

وزارة التعليم العالي والبحث العلمي

Université
Abdelhafid Boussouf - Mila
Abdelhafid Boussouf
University of Mila



جامعة
عبد الحفيظ بوالصوف – ميله –

Institute of Mathematics and Computer Science

Department of Computer Science

Master Thesis

Field : Mathematics and Computer Science

Option : Computer Science

Specialization : Information and communication technologies

Theme

An AI-powered self-defence system for Web Applications

Submitted by : Boulbene Fareh & Bouakline Yahia

Jury members

Mr Hattab Abdelkamel	MCB , University of Mila	President
Mrs Boumassata Meriem	MAA , University of Mila	examiner
Mr Abderrahim Dib	MAA , University of Mila	Advisor

Dedication

We dedicate this work to our institution, the University of Abdehafid Boussouf Mila, and we hope that it will constitute an added value to the library of our department and contribute to the enrichment of future works.

I would like to express my deep gratitude to all my dear teachers, without any exception, from the first year of the Bachelor's degree to the second year of the Master's degree, to whom I owe the essence of my success. Their dedication, sacrifices, and constant support have been the solid foundation of my academic journey.

My sincere thanks also go to my supervisor, Dib Abderrahim, for his valuable guidance, patience, and continuous support.

I also do not forget my classmates, who made this journey lighter and more motivating through their kindness, mutual support, and the pleasant atmosphere they created every day.

Acknowledgements

We would like to thank all the heads and staff of the Department of Science and Technology at the University Center of Mila for the education and training provided.

Our sincere thanks go to our supervisor, Dib Abderrahim, for his valuable advice and guidance.

We also thank the members of the jury, Mr.Hattab Abdelkamel and Mrs. Boumassata Meriem , for dedicating their time to reading, evaluating, and reviewing the content of this work.

Finally, we would like to thank everyone who contributed, directly or indirectly, to the completion of this dissertation.

Résumé

Avec le développement rapide des applications web, les sites internet sont devenus des cibles fréquentes pour différentes attaques informatiques. Les méthodes de sécurité traditionnelles restent utiles, mais elles ne sont plus toujours suffisantes face à des attaques de plus en plus complexes.

Ce mémoire présente la conception et le développement d'un système d'auto-défense pour applications web basé sur l'intelligence artificielle. L'objectif principal est d'exploiter les capacités de l'apprentissage automatique, une branche clé de l'IA, afin de détecter automatiquement les requêtes malveillantes avant qu'elles ne compromettent la sécurité de l'application. Dans ce cadre, nous étudions plusieurs types d'attaques web courantes, notamment les injections SQL, les attaques Cross-Site Scripting (XSS), les attaques Cross-Site Request Forgery (CSRF) ainsi que d'autres menaces ciblant les applications web.

Pour entraîner et évaluer les modèles d'intelligence artificielle, deux ensembles de données spécialisés ont été utilisés : SRBH-2020 et CSIC-ECML Combined Dataset. Les données ont subi différentes étapes de prétraitement, de nettoyage et de transformation afin de les rendre exploitables par les algorithmes de classification. Plusieurs modèles d'apprentissage automatique, notamment CatBoost, LightGBM et Random Forest, ont ensuite été implémentés et comparés afin d'identifier la solution la plus performante pour la détection des attaques.

Enfin, le modèle retenu a été intégré dans une application web de vente de matériel informatique afin de constituer un mécanisme de défense intelligent capable d'analyser les requêtes HTTP en temps réel et de distinguer automatiquement les requêtes légitimes des tentatives d'attaque. Les résultats expérimentaux obtenus démontrent que l'intégration de l'intelligence artificielle dans les applications web permet d'améliorer significativement la détection des menaces et de renforcer la sécurité globale du système.

Mots-clés :

Intelligence artificielle, apprentissage automatique, cybersécurité, applications web, détection d'intrusion, CatBoost, LightGBM, Random Forest, Logistique Régression , SR-BH2020 attaques web.

Abstract

With the rapid growth of web applications, websites have become frequent targets of various cyberattacks. Traditional security mechanisms remain useful; however, they are not always sufficient to cope with increasingly sophisticated threats.

This thesis presents the design and development of an AI-based self-defense system for web applications. The main objective is to leverage the capabilities of machine learning, a key branch of AI, to automatically detect malicious requests before they can compromise the security of the application. In this context, several common web attacks are investigated, including SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and other threats targeting web applications.

To train and evaluate the artificial intelligence models, two specialized datasets were used: SRBH-2020 and the CSIC-ECML Combined Dataset. The data underwent several preprocessing, cleaning, and transformation stages to make them suitable for classification algorithms. Several machine learning models, namely CatBoost, LightGBM, and Random Forest, were then implemented and compared in order to identify the most effective solution for attack detection.

Finally, the selected model was integrated into a web-based computer hardware sales application to establish an intelligent defence mechanism capable of analyzing HTTP requests in real time and automatically distinguishing legitimate requests from attack attempts. The experimental results demonstrate that the integration of artificial intelligence into web applications can significantly improve threat detection and enhance the overall security of the system.

Keywords:

Artificial Intelligence, Machine Learning, Web Security, Intrusion Detection, Web Applications, CatBoost, LightGBM, Random Forest, Logistic Regression , SRBH-2020, Cybersecurity.

الملخص

مع النمو السريع لتطبيقات الويب، أصبحت المواقع الإلكترونية أهدافاً متكررة لمختلف الهجمات السيبرانية. ورغم أن آليات الحماية التقليدية لا تزال مفيدة، إلا أنها لم تعد كافية دائماً لمواجهة التهديدات المتزايدة من حيث التعقيد. تقدم هذه الأطروحة تصميم وتطوير نظام دفاع ذاتي لتطبيقات الويب يعتمد على الذكاء الاصطناعي. الهدف الرئيسي هو تسخير قدرات تعلم الآلة (Machine Learning)، باعتباره فرعاً أساسياً من فروع الذكاء الاصطناعي، للكشف تلقائياً عن الطلبات الخبيثة قبل أن تُلحق ضرراً بأمن التطبيق. وفي هذا السياق، تم دراسة عدة أنواع من الهجمات الشائعة على الويب، بما في ذلك حقن SQL (SQL Injection)، والبرمجة النصية عبر المواقع (Cross-Site Scripting - XSS)، وتزوير الطلبات عبر المواقع (Cross-Site Request Forgery - CSRF)، وغيرها من التهديدات التي تستهدف تطبيقات الويب.

من أجل تدريب وتقييم نماذج الذكاء الاصطناعي، تم استخدام مجموعتين من البيانات المتخصصة: مجموعة بيانات-SRBH 2020 ومجموعة البيانات المدمجة CSIC-ECML. وقد خضعت هذه البيانات لعدة مراحل من المعالجة المسبقة والتنظيف والتحويل لجعلها مناسبة لخوارزميات التصنيف. بعد ذلك، تم تنفيذ عدة نماذج من تعلم الآلة، وهي CatBoost و Random Forest و LightGBM، ومقارنتها من أجل تحديد الحل الأكثر فعالية في كشف الهجمات.

وأخيراً، تم دمج النموذج المختار داخل تطبيق ويب خاص ببيع أجهزة الحاسوب، وذلك لإنشاء آلية دفاع ذكية قادرة على تحليل طلبات HTTP في الوقت الحقيقي والتميز تلقائياً بين الطلبات الشرعية ومحاولات الهجوم. وقد أظهرت النتائج التجريبية أن دمج الذكاء الاصطناعي في تطبيقات الويب يمكن أن يحسن بشكل كبير من اكتشاف التهديدات ويعزز أمن النظام بشكل عام.

الكلمات المفتاحية

الذكاء الاصطناعي، تعلم الآلة، أمن الويب، كشف التسلل، تطبيقات الويب، CatBoost، LightGBM، Random Forest، Logistic Regression، SRBH-2020، الأمن السيبراني

List of figures

Figure 1: Example of SQL injection	6
Figure 2: Dos / DDos	8
Figure 3: Brute Force	9
Figure 4: Reflected XSS	10
Figure 5: Stored XSS (persistante)	11
Figure 6: CSRF Attack	12
Figure 7: OWSP TOP 10 Attacks Distribution	13
Figure 8: Percentage of normal and malicious queries in the CSIC_ECML_ combined dataset	24
Figure 9: The number of attack types Before Pre-processing	33
Figure 10: Percentage of attack types After Pre-processing	34
Figure 11: Use Case of the Web Application.....	35
Figure 12: user interface of the application	36
Figure 13 : Creation a user account form.....	36
Figure 14: Comparison of Accuracy, Precision, Recall, and Macro F1-score of the Evaluated Models on the SRBH-2020 Dataset	49
Figure 15: Comparison of Logistic Regression and LightGBM Performance on the CSIC_ECML_combined Dataset	51
Figure 16: Architecture of the AI-based Web Self-Defense System	53
Figure 17: Meaning of the Information Displayed in the Prediction Logs	58

List of Tables

Table 1: Features of the 'CSIC_ECML_combined' dataset	16
Table 2: Distribution of attack categories in “dataset_capec_combine.csv”	18
Table 3: Distribution of attack categories in “dataset_capec_transfer.csv”	19
Table 4: The number of attack types Before Pre-processing	32
Table 5: The number of attack types After Pre-processing	33
Table 6: CatBoost Model Results for Multiclass Classification	41
Table 7: Global Evaluation Metrics of the CatBoost Model	41
Table 8: Random Forest Model Results for Multiclass Classification	42
Table 9: Global Evaluation Metrics of the Random Forest Model	43
Table10: LightGBM Performance Results for Multiclass Classification (SRBH-2020)	45
Table 11: LightGBM Global Performance Metrics (Multiclass Classification - SRBH-2020)	45
Table12: LightGBM Performance Results for Binary Classification (CSIC-ECML-combined)	46
Table 13: Global Performance Metrics of the LightGBM Model (Binary Classification - CSIC_ECML_combined)	46
Table 14: Logistic Regression Performance Results for Binary Classification on the CSIC_ECML_combined Dataset	47
Table 15: Global Evaluation Metrics of the Logistic Regression Model (Binary Classification - CSIC_ECML_combined)	48

Table 16: Comparison of Models on the SRBH-2020 Dataset (Multiclass Classification)	49
Table 17: Comparison of Models on the CSIC_ECML_combined Dataset (Binary Classification)	50
Table 18: Results for different http requests examples	58

Table of Contents

Résumé	D
Abstract	E
المخلص.....	F
List of figures	G
List of Tables	H
Table of Contents	J
Introduction	2
Chapter 1: Web security and IA	4
I- Web Security.....	4
2- Web attacks and OWASP	4
3- OWASP TOP 10	6
3.1 - SQL injection.....	6
3.2 - Denial-of-Service Attack / Distributed Denial-of-Service Attacks	7
3.3 - Brute-force attackAttacks	9
3.4 - Cross-Site Scripting (XSS) Attack.....	10
3.5 Cross-Site Request Forgery (CSRF)	12
4. Distribution of Web attacks	13
5. IA techniques for Web security	13
5.1 Machine Learning	13
5.2 The Use of Machine Learning in our Work	14
II. Web security datasets	15

1. Introduction	15
2- Presentation of the datasets	15
2.1 CSIC_ECML_combined Dataset:	15
2.2 Dataset SR-BH 2020.....	18
2.3 Comparison and Justification of the Datasets	19
Chapter 2	21
The design of IA model for Web application	21
1- Introduction	21
2- Data Pre-processing	22
2.1 CSIC_ECML_combined Dataset preprocessing	22
2.1.1 Data Cleaning of the “CSIC_ECML_combined” Dataset	22
2.1.2 Data Balancing of the “CSIC_ECML_combined” Dataset	23
2.1.3 Data Transformation of the “CSIC_ECML_combined” Dataset	24
2.1.4 Label Encoding of the “CSIC_ECML_combined” Dataset	26
2.1.5 Data Splitting of the “CSIC_ECML_combined” Dataset	27
2.2 SR-BH 2020 Dataset pre-processing	27
2.2.1 Dataset Merging	28
2.2.2 Data Cleaning of the SR-BH 2020 dataset	28
2.2.3 Class Balancing of the SR-BH 2020 Dataset	28
2.2.4 Dataset Splitting	29
2.2.5 Vectorization of Textual Data	29
2.2.6 Label encoding.....	31
3- Analysis of Class Distribution Before and After Pre-processing	31
4- Proposed AI-Based Web Security Architecture	34

5- Integration Strategy of the AI Model	37
Chapter 3 Implementation and Results	38
1-Introduction	38
2- Development Environment	38
2.1 Hardware Environment	38
2.2 Software Environment	39
3- Implementation and Evaluation of the Machine Learning Models	39
3.1 Implementation of the CatBoost Model	40
3.2 Implementation of the Random Forest model	41
3.3 Implementation of the LightGBM Model	43
3.3.1 Results on SRBH-2020 (Multiclass Classification)	44
3.3.2 LightGBM on CSIC_ECML_combined (Binary Classification)	46
3.4 Implementation of the Logistic Regression Model	46
4- Comparison of Models	48
4.1 Comparison of Models on the SRBH-2020 Dataset (Multiclass Classification)	48
4.2 Comparison of Models on the CSIC_ECML_combined Dataset (Binary Classification)	50
5- Implementation of the AI Model Integration in the Web Application	52
5.1 Global Request Interception	54
5.2 Detection Engine Architecture (Model and Vectorizer)	54
5.3 Communication and Normalization (Flask API)	54
5.4 Security Decision and Mitigation	55
5.5 Normal Request Processing	55

5.6 Targeted Controller-Level Integration (contacts.c.php)	56
5.6.1 Filtering Mechanism	56
5.6.2 Decision Logic and Security Response	56
6- Experimental Results and Performance Evaluation of the Integrated AI Security System	57
6.1 Overview of the Evaluation Process	57
6.2 Example of Real-Time Attack Detection	57
6.3 Analysis of Prediction Logs	58
6.4 Results for Different HTTP Request Scenarios	58
6.5 Discussion and Evaluation of System Performance	59
Conclusion and Future Work	61
References	63

INTRODUCTION

Introduction

With the rapid development of the Internet, web applications have become essential in several fields such as trading, services, and administration. However, this evolution has also led to a significant increase in security risks. Today, web applications are frequently targeted by various attacks such as SQL Injection, Cross-Site Scripting (XSS), and Distributed Denial-of-Service attacks. These threats generally exploit vulnerabilities related to poor input handling or configuration errors.

Traditional security solutions, such as firewalls or simple filtering systems, remain useful, but they are no longer always sufficient against modern attacks, which are becoming more intelligent and adaptive. For this reason, new approaches based on artificial intelligence have been introduced. In particular, machine learning makes it possible to analyze data and automatically detect abnormal behaviors while adapting to new forms of attacks.

In this context, our work consists of designing an intelligent system capable of detecting attacks in web applications using machine learning techniques. To achieve this, we use datasets that represent current attacks, which are prepared through different preprocessing steps. Then, we apply and compare several classification models in order to identify malicious requests. Finally, a simulation is carried out in a local environment to test the system under conditions close to reality.

To achieve these objectives, this thesis is organized into three main chapters.

The first chapter, entitled "Web Security and AI," presents the fundamental concepts of web security. We describe the main threats targeting web applications, particularly the attacks listed in the OWASP ranking, as well as their significance and frequency in 2025. We also present traditional protection methods used to secure web applications.

The second chapter is dedicated to the datasets used in our study. We introduce the CSIC-ECML Combined and SR-BH 2020 datasets, as well as the various features they contain. This chapter details the data preprocessing steps, specifically the use of the TF-IDF method for the textual representation of HTTP requests. Finally, we present our proposed architecture, the "AI-Based Web Security Architecture," designed to enable the integration of the artificial intelligence model into the developed web application.

The third chapter presents the experiments conducted and the results obtained. We evaluate the performance of several Machine Learning models—specifically Logistic Regression, Random Forest, LightGBM, and CatBoost—on both studied datasets. A detailed comparison of the results is performed to identify the most effective models. This chapter also describes the practical integration of the selected model into the SWSite_Version2 web application, as well as the results achieved after its deployment in a test environment that simulates real-world conditions.

Chapter 1

Web security and IA

I- Web Security

Web application security has become an essential element in today's digital world. With the widespread use of the Internet, websites and web applications are used in several fields such as e-commerce, social networks, banking, universities, and administrative services. These applications allow users to easily access information and online services. However, this extensive use has also increased the risks related to cyberattacks. [1]

Web security refers to all the techniques, methods, and tools used to protect web applications against threats and unauthorized access. Its main objective is to ensure the confidentiality, integrity, and availability of data. Confidentiality means that sensitive information should only be accessible to authorized users. Integrity guarantees that data is not modified illegally, while availability ensures that services remain accessible to users at all times.

Web applications may contain different vulnerabilities that allow attackers to exploit the system. Among the most well-known attacks are SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), as well as Denial-of-Service Attack and Distributed Denial-of-Service attacks. These attacks can cause data loss, theft of sensitive information, service interruptions, or even allow attackers to take control of certain parts of the system.

