Software Requirement Specification Document for Enhancement WAF Model using Deep Learning and Machine Learning Techniques

Supervised by: Dr. Heba Osama, Eng. Saja Saadoun

Monica Medhat Maurice Gendy, Martina Milad Ragheb Bishay, Samer Raafat Samir Youssef, Kevin Emad Rizk Abdelmessih

February 14, 2024

Table 1: Document version history

Version	Date	Reason for Change
1.0	14-Jan-2024	SRS First version's specifications are defined.

GitHub: https://github.com/MonicaMedhat20/Web-Application-Firewall-WAF-Enhancement-.git

Contents

Abstract

These days, as a result of the globalization and expansion of technology, there is a significant increase in both the quantity and variety of online attacks, which poses several security risks to internet users. To prevent and filter these online attacks, Web Application Firewalls (WAF) are designed to stop any malicious requests or actions that could potentially expose a website. This project aims to enhance the performance and the run time of the proposed WAF by using Machine learning and Deep learning techniques. Finally, WAF acts as an essential security measure that strengthens cybersecurity postures and web applications to meet the demands of the modern digital world.

1 Introduction

1.1 Purpose Of this document

Software Requirements Specification is a document that outlines what the project will accomplish and how it should perform. The SRS outlines the functional requirements of the web application, identifying what the application is predicted to do. For a WAF, this may include features such as request filtering, traffic monitoring, and protection against common web vulnerabilities like SQL injection or cross-site scripting (XSS). It will enhance these features like scalability, security, performance, and other essentials that the WAF needs to successfully safeguard the online application. The SRS establishes benchmarks for the WAF's performance. This involves taking into consideration how to provide strong security measures with as little negative influence on the web application's performance as possible.

1.2 Scope of this document

The scope of the (SRS) in the context of (WAF) refers to the boundaries and extent of what the SRS document covers to make the system's fundamentals clear so that it can be understood. It includes the fundamental class diagram for the application, design constraints, data design, and functional and non-functional needs. Request Filtering: Define how incoming requests to the web application should be filtered by the WAF. This includes policies that prohibit or permit particular kinds of traffic by predetermined standards. Security Policies: Specify the protections against SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and other prevalent online vulnerabilities that the WAF should enforce.

1.3 Business Context

By matching the technical requirements of the WAF with the overarching objectives and requirements of the company, (SRS) contributes to the establishment of the business context within the framework of (WAF). An understanding of the WAF's role in accomplishing business goals and meeting security requirements is made possible by the SRS's business context. The following are important components of the SRS for a WAF's business context. Business Objectives and Goals: Give a clear explanation of the broad business goals and objectives that the WAF seeks to promote. This can entail safeguarding sensitive client information, guaranteeing regulatory compliance, or enhancing the web applications' overall security posture. Evaluation of Risk: Determine and record the precise risks and threats that the company faces concerning its web applications. This covers a review of possible weak points and how security events affect the company. Requirements for Compliance: Indicate which legal and regulatory requirements the organisation needs to follow and how the WAF will help them comply both now and in the future. This could include local data protection laws or industry-specific standards (like PCI DSS and HIPAA).[1]

2 Similar Systems

2.1 Academic

- **Montarulial et al.**[2]The ModSecurity WAF is mostly unsuccessful because it does not construct any model of the traffic observed by the protected web services, resulting in many false alarms, and it is readily defeated by adversarial SQLi attacks. Through adversarial training, the suggested AdvModSec methodology enhances robustness to adversarial SQLi assaults while improving the trade-off between detection and false positive rates. Based on empirical findings, AdvModSec enhances the adversarial robustness of vanilla ModSecurity by up to 42% and increases its detection rate by 21%. The approach offers a first tangible illustration of how adversarial machine learning may be applied to improve the resilience of WAFs against adversarial attacks. Along with promising future research possibilities focused on reinforcing classical rule-based solutions with machine learning-based approaches, and bridging the gap between these two worlds, the study also analyses related work and the limitations of the suggested approach. The research comes to the conclusion that web threats, such as DDoS, XSS, and SQL injection attacks, can be successfully identified and stopped by the suggested deep learning-based WAF. The suggested approach was able to identify novel and unidentified attacks and performed more accurately than conventional WAFs. Additionally, the study found several attack-indicating characteristics and factors that can be applied to raise threat detection accuracy. When compared to conventional WAFs, the suggested layered architectural paradigm increased threat detection accuracy. The paper suggests training the WAF and improving its performance with deep learning algorithms, such as CNNs and LSTMs. The paper's overall findings demonstrate how deep learning techniques can be used to strengthen web application security and fend against cyberattacks.

- **Dawadi et al.**[3] The suggested deep learning-based WAF is successful in identifying and stopping online threats, such as DDoS, XSS, and SQL injection assaults which are the most common attacks that face websites, according to the paper's conclusion. The suggested approach was able to identify novel and unidentified attacks and performed more accurately than conventional WAFs. In addition, the study found several attack-indicating characteristics and factors that can be applied to raise threat detection accuracy. When compared to conventional WAFs, the suggested layered architectural paradigm increased threat detection accuracy. The paper suggests training the WAF and improving its performance with deep learning algorithms, such as CNNs and LSTMs. The paper's overall findings demonstrate how deep learning techniques can be used to strengthen web application security and fend against cyberattacks.

- **Applebaum et al.**[4] The author mentions the limitations of signature-based web application firewalls and the potential of machine learning-based approaches to overcome these limitations. Signature-based web application firewalls use only a pre-defined set of rules and patterns to detect known threats. However, they may not be effective in detecting zero-day attacks (new or unknown attacks).In addition, this paper also provides a review of machine-learning-based WAF methods and identifies open issues. The authors conclude that machine-learning-based methods have advantages over signature/rule-based methods as they can address the vulnerability to zero-day attacks and can be easier to configure and keep up to date. However, the effectiveness of machine-learning-based WAFs in protecting current attack patterns targeting web application frameworks is still an open area for further investigation. The paper also lists applicable datasets, such as CSIC2010, for testing and training machine learning-based algorithms for web application firewalls.

- **Alghawazi et al.** [5] suggested deep learning architecture based on an RNN autoencoder is a viable solution for detecting SQL injection attacks on online applications, according to the article. The testing findings indicated that the suggested strategy outperformed other current approaches for identifying SQL injection attacks in terms of accuracy and F1-score. The authors also note that the dataset utilised in this study was quite limited, and they urge that future studies extend the dataset and use the models in real-world circumstances. The study gives a detailed assessment of the literature on ML and DL strategies for detecting SQL injection attacks and examines the possibility of future research employing more complicated architectures for the RNN autoencoder.

- **Seyyar et al.** [6] In the categorization of anomalous and normal requests, the suggested strategy in the study based on BERT and deep learning approaches produced a success rate of over 99.98% and an F1 score of over 98.70%. The BERT model has a strong representational capability of URLs, which greatly benefits high-performance web attack detection. The authors employed an aggregate dataset created by combining three separate datasets in both the training and validation phases. As a result, their methodology is capable of successfully generalising data from numerous datasets.

- **Prandl et al.** [7] explains the various harmful malware types that web application developers and internet security experts should be aware of. Finding out how effective free WAFs are generally and whether or not they can be thought of as competitive alternatives to proprietary solutions are its main objectives. It also examines the harmful kinds of patterns that routinely get past the free WAFs and how successful they are at blocking them. Three methods are utilized to check: Guardian, Webknight, and ModSecurity. The study concluded that ModSecurity is the best technique for free WAFs since it is very good at thwarting frequently occurring attacks like XSS and SQL injection and can handle a sizable number of attacks while producing a negligible number of false alerts.

- **Gandotra et al.** [8] explains how to use iptables to protect against Application Layer and Web-based attacks without causing a prolonged system degradation that maintains the system's speed and network quality. Its main objective is to evaluate the effectiveness and speed of Iptables rules for Web-based attack detection and mitigation, as well as other application layer attacks. Techniques like SQL injection and cross-site scripting are used to test web-based attacks, whereas HTTP flooding, FTP flooding, and FTP bounce scanning are used to test application layer attacks. The study concluded that iptables can be utilized as an Application Layer Firewall with the right attack signatures since it is effective at identifying attacks.

2.2 Business Application

1. **Imperva WAF:** [1] Web application firewall (WAF) hosted in the cloud, Imperva offers DDoS protection, load balancing, failover, and content delivery network (CDN) services. Additionally, it offers defense against the OWASP Top Ten attack types. It functions by altering the protected application's DNS. Additionally, it has several inbuilt WAF rules that may be customized using an editor to defend against various kinds of attacks. Additionally, Imperva offers two-factor authentication that works with any website without requiring modifications to the application code.

