



**WANER SHAN**

BSc in Electrical and Computer Engineering

# **XSS ATTACK DETECTION BASED ON MACHINE LEARNING**

DISSERTATION FOR OBTAINING THE MASTER'S DEGREE  
IN ELECTRICAL AND COMPUTER ENGINEERING

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon  
September, 2022





DEPARTMENT OF ELECTRICAL  
AND COMPUTER ENGINEERING

---

# XSS ATTACK DETECTION BASED ON MACHINE LEARNING

DISSERTATION FOR OBTAINING THE MASTER'S DEGREE  
IN ELECTRICAL AND COMPUTER ENGINEERING

**WANER SHAN**

BSc in Electrical and Computer Engineering

**Adviser:** João Almeida das Rosas  
*Assistant Professor, NOVA University Lisbon*

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon  
September, 2022



## **XSS attack detection based on machine learning**

Copyright © Waner Shan, NOVA School of Science and Technology, NOVA University Lisbon.

The NOVA School of Science and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.



## ABSTRACT

As the popularity of web-based applications grows, so does the number of individuals who use them. The vulnerabilities of those programs, however, remain a concern. Cross-site scripting is a very prevalent assault that is simple to launch but difficult to defend against. That is why it is being studied.

The current study focuses on artificial systems, such as machine learning, which can function without human interaction. As technology advances, the need for maintenance is increasing. Those maintenance systems, on the other hand, are becoming more complex. This is why machine learning technologies are becoming increasingly important in our daily lives.

This study use supervised machine learning to protect against cross-site scripting, which allows the computer to find an algorithm that can identify vulnerabilities. A large collection of datasets serves as the foundation for this technique. The model will be equipped with functions extracted from datasets that will allow it to learn the model of such an attack by filtering it using common Javascript symbols or possible Document Object Model (DOM) syntax.

As long as the research continues, the best conjugate algorithms will be discovered that can successfully fight against cross-site scripting. It will do multiple comparisons between different classification methods on their own or in combination to determine which one performs the best.

**Keywords:** Cross-site scripting, supervised learning algorithms, classifiers, javascript, DOM, HTTP





## RESUMO

À medida que a popularidade dos aplicativos da internet cresce, aumenta também o número de indivíduos que os utilizam. No entanto, as vulnerabilidades desses programas continuam a ser uma preocupação para o uso da internet no dia-a-dia. O cross-site scripting é um ataque muito comum que é simples de lançar, mas difícil de-se defender. Por isso, é importante que este ataque possa ser estudado.

A tese atual concentra-se em sistemas baseados na utilização de inteligência artificial e Aprendizagem Automática (ML), que podem funcionar sem interação humana. À medida que a tecnologia avança, a necessidade de manutenção também vai aumentando. Por outro lado, estes sistemas vão tornando-se cada vez mais complexos. É, por isso, que as técnicas de machine learning torna-se cada vez mais importantes nas nossas vidas diárias.

Este trabalho baseia-se na utilização de Aprendizagem Automática para proteger contra o ataque cross-site scripting, o que permite ao computador encontrar um algoritmo que tem a possibilidade de identificar as vulnerabilidades. Uma grande coleção de conjuntos de dados serve como a base para a abordagem proposta. A máquina virá ser equipada com o processamento de linguagem natural, o que lhe permite a aprendizagem do padrão de tal ataque e filtrando-o com o uso da mesma linguagem, javascript, que é possível usar para controlar os objectos DOM (Document Object Model).

Enquanto a pesquisa continua, os melhores algoritmos conjugados serão descobertos para que possam prever com sucesso contra estes ataques. O estudo fará várias comparações entre diferentes métodos de classificação por si só ou em combinação para determinar o que tiver melhor desempenho.

**Palavras-chave:** Cross-site scripting, machine learning supervisionado, classificadores, javascript, DOM, HTTP



# CONTENTS

<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>ix</b>
<b>Glossary</b>	<b>xi</b>
<b>Acronyms</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Research Question . . . . .	3
1.3 Research Objective . . . . .	3
1.4 Reading Guide . . . . .	4
<b>2 Background and Related Studies</b>	<b>5</b>
2.1 Web Applications . . . . .	5
2.2 JavaScript . . . . .	6
2.2.1 JS Features . . . . .	8
2.2.2 JS Composition . . . . .	8
2.2.3 Add JS to webpage . . . . .	9
2.2.4 JS Security . . . . .	9
2.3 Cross-Site Scripting Attack . . . . .	10
2.3.1 Stored/Persistent XSS . . . . .	11
2.3.2 Reflected/Non-persistent XSS . . . . .	12
2.3.3 DOM-based XSS . . . . .	13
2.4 Machine Learning . . . . .	14
2.4.1 Machine Learning Algorithms . . . . .	15
2.4.2 Supervised Learning . . . . .	17
2.4.3 Evaluation . . . . .	18
2.5 Existing XSS Detection Mechanisms . . . . .	20

## CONTENTS

---

2.5.1	Standard Methods . . . . .	20
2.5.2	Static/Dynamic analysis . . . . .	21
2.5.3	Machine Learning Analysis . . . . .	22
2.6	Summary . . . . .	26
<b>3</b>	<b>Datasets</b>	<b>27</b>
3.1	Data Collect . . . . .	27
3.2	Features Extraction . . . . .	28
3.3	Summary . . . . .	36
<b>4</b>	<b>Classifiers</b>	<b>37</b>
4.1	Train Model . . . . .	37
4.1.1	Linear Regression . . . . .	37
4.1.2	Decision Tree . . . . .	40
4.1.3	Support Vector Machine . . . . .	42
4.1.4	K-Nearest Neighbors . . . . .	43
4.1.5	Random Forest . . . . .	45
4.1.6	Naive Bayes . . . . .	47
4.1.7	Execution Time . . . . .	48
4.2	Discussion . . . . .	49
<b>5</b>	<b>Ensemble algorithms</b>	<b>51</b>
5.1	Introduction to ensemble algorithms . . . . .	51
5.2	The framework . . . . .	52
5.3	Random Forest with Decision Tree . . . . .	55
5.4	K-Nearest Neighbors with Random Forest . . . . .	56
5.5	K-Nearest Neighbors with Decision Tree . . . . .	56
5.6	Execution Time . . . . .	57
5.7	Discussion . . . . .	58
<b>6</b>	<b>Real Time Application</b>	<b>60</b>
6.1	Introduction to Chrome Extension . . . . .	60
6.2	The Framework and Language . . . . .	62
6.3	Backend Services . . . . .	63
6.4	Frontend Application . . . . .	65
6.5	Discussion . . . . .	70
<b>7</b>	<b>Conclusion</b>	<b>71</b>
7.1	Research Summary . . . . .	71
7.2	Research Discussion . . . . .	72
7.3	Future Work . . . . .	73
	<b>Bibliography</b>	<b>76</b>

## LIST OF FIGURES

2.1	Web Application Architecture . . . . .	6
2.2	Composition of JS . . . . .	8
2.3	Web Attack . . . . .	10
2.4	Types of XSS attacks . . . . .	11
2.5	Stored XSS process . . . . .	12
2.6	Reflected XSS process . . . . .	13
2.7	DOM-based XSS process . . . . .	14
2.8	Machine Learning Algorithms . . . . .	15
3.1	Model Creation Steps . . . . .	27
3.2	XSS payloads collected . . . . .	28
3.3	Extract XSS features process . . . . .	36
4.1	XSS evaluation process . . . . .	39
4.2	Create model process . . . . .	40
4.3	Cross-validation process . . . . .	45
4.4	Random Forest Process . . . . .	46
5.1	Ensemble Framework . . . . .	54
6.1	Chrome Extension System Framework . . . . .	62
6.2	Backend Service implementation . . . . .	65
6.3	Chrome Extension Directory Structure . . . . .	66
6.4	Chrome Extension Directory Structure Complex . . . . .	67
6.5	Chrome Extension Manifest File . . . . .	68
6.6	Popup HTML code . . . . .	69
6.7	Popup JS code . . . . .	69



## LIST OF TABLES

2.1	Confusion Matrix . . . . .	18
2.2	Literature comparisons in static and dynamic analysis . . . . .	22
2.3	Predicting results of the literature [0] . . . . .	22
2.4	Predicting results of the literature [0] . . . . .	23
2.5	Predicting results of the study [0] . . . . .	23
2.6	Predicting results of the hybrid and semi-supervised learning approach . . . . .	23
2.7	Predicting results of the study[0] . . . . .	24
2.8	Predicting results of the literature[0] . . . . .	24
2.9	Predicting results of the research [0] . . . . .	24
2.10	Predicting results of the research [0] . . . . .	24
2.11	Predicting results of the model[0] . . . . .	25
2.12	Predicting results of the approach[0] . . . . .	25
2.13	Predicting results of the research [0] . . . . .	25
2.14	Ensemble-based Machine Learning (ML) Techniques . . . . .	26
3.1	Collected datasets . . . . .	28
3.2	Non-Alphanumeric Features . . . . .	30
3.3	Alphanumeric Features . . . . .	32
3.4	Alphanumeric Features . . . . .	33
3.5	Alphanumeric Features . . . . .	34
3.6	Features extraction of malicious payloads . . . . .	35
3.7	Features extraction of benign payloads . . . . .	35
4.1	Linear Regression Evaluation . . . . .	39
4.2	Decision Tree Evaluation . . . . .	41
4.3	Support Vector Machine Evaluation . . . . .	43
4.4	K-Nearest Neighbors Evaluation . . . . .	44
4.5	Random Forest Evaluation . . . . .	46
4.6	Naive Bayes Multinomial Evaluation . . . . .	48
4.7	Simple Classifiers Execution Time . . . . .	49

## LIST OF TABLES

---

5.1	Soft voting classifier probabilities . . . . .	53
5.2	Evaluation: Random Forest with Decision Tree . . . . .	55
5.3	Evaluation: K-Nearest Neighbors with Random forest . . . . .	56
5.4	Evaluation: K-Nearest Neighbors with Decision Tree . . . . .	57
5.5	Ensemble Classifier Execution Time . . . . .	58



## GLOSSARY

<b>Browser</b>	A web browser or browser is application software for accessing the World Wide Web. When a user requests a web page from a particular website, the web browser retrieves the necessary content from a web server and then displays the page on the user's device. 5, 8, 11, 13
<b>cookie</b>	A short text file saved on the PC of the user A cookie's maximum file size is 4KB. An HTTP cookie, also known as a web cookie or an internet cookie, is a type of cookie that is stored on a user's computer. When a user visits a website for the first time, the site transmits data packets to the user's computer in the form of a cookie. 7, 11
<b>RESTful or REST API</b>	A REST (representational state transfer) API is an application programming interface that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding. 64
<b>session</b>	Used to temporarily store data on the server so that it can be used across multiple pages of the website It refers to the total amount of time spent on a particular activity. When a user registers in to a network application, the user session begins and ends when the user logs out of the program or shuts down the machine. 11

**token** Can be used in place of or in addition to a password. It functions as an electronic key that allows you to gain access to something. A wireless keycard, for example, can enter a locked door, or a customer seeking to access their bank account online can use a bank-provided token to show that they are who they say they are. 11

**Website** A website (also written as web site) is a collection of web pages and related content that is identified by a common domain name and published on at least one web server. 11

# ACRONYMS

<b>AI</b>	Artificial Intelligence 19
<b>AJAX</b>	Asynchronous JavaScript And XML 7, 13
<b>API</b>	Application Programming Interface 8, 14, 47, 60, 61, 63, 64
<b>BOM</b>	Browser Object Model 8, 14
<b>CLI</b>	Command Line Interface 66
<b>CSS</b>	Cascading Style Sheets 6, 60, 62, 63
<b>DDoS</b>	Distributed Denial of Service 2
<b>DOM</b>	Document Object Model 8, 13, 14, 31
<b>Gnb</b>	Gaussian Naive Bayes 25, 26, 37
<b>HTML</b>	HyperText Markup Language 5, 6, 7, 9, 10, 29, 31, 32, 60, 61, 62, 63, 65, 69
<b>HTTP</b>	Hypertext Transfer Protocol 4, 27, 28, 29, 34, 35, 69, 73
<b>HTTPS</b>	Hypertext Transfer Protocol Secure 34, 35
<b>JS</b>	JavaScript 3, 5, 6, 7, 8, 9, 10, 11, 14, 21, 22, 27, 29, 31, 34, 35, 60, 61, 62, 63, 65, 66, 67, 69
<b>KNN</b>	K-Nearest Neighbor 18, 22, 25, 26, 37, 43, 44, 48, 49, 51, 56, 57, 59, 72, 73, 74
<b>ML</b>	Machine Learning ix, 14, 15, 16, 22, 25, 26, 37
<b>NPM</b>	Node Package Manager 66
<b>OWASP</b>	Open Web Application Security Project 1

<b>RBF</b>	Radial Basis Function 42, 71
<b>RegEx</b>	Regular Expressions 34
<b>SVM</b>	Support Vector Machine 22, 23, 24, 25, 37, 42, 48, 72
<b>URL</b>	Uniform Resource Locator 10, 13, 14, 31, 34, 35, 60, 64, 70
<b>Web</b>	World Wide Web 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 22, 27, 30, 31, 37, 41, 51, 57, 60, 62, 63, 64, 65, 71, 72, 73, 74
<b>XSS</b>	Cross-Site Scripting 1, 2, 3, 4, 5, 11, 12, 13, 14, 20, 21, 22, 23, 24, 25, 26, 27, 28, 31, 37, 41, 42, 43, 45, 47, 48, 49, 51, 53, 55, 56, 57, 58, 59, 60, 62, 63, 64, 71, 72, 73, 74

# INTRODUCTION

This chapter will begin with analysing the backgrounds of the Cross-Site Scripting (XSS) and introducing the recent development of the XSS detecting systems. With different systems, it explains the disadvantages and how to evolve to take better revolution for this research. Also, it elaborates the main work and the organizational structure of this research.

## 1.1 Motivation

As the technology evolves, a huge number of World Wide Web (Web) applications have appeared. The real world's information has been stored to a virtual world. Since then, the growth of the internet has brought us a lot of convenience. However, the risks are coming at the same time. Consequently, the security of the Web applications has been an indispensable part for the protection of our privacy towards different technology devices through the network. Therefore, how to defend or prevent and detect against those vulnerabilities, has become an important topic.

Based on Open Web Application Security Project (OWASP)'s Top-10 Project[0], the XSS is now a part of Injections. It has become one of the most common and critical Web attacks. For further information about top vulnerabilities, it can be found below:

1. **Broken Access Control:** By altering URLs and HTML pages, an attacker can get around access control measures. They can also gain unauthorized access (vertical and parallel) to sensitive resources, impersonate users, administrators, or privileged users, and create, access, update, or delete any record.
2. **Cryptographic Failures:** The system suffers damage due to the leaking of sensitive data caused by the use of outdated, unencrypted, or weakly encrypted hash functions like MD5 or SHA1, the default encryption key, or frequent usage of a weakly encrypted key.
3. **Injections:** Through a SQL injection vulnerability, a hostile attacker can create SQL injection statements, send specific error information to the server to gather relevant information, alter the contents of the database, and exercise their right to withdraw.

4. **Insecure Design:** A design flaw in the development process could result in an injection exploit, file upload, and other problems.
5. **Security Misconfiguration:** This points to a significant number of wrongly configured security settings, such as a completely insecure application stack, improperly configured cloud server authorization, functionalities that are deployed or activated without a need, etc.
6. **Vulnerable and Outdated Components:** One example of this type of vulnerability is the inability of administrators to swiftly assess the security posture of the components since they are uninformed of the version of every used component.
7. **Identification and Authentication Failures:** To stop threats connected to authentication, user authentication, authentication, and session management are necessary.
8. **Software and Data Integrity Failures:** Code and infrastructure are related to errors in software and data integrity. For instance, dangerous deserialization is easily caused if an object or piece of data is encoded or serialized into a structure that an attacker may view and alter.
9. **Security Logging and Monitoring Failures:** This category is intended to support active intrusion detection, upgrading, and response. It is impossible to find infractions without logging and observation. At any time, insufficient logging, detection, monitoring, and proactive response may take place.
10. **Server-Side Request Forgery:** When a Web application obtains a remote resource without checking the URL the user gave, SSRF flaws happen. Even in the presence of a firewall, virtual private network (VPN), or another kind of network access control list (ACL) security, it enables an attacker to force a program to submit a well prepared request to an unexpected location.

XSS attacks start up on the client side browser or frontend. For itself it can get user's password directly or it can be just the initial setup for the next attack. The followings are some examples about precautions of XSS:

- To get the client's cookie, to block browser's communication and to execute bad behaviors with other information of the user
- To plug trojan in Web pages for controlling the clients computers
- To trick users to access privacy information such as credit card password by phishing
- To govern the users as span and perform a Distributed Denial of Service (DDoS)
- To spread anti-virus to damage the network environment

Hence, for the network security, it is needed to detect and defend against the abnormal harms. To sum up, the XSS is becoming the most threatening invasion. To conquer it has been the essential research topic in Web security in nowadays.

## 1.2 Research Question

In the era of Internet technology with such rapid development, it is becoming more and more normal to deal a lot of security problems. The XSS is one of them. Even though the XSS attack detection study is keeping going, the techniques is still facing serious challenges. Not only the detection methods evolve, the attacks also become complex and challengeable. Consequently, if the detection methods are being stagnant, it will be unable to cope with the complex and challengeable XSS injections.

With more research, it was discovered that XSS assaults on existing online applications are simple to carry out, but difficult to detect and prevent due to the complicated syntactic JavaScript (JS) source code. This is the starting point for investigating the following primary question: **What is the efficiency and performance of machine learning techniques for XSS detection?**

## 1.3 Research Objective

Machine Learning has been used in image processing, voice recognition, machine translation, etc. In the big data environment, machine learning suits very well on huge and complex tasks. The main purpose of this dissertation are the detection based on machine learning, and how the research is going to use the machine learning to solve the difficult problems that still exist in detection of XSS vulnerabilities.

By analyzing the current situation, the study concludes the insufficient parts (efficiency of using machine learning to detect XSS vulnerabilities) of the literatures presented above. Hereupon, by reviewing the gaps propose some possible solutions. Specially, the study can be sum up to the followings:

1. (G1) To deeply understand about the behaviors and trigger mechanism of XSS problems.
2. (G2) To find the best algorithm that can automatically detect through the features analysis.
3. (G3) To test and evaluate the results in real time.

To reach the goals defined, the proposed research will focus on features extractions and machine learning algorithms. The key aspects to focus can be listed as follows:

1. Collecting a large number of datasets

2. Preprocessing the data: sanitization, tokenization and vectorization
3. Training the datasets and testing them

As previously stated, it is possible to verify if this model can overpass the problems discussed in section 1.2, and take suggestion for future works.

## 1.4 Reading Guide

The main structure of this dissertation is organized as follows:

The first chapter is the introduction. This chapter will present the background and the objective of the research first. Besides, a brief understanding about the recent researches will be discussed. At the end, it proposed the main work and study content of the research.

The second chapter is related to studies that have been done under the XSS attack topic. This chapter begins with XSS attack theory, its types and its principle. After that, it introduces the principal machine learning techniques, proposing the possible detection model.

The third chapter is dedicated to model preparation, which includes dataset gathering and XSS feature extraction.

The fourth chapter discusses many approaches to classification algorithms, including the support vector machine, linear regression, naïve bayes, decision tree, random forest, and k-nearest neighbor tests. After the models are completed, their performance will be compared, and the models with the highest ratings will be chosen.

The fifth chapter discusses ensemble classifications, a mixture of algorithms. It incorporates various classifiers in different ways. The majority of this research is based on the combination principle. Alternately, it indicates that the final class will be the one with the largest portion of the votes. At the conclusion, evaluate the performance of the obtained classifiers and select the top one to use in the next chapter.

Real-time testing and evaluations are covered in the sixth chapter. In other words, it will create a web software that can identify XSS features and assess how dangerous the website the user is now on is. This is constructed of two service layers, one of which correlates to the user interface and displays the app's content to the user, and the other of which corresponds to a model in which the user's inputs or payload serve as input and are used to check for XSS threats as output. The Hypertext Transfer Protocol (HTTP) protocol is being used by them to communicate between two service tiers.

The research will be summarized in the final chapter, which is the conclusion. It also talks about the project and what can be done for upcoming projects.



## BACKGROUND AND RELATED STUDIES

The chapter below focuses on the basic technologies and similar studies which have been involved in the previous researches under the topic, typically including: the basic principle of the World Wide Web (Web) applications, Cross-Site Scripting (XSS) attack and its types, machine learning models and its principle and, finally, the techniques to prevent and detect XSS vulnerabilities.

### 2.1 Web Applications

As the name implies, Web application is a computer program that performs tasks on the Internet using Web Browser and Web technologies. These applications use a browser as the client side, which is called the B/S structure. Through the common use of browsers (like Google Chrome), it does not require installing the specific client application.

From a technical point of view, the application layer protocol for the Web application is based on the HTTP protocol, which is characterized over the network layer protocol (TCP/UDP). This structure helps to improve the efficiency of the network speed.

Web applications are typically written in browser-supported languages such as JavaScript (JS) and HyperText Markup Language (HTML), which rely on the browser to render the program and thus make it executable. Some applications are dynamic, requiring serverside processing, while others are completely static, without any processing on the server. Web applications require a Web server to manage requests from clients, an application server to perform the requested tasks, and sometimes a database to store information. In the figure 2.1, there is an architecture of a Web application.