To reduce these risks, several protection mechanisms are used in modern web applications. For example, developers apply user input validation techniques, use secure authentication systems, encrypt sensitive data, and implement Web Application Firewall (WAFs). Despite this, attacks are becoming increasingly sophisticated, which sometimes makes traditional solutions insufficient.. [2] [3]

2- Web attacks and OWASP

Web attacks are malicious actions targeting web applications, web servers, or online services through the exploitation of security weaknesses. The main objectives of these attacks are to gain unauthorized access to sensitive information, alter data, disrupt services, or compromise the normal functioning of web systems. As web applications have become essential components of modern information systems, web attacks have emerged as one of

the most significant cybersecurity threats, affecting the confidentiality, integrity, and availability of digital resources.

OWASP, which stands for Open Worldwide Application Security Project, is an international organization specialized in web application security. This organization is non-profit and its main objective is to help developers, companies, and researchers improve the security of applications and computer systems. Today, OWASP is considered an important reference in the field of web cybersecurity.

One of the most well-known OWASP projects is called the OWASP Top 10. It is a list of the most dangerous and most common vulnerabilities found in web applications. This list is regularly updated in order to follow the evolution of cyber threats. Developers and security experts often use this classification to identify risks and strengthen the protection of web applications.

Among the vulnerabilities included in the OWASP Top 10 are SQL Injection, Cross-Site Scripting (XSS), authentication issues, security misconfigurations, and attacks related to access control. These vulnerabilities may allow attackers to access sensitive data, modify information, or disrupt the operation of web services.

OWASP also provides several free tools, guides, and documents intended to raise developers' awareness about security best practices. For example, the organization offers recommendations for securing source code, protecting user data, and testing applications against known vulnerabilities. Thanks to these resources, development teams can design applications that are more reliable and more resistant to attacks.

In the field of research and intelligent intrusion detection systems, OWASP also plays an important role. The attacks mentioned in the OWASP Top 10 are often used as references in cybersecurity datasets and in machine learning experiments. This allows artificial intelligence models to be trained to recognize the most common malicious behaviors in web applications.

Therefore, OWASP represents an essential resource for understanding the main web security threats and for developing more effective protection mechanisms against modern cyberattacks.

The vulnerabilities listed in the OWASP Top 10 represent some of the most common threats faced by modern web applications. Among these vulnerabilities, certain attacks are particularly widespread due to their impact on the confidentiality, integrity, and availability of information systems. To provide a structured understanding of the OWASP Top 10 vulnerabilities, the following sections present a detailed and systematic analysis of the most

significant web attacks. Each attack will be examined according to a common methodology covering its definition, exploitation mechanism, security impact, and possible countermeasures. The first vulnerability addressed is SQL Injection, due to its severity and frequent exploitation in real-world web applications. [4] [5]

3- OWASP TOP 10

3.1 - SQL injection:

SQL Injection, often called SQLi, is one of the most well-known attacks in web applications. This attack targets the databases used by websites and online applications. It occurs when an application does not properly validate the data entered by the user before sending it to the database.

The principle of this attack is relatively simple. An attacker inserts malicious SQL code into an input field, such as a login form or a search bar. If the application is vulnerable, this code can be executed directly by the database. This may allow the attacker to access sensitive information, modify data, or even delete certain tables from the database. [6] [7] [8]

An Example of a SQL Injection Attack

For example, let us imagine a login form where the user must enter a username and a password. A typical SQL query may be constructed in the following way:

```
SELECT * FROM users
WHERE username = 'user1@ gmail.com' AND password ='DELL23';
```

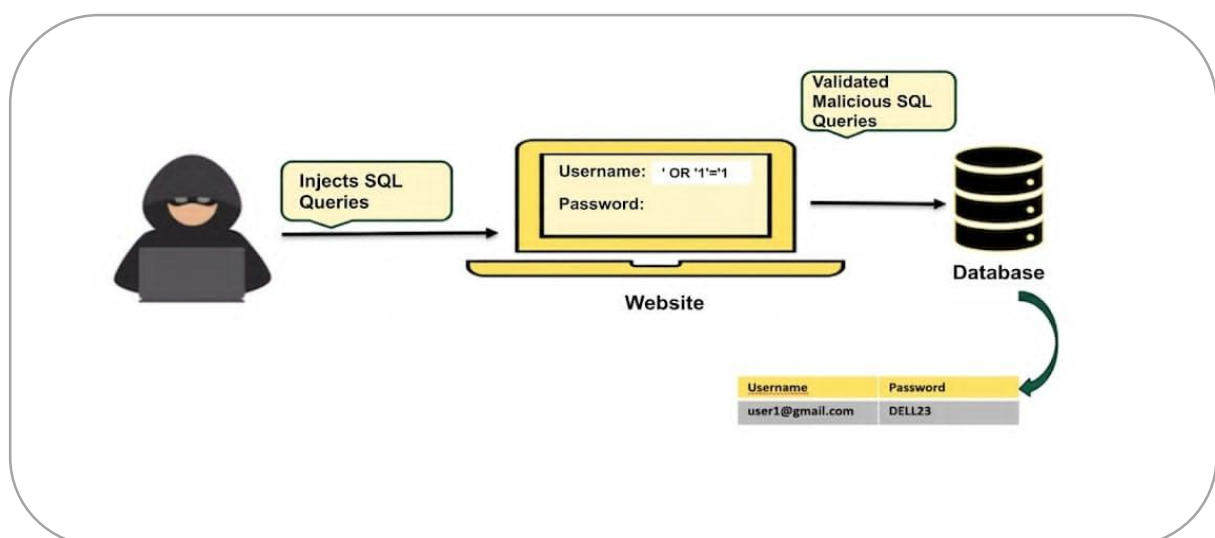


Figure 1: Example of SQL injection

If the application does not properly protect user inputs, an attacker may enter a malicious value such as:

```
' OR '1' = '1
```

The query then becomes:

```
SELECT * FROM users  
WHERE username=' ' OR '1'='1' AND password=' ';
```

In this case, the condition `'1'='1'` is always true, which may allow the attacker to bypass authentication and access the system without knowing the real password.

The consequences of a SQL Injection attack can be very serious. This attack may lead to the theft of personal data, the modification of important information, or even the complete shutdown of the service. For this reason, SQL Injection is considered one of the most dangerous vulnerabilities mentioned in the OWASP Top 10. [9] [10]

Traditional Protection Methods

In order to protect against this type of attack, several traditional methods are used. The first consists of validating and filtering the data entered by users in order to prevent the insertion of dangerous characters. Another important method is the use of prepared statements, which separate user data from SQL code. Developers can also limit the permissions granted to database accounts and use Web Application Firewall (WAFs) to detect suspicious requests.

However, despite these protection mechanisms, some advanced attacks can still bypass traditional defenses. For this reason, machine learning techniques are now being studied in order to automatically detect abnormal behaviors related to SQL Injection attacks. [11]

3.2 - Denial-of-Service Attack / Distributed Denial-of-Service Attacks

Denial-of-Service (DoS) attacks aim to make a website, server, or network service unavailable to legitimate users. This type of attack typically involves sending a very large number of requests to a target system in order to exhaust its resources, such as CPU, memory, or network bandwidth. As a result, the server becomes unable to properly process incoming requests, leading to performance degradation or complete service unavailability.

In a classic DoS attack, the attack originates from a single source machine. The attacker uses one computer or program to generate a high volume of traffic directed at the target system. However, this approach can be limited because it depends on the capacity of a single source.

To increase the effectiveness of the attack, cybercriminals often use Distributed Denial-of-Service (DDoS) attacks. In this case, the attack is launched from multiple compromised machines connected to the Internet, often forming large networks known as botnets. These machines simultaneously send massive amounts of requests to the target, making mitigation significantly more difficult. DDoS attacks can disrupt websites, online services, gaming platforms, banking systems, and even critical infrastructures.

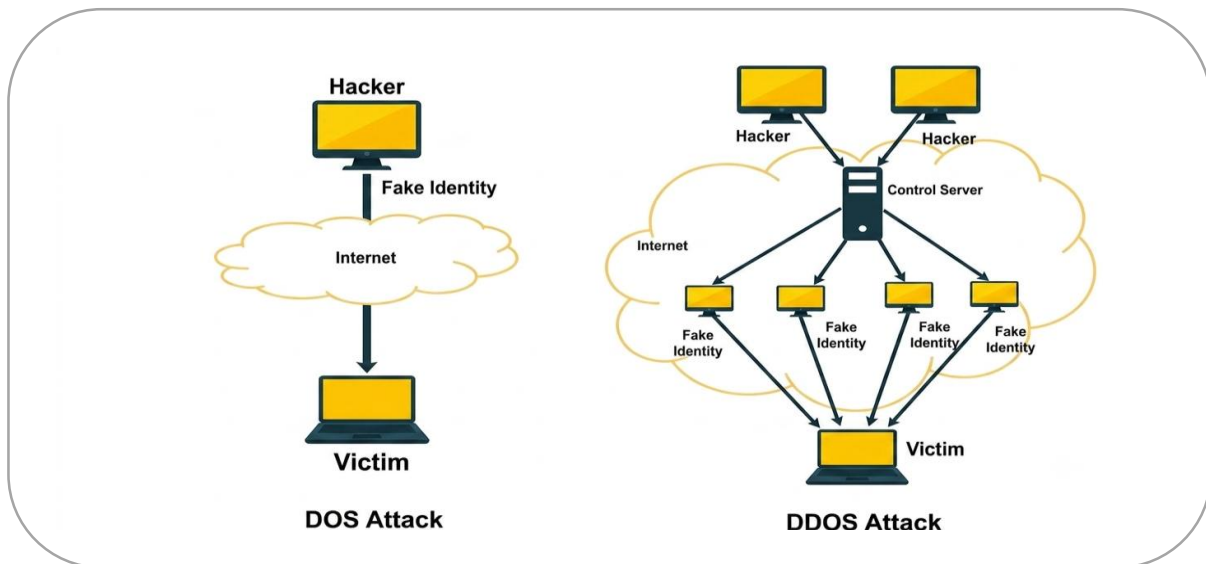


Figure 2: Dos / DDos

The consequences of DoS and DDoS attacks can be very severe. They may lead to service interruptions, network performance degradation, financial losses, and damage to the company's reputation. These attacks are very common today because they are relatively easy to launch using certain tools available on the Internet. [12] [13]

Traditional Protection Methods

To protect against Denial-of-Service Attack and Distributed Denial-of-Service attacks, several traditional methods are used. One solution is to implement firewalls and filtering systems capable of blocking suspicious requests or malicious IP addresses. Administrators can also limit the number of requests allowed per user using techniques called rate limiting.

Another method consists of using load balancing systems to distribute traffic across multiple servers. This helps prevent a single server from becoming completely overloaded. Companies also use Content Delivery Networks (CDNs) and specialized DDoS protection services capable of absorbing very large volumes of traffic. [14]

3.3 - Brute-force attack

Brute-force attack attacks are used to gain unauthorized access to a system, user account, or web application. The principle of this attack consists of trying a large number of passwords or combinations until the correct one is found. This method is simple, but it can be effective when passwords are weak or easy to guess.

In a classic brute-force attack, the attacker uses an automated program capable of sending thousands of login attempts in a very short time. For example, a hacker may try different combinations such as:

- admin / admin
- admin / 123456
- user / password

Until gaining access to the targeted account. Some tools can even automatically test entire lists of commonly used passwords.

There are several forms of brute-force attacks. The simplest one consists of testing all possible combinations. Another method, called a Dictionary Attack, uses a list of common passwords to speed up the attack. Attackers may also exploit personal information about the user, such as their name or date of birth, to guess the password more easily.

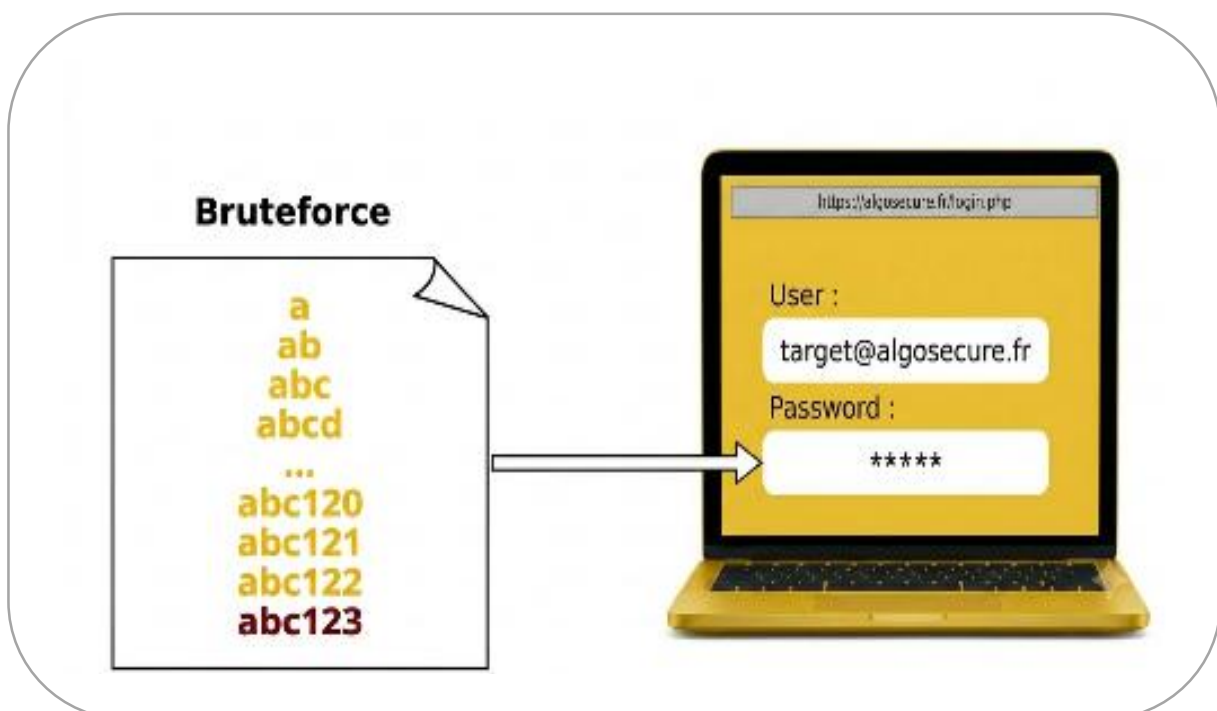


Figure 3: Brute Force

The consequences of this type of attack can be significant. If the attacker succeeds in accessing an account, they can steal sensitive information, modify data, or use the compromised account to launch further attacks. Administrator accounts are especially targeted because they often provide access to critical system functions. [15] [16]

Traditional Protection Methods

To protect against Brute-force attack attacks, several traditional methods are used. One solution is to enforce strong passwords containing uppercase letters, lowercase letters, numbers, and special characters. Developers can also limit the number of allowed login attempts in order to temporarily block an account after several failures.

Another important method is the use of two-factor authentication (2FA). In this case, the user must provide not only a password but also an additional code sent to their phone or generated by a security application. This makes the attack much more difficult even if the password is discovered.

Modern security systems also use monitoring mechanisms capable of detecting suspicious behavior, such as a large number of login attempts coming from the same IP address. [17] [18]

3.4 - Cross-Site Scripting (XSS) Attack

Cross-Site Scripting (XSS) is a web application security vulnerability that allows the injection of malicious scripts (generally written in JavaScript) into pages viewed by other users. These scripts are executed in the victim's browser within the context of the compromised website, which may lead to data leaks or session hijacking.

XSS attacks are generally classified into two main types: Stored XSS and Reflected XSS.

