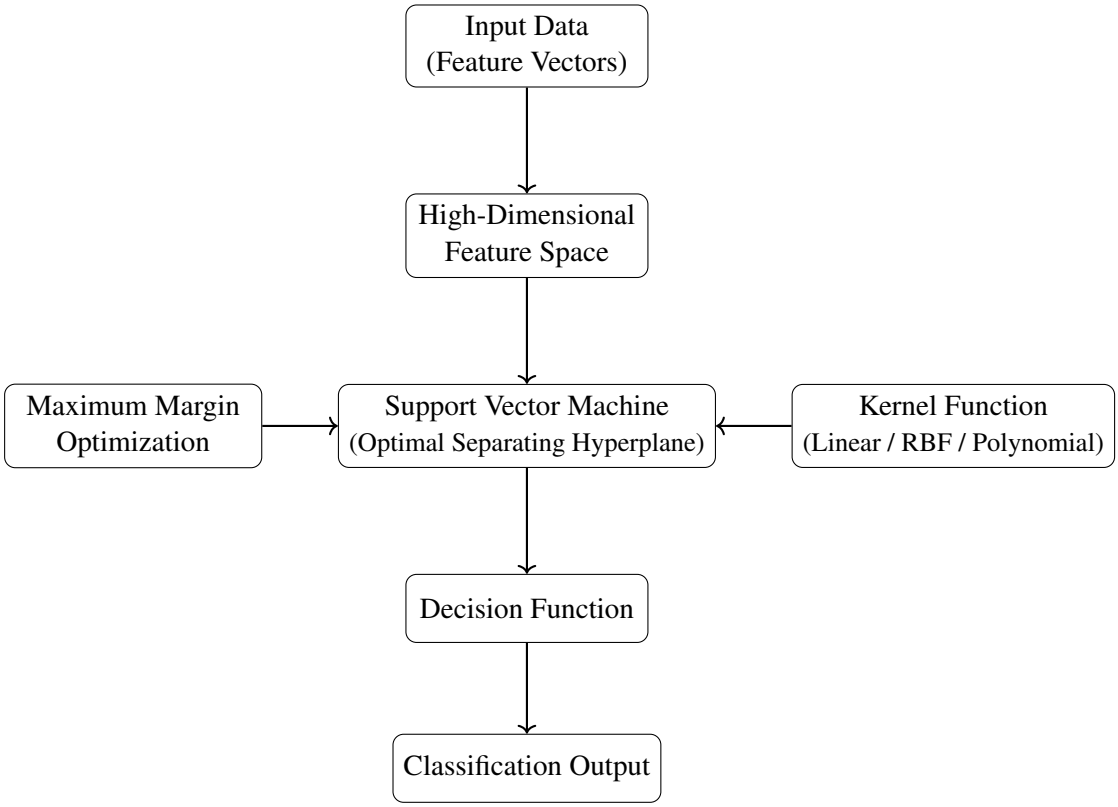

Figure 2.3: Architecture of a Support Vector Machine (SVM) Classifier

2.10 TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is a text mining method that scores word importance in a document relative to a whole collection (corpus), balancing how often a word appears in one text called as Term Frequency (TF) with how rare it is across all texts, called as Inverse Document Frequency (IDF), giving high scores to unique, relevant keywords for tasks like search engines, text classification, and topic modeling. This vectorization provides vector representation of data with ample features for many natural language processing problems.

3 PROBLEM FORMULATION

Contemporary Web Application Firewall systems are based on rule based sets, which are largely successful but can not protect from patterns not known to them. When a new pattern is found in 0-day, it is not easy to update the rule set. They struggle to detect obfuscated, polymorphic or zero-day injection patterns. Recent studies have explored the usage of Machine Learning and Deep Learning for detecting injection attacks which may be employed in an agent driven architecture. Deep Learning outperforms Machine Learning, but the challenge remains of overfitting, data size, robustness to noise and most importantly, performance metrics. Deep Learning models are upto 600 times slower per sample, making them an expensive trade for a high bandwidth application of a reverse proxy server.