The following steps are an example of a typical Web application flow:

1. A request to a Web server is triggered over the internet by a user through a web browser or the user interface of an application
2. The server performs the requested tasks, such as querying the database or processing the data, and then generating the results of the requested data.

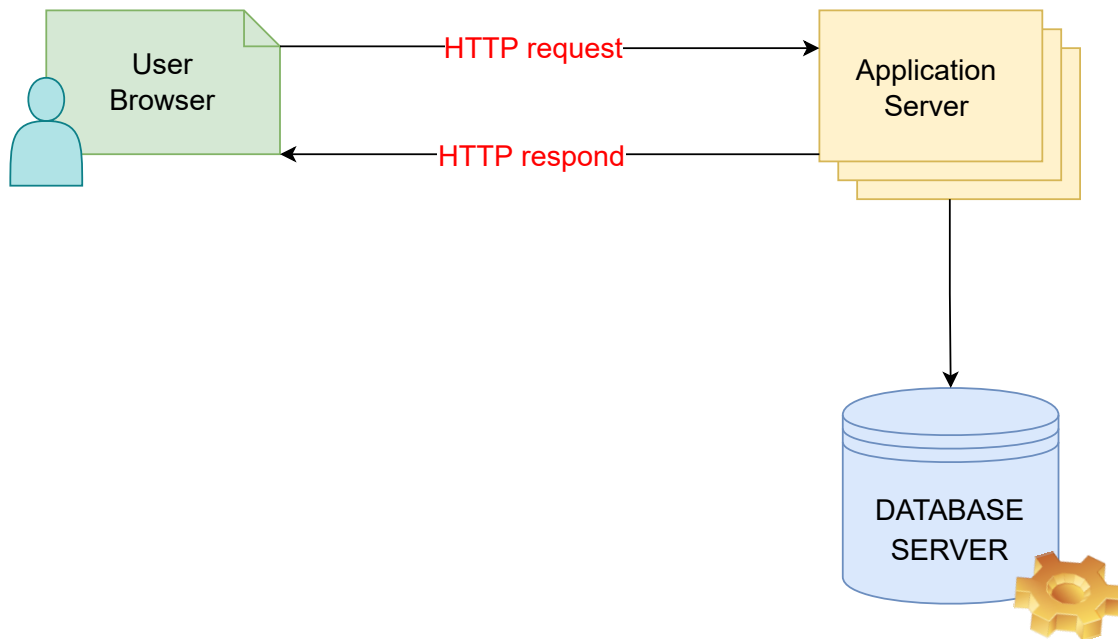


Figure 2.1: Web Application Architecture

3. The server responds back to the client with the processed data or the requested information or the results of the processed data together, which then appears on the user's screen.

Online forms, shopping carts, word processors, spreadsheets, video and photo editing, file conversion, file scanning, and email systems like Gmail, Yahoo and Outlook are examples of Web apps. More concretely, there are Gmail, YouTube, Dropbox, Google Document, and others Web Application products used in office. Among them, documents and calendars can be shared online which realizes that each team member can work on a same document at the same time.

The growth of technology has been influenced by the increase in the use of the Internet. As a result, Web applications are rapidly moving from the old architecture to the cloud model. Users can use those applications anywhere, without installation, thus, improving the productivity and user experience.

## 2.2 JavaScript

Normally, Web pages are built with HTML, Cascading Style Sheets (CSS) and JS:

- HTML: A markup language that defines the web page's backbone.
- CSS: A style language for statically customizing an HTML page.
- JS: A scripting language that can dynamically respond to user input.

JavaScript is built with the intention of "making the page more vibrant". Programs developed under this programming language are called scripts. They can be written directly into a Web page's HTML and run as soon as the page loads. Scripts are provided in plain text format and executed in that, working without any additional setup or compilation.

JS in the browser may perform a wide range of tasks, including:

- Modify the existing content and layout of a web page by adding new HTML.
- Respond to user actions, such as clicking the mouse, moving the pointer, or pressing a key.
- Use Asynchronous JavaScript And XML (AJAX) methods to send network requests to remote servers to download and upload files.
- Get or set cookie, in order to recognize the visitor's identity.
- Keep track of the client's information by using local storage.

All those operations can be done using the JS interpreter (JS engine). It is a component present on the browser, which is commonly utilized in client-side scripting languages.

Here is an example of Web application code (HTML and JS):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Web App</title>
</head>
<body>
  <div class="box">
    BOX
  </div>
  <script>
    const box = document.querySelector('.box') //DOM
    setTimeout(() => { //BOM
      box.style.backgroundColor = 'black'
    }, 1000)
  </script>
</body>
</html>
```

### 2.2.1 JS Features

#### (1) **Dynamic**

JS is an event-driven scripting language that responds to user input without the need for a Web server. It can directly respond to these events by clicking or moving the mouse up or down, changing the window, etc. and so on while browsing a Web page.

#### (2) **Weakly typed**

JS is an interpreted scripting language. Unlike C, C++, and other languages, JS is interpreted line by line as the program executes. The variable types is a weak type, and the data type is not strictly necessary. Its design is basic and compact.

#### (3) **Object-oriented/Prototype-based**

JS is an object-oriented scripting language which uses prototypes to implement class inheritance. The prototypes can bind objects together, giving the related object access to both methods and variables.

### 2.2.2 JS Composition

The basic grammar components (such as operators, control structures, and statements) and the standard library (a collection of objects with various functions such as Array, Date, Math, and so on) are the only two aspects of JS's core syntax (ECMAScript). Additionally, for JS calls, the various host environments give additional Application Programming Interface (API)s. Those APIs correspond to interfaces that can only be used in that context. The additional APIs provided by browsers are separated into Document Object Model (DOM) and Browser Object Model (BOM). So, as shown in figure 2.2, JS programming language has three parts:

1. ECMAScript: describes the syntax and basic objects of the language.
2. DOM: is an API for the methods and interfaces that works with the document content. In the Web every single Web page is also a document.
3. BOM: refers to the APIs, which includes methods and interfaces, related with Browser interaction.

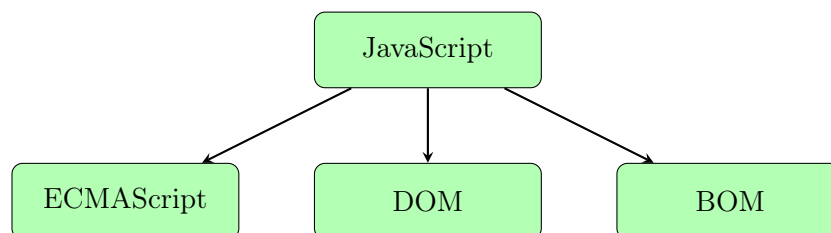


Figure 2.2: Composition of JS

### 2.2.3 Add JS to webpage

#### (1) Inline code

Internal and external JS do not necessitate the use of events in tandem, whereas inline code does.

```
<body>
  <button onClick="alert('JS code here')">CLICK HERE</button>
</body>
```

#### (2) Embedding code

The script tag can be used to insert JavaScript code into a head or body element. Here is an example:

```
<head>
  <script>
    alert('JS code has inserted in head')
  </script>
</head>
<body>
  <script>
    alert('JS code has inserted in body')
  </script>
</body>
```

#### (3) External file

The JS code can alternatively be stored in a separate JS extension file. The script tag must be used to introduce it into the HTML file.

In HTML file, there is an import:

```
<script src="jsFile.js"></script>
```

In a separated JS file, it can directly write JS code:

```
alert('This is a javascript file.')
```

### 2.2.4 JS Security

It is feasible to improve the user experience of Web apps by using JS. However, it also provides opportunities for hackers to launch Web attacks. For this reason, it's also crucial to understand the security of Web apps.

The majority of JS security flaws are created by end-user involvement. Malicious users have access to JS security, which includes detecting, preventing, protecting, and addressing

security concerns in JSbased applications. Cross-site scripting, malicious code, man-in-the-middle attacks, and exploiting vulnerabilities in Web application source code are the most prevalent JS vulnerabilities.

In the circumstance of online attacks, hackers typically embed or insert malicious code into HTML pages, and then execute the code as scripts to achieve the attack's goal. For example, attackers can exploit vulnerabilities in the background management system to gain direct access to database information and administrator privileges, or even control the host remote loading malicious code or software, or redirect the access address to a fake web page when the user browses the web, all with the goal of stealing user password information and obtaining financial gain.

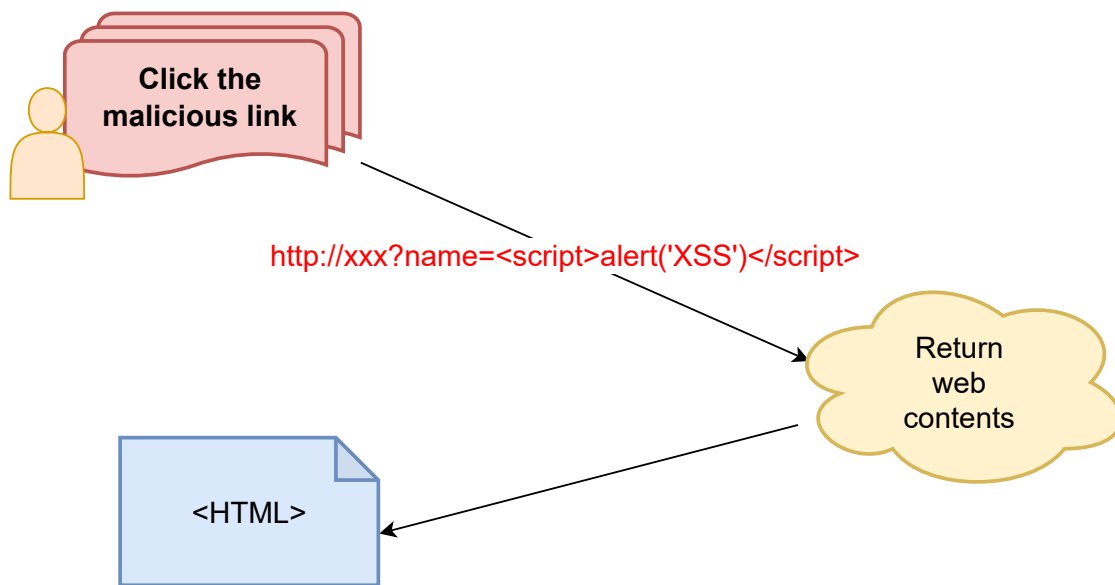


Figure 2.3: Web Attack

As the process described in figure 2.3, during the execution of the JS program, the server cannot determine whether the request is at risk of an attack. As a result, with a variety of malicious attack measures, it can be difficult to detect all types of attacks in particular instances.

## 2.3 Cross-Site Scripting Attack

As many other types of attacks, the success attack is against the input of the web applications, especially the user input or key-value input. In the web applications, the input is mainly from HTML forms, and is delivered to server through GET or POST requests. For GET method, the request is covered in Uniform Resource Locator (URL) a pair of key-values. About the POST method, a pair of key-value is presented on the body of payload. Moreover, the malicious code is saved inside of those key-values.

As far as XSS is concerned, it is introduced as one of the injection attack types, which occurs when the victim visits the Web page, if an attacker has inserted malicious script through the web server and the script is executed on the client-side. The major compromise of XSS attack is the Web user, because it can steal the cookie or session token and other sensitive information that can modify the contents of the Website. As a result, it won't cause damage on the Web server directly.

Taking the online marketplace Website as a demonstration, the buyers are allowed to ask questions about the products in trade which can be answered by the seller afterwards. It seems to be a very common action of exchange. However, it becomes a breakthrough of the attack when it reaches the eyes of the hackers. If a buyer has inserted the question that contains embed JS code, anyone who views this question will get a malicious script executed in his Browser. Thereby, the user will bid a product unconsciously that is not interested in, or the seller will accept the lower bid committed by the hacker to end the trade.

Generally, the XSS can be described in four steps:

1. Constructing malicious code: the attacker insert a malicious code into a webpage and waits for touching off
2. HTTP requests: when the user triggers the bad code it will send a request to server
3. HTTP responses: the server receives and handles the request and sends a response with malicious code to client-side
4. Sensitive data acquisition: browser executes the evil code and the hacker gets the sensitive data, such as cookies or session IDs

In brief, when a Web application is not able to purify the user's input, as a dynamic Web page, it may cause XSS attack. To classify the types of XSS attack, there are three of them, which are reflected XSS, stored XSS, and DOM-based XSS as shown in Figure 2.4.

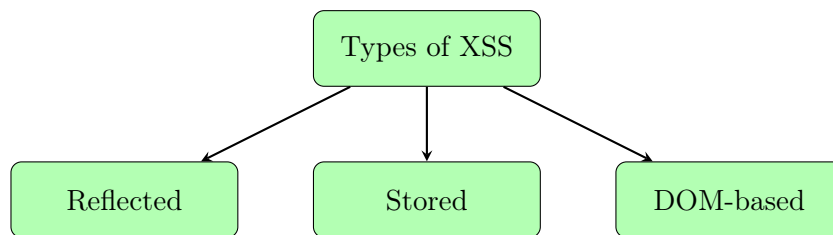


Figure 2.4: Types of XSS attacks

### 2.3.1 Stored/Persistent XSS

In stored or persistent XSS attack (type-I XSS), the malicious script is injected into servers and performs every time when the website is requested. The scenario of the stored XSS can be characterized in phases stated as follow:

1. Hacker adds malicious script on vulnerable website through forms, posts or comments
2. The script is saved into target webpage server's database
3. Victim browses the target website and gets the malicious script from database
4. The script is loaded and it conducts the victim into malicious database
5. Hacker collects victim's private information from the malicious database

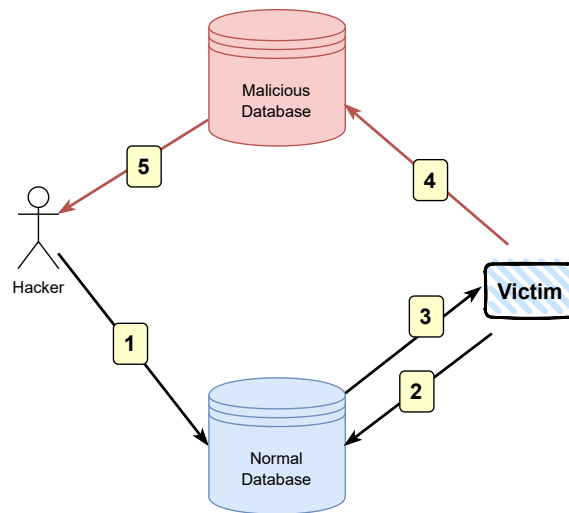


Figure 2.5: Stored XSS process

The malicious XSS code is saved to the database using various syntax symbols, and it only executes when the user views the evil page, as shown in the code below:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title></title>
</head>
<body>
  <p><script>XSS CODE</script></p>
</body>
</html>
```

### 2.3.2 Reflected/Non-persistent XSS

In reflected or non-persistent XSS attack (type-II XSS), the script is only reported on the client-side, which means the server won't keep the script file. This type of hacking can be represented with steps in Figure 2.6:



1. Hacker puts a malicious URL on the website
2. The user visits the target webpage and clicks on the link inserted by hacker
3. Website returns to the malicious page making it look like a normal Browser to the victim
4. On user side, the website parses the XSS script and requests to the malicious web-server
5. Hackers invoke victims to type in their privacy and obtain it

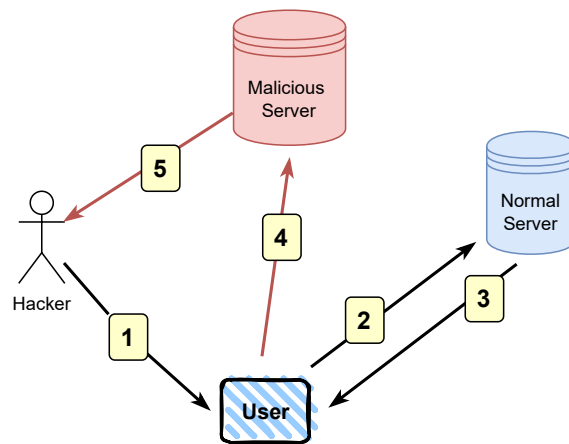


Figure 2.6: Reflected XSS process

If there is an input box (form or search bar) in web application, the hacker may use it to put the harmful code:

```
<script>alert('XSS CODE')</script>
```

With the code above, it is going to reveal in URL as shown format: `http://www.xss.page/search.php?keyword=<script>alert('XSSCODE')</script>`

### 2.3.3 DOM-based XSS

DOM-based XSS attack (type-0 XSS) is also Browser side vulnerability where the script is not sent to server. At this context, hacker makes alterations to DOM elements under document properties. With the increase of the network facilities, the user experience has become progressively popular. Thus, to enhance the performance, the collaborative work between the server and client appears. The client side will display only the minimums of the website, and only when the user needs the browser will request to server for loading more functionalities using AJAX.

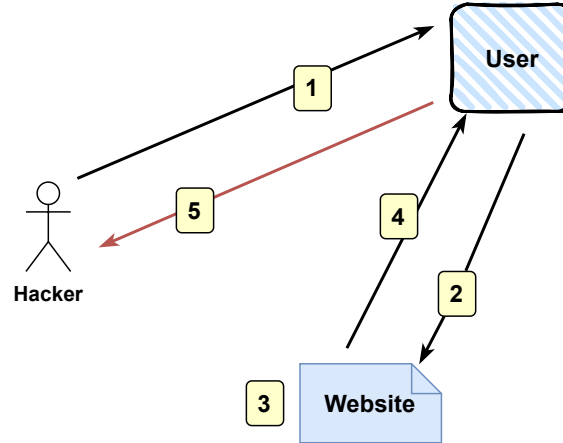


Figure 2.7: DOM-based XSS process

On the web applications, the JS code is always on execution for processing dynamic behaviors. Hence, JS is composed by ECMAScript (the basic language syntax) and Web APIs (DOM and BOM). The webpage can be likened to a document and each website is a file, with which the developers are able to reach the document content through the DOM and BOM. DOM is an object model which is useful to access every HTML element and read or modify. In this kind of attack, the malicious code is processed in client-side. Unlike the reflected XSS, the user does not need to click on any link to start punishing and the data does not reach to server. It is fully carried by JS implementation. As evidenced in Figure 2.7, the procedure of the DOM-based XSS is:

1. Hacker injects malicious script involving DOM
2. User explores the target webpage and triggers the DOM malicious source
3. The page will receive spiteful request
4. The original website will respond and adapt with harmful issues to user's DOM
5. When the DOM XSS fills up, hacker may gather victim's details

The evidence presented thus far supports the idea that modifying DOM elements can be done directly on URL parameters: `http://www.xss.page/search.php?keyword=<script>alert('XSS')</script>`. Since this result has been found similar to reflected XSS, the impacts would be effectively the same.

## 2.4 Machine Learning

In recent decades, the field of Machine Learning (ML) has exploded, not just to include jobs that are not only performed on a regular basis, but also on a more extensive basis. In

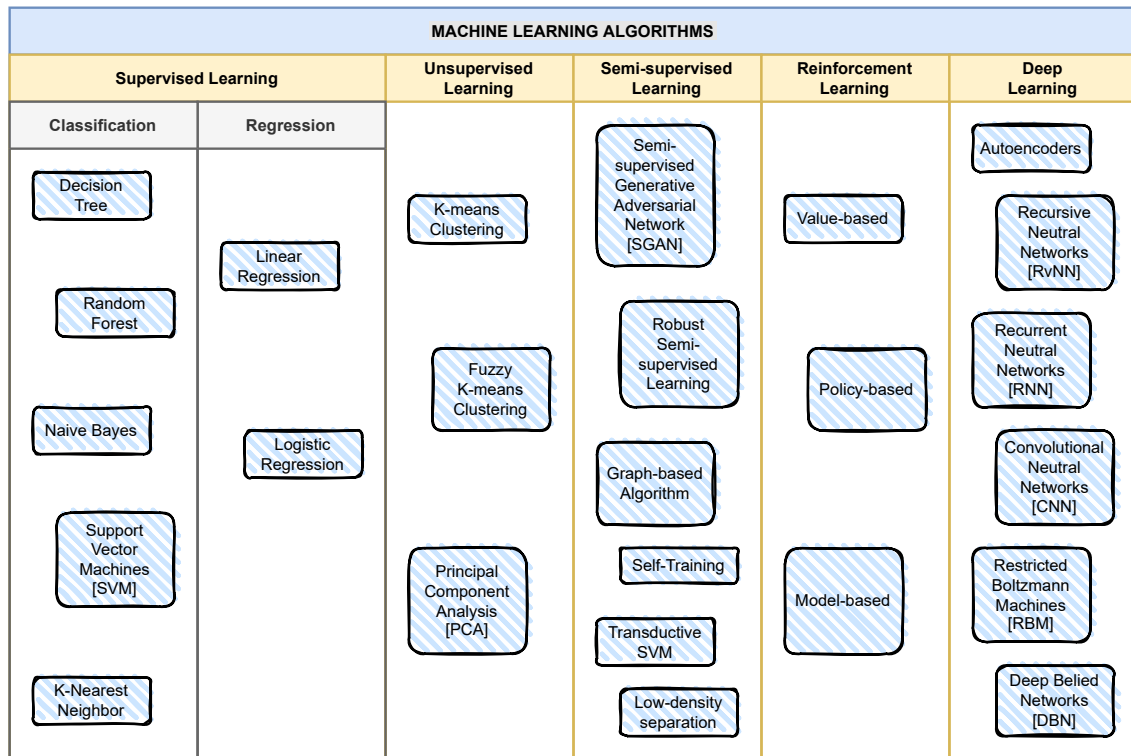


Figure 2.8: Machine Learning Algorithms

general, ML is defined as any stimulus that can be used to create an autonomous system which can perform and master itself after being educated using collected data.