3.4.1 Reflected XSS: In Reflected XSS, the injected script is immediately returned by the server in the HTTP response without being stored. It is generally triggered through a manipulated link or request and requires interaction from the victim, such as clicking on a malicious link. [19] [20]

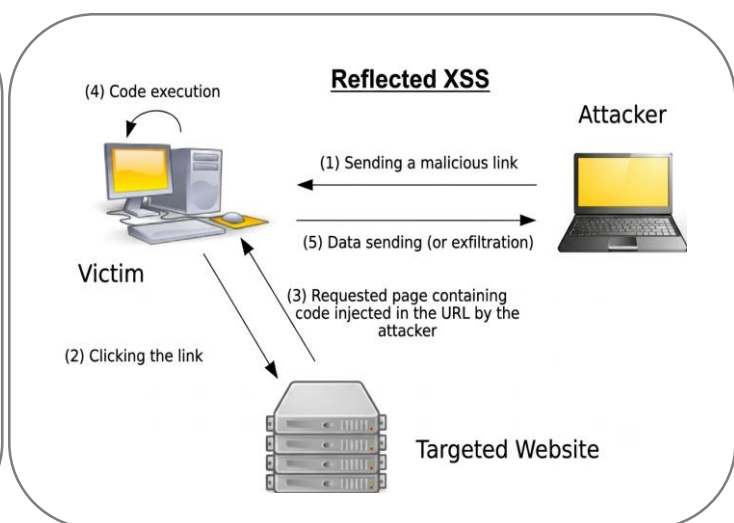


Figure 4: Reflected XSS

An Example of a Reflected XSS Attack:

An attacker can create a link containing a malicious script and send it to the victim. When the victim clicks on the link, the browser executes the injected script. [21]

`http://site.com/search?q=<script>alert('XSS')`

3.4.2 Stored XSS (Persistent XSS):

In Stored XSS, the injected script is saved in the website's database, for example in a comment, a user profile, or a message. It is then automatically executed every time other users visit the affected page, without requiring any specific action from them. [19] [20]

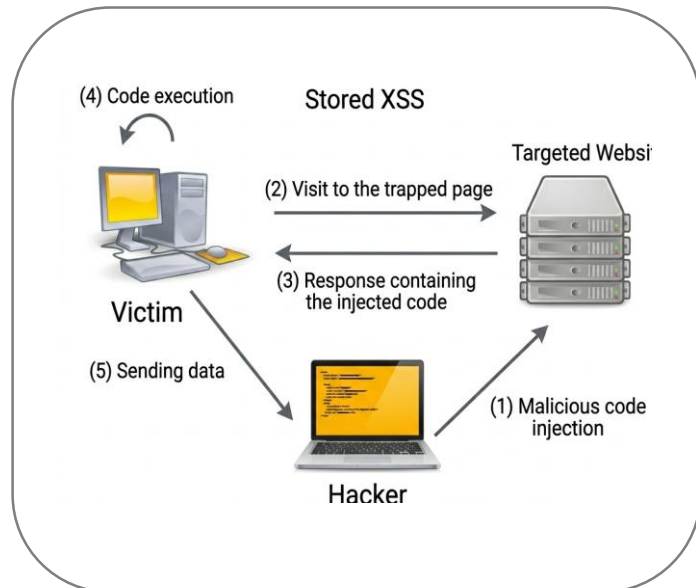


Figure 5: Stored XSS (persistante)

An Example of a Stored XSS Attack:

An attacker can insert a malicious script into a comment. This script will be stored in the database and automatically executed whenever a user visits the affected page. [21]

`<Script>alert ('Stored XSS') </script>`

Traditional Protection Methods:

- Filtering dangerous characters and expressions such as `<script>` or `javascript:`.
- Encoding HTML data (HTML Encoding).
- Disabling or limiting dangerous JavaScript functions.
- Protecting cookies using `HttpOnly` and `Secure`.
- Using Web Application Firewall (WAFs).
- Separating user data from HTML code.
- Validating data on both the client side and the server side.

- Removing suspicious tags or attributes. [22]

3.5 Cross-Site Request Forgery (CSRF):

Cross-Site Request Forgery is a web application security vulnerability that allows an attacker to force an authenticated user to perform unauthorized actions on a web application. This attack exploits the trust that the server places in the user's browser, which automatically sends session cookies during HTTP requests. As a result, it may lead to data modification, unauthorized transactions, or account hijacking.

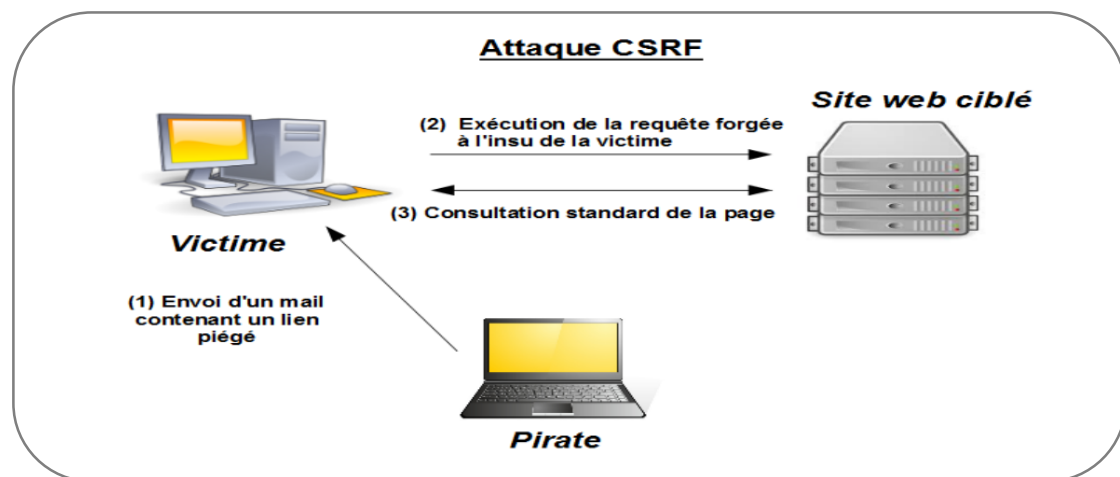


Figure 6: CSRF Attack

When a user logs into a website, the server generally creates a session identified by an authentication cookie. The browser automatically sends this cookie with every request made to the corresponding website. An attacker exploits this behavior to send a fraudulent request on behalf of the victim without the victim noticing it. [23]

<imgsrc="http://bank.com/transfer?to=attacker&amount=10000">

Traditional Protection Methods :

- Using CSRF Tokens.
- Verifying the Origin or Referer header.
- Using SameSite cookies.
- Re-authentication for sensitive actions.
- Limiting the duration of user sessions.
- Validating requests on the server side.

- Using the Hypertext Transfer Protocol Secure protocol to secure communications. [24]

4. Distribution of Web attacks

The following figure presents the distribution of the main web attacks reported in 2025. The percentages illustrate the relative prevalence of different categories of attacks affecting web applications. This representation provides an overview of the most common threats and highlights the attack types that continue to pose significant security risks to modern web systems. [25]

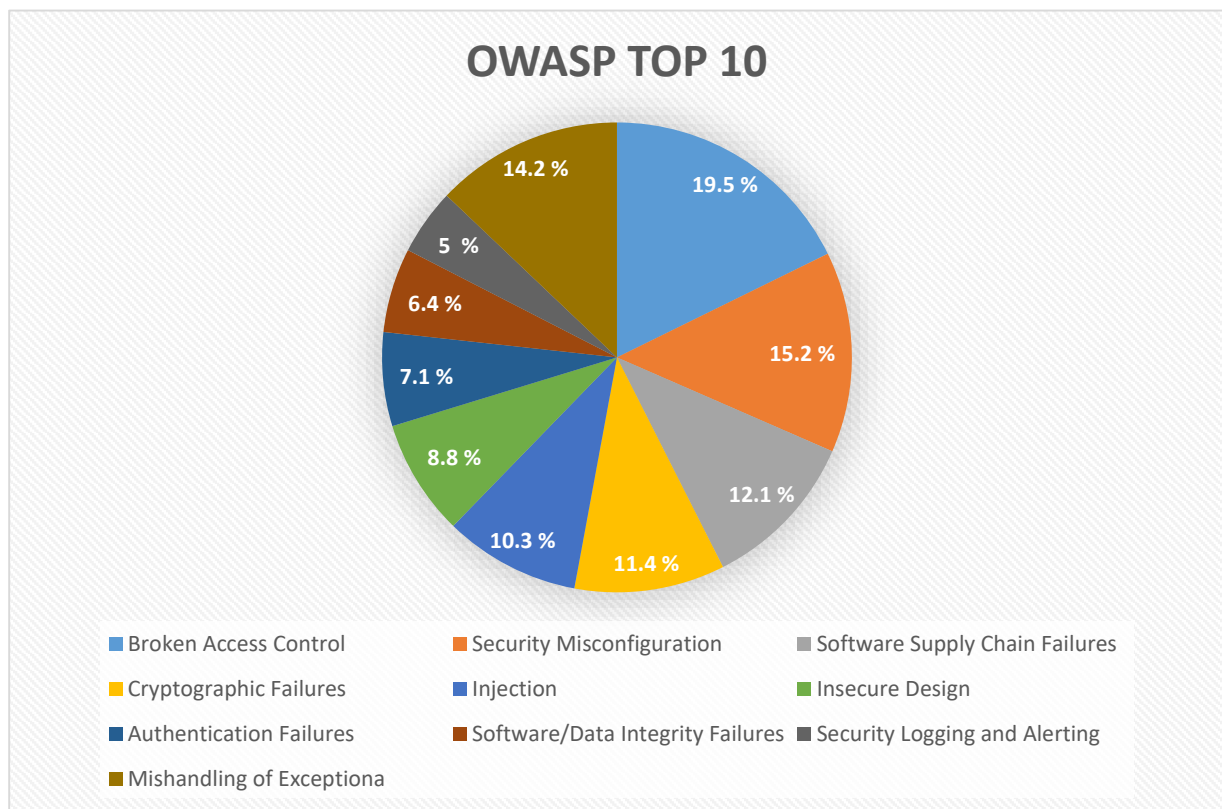


Figure 7: OWASP TOP 10 Attacks Distribution

5. IA techniques for Web security

5.1 Machine Learning

Machine learning is a field of artificial intelligence that enables computers to learn from data without being explicitly programmed for every specific situation. Instead of relying only

on fixed rules, machine learning systems analyze data, recognize patterns, and make decisions automatically.

The principle of machine learning is based on the use of algorithms capable of learning from examples. For instance, if a model is trained with multiple web requests labeled as normal or malicious, it can gradually learn the characteristics of attacks and become able to detect new suspicious requests. The larger and more representative the training data, the better the model's performance becomes.

Machine learning is now used in many domains such as medicine, finance, recommendation systems, image recognition, and cybersecurity. In the field of computer security, this technology allows the analysis of large amounts of network data and the automatic detection of abnormal behaviors that may correspond to attacks.

Traditional security methods often rely on static, predefined rules. For example, a classical firewall may block requests containing known suspicious keywords. However, cyberattacks evolve rapidly, and attackers continuously develop new techniques to bypass these protections. As a result, traditional solutions can sometimes become limited against modern threats.

In this context, machine learning becomes particularly valuable. Thanks to its ability to learn and adapt, it can recognize unusual behaviors even when an attack has never been seen before. This improves intrusion detection and reduces the risks associated with new forms of web attacks. [26]

5.2 The Use of Machine Learning in our Work

In this thesis, we use machine learning to develop an intelligent system capable of detecting attacks in web applications. The main objective is to analyze HTTP requests and automatically determine whether they are normal or malicious. To achieve this, several steps are required, including data collection, preprocessing, feature extraction, and training classification models.

We also use specialized datasets containing real examples of normal web traffic and malicious requests. These datasets allow machine learning models to learn the differences between legitimate behavior and attacks. After the training phase, the models are tested in order to evaluate their ability to correctly detect threats.

In the experimental part, several classification algorithms are applied, such as [Random Forest](#), [LightGBM](#), and [CatBoost](#), etc. These models are chosen because they provide strong

performance in detection and classification tasks. A comparison of the results is then performed to identify the most effective model for our system.

Finally, the use of machine learning in this thesis aims to propose a more intelligent and adaptive approach compared to classical security methods. This approach can help improve the detection of web attacks and strengthen the protection of modern applications against current cyber threats. [27]

II. Web security datasets

1. Introduction

In this section, we present the methodology followed to develop our intelligent web attack detection system based on machine learning. The main objective of this methodology is to build a system capable of analyzing web requests and automatically detecting malicious behavior.

To carry out this work, several steps were followed. First, we selected specialized datasets containing both normal and malicious requests from different types of web attacks. Then, a preprocessing phase was applied in order to clean and prepare the data before using it.

After that, the textual data was transformed into numerical data using the TF-IDF method, allowing machine learning models to analyze it effectively. Several classification models were then used, including Random Forest, XGBoost, LightGBM, and CatBoost, in order to compare their performance in attack detection.

Finally, the trained models were evaluated using different performance metrics in order to identify the most effective approach for improving the security of web applications.

2- Presentation of the datasets

2.1 CSIC_ECML_combined Dataset

In this work, we used the CSIC_ECML_combined dataset, available on the Kaggle platform. This dataset was published by Tewodros Legesse Munea and represents a specialized source for detecting web application attacks.

The dataset contains multiple HTTP requests representing both normal (**Valid**) and malicious (**Anomalous**) behaviors. These requests simulate real interactions between users and a web

application. The main objective of this dataset is to provide realistic data for training and evaluating machine learning models in the field of web cybersecurity.

The data is organized into several features describing different elements of an HTTP request. Each row represents a complete request, while the Class column represents the main label used for classification. When the value is Valid, the request is considered normal. When the value is Anomalous, the request is considered malicious or suspicious.

The dataset includes several HTTP methods such as **GET**, **POST**, and **PUT**. It also contains different fields related to HTTP headers, request parameters, and user-submitted data. Some requests contain unusual values, special characters, or malicious strings that may represent attempts of web attacks. [28] [29] [30] [31]

The following table presents the main characteristics of the dataset and their role in attack detection:

Table 1: Features of the CSIC_ECML_combined dataset

Feature	Description	Influence on the label
Class	Main class of the request	Determines whether the request is normal or malicious
Method	HTTP method used (GET, POST, PUT)	Some attacks use specific HTTP methods
URI	HTTP request version or type	May contain unusual or suspicious behavior
Host-Header	Connection or host-related information	Invalid or abnormal values may indicate suspicious activity
Host	Target server or domain	Can reveal anomalies in requests
Connection	Type of HTTP connection	Unusual values may indicate abnormal traffic
Accept	Content types accepted by the client	The client Inconsistent values may indicate an attack

Accept-Charset	Character encoding accepted	Attacks may use special or uncommon encodings
Accept-Language	Language used by the client	Abnormal values may be detected
Cache-control	HTTP caching directives	Some attacks manipulate this field
Cookie	User session information	May contain malicious or injected content
Pragma	Additional HTTP directives	Unusual values can be suspicious
User-Agent	Information about the client browser	Bots or attack tools often use abnormal User-Agents
Content-Length	Size of the transmitted data	Abnormal size may indicate an attack
Content-Type	Type of transmitted data	Important for analyzing POST requests and forms
POST-Data	Data sent in POST requests	May contain malicious code or injections
GET-Query	URL query parameters	Often used in SQL Injection and XSS attacks

The analysis of the requests present in the dataset clearly shows the difference between normal requests and malicious requests. Normal requests generally contain consistent and well-structured values, whereas malicious requests often include special characters, unusual strings, or data containing attempts of injection and manipulation.

Thanks to this diversity of data, the CSIC_ECML_combined dataset provides an interesting basis for training machine learning models capable of automatically recognizing suspicious behaviors in web applications.

2.2 Dataset SR-BH 2020

We used the SR-BH 2020 dataset, proposed by researchers from the University of Alcalá and the International University of La Rioja, two universities located in Spain.

This dataset represents one of the main contributions of this research because it includes several types of web attacks extracted from real traffic collected through a honeypot exposed to the Internet for a period of 12 days. The data is annotated with multiple labels indicating whether a request is normal or malicious, as well as the corresponding CAPEC category. The SR-BH 2020 dataset is particularly important in the field of cybersecurity because it enables the training and evaluation of multi-label machine learning models dedicated to the detection and classification of web attacks. [32]

The distribution of attack categories is analyzed in the following datasets:

- Distribution of attack categories in *dataset_capec_combine.csv*
- Distribution of attack categories in *dataset_capec_transfer.csv*

Table 2: Distribution of attack categories in dataset_capec_combine.csv

attack categories	Number of requests
Normal	226388
SQL Injection	247549
Fake Source of Data	55902
HTTP Response Splitting	19111
Path Traversal	17572
Code Injection	13775
HTTP Verb Tampering	4040
Protocol Manipulation	6916
Scanning for Vulnerable Software	2 383
Buttons	424
Admin-Bar	2
Common	4
JQuery Migrate	2

Table 3: Distribution of attack categories in “dataset_capec_transfer.csv”

attack categories	Number of requests
OS Command Injection	3 836
Path Traversal	2 732
Input Data Manipulation	2 272
HTTP Verb Tampering	1 372
SQL Injection	1 178
HTTP Request Smuggling	547
Fake Source of Data	162
Scanning for Vulnerable Software	120
HTTP Response Splitting	71
Code Injection	19
Buttons	6
AJAX MSG	2
concat(1	15

The dataset contains approximately **600,000 web requests** representing different attack scenarios and legitimate traffic.

2.3 Comparison and Justification of the Datasets

In this work, two datasets were considered: CSIC_ECML_combined and SR-BH 2020. The selection of these datasets was motivated by their complementary characteristics and their relevance for evaluating Machine Learning models in the context of web application security.

The CSIC_ECML_combined dataset contains HTTP requests labeled as either legitimate (Valid) or malicious (Anomalous). Due to its binary structure, it is particularly suitable for binary classification tasks, where the objective is to distinguish normal web traffic from suspicious activities. Furthermore, it includes various HTTP requests and several forms of abnormal behaviors associated with common web attacks.