2. **AWS WAF:** [9] It stands for Amazon Web Services, a cloud computing platform that Amazon offers. It provides a wide range of services, one of which is a WAF service that aids in protecting web applications from frequent attacks that can endanger their security or impact their availability. Control over access to online apps is possible thanks to this managed service. Additionally, it can be used to build unique rules for certain applications that prevent common pattern attacks like SQL injection and cross-site scripting (XSS). Additionally, AWS WAF interfaces with AWS Shield and Amazon CloudFront to offer a comprehensive security solution for web applications.

3. **Azure WAF:** [10] it is a security feature provided by Microsoft Azure that helps protect web applications from common web-based attacks such as SQL injection, cross-site scripting (XSS), and cross-site forgery. Located between the web application and the internet, this layer 7 firewall examines incoming traffic and prevents any fraudulent requests. Custom rules and policies can be set up in Azure WAF to satisfy certain security needs.

Figure 1: Business Applications

3 System Description

3.1 Problem Statement

The main problem is to enhance the accuracy of the traditional WAF by training and testing datasets to detect known malware attacks and newly discovered attacks (zero-day attacks) and to provide quick response time to attack detection and prevention.

3.2 System Overview

The WAF's main purpose is to monitor traffic and filter/block any malicious requests sent. It can also create policies to define what is allowed and reject the rest.

The requests that are sent to access a web application pass-through data preprocessing, then data splitting; the split data is trained and tested using ML and DL techniques. The model is then evaluated, and the requests sent are classified accordingly to normal and malicious, the malicious ones are blocked by the WAF and cannot reach the the destination server, while the normal/ safe ones are passed seamlessly through the WAF reaching the destination server.

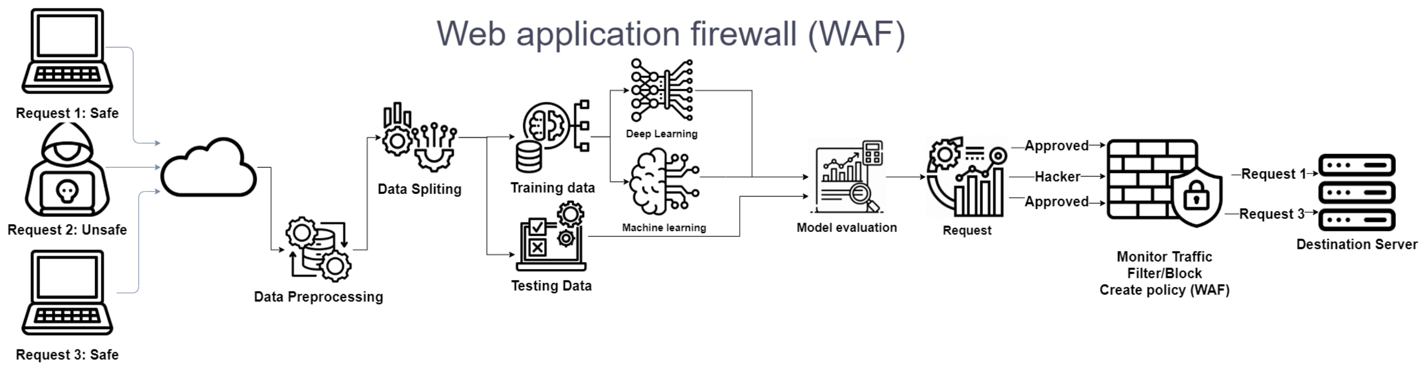


Figure 2: WAF system overview

3.3 System Scope

The main focus will be on the Web Applications.

1. It is easier for end users as they don't need to install an application (online usage).
2. Users have the opportunity to use the website from different browsers.
3. Web applications are efficient in development for businesses as they are relatively simple and cost-effective.
4. Web applications store the data on a server so there is no need to install them on a hard drive
5. It is beneficial for companies as there are no storage limitations online.

3.4 System Context

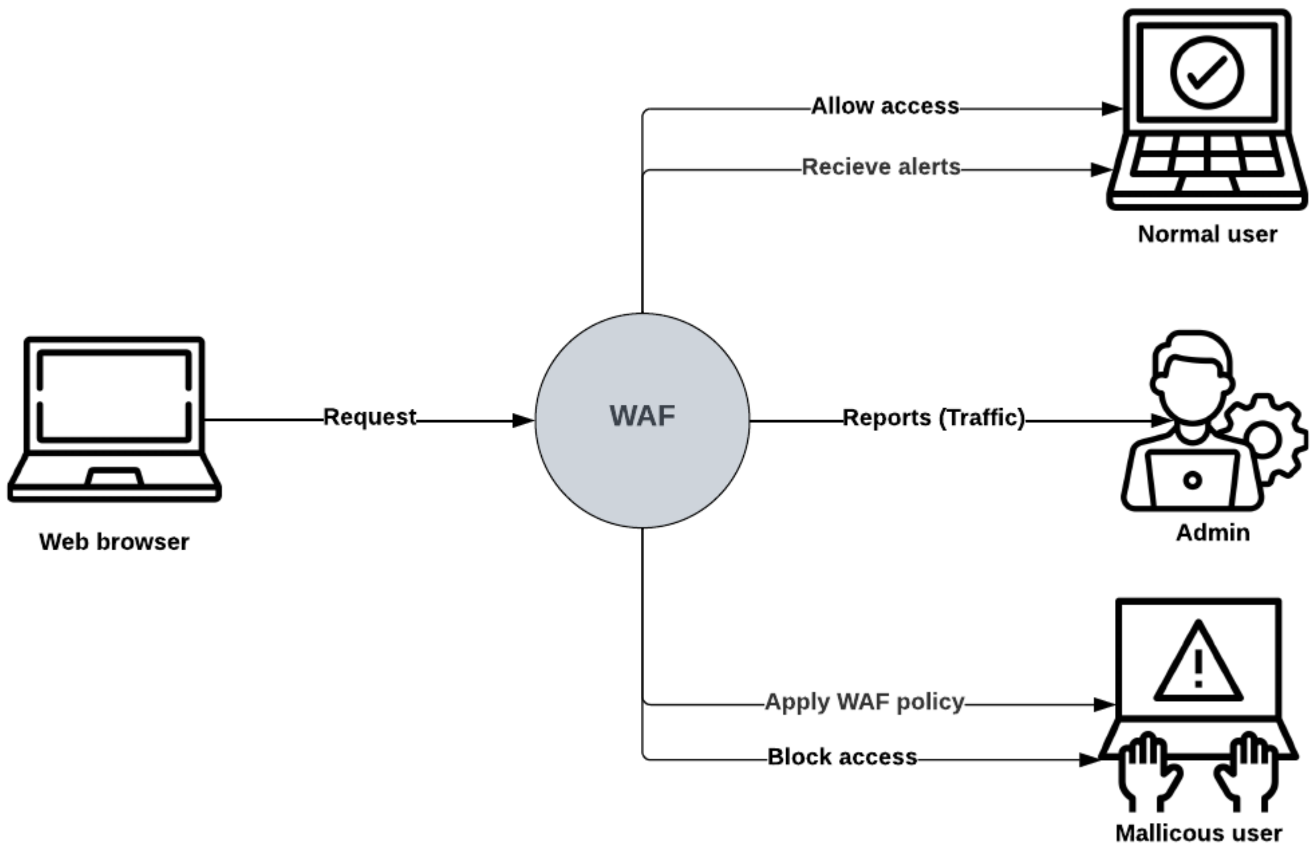


Figure 3: WAF system context

3.5 Objectives

- Addresses the limitations of traditional WAFs in detecting known, new, and unknown attacks (the zero-day attacks).
- Proposes a layered architecture of WAF to improve the accuracy of threat detection.
- Proposes solutions to reduce the attack detection of overhead time using intelligent algorithms that can predict and prevent web attacks.
- Provides blocking or prevention from different classes of attack types patterns that consistently manage to bypass the WAFs.

3.6 User Characteristics

This Web Application Firewall (WAF) is designed to be used by both IT security experts and administrators with experience in web application security and with knowledge about viruses and requests in charge of maintaining web application security in organizations.