A good example of common use of ML today is a search engine or advertisement recommendation. A descriptive study differs from an exploratory study in that it is analyzed by a set of mechanisms that look for patterns in data to forecast what will happen next.

### 2.4.1 Machine Learning Algorithms

To figure out how ML works, exploring the related algorithms is needed. Those algorithms are separated into five different types, which is furnished in Figure 2.8. Simultaneously, from those algorithms, it might find a classifier that generate a match rule in which the input object is related, in somehow, to output object. Therefore, through the classifier, it may realize a model with the capability to hang out the automatic assignments.

Supervised learning is a goal-oriented based method. Before training, you can clearly know what you can get from there. To be trained, all the datasets need to be afforded with labels. Therefore, with a deterministic result, it can be easily evaluated.

For unsupervised learning, unlike the supervised one, it does not need the labelled data due to its uncertain aiming. That's why the effects are unable to quantify. This type of algorithms is normally used for intrusion detection, user segmentation or recommendation

systems.

Semi-supervised learning is a model with half labelled data and half unlabeled as indicated by its naming, but usually, it has more unlabeled data than labelled. Although it seems advantageous for containing a lot of unlabeled data, the semi-supervised learning model demands much more rules. Therefore, some requirements are actually quite overwhelming in practical application. However, under the big data era, it still hard to collect data with label. In this case, it shows its potential.

Reinforcement learning has a very straightforward principle. Once it gets the good outcome, it will strengthen the same strategy to achieve the best result. Compared with the supervised and unsupervised systems, this system mainly differs that it learns some abilities by endless attempt instead of being feed with a massive dataset. Currently, the reinforcement learning is not mature enough, the application sceneries are still limited. Whereas it is very popular in big game projects, such as "AlphaGo Zero", "Open AI", etc. In addition, it also appears in many other fields like balancing control, reasoning ability and trace tracking.

The deep learning concept derives from Artificial Neural Network. However, it is unequal to Neutral Network but an upgrade based on it. The deep learning is built based on the flows, which is a complex data processing. It can be assumed that all information is a "electricity flow". The input is a power supply, allowing the current to flow to destiny through a number of interrupters. The big tasks are divided into small ones and set in multiple layers of the interrupters which are all connected, forming a fully connected system. This type of model can apply some actions: sanitizing and labelling data, normalizing, denoising and reducing dimension. The major disparity to the above algorithms is the artificial, which means that the features are extracted automatically by machine. Consequently, it performs oneself well but it has poor interpretability.

After reviewing different types of ML algorithms, this research is going to focus on the supervised learning algorithms. The workflow of this ML model can be defined as the follows:

1. Choose the suitable mathematical model
2. Provide some known "questions and answers"(training sets) to system
3. Get the training methodology
4. Give new questions to get answers from the methodology (test model)

The supervised part is interjected into the above-mentioned topic 2. As shown in figure 2.8, it is also critical to comprehend the two types of supervised learning jobs. The regression is a method for predicting both continuous and discrete numerical values. And then there's the classification, which deals with any remaining concerns in discrete mode.

### 2.4.2 Supervised Learning

Supervised learning has three key elements to consider:

1. Model: review the internal logic, describe the system using mathematical terms
2. Strategy: select the optimal evaluation criteria for best model
3. Algorithm: select the specific method of the best model

Simultaneously, the two main types of supervised machine learning problems as mentioned before are classification and regression, which are, respectively, responsible for predicting the class it belongs to (discrete or categorical output), and predicting an exact value (continuous output).

#### 2.4.2.1 Classification

When the output variable has a finite scatter value, the prediction issue becomes a classification learning system model in supervised learning. A classification model or classification decision function, known as a classifier, is learned from data via supervised learning. Classifiers predict new inputs and export classes or targets/labels or categories. Classification problems involve two processes: learning and classification. In the process of learning, according to the known training data set, the machine learns the classifier. During the classification process, the new input is tested using the learned classifier to get categories.

To sum up, the key point of the classification problem is to predict category labels from a predetermined optional list. Its problems can be divided into three groups:

1. Binary classifier: a special case of making a distinction between two categories, such as 0 or 1, yes or no.
2. Multi-class classifier: making a distinction between more than two categories, like the grades.
3. Multi-label classifier: turn a problem into one or more single-objective classification problems or regression problems, e.g. B. the division of a film into action and crime. In the same case, there can be multiple tags, or it can be divided into multiple classes. And what differs to the multi-class is that multi-class has only one tag each.

#### 2.4.2.2 Regression

In contrast to classification, the regression has a purpose of predicting a continuous value, which maps input variables to continuous output variables. A simple example would be to predict the electricity consumption, money spent, etc.

Regression problems can get target algorithm based on the provided input, which is called regression equation. The coefficient is predicted through the method below:

1. Plot all input data into a graph
2. Get the most probabilistic line or curve with a coefficient (the minimum between the line and the points)

### 2.4.3 Evaluation

After training a model for the classification task, the model's performance must be evaluated. An evaluation of the models is performed to determine the value of various classifiers in terms of demonstrating the speculation with respect to their presentation once prepared. To survey the exhibition of the models, cross-validation, confusion matrix, and scores such as accuracy, precision, recall, and F1 are used.

#### 2.4.3.1 Cross-validation

Cross approval [0] is a model assessment technique that separates the preparation dataset into two sections. The first is utilized to prepare the model and the second is utilized to test the model. The most widely recognized explicit strategy in such manner is k-crease cross approval, in which the preparation dataset is separated into  $k$  subsets,  $k - 1$  of which are utilized for preparing and the excess for testing. This last option process is rehashed  $k$  times, with each time involving an alternate subset for testing and the leftover subsets  $k - 1$  for preparing. The typical misunderstanding is that all outcomes related to all subgroups are not completely settled. The K-Nearest Neighbor (KNN) adopts this strategy in the current study to find the best  $k$  in the range of 1 to 11.

#### 2.4.3.2 Performance Scores

The confusion matrix methodology, as demonstrated in table 2.1, was used to test and analyze the performance of the classifiers, which is the reason why it is vital to understand it before digging more into these scoring notions. For each class, this table illustrates both the correct and wrong classifications that occurred.

Table 2.1: Confusion Matrix

		Predicted Class	
		ClassA	ClassB
True Class	ClassA	TP	FP
	ClassB	FN	TN

The model is tested after training for each classification task, and the following four situations happened:

1. **True Positive (TP):** The malware payloads have been classified as malicious.

2. **False Positive (FP):** The harmful payloads are categorized as benign.
3. **False Negative (FN):** The malicious payloads are sorted as harmless.
4. **True Negative (TN):** The benign payloads are assigned as benign.

As can be seen, TP and TN predict positive outcomes, while FN and FP predict negative outcomes. It's significant to note that the motivation behind this investigation is to avoid adding to the number of fictitious problems, which is a drawback of the Artificial Intelligence (AI) technique. A false positive is a phony problem that poses a risk to the security of online applications in this examination. This shows that the classifier has classified a malicious payload as harmless, allowing the malicious payload to be stored in the web application data set. For the web application, this is a high-risk situation. Because the harmless payload is segregated and not authorized to be put away in the web application data set, bogus negatives, in which a harmless payload is erroneously designated as hazardous, can cause an application failure. This circumstance causes problems with the program's functionality, but it is safer in terms of security and does not put the web application in danger. The accuracy, precision, sensitivity (TP rate), and specificity are all factors in determining efficacy (TN rate) of the trained model.

#### 2.4.3.3 Accuracy

Regardless of whether the predicted samples are positive or negative, accuracy is defined as the ratio of accurately predicted samples to total predicted samples. The proportion of accurately predicted data points to the total number of data points. This value is calculated by dividing the total number of true positives and true negatives by the total number of data points, and the formula 2.1 demonstrates how to calculate the accuracy value. Many of the classifiers used here, for example, can be evaluated based on whether they classified malicious payloads as malicious and benign payloads as benign.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.1)$$

#### 2.4.3.4 Precision

Precision is defined as the ratio of accurately predicted positive samples to the number of samples predicted to be positive, or how many of the samples projected to be positive are real positive samples. Precision only includes parts of the forecast that are positive samples when making a contract with accuracy, whereas accuracy considers all samples. If there are no false positives but only real positives, the accuracy is 1. The computation in equation from is:

$$precision = \frac{TP}{TP + FP} \quad (2.2)$$

### 2.4.3.5 Recall

The possibility of correctly anticipating a positive sample from the first sample is known as the recall rate. In other words, it's the ratio of the number of true positive samples to the number of accurate positive samples. The formula below clearly shows how to estimate recall.

$$recall = \frac{TP}{TP + FN} \quad (2.3)$$

### 2.4.3.6 F1 Score

When FN and FP are both 0 and their values are 1, high precision and recall are required in practice. They are mutually exclusive in other situations, such as this one, where 50 positive samples and 50 negative samples all predict a positive outcome with a precision of 1 and a recall of 0.5. As a result, a compromise must be found, and the F1-score measurement is born. The harmonic average index of these measurements is indicated by the F1-score, which pertains to both precision and recall. From the calculation equation 2.4, the F1-score drops as either recall or precision decreases, and vice versa.

$$F1 - score = \frac{2 * precision * recall}{precision + recall} \quad (2.4)$$

### 2.4.3.7 Execution Time

A key performance metric for a good model is the execution time. The execution time, as its name suggests, is the amount of time required for the code to run through the computer (computer). And it goes without saying that the time will vary with diverse machine capabilities possibilities.

## 2.5 Existing XSS Detection Mechanisms

A more detailed explanation of different XSS detection mechanisms is given in the following section. It is important to review detections of different system, for taking the most desirable measure by analyzing their benefits and obstructions. Those mechanisms are classified as standard methods, source code analysis and machine learning approaches as described below.

### 2.5.1 Standard Methods

The standard methods refer to the principal measure against XSS attack which are filtering and sanitization. From the web developers' perspective, it is their responsibility to do some measure to prevent those kinds of security problems. It can be categorized into:

- (A) **Input Validation:** syntactic and semantic part of the input data is reviewed to confirm if it is harmless



- (B) **Input Filtering:** it detaches any special character with particular meaning or just restricts the length
- (C) **Input Escaping:** this technique makes sure that the data is secure before passing into code execution, whose function is to encode special characters, such as '<', '>', '/' ...
- (D) **Blacklisting Values:** it congests certain behaviors values
- (E) **Content-Security Policy:** it can restrict the JS code allowed on the website
- (F) **Trusted Types:** the input data arguments are required by centralized policy code

### 2.5.2 Static/Dynamic analysis

On the other hand, in spite of these standard preventions, other works have been done. The current study focuses on source code of the program, which analyzes in static and dynamic mode.

In static studies, the root code of the program is tested to identify susceptibilities beyond the code. Therefore, it scans or crawlers the web application and gets the code and its properties. According to this, it is able to predict and prevent the future actions. However, the JS is a dynamic language, as a static analysis, with lack of strict type of information that will make this study more challenging.

In contrast, dynamic analysis checks the program code under real time execution, but it cannot access the code that is not executed yet. It is also frequently verified under JS language. This kind of method will incur runtime overhead and require a lot of engineering work to modify the complex runtime environment.

In the work[0], Vogt et al. exploit code by tracking sensitive data. If the code is transferred to a location other than the server, the page will display a request authorization for the user to pick from. In essence, it can be described as a client-side supplementary protective layer.

Kirda et al.[0] introduces Noxes, which operates as a mini server, it represents a gateway between client and real server. Noxes can automatically or manually generate rules that can mitigate XSS attacks.

Halfond et al.[0] studied the prototype tool called SDAPT, with which it is possible to implement a system that is composed by two steps. Through static analysis it learns the input vectors. And with dynamic analysis, it is possible to discover the vulnerabilities in real-time execution of the web application.

Shar et al.[0] employed a hybrid (static and dynamic) analysis to find all potentially hazardous input code. As the first step, it plays a critical function. It is feasible to train the data and use machine learning methods to create a model that detects and prevents XSS vulnerabilities using those properties.

The above approaches are summarized in the table 2.2. In addition, the limitations are reviewed, since they are useful suggestions for this study.

Table 2.2: Literature comparisons in static and dynamic analysis

Literature	Technology	Limitations
[0]	Tainting and static analysis	Taint variables relies on dependencies
[0]	Generating rules	Cannot handle DOM based XSS
[0]	Hybrid approach	Bad effectiveness and not automatic
[0]	Hybrid with ML approach	Low accuracy

### 2.5.3 Machine Learning Analysis

Recently, with the growth of the ML popularity, more and more researches have been done under this topic. As a result, the problem reveals an importance to study and implement. This section is going to discuss the diverse XSS attack prevention approaches based on machine learning algorithms.

Choi et al.[0] advised to use N-gram and SVM classifier to determine untrusted input string. The N-gram is a probabilistic category of language model which is represented by endless subsequence of N successive tokens that will be used for the detail abstraction. SVM algorithm is adopted for nonlinear partition. The tokenizer takes over the validate data and breaks data into tokens and decodes it. Although, the tokenizer must be trained repeatedly for latest harmful codes.

Table 2.3: Predicting results of the literature [0]

Algorithm	Recall	Precision
Support Vector Machine (SVM)	0.985	0.98
KNN	0.978	0.974

Gupta et al.[0] proposed SVM, NB, Bagging, J48 and JRip classifiers (machine learning) to deduce user input situation attributes and other basic attributes (input, output, validation and sanitization routine characteristics). The table below shows the results of this study.

Another work from Gupta et al.[0] shows that the automatic system can spot JS evil code, which is estimated by three key representatives of PHP Web page: XSS Attack Vector Injector, HTML Crawler and Script Locator. In this system, the detection rate has reached 80%.

Table 2.4: Predicting results of the literature [0]

Algorithm	Accuracy	Precision
SVM	0.889	0.887
Naive Bayes	0.695	0.678
Bagging	0.926	0.934
Random forest	0.876	0.869
J48	0.92	0.929
JRip	0.836	0.992

Shar et al.[0] recommended prognosis based on classification and clustering mechanism to process the attack. To reach the hybrid features to anticipate XSS vulnerabilities via supervised and unsupervised learning, it starts from a static process that prepares an organized security-related nodes, and then those nodes are sent for paired function to be categorized.

Table 2.5: Predicting results of the study [0]

Algorithm	Recall	Precision
Logistic regression	0.86	0.84
Multi-Layer perceptron forest	0.78	0.89

Another work from literature[0] supplements their prediction to enhance the precision by increasing the benefits came up with the hybrid study. In addition, more evaluations must be done to take the advantages of integrating various proposals in prediction model. The related results about XSS detection are presented in table below:

Table 2.6: Predicting results of the hybrid and semi-supervised learning approach

Algorithm	Recall	Precision
Logistic regression	0.73	0.61
Random forest	0.72	0.70

The study[0] implemented the model which was built in two major steps: first, to create a features extractor that can get the feature on its own, and secondly, using the features, the computer can identify the feature as harmful or not.

Rui wang et al.[0] present a detection method with their own input vector features extraction, which is based on the n-gram model. Then the study has also made a combination between diverse classifiers, and finally created their own classifier. The author suggested

Table 2.7: Predicting results of the study[0]

Algorithm	Recall
ADTree	0.952
ADABoost.M1	0.958

two implementations. The first one is based on classifier to detect in code level, and then uses the improved n-gram. The second implementation is based on the first but beyond it. Finally, it uses another classifier to conclude the detection system.

Table 2.8: Predicting results of the literature[0]

Algorithm	Recall	Precision
1st implementation	0.965	0.900
2nd implementation	0.954	0.918

[0] suggested the utilization of supervised learning algorithms to detect XSS attacks. It has two phases, the first is collecting malicious code features, and then the authors used SVM classifier to train the machine over the dataset collected and preprocessed with extracted input vector features.

Table 2.9: Predicting results of the research [0]

Algorithm	Accuracy	Precision
SVM	0.954	0.956

Zhou et al.[0] proposed the model with Bayesian network, which is based on the features extracted from ontology of XSS. To increase the performance, the authors have combined the bagging and majority voting method to get an ensemble technique.

Table 2.10: Predicting results of the research [0]

Algorithm	Accuracy
Proposed model	0.985

Sharma et al.[0] has aimed for a model which is based on three classification algorithms: J48 (decision tree), one rule (rule system) and Naive Bayes. It uses the J48 to evaluate the algorithm chosen and it can improve the performance of it. The one rule is used for choosing the best rule to fit the detection. And the Naive Bayes is for probability determination of the issue belonging to each class.

Table 2.11: Predicting results of the model[0]

Algorithm	Recall	Precision
J48	0.945	0.947
Naive Bayes	0.94	0.945
One rule	0.925	0.931

In[0], the authors resolved the problem based on generic algorithm and reinforcement learning to deal with payload patterns. Due to the generic algorithm, it is necessary to find the chromosomes that is the features extracted of the malicious code. After it, the machine is trained and fitted with those patterns and it can find the similarities for the input source, which is XSS code.

Table 2.12: Predicting results of the approach[0]

Algorithm	Recall	Precision
Proposed model	0.998	0.999

Munonye et al.[0] has contemplated seven supervised learning algorithms. In this model, the system takes the relationship as a key point based on the state transitions of the input parameters. Through this, it is possible to generate different state on the output. And it just needs to focus on the workflow. Thus, if there is a change between input and output, it means there is a vulnerability.

Table 2.13: Predicting results of the research [0]

Algorithm	Recall	Precision
Gradient Boosting	0.747	0.803
Random Forest	0.677	0.782
Naïve Bayes	0.700	0.694
Decision Tree	0.718	0.735
KNN	0.690	0.751
Logistic Regression	0.795	0.786
SVM	0.689	0.714

A recent research [0] implemented ensemble-based ML method, which is composed by five classification algorithms: linear regression, decision tree, k-nearest neighbor, SVM and Gaussian Naive Bayes (Gnb). This is a normal classification method but based on those

five algorithms instead of only one. This technique has reached high accuracy and it has been compared to each of the algorithms in use (table 2.14).

Table 2.14: Ensemble-based ML Techniques

Algorithm	Accuracy	Precision
Linear regression	0.71	0.706
Decision tree	0.971	0.971
KNN	0.885	0.887
SVM	0.725	0.722
Gnb	0.617	0.671
Ensemble	0.981	0.982

As machine learning has become popular, many studies have found a solution to this kind of problem, but it still has its limitations to get a perfect score on an assessment, and it can be helpful for the user in daily life. This is the reason for this study, to create a program that can be easily used in our daily life and that will guarantee our privacy.

## 2.6 Summary

It is easy to recall the flaws of each consideration from several works. Based on the previous work, it is possible to complete a better model to recognize significant vulnerabilities of XSS attacks.

To sum up, after examining the literatures supplied above, the researches may be generalized with the following flaws:

1. The XSS code can be obfuscated to avoid the detecting, resulting in bad readability that cannot be identified.
2. Current technology cannot take advantage of relational information provided by injected XSS code.
3. Combining three or more ML algorithms is required for improved efficiency and performance, which adds to the effort.

As seen, supervised algorithms work better than others. So far, this work has focused on determining the most appropriate combination of two supervised algorithms to achieve the best performance in combination with learning-based automatic text feature extraction.

## DATASETS

This chapter tries to prepare the datasets for them to be trained and tested in real-world scenarios. The process includes benign and malicious data collecting from different Web pages using crawler tools, selecting and extracting identifiable features of the typical XSS attack from collected payloads as well as the process of splitting instances in training and testing data.

### 3.1 Data Collect

XSS is becoming increasingly complex as network technology advances. Because of the prevalence of obfuscation techniques and the rising complexity of semantic reasoning in attack statements, previously identifiable XSS attacks are becoming difficult to detect. The use of obfuscation techniques and the addition of complicated semantic logic make the initial vulnerabilities more difficult to detect. To begin with, the steps to create a template are shown in the figure 3.1 as followed.

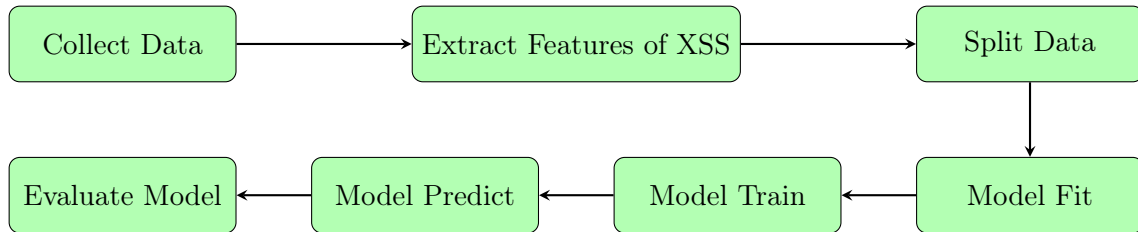


Figure 3.1: Model Creation Steps

This dissertation heavily relies on JS scripts and Hypertext Transfer Protocol (HTTP) payloads, hence using a dataset library is expected. The payloads of JS and HTTP requests are acquired via a crawler using GitHub crawler tools [0], taking the malicious data from [0] site and the benign payloads from trusted web pages.