3.1 PROBLEM STATEMENT

To design and implement a Agent Based Web Application Firewall which employs reflex agents powered by Machine Learning Models, namely SVM and XGBoost, to detect and mitigate Injection attacks in real time.

3.2 CONSTRAINTS AND SCOPE

1. The system currently focuses on SQL Injection and XSS only, given their severity and broad access points.
2. It is trivial to extend from HTTP to HTTPS, so only HTTP 1.1 has been prototyped.
3. Only XGBoost and SVM models have been implemented, given the research and reasons.
4. Encrypted Traffic and other web vulnerabilities remain out of scope for this work.

4 PROPOSED WORK

In this project, a Web Application Firewall (WAF) is proposed that operates as a reverse proxy between clients and the protected web application. The proposed system employs reflex agents to inspect incoming HTTP requests and take immediate actions based on the inferred threat level.

The architecture consists of modular middleware components, each responsible for detecting a specific class of web attacks. In the current implementation, the system focuses on SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks. For each incoming request, relevant features are extracted from the request payload and forwarded to machine learning–based detection modules.

To enable efficient and real-time detection, Support Vector Machine (SVM) and XGBoost classifiers are utilized due to their relatively low inference latency and robustness in handling noisy and high-dimensional data. Based on the classification output, the reflex agent takes an appropriate action, such as allowing the request, blocking it, or logging it for further analysis.

The proposed WAF is designed to be lightweight, extensible, and scalable, allowing additional detection agents or models to be integrated in the future. By combining machine learning–based detection with an agent-oriented decision mechanism, the system aims to improve adaptability against evolving attack patterns while maintaining performance suitable for real-time web traffic inspection.

The WAF is tested with DVWA (Damn Vulnerable Web Application) which poses as a suitable vulnerable application for safe and practical demonstration purposes.