- IT Security Professionals: an in-depth understanding of common vulnerabilities, attacks and web application security.
- System Administrators: knowledge of network infrastructure, web server configuration and deep knowledge of administration tasks.
- DevOps Teams: Collaboration between development and operation for deployment practices.
- IT Managers: Strategic thinking for implementing security solutions and overall IT security.

4 Functional Requirements

4.1 System Functions

- **ID 01:** The user shall request to access a web application.
- **ID 02:** The system shall be able to detect normal requests.
- **ID 03:** The system shall be able to detect malicious requests.
- **ID 04:** The system shall be able to monitor and filter the traffic going in and out of the system.
- **ID 05:** The system should implement access control to prevent unauthorized access to web applications.
- **ID 06:**The system should be designed to prevent attacks like SQL injection, and cross-site scripting (XSS).
- **ID 07:**The system shall be able to allow normal requests/users to access the web application they requested.
- **ID 08:** The system should be able to create detailed reports of web traffic logs and security events for analysis.
- **ID 09:** The system shall allow custom rules/policies to be created by the admin to define what is allowed and what is rejected.
- **ID 10:** The system should be able to protect the user authentication process to avoid unauthorized access.
- **ID 11:** The system should provide mechanisms for responding to security incidents, such as the ability to block malicious IPs or URLs.
- **ID 12:** The system should be consistent and integrate effectively and seamlessly with other security tools and systems to ensure a comprehensive security strategy.
- **ID 13:** The system shall continuously monitor and examine web traffic for potential security threats.
- **ID 14:** The system shall allow admins to create specific rules to address unique application vulnerabilities.
- **ID 15:**The admin shall receive reports about traffic for the system.
- **ID 16:** The admin shall be able to set policies to allow and block requests.