In simple terms, a web crawler is an automated program that accesses a website, extracts and stores the information it collects. The process usually goes like this: There are usually

many third-party libraries that can help with development. Users can easily use these libraries to simulate real HTTP requests and access requests and responses that can be represented in data structures provided by third-party libraries. Therefore, only part of the body needs to be scanned to analyze the content of the response, useful for later content analysis. The next step is information extraction. In this process the crawler retrieves the information content from the website and this content needs to be analyzed and extracted into the data required by the crawler. Finally, after successfully extracting the information, the robot should save the information data in the specified location.

In this investigation, the template of dataset collected has in total 400000 instances in total, which contains half normal data and half malicious XSS data. For the first analysis, the data is divided into training and testing in a ratio of 80:20. In order for the model to work properly, the 20% of tests are enough to get the final results, because of the large amount of data available, we don't have to compromise and can use more of it to train the model. The table below 3.1 summarizes all the information gathered.

Table 3.1: Collected datasets

Usage	Malicious	Normal
Training	160000	160000
Testing	40000	40000
Total	200000	200000

For a better idea of what the load looks like after the search, it is shown in figure 3.2.

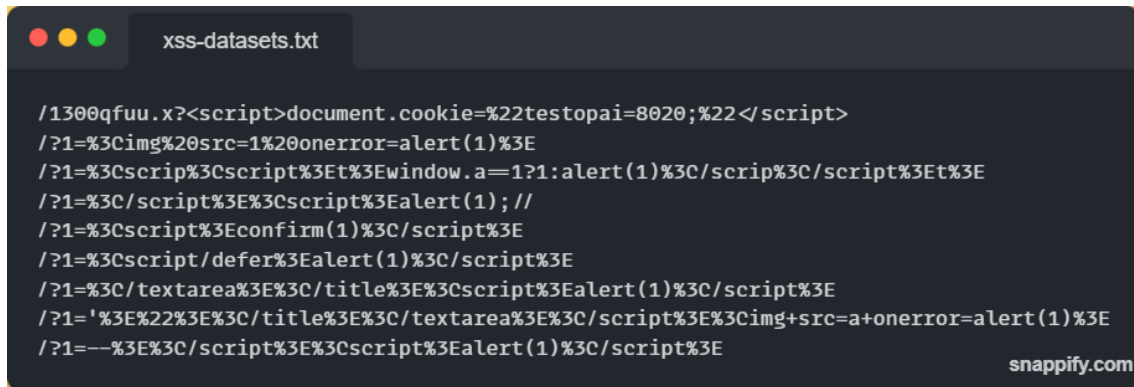


Figure 3.2: XSS payloads collected

## 3.2 Features Extraction

Since the gathered payloads may contain specific issues that cannot help our analysis, and all the instances aren't coordinated in any way, different arrangements should be made



at prior to continuing with include extractions. Here are the means to considerate before passing into model classification:

- **Eliminating additional spaces as well as extra lines:**

The programmers usually use a lot of spaces or extra lines while writing the script code to have a better structure and make the programming process easier. As a result, to avoid the machine taking confusion while analyzing these untreated original codes, it is important to remove all these extras before passing the blacklist filters.

- **Lowercasing all letters:**

The JS language is a case sensitive language, while the HTML language isn't. Hacker can handle diversity of the code using exchange between capital or lower case in order to bypass the normal filters. The simple code '`<iFrAmE sRC="javascript"></IFraMe>`' the tag '`<iFrAmE`' is a HTML tag and "sRC" is the attribute for this tag, so those aren't influenced by JS case sensitive rules. But, the code inside "sRC" needs to be case sensitive. Through these manipulations it is able to bypass some of the rules present in blacklists.

- **Removing duplicated data:**

Duplicated data are not only useless for computer analyze, but also take wrong proportions to it. That is the reason for removing it.

- **Converting ASCII codes:**

Some sensitive and most used codes are normally present in blacklists. To overcome this, the hackers usually use combination of ASCII code and normal letters to disturb the filter list and bypass it. Consequently, this is also an important topic to consider in dataset treatment.

These methods are founded on the principle that everything should be planned in the same way. Alphanumeric and non-alphanumeric characters are two types of characters that must be considered in those HTTP payloads.

As the name indicates, non-alphanumeric category is looking for symbols, punctuations and, combination of both. It is summarized in table 3.2. With an addition of an ordinary accentuation, an attacker might embed JS pernicious code into a site page. With the usage of these non-alphanumeric characters, hackers can add some garbage into a normal script or code. It is a common technique for hacker to bypass the normal network defenses, thus it makes the malicious code more difficult to detect. The logic behind this usage is that the computer cannot resolve the additional unrecognized symbols, and the attackers inject it to make the script seems more confuse so that it cannot be filtered by the normal blacklist. The blacklist is a list of danger stuffs that can be inserted by an attacker, such as tags or attributes or even symbols. When the application receives the user inputs, it will filter it with these blacklists.

For instance, in a normal Web application the brackets ‘<’, ‘>’ are very common in the blacklist. If it is the filter case, the hacker pretend to insert a ‘<’, and then it needs to add one more ‘<’ or other codes to bypass. So, if the hacker intends to insert the code:

```
<SCRIPT>alert ("XSS")</SCRIPT>
```

Then, the code inserted to be filtered should be:

```
<<SCRIPT>alert ("XSS");//<</SCRIPT>
```

It is because the brackets and some punctuations are filtered, so the final result is what the hacker need.

Table 3.2: Non-Alphanumeric Features

Features	Terms
Punctuations	&, %, /, +, ', ?, ., #, =, [, ], \$, (, ), ^, *, -, <, >, @, _, :, {, },  , "
Combinations	"><, "><, [], ==, &#, //, &lt;

Each of the combination of the symbols has different meanings. All those features are described and explained in the table 3.2.

(1) “><, ”><

Normally, those two symbol combinations are used to close the previous tag and begin a new tag with an executable code.

The example of the code is: “”><script> alert("XSS") </script>”, which refers to ><script> alert("XSS") </script>.

(2) []

Through PHP-Shell, it is possible to inject encoded malicious code with those double brackets, thus it can be insert into a server request.

The following is an example of an alert which says 'Hacked' covered with script tags:

```
'( ++[])[ - ^[]] + (![] + [])[ - ~ - ~[]]
+ ([[] + [] + [])[ - ~ - ~ - ~[]] + (![] + [])
[ - ~[]] + (![] + [])[ + []] ); '
```

(3) ==

Hackers can use double equals to perform the logical operations (if, for) or use base64 encoding method to encode vulnerable payloads.

Also as a same example of the base64 encoding for alerting 'XSS' text covered with script tags:

```
PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4=
```

(4) `&#`

Through the Decimal form encryption, hackers are able to avoid the blacklist filters of the HTML tags, and, consequently, insert malicious code.

The code `<IMG SRC=javascript:alert("XSS")>` is the same as encrypted decimal form of the referred HTML entity:

```
\&\#60;\&\#73;\&\#77;\&\#71;\&\#32;  
\&\#83;\&\#82;\&\#67;\&\#61;\&\#106;  
\&\#97;\&\#118;\&\#97;\&\#115;\&\#99;  
\&\#114;\&\#105;\&\#112;\&\#116;  
\&\#58;\&\#97;\&\#108;\&\#101;  
\&\#114;\&\#116;\&\#40;\&\#39;\&\#88;  
\&\#83;\&\#83;\&\#39;\&\#41;\&\#62;
```

(5) `//`

Double slashes helps the hacker bypass the filters of those normal blacklists, normally used in redirecting to other URL or escaping JS vulnerable codes. For instance, the redirection of URL is present in the following code:

```
(<SCRIPT>var a="\ \"alert( \"URL\" );// \";</SCRIPT>)
```

(6) `&lt`

After decoding, the symbol “&lt” is the same as “<”. Thus, the hacker can use it to insert HTML tags, because these tags are normally beginning with “<”. Usually in XSS attacks, the “script” and “img” tag is especially important for inserting JS file. Consequently, hackers intend to take ‘&lt script >’ rather than normal ‘<script>’ tag.

The alphanumeric properties, on the other hand, are crucial for identification. Hackers frequently use keywords or key tags like “script” or “image” to implant harmful executable scripts or malware. The executable code should be written in JS due to the characteristics of Web applications. These JS code should be covered between “script” tags for becoming executable, which is the reason the term “script” is filtered. Other alphanumeric features include DOM events let the script to listen for the target user’s click and redirect the user to the XSS page, which is not declared in the application. The password can then be sent to the hacker from here. This feature’s reasoning is different from that of non-alphanumeric features. Non-alphanumeric qualities can be used to get around the blacklist, but the attack’s performance is largely dependent on alphanumeric ones, so it’s critical that the application can prevent it. All these features can be found in the table 3.3 below.

Table 3.3: Alphanumeric Features

Features	Terms	Description
Readability		If the script is not readable for human, then it has a high possibility that it gets encoded. As a result, it becomes more difficult to decipher and prevent the vulnerabilities.
Objects	document, window, iframe, location	The objects in JS are useful for programmers to use when accessing online APIs. The code can impact the normal application and turn it into a susceptible one by using these APIs.
Events	onload, onerror, onclick...	The events in JS can signify that something dynamic on a web page has occurred, such as user clicks, keyboard inputs, page loading state, or even the specified time countdown. The hacker can introduce code to handle various events in order to force the user to click on a malicious link, allowing the hacker to get access to his machine.
Methods, Functions	String.normalize, Search, Array, eval...	Functions and methods in JS, like any other language, are pre-defined processes that programmers can utilize to solve implicit problems. The hacker can easily reuse them and discover a means to enter harmful code outside the filter's influence by employing a single or combination of the methods and functions. The "eval" function in JS, for example, is used to evaluate a sentence in a JS context, resulting in a JS executable, which a hacker can exploit by inserting harmful code into.
Tags	div, img, script, break, line	As mentioned in section 2.2 of chapter 2, the HTML language is merely a descriptive language, which it is made up of tags. For example, using the "<img>" tag to insert image sources or the "<script>" tag to enter script sources to be recognized by the browser. As a result, rather than entering malicious code directly, hackers might examine susceptible code within a script file or simply introduce a malicious link through the "src" element of the "<img>" tag to request client access, resulting in an XSS attack.

Table 3.4: Alphanumeric Features

Features	Terms	Description
Attributes	src, href, cookie	As described in the tags feature, tags must be used in conjunction with the characteristics of it in order to function effectively. Therefore, this is another crucial criterion to consider. Like the cookie characteristic, this is utilized by the computer to differentiate between distinct users. Cookies are required for all online data that requires user identity to be saved in databases. As a result, cookies behave as a key to gain access to the user's privacy. If the hackers capture this key, they will have access to all the user's personal information. The most obvious example is the use of the credit card. If the user's credit card is linked to the account, the hacker can easily purchase items from online stores and pay with another card.
Reserved words	var, let, const...	The JS programming language, like any other programming languages, features a large number of reserved words. Variable declaration can be done with "var", "const", or "let"; any of these can assist the hacker in defining variables to complete the malicious code of the attack.
Protocol	HTTP, HTTPS	Over a TCP connection, the browser and server can interact using HTTP or HTTPS. These protocols allow the browser to transmit and receive payloads from the server. This method is used by hackers to transmit malware to a server and have them stored, granting them access to all data saved on it. As a result, the payloads of direct network protocols must be filtered before being stored in databases or shown on the client side.
External file	.js, .css	External files that can be inserted into a browser are usually script files in the ".js" format or style files in the ".css" format. External file importation allows a cyber criminal to implant malicious content inside a file, skipping the blacklist, whilst the browser can execute these programs and disrupt the application's normal operation.

Table 3.5: Alphanumeric Features

Features	Terms	Description
JavaScript Command	alert, prompt, console.log	The typical program can easily throw up a redirect link to request the user to click it and take him or her to another susceptible webpage, thereby being attacked, using JS commands like “alert”. Otherwise, if the user does not accept to be redirected to another page, this dialog will repeat itself until the user agrees to proceed to another site. As a result, it is vital to draw attention to and prevention.
Letters, numbers	ASCII codes	In a computer, all the letters and symbols are, at the end, transformed to zeros and ones. Therefore, the hacker can write the planned dangerous program using the most basic symbols - ASCII characters. As a result, the ability of ASCII codes to perplex machines is an important characteristic to consider.

For model training, characteristics extracted from crawling cross-site scripting data are gathered. This investigation has considered four significant features for extraction: the length of the payload, whether it is URL or not, the number of non-alphanumeric (symbols) content, and the number of alphanumeric (words) content. The tables 3.6 and 3.7 give the examples of the malicious or benign payloads and the respective characteristics extracted of them.

The length of each payload is the first characteristic to examine between the instances, as indicated in section 3.2. User inputs, whether it is the messages or the search contents, are usually not as long as the executable code or script file. As a result, the transmitted payload length should be kept to a minimum and so take difference for machine analyzes.

The type of input is the next point to examine. It leverages the terms “HTTP” or “Hypertext Transfer Protocol Secure (HTTPS)” from network transmission protocols to distinguish whether the respect payload is URL or not. It is a URL if the data has the appropriate protocol; otherwise, it is not. Examining the syntax used, as well as the links, files, or JS commands inserted, will make a difference.

It is feasible to generate a dictionary of malicious symbols and words using the tables 3.3 and 3.2 defined above. In order to extract them, Regular Expressions (Regex) methodology is used; the characteristics are simply extracted using this technique and the dictionaries defined with alphanumeric and non-alphanumeric features. The Regex approach uses a standard and agreed-upon format and regular code to get or change anything that follows a stated rule.

Table 3.6: Features extraction of malicious payloads

Payloads	Features			
	Length	Is URL	Symbols	Words
<code>/%3Cscript%3Ealert("XSS")</code>	25	no	7	2
<code>?%3Ciframe%20src=http://xxs.js%3E</code>	40	yes	13	5
<code>/?a=le%3Cimg%20src=1%20onerror=alert(1)%3E</code>	46	no	11	5

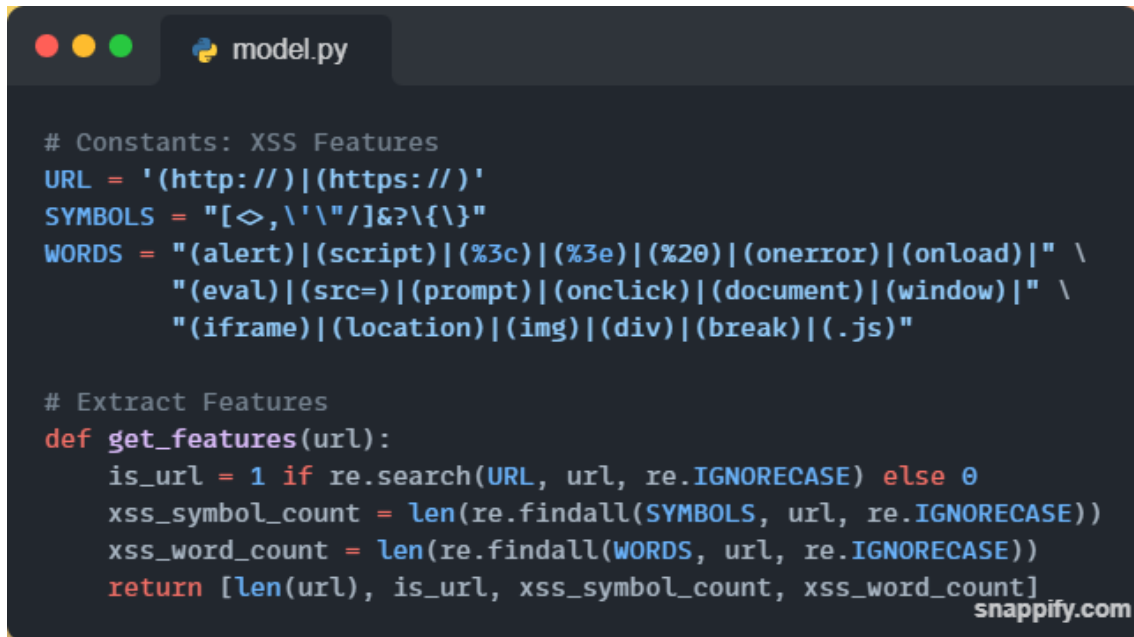
The third payload of the table 3.6 is used to show how it works in further detail. To begin with, the instance is 46 characters long, and all that requires is to count the letters and symbols. Then, if it's a URL, it must have a network protocol, such as HTTP or HTTPS, while it's not a URL if it doesn't. The number of symbols denotes the quantity of symbols present; hence, there are 11 symbols. Finally, the words stand for the frequency of alphanumeric characteristics. There are four: the '<img>' tag, the 'src' attribute, the 'onerror' event, and the 'alert' JS command.

The similar method is used in table 3.7 for benign cases. For same type of analysis, the third message has 22 pieces of writing. It is a URL since it starts with HTTP. It contains 7 symbols and there are no alphanumeric properties.

Table 3.7: Features extraction of benign payloads

NO.	Payloads	Features			
		Length	Is URL	Symbols	Words
1	<code>/stylesheet.php?version=1331749591</code>	34	no	4	1
2	<code>/135482/</code>	8	no	2	0
3	<code>http://.:80/nessus.txt</code>	22	yes	7	0
4	<code>/javascript/javascript.key</code>	26	no	3	2

Figure 3.3 shows what happens in a real project and how the feature extraction process is implemented in Python language.

A screenshot of a code editor window titled 'model.py'. The code defines constants for XSS features and a function to extract them from a URL. The constants include a list of URL schemes, a list of symbols, and a list of words. The function 'get\_features' uses regular expressions to search for these patterns in a given URL and returns a list of features: the length of the URL, whether it's a valid URL, the count of symbols, and the count of words.

```
# Constants: XSS Features
URL = '(http://)|(https://)'
SYMBOLS = "[<,\\"/>
WORDS = "(alert)|(script)|(%3c)|(%3e)|(%20)|(onerror)|(onload)|" \
        "(eval)|(src=)|(prompt)|(onclick)|(document)|(window)|" \
        "(iframe)|(location)|(img)|(div)|(break)|(.js)"

# Extract Features
def get_features(url):
    is_url = 1 if re.search(URL, url, re.IGNORECASE) else 0
    xss_symbol_count = len(re.findall(SYMBOLS, url, re.IGNORECASE))
    xss_word_count = len(re.findall(WORDS, url, re.IGNORECASE))
    return [len(url), is_url, xss_symbol_count, xss_word_count]
```

Figure 3.3: Extract XSS features process

### 3.3 Summary

This chapter is the initial stage in the investigation’s practical component. It details the numerous datasets used, their sizes, and how they are put together, as well as how the data is acquired in comparison to their sources. It also includes a description of the component determination cycle and how they are handled in the testing, with each trademark explained in detail alongside the hijacking process with which it is linked. The mechanism for extracting highlights from payloads is also investigated. It also makes sense of the relationship between individual highlights and a class, as well as the level of the relationship between them.



## CLASSIFIERS

A variety of classifier types are utilized in the present chapter to develop models that can detect XSS, built using the extricated characteristics which have been depicted in Chapter 2. Decision tree, SVM, KNN, linear regression, Gnb and random forest classifiers are totally developed through features which can be extracted as (0,1) values and later learnt via prepared model, showing the aftereffects of the prepared model at the end.

### 4.1 Train Model

In this chapter, it comes the model fit and train step. To detect attacks, the selected characteristics are used as inputs for training the classifiers. The decision tree classifier uses features with values ranging from 0 to 1, reflecting the fraction of the feature included in the payload. All the classifiers mentioned above are tuned using a variety of parameters in order to attain the best classification results across all of them. The classifiers are trained using supervised learning as a general approach. These ML algorithm are chosen due to the well-known approaches to categorization and consistent results for the Web attacks detection.

Furthermore, relevant parameters for use in developing the model have been discovered. These parameters have been tweaked to produce the best outcome, and the optimization is presented in respect to each classifier separately.

#### 4.1.1 Linear Regression

Opposed to mathematics, statistics and biology, many techniques of machine learning come from other subjects, limited to the infancy of it. And linear regression is no exception. It's a method that comes from statistics.

Feature contraction or feature selection in linear regression refers to the selection of a subset of features to be included in the model from the available features in order to lower the model's dimension. On the other hand, it refers to reducing the size of the coefficient estimations, which can be zero. It's worth noting that if the coefficient is reduced to zero,

the related variable is removed from the model. As a result, this circumstance might be thought as feature selection.

Feature selection or feature contraction is a technique for improving simple linear regression. There are two basic reasons why changes may be required:

1. Precision of prediction: linear regression estimates have a low bias and a high variance or over-fitting. To reduce variance and improve model stability, the model's complexity (the number of parameters to be estimated) is lowered, but at the cost of introducing additional deviations. If the best site for the overall error can be identified, the accuracy of the model's predictions can be improved by reducing the variance in the error produced by the error that is easiest to detect.
2. Interpretability of the model: it is challenging to get a handle on each one of the connections among the factors since there are an excessive number of prescient factors. At times, few factors can be distinguished that have the best effect, consequently forfeiting a few factors to upgrade the interpretability of the model.