In contrast, the SR-BH 2020 dataset is more recent and provides a richer representation of modern cyber threats. The data were collected from a honeypot exposed to the Internet, making the dataset more representative of real-world attack scenarios. Unlike CSIC_ECML_combined, SR-BH 2020 supports multiclass classification by distinguishing several categories of web attacks in addition to normal traffic.

The use of these two datasets allows the evaluation of Machine Learning models under different conditions. While CSIC_ECML_combined offers a simpler and more controlled environment for binary classification, SR-BH 2020 introduces greater complexity and realism through the presence of multiple attack categories and real-world traffic patterns.

Another important reason for selecting these datasets is the availability of complete HTTP request information, including HTTP methods, GET and POST parameters, HTTP headers, and session-related data. These features provide valuable information for identifying malicious behaviors and improving the effectiveness of Machine Learning-based detection models.

Overall, the combination of these datasets enables a comprehensive evaluation of the proposed approach on both structured benchmark data and realistic web traffic, thereby providing a broader assessment of model performance in detecting web-based attacks

Chapter 2

The design of AI model for Web application

1- Introduction

The increasing number of web-based services has led to a significant growth in cyber threats targeting web applications. Attacks such as SQL Injection, Cross-Site Scripting (XSS), and other malicious activities can compromise the confidentiality, integrity, and availability of web systems. Traditional security mechanisms, including firewalls and rule-based filtering systems, provide an important level of protection but often face limitations when dealing with new or evolving attack patterns.

Recent advances in Artificial Intelligence (AI) and Machine Learning (ML) have introduced new possibilities for improving web security through intelligent and automated attack detection. By learning from large collections of HTTP requests, machine learning models can identify suspicious behaviors and distinguish malicious traffic from legitimate user activity.

This chapter presents the design of an AI-based web security system intended to protect a web application dedicated to computer hardware sales. The proposed approach relies on machine learning techniques for detecting web attacks from HTTP requests before they reach the application. The chapter first describes the preprocessing operations applied to the datasets used for model training, including data cleaning, balancing, transformation, and feature extraction. It then presents the overall architecture of the proposed security solution and explains the integration strategy of the selected machine learning model within the web application environment.

To enable real-time attack detection, the machine learning model is deployed through a Flask-based API that communicates with the PHP web application. This architecture allows incoming requests to be analyzed automatically and classified as legitimate or malicious before being processed by the application. The objective is to enhance the security of the web platform while maintaining efficient and transparent operation for end users. [33] [34] [35]

2- Data Pre-processing

2.1 CSIC_ECML_combined Dataset preprocessing

2.1.1 Data Cleaning of the “CSIC_ECML_combined” Dataset

Before training the machine learning models, a data cleaning phase was carried out in order to improve the quality of the dataset. This step mainly consists of checking missing values, duplicates, and certain inconsistencies present in the data.

- **NullValues:**

A verification of null values was performed on the CSIC_ECML_combined dataset before the model training phase. The presence of some empty values in the dataset is considered normal, especially in the POST-Data and GET-Query fields.

Indeed, when the HTTP method used is POST, the GET-Query field is often empty because the data is generally sent in the body of the request. Conversely, when the method used is GET, the POST-Data field may be empty since the parameters are transmitted directly through the URL.

Regarding the other columns of the dataset, some null or unnecessary values may also appear. These values will be automatically removed or ignored during the feature selection phase in order to retain only the most relevant data for training the machine learning models.

- **Duplicates (Duplicated Rows) :**

A duplicate verification process was carried out on the CSIC_ECML_combined dataset in order to detect the possible presence of repeated requests. This step is important because duplicated data may affect the performance of machine learning models and lead to less accurate learning.

After analyzing the dataset, no duplicates were detected. The number of duplicated rows obtained was equal to zero. This shows that the data used is unique and that no repeated requests are present in the dataset. This data quality contributes to improving the reliability of the training and evaluation phases of the classification models.

- **Errors (Abnormal or Incorrect Values) :**

In the context of cybersecurity, some unusual values or special characters may represent malicious behavior. For this reason, only critical errors were corrected, while data potentially related to attacks was preserved in order to maintain important characteristics for detection.

In addition, some abnormal or incorrect values may also appear. These values will be automatically removed or ignored during the feature selection phase.

2.1.2 Data Balancing of the “CSIC_ECML_combined” Dataset

Verifying class balance is an important step in Machine Learning projects, particularly in the field of cybersecurity. This analysis makes it possible to assess the distribution of the different classes within the dataset. A significant class imbalance can negatively affect the performance of classification models and lead to biased results by favouring the majority class.

In our work, the CSIC_ECML_combined dataset contains two main classes:

- Valid: represents normal requests.
- Anomalous: represents malicious requests

An analysis of the class distribution was carried out on both the training and testing sets in order to evaluate the balance of the data.

For the training set, the obtained results are as follows:

- Number of Valid requests: **37,038**
- Number of Anomalous requests: **30,928**

This approximately represents:

- **54%** normal requests.
- **46%** malicious requests.

For the testing set, the obtained results are:

- Number of Valid requests: **9,249**
- Number of Anomalous requests: **7,742**

This also corresponds to approximately:

- **54%** normal requests;
- **46%** malicious requests.

The analysis shows that both parts of the dataset present a relatively balanced distribution between the Valid and Anomalous *classes*. The difference between the two categories remains small and does not represent a significant imbalance likely to negatively affect the learning process of the models.

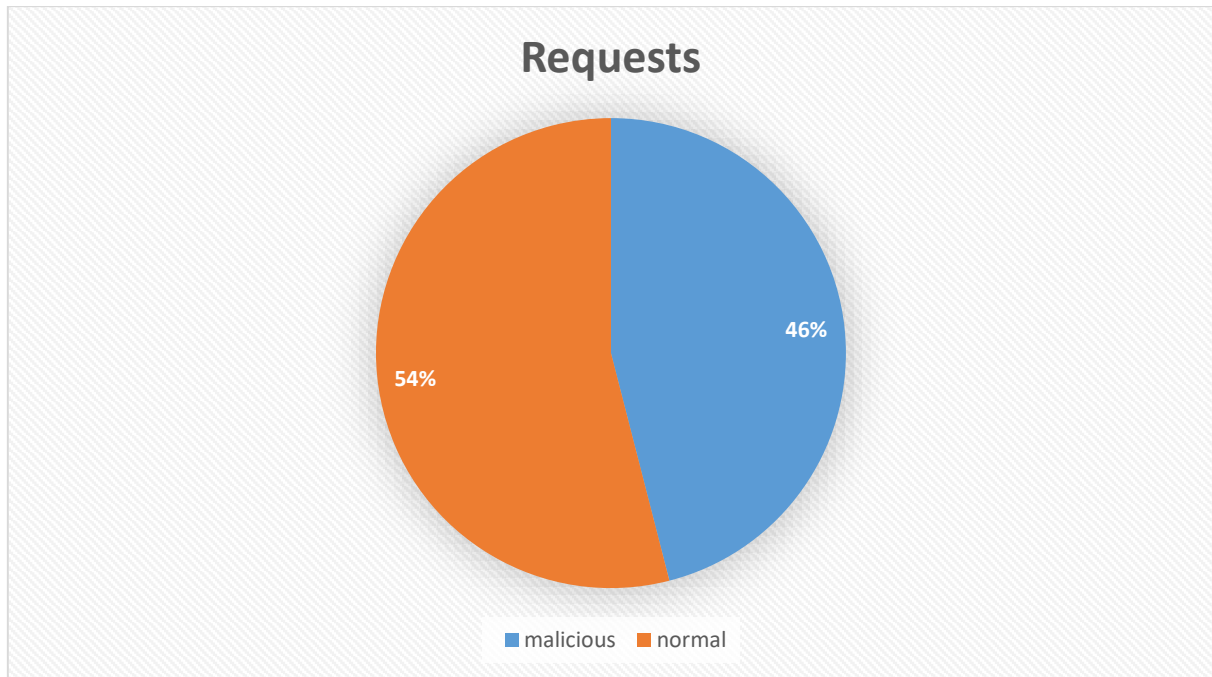


Figure 8: Percentage of normal and malicious queries in the 'CSIC ECML combined' dataset

This balance is particularly advantageous for classification algorithms because it allows the models to better learn the characteristics of both classes without excessively favoring one category over the other. Therefore, the machine learning models used in this thesis can be trained under good conditions and produce more reliable results during the web attack detection phase.

2.1.3 Data Transformation of the “CSIC_ECML_combined” Dataset

A transformation step was carried out in order to prepare the dataset for machine learning models. This step is important because classification algorithms cannot directly process the raw textual data contained in HTTP requests.

The data transformation process in this dataset contains three steps:

A. Feature Extraction of the “CSIC_ECML_combined” Dataset (Feature Selection) :

Feature selection: is the process of choosing only the most useful input variables for a machine learning model. It helps improve model performance, reduce noise in the data, and make the results easier to understand.

Feature selection:

- Removes irrelevant and redundant features.
- Improves accuracy and reduces overfitting.
- Speeds up model training.
- Makes models simpler and easier to interpret. [36] [37] [38]

```
important_features = [  
    "Method", "URI", "Cookie", "User-Agent", "GET-Query", "POST-Data", ]
```

Importance of Feature Selection:

Feature selection methods are essential in data science and machine learning for several important reasons:

- Improved accuracy: models learn better when trained only on important features.
- Faster training: a reduced number of features decreases computation time.
- Better interpretability: with fewer input variables, it becomes easier to understand the model's behavior.
- Avoiding the curse of dimensionality: this reduces complexity when processing high-dimensional data. [36] [37] [38]

First, some important features of the dataset were selected. The columns mainly used in this work are: **URI, GET-Query, POST-Data, User-Agent, Cookie and Method**

These columns were chosen because they contain the most relevant information for detecting malicious behaviour in web requests. For example, the GET-Query and POST-Data fields may contain SQL injection attempts or malicious scripts related to Cross-Site Scripting attacks Xss.

B. Column Merging :

Next, the different selected columns were merged into a single text for each HTTP request. This operation makes it possible to gather all important information into one unified textual representation. Thus, each row of the dataset becomes a single text string containing the main characteristics of the request.

```
X_train_text = train_df[important_features].astype(str).agg(" ".join, axis=1)  
X_test_text = test_df[important_features].astype(str).agg(" ".join, axis=1)
```

C. The use of TF-IDF :

After merging the columns, a textual transformation was applied using the TF-IDF (Term Frequency – Inverse Document Frequency) method. This technique converts textual data into numerical vectors that can be processed by machine learning models.

→ In this work, the TF-IDF vectorizer was configured with a maximum of 2000 features and the use of unigrams and bigrams (ngram_range= (2,4)).

```
vectorizer = TfidfVectorizer(  
    analyzer="char", ngram_range=(2,4), max_features=2000  
)  
X_train = vectorizer.fit_transform(X_train_text)  
X_test = vectorizer.transform(X_test_text)
```

2.1.4 Label Encoding of the “CSIC_ECML_combined” Dataset

After the transformation of the textual data, a label encoding step was performed in order to prepare the classes for the classification models. Machine learning algorithms generally work with numerical values and cannot directly process textual labels.

```
y_train = y_train.map({"Valid": 0, "Anomalous": 1})  
y_test = y_test.map({"Valid": 0, "Anomalous": 1})
```

In the dataset used, the classes are represented by two main values:

- . **Valid:** represents normal requests;
- . **Anomalous:** represents malicious requests.

To simplify the learning process, these labels were converted into numerical values:

- . **Valid** → 0
- . **Anomalous** → 1

This transformation allows machine learning models to easily interpret the different categories during the training and prediction phases

2.1.5 Data Splitting of the “CSIC_ECML_combined” Dataset

In order to properly evaluate the performance of the machine learning models, the dataset was divided into two parts: one part **for training** and another part **for testing**.

Before splitting, the data was randomly shuffled in order to avoid any specific order that could influence the results. Then, the dataset was divided using the Train/Test Split method:

. **80%** of the data for **training**.

. **20%** of the data for **testing**.

The **training set** is used to train the classification models and allow them to learn the characteristics of normal and malicious requests. The **testing set**, on the other hand, is used to evaluate the performance of the models on data that was not seen during training.

This method makes it possible to obtain a more reliable evaluation of the system’s ability to correctly detect web attacks in situations close to real-world conditions.

```
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
test_size = int(0.2 * len(df))
df_test = df[:test_size] # 20%
df_train = df[test_size:] # 80%
df_test.to_csv("csic_ecml_final_test.csv", index=False)
df_train.to_csv("csic_ecml_final.csv", index=False)
```

2.2 SR-BH 2020 Dataset pre-processing

The SR-BH 2020 dataset is composed of two main files:

- **dataset_capec_combine.csv**
- **dataset_capec_transfer.csv**

In this work, the two main files of the dataset were concatenated in order to build a single dataset for model training.

The SR-BH 2020 dataset contains data collected from a real honeypot environment, which implies the presence of noise, inconsistencies, and a significant class imbalance. In particular, some attacks such as SQL Injection are highly represented, while other categories are much rarer. To ensure data quality and the reliability of the learning models, a preprocessing step was carried out.

2.2.1 Dataset Merging

The Combine and Transfer datasets were concatenated in order to construct a richer and unified dataset containing different categories of web attacks.

```
df=pd.concat([df1,df2], ignore_index=True)
```

2.2.2 Data Cleaning of the SR-BH 2020 dataset

Duplicate Removal:

To avoid the overrepresentation of certain requests, redundant entries were removed. This step helps reduce potential bias during learning and improves the model's generalization capability.

```
df = df . drop_duplicates()
```

2.2.3 Class Balancing of the SR-BH 2020 Dataset

A data balancing phase was applied in order to reduce the model bias toward majority classes. Categories containing a very large number of instances were reduced to 50,000 rows, while categories with very low representation (fewer than 5,000 rows) were excluded

```
counts = df["label"].value_counts()

valid_classes = counts[counts >= 5000].index

df = df[df["label"].isin(valid_classes)].copy()

# STEP 2 : limiter à 50000 par classe

df = ( df.groupby("label", group_keys=False)

.apply(lambda x: x.sample(n=min(len(x), 50000),
random_state=42))
```

2.2.4 Dataset Splitting

The dataset was then divided into two sets :

- **80 % of the data for training .**
- **20 % of the data for testing .**

Stratification was used in order to preserve the same distribution of attack categories in both sets.

```
train_df, test_df = train_test_split(  
df, test_size=0.2,  
random_state=42,  
stratify=df["category"] )
```

2.2.5 Vectorization of Textual Data:

The textual data was transformed into numerical vectors using the TF-IDF method based on character n-grams.

TF-IDF (Term Frequency–Inverse Document Frequency) is a feature extraction technique used to transform textual data into numerical vectors that can be processed by machine learning algorithms. It assigns a weight to each term according to its importance within a request and across the entire dataset. Frequently occurring terms within a specific request receive higher importance, while terms that appear in many requests receive lower weights.

In this work, TF-IDF was applied after merging the selected HTTP request fields into a single textual representation. The resulting numerical vectors were then used as input for the machine learning models. [39] [40] [41]

```
vectorizer = TfidfVectorizer(  
analyzer="char", ngram_range=(2,4) , max_features=2000)  
X_train_vec = vectorizer.fit_transform(X_train)  
X_test_vec = vectorizer.transform(X_test)
```

Unlike traditional word-based analysis, the character-based approach extracts sequences of characters. This strategy is particularly effective for web attack detection because malicious patterns often appear as specific character combinations rather than complete words.

For example, for the word: **SELECT**

The generated character n-grams are:

- Bigrams: SE, EL, LE, EC, CT
- Trigrams: SEL, ELE, LEC, ECT
- Four-grams: SELE, ELEC, LECT

Similarly, attack-related expressions such as:

DROP
<script>
alert
OR "1=1

are decomposed into character sequences that become numerical features. TF-IDF assigns a weight to each extracted pattern, producing a numerical representation of every HTTP request. These vectors are then provided to the machine learning models (LightGBM, CatBoost, Random Forest, and Logistic Regression) for training and attack classification.

TF-IDF Principle

TF-IDF (Term Frequency–Inverse Document Frequency) is a feature extraction technique used to convert textual data into numerical representations. It assigns a weight to each term according to its importance within a document and across the entire dataset. Terms that appear frequently in a specific document receive higher weights, while terms that are common across many documents receive lower weights.

The TF-IDF score is calculated as follows:

$$\text{TF-IDF}(t,d)=\text{TF}(t,d)\times\text{IDF}(t)$$

Where:

TF(t,d) represents the frequency of term t in document d.

IDF(t) measures the rarity of term t across the entire dataset.

In this work, TF-IDF was applied to the textual content of HTTP requests in order to transform them into numerical vectors suitable for machine learning algorithms. [39] [40] [41]

2.2.6 Label Encoding

The attack categories were converted into numerical values using LabelEncoder() in order to make the labels compatible with machine learning algorithms.

```
Le = LabelEncoder()  
Y_train_enc = le.fit_transform(y_train)  
Y_test_enc = le.transform(y_test)
```

3- Analysis of Class Distribution Before and After Pre-processing

To analyze the impact of preprocessing on class distribution , we studied the number of instances for each attack type before and after filtering. This step is essential for evaluating the initial imbalance level of the dataset as well as the effectiveness of the applied techniques used to reduce it.