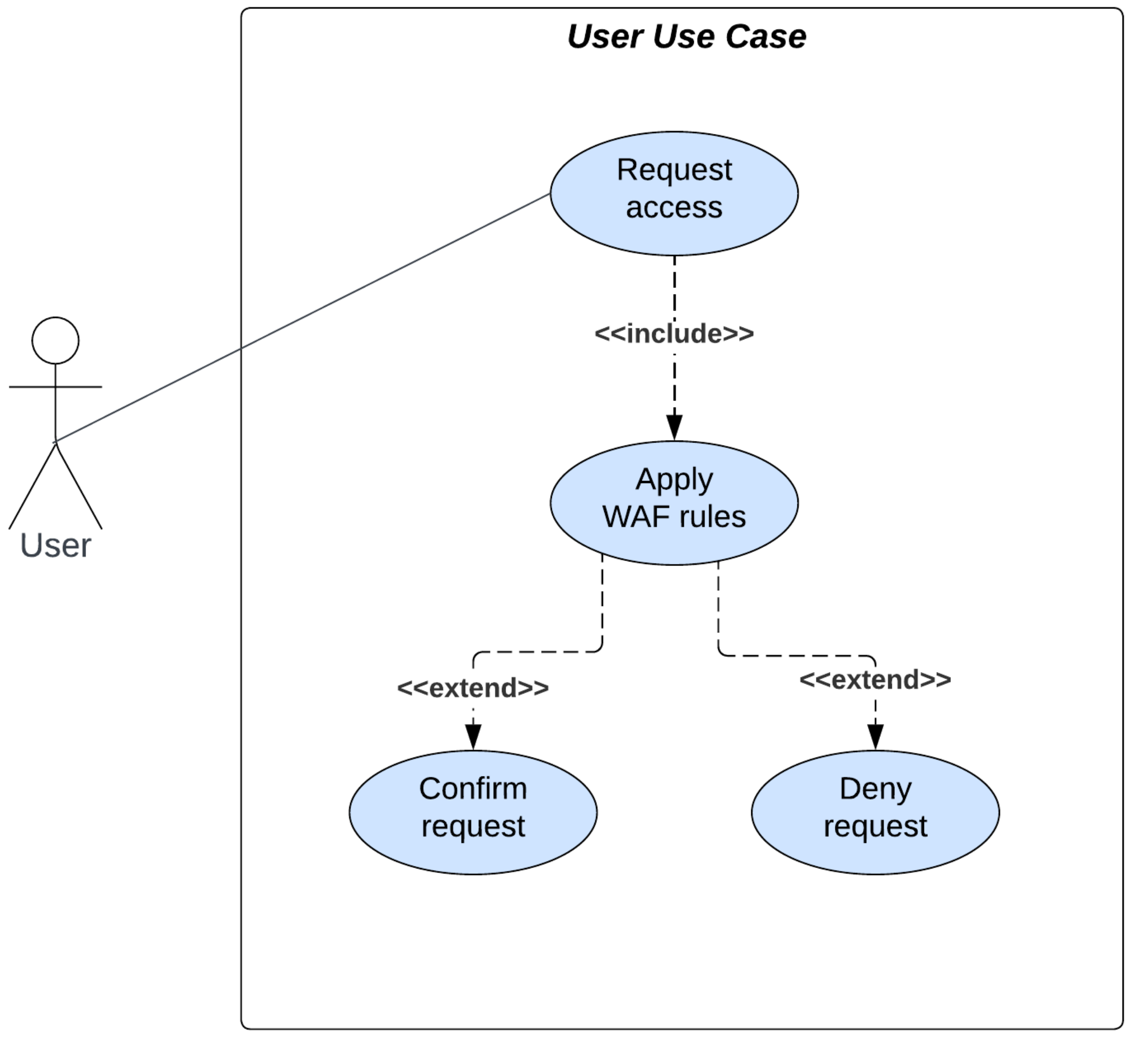


Figure 4: User Use Case

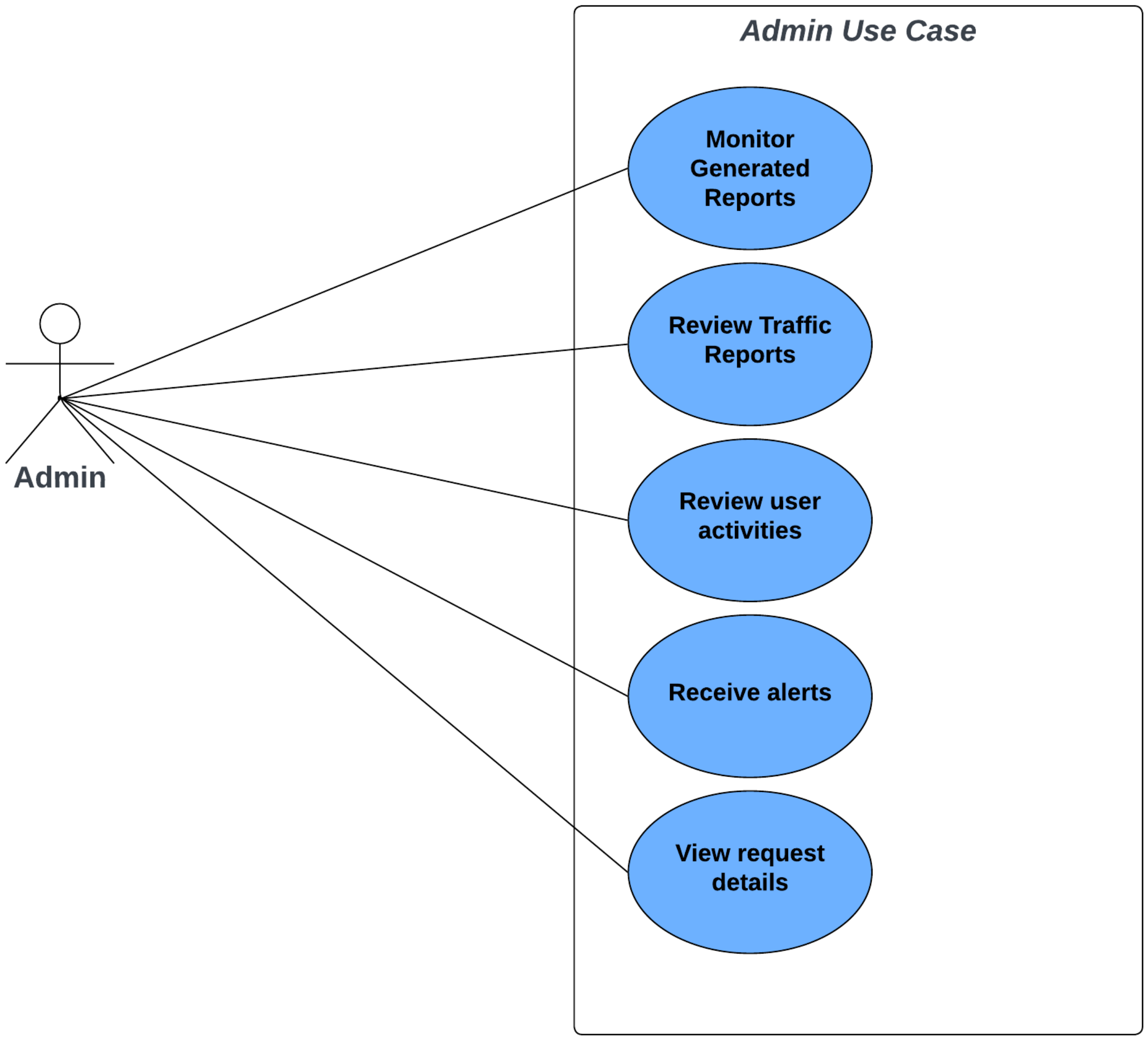


Figure 5: Admin Use Case

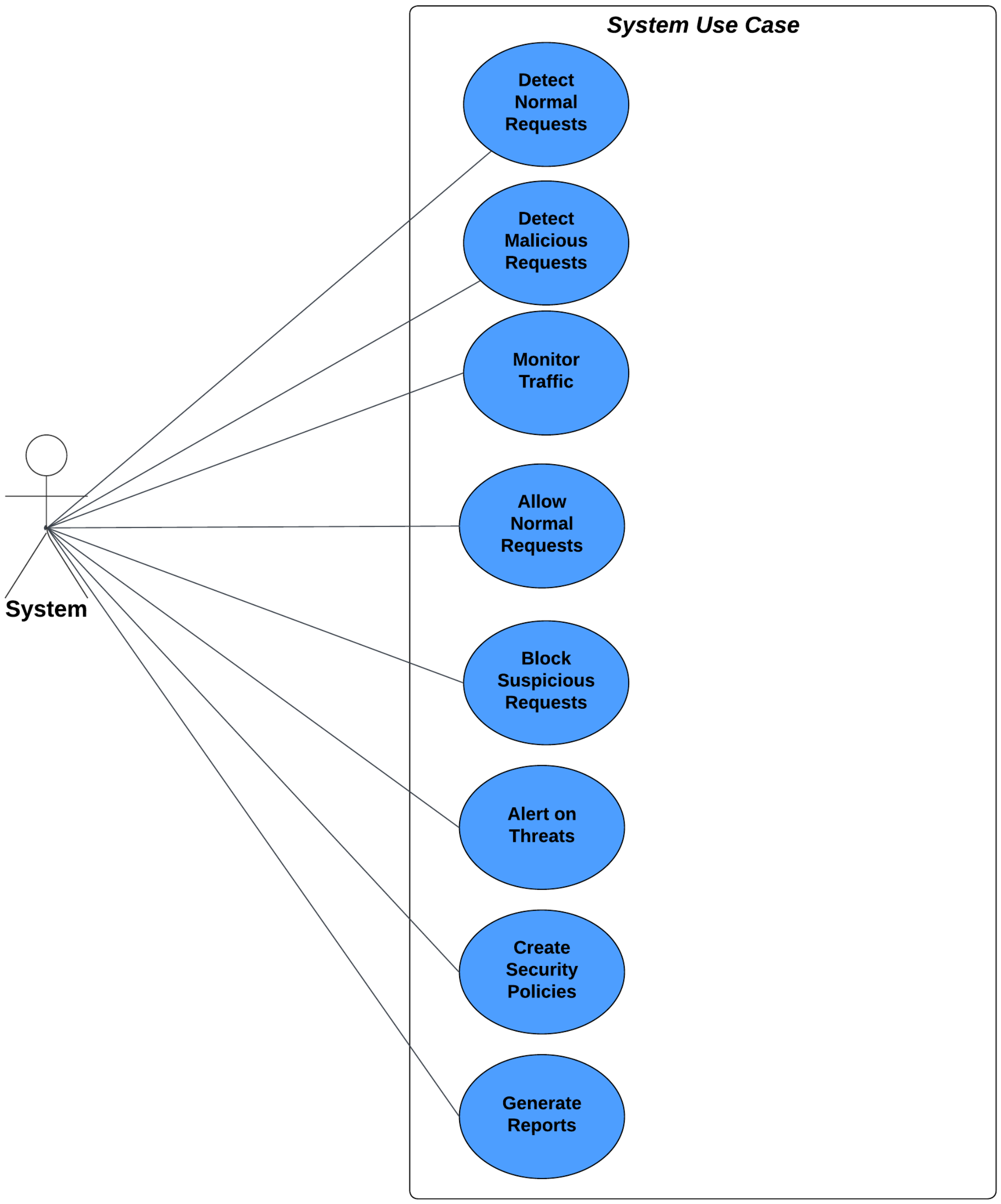


Figure 6: System Use Case

4.2 Detailed Functional Specification

Name	Normal Request
Code	ID06
Priority	Low
Critical	-
Description	The user sends a request to access a web application.
Input	Requesting a web application.
Output	Authorized access to a web application.
Pre-Condition	User must have a laptop and internet connection to send a request.
Post-Condition	-
Dependency	-
Risk	False positive and False negative may occur.

Table 2: Normal Request

Name	Malicious Request
Code	ID07
Priority	High
Critical	-
Description	The user sends a request to access a web application.
Input	Requesting a web application.
Output	Unauthorized access to a web application.
Pre-Condition	User must have a laptop and internet connection to send a request.
Post-Condition	The system will block the malicious request received.
Dependency	-
Risk	Zero-day attacks may still happen with no updates for the rules.

Table 3: Malicous Request

Name	Receive alerts
Code	ID15
Priority	High
Critical	-
Description	Receiving alerts as a report for any malicious request, sent on the web application to the admin to be updated.
Input	A user requests to access a web application maliciously.
Output	Unauthorized access to a web application and admin receive a threat alert.
Pre-Condition	Define thresholds for various types of attacks or suspicious activities.
Post-Condition	Send notifications to administrators.
Dependency	Depends on a malicious request being sent by the user.
Risk	-

Table 4: Allow normal requests

Name	Request analysis and filtering
Code	ID10
Priority	High
Critical	-
Description	Analyze incoming requests and filter out malicious ones.
Input	Incoming requests by the user.
Output	Filtered requests of the user if found malicious.
Pre-Condition	Receive incoming requests for analysis.
Post-Condition	Filtering the malicious requests.
Dependency	Depends on a request being sent by the user.
Risk	Possibility that the filter may pass by a malicious request.

Table 5: Request analysis and filtering

5 Design Constraints

5.1 Standards Compliance

The Web Application Firewall (WAF) runs on computers to protect web applications from various online threats and it usually runs at the server level or as part of the network infrastructure instead of running within web browsers.

5.2 Hardware Limitations

The user or company using WAF must have a computer that has an efficient CPU to handle complex security rules and inspect the traffic in real-time; as more powerful CPUs can process requests more quickly and efficiently. Moreover, adequate memory (RAM) is required to store and retrieve security rules, session data, and other information used during the inspection process. In addition to that, the number and speed of network interfaces on the WAF affect its ability to handle incoming and outgoing traffic, so using a high-speed network interface card is essential to avoid becoming a bottleneck in the network.

6 Non-Functional Requirements

6.1 Security

- Accuracy: Indicate the degree of accuracy in identifying and stopping malicious activity without blocking authorized traffic.
- Effectiveness: Indicate the level of protection the WAF needs to offer against typical web-based application failures and attacks.

6.2 Usability

- Configuration and Management: Indicate how simple it is for administrators to set up and maintain the WAF configuration.
- Logging and Reporting: Specify what has to be included for the logs and reports that the WAF produces to be understandable and clear.

6.3 Adaptability

- Flexible WAF to accommodate changing web applications and new attacks.

6.4 Availability

- Indicate the proportion of time that the WAF should be up and running for handling requests without any interruptions.