Starting with simple linear regression, it will model  $y$  target to  $p$  predictions or linear features  $X$  groups:

$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j \quad (4.1)$$

In the model, there is  $p + 2$  parameters which must be calculated from the training data:

1. The effect of the corresponding feature on target prediction is shown by the  $p$  characteristics of  $p$ 's beta coefficient.
2. An intercept parameter, shown by the letter  $\beta$  above, is a forecast in which all  $X$  is equal to zero. It is not necessary to include it in the model, and in some circumstances it should be eliminated, as seen below. However, it usually provides the model additional flexibility.
3. A variance parameter of the Gauss error term.

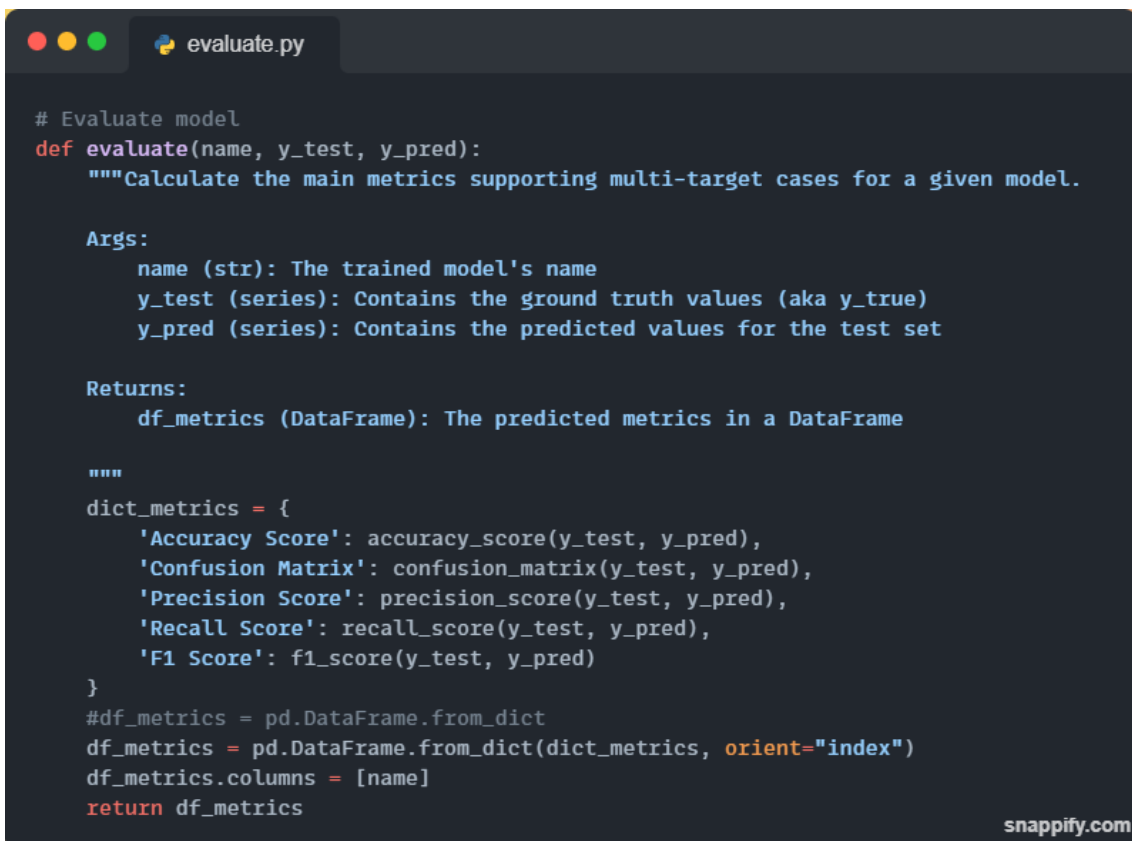
Finally, the table 4.1 is a result of the linear regression model's evaluation.

The experimental results are based on the expressions explained in Chapter 2, Section 2.4.3, which in practice means using functions already available in Python's package - sklearn, as shown in Figure 4.1.

By putting everything together like this, the model can be easily created as shown in the code in Figure 4.2.

Table 4.1: Linear Regression Evaluation

Linear Regression Results	
Accuracy Score	98.3087%
Precision Score	96.0613%
Recall Score	87.8135%
F1 Score	91.7524%
Confusion Matrix	[[53931, 234],[792, 5707]]



```

# Evaluate model
def evaluate(name, y_test, y_pred):
    """Calculate the main metrics supporting multi-target cases for a given model.

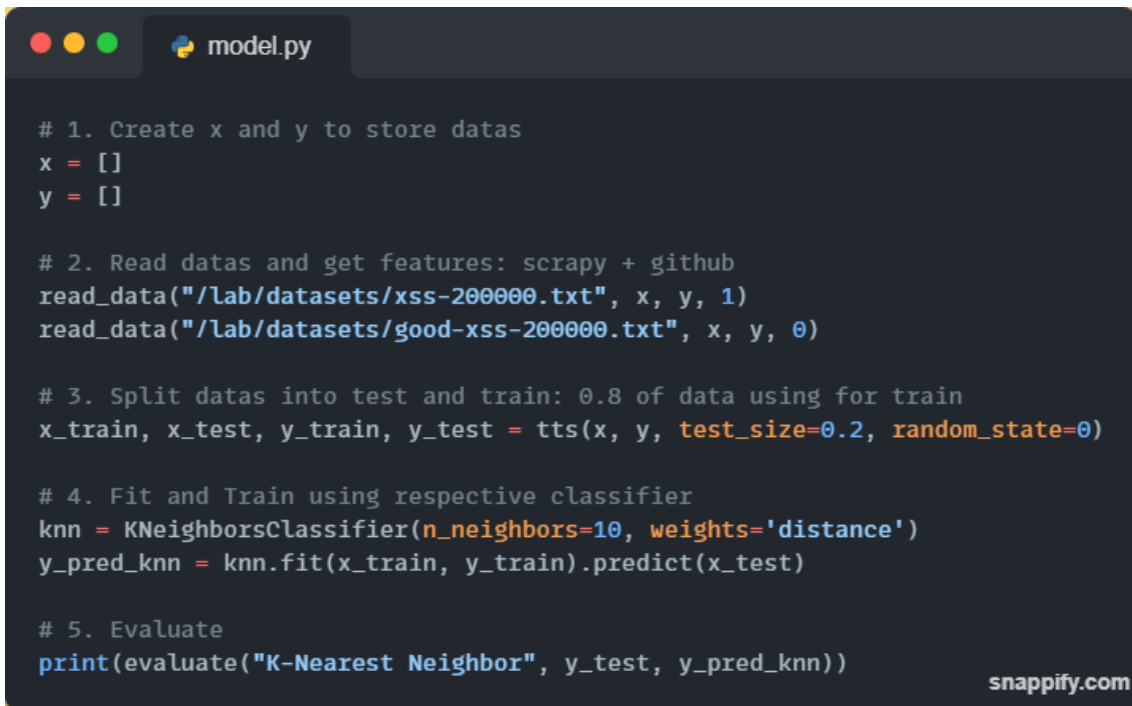
    Args:
        name (str): The trained model's name
        y_test (series): Contains the ground truth values (aka y_true)
        y_pred (series): Contains the predicted values for the test set

    Returns:
        df_metrics (DataFrame): The predicted metrics in a DataFrame

    """
    dict_metrics = {
        'Accuracy Score': accuracy_score(y_test, y_pred),
        'Confusion Matrix': confusion_matrix(y_test, y_pred),
        'Precision Score': precision_score(y_test, y_pred),
        'Recall Score': recall_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred)
    }
    #df_metrics = pd.DataFrame.from_dict
    df_metrics = pd.DataFrame.from_dict(dict_metrics, orient="index")
    df_metrics.columns = [name]
    return df_metrics
  
```

snappify.com

Figure 4.1: XSS evaluation process



```
# 1. Create x and y to store datas
x = []
y = []

# 2. Read datas and get features: scrapy + github
read_data("/lab/datasets/xss-200000.txt", x, y, 1)
read_data("/lab/datasets/good-xss-200000.txt", x, y, 0)

# 3. Split datas into test and train: 0.8 of data using for train
x_train, x_test, y_train, y_test = tts(x, y, test_size=0.2, random_state=0)

# 4. Fit and Train using respective classifier
knn = KNeighborsClassifier(n_neighbors=10, weights='distance')
y_pred_knn = knn.fit(x_train, y_train).predict(x_test)

# 5. Evaluate
print(evaluate("K-Nearest Neighbor", y_test, y_pred_knn))
```

Figure 4.2: Create model process

### 4.1.2 Decision Tree

The decision tree classifier model has in base supervised machine learning method. This method has a tree structure, also called top to bottom architecture. In this structure the tree is composed by:

1. **Root Node:** The root node of the classification, has a high degree of relevance.
2. **Branches:** representing feature choices, the intermediate decision making process, where each node is comparable to a partition of the data set. When there are more features than nodes, the data is divided into smaller chunks.
3. **Leaves:** representing the class (labels) or targets, and the final decision is made at the leaf node.

Each node holds a set of samples that are separated into sub-nodes based on the feature test results, with the root node containing the entire set of samples. A decision test sequence corresponds to the path from the root node to each leaf node. So, once the decision tree is built, all we have to do now is to judge the samples from the root node layer by layer, till we reach the leaf node. In the decision-making process, we can also rank features in the order of relevance from top to bottom, which explains why the tree model contains a feature parameter. The function seeks to return the most appropriate classification for

an instance by starting at the root of the tree and going through the branches (decisions) until it reaches a leaf, from which the class (label) is obtained.

The selection process and feature segmentation, more specifically, how to build and prune a decent decision tree, are the most challenging aspects of this technique. There are three primary steps that can be completed in the described algorithm:

1. **Feature Selection:** the detection of the XSS attacks is presented in the section 3.2.
2. **Build Process:** calculate information gain or other indicators to find the best feature. The decision tree is built recursively from the root node, with the locally optimal features selected continually to split the training set into subsets that can be categorized roughly correctly.
3. **Pruning process:** pre-pruning and post-pruning are two typical pruning techniques.

The input data is classified as normal or dangerous payload using a decision tree classifier. Because it requires examination of all possible choice outcomes and traces every path to a conclusion, this algorithm was employed to classify the scripts. A decision tree that employs all its features if there are no constraints risks overfitting because it will exhaust all its features throughout the generation stage until the stop condition. The number of leaf nodes is at its highest point at this point, and the more leaf nodes there are, the more probable there will be over-fitting and a lack of generalization ability, necessitating pruning to overcome it.

Furthermore, the decision tree technique makes classifications quickly and works well with extremely non-linear data. As a result, the decision tree classifier is a good option for these types of classifications. This type of classifier is also useful for the first in the series of cascading classifiers used to identify XSS vulnerabilities on Web applications. The table 4.2 shows the results of the trained decision tree model, and all the evaluation methods will be explained later in 2.4.3 section.

Table 4.2: Decision Tree Evaluation

Decision Tree Results	
Accuracy Score	99.4939%
Precision Score	97.8812%
Recall Score	97.3842%
F1 Score	97.6321%
Confusion Matrix	[[54028, 137],[170, 6329]]

### 4.1.3 Support Vector Machine

The support vector machine classifier model is another supervised learning approach that is beneficial for detecting susceptible code. The classifier learns from samples, each of which has a set of attributes associated to class names (labels) in some way. The presence of these traits is then used by the classifier to predict the classes of new data. The purpose of SVM is to devise a mathematically efficient method for separating important hyperplanes in a high-dimensional feature space. Support vector machines can be utilized in both classification and regression supervised learning. SVM is based on Vapnik's statistical learning theory [0], which focuses on pattern classification problems and includes a method in which the feature-space is divided into two subspaces using a two-way linear function. The hyperplane defined by this function divides the multi-dimensional space into two areas, one representing one class and the other representing non-membership of that class. To map the feature space into a new features space in such way that the class-dividing hyperplane may be described in a linear form, non-linear functions are also utilized. In this study, a linear kernel and a Gaussian Radial Basis Function (RBF) kernel were used [0]. The polynomial kernel was also examined, but the results were not as good as the other two, therefore it was not considered. Both considered kernels are used to categorize web application attack. These kernels are used to distinguish between payloads, whether they contained malicious or benign scripts.

The equation 4.2 describes the expression for the linear model, where the  $x_1 x_n$  are the  $n$  characteristics for the model input and  $\theta_0 \theta_n$  are the parameters of the  $n + 1$  linear model. The express of the SVM model is obtained by substituting  $f_i$  for  $x_i$  in the linear model, as illustrated in the equation 4.3, where  $f_i$  is the kernel function of the  $x_i$ , i.e. the non-linear polynomial term of  $x_i$ , for example  $f_1 = x_1 * x_2$ . SVM and linear regression are quite similar as shown. Because of the non-linear polynomial factor, SVM may fit non-linearity very well, but it is influenced by noise.

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \quad (4.2)$$

$$h_{\theta}(x) = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \dots + \theta_n f_n \quad (4.3)$$

SVM is chosen because it is a common supervised learning method that can learn from examples to predict new data, and the model will be trained using the XSS dataset. Furthermore, the purpose of SVM is to orient the hyperplane as far away from the nearest member of both classes as possible in order to find a hyperplane that divides the two categories and has the highest margin of both classes. This can be used with the XSS dataset because the margins between the two classes are meant to be as wide as possible in order to achieve high classification accuracy. As a result of the preceding, SVM classifiers with a linear kernel and RBF kernel are developed, and the evaluation results are showed in table 4.3.

Table 4.3: Support Vector Machine Evaluation

SVM with Linear Kernel Results	
Accuracy Score	98.3087%
Precision Score	96.0613%
Recall Score	87.8135%
F1 Score	91.7524%
Confusion Matrix	[[53931, 234],[792, 5707]]
SVM with RBF Kernel Results	
Accuracy Score	98.2642%
Precision Score	98.8343%
Recall Score	84.7977%
F1 Score	91.2795%
Confusion Matrix	[[54100, 65],[988, 5511]]

#### 4.1.4 K-Nearest Neighbors

KNN is a classification method that sorts new data into categories based on the previous encountered instances which are closest in terms of feature space. This strategy is also applicable to both classification and regression. By evaluating the distance between new inputs and training instances, KNN tries to analyze the similarity between fresh input and examples of the training data. New data is classified using KNN by the most common class discovered among its closest neighbors in space. In order to identify XSS payloads, the approach has been used to classify input data as malicious or benign [0]. To summarize, the algorithm's markable phases are as follows:

1. Each sample point in the training sample and the test sample is measured and the distance between them is determined.
2. Sort all the distance values that have been determined.
3. Choose a sample of the first K minimum distances.
4. To get the final classification categories, vote on the k label samples.

The distance between two instance points in feature space represents their degree of similarity. Distance metrics such as Euclidean distance 4.4, Manhattan distance 4.6, Minkowski distance 4.5, and others are routinely used, and this work solely uses the default Minkowski distance to train the model. The distance between two or more points in definition implies Euclidean distance in Euclidean space, and the N features of the sample

data are represented by the value on axis  $x_1$   $x_n$ . The Manhattan distance is calculated by adding the projected distances of lines formed at fixed right angles to Euclidean space. Minkowski distance is a collection of distance definitions that is a generalization of many distance measuring formulas. The European and Manhattan lengths are both variants of the Minkowski distance.

$$d(A, B) = \sqrt{\sum_{i=1}^n (x_{ia} - x_{ib})^2} \quad (4.4)$$

$$d(a, b) = \sqrt[p]{\sum_{i=1}^n (|x_{ia} - x_{ib}|)^p} \quad (4.5)$$

$$d(a, b) = \sum_{i=1}^n (|x_{ia} - x_{ib}|) \quad (4.6)$$

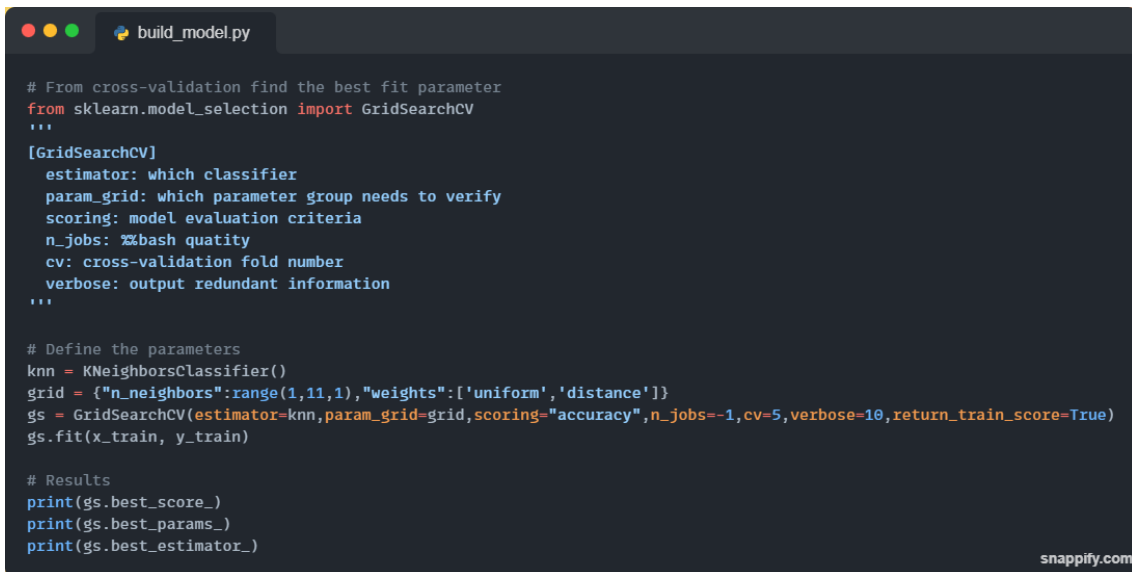
The reason for using KNN is that it does a test to see how close the new instances are to the training data before storing a large amount of categorized data. In this study, the model will identify instances that are closest to the training space as harmful or benign. KNN uses the distance between an unknown instance and the nearest training instance to categorize unknown instances, with classification based on the majority vote of neighbors.

The number of neighbors evaluated for each classification can be modified to fine-tune the KNN classifier. As  $k$  declines, the model becomes more sophisticated, and the filter becomes more prone to over-fitting due to noise in the training data. The model becomes simpler as the value of  $k$  grows. Because the KNN may contain distant and heterogeneous data points, the nearest neighbor classifier may misclassify the test sample. If  $k$  is too large in the application,  $k$  is typically chosen as a smaller value, and cross-validation (explained in the section 2.4.3.1 of the chapter 2) is performed to determine the best  $k$  value. In the current model,  $k = 10$  is found to be the best  $k$ . The findings are shown in table 4.4 and the experimental results in the figure 4.3.

Table 4.4: K-Nearest Neighbors Evaluation

K-Nearest Neighbors Results	
Accuracy Score	99.4873%
Precision Score	97.2511%
Recall Score	97.9843%
F1 Score	97.6163%
Confusion Matrix	[[53985, 180],[131, 6368]]





```
# From cross-validation find the best fit parameter
from sklearn.model_selection import GridSearchCV
'''
[GridSearchCV]
  estimator: which classifier
  param_grid: which parameter group needs to verify
  scoring: model evaluation criteria
  n_jobs: %%bash quaitity
  cv: cross-validation fold number
  verbose: output redundant information
'''

# Define the parameters
knn = KNeighborsClassifier()
grid = {"n_neighbors":range(1,11,1),"weights":["uniform','distance']}
gs = GridSearchCV(estimator=knn,param_grid=grid,scoring="accuracy",n_jobs=-1,cv=5,verbose=10,return_train_score=True)
gs.fit(x_train, y_train)

# Results
print(gs.best_score_)
print(gs.best_params_)
print(gs.best_estimator_)
```

Figure 4.3: Cross-validation process

### 4.1.5 Random Forest

Random forest is another classification approach that incorporates a number of tree-based predictors. Each tree is constructed using randomly chosen variables. All the trees in the forest have the same distribution. More trees lead to more accurate results in a random forest classifier. Random forests are characterized as “Methods that embody the level of integrated learning technology” since they are simple, quick to build, having low computing overhead, and, even more surprising, demonstrate excellent results in classification and regression. The random forest algorithm can be broken down into the following steps:

1. N samples from the data set are chosen as a training set using the bootstrap method.
2. To construct a decision tree, use sampling to obtain a sample set. Each node’s generation includes the following steps:
  - D characteristics are chosen at random on a regular basis.
  - To divide the sample set and determine the best feature, D features are utilized.
3. Steps 1 and 2 are repeated k times, where k is the number of random decision trees in the forest.
4. To obtain the forecast test sample, the random forest, and the decision to anticipate the results of the voting method.