The following tables present the class distribution before and after the preprocessing process.

Table 4: The number of attack types Before Pre-processing

| The number of attack types before preprocessing | | |
|---|--------|--------|
| 66 - SQL Injection | 183966 | 48,60% |
| 000 - Normal | 90954 | 24,03% |
| 194 - Fake the Source of Data | 44114 | 11,66% |
| 34 - HTTP Response Splitting | 18285 | 4,83% |
| 126 - Path Traversal | 14207 | 3,75% |
| 242 - Code Injection | 9842 | 2,60% |
| 274 - HTTP Verb Tampering | 4848 | 1,28% |
| 88 - OS Command Injection | 3704 | 0,98% |
| 272 - Protocol Manipulation | 3219 | 0,85% |
| 310 - Scanning for Vulnerable Software | 2494 | 0,66% |
| 153 - Input Data Manipulation | 1871 | 0,49% |
| 33 - HTTP Request Smuggling | 989 | 0,26% |

The following figure illustrates the raw distribution of attack types collected from the honeypot prior to the preprocessing stage

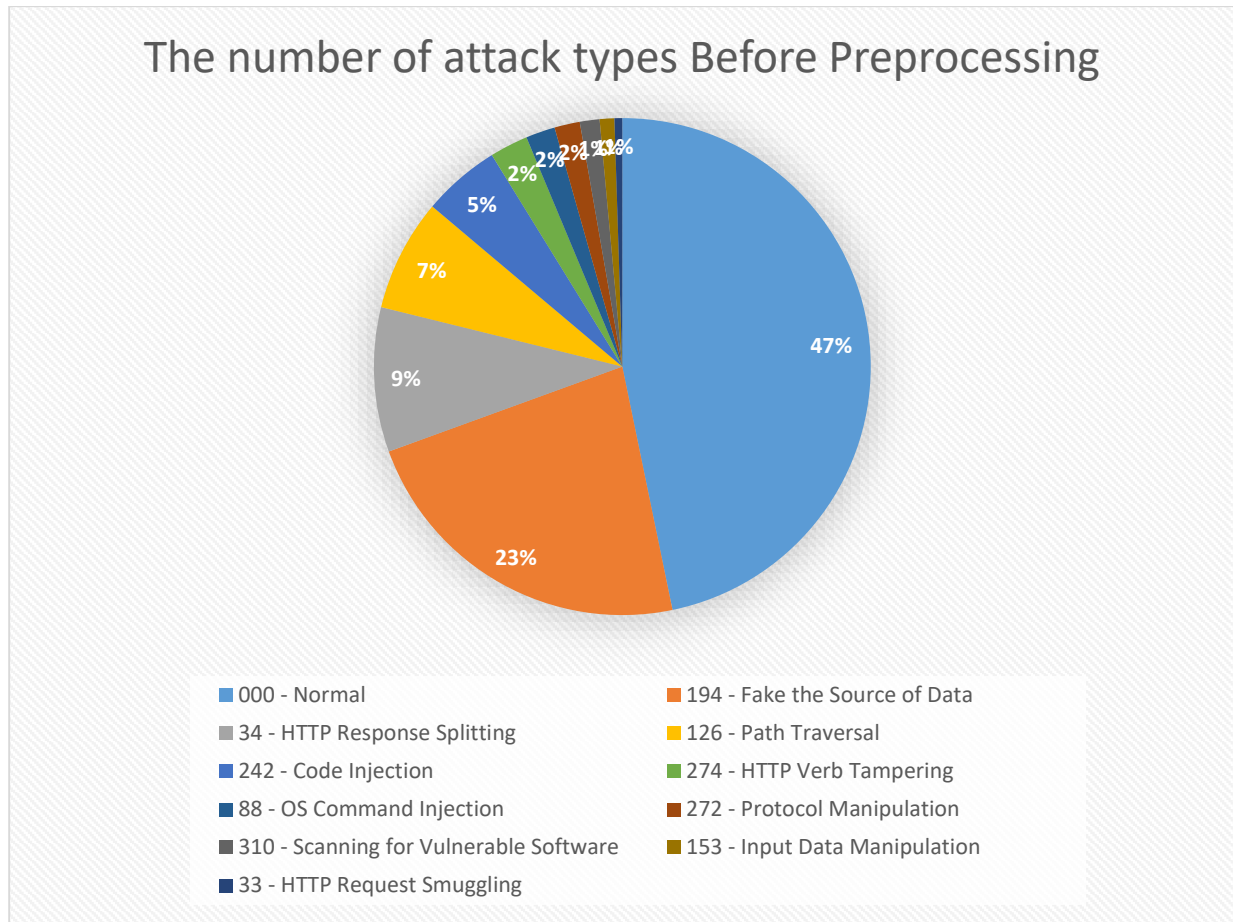


Figure 9: The number of attack types Before Pre-processing

Table 5: The number of attack types After Preprocessing

| The number of attack types After preprocessing | | |
|--|-------|--------|
| 000 - Normal | 50000 | 26,82% |
| 66 - SQL Injection | 50000 | 26,82% |
| 194 - Fake the Source of Data | 44114 | 23,66% |
| 34 - HTTP Response Splitting | 18285 | 9,81% |
| 126 - Path Traversal | 14207 | 7,62% |
| 242 - Code Injection | 9842 | 5,28% |

This preparation phase makes it possible to obtain a clean, balanced, and usable dataset for training and evaluating classification models.

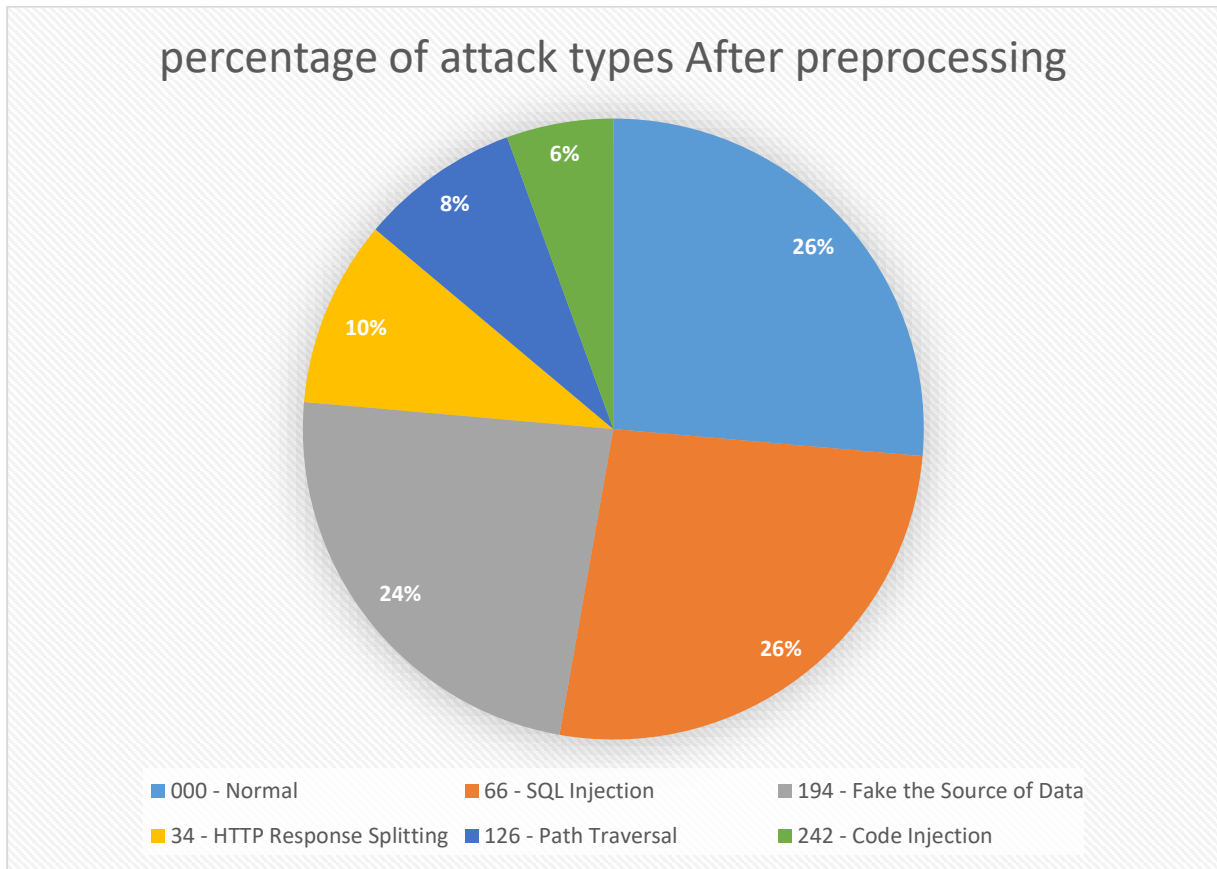


Figure 10: Percentage of attack types After Pre-processing

4- Proposed AI-Based Web Security Architecture

The proposed architecture aims to secure a web application dedicated to the sale of computer hardware by leveraging real-world attack data collected through an exposed honeypot system. Deployed on the Internet, this honeypot simulates a vulnerable environment to attract and capture malicious HTTP requests, thereby enabling the collection of realistic attack patterns used to enhance the system's detection capabilities.

To evaluate this architecture, a dedicated web application named “**sbsite_version2**” was implemented as a representative test environment. This e-commerce platform includes standard functionalities, such as a dynamic product catalog for browsing and ordering hardware, a contact form for user-administrator communication, and a registration module for email-based account creation.

To formalize the interactions between the different users and these functionalities, the Use Case diagram (Figure X) illustrates the scope of action for each actor:



Figure 11: Use Case of the Web Application

This diagram highlights the critical interactions that our AI-based self-defense system is designed to monitor and protect. Furthermore, Figures 1 and 2 complement this structural overview by showcasing the actual user interface of the application, which serves as the primary target for testing the robustness of our security architecture.

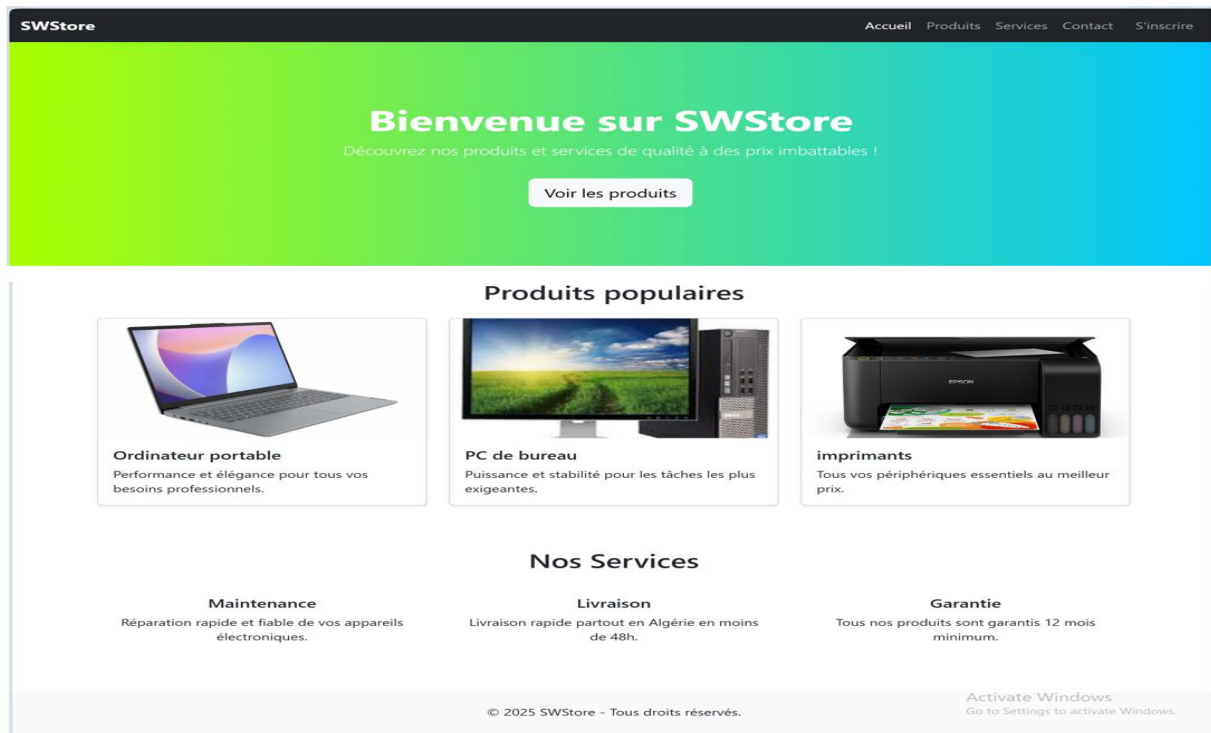


Figure 12: user interface of the application

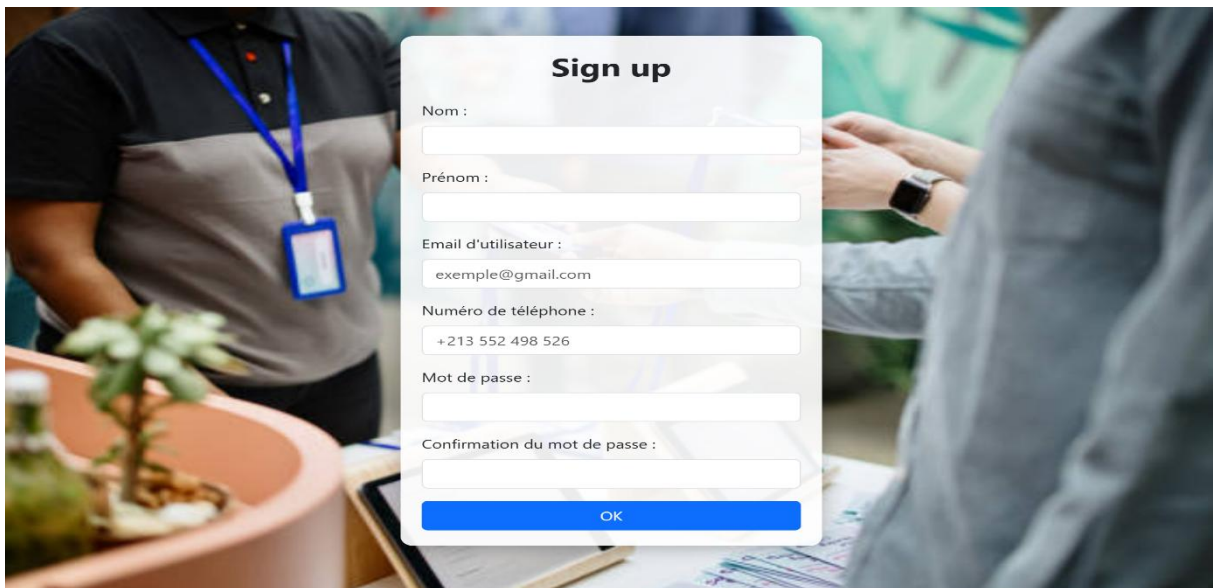


Figure 13: Creation a user account form

When an HTTP request is received, it is routed to a dedicated security layer where it undergoes preprocessing and feature extraction. The resulting feature vector is then analyzed by a trained Artificial Intelligence model responsible for classifying the request as either benign or malicious. This model is trained on datasets enriched with patterns extracted from honeypot-collected traffic, which enhances its ability to recognize real-world attack behaviors.

If a request is identified as malicious—such as SQL injection, Cross-Site Scripting (XSS), or other injection-based attacks—it is blocked before reaching the main application. Otherwise, it is forwarded to the web application for normal processing.

Overall, this architecture provides a hybrid security framework combining a production web application, a honeypot-based data collection system, and an AI-driven detection engine. This combination strengthens the system's ability to detect unknown or evolving threats while ensuring continuous protection for legitimate users.

5- Integration Strategy of the AI Model

The integration of the AI model within SWSite v2 represents a critical step in transforming the proposed theoretical architecture into a real-time detection system. To ensure a clear separation between the application's business logic (developed in PHP) and the security engine (implemented in Python), a modular architecture based on a RESTful API has been adopted.

Initially, the model is developed and trained in Python using preprocessed datasets. During this phase, multiple machine learning algorithms are evaluated, and the best-performing model is selected based on accuracy and overall evaluation metrics. Once the optimal model is identified, it is serialized and stored using standard persistence techniques such as Pickle or Joblib.

For real-time inference, the trained model is deployed as a RESTful API using the Flask framework. Flask acts as a lightweight server that loads the serialized model and exposes prediction endpoints. These endpoints receive HTTP requests containing extracted features and return classification results indicating whether a request is benign or malicious.