7 Data Design

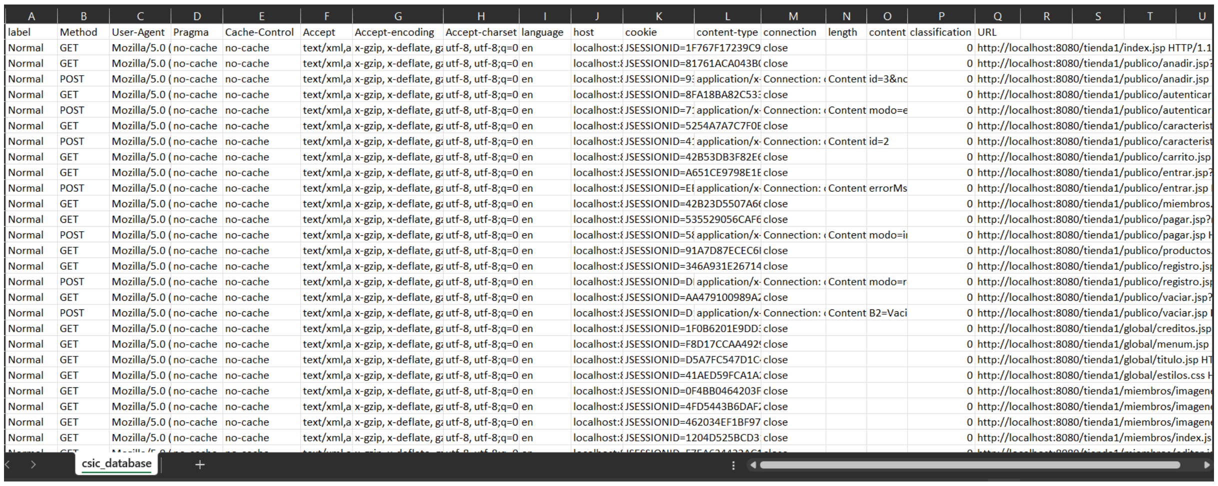
Two datasets were used:

- The first dataset is the same one utilized for all of the research publications.**The utilized dataset used is: HTTP dataset CSIC 2010**[3] [6] [4], This dataset was produced by the Spanish Research National Council's "Information Security Institute" (CSIC). Over 25,000 fraudulent or hazardous requests and 36,000 normal or safe requests make up the automatically generated dataset. The dataset contains numerous attacks, including server-side include, parameter manipulation, CRLF injection, SQL injection, buffer overflow, information collecting, file disclosure, and XSS. Three types of harmful or unexpected requests were taken into consideration in this dataset:

  1. **Static attacks** [Such attacks attempt to obtain resources that are hidden or nonexistent, such as configuration files, default files, the session ID in URL rewrites, outdated files, etc.]
  2. **Dynamic attacks** [These types of attacks alter legitimate request parameters, such as buffer overflows, cross-site scripting, SQL injection, and CRLF injection.]
  3. **Unintentional illegal requests** [These requests are not harmful, but they lack the web service's usual protocol and fail to have the same format as regular parameter values (a phone number composed of letters, for instance)].

  Tools like Paros and W3AF were used to build each attack in this dataset. The dataset is finally divided into three components. For the training phase, there is a single subset that only includes normal traffic. as well as two test subgroups: one for legitimate traffic and the other for spam traffic.

  More datasets will be used later on if needed and in case this dataset is insufficient. And regarding the newly discovered attacks (zero-day attacks) have no dataset, and patterns will be used to discover them.

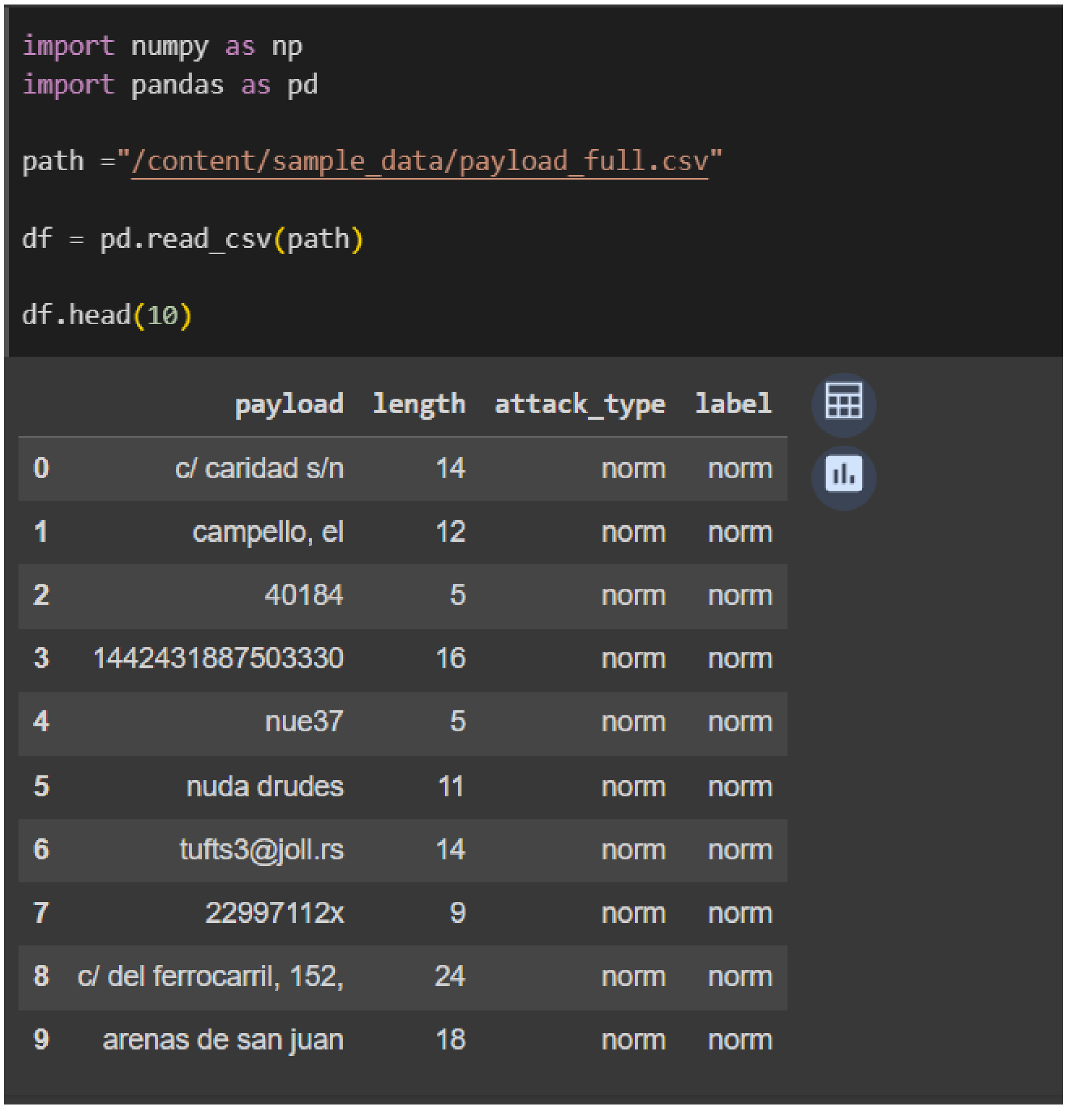


label	Method	User-Agent	Pragma	Cache-Control	Accept	Accept-encoding	Accept-charset	language	host	cookie	content-type	connection	length	content	classification	URL
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=1F767F17239C9	close				0	http://localhost:8080/tienda1/index.jsp HTTP/1.1
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=81761ACA043B	close				0	http://localhost:8080/tienda1/publico/anadir.jsp?
Normal	POST	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=9	application/x-	Connection:	Content	id=3&nc	0	http://localhost:8080/tienda1/publico/anadir.jsp
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=8FA18BA82C53	close				0	http://localhost:8080/tienda1/publico/autenticar
Normal	POST	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=7	application/x-	Connection:	Content	modo=e	0	http://localhost:8080/tienda1/publico/autenticar
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=5254A7A7C7F0E	close				0	http://localhost:8080/tienda1/publico/caracterist
Normal	POST	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=4	application/x-	Connection:	Content	id=2	0	http://localhost:8080/tienda1/publico/caracterist
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=42B53DB3F82E6	close				0	http://localhost:8080/tienda1/publico/carrito.jsp
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=A651CE9798E1E	close				0	http://localhost:8080/tienda1/publico/entrar.jsp?
Normal	POST	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=E	application/x-	Connection:	Content	errorMs	0	http://localhost:8080/tienda1/publico/entrar.jsp
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=42B23D5507A6	close				0	http://localhost:8080/tienda1/publico/miembros.
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=535529056CAF6	close				0	http://localhost:8080/tienda1/publico/pagar.jsp?
Normal	POST	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=5	application/x-	Connection:	Content	modo=i	0	http://localhost:8080/tienda1/publico/pagar.jsp H
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=91A7D87ECEC6	close				0	http://localhost:8080/tienda1/publico/productos.
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=346A931E26714	close				0	http://localhost:8080/tienda1/publico/registro.js
Normal	POST	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=D	application/x-	Connection:	Content	modo=r	0	http://localhost:8080/tienda1/publico/registro.js
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=AA479100989A	close				0	http://localhost:8080/tienda1/publico/vaciar.jsp?
Normal	POST	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=D	application/x-	Connection:	Content	B2=Vaci	0	http://localhost:8080/tienda1/publico/vaciar.jsp
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=1F0B6201E9DD	close				0	http://localhost:8080/tienda1/global/creditos.jsp
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=F8D17CCAA492	close				0	http://localhost:8080/tienda1/global/menum.jsp
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=D5A7FC547D1C	close				0	http://localhost:8080/tienda1/global/titulo.jsp HT
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=41AED59FCA1A	close				0	http://localhost:8080/tienda1/global/estilos.css H
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=0F4BB0464203F	close				0	http://localhost:8080/tienda1/miembros/imagen
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=4FD5443B6DAF	close				0	http://localhost:8080/tienda1/miembros/imagen
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=462034EF1BF97	close				0	http://localhost:8080/tienda1/miembros/imagen
Normal	GET	Mozilla/5.0 (	no-cache	no-cache	text/xml,a	x-gzip, x-deflate, g	utf-8, utf-8;q=0	en	localhost:8	JSESSIONID=1204D525BCD3	close				0	http://localhost:8080/tienda1/miembros/index.js