All the above steps are clearly shown in the figure 4.4. The random forest is chosen as the model for categorizing payloads as malicious or benign because of its flexibility in accommodating binary, category, and numerical information. The XSS dataset has a large

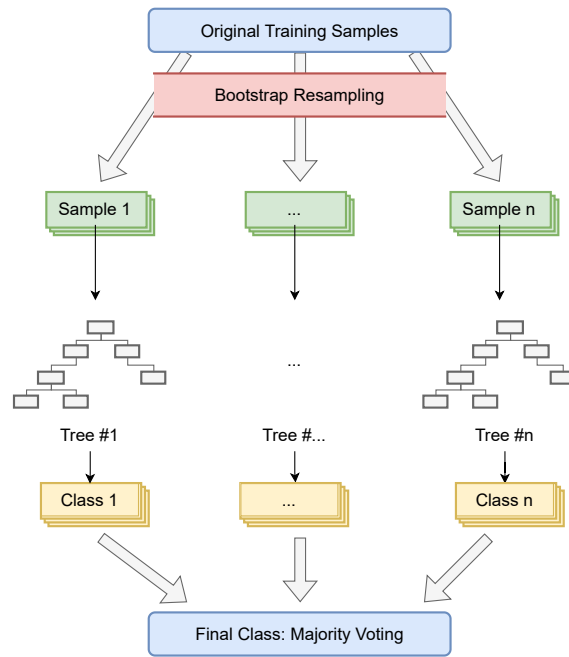


Figure 4.4: Random Forest Process

number of attributes, and while the random forest can only operate with a subset of them at a time, it can manage hundreds. Outliers are handled similarly by the random forest, which effectively ignores them. Random forest's purpose is to reduce overall error rates. The random forest is chosen for above reasons, with the goal of having a high accuracy rate.

In order to build a random forest classifier, two factors must be set: the number of trees and the algorithm. By setting the number of trees parameter to an initial value, computing the misclassification created by a test run, incrementing the value, and testing again, the number of trees parameter is tuned. When determining the value of this parameter, the best way to lowest rate of misclassification achieved is considered, as well as the time spent on creating the model.

Table 4.5: Random Forest Evaluation

Random Forest Results	
Accuracy Score	99.1758%
Precision Score	98.0458%
Recall Score	94.1837%
F1 Score	96.0760%
Confusion Matrix	[[54043, 122],[378, 6121]]

#### 4.1.6 Naive Bayes

Bayes' theorem describes the likelihood of an event depending on prior knowledge of the event in probability theory and statistics. The following is a representation of Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (4.7)$$

The meanings of each probability event are similar. The edge probability of independent occurrences A and B is P(A) or P(B). P(A|B) is the likelihood of B occurring under the conditional probability of A, while P(B|A) is the likelihood of A occurring under the conditional probability of B. According to the full probability formula depending on the nature of the denominator, P(B) can be split into:

$$P(B) = \sum_{i=1}^n P(A_i)P(B|A_i) \quad (4.8)$$

Bayesian inference, a special type of statistical inference in which Bayes theorem can be used to update the probability of the assumptions as the amount of information rises, is one of the numerous uses of Bayes' theorem. Bayesian inference is commonly referred to as "Bayesian probability" in decision theory since it is closely related to subjective probability. Bayes inference is based on prior probability and a probability model to export the outcome as a "likelihood function", which can then be used to determine the posterior probability based on this inference.

The Naive Bayes classifier in machine learning is a simple probability classifier based on the Bayes theorem, where naive refers to the assumption that each feature in the model is largely independent and that correlation between features is ignored. Spam detection with the Naive Bayes classifier, which employs text attributes to identify spam, is the most well-known one among the applications. Naive Bayes classifiers e-mails by using a token to detect the relationship between spam and non-spam, and then computing the probability using Bayes' theorem. As a result, when compared to spam detection, XSS detection is similar, while actually it is an effective method for detecting XSS problems.

In the scikit-learn API [0], there is five classes of naive bayesian classification algorithm:

- **Gaussian NB:** The sample features are primarily continuous values, hence naive bayesian prior to gaussian distribution is employed.
- **Multinomial NB:** Prior to the distribution of polynomial naïve bayes classifier, it was mostly used to classify discrete features based on the number of occurrences of value.
- **Bernoulli NB:** It is likewise used for discrete feature classification prior to the standard bayesian Bernoulli distribution. However, unlike Multinomial NB, it is for a binary or boolean attributes.

- **Categorical NB:** For categorical features, naive bayes is utilized, which is used for discrete indicators that are logically distributed.
- **Complement NB:** For imbalanced datasets, naive bayes is utilized to rectify several “hypotheses” in the normal Multinomial NB classifier.

The Multinomial NB fits the best for XSS detections in this study, and table 4.6 shows the results of these detections in the same way. To gain a better understanding of Multinomial NB, it can be assumed that the feature’s prior probability is a normal distribution, as shown in the following formula:

$$P(X_j = x_j | Y = C_k) = \frac{1}{\sqrt{2\pi\omega_k^2}} \exp\left(-\frac{(x_j - \mu_k)^2}{2\omega_k^2}\right) \quad (4.9)$$

$\mu_k$  and  $\omega_k^2$  are the values to be estimated from the training set, and  $C_k$  is the  $k_{th}$  category of  $Y$ . According to these training sets, Gaussian NB will get  $\mu_k$  and  $\omega_k^2$ . In category  $C_k$ ,  $\mu_k$  is the average of the all  $X_j$ , and the  $\omega_k^2$  is the variance of all  $X_j$ .

Table 4.6: Naive Bayes Multinomial Evaluation

Naive Bayes Multinomial Results	
Accuracy Score	96.8136%
Precision Score	83.9530%
Recall Score	86.8595%
F1 Score	85.3815%
Confusion Matrix	[[53086, 1079],[854, 5645]]

#### 4.1.7 Execution Time

The MacBook Pro 2017 with 3.3GHz dual-core Intel Core i5 and 16GB of RAM memory is the device employed in this study. The execution times of each of the classifiers mentioned above are shown in the table 4.7 in this way. As a result, the decision tree is frequently the quickest classifier, according to some. This classifier is the best choice for this thesis because it is also the most accurate.

The table demonstrates that the SVM classifiers, particularly with the linear kernel one, typically execute at the slowest speed. Next, in descending order respectively of execution time, decision trees outperform KNN, random forests, linear regression, and naive bayes.

Table 4.7: Simple Classifiers Execution Time

Seq.	SVM-linear	SVM-RBF	LR	RF	KNN	DT	NB
1	1m21s	1m	467ms	1.76s	4.94s	407ms	422ms
2	1m18s	1m1s	416ms	1.78s	4.86s	365ms	401ms
3	1m14s	1m	414ms	1.89s	4.85s	367ms	408ms
4	1m17s	1m	438ms	1.8s	5.26s	417ms	407ms
5	1m17s	1m	401ms	1.81s	4.96s	396ms	376ms
6	1m15s	1m	437ms	1.88s	4.98s	381ms	392ms
7	1m17s	1m	479ms	1.88s	4.99s	379ms	393ms
8	1m16s	1m1s	448ms	1.98s	4.95s	433ms	406ms
9	1m18s	1m1s	445ms	1.89s	4.84s	402ms	395ms
10	1m17s	1m5s	431ms	1.83s	4.92s	368ms	451ms
11	1m15s	1m11s	443ms	1.8s	4.81s	388ms	429ms
12	1m17s	1m4s	432ms	2.01s	5.1s	405ms	405ms
13	1m16s	1m2s	442ms	1.91s	5.07s	386ms	384ms
14	1m16s	1m13s	481ms	1.79s	5.29s	407ms	396ms
15	1m8s	1m3s	467ms	1.95s	5.01s	375ms	382ms
16	1m13s	1m5s	461ms	1.91s	4.82s	391ms	418ms
17	1m25s	1m4s	456ms	1.83s	4.25s	402ms	388ms
18	1m26s	1m1s	414ms	1.9s	4.95s	398ms	397ms
19	1m20s	1m1s	445ms	1.96s	5.1s	423ms	411ms
20	1m22s	1m	439ms	1.97s	4.86s	405ms	411ms
Average	1m19s	1m3s	438ms	1.88s	4.84s	395ms	404ms

## 4.2 Discussion

This section is the basis of the thesis, and it examines classifiers in connection to detecting XSS attacks against web applications, which is the focus of the thesis. When looking at the findings of the various classifiers, it can be seen that they all performed well in terms of detecting harmful payloads. Several criteria must be considered while choosing the optimal classifier, the most significant of which is accuracy and precision, as these measurements define the classifier's usefulness. With a 98.04% precision and 99.17% accuracy, the Random Forest classifier produced the best results of all the classifiers. Then, the decision tree and the KNN are also a good classifier for the XSS detections, which are the core focus for the next section.

The anomalies returned by all the classifiers are found to be limited in terms that only a small number of factors seem to have caused them. First, some short scripts which are classified as benign are found to contain a redirection using an IP address. In addition, some long scripts are defect because they did not contain enough features. Some cases are incorrectly categorized due to insufficient characteristics in the testing dataset. Some suggestions for overcoming these false positive classifications include the addition of new features, such as more ASCII codes. Besides, the testing dataset ought to be analyzed to ensure there are no erroneous marks.

## ENSEMBLE ALGORITHMS

The performances of several types of classifiers are demonstrated in the previous chapter, as well as how to determine whether the payload contains XSS attacks. The present chapter will concentrate on the employment of the combination of classifiers to improve the detection of those threats. It will also test with additional parameters in order to improve the main goal's performance and accuracy. Since the random forest, KNN, and decision tree are the best performers, the objective will be the integration of those three.

### 5.1 Introduction to ensemble algorithms

General machine learning has been extensively used to detect Web attacks in the related work that has already been done. Those models are typically created by training on massive amounts of data. It may efficiently model, recognize, and identify data with particular patterns or qualities on the basis of this. Although the deep learning model's performance is generally excellent. There may still be a few flaws, such as inaccurate modeling of attack types for this data for a limited number of samples.

When detecting XSS assaults, for instance, the payload might have some of the crucial characteristics given in table 3.2 and 3.3. If there aren't enough of them, the attack can't be categorized as an XSS attack, leading to a misclassification.

In most cases, gathering data is much simpler, and training the model will produce excellent detections. However, gathering data of this kind will be somewhat challenging for some uncommon or specialized varieties.

Due to a paucity of training data for this subdivision type and insufficient training of the model, the model will have a higher false negative rate when attempting to detect this sort of event. When faced with this circumstance, the typical course of action is to repeatedly sample this portion of a small number of sample types or to utilize the sampling algorithm to extend the sample, but the new. Sample overlap and low quality are common issues. The model frequently trains these samples repeatedly.

A generic meta approach to machine learning called ensemble learning [0] combines the predictions from various models to get greater predictive performance.

There are three techniques that rule the world of ensemble learning, despite the fact that you can create an apparently limitless number of ensembles to solve your predictive modeling problem. So that instead of being algorithms in and of itself, each is a subject of study that has given rise to numerous more specific techniques.

The three primary classes of ensemble learning techniques are bagging, stacking, and boosting, and it's critical to grasp each technique in-depth and take it into account in the predictive modeling project.

- **Bagging:** entails averaging the predictions from many decision trees that have been fitted to various samples of the same dataset.
- **Stacking:** when numerous distinct model types are fitted to the same data, it is used to learn how to combine the predictions most effectively.
- **Boosting:** a weighted average of the predictions is produced by it, which requires adding ensemble members in a sequential manner that corrects the predictions provided by earlier models.

## 5.2 The framework

A novel model is introduced based on the technology of fusing various machine learning algorithms, and in this study the voting classifier (bagging) is applied. Using a majority vote or the average projected probability, the voting classifier combines conceptually distinct machine learning classifiers to predict class labels. In order to counteract the flaws of a group of models that perform equally well, such a classifier can be helpful. There are two types of voting:

1. **Hard/Majority:** the class label that represents the majority (mode) of the class labels predicted by each individual classifier is the one used as the expected class label for a given sample.
2. **Soft:** each classifier can have a specific weight applied to it using the weights option. The predicted class probabilities for each classifier are gathered, multiplied by the classifier weight, and averaged where weights are given. The class label with the highest average probability is then used to determine the final class label.

Let's use a straightforward example to demonstrate the soft mode. Assume we have three classifiers and a classification issue with three classes. We give each classifier the same weight:  $w_1=1$ ,  $w_2=1$ , and  $w_3$ . The sample's weighted average probabilities would then be determined using table 5.1, which would produce the anticipated label B because it has the highest average probability.

Figure 5.1 depicts the detection model architecture. The initial stages of collection and characteristic extraction are the same as those shown for the straightforward classifier in



Table 5.1: Soft voting classifier probabilities

Classifier	A	B	C
Classifier A	$w1*0.1$	$w1*0.6$	$w1*0.2$
Classifier B	$w2*0.5$	$w2*0.4$	$w2*0.3$
Classifier C	$w3*0.2$	$w3*0.2$	$w3*0.5$
Average	0.27	0.4	0.34

the preceding chapter. The next model's primary distinction is that it has undergone two separate training sessions using various classifiers. The success rate of the model will rise if the second classifier can filter the XSS attack features while the first cannot.

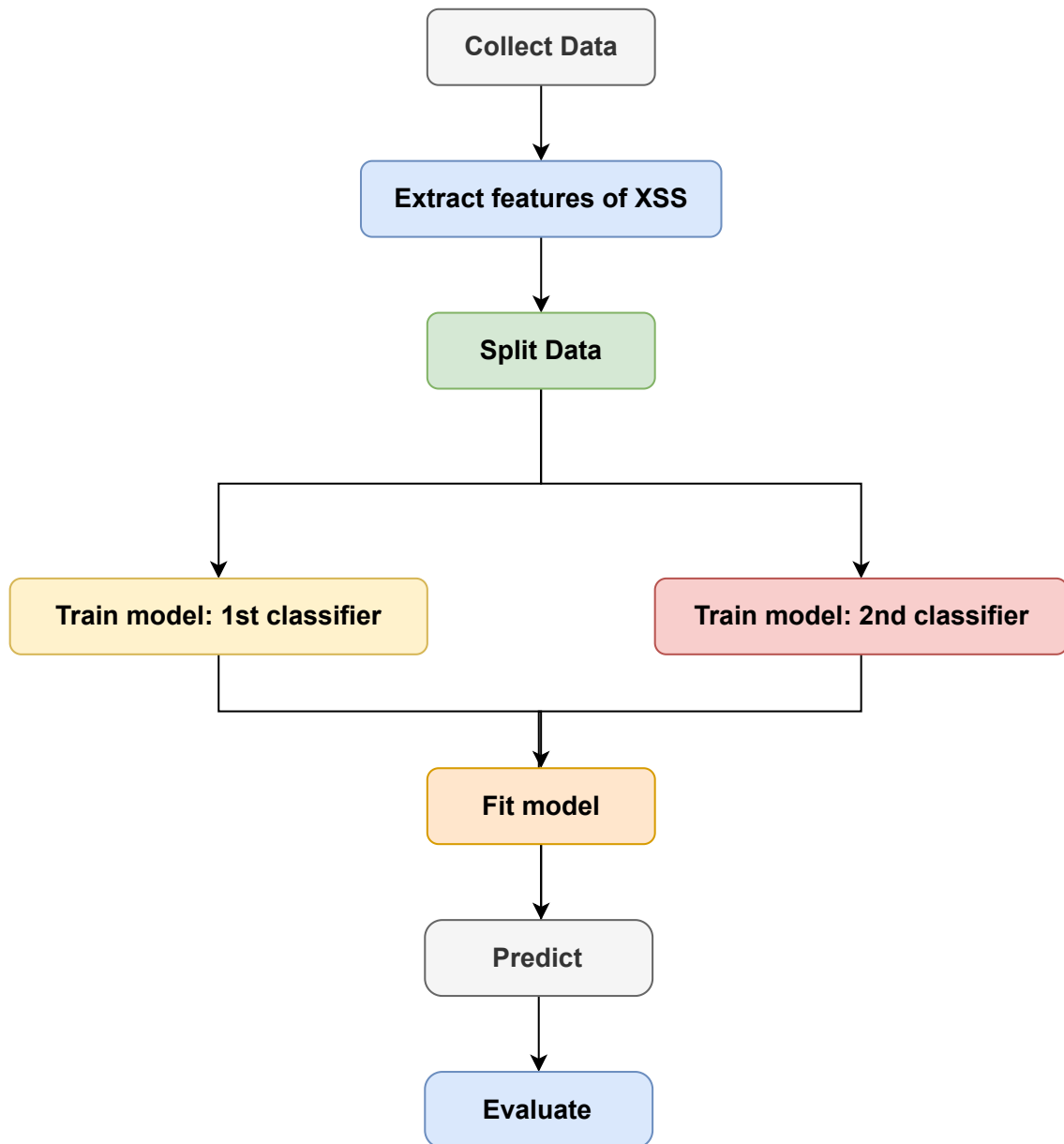


Figure 5.1: Ensemble Framework

### 5.3 Random Forest with Decision Tree

To enhance the form of the code in the current XSS vulnerability, the code replaces the original module with some function combinations that perform the same purpose, or inserts some sophisticated conditional branches, loop statements, and other structures. In its logical form, these methods make the code complex and diversified. The more complicated the script logic is, the more complex the semantics it contains, and the challenge of detecting malicious code increases dramatically. Script code is similar to language text, and the code has a meaningful link. In addition, the following information can be used to deduce the meaning of some attack information.

Combinations of the best classifiers were suggested for training a classifier to recognize XSS kinds with the same data collected as referred in the chapter 3. The most up-to-date strategy is to rely on the features provided in section 3.2 as the categorization basis. They're more likely to be found in XSS scripts, making them useful for detecting such attacks.

The combinations of the classifiers proposed between the best accuracy classifier are mentioned at the end of the chapter above. The first combination to focus on is random forest and decision tree, and the outcomes are displayed in table 5.2. The dataset payloads are first classified using a random forest classifier and then use a decision tree in this system's pipeline procedure. According to the hard voting method, the anticipated class label for a given sample serves as the class label that represents the majority.

Table 5.2: Evaluation: Random Forest with Decision Tree

RF + DT Results	
Accuracy Score	99.1741%
Precision Score	98.4648%
Recall Score	93.7529%
F1 Score	96.0511%
Confusion Matrix	[[54070, 95],[406, 6093]]

The datasets with the characteristics retrieved are utilized for the first phase, which is the same process as described in sub-section 3.2 chapter 3. The decision tree classification, as described in sub-section 4.1.2 of the chapter 4, is then applied to the random forest classified model (in sub-section 4.1.5), resulting in the ensemble classifier, which is the final model and can identify XSS attacks. Using both classifiers has improved the trained model's precision when compared to using either classifier alone (97.88% and 98.04%), as seen in the results table above. Other test results, however, fall short of expectations. It can be accounted by the fact that the results of the straightforward classifiers are already almost 100%.

## 5.4 K-Nearest Neighbors with Random Forest

The following system is using both KNN and random forest for the same proposal because these two also received high performance ratings in the chapter 4 evaluation. The suggested system is distinctive since its list of pioneering techniques includes one of the first approaches to employ the stacked ensemble methodology to identify XSS attacks on the server side. The system's objectives are to increase the precision of classifiers that identify XSS vulnerabilities and filter inputs by only sending scripts to the next step as described in the figure 5.1.

The contextual uses the results from the base level to create its own final classification, which employs XSS features to distinguish between malicious and benign scripts. The base level gives this classification based on XSS features (of either benign or malicious) as explained in the section 3.2 of chapter 3. Remember that all targets have a value of 0 or 1, with 0 denoting that a feature does not appear in a script and 1 denoting the contrary, that a feature does appear in a script.

Table 5.3: Evaluation: K-Nearest Neighbors with Random forest

KNN + RF Results	
Accuracy Score	99.1708%
Precision Score	98.4330%
Recall Score	93.7529%
F1 Score	96.0359%
Confusion Matrix	[[54068, 97],[406, 6093]]

The decision tree classifier has been replaced with a KNN one for the identical method described in the section above. Therefore, it is clear from comparing the results showed in the table 5.3 that they are fairly comparable. Despite not being exactly equal, they produce comparable outcomes. However, it shows that the performance is unchanged. Consequently, it is necessary to keep merging the additional classifiers.

## 5.5 K-Nearest Neighbors with Decision Tree

The following specific models are utilized for evaluation after classifiers are improved during training. By training them on the complete training dataset, the final trained classifiers utilized on the test data are created. Again, the decision tree is used in place of the random forest and the KNN is maintained. The method of classification is the same as that explained in the sections 4.1.2 and ?? in the chapter 4. Given that the majority voting method is used as in the previous ensemble classifications, the system does not care about the order of the classifiers.

On the testing dataset, which has not been utilized in the tuning or training, these classifiers are then assessed. This test's objectives include choosing the top classifier simulating an actual attack. The datasets, which includes both scripts and links, was used to test the final, trained decision tree and KNN classifier for categorizing the payload type. The testing dataset, which consists of 200000 instances with labels for text and 200000 instances with labels for script, is prepared to test the ensemble classifier. This approach differs from the approach in that the particular classifier is used to detecting XSS datasets rather than the ensembling classifier.

Table 5.4: Evaluation: K-Nearest Neighbors with Decision Tree

KNN + DT Results	
Accuracy Score	99.4939%
Precision Score	97.8960%
Recall Score	97.3688%
F1 Score	97.6317%
Confusion Matrix	[[54029, 136],[171, 6328]]

The accuracy has gone up from 99.2 percent to 99.5 percent, while the other scores have all increased to about 98 percent, as seen in table 5.4. It indicates that the developed XSS detecting system will function significantly better. As a result, it can go on to the real-time application, which will be discussed in the following chapter.