5 SYSTEM DESIGN

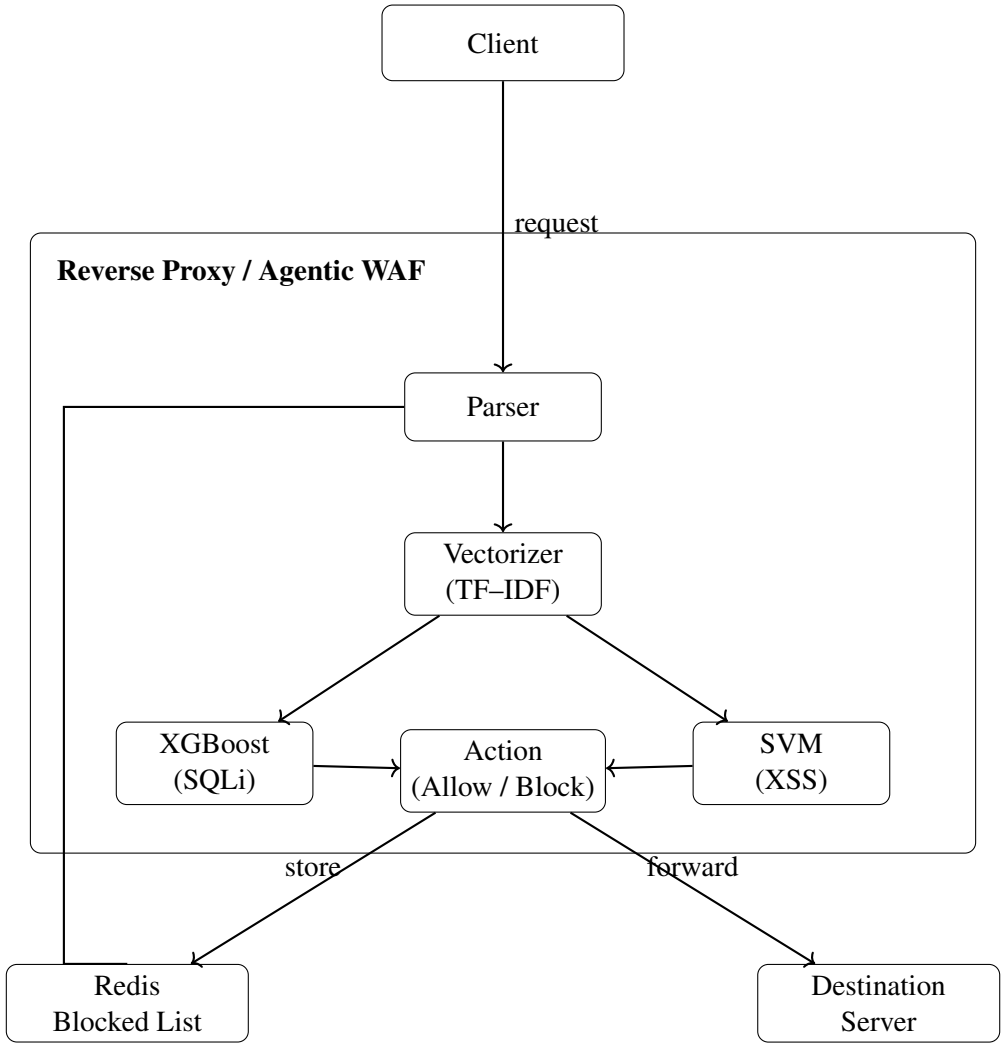

Figure 5.1: Architecture of Agentic Web Application Firewall using Reflex Agents

5.1 CLIENT

The client sends HTTP requests to the reverse proxy server. It is assumed that it may act as a threat actor.

5.2 DESTINATION SERVER

This is the server hosting the vulnerable application. For demonstration purposes, this application is chosen to be DVWA (Damn Vulnerable Web Application)

5.3 REVERSE PROXY

The Reverse Proxy Server will analyze the request before forwarding it to the vulnerable application server.

5.3.1 Parser

Parses incoming HTTP requests for their headers, meta data and form data. It also presents the request IP to the in-memory database to check if the user is already blocked.

5.3.2 Vectorizer

The TF-IDF Vectorizer is a part of the ML Pipeline that vectorizes the request inputs for Classification.

5.3.3 XGBoost

This classifies the user input against SQL Injection.

5.3.4 SVM

This classifies the user input against XSS.

5.3.5 Action

This automaton (program) performs an AND operation to decide whether the input is safe.

5.4 REDIS

Redis acts as a in-memory database for WAF settings and blocked IPs. The blocked IPs are stored in form of **Tries** for $O(1)$ time access as IPV4 addresses are essentially 4 octets. Redis also ensures scalability and consistency of the server.

6 IMPLEMENTATION

6.1 TECHNOLOGY STACK

- Server Language: Python 3
- Framework: aiohttp - Asynchronous IO HTTP Server
- Vectorizer: TfidfVectorizer (sklearn)
- Machine Learning: scikit-learn
- Database: Redis
- Desktop Application: Tkinter
- SIEM: HTML, CSS, Tailwind, Javascript
- Deployment: Docker

6.2 REVERSE PROXY IMPLEMENTATION

The reverse proxy is implemented using the aiohttp asynchronous framework. It listens for incoming HTTP requests and forwards them to the upstream server after inspection. All HTTP methods, headers, query parameters, and request bodies are preserved during forwarding, except Hop-by-hop headers that are meant to change with each node in the network path for consistency.

6.3 PARSER

The request parser extracts relevant components such as URL path, query parameters, headers, and request body. Inputs are normalized to a standard textual format before feature extraction.

6.4 VECTORIZER

Feature extraction is performed using TF–IDF vectorization. Parsed request data is transformed into numerical feature vectors compatible with machine learning classifiers.