csic_database

Figure 7: CSIC 2010 dataset

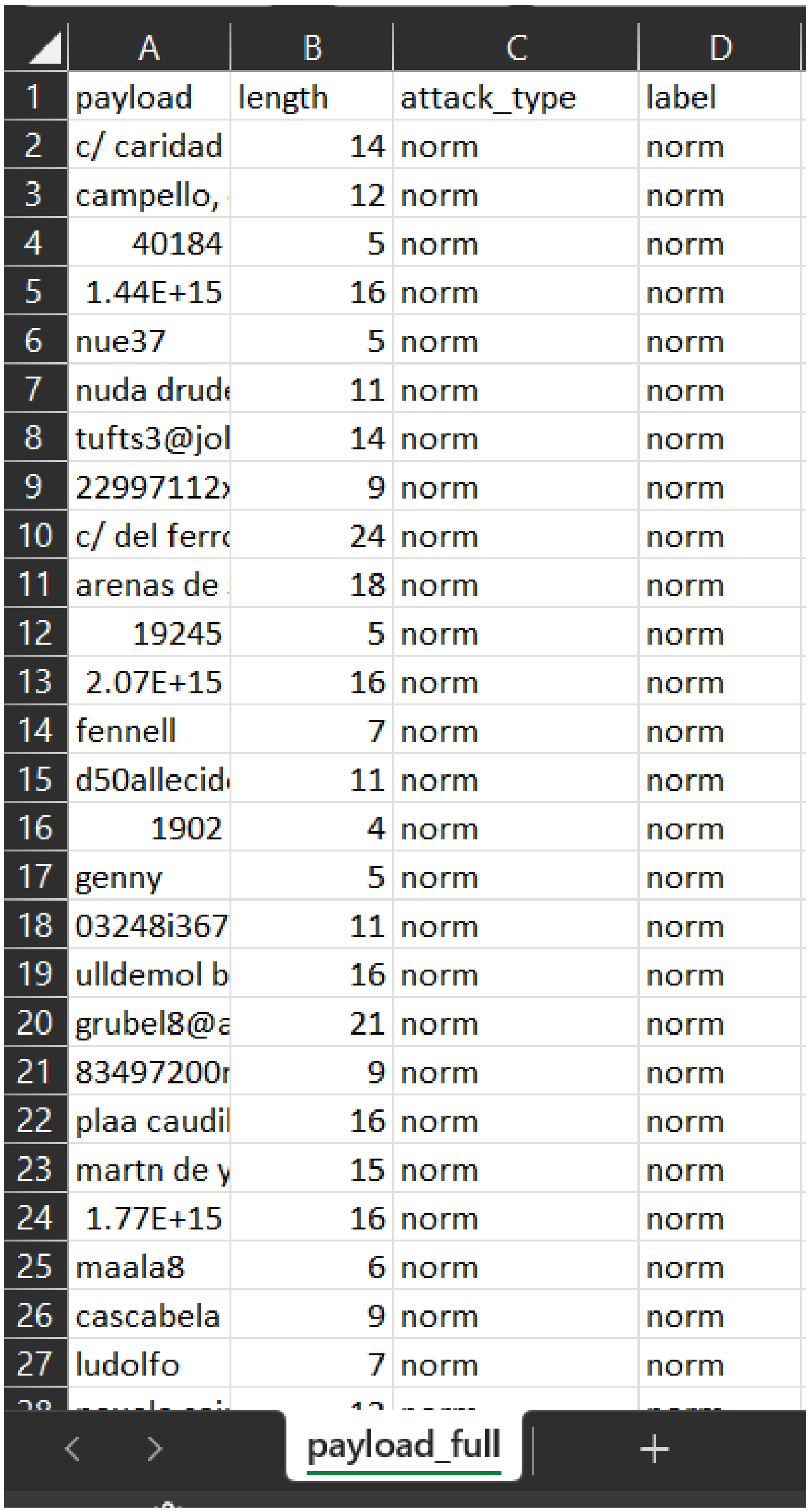
- The second dataset is **FWAF** which is a Web-Application Firewall dataset which is generated from HTTP traffic recorded by the Web Attacks Firewall (WAF). The dataset includes both benign traffic and the most recent examples of typical attacks that are relevant to real-world situations. Through domain merging, the dataset consists of a mix of URLs from many domains, including names, source paths, and attribute keys. Two categories were used to classify the data: normal and anomalous. Roughly 1,290,000 normal requests and 48,000 anomalous requests make up the approximately 1,350,000 records that have been collected and processed for the CSV structure format. Several attack types, including cross-site scripting (XSS) and SQL injection (SQLI), are among the unusual requests. The dataset contains more than one CSV file. The file called payload full is the one that was used for training and testing using machine learning algorithms and it contains 31k requests around 9k normal requests and 22k malicious requests.[11]



```
import numpy as np
import pandas as pd

path ="/content/sample_data/payload_full.csv"

df = pd.read_csv(path)

df.head(10)
```

```
                 payload  length  attack_type  label
0          c/ caridad s/n      14         norm   norm
1            campello, el      12         norm   norm
2                   40184       5         norm   norm
3        1442431887503330      16         norm   norm
4                   nue37       5         norm   norm
5             nuda drudes      11         norm   norm
6           tufts3@joll.rs      14         norm   norm
7               22997112x       9         norm   norm
8  c/ del ferrocarril, 152,      24         norm   norm
9       arenas de san juan      18         norm   norm
```

Figure 8: FWAF preprocessing



	A	B	C	D
1	payload	length	attack_type	label
2	c/ caridad	14	norm	norm
3	campello,	12	norm	norm
4	40184	5	norm	norm
5	1.44E+15	16	norm	norm
6	nue37	5	norm	norm
7	nuda drud	11	norm	norm
8	tufts3@jol	14	norm	norm
9	22997112	9	norm	norm
10	c/ del ferr	24	norm	norm
11	arenas de	18	norm	norm
12	19245	5	norm	norm
13	2.07E+15	16	norm	norm
14	fennell	7	norm	norm
15	d50allecid	11	norm	norm
16	1902	4	norm	norm
17	genny	5	norm	norm
18	03248i367	11	norm	norm
19	ulldemol b	16	norm	norm
20	grubel8@a	21	norm	norm
21	83497200	9	norm	norm
22	plaa caudil	16	norm	norm
23	martn de y	15	norm	norm
24	1.77E+15	16	norm	norm
25	maala8	6	norm	norm
26	cascabela	9	norm	norm
27	ludolfo	7	norm	norm