## 5.6 Execution Time

Now, the average run time of the model's code is tested using the same system and same processors from the MacBook Pro 2017. Noting that there are two classifiers to pass instead of the simple classifiers, the time required to complete the model won't rise by a factor of two. Although it's fast is quick through the table 5.5, the user will experience these models as being immediate, which will enhance their interaction with the Web application that will be constructed in a later chapter.

The random forest and decision tree is the best classifier combination in terms of time performance, while the KNN with random forest must be the worst, as the table demonstrates. KNN and decision trees will be used for the construction of Web applications because they have a medium performance level, also due to these classifiers have the highest levels of accuracy and precision.

Table 5.5: Ensemble Classifier Execution Time

Seq.	RF+DT	KNN+RF	KNN+DT
1	2.73s	6.67s	5.14s
2	2.55s	7.28s	5.13s
3	2.56s	7.53s	5.13s
4	2.44s	6.9s	5.92s
5	2.44s	7.47s	5.3s
6	2.72s	7.36s	5.56s
7	2.65s	7.21s	5.21s
8	2.53s	6.75s	5.6s
9	2.66s	7.31s	5.49s
10	2.61s	7.22s	5.38s
11	2.58s	7.36s	5.36s
12	2.42s	7.16s	5.59s
13	2.84s	7.44s	5.47s
14	2.44s	7.56s	5.27s
15	2.63s	6.78s	5.57s
16	2.8s	6.65s	5.86s
17	2.62s	6.87s	5.7s
18	2.76s	7.31s	5.75s
19	2.98s	7.34s	5.81s
20	2.57s	6.79s	5.53s
Average	2.63s	7.15s	5.21s

## 5.7 Discussion

An introduction to the ensemble technique has been given in this chapter. It contains a summary of the key techniques employed with this strategy as well as information on cascading classifiers. Additionally, it examines the earlier research on the use of ensemble approaches and the fusion of classifiers with intrusion detection systems and provides a thorough explanation of the suggested approach for spotting XSS assaults, along with an explanation of the stages that are utilized to distinguish between malicious and good scripts. The evaluations of the classifiers in both phases utilizing cross validation and holdout approaches are provided with their findings. The findings of the system-wide test have also been presented. The methods that may be employed to get the best performance are shown along with the execution time performance of the overall system.

This chapter has shown that the bagging ensemble technique has produced accuracy values that are strikingly similar to those of single classifiers. The bagging strategy is more expensive in terms of time than utilizing single classifiers. Therefore, using single classifiers rather than the bagging strategy can be recommended. The takeaway from this chapter is that one of the pioneering attempts in XSS attack detection is the bagging ensemble technique.

In addition, at the end of this chapter, the entire process of achieving results over time is recorded and evaluated. This is useful for real-time program analysis, which is described in the next chapter. The next chapter will provide an application design for the browser, taking the Chrome extension as an example, to test the design system in real-time. The ensemble classifier with KNN and decision tree performs better in terms of accuracy and precision, but it is also important to consider the time factor when think about real-time application performance. So, it is necessary to verify if this ensemble classifier can be used in the final model.





## REAL TIME APPLICATION

The suggested system from the preceding chapter will be put to the test in this chapter in real-time. The goal is for the browser to warn the user to exercise caution when it encounters a URL that contains an XSS attack. As a result, XSS vulnerabilities are closed, protecting users from harms.

This proposal suggests developing a Chrome extension, a browser application that may identify user behavior and take appropriate action to prevent disrespect. Beginning with the introduction of web applications, moving on to the framework and language to be used, and ending with the features from the backend layer services to the frontend requests, are the steps in developing a respect system.

### 6.1 Introduction to Chrome Extension

The biggest advantage of Chrome is that it supports a wide range of reliable and useful add-ons. In order to properly recommend some of the best plug-ins, it is necessary to analyze how plugins are generated and how they differ from the way consumers frequently create web pages for browsers. A saying goes that since decent Chrome doesn't have add-ons, it doesn't smell as wonderful. An add-in is a customized browsing interface for brief software applications. Users can alter how Chrome behaves and functions in a number of different ways. The following features are offered by various plug-ins: productivity tools, more knowledge about the web, and entertainment (games).

Through the creation of the Chrome extension, it has the following meanings: improving browser functionality, putting their own bespoke plug-in function in place, comprehending current plug-ins, and maximizing their functionality. However, the purpose of this study is to once more conduct a real-time test of the trained model once more to thwart XSS attacks.

Web technologies like HTML, JS, and CSS are used to build plug-ins. They interact with the Chrome browser while operating in a separate sandbox execution environment. By modifying browser behavior and accessing Web material through the use of APIs, plug-ins enable us to expand and improve the capabilities of the browser using APIs. Plug-ins work

with developer APIs and end-user user interfaces:

1. The user interface should be expanded so that users may manage their extensions in a consistent manner.
2. The API's expansion allows the browser to access functions such as activating tags and changing network requests.

In order to develop a plug-in, we must first create a list of the resources that go into it, such as the JS and HTML files, graphics, and so on. To load these “unzips” into Chrome for testing and development, utilize the developer mode plugin. If we are satisfied with our plug-ins, we can bundle and distribute them to other users via the internet shop. If you create a plugin and intend to make it available on the Chrome Web App Store, you must abide by the following rules:

1. Plug-ins must have a single purpose, which is clearly stated and simple to comprehend. As long as each component and function helps achieve the overall objective, a single plug-in can have a variety of parts and functions.
2. The user interface should be purposefully simple and uncluttered. Simple icons to the creation of a new window with a form are all possible.

There are no formal project structure requirements for Chrome plugins, such as directories like `src`, `public`, `components`, etc. As a result, if we examine the source code of numerous plugins, it can discover that each plugin has a unique project structure, and even the file names within the project are distinctive. However, it must contain a manifest file in the executable file of root directory. It is a json language file, which serves as the applet's app and is the plug-in configuration file that describes the various information about the plug-in. A variety of helpful APIs for modifying browser behavior are provided by the Chrome plugin, including but not limited to:

- Bookmark management
- Download control
- Window management
- Label management
- Control of the network requests and monitoring all occurrences
- Specific raw menu
- Ideal communication system

Additionally, Chrome plug-ins can work with DLL (dynamic link libraries) written in C++ to implement some more underlying functions than just front-end technology.

## 6.2 The Framework and Language

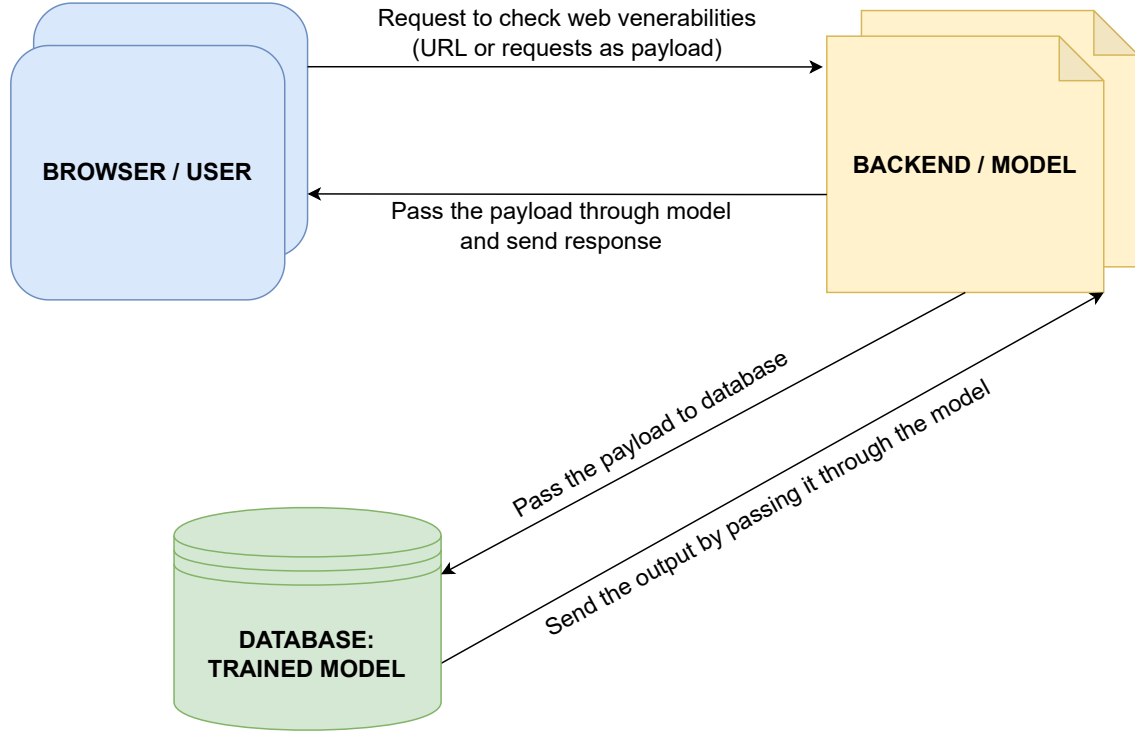


Figure 6.1: Chrome Extension System Framework

As was mentioned in the section above, the languages to utilize are once more JS for logical behaviors, HTML for the structure of web content, and CSS for styling and making content more aesthetically pleasing. Figure 6.1 depicts the basic system structure, which is comparable to any other typical Web application in that it consists of a frontend user interface and backend services. In this instance, the model that was created and trained in the previous chapter is represented by the backend services. This model is now used as the service layer that will supply the response to the request from the front-end app. And in this instance, it relates to the XSS vulnerabilities on the browser page being used by the current user. If there is a risk of XSS on the current web page, the frontend application, which corresponds to the user view, will make a signal or dialog for the user.

For this study, in the frontend user interface side, the progressive framework Vue.js is used. A JS framework for creating user interfaces is called Vue. It provides a declarative and component-based programming approach that aids in the speedy development of user interfaces, whether they are basic or complicated, and built on top of industry-standard HTML, CSS, and JS.

Python is used for the backend services because it is also used for model development. It contains a straightforward backend micro framework called Flask and a basic language for quick learning. “Micro” doesn’t mean you have to put the entire web application in

a Python file, nor does it mean that Flask is functionally deficient. “Micro” in the micro framework means that Flask is designed to keep core functions simple and easy to extend. Flask doesn’t make many decisions for you, such as which database to use. The decisions that Flask makes for you (such as which template engine to use) are easy to replace.

## 6.3 Backend Services

Simply by reading the quickstart section of the Flask official manual [0], it is easy to create a basic website using Flask. The level of HTML, CSS, and JS, which are outside the purview of Flask, will naturally determine how well the site looks. As was previously mentioned, backend services employ the Flask micro Web framework. The following are some features of Flask and the rationale for using it:

- It is free, versatile, expandable, and has a large range of third-party libraries, when combined with your preferred wheels and the most well-known and potent Python libraries.
- It is easy to get started and create a website even if you have no expertise with web development.
- It is ideally suited for the creation of tiny websites and web service APIs.
- Large websites can be developed without pressure, but they must have a code architecture, and their cost will depend on the developers’ skill and expertise.

Programming in Flask is therefore flexible and user-friendly. The trained model data are saved in files with the extension “pkl” (database) so that they can be used as determined in the development of the previous chapter. The goal of the backend layer service is to connect the front Web application request and database response in some way. It receives the request sent by the front application, passes the payload to the database side (which also constitutes a portion of the backend), and then sends the response to the front web application after testing to see whether there is an XSS vulnerability. An example Flask application that forms a site utilizing only a couple of lines of direct code is as follows:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

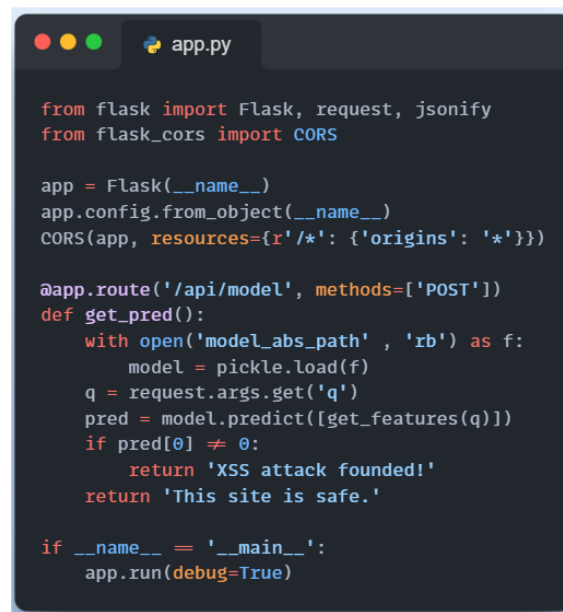
if __name__ == '__main__':
    app.run()
```

Information or error return, page rendering, static file loading, URL generation, session or cookie management, request and response processing, etc. are some of the website's most fundamental operations. It only requires creating an instance of Flask (similar to the code before) and calling the “run()” method to generate a straightforward website. The functionality to be used while accessing the URL is implemented by the following functions. The route (‘/’) specifies the website's URL path. The functionality of any Python library may be converted into a website or a simple Web API using the logic of the preceding uncomplicated code. Although using third-party modules connected to RESTful or REST API is preferred, this logic can be used to build a RESTful or REST API from scratch.

Even for a small site, the best way to do this is to put different logic into different files, which are organized in a way that suits your tastes. As the functionality of the site increases, so does the number of lines of code, which can be difficult to manage when all the code is in one file.

Flask can connect directly to the database to retrieve the data without any issues, but doing so will make the application more tightly tied with the database, which is bad for expansion and will increase the workload required to maintain the code logic. By using the database abstraction layer, the logic of Flask interacting with the database will be made simpler and transferred to the database abstraction layer, improving the clarity of the business logic and narrowing the focus of development. Because Flask is a micro framework, it lacks a database abstraction layer of its own. However, there are many database abstraction layers accessible in Python, and there are many possibilities for Flask, so it can select the most comfortable database abstraction layer for development. Since this study's theme does not primarily center on this application, the database will only consist of one file in order to facilitate real-time testing. Therefore, the file is posted immediately on the Flask app.

The development of the backend service is shown in Figure 6.2. It proves that the Flask framework makes the work easier. All left is implementing the appropriate API whose URL ends in “\api\model” and making a response to the request of the front-end program. In this specific case, the program takes the arguments from the request sent by the client and inserts them as input into the machine learning model implemented in the previous chapters, and when it has an output or result, the service responds to the client with whether XSS vulnerabilities exist.

The image shows a code editor window titled 'app.py' with a dark background and light-colored text. The code is a Python Flask application that implements a simple XSS detection service. It imports Flask, request, jsonify from flask, and CORS from flask\_cors. It creates a Flask app, configures it, and sets up CORS for all origins. A POST endpoint '/api/model' is defined with a function 'get\_pred()' that loads a model from a file, predicts based on the request's 'q' parameter, and returns 'XSS attack founded!' if the prediction is non-zero, otherwise 'This site is safe.'. The app is run in debug mode if executed as the main script.

```
from flask import Flask, request, jsonify
from flask_cors import CORS

app = Flask(__name__)
app.config.from_object(__name__)
CORS(app, resources={r'/*': {'origins': '*'}})

@app.route('/api/model', methods=['POST'])
def get_pred():
    with open('model_abs_path', 'rb') as f:
        model = pickle.load(f)
    q = request.args.get('q')
    pred = model.predict([get_features(q)])
    if pred[0] != 0:
        return 'XSS attack founded!'
    return 'This site is safe.'

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

Figure 6.2: Backend Service implementation

## 6.4 Frontend Application

The Vue framework with JS as a basis is chosen as the technology to construct web applications for the frontend side. The Vue framework and ecosystem are renowned for providing the majority of the typical functionality needed in frontend development. The Web, though, is incredibly diverse, and the things we create there may differ greatly in scale and form. In light of this, Vue is made to be adaptable and gradually adopted. Vue has a variety of uses that you can choose from according to your use case:

- Static HTML improvement without a build step.
- Web components can be embedded on any page.
- Single-Page Application (SPA)
- Full-stack / Server-Side Rendering (SSR)
- Jamstack / Static Site Generation (SSG)
- Targeting the terminal, WebGL, mobile, and even desktop

Despite the flexibility, all these use cases share the fundamental understanding of how Vue operates. Even if you are only starting out right now, the knowledge you pick up along the way will be helpful as you advance and take on more challenging objectives in the future. If you are an experienced user, you may choose the best strategy to use Vue based on the issues you are trying to resolve while maintaining your productivity. Vue is known

as “the progressive framework” because it is a framework that can change and develop with the change as necessary.

It is advised to start a development of chrome extensions using a Node Package Manager (NPM) package [0] to simplify the design of the project’s schematic. The default package manager for Node.js, the runtime for JS, is NPM. A Command Line Interface (CLI) tool for publishing and downloading packages and an online repository that houses JS packages make up the two primary components of NPM. Thus, adopting the NPM package makes using the Chrome extension development CLI easier. And using this CLI, all that is required is to select the options that are necessary for the extension.

During the development of the Chrome extension, the directory folder structure is shown in figure 6.3. In particular, the actual directory structure will be more complex, also shown below, in figure 6.4.

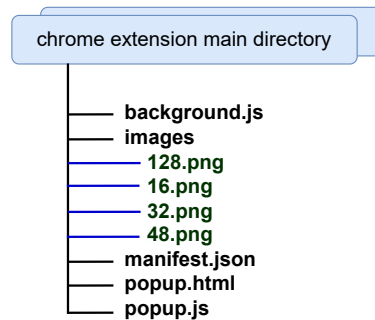


Figure 6.3: Chrome Extension Directory Structure

The manifest file is a configuration file that lists the files required for the application to run. The manifest file also contains the extension’s fundamental details in JSON format, such as its name, version, needed permissions, etc. The following Chrome extension will use JSON objects, a language and platform-independent data standard, to send structured static data between modules. It is transmitted as a straightforward string and received as an object by JS when it gets JSON data, which may contain hierarchical information. Due to its low data redundancy, readability, and ease of parsing, this format can be used for data sharing not only between Chrome extensions but also across platforms and other programs. A sample manifest is shown below in JSON format:

```
{
  "name": "chrome-plugin",
  "version": "1.0",
  "description": "Build an Extension!",
  "manifest_version": 3
}
```

First, the above string surrounded in curly brackets “{}” is a JSON object and serves as an illustration of how we interpret the JSON format. In essence, a JSON object is

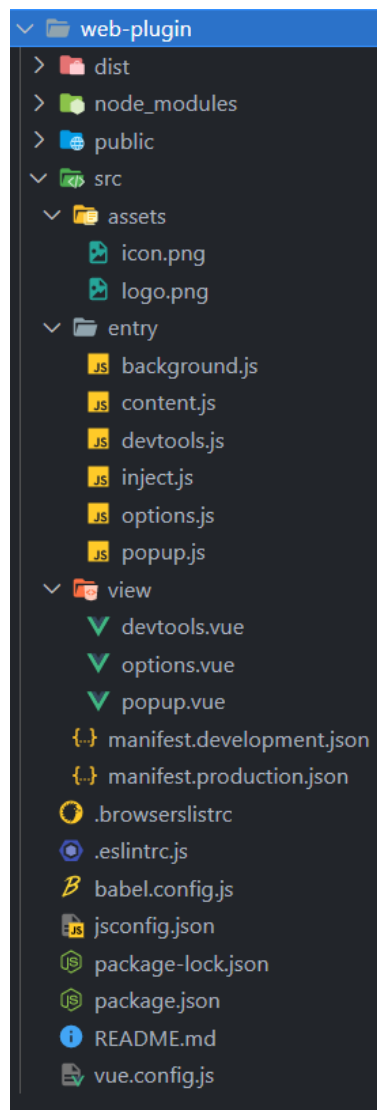


Figure 6.4: Chrome Extension Directory Structure Complex

a structure with fields that are separated by commas and have a space after the final field. Every field has the format “key: value”, where each “key” denotes an attribute of the structure. An attribute must have a concrete value in order to be considered concrete. To write the manifest in accordance with Chrome’s requirements, it has been discovered to employ a set of “Templates” that Chrome has established for JSON. More details on a real case are shown in figure 6.5.

When a Chrome extension is published and installed, both the Chrome store and Chrome will verify the manifest file. With JSON format, it extracts the essential data from it, and determine whether the writing is legal. For instance, if the content field in popup.html, which has a built-in JS script and is empty, an error will be returned when attempting to load your extension in Chrome. The information is not only necessary for users to see, but also the browser. The latter informs Chrome of the entrance point to your





```
{
  "manifest_version": 3,
  "name": "XSS-det",
  "description": "Detect XSS attacks",
  "version": "1.0.0",
  "background": {
    "service_worker": "/background.js"
  },
  "action": {
    "default_popup": "popup.html"
  },
  "content_scripts": [
    {
      "matches": [
        "<all_urls>"
      ],
      "js": [
        "/content.js"
      ]
    }
  ],
  "options_page": "options.html",
  "devtools_page": "devtools.html",
  "permissions": [
    "activeTab",
    "storage",
    "webNavigation",
    "tabs",
    "http://*/*",
    "https://*/*"
  ],
  "icons": {
    "16": "/assets/icon.png",
    "48": "/assets/icon.png",
    "128": "/assets/icon.png"
  }
}
```

Figure 6.5: Chrome Extension Manifest File

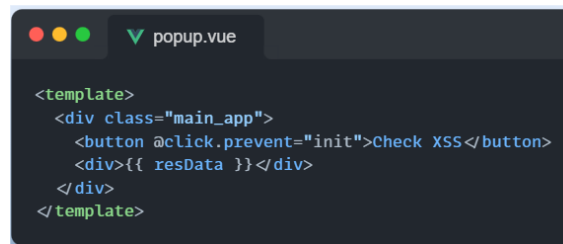
application and other details about how it is functioning.