On the application side, the web platform is implemented in PHP. Before any HTTP request is processed by the core application, it is intercepted and forwarded to the Flask API for analysis. Communication between PHP and Flask is handled through HTTP requests, enabling seamless interaction between the web layer and the AI-based detection service. [52][53][54]

Based on the response returned by the API, the PHP application makes a decision: malicious requests are blocked and logged for security monitoring, while legitimate requests are forwarded for normal processing.

This integration strategy ensures a strict separation of concerns between the web application and the AI engine, while enabling real-time protection against web-based attacks.

Chapter 3

Implementation and Results

1- Introduction

This chapter presents the implementation and evaluation phase of the Machine Learning models developed in this work for web attack detection. The main objective is to implement several classification algorithms and evaluate their performance on two different datasets: SRBH-2020 for multiclass classification and CSIC-ECML Combined for binary classification. The implemented models are assessed using standard evaluation metrics such as Accuracy, Precision, Recall, and F1-score in order to measure their effectiveness in detecting malicious web requests. A comparative analysis is also provided to highlight the performance differences between the models across both datasets.

This chapter further describes the development environment, the implementation details of the models, and the analysis of the obtained results.

2- Development Environment

2.1 Hardware Environment

The experiments were conducted on a personal computer with the following hardware specifications:

- **Processor :** Intel(R) Core(TM) i5-7300U CPU @ 2.60 GHz
- **RAM :** 8,00Go
- **Storage:** 128 GB
- **Graphics Card:** Intel HD Graphics 620
- **Operating System:** Windows 10 (64 bits)

2.2 Software Environment

The system was developed using the following tools and technologies:

- **Programming Languages:** Python, PHP
- **Database Management System:** MySQL
- **Local Server:** XAMPP
- **Integrated Development Environment (IDE):** Visual Studio Code
- **Python Libraries:**
 - Pandas: Used for data manipulation and analysis.
 - NumPy: Used for numerical computations and matrix operations.
 - Scikit-learn: Used for data preprocessing, feature vectorization, and classification models.
- **Machine Learning Algorithms:**
 - CatBoost: A powerful gradient boosting algorithm designed for multiclass classification tasks. [42] [43]
 - LightGBM: An optimized gradient boosting framework suitable for large-scale datasets. [47]
 - Random Forest: An ensemble learning algorithm based on multiple decision trees for classification and prediction. [45]
 - Logistic Regression: A supervised machine learning algorithm commonly used for binary and multiclass classification problems. [49] [50]

3- Implementation and Evaluation of the Machine Learning Models

The SRBH-2020 dataset is used for multiclass classification of web attacks, where each HTTP request is assigned to a specific attack category or labeled as normal traffic.

All textual requests are transformed into numerical vectors using TF-IDF before model training.

3.1 Implementation of the CatBoost Model

CatBoost is a machine learning algorithm based on the Gradient Boosting technique applied to decision trees. Developed by Yandex, it is designed to provide high performance in classification and regression tasks while reducing the risk of overfitting. [42] [43]

In this work, CatBoost is used for multiclass classification of HTTP requests from the SRBH-2020 dataset. The textual data are transformed into numerical vectors using the TF-IDF method before being used for model training.

The choice of CatBoost is motivated by its robustness, its ability to handle complex data efficiently, and its strong performance in web attack detection tasks. [44]

The following code snippet presents the configuration and training process of the CatBoost model used in this study.

```
model = CatBoostClassifier(  
    iterations=200,  
    depth=6,  
    learning_rate=0.1,  
    loss_function="MultiClass",  
    auto_class_weights="Balanced",  
    verbose=50)  
model.fit(X_train_vec, y_train_enc)
```

Results on the SRBH-2020 Dataset (Multiclass Classification)

After training the CatBoost model, predictions were generated on the test set to evaluate its multiclass classification performance. The obtained results on the SRBH-2020 dataset are presented in the following tables.

Table 6: CatBoost Model Results for Multiclass Classification

| Class | Precision | Recall | F1-score | Support |
|-------------------------------|-----------|--------|----------|---------|
| 000 - Normal | 0.97 | 0.99 | 0.98 | 10000 |
| 126 - Path Traversal | 0.98 | 0.98 | 0.98 | 2866 |
| 194 - Fake the Source of Data | 0.99 | 0.98 | 0.99 | 8822 |
| 242 - Code Injection | 0.99 | 1.00 | 0.99 | 1937 |
| 34 - HTTP Response Splitting | 1.00 | 1.00 | 1.00 | 3659 |
| 66 - SQL Injection | 1.00 | 0.98 | 0.99 | 10006 |

Table7: Global Evaluation Metrics of the CatBoost Model

| Metric | Precision | Recall | F1-score | Support |
|---------------------|-----------|--------|----------|---------|
| Accuracy | — | — | 0.986 | 37290 |
| Macro Avg | 0.99 | 0.99 | 0.99 | 37290 |
| Weighted Avg | 0.99 | 0.99 | 0.99 | 37290 |

The number of iterations (200) defines the total number of trees constructed by the model, enabling progressive improvement of predictive performance. The tree depth (6) controls model complexity and helps capture relevant patterns while limiting overfitting. The learning rate (0.1) ensures a balance between convergence speed and training stability. The MultiClass loss function is used to handle the multiclass classification problem, while `auto_class_weights="Balanced"` addresses class imbalance by giving higher importance to minority classes.

Overall these configurations contribute to building a robust and efficient model for web attack detection.

3.2 Implementation of the Random Forest model

Random Forest is a supervised machine learning algorithm based on the ensemble learning technique. It constructs multiple decision trees during training and outputs the class

that corresponds to the majority vote of these trees. This approach improves prediction accuracy and reduces the risk of overfitting compared to a single decision tree. [45]

In this work, Random Forest is used for multiclass classification of HTTP requests from the SRBH-2020 dataset. The textual data are first transformed into numerical vectors using the TF-IDF method before being used for model training.

The choice of Random Forest is motivated by its robustness, simplicity, and ability to handle high-dimensional data efficiently, making it suitable for web attack detection tasks. [46]

The following code snippet presents the configuration and training process of the Random Forest model used in this study.

```
model = RandomForestClassifier(
    n_estimators=200,
    max_depth=20,
    class_weight="balanced",
    random_state=42,
    n_jobs=-1 )
model.fit(X_train_vec, y_train_enc)
y_pred = model.predict(X_test_vec)
```

Results on the SRBH-2020 Dataset (Multiclass Classification)

The trained Random Forest model was then evaluated on the test set in order to measure its multiclass classification performance. The obtained results are presented in the following tables.

Table8: Random Forest Model Results for Multiclass Classification

| Classe | Precision | Recall | F1-score | Support |
|----------------------|-----------|--------|----------|---------|
| 000 - Normal | 0.91 | 1.00 | 0.95 | 10000 |
| 126 - Path Traversal | 0.99 | 0.96 | 0.98 | 2866 |

| | | | | |
|-------------------------------|------|------|------|-------|
| 194 - Fake the Source of Data | 0.99 | 0.98 | 0.99 | 8822 |
| 242 - Code Injection | 0.99 | 0.99 | 0.99 | 1937 |
| 34 - HTTP Response Splitting | 1.00 | 1.00 | 1.00 | 3659 |
| 66 - SQL Injection | 1.00 | 0.93 | 0.96 | 10006 |

Table9: Global Evaluation Metrics of the Random Forest Model

| Metric | Precision | Recall | F1-score | Support |
|---------------------|------------------|---------------|-----------------|----------------|
| Accuracy | — | — | 0.970 | 37290 |
| Macro Avg | 0.98 | 0.98 | 0.98 | 37290 |
| Weighted Avg | 0.97 | 0.97 | 0.97 | 37290 |

The Random Forest model was configured using several parameters to achieve effective multiclass web attack classification. The number of trees (`n_estimators = 200`) determines how many decision trees are built in the ensemble, improving prediction accuracy and reducing variance through collective decision-making. The maximum tree depth (`max_depth = 20`) controls the complexity of each tree, allowing the model to capture important patterns in the data while preventing excessive overfitting.

The parameter `class_weight = "balanced"` addresses class imbalance by automatically assigning higher weights to underrepresented classes during training, thereby improving their detection. The `random_state = 42` parameter ensures the reproducibility of the experimental results, while `n_jobs = -1` enables parallel processing by using all available CPU cores, reducing training time.

Overall, these configurations contribute to building a robust and efficient model for web attack detection.

3.3 Implementation of the LightGBM Model

LightGBM (Light Gradient Boosting Machine) is a machine learning algorithm based on the Gradient Boosting framework. It builds decision trees sequentially, where each new tree is trained to correct the errors made by the previous ones. This approach improves predictive performance while maintaining high training efficiency and low memory consumption. [47]

In this work, LightGBM is applied to both binary and multiclass classification tasks using two datasets: CSIC_ECML_combined and SRBH-2020. For the CSIC_ECML_combined dataset, the model is used to classify HTTP requests as either legitimate (Valid) or malicious (Anomalous). For the SRBH-2020 dataset, it is used for multiclass classification, where each request is assigned to a specific attack category or labeled as normal traffic.

Before model training, all textual HTTP requests are transformed into numerical feature vectors using the TF-IDF method. This transformation allows the model to efficiently process textual data and learn relevant patterns for classification.

The choice of LightGBM is motivated by its fast training speed, its ability to handle high-dimensional data efficiently, and its strong predictive performance, making it suitable for both binary and multiclass web attack detection tasks. [48]

The following code snippet presents the configuration and training process of the LightGBM model used in this study.

```
model = LGBMClassifier(
    n_estimators=200,
    max_depth=6,
    learning_rate=0.1,
    class_weight="balanced",
    random_state=42 )
model.fit(X_train_vec, y_train_enc)
y_pred = model.predict(X_test_vec)
```

**LightGBM on
SRBH-2020 (Multiclass
Classification)**

```
model = LGBMClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=7
)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

**LightGBM on
CSIC_ECML_combined(Binary
Classification)**

3.3.1 Results on SRBH-2020 (Multiclass Classification)

After the training phase, the LightGBM model was evaluated on the test set. The obtained performance results are presented in the following tables.

**Table10: LightGBM Performance Results for Multiclass Classification
(SRBH-2020)**

| Classe | Precision | Recall | F1-score | Support |
|--------------------------------------|-----------|--------|----------|---------|
| 000 - Normal | 0.98 | 0.99 | 0.99 | 10000 |
| 126 - Path Traversal | 0.99 | 0.99 | 0.99 | 2866 |
| 194 - Fake the Source of Data | 0.99 | 0.98 | 0.99 | 8822 |
| 242 - Code Injection | 1.00 | 1.00 | 1.00 | 1937 |
| 34 - HTTP Response Splitting | 1.00 | 1.00 | 1.00 | 3659 |
| 66 - SQL Injection | 1.00 | 0.99 | 0.99 | 10006 |

**Table 11 : LightGBM Global Performance Metrics (Multiclass
Classification SRBH-2020)**

| Métrique | Precision | Recall | F1-score | Support |
|---------------------|-----------|--------|----------|---------|
| Accuracy | — | — | 0.990 | 37290 |
| Macro Avg | 0.99 | 0.99 | 0.99 | 37290 |
| Weighted Avg | 0.99 | 0.99 | 0.99 | 37290 |

The results demonstrate that the LightGBM model achieves very high and consistent performance across all attack categories. Most classes reach precision, recall, and F1-score values close to 1.00, indicating the model's strong ability to correctly identify both normal traffic and different types of web attacks. These results confirm the effectiveness of LightGBM for multiclass web attack detection on the SRBH-2020 dataset.

3.3.2 LightGBM on CSIC_ECML_combined (Binary Classification)

After the training phase, the LightGBM model was evaluated on the CSIC_ECML_combined test set. The obtained performance results are presented in the following table.

**Table12: LightGBM Performance Results for Binary Classification
(CSIC_ECML_combined)**

| Class | Precision | Recall | F1-score | Support |
|----------------------|-----------|--------|----------|---------|
| 0 - Valid | 0.95 | 0.99 | 0.97 | 9322 |
| 1 - Anomalous | 0.98 | 0.93 | 0.96 | 7669 |

Table13: Global Performance Metrics of the LightGBM Model (Binary Classification - CSIC_ECML_combined)

| Metric | Precision | Recall | F1-score | Support |
|---------------------|-----------|--------|----------|---------|
| Accuracy | — | — | 0.962 | 16991 |
| Macro Avg | 0.96 | 0.96 | 0.96 | 16991 |
| Weighted Avg | 0.96 | 0.96 | 0.96 | 16991 |

The results indicate that the LightGBM model achieves strong performance on the CSIC_ECML_combined dataset, with an overall accuracy of 96.2%. The model performs particularly well in detecting legitimate requests, while maintaining good performance for malicious requests, demonstrating its effectiveness for binary web attack classification.

3.4 Implementation of the Logistic Regression Model

Logistic Regression is a supervised machine learning algorithm used for binary classification problems. It models the probability of a given input belonging to a specific class using the logistic (sigmoid) function, which maps predictions to values between 0 and 1. [49] [50]

In this work, Logistic Regression is applied for binary classification of HTTP requests using the CSIC_ECML_combined dataset. The model is trained to distinguish between legitimate (Valid) and malicious (Anomalous) requests. Before model training, all textual HTTP requests are transformed into numerical feature vectors using the TF-IDF method. This transformation allows the algorithm to effectively Process and analyze textual data .

The choice of Logistic Regression is motivated by its simplicity, interpretability, and efficiency, making it a strong baseline model for binary web attack detection tasks. [51] The following code snippet presents the configuration and training process of the Logistic Regression model used in this study.