Figure 9: FWAF dataset

8 Preliminary Object-Oriented Domain Analysis

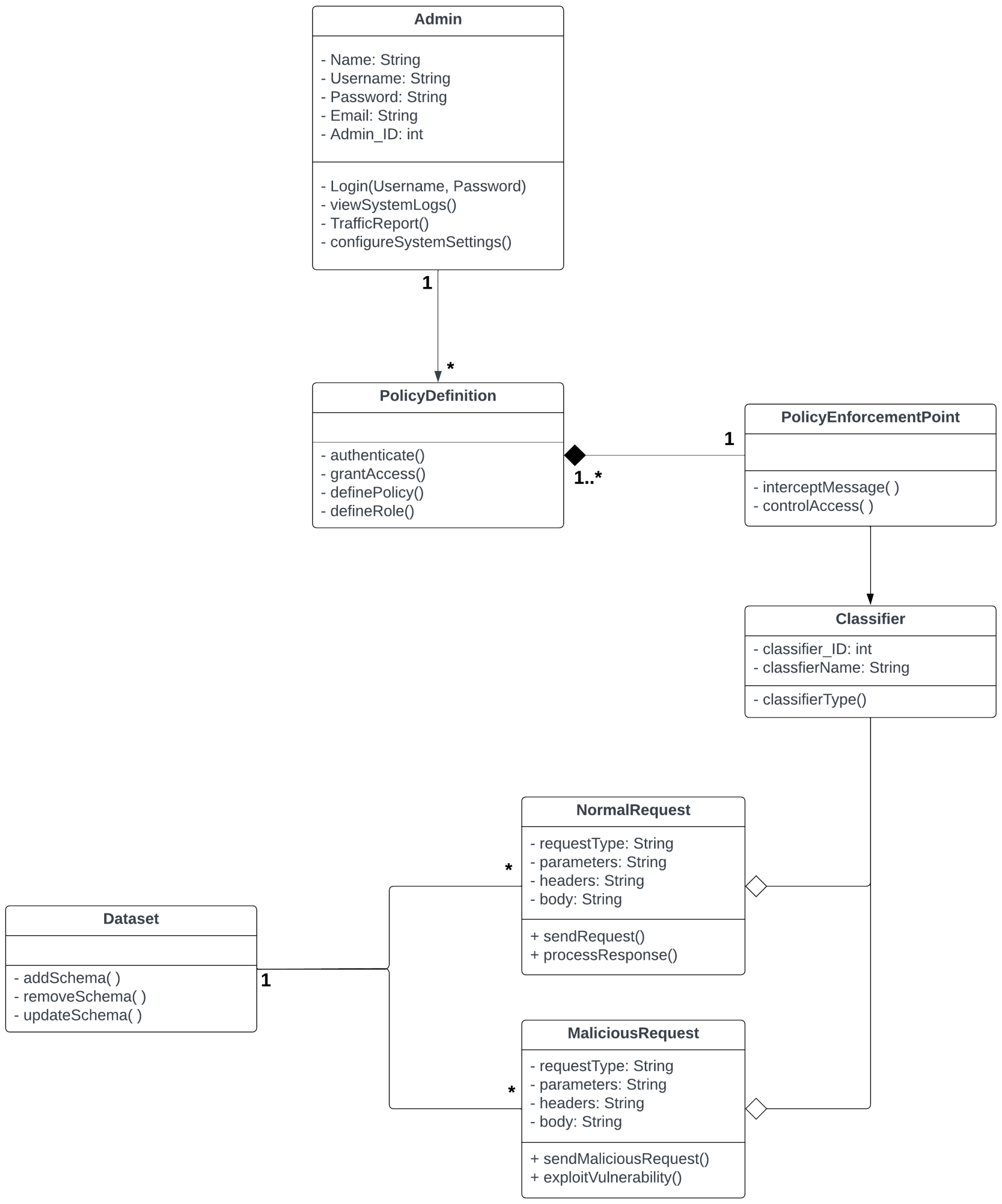


Figure 10: UML Class Diagram

9 Operational Scenarios

9.1 Normal Request Scenario

- **Initial Assumption:** WAF accepts the incoming HTTP/HTTPS request as safe with no malicious intents.
- **Normal Flow of Events:** Legitimate requests are permitted to bypass the firewall and reach the web application destination server.
- **What can go wrong:** False positives and false negatives may occur. **False positive:** The WAF may incorrectly allow malicious requests and thus block normal/valid users. **False negative:** The WAF may incorrectly disallow normal/legitimate requests and thus allow malicious users.

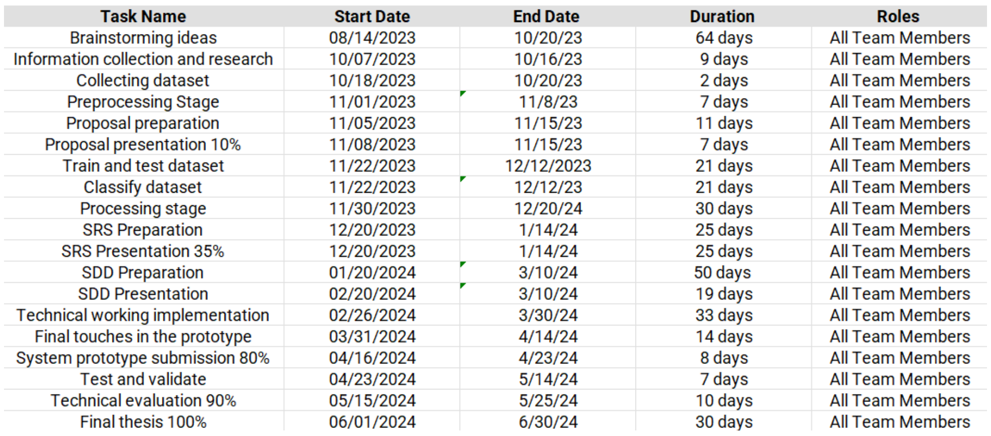
9.2 Malicious Request Scenario

- **Initial Assumption:** WAF assumes that the incoming requests are malicious or contain malicious content and thus, prevents them from accessing the web application.
- **Normal Flow of Events:** Suspicious or malicious requests which may have security threats such as SQL injection, XSS or any other common web application attacks are blocked by the firewall and cannot reach the destination server or access the web application.
- **What can go wrong:** Malicious actors may use complex approaches to evasion to avoid detection by the WAF, exploiting vulnerabilities and introducing malicious payloads into the web application; Or a new or previously unknown attack (zero-day attacks) may happen and WAF may not be updated with rules to detect and block such type of attacks immediately.

9.3 Empty Request Scenario

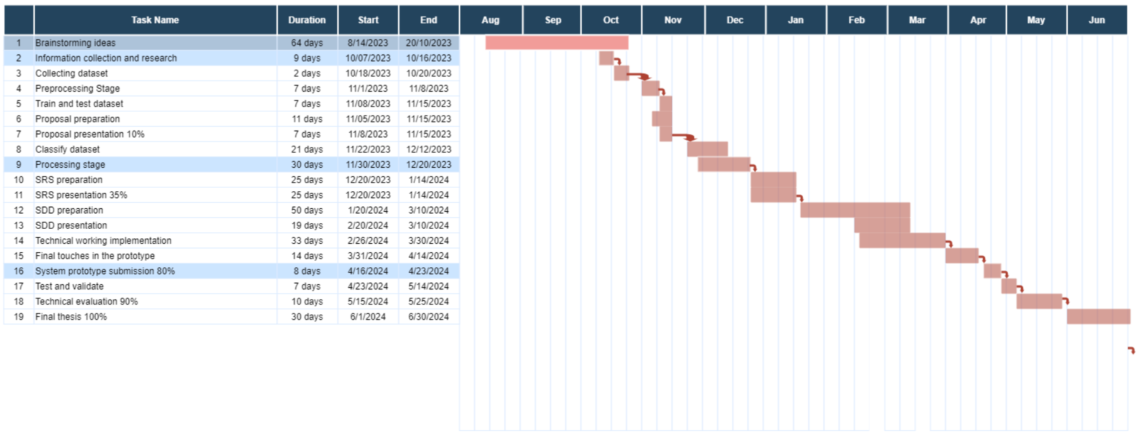
- **Initial Assumption:** If an empty request is detected by WAF, it will respond with an error to inform that the request is lacking necessary data or is invalid to prevent any further malicious action from that empty request.
- **Normal Flow of Events:** The WAF ensures the completeness and correctness of incoming requests, so if the request is empty; an error will be issued as the data is invalid or not meaningful thus, the request is not allowed to bypass the firewall.
- **What can go wrong:** Attackers may purposefully send empty or minimal data requests in an attempt to overwhelm the WAF and waste its resources, perhaps resulting in a denial of service attack.