Name, version, and manifest version are the three properties that must be present in a number of common fields in the manifest file. The extension's name is the calling that has been established for it. It must be a string saved in UTF-8 encoding for the JSON to work correctly when loaded into the browser. Version is an acronym for the extension's version number. However, it should be a string rather than a string of numbers as the version number is typically a string of integers. The version number of the manifest file format is specified by manifest version (defined by Chrome). It should be 2 after Chrome 18 and will migrate from V2 to V3 in November 2020.

The background script is typically used for background debugging of other files. In this work, the background script is not implemented. The app's logos are represented by

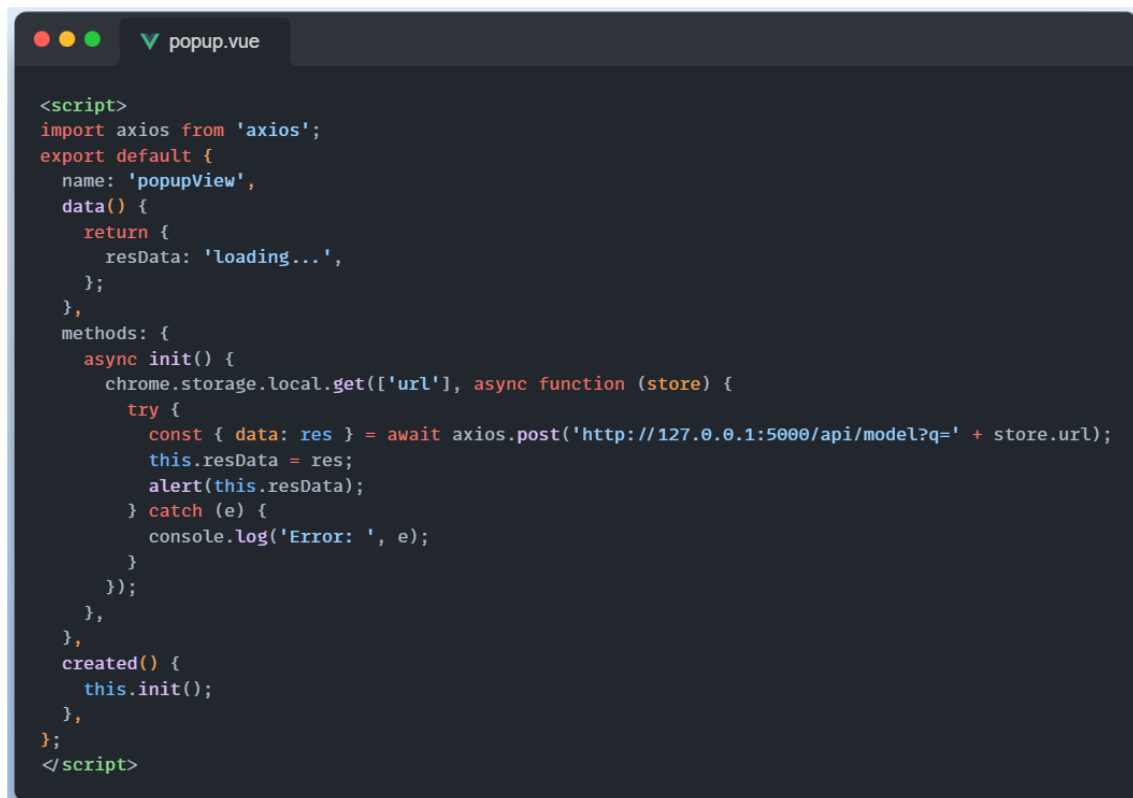
images' directory (it includes various proportions: 128x128, 64x64, etc.).

The file 'popup.vue' also contains the application and the content of the popup. The popup content display behavior for user interaction, and it is represented by the popup.js script file. Therefore, the main file distributed in this work is the popup script. In this particular case, the main work of a request to services using the HTTP protocol is implemented there. The main process is to use the third-party provided package, called "axios" [0], to send a query to the backend. And when it gets a result, it will pop up a dialog box for the user to know the result. Since Vue.js is used in this project, 'popup.vue' can be split into two parts: the first HTML part, shown in figure 6.6, and the second JS part, shown in figure 6.7.



```
<template>
  <div class="main_app">
    <button @click.prevent="init">Check XSS</button>
    <div>{{ resData }}</div>
  </div>
</template>
```

Figure 6.6: Popup HTML code



```
<script>
import axios from 'axios';
export default {
  name: 'popupView',
  data() {
    return {
      resData: 'loading...',
    };
  },
  methods: {
    async init() {
      chrome.storage.local.get(['url'], async function (store) {
        try {
          const { data: res } = await axios.post('http://127.0.0.1:5000/api/model?q=' + store.url);
          this.resData = res;
          alert(this.resData);
        } catch (e) {
          console.log('Error: ', e);
        }
      });
    },
  },
  created() {
    this.init();
  },
};
</script>
```

Figure 6.7: Popup JS code

According to this constitution, the popup script file is in charge of identifying changes to the web page, which is the fundamental logic that underpins how this program functions. The popup script will immediately send request (with the URL information) from the backend to determine whether the website is secure if the user changes the web page. In the absence of a warning dialog, it will state that the active page is safe. Naturally, if the client clicks on the extension icon directly, the same approach is taken.

## 6.5 Discussion

In this chapter, a brief overview of the concepts involved with Chrome extension and, in particular, the web application with the detection of XSS - along with a description of the backend layer service interacting with the designed model (database) and web application with user interface, including sending information through the network and checking results by passing this information through the trained model. The proposed system, which is considered to be the contribution of this thesis, is described in terms of detecting and preventing the XSS attacks through the machine learning - with a detailed explanation of all the factors which it depends on. Emphasis is placed on the sampling system whereby samples are derived from the training dataset and then utilized to find the correct response (class) for each rule in the truth table.

Moreover, the practical application of the developed model using the ensemble methods in machine learning is covered in this chapter. This will enhance user experience and secure users' privacy across the network while also increasing the quality of machine existence. Although it is impossible to provide a precise number based on test findings, it is highly advised that clients install it and use it for their regular network browsing.



## CONCLUSION

### 7.1 Research Summary

Finally, by employing two categories of features (alphanumeric and non-alphanumeric) and representing them as Boolean (false or true), this research has created a machine learning system that detects XSS attacks. Linear Regression, Support Vector Machine with both linear and RBF kernels, K-Nearest Neighbor, Random Forest, Decision Tree and Naive Bayes Gaussian are among the classification techniques utilized in the trials. These classifiers were used to distinguish between harmful and benign user input into a Web application. The efficacy of applying machine learning to defend Web applications from XSS attacks has been shown through experiments carried out during this research. All the examined classifiers have high rates of accuracy and precision, all of which are higher than 96%. In terms of precision, the results of this study can be compared to those of earlier methods. Other studies may also obtain a precision rate of more than 96%, but this study has tried other methods.

The Decision Tree classifier is used in this investigation to reach an accuracy rate of 99.5%, whereas the same classifier was used in a prior study to achieve an accuracy rate approximately of 99%. The classifiers are trained using a dataset designed for this purpose, and malicious and benign scripts are gathered from several trustworthy sources as stated in chapter 3 to construct the dataset.

Additionally, the retrieved features are the primary factor in the classifiers' high accuracy. Since these qualities are chosen based that they are present in the payload. Additionally, a boolean value that represents a feature also helps to boost accuracy. Chapter 3 discusses how to select the features for each sort of assault and how to portray them for training purposes. Moreover, choosing useful traits to recognize XSS threats is crucial for the detection of such attacks. The characteristics are divided into two categories: characteristics common to all types of attacks and characteristics unique to specific types of attacks.

In order to confirm that the features can be combined, multi-class classifiers that could categorize all three types of attack are developed. The development of multi-class classifiers that may divide user input for a Web application into the XSS class and benign categories

has been thoroughly discussed in chapter 5. As described in this chapter more detailed, ensemble approaches and cascading classifiers are utilized to filter the inputs and improve accuracy. The classification results produced by this approach are marginally superior to those by utilizing a single classifier. These classifiers achieved accuracy rates of more than 99% and precision rate more than 98%. This technique is one of the earliest to employ stacking to identify XSS vulnerabilities.

Rules from the decision tree and KNN classifiers that described decision-making are retrieved in order to comprehend the choices made by these “black-box” classifiers. In chapter 6, a potential web application is described in detail. It would execute the model in real time and extract rules from classifiers. The chapter focuses on the framework of the application, which is split into the front and back sides, correspond to the two service sections. In order to bypass the payload gathered on the front side and obtain the classification, the back side attempts to apply the prior ensemble classifier model. Send the outcome to the front end through the network, where it will be shown on the user’s screen.

Processing-time performance, accuracy, and precision are crucial, when choosing a classifier, especially one that will serve as a protective layer for a web service. Therefore, an attempt has been made to relate processing times, accuracy and precision scores to user-friendliness. In these terms, it is determined that the final ensemble classifier performs well enough to be applied in this situation.

## 7.2 Research Discussion

The research aiming to identify XSS attacks against Web applications is reviewed in this section. These results help to choose a classifier or method that can be used as a security layer for Web pages. Understanding the fundamentals of how a hacker can utilize coding to conduct XSS threats, as described in chapter 2, is important to accomplish this. This knowledge satisfies the study’s first aim (G1).

The rates of a classifier’s accuracy, precision, false positive rate, and speed are all acceptable criteria. High accuracies were obtained while using a single classifier to identify XSS vulnerabilities. SVM with a linear kernel completed the testing dataset classification in 1 minute and 19 seconds in average of 20 tries with a 98.31% accuracy, 96.06% precision, and a recall score of 87.81%. With 234 cases labeled as false positives, SVM with a polynomial kernel obtained a 98.26% accuracy and 98.83% precision; it took 1 minute and 3 seconds in average to finish classifying the testing dataset. With an accuracy and precision of 99.49% and 97.25%, respectively, and a false positive rate of 180 instances, the KNN classifier classified the testing dataset in 4.84 seconds. Likewise, the random forest classifier classified the testing dataset in 1.88 seconds with a 99.18% accuracy, 98.05% precision, and 122 false positive occurrences reported. The testing dataset classification was finished in 0.404 seconds by the naive bayes Gaussian classifier, which had a poor accuracy rate of 96.81 percent, a poor precision of 83.95 percent, and returned 1079 false

positive occurrences. The linear regression made 98.31% of accuracy, 96.06% of precision, 234 false positives and takes 0.438 seconds in average to complete. The decision tree takes in accuracy 99.50%, 97.88% in precision, 137 of false positives and wasted 0.395 seconds to finish. Since the decision classifier is the quickest, it was chosen from a security standpoint, which also provided the lowest amount of false positive cases, is the best classifier for detecting XSS attacks, as the data shows in the chapter 4. Therefore, using the decision tree classifier, in this case, as a protective layer for a Web application is justifiable.

Using an ensemble technique with cascade classifiers, the second objective (G2) of this study has completed, which is to find the model to detect XSS vulnerabilities and increase its accuracy. The detection of XSS attacks is highly accurate thanks to this combination approach. The suggested system uses cascading classifications to categorize the inputs into different kind of features. In this stage, the inputs are categorized using the classifiers that have already been discussed, and the results are then used as inputs for the Web application's backend service. With 136 false positive cases returned (in 80000 cases), the proposed system attained a 99.50% accuracy and 97.9% precision. It took 5.21 seconds to complete the classification throughout both phases utilizing KNN and decision tree classifiers. The proposed system, on the other hand, is one of the first to use the stacking ensemble technique to find XSS assaults.

The classifiers that have been generated should be used on the server side as a barrier between the Web application and HTTP requests that are loaded with user inputs. This allows achieving the third objective of this research possible (G3). The classifier receives features that are taken from the request and uses them to determine whether the request is malicious. Whether the request is valid or not, it is always sent to the Web program. It merely serves as a user alert and does not actively block the website. By developing a website that would accept inputs and categorize them as harmful or benign, the classifiers were put to the test. The classifiers developed in this study have produced good results for this site, which is created for a single user.

## 7.3 Future Work

As was said in chapter 4 and 5, the performance results of the classifiers are in line with the literature's use of machine learning to identify XSS attacks against Web applications. Classifiers have been shown to be capable of identifying both old and new XSS. However, classifiers are constrained by the following factors:

- The false positives: these chapters have described how XSS attacks are incorrectly categorized. A false positive occurs when a harmful occurrence is mistakenly labeled as benign. False positives occur when the relevant events are either too brief or lack sufficient characteristics to be classified as malicious, or when the features are not as noteworthy as they were retrieved. Alternatively, an instance may be too long to allow for comparisons to be conducted with less aberrant texts or the attributes

are not very noteworthy. In connection with these characteristics, the inclusion of non-alphanumeric aspects may be the primary cause of the majority of the false positives.

- Processing time performance: KNN and decision tree are the classifiers that produced results with high accuracy and precision, although they required longer than other researches. Both the single classifier and the cascade classifier scenarios showed this.
- Dataset size: since XSS attacks have such brief lives on the Web before being erased, it is difficult to detect them in large numbers. Therefore, there are extremely few sites that provide examples of attacks.

Some final thoughts on this project, in order for the program to run in real time without too much impact on the user, the execution time needs to be reduced. Since we need reliable execution time, we have to abandon the absolute best scoring model and choose a model with lower scores for better real-time execution of the program. Thus, a more mature technological approach could be used, which reduces the real-time execution time of the application and improves the interaction with the user. Therefore, to get suggestions for future work, should consider combining the assessment of precision and accuracy with runtime, which may include other more advanced models.

Finally, this research has gained a wider foundation on which future work can be performed by avoiding focusing on just one type of assault or just one type of machine learning technique. Based on this, the future research projects can be relevant to the followings:

- To increase performance, it may be useful to investigate deep learning as well as other types of unsupervised machine learning algorithms that can be used to identify XSS attacks on Web programs. To further develop the order productivity of classifiers that detect XSS attacks and lessen the quantity of dishonestly hopeful expectations, the focal point of the examination of these calculations will be on involving similar elements as in this review and similar strategy as the highlights are addressed. Parameters should be tuned, and attributes that separate various attacks should be investigated using a way other than employing typical or comparable features.
- Adjust to similar practical standards, remembering for the examination subject every one of the weaknesses in web applications. Injection of operating system commands, CRLF (Carriage Return and Line Feed) injection, code injection, and access points into web applications are all examples of intentional vulnerabilities. The study field will be widened by eliminating other helpful features that can be utilized to differentiate between various attack types based on their similarities and differences in order to develop a classifier that can classify any assault that utilizes holes in online programs.



- The suggested method for removing the rules from the black box classifiers can be created using values other than 0 and 1. The extraction of a function whose conditions employ the logical keywords can be facilitated by restricting the data to a particular range during the development process. A change in the sampling technique from using nearby samples to using distant ones could have additional potential results, which can be achieved by keeping track of samples, comparing them to practice data to eliminate out samples that are exact replicas of practice data, and using new samples can all help achieve this. In order to extract the rules from classifiers, processing-time performance is being watched carefully.



## BIBLIOGRAPHY

- [0] 1.11. *Ensemble methods — scikit-learn 0.22.1 documentation*. Scikit-learn.org, 2012. URL: <https://scikit-learn.org/stable/modules/ensemble.html> (cit. on p. 51).
- [0] 1.4. *Support Vector Machines — scikit-learn 0.20.3 documentation*. Scikit-learn.org, 2018. URL: <https://scikit-learn.org/stable/modules/svm.html> (cit. on p. 42).
- [0] *API Reference — scikit-learn 0.23.1 documentation*. scikit-learn.org. URL: [https://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive\\_bayes](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes) (cit. on p. 47).
- [0] axios. *axios/axios*. GitHub, May 2019. URL: <https://github.com/axios/axios> (cit. on p. 69).
- [0] R. Binux. *pyspider*. GitHub, Jan. 2023. URL: <https://github.com/binux/pyspider> (visited on 01/30/2023) (cit. on p. 27).
- [0] J. Choi et al. “Efficient malicious code detection using n-gram analysis and SVM”. In: *2011 14th International Conference on Network-Based Information Systems*. IEEE, 2011, pp. 618–621 (cit. on p. 22).
- [0] scikit-learn developers. *sklearn.neighbors.KNeighborsClassifier — scikit-learn 0.22.1 documentation*. Scikit-learn.org, 2019. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html> (cit. on p. 43).
- [0] M. K. Gupta, M. C. Govil, and G. Singh. “Predicting Cross-Site Scripting (XSS) security vulnerabilities in web applications”. In: *2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. 2015, pp. 162–167. DOI: 10.1109/JCSSE.2015.7219789 (cit. on pp. 22, 23).
- [0] S. Gupta and B. Gupta. “Automated discovery of JavaScript code injection attacks in PHP web applications”. In: *Procedia Computer Science* 78 (2016), pp. 82–87 (cit. on p. 22).

- [0] W. G. Halfond, S. R. Choudhary, and A. Orso. “Improving penetration testing through static and dynamic analysis”. In: *Software Testing, Verification and Reliability* 21.3 (2011), pp. 195–214 (cit. on pp. 21, 22).
- [0] G. Kaur et al. “Detecting Blind Cross-Site Scripting Attacks Using Machine Learning”. In: *Proceedings of the 2018 International Conference on Signal Processing and Machine Learning*. 2018, pp. 22–25 (cit. on p. 24).
- [0] E. Kirda et al. “Noxes”. In: *Proceedings of the 2006 ACM symposium on Applied computing - SAC '06* (2006). DOI: 10.1145/1141277.1141357 (cit. on pp. 21, 22).
- [0] K. Munonye and M. Péter. “Machine learning approach to vulnerability detection in OAuth 2.0 authentication and authorization flow”. In: *International Journal of Information Security* (2021), pp. 1–15 (cit. on p. 25).
- [0] OWASP. *OWASP Top 10:2021*. owasp.org, 2021. URL: <https://owasp.org/Top10/> (cit. on p. 1).
- [0] *Quickstart — Flask Documentation (2.1.x)*. flask.palletsprojects.com. URL: <https://flask.palletsprojects.com/en/2.1.x/quickstart/> (visited on 07/28/2022) (cit. on p. 63).
- [0] SciKit-Learn. *3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.21.3 documentation*. Scikit-learn.org, 2009. URL: [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html) (cit. on p. 18).
- [0] L. K. Shar, L. C. Briand, and H. B. K. Tan. “Web application vulnerability prediction using hybrid program analysis and machine learning”. In: *IEEE Transactions on dependable and secure computing* 12.6 (2014), pp. 688–707 (cit. on pp. 21, 22).
- [0] L. K. Shar, L. C. Briand, and H. B. K. Tan. “Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning”. In: *IEEE Transactions on Dependable and Secure Computing* 12.6 (2015), pp. 688–707. DOI: 10.1109/TDSC.2014.2373377 (cit. on p. 23).
- [0] L. K. Shar, H. B. K. Tan, and L. C. Briand. “Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis”. In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. 2013, pp. 642–651 (cit. on p. 23).
- [0] S. Sharma and N. S. Yadav. “Ensemble-based Machine Learning Techniques for Attack Detection”. In: *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE. 2021, pp. 1–6 (cit. on p. 25).
- [0] S. Sharma, P. Zavorsky, and S. Butakov. “Machine Learning based Intrusion Detection System for Web-Based Attacks”. In: *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE. 2020, pp. 227–230 (cit. on pp. 24, 25).

- [0] I. Tariq et al. “Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning”. In: *Expert Systems with Applications* 168 (2021), p. 114386 (cit. on p. 25).
- [0] V. N. Vapnik. *The nature of statistical learning theory*. Springer, 1998. URL: <https://dl.acm.org/citation.cfm?id=211359> (visited on 03/26/2019) (cit. on p. 42).
- [0] P. Vogt et al. “Cross site scripting prevention with dynamic data tainting and static analysis.” In: *NDSS*. Vol. 2007. 2007, p. 12 (cit. on pp. 21, 22).
- [0] *vue-cli-plugin-chrome-extension-cli*. npm. URL: <https://www.npmjs.com/package/vue-cli-plugin-chrome-extension-cli> (visited on 07/27/2022) (cit. on p. 66).
- [0] R. Wang et al. “Improved N-gram approach for cross-site scripting detection in Online Social Network”. In: *2015 Science and Information Conference (SAI)*. IEEE. 2015, pp. 1206–1212 (cit. on pp. 23, 24).
- [0] R. Wang et al. “Machine learning based cross-site scripting detection in online social network”. In: *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*. IEEE. 2014, pp. 823–826 (cit. on pp. 23, 24).
- [0] *XSSed / Cross Site Scripting (XSS) attacks information and archive*. xssed.com. URL: <http://xssed.com> (visited on 04/28/2022) (cit. on p. 27).
- [0] Y. Zhou and P. Wang. “An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence”. In: *Computers & Security* 82 (2019), pp. 261–269 (cit. on p. 24).