6.5 MACHINE LEARNING

6.5.1 SVM - XSS

SVM is trained with linear kernel on an open dataset from Kaggle by Hussain (2020)[18]. Data has been synthetically augmented to balance with natural queries using Claude, as the dataset was meant for classification of SQLi detection within SQL Queries only.

6.5.2 XGBoost - SQL Injection

XGBoost is trained on an open dataset from GitHub by Kumar(2024)[19]. Data has been synthetically augmented to balance with natural queries using Python's **Faker** library.
Tree parameters: max_depth is set to 4, min_child_weight to 10, gamma=2, reg_alpha=1 and reg_lambda=3

6.5.3 Miscellaneous

Redis and DVWA are setup with Docker. Launch window is made with Tkinter. SIEM is made with HTML CSS and Javascript.

7 RESULT ANALYSIS

A fully working model has been constructed, with the XGBoost's efficiency 98.54% and SVM's efficiency 96.15%. Appropriate Demonstrations with DVWA provided.

Table 7.1: Performance Evaluation of Machine Learning Models

Model	**Attack Type**	**Accuracy**	**Precision**	**Recall**	**F1-score**
XGBoost	SQL Injection	98.54%	0.96	0.98	0.97
SVM	XSS	96.15%	0.94	0.95	0.95

The server works with adding minimal latency.

8 CONCLUSION, LIMITATIONS AND FUTURE SCOPE

8.1 CONCLUSION

This project presented an Agentic Web Application Firewall implemented using a reverse-proxy architecture with reflex agents. The system intercepts incoming client requests, performs feature extraction, and applies machine learning models to detect SQL injection and cross-site scripting attacks.

Support Vector Machine and XGBoost classifiers were trained and integrated for XSS and SQL injection detection respectively. Experimental results demonstrate that the proposed system is capable of accurately identifying malicious requests while maintaining acceptable inference latency for real-time deployment.

The modular design of the system allows independent improvement of individual components, making it adaptable to evolving web security threats.

8.2 LIMITATIONS

Despite its effectiveness, the proposed system has certain limitations. The current implementation focuses only on SQL injection and cross-site scripting attacks and does not address other web vulnerabilities such as CSRF, command injection, or authentication bypass attacks.

The machine learning models are trained on specific datasets, which limits generalization to highly novel or adversarial payloads, which would require a training feedback loop. The system also returns false positives at inputs crafted to resemble SQL Injection or XSS payloads. Additionally, the system currently performs request-level analysis and does not incorporate long-term user behavior profiling or session-based anomaly detection with unsupervised learning.

Scalability may also be impacted as the number of deployed models increases, potentially affecting inference latency under very high traffic conditions.

8.3 FUTURE SCOPE

The proposed system can be extended in several directions. The covered vulnerability classifiers can be improved by combining multiple models, or using Deep Learning for stricter security policies in a federated manner. Additional detection modules can be incorporated to handle a wider range of web vulnerabilities beyond SQL injection and XSS.

Future work may include integrating user behavior analysis and clustering techniques to enable adaptive and context-aware security policies. The use of lightweight deep learning models or ensemble decision strategies can also be explored to improve detection accuracy while maintaining low latency.

REFERENCES

[1] E. Iannone, R. Guadagni, F. Ferrucci, A. D. Lucia, and F. Palomba, "The secret life of software vulnerabilities: A large-scale empirical study," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 44–63, Jan. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9672730

[2] A. Mohsin, H. Janicke, A. Wood, I. H. Sarker, L. Maglaras, and N. Janjua, "Can We Trust Large Language Models Generated Code? A Framework for In-Context Learning, Security Patterns, and Code Evaluations Across Diverse LLMs," *arXiv preprint arXiv:2406.12513*, 2024.