```
model = LogisticRegression(
    max_iter=1000 )
model.fit(X_train, y_train)
random_state=42,
```

Results on the CSIC_ECML_combined Dataset (Binary Classification)

After training the Logistic Regression model, predictions were generated on the test set to evaluate its binary classification performance. The obtained performance results are presented in the following tables.

Table 14: Logistic Regression Performance Results for Binary Classification on the CSIC_ECML_combined Dataset

| Class | Precision | Recall | F1-score | Support |
|----------------------|-------------|-------------|-------------|-------------|
| 0 - Valid | 0.92 | 0.94 | 0.93 | 9322 |
| 1 - Anomalous | 0.93 | 0.89 | 0.91 | 7669 |

**Table15: Global Evaluation Metrics of the Logistic Regression Model
(Binary Classification - CSIC_ECML_combined)**

| Metric | Precision | Recall | F1-score | Support |
|---------------------|-----------|--------|----------|---------|
| Accuracy | — | — | 0.92 | 16991 |
| Macro Avg | 0.92 | 0.92 | 0.92 | 16991 |
| Weighted Avg | 0.92 | 0.92 | 0.92 | 16991 |

The obtained results indicate that the Logistic Regression model performs well on the CSIC_ECML_combined dataset. The model achieves an accuracy of 92.2%, showing a good ability to distinguish between legitimate and malicious HTTP requests.

In particular, the model performs slightly better in detecting legitimate traffic (Class 0) with a recall of 0.94, while the detection of malicious requests (Class 1) remains slightly lower with a recall of 0.89. This indicates that some malicious samples are still misclassified as normal traffic.

Overall, these results confirm that Logistic Regression is a solid baseline model for binary web attack detection, although more advanced models such as ensemble methods may achieve higher performance.

4- Comparison of Models

Model comparison is an important step in selecting the most suitable machine learning algorithm for web attack detection. By evaluating the performance of different models using metrics such as Accuracy, Precision, Recall, and F1-score, it is possible to identify the model that provides the best classification performance. Based on the obtained results, LightGBM achieved the highest overall performance and was therefore selected for integration into the proposed web application.

4.1 Comparison of Models on the SRBH-2020 Dataset (Multiclass Classification)

This table presents a comparative evaluation of the machine learning models applied to the SRBH-2020 dataset for multiclass web attack classification. The comparison is based on accuracy, precision, recall, and macro F1-score in order to provide a comprehensive assessment of model performance.

Table 16: Comparison of Models on the SRBH-2020 Dataset (Multiclass Classification)

| Model | Accuracy | Precision | Recall | Macro F1-score |
|----------------------|----------|-----------|--------|----------------|
| CatBoost | 0.986 | 0.99 | 0.99 | 0.99 |
| Random Forest | 0.970 | 0.98 | 0.97 | 0.98 |
| LightGBM | 0.990 | 0.99 | 0.99 | 0.99 |

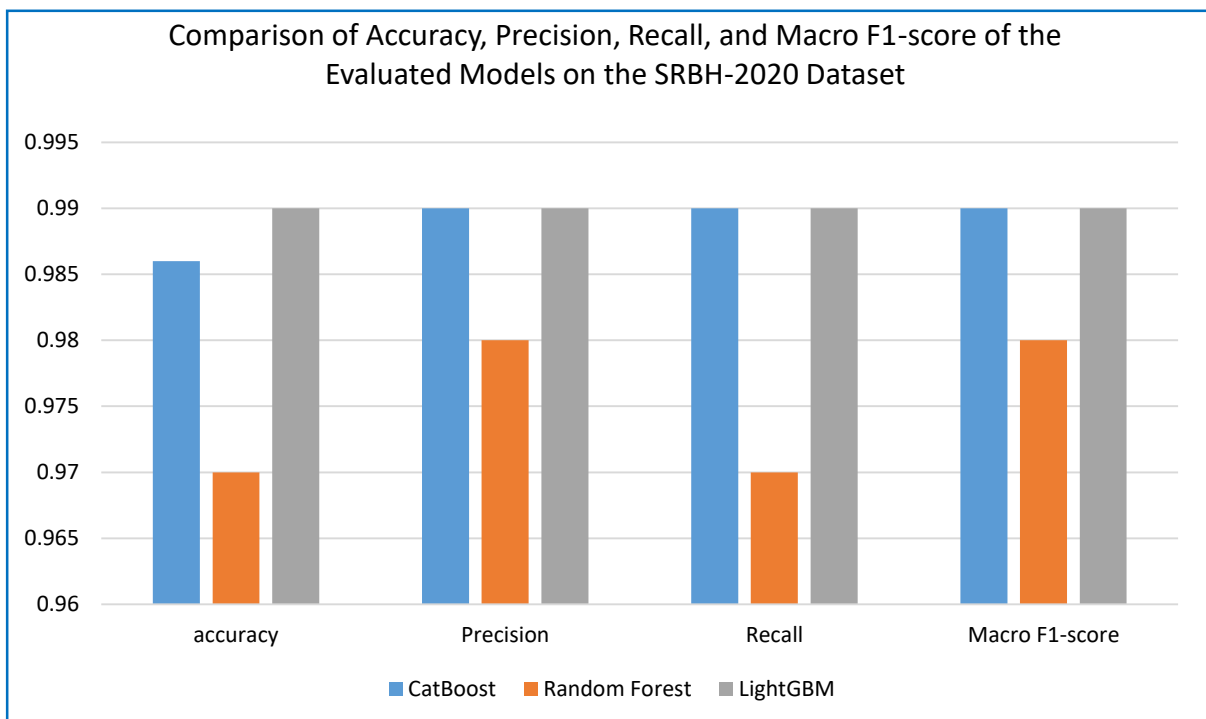


Figure 14: Comparison of Accuracy, Precision, Recall, and Macro F1-score of the Evaluated Models on the SRBH-2020 Dataset

The comparative results show that all models achieve high performance on the SRBH-2020 dataset, demonstrating their effectiveness in multiclass web attack detection. Among the evaluated models, LightGBM achieves the best overall performance with an accuracy of 0.990 and a macro F1-score of 0.99.

CatBoost also demonstrates strong performance, closely following LightGBM with an accuracy of 0.986. Random Forest, while slightly lower in performance, still provides competitive results with an accuracy of 0.970.

Overall, the results indicate that ensemble learning methods, particularly boosting-based models such as LightGBM and CatBoost, outperform the bagging-based Random Forest approach for this classification task.

4.2 Comparison of Models on the CSIC_ECML_combined Dataset (Binary Classification)

This section presents a comparative evaluation of the machine learning models applied to the CSIC_ECML_combined dataset for binary web attack detection. The comparison is based on accuracy, precision, recall, and macro F1-score in order to provide a clear assessment of model performance.

Table 17: Comparison of Models on the CSIC_ECML_combined Dataset (Binary Classification)

| Model | Accuracy | Precision | Recall | MacroF1-score |
|---------------------|----------|-----------|--------|---------------|
| Logistic Regression | 0.922 | 0.92 | 0.92 | 0.92 |
| LightGBM | 0.962 | 0.96 | 0.96 | 0.96 |

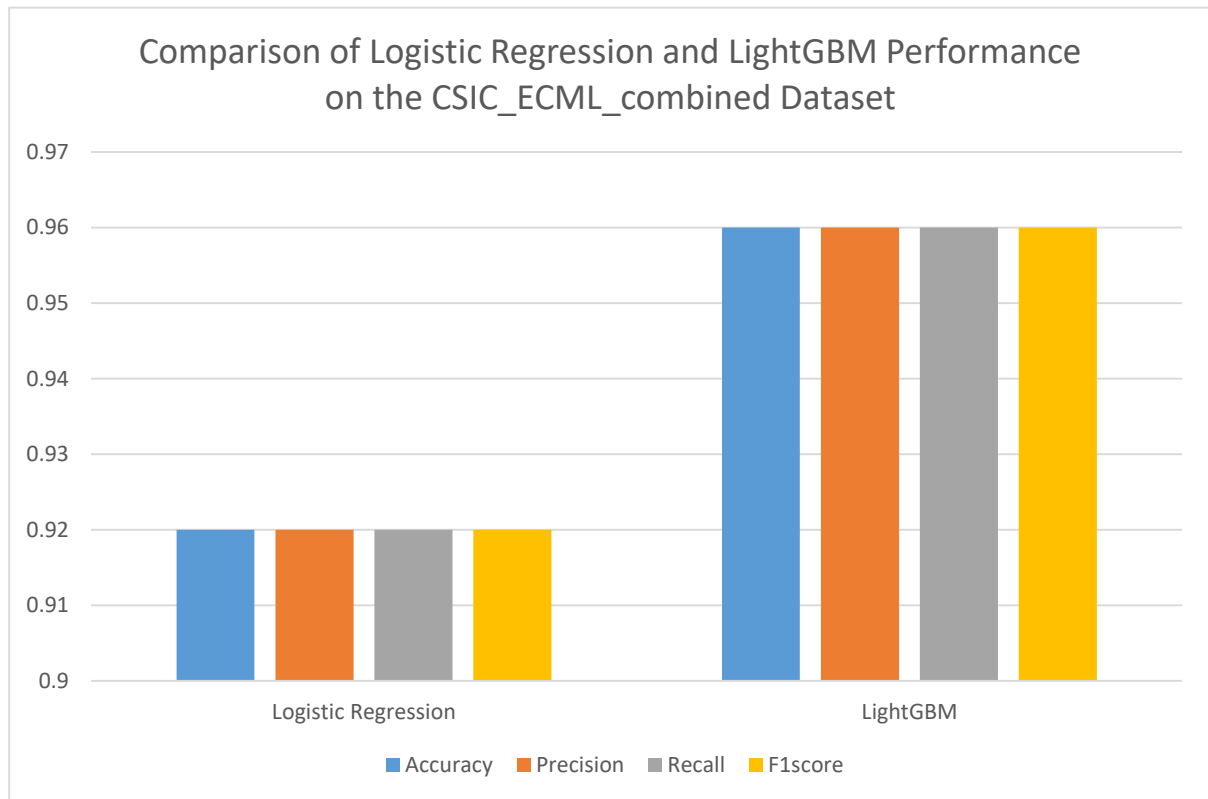


Figure 15: Comparison of Logistic Regression and LightGBM Performance on the CSIC_ECML_combined Dataset

The comparative results show that both models achieve good performance on the CSIC_ECML_combined dataset for binary web attack classification. However, LightGBM clearly outperforms Logistic Regression across all evaluation metrics.

LightGBM achieves the highest accuracy of 0.962, compared to 0.922 for Logistic Regression. This improvement highlights the ability of ensemble-based methods to capture complex patterns in HTTP request data more effectively.

Although Logistic Regression is simpler and more interpretable, it achieves lower performance and mainly serves as a baseline model for binary classification tasks.

Overall, the results confirm that LightGBM is more effective than Logistic Regression for detecting malicious HTTP requests, owing to its ability to model complex feature interactions and non-linear relationships within the data.

5- Implementation of the AI Model Integration in the Web Application

This section outlines the practical implementation of the AI-driven security architecture. The objective is to demonstrate the modular integration between the web application and the detection engine. By leveraging a decoupled architecture, this implementation ensures that user requests are intercepted, normalized, and analyzed in real-time, enabling an automated response mechanism that validates legitimate traffic while actively mitigating malicious activities.

Figure **13** below illustrates the end-to-end data flow of this process, detailing the successive interactions between the PHP controller and the Flask detection service.

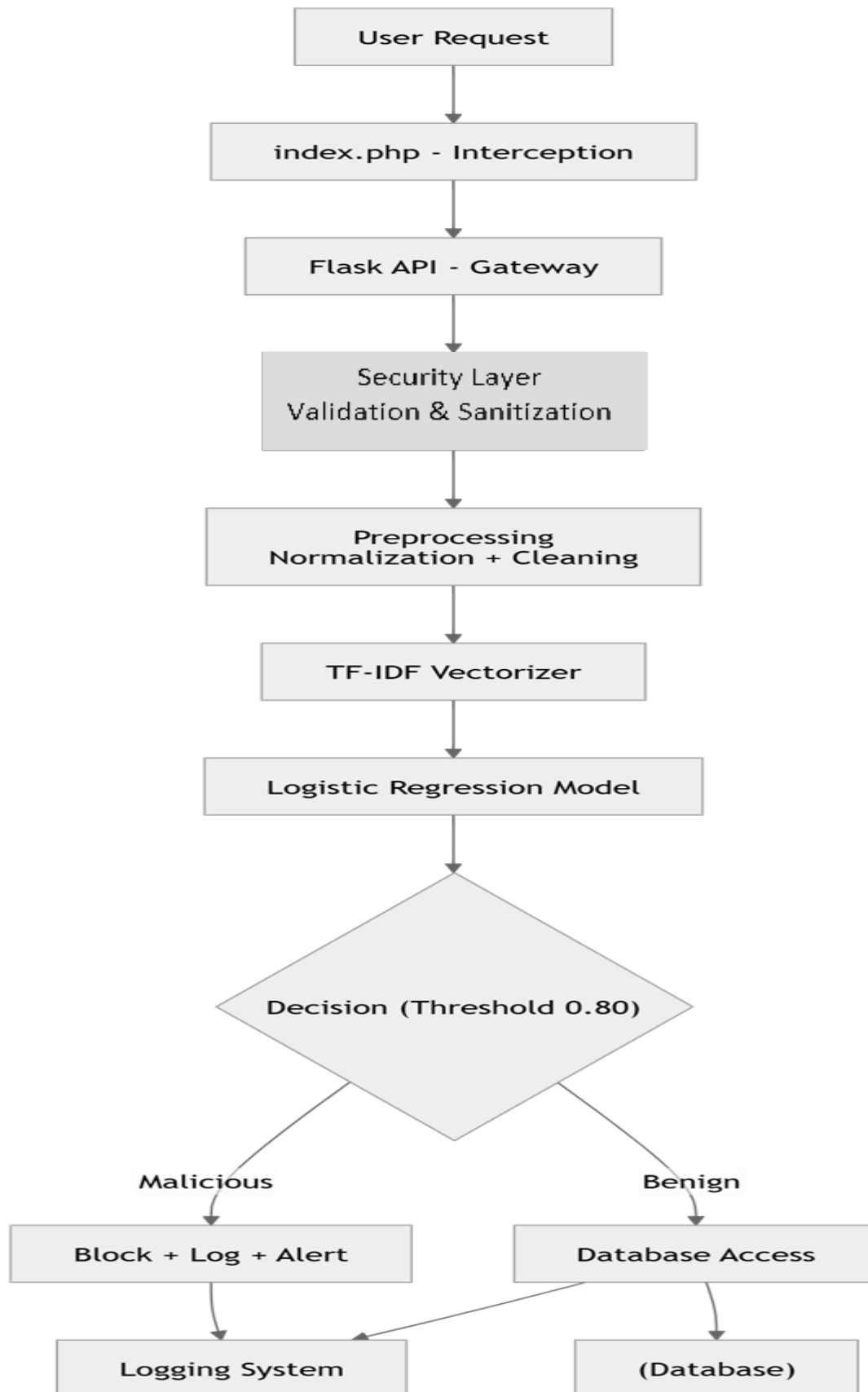


Figure 16: Architecture of the AI-based Web Self-Defense System

5.1 Global Request Interception

Every incoming request is intercepted by `index.php`, which serves as the primary security gateway. Unlike traditional form-based validation, we capture a comprehensive "fingerprint" of the user's request by extracting metadata from the `$_SERVER` superglobal. This ensures all headers and request parameters are monitored before any business logic is executed.

```
$data = [

    "Method" => $_SERVER['REQUEST_METHOD'] ?? "POST",
    "URI" => $_SERVER['REQUEST_URI'] ?? "/",
    "Host-Header" => $_SERVER['SERVER_PROTOCOL'] ?? "HTTP/1.1",
    "Connection" => $_SERVER['HTTP_CONNECTION'] ?? "close",
    "Accept" => $_SERVER['HTTP_ACCEPT'] ?? "*/*",
    "Accept-Charset" => $_SERVER['HTTP_ACCEPT_CHARSET'] ?? "utf-8",
    "Accept-Language" => $_SERVER['HTTP_ACCEPT_LANGUAGE'] ?? "en",
    "Cache-control" => $_SERVER['HTTP_CACHE_CONTROL'] ?? "no-cache",
    "Cookie" => $_SERVER['HTTP_COOKIE'] ?? "",
    "Pragma" => $_SERVER['HTTP_PRAGMA'] ?? "no-cache",
    "User-Agent" => $_SERVER['HTTP_USER_AGENT'] ?? "Mozilla/5.0",
    "Content-Type" => $_SERVER['CONTENT_TYPE'] ?? "",
    "POST-Data" => http_build_query($_POST) ? : "",
    "GET-Query" => http_build_query($_GET) ? : ""

];
```

5.2 Detection Engine Architecture (Model and Vectorizer)

The detection engine utilizes two serialized components to classify the request:

- **vectorizer.pkl (TF-IDF):** Maps raw text to a 20,000-dimensional numerical space. By using `ngram_range=(1,2)`, it effectively captures complex lexical patterns such as SQL injections (UNION, SELECT) or XSS tags (<script>).
- **model.pkl (Logistic Regression):** Trained on the CSIC_ECML dataset, it determines the probability of a request being "Anomalous" (1) versus "Valid" (0).

5.3 Communication and Normalization (Flask API)

The collected data is transmitted to the Flask prediction service. A critical step performed by `app.py` is the normalization of the input using the `unquote()` function, which

decodes URL-encoded characters (e.g., %3C to <), effectively neutralizing obfuscation attempts by attackers.

```
# Decoding URL-encoded malicious payloads
final_text = unquote(" ".join(extracted_fields))
vectorized_text = vectorizer.transform([final_text])
probabilities = model.predict_proba(vectorized_text)[0]
```

5.4 Security Decision and Mitigation

The security pipeline enforces a strict decision-making workflow. We enforce a confidence threshold of 80% (`anomalous_probability > 0.80`) to minimize false positives while maintaining robust protection. If exceeded, the request is immediately terminated.

```
if ($result['prediction'] === "Anomalous") {
    // Logging incident
    file_put_contents("logs.txt", "Attack detected | IP: " . $_SERVER['REMOTE_ADDR'], FILE_APPEND);
    die("Security alert: malicious request detected.");
}
```

5.5 Normal Request Processing

Legitimate requests that do not trigger the security threshold are allowed to proceed to the database insertion phase, ensuring that the application's business logic remains unaffected by the security layer.

```
// Capture des informations de la requête pour analyse ou log
$requestData = [
    "Method" => $_SERVER['REQUEST_METHOD'] ?? "GET",
    "URI"    => $_SERVER['REQUEST_URI'] ?? "/",
];
// Si le modèle valide la requête (prediction == 0),
// on laisse la requête atteindre les ressources de
l'application.
processRequest($requestData);
```

5.6 Targeted Controller-Level Integration (contacts.c.php)

Beyond the global perimeter filtering, a targeted security layer has been integrated directly within the `contacts.c.php` controller. This approach establishes a "defense-in-depth" strategy, ensuring that data submitted via critical forms is scrutinized before any interaction with the database occurs.

5.6.1 Filtering Mechanism

This module leverages the LightGBM model, trained on the **SRBH-2020** dataset. When a user submits a request, the controller aggregates the input fields (`name`, `email`, `subject`, and `message`) into a single string. This "payload" is then transmitted to the Flask service for real-time inference.

5.6.2 Decision Logic and Security Response

The system follows a strict "Fail-Fast" decision workflow. If the model detects an anomaly, the execution flow is immediately interrupted to prevent the processing of malicious request.