10 Project Plan



Task Name	Start Date	End Date	Duration	Roles
Brainstorming ideas	08/14/2023	10/20/23	64 days	All Team Members
Information collection and research	10/07/2023	10/16/23	9 days	All Team Members
Collecting dataset	10/18/2023	10/20/23	2 days	All Team Members
Preprocessing Stage	11/01/2023	11/8/23	7 days	All Team Members
Proposal preparation	11/05/2023	11/15/23	11 days	All Team Members
Proposal presentation 10%	11/08/2023	11/15/23	7 days	All Team Members
Train and test dataset	11/22/2023	12/12/2023	21 days	All Team Members
Classify dataset	11/22/2023	12/12/23	21 days	All Team Members
Processing stage	11/30/2023	12/20/24	30 days	All Team Members
SRS Preparation	12/20/2023	1/14/24	25 days	All Team Members
SRS Presentation 35%	12/20/2023	1/14/24	25 days	All Team Members
SDD Preparation	01/20/2024	3/10/24	50 days	All Team Members
SDD Presentation	02/20/2024	3/10/24	19 days	All Team Members
Technical working implementation	02/26/2024	3/30/24	33 days	All Team Members
Final touches in the prototype	03/31/2024	4/14/24	14 days	All Team Members
System prototype submission 80%	04/16/2024	4/23/24	8 days	All Team Members
Test and validate	04/23/2024	5/14/24	7 days	All Team Members
Technical evaluation 90%	05/15/2024	5/25/24	10 days	All Team Members
Final thesis 100%	06/01/2024	6/30/24	30 days	All Team Members

Figure 11: Time Plan Of The Proposed System



	Task Name	Duration	Start	End
1	Brainstorming ideas	64 days	8/14/2023	20/10/2023
2	Information collection and research	9 days	10/07/2023	10/16/2023
3	Collecting dataset	2 days	10/18/2023	10/20/2023
4	Preprocessing Stage	7 days	11/1/2023	11/8/2023
5	Train and test dataset	7 days	11/08/2023	11/15/2023
6	Proposal preparation	11 days	11/05/2023	11/15/2023
7	Proposal presentation 10%	7 days	11/8/2023	11/15/2023
8	Classify dataset	21 days	11/22/2023	12/12/2023
9	Processing stage	30 days	11/30/2023	12/20/2023
10	SRS preparation	25 days	12/20/2023	1/14/2024
11	SRS presentation 35%	25 days	12/20/2023	1/14/2024
12	SDD preparation	50 days	1/20/2024	3/10/2024
13	SDD presentation	19 days	2/20/2024	3/10/2024
14	Technical working implementation	33 days	2/26/2024	3/30/2024
15	Final touches in the prototype	14 days	3/31/2024	4/14/2024
16	System prototype submission 80%	8 days	4/16/2024	4/23/2024
17	Test and validate	7 days	4/23/2024	5/14/2024
18	Technical evaluation 90%	10 days	5/15/2024	5/25/2024
19	Final thesis 100%	30 days	6/1/2024	6/30/2024

Figure 12: Time Plan Chart

11 Appendices

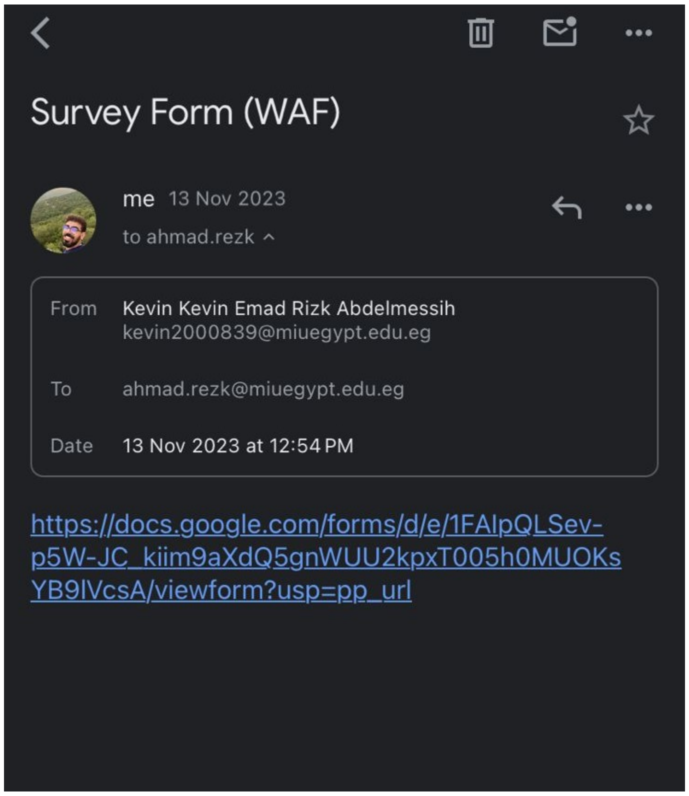
11.1 Definitions, Acronyms, Abbreviations

Term	**Stands for**
WAF	Web Application Firewall
ML	Machine Learning
DL	Deep Learning
XSS	Cross-Site Scripting
SQLI	Structured Query Language Injection
CNN	Convolutional Neural Network
LSTM	Long short-term memory
RNN	Recurrent Neural Network
BERT	Bidirectional Encoder Representations from Transformers
CSRF	Cross-site Request Forgery
CRLF injection	Carriage Return and Line Feed injection

Table 6: Abbreviations Table

11.2 Supportive Documents

To have more reliable responses for our WAF project, we have contacted the University IT director Mr.Ahmad Rezk to fill out the survey. The survey was sent through an email as shown in Figure [13].



Survey Form (WAF)

me 13 Nov 2023
to ahmad.rezk

From Kevin Kevin Emad Rizk Abdelmessih
kevin2000839@miuegypt.edu.eg

To ahmad.rezk@miuegypt.edu.eg

Date 13 Nov 2023 at 12:54 PM

https://docs.google.com/forms/d/e/1FAIpQLSev-p5W-JC_kiim9aXdQ5gnWUU2kpxT005h0MUOKsYB9lVcsA/viewform?usp=pp_url

Figure 13: Email Snapshot

References

[1] Tomás Sureda Riera, Juan Ramón Bermejo Higuera, Javier Bermejo-Higuera, et al. “Systematic Approach for Web Protection Runtime Tools’ Effectiveness Analysis”. In: (2022).

[2] Biagio Montaruli, Luca Demetrio, Andrea Valenza, et al. “Adversarial ModSecurity: Countering Adversarial SQL Injections with Robust Machine Learning”. In: *arXiv preprint arXiv:2308.04964* (2023).

[3] Babu R Dawadi, Bibek Adhikari, and Devesh Kumar Srivastava. “Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks”. In: *Sensors* 23.4 (2023), p. 2073.

[4] Simon Applebaum, Tarek Gaber, and Ali Ahmed. “Signature-based and machine-learning-based web application firewalls: A short survey”. In: *Procedia Computer Science* 189 (2021), pp. 359–367.

[5] Maha Alghawazi, Daniyal Alghazzawi, and Suaad Alarifi. “Deep Learning Architecture for Detecting SQL Injection Attacks Based on RNN Autoencoder Model”. In: *Mathematics* 11.15 (2023), p. 3286.

[6] Yunus Emre Seyyar, Ali Gökhan Yavuz, and Halil Murat Ünver. “An attack detection framework based on BERT and deep learning”. In: *IEEE Access* 10 (2022), pp. 68633–68644.

[7] Stefan Prandl, Mihai Lazarescu, and Duc-Son Pham. “A study of web application firewall solutions”. In: *Information Systems Security: 11th International Conference, ICISS 2015, Kolkata, India, December 16-20, 2015. Proceedings 11*. Springer. 2015, pp. 501–510.

[8] Nikita Gandotra and Lalit Sen Sharma. “Exploring the use of Iptables as an Application Layer Firewall”. In: *Journal of The Institution of Engineers (India): Series B* 101 (2020), pp. 707–715.

[9] Se-Joon Park, Yong-Joon Lee, and Won-Hyung Park. “Configuration Method of AWS Security Architecture That Is Applicable to the Cloud Lifecycle for Sustainable Social Network”. In: *Security and Communication Networks* 2022 (2022), pp. 1–12.

[10] Stefan Boneder. “Evaluation and comparison of the security offerings of the big three cloud service providers Amazon Web Services, Microsoft Azure and Google Cloud Platform”. PhD thesis. Technische Hochschule Ingolstadt, 2023.

[11] Van-Hau Pham, Hoang Khoa Nghi, Huu Quyen Nguyen, et al. “Deception and Continuous Training Approach for Web Attack Detection using Cyber Traps and MLOps”. In: *VNUHCM Journal of Science and Technology Development* 26.2 (2023), pp. 2729–2740.