[3] O. Asare, M. Nagappan, and N. Asokan, "Is github's copilot as bad as humans at introducing vulnerabilities in code?" *Empirical Software Engineering*, vol. 28, no. 6, p. 129, 2023. [Online]. Available: https://doi.org/10.1007/s10664-023-10380-1

[4] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," *arXiv preprint arXiv:2307.03109*, 2023. [Online]. Available: https://arxiv.org/abs/2307.03109

[5] Amplitude Labs, "2022 App vs. Website Trend Report," https://amplitude.com/guides/2022-app-vs-website-report, April 2022, accessed: [Insert Date You Accessed the Report].

[6] OWASP Foundation, "OWASP Top 10: 2021 – The Ten Most Critical Web Application Security Risks," OWASP Foundation, Report, September 2021, project Leader: Andrew van der Stock. [Online]. Available: https://owasp.org/Top10/2021/

[7] The MITRE Corporation and CISA, "2025 CWE Top 25 Most Dangerous Software Weaknesses," The MITRE Corporation, Bedford, MA, Tech. Rep., December 2025, the full list is available online. [Online]. Available: https://cwe.mitre.org/top25/archive/2025/2025_cwe_top25.html

[8] "CVE: Common Vulnerabilities and Exposures," https://www.cve.org/, The MITRE Corporation, 1999, accessed: 2025-12-15.

[9] S. Kumar and C. Fetzer, "A comprehensive study on the impact of vulnerable dependencies on open-source software," *IEEE Transactions on Software Engineering*, vol. XX, no. YY, pp. 1–12, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10771210

[10] K. E. W. Lucas Jasper Jacobsen, "The promises and pitfalls of large language models as feedback providers: A study of prompt engineering and the quality of ai-driven feedback," *AI*, vol. 6, no. 2, p. 35, 2024. [Online]. Available: https://www.mdpi.com/2673-2688/6/2/35

[11] M. Jackson, "The state of sql injections," https://www.aikido.dev/blog/the-state-of-sql-injections, 2024.

[12] "A05:2025 Injection - OWASP Top 10:2025," https://owasp.org/Top10/2025/A05_2025-Injection/, OWASP Foundation, 2025, oWASP Top 10:2025 Release Candidate 1, accessed 2025-12-15.

[13] SecurityWeek, "Millions of user records stolen from 65 websites via sql injection attacks," *SecurityWeek*, February 2024, accessed: 2025-12-15. [Online]. Available: https://www.securityweek.com/millions-of-user-records-stolen-from-65-websites-via-sql-injection-attacks/

[14] U. Habib, "A survey on implication of artificial intelligence in detecting sql injections," https://www.researchgate.net/publication/378496266_A_Survey_on_Implication_of_Artificial_Intelligence_in_detecting_SQL_Injections, 2023, preprint or article hosted on ResearchGate, accessed 2025-12-15.

[15] S. D. R. N Anitha Devi *et al.*, "Web application security enhancement through automated form analysis and ai-driven attack detection," *International Research Journal on Advanced Engineering Hub*, vol. 3, no. 4, pp. 1460–1464, 2024. [Online]. Available: https://www.researchgate.net/publication/393988516_Web_Application_Security_Enhancement_Through_Automated_Form_Analysis_and_AI-Driven_Attack_Detection

[16] K. A.-A. Vandit R Joshi *et al.*, "Hybrid ai intrusion detection: Balancing accuracy and efficiency," *Sensors*, vol. 25, no. 24, p. 7564, 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/24/7564

[17] J. Riebesell and S. Bringuier, "Collection of scientific diagrams," 2020, 10.5281/zenodo.7486911 - https://github.com/janosh/diagrams. [Online]. Available: https://github.com/janosh/diagrams

[18] S. S. Hussain Shah, "Cross site scripting xss dataset for deep learning," https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning, 2020, kaggle dataset.

[19] A. Kumar, "Sql injection dataset: Sqlqueriesdata.csv," https://raw.githubusercontent.com/ankitkumarhello20/sql-injection-dataset/main/SqlQueriesData.csv, 2024, cSV file from GitHub repository sql-injection-dataset.