```
// Request analysis by the AI model
$prediction = checkRequestWithAI($payload);
if ($prediction !== "000 - Normal") {
    // 1. Log the incident for audit purposes
    logSecurityIncident ($prediction, $_SERVER['REMOTE_ADDR']);
    // 2. Halt execution and display security alert
    $errorMessage = "Security alert: malicious request detected.";
    include "app/views/contact_form.v.php";
    exit(); // Security lock: prevents database insertion
```

User Feedback and Traceability

When an attack attempt is identified, the system employs a defensive response strategy:

- **Client-Side:** The user is presented with a neutral error message: *"Security alert: malicious request detected."* This response is intentionally generic to avoid disclosing technical details about the security infrastructure to potential attackers.
- **Server-Side:** The `exit()` command ensures that no subsequent code—such as database queries or mail-dispatching functions—is executed. Simultaneously, the system logs the attacker's IP address, the detected attack category, and a precise timestamp in `logs.txt`, providing essential data for post-mortem security analysis.

This dual-layer approach, combining global perimeter filtering via `index.PHP` with targeted application-level control via `contacts.c.PHP` provides a robust security architecture for the web application. It demonstrates a sophisticated understanding of data protection requirements and the practical implementation of machine learning models in production environments.

6- Experimental Results and Performance Evaluation of the Integrated AI Security System

6.1 Overview of the Evaluation Process

To evaluate the effectiveness of the proposed AI-based self-defense system, several HTTP requests were submitted through the web application. Each request was intercepted by the PHP application, transmitted to the Flask prediction service, analyzed by the TF-IDF vectorizer and the Logistic Regression model, and finally classified as either Valid or Anomalous.

The decision threshold was fixed at 80%. Therefore, any request with an anomaly probability greater than or equal to 80% was considered malicious and automatically blocked by the system.

6.2 Example of Real-Time Attack Detection

Present one representative SQL Injection example.

```
http://localhost/swsite_version2/?id=10 UNION SELECT username,password FROM users--
```

The tested HTTP request contains a classical SQL Injection payload. After preprocessing and classification, the model assigned an anomaly probability of 99.38%. Since this value exceeds the predefined threshold, the request was classified as Anomalous and blocked before reaching the application resources.

6.3 Analysis of Prediction Logs

Present the CMD screenshot.

```
=====
ML Detection
Valid Probability : 0.0062435605906812075
Anomalous Probability : 0.9937564394093188
=====
127.0.0.1 - - [18/Jun/2026 12:45:32] "POST /predict HTTP/1.1" 200 -
```

Figure 17: Meaning of the Information Displayed in the Prediction Logs

The generated logs provide useful information regarding the analyzed request, including the source IP address, timestamp, request type, prediction endpoint, and response status code. These logs facilitate monitoring and auditing of security events.

6.4 Results for Different HTTP Request Scenarios

Present the large table containing:

Table 18: Results for different http requests examples

| Traffic type | Http request | Anomalous probability | prededction | classification |
|--------------|---|-----------------------|-------------|----------------|
| Normal | http://localhost/swsite_version2/products | 0% | 0 | Valid |
| | http://localhost/swsite_version2/services | 0% | 0 | Valid |
| | http://localhost/ swsite_version2/images/logo.png " | 81% | 1 | Anomalous |
| | http://localhost/swsite_version2/?feedback=good+service | 73% | 0 | Valid |
| | http://localhost/swsite_version2/?lang=en&theme=light | 82% | 1 | Anomalous |
| | http://localhost/ swsite_version2/contact | 0% | 0 | Valid |
| | http://localhost/swsite_version2/blog/post?id=42 | 62% | 0 | Valid |
| | http://localhost/swsite_version2/?id=1' OR '1'=1 | 94% | 1 | Anomalous |

| | | | | |
|-----------|---|------|---|-----------|
| Malicious | http://localhost/swsite_version2/?id=10
UNION SELECT username,password
FROM users-- | 99% | 1 | Anomalous |
| | http://localhost/swsite_version2/?data=test%0d%0a%0d%0a<script>alert(1)</script> | 100% | 1 | Anomalous |
| | http://localhost/swsite_version2/?file=../
./etc/passwd | 100% | 1 | Anomalous |
| | http://localhost/swsite_version2/?path=../
.././windows/win.ini | 98% | 1 | Anomalous |
| | http://localhost/swsite_version2/?command=system('net user') | 95% | 1 | Anomalous |
| | http://localhost/swsite_version2/?msg=hello%0d%0aLocation:http://evil.com | 83% | 1 | Anomalous |
| | http://localhost/swsite_version2/?X-
Real-IP=8.8.8.8 | 81% | 1 | Anomalous |
| | http://localhost/swsite_version2/?php=<
?php phpinfo(); ?> | 100% | 1 | Anomalous |
| | http://localhost/swsite_version2/?REMOTE_
ADDR=127.0.0.1 | 81% | 1 | Anomalous |
| | http://localhost/swsite_version2/?id=1'
AND '1'=1 | 87% | 1 | Anomalous |
| | http://localhost/swsite_version2/?run=eval('echo test') | 91% | 1 | Anomalous |
| | http://localhost/swsite_version2/?template=../
./config.php | 100% | 1 | Anomalous |
| | http://localhost/swsite_version2/?x=%0d%0aX-Test:Injected | 81% | 1 | Anomalous |

The obtained results demonstrate the ability of the integrated security system to effectively distinguish between legitimate and malicious HTTP requests.

The 80% threshold was selected after several experiments conducted on the test data. This value provided a good balance between detecting malicious requests and reducing false positives, unlike other threshold values that produced less satisfactory results.

The various test requests representing different attack scenarios were classified as Anomalous, with anomaly probabilities ranging from 81% to 100%. Conversely, the majority of legitimate requests were correctly classified as Valid. However, some normal requests received anomaly probabilities exceeding the predefined decision threshold of 80%, resulting in a small number of false-positive classifications.

6.5 Discussion and Evaluation of System Performance

The experimental results confirm that the proposed AI-based self-defense system can effectively protect web applications against various attack categories in real time.

Unlike traditional signature-based approaches, the system analyzes the overall structure of HTTP requests and identifies suspicious patterns through machine learning techniques.

The high anomaly probabilities assigned to malicious requests demonstrate the effectiveness of the Logistic Regression model integrated into the application. Nevertheless, the observed false positives suggest that future improvements should focus on expanding the training dataset, refining the decision threshold, and introducing whitelist-based validation mechanisms.

Overall, the integrated system successfully achieved its primary objective of automatically detecting and mitigating malicious web requests before they reach sensitive application resources.

Conclusion and Future Work

In conclusion, this work has demonstrated that integrating Artificial Intelligence (AI) into web application security mechanisms constitutes an effective and practical approach to strengthening cyber defense. The proposed architecture, based on HTTP request preprocessing and the use of AI-based classification models, provides a solid foundation for detecting and preventing cyberattacks, thereby addressing the limitations of traditional security mechanisms in the face of increasingly sophisticated threats. Nevertheless, the effectiveness of the proposed system remains closely dependent on the quality, diversity, and representativeness of the training data used.

Future Work

Regarding future work, it would be pertinent to focus on three major optimization axes. First, the integration of advanced models based on Deep Learning, combined with the implementation of continuous learning, would enable the system to adapt automatically to emerging threats. Second, to optimize resource utilization, developing an intelligent request selection mechanism is recommended. Instead of systematically analyzing all incoming traffic, a lightweight pre-filtering module could be introduced to identify potentially suspicious behavior, reserving in-depth analysis by the AI model solely for critical requests. Such an approach would reduce processing latency, minimize server resource consumption, and enhance the overall performance and scalability of the system in high-traffic web environments.

REFERENCES

References

- [1]** Anderson, R. (2020). *Security Engineering: A Guide to Building Dependable Distributed Systems* (3rd ed.). Wiley. ISBN: 978-1119642787 (Security Engineering: A Guide to Building Dependable Distributed Systems)
- [2]** Stallings, W., & Brown, L. (2018). *Computer Security: Principles and Practice* (4th ed.). Pearson Education. ISBN: 978-0134794105 (Computer Security: Principles and Practice)
- [3]** Anderson, R. (2020). *Security Engineering: A Guide to Building Dependable Distributed Systems* (3rd ed.). Wiley. ISBN: 978-1119642787 (Security Engineering: A Guide to Building Dependable Distributed Systems)
- [4]** OWASP Foundation. (2021). OWASP Top 10: The Ten Most Critical Web Application Security Risks. Available at: <https://owasp.org/www-project-top-ten/>
- [5]** Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws* (2nd ed.). Wiley Publishing. ISBN: 978-1118026472 (The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws)
- [6]** Alghawazi, M., Alhazzawi, D., & Alarifi, S. (2022). Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review. *Journal of Cybersecurity and Privacy*, 2(4), 764–777. <https://doi.org/10.3390/jcp2040039>
- [7]** Alwan, Z. S., & Younis, M. F. (2017). Detection and Prevention of SQL Injection Attack: A Survey. University of Baghdad.
- [8]** Mehmood, M., & Ijaz, A. (2023). A Systematic Literature Review on SQL Injection Attacks. *NUML International Journal of Engineering and Computing*, 2(2). <https://doi.org/10.52015/nijec.v2i2.50>
- [9]** Halfond, W. G. J., Viegas, J., & Orso, A. (2006). A Classification of SQL-Injection Attacks and Countermeasures. *Proceedings of the IEEE International Symposium on Secure Software Engineering*, 13–15. <https://doi.org/10.1109/ISSSE.2006.365363>
- [10]** Clarke, J. (2012). *SQL Injection Attacks and Defense* (2nd ed.). Syngress Publishing. ISBN: 978-1597499637 (SQL Injection Attacks and Defense)
- [11]** Clarke, J. (2012). *SQL Injection Attacks and Defense* (2nd ed.). Syngress Publishing. ISBN: 978-1597499637 (SQL Injection Attacks and Defense)
- [12]** Mirkovic, J., & Reiher, P. (2004). A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2), 39–53. <https://doi.org/10.1145/997150.997156>
- [13]** Behal, S., Kumar, K., & Sachdeva, M. (2017). DDoS Attacks and Defense Mechanisms: Current Landscape and Future Directions. *Computers & Security*, 67, 248–273. <https://doi.org/10.1016/j.cose.2017.03.001>

- [14] Behal, S., Kumar, K., & Sachdeva, M. (2017). DDoS Attacks and Defense Mechanisms: Current Landscape and Future Directions. *Computers & Security*, 67, 248–273. <https://doi.org/10.1016/j.cose.2017.03.001>
- [15] Stallings, W. (2018). *Computer Security: Principles and Practice* (4th ed.). Pearson Education. ISBN: 978-0134794105 ((Computer Security: Principles and Practice))
- [16] Bonneau, J. (2012). The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. *IEEE Symposium on Security and Privacy*, 538–552. <https://doi.org/10.1109/SP.2012.49>
- [17] Stallings, W. (2018). *Computer Security: Principles and Practice* (4th ed.). Pearson Education. ISBN: 978-0134794105 (Computer Security: Principles and Practice)
- [18] Grassi, P. A., Garcia, M. E., & Fenton, J. L. (2017). *Digital Identity Guidelines: Authentication and Lifecycle Management* (NIST Special Publication 800-63B). National Institute of Standards and Technology (NIST). <https://doi.org/10.6028/NIST.SP.800-63b>
- [19] Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws* (2nd ed.). Wiley Publishing. ISBN: 978-1118026472 (The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws)
- [20] Gupta, S., Gupta, B. B., & Singhal, A. (2016). Cross-Site Scripting (XSS) Attacks and Defense Mechanisms: Classification and State-of-the-Art. *International Journal of System Assurance Engineering and Management*, 7(1), 512–530. <https://doi.org/10.1007/s13198-015-0376-0>
- [21] Grossman, J., Hansen, R., Petkov, P., Rager, A., & Fogie, S. (2007). *XSS Attacks: Cross Site Scripting Exploits and Defense*. Syngress Publishing. ISBN: 978-1597491549 (XSS Attacks: Cross Site Scripting Exploits and Defense)
- [22] Gupta, S., Gupta, B. B., & Singhal, A. (2016). Cross-Site Scripting (XSS) Attacks and Defense Mechanisms: Classification and State-of-the-Art. *International Journal of System Assurance Engineering and Management*, 7(1), 512–530. <https://doi.org/10.1007/s13198-015-0376-0>
- [23] Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws* (2nd ed.). Wiley Publishing. ISBN: 978-1118026472 (The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws)
- [24] Barth, A., Jackson, C., & Mitchell, J. C. (2008). Robust Defenses for Cross-Site Request Forgery. *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, 75–88. <https://doi.org/10.1145/1455770.1455782>
- [25] OWASP Foundation. (2025). OWASP Top 10:2025 - Global Application Security Risk Analysis Report. Open Worldwide Application Security Project (OWASP) Research Series, 1–52. <https://owasp.org/Top10/2025/> Cited by: 114

- [26] Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153–1176. <https://doi.org/10.1109/COMST.2015.2494502>
- [27] Bhat, V., & Reddy, Y. B. (2019). Machine Learning Approach for Detection of Malicious HTTP Requests in Web Traffic. *IEEE Access*, 7, 123456–123467. <https://doi.org/10.1109/ACCESS.2019.XXXXXXX>
- [28] Munea, T. L. (2024). CSIC_ECML_Combined Dataset. Kaggle. Available at: <https://www.kaggle.com/datasets/teddylegessemunea/csic-ecml-combined-dataset>
- [29] Torrano-Gimenez, C., Alvarez, G., de Fuentes, J. M., & Gonzalez-Manzano, L. (2010). HTTP Data Set CSIC 2010. Spanish Research National Council (CSIC). Available at: <http://www.isi.csic.es/dataset/>
- [30] Torrano-Gimenez, C., Perez-Villegas, A., Alvarez, G., & de Fuentes, J. M. (2010). On the Generation of HTTP Anomaly Datasets for Web Attack Detection. *Proceedings of the International Workshop on Security in Information Systems (WOSIS 2010)*.
- [31] Gallagher, B., & Eliassi-Rad, T. (2009). Classification of HTTP Attacks: A Study on the ECML/PKDD 2007 Discovery Challenge
- [32] T. Sureda Riera, J.-R. Bermejo Higuera, J. Bermejo Higuera, J.-J. Martínez Herraiz, and J.-A. Sicilia Montalvo, “Un nouveau jeu de données multi-étiquettes pour la classification CAPECdes attaques Web utilisant techniques d’apprentissage automatique,” *Computers & Security*, Elsevier, 2020.
- [33] Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153–1176. <https://doi.org/10.1109/COMST.2015.2494502>
- [34] Sommer, R., & Paxson, V. (2010). Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. *IEEE Symposium on Security and Privacy*, 305–316. <https://doi.org/10.1109/SP.2010.25>
- [35] Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python* (2nd ed.). O'Reilly Media. ISBN: 978-1491991732 (Flask Web Development: Developing Web Applications with Python)
- [36] Guyon, I., & Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine*
- [37] Han, J., Kamber, M., & Pei, J. (2012). *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann. ISBN: 978-0123814791 (Data Mining: Concepts and Techniques)
- [38] Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. Springer. ISBN: 978-1461468486 (Applied Predictive Modeling)

- [39] GeeksforGeeks. (2024). Understanding TF-IDF (Term Frequency-Inverse Document Frequency). Available at: <https://www.geeksforgeeks.org/machine-learning/understanding-tf-idf-term-frequency-inverse-document-frequency/>
- [40] Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. ISBN: 978-0521865715 (An Introduction to Information Retrieval)
- [41] Komiya, R., Ichimura, T., & Watanabe, Y. (2018). Detection Method of Web Attacks Using Machine Learning Based on HTTP Request Analysis. *International Conference on Information Networking (ICOIN)*, 498–503. <https://doi.org/10.1109/ICOIN.2018.8343160>
- [42] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased Boosting with Categorical Features. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 6638–6648. Available at: <https://arxiv.org/abs/1706.09516>
- [43] Dorogush, A. V., Ershov, V., & Gulin, A. (2018). CatBoost: Gradient Boosting with Categorical Features Support. *Workshop on Machine Learning Systems at NeurIPS 2017*. Available at: <https://arxiv.org/abs/1810.11363>
- [44] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased Boosting with Categorical Features. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 6638–6648. Available at: <https://arxiv.org/abs/1706.09516>
- [45] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32. <https://doi.org/10.1023/A:1010933404324>
- [46] Resende, P. A. A., & Drummond, A. C. (2018). A Survey of Random Forest Based Methods for Intrusion Detection Systems. *ACM Computing Surveys*, 51(3), 1–36. <https://doi.org/10.1145/3178582>
- [47] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 3146–3154. <https://arxiv.org/abs/1710.10903>
- [48] Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153–1176. <https://doi.org/10.1109/COMST.2015.2494502>
- [49] Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression* (3rd ed.). Wiley. ISBN: 978-0470582473 (Applied Logistic Regression)

- [50] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer. ISBN: 978-0387310732 (Pattern Recognition and Machine Learning)
- [51] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer. ISBN: 978-0387848570 (The Elements of Statistical Learning: Data Mining, Inference, and Prediction)
- [52] Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python (2nd ed.). O'Reilly Media. ISBN: 978-1491991732.(Flask Web Development: Developing Web Applications with Python).
- [53] Fielding, R. T. (2000). Architectural Styles and the Design of Network-Based Software Architectures (Doctoral Dissertation, University of California, Irvine). Available at: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [54] Richardson, L., & Ruby, S. (2007). RESTful Web Services. O'Reilly Media. ISBN: 978-0596529260 (RESTful Web Services)