

# **Research on Penetration Testing Based on Machine Learning**

## **Abstract**

In recent years, the number of network vulnerabilities found in web applications has shown a sharp upward trend. These vulnerabilities pose a certain threat to the security of data. Once exploited by attackers, they may not only cause varying degrees of damage to the confidentiality, integrity and availability of data, but also utilize the vulnerable website as their tools to expand their malicious activities. In order to ensure the security of data involved in web applications, running penetration testing could be particularly important. However, traditional penetration testing relies heavily on the experience and technology of security personnel, which often means high time cost and economic cost, and it is usually difficult for small enterprises and individual web holders to come up with sufficient budget to ensure the safety of their websites. In this context, even if their websites are hacked and turned into malicious, it could be difficult for them to find problems. So, there is an urgent need for a faster method to carry on penetration testing at low cost as much as possible, to ensure the security of web application. Applying machine learning to penetration testing could be a direction which is worthy of exploration. This paper studies the penetration testing of web applications. The main achievement of this project is to develop a method based on machine learning to automatically predict malicious vulnerable web pages of target website and launch exploit automatically. The software is mainly divided into three parts. The first part uses crawlers to crawl and save the web page data of the target web application, and then formats the data and submits it to the second part. The second part uses the trained machine learning model to analyze the target web page, evaluate whether there is any malicious code in the web page, and submit it to the third part if there are exploitable targets. The third part is the vulnerability exploitation module, which calls exploiting tools to scan and attack web pages that are predicted to be risky in order to exploit vulnerabilities.

## **Acknowledgements**

Firstly, I would like to thank my supervisor Amir Javed for giving me this opportunity to do projects that I am interested in and patiently help me when I encounter difficulties. Without his support, it is difficult for me to successfully complete this project. Secondly, I want to thank my friend Lujing Ge. She has given me a lot of help in my study and life. I cherish the time I spent with her. Finally, I would like to thank my family, who have always been my strong backing and given me strong support in all aspects.

## Table of Contents

<b>List of figures.....</b>	<b>7</b>
<b>List of tables.....</b>	<b>8</b>
<b>Chapter 1 .....</b>	<b>9</b>
<b>Introduction .....</b>	<b>9</b>
1.1 Preface .....	9
1.2 Research problem.....	9
1.2 Research motivation .....	10
1.3 Project aims and objectives .....	11
1.4 Intended audience .....	12
1.5 Dissertation outline .....	12
<b>Chapter 2 .....</b>	<b>13</b>
<b>Background and literature review .....</b>	<b>13</b>
2.1 Introduction of penetration testing.....	13
2.2 Introduction of machine learning.....	14
2.2.1 Supervised learning .....	15
2.2.2 Unsupervised learning .....	17
2.2.3 Reinforcement learning .....	17
2.3 Machine learning algorithm used in this project .....	17
2.3.1 Random forests .....	18
2.3.2 Naive Bayes .....	19
2.3.3 K-nearest neighbor .....	21
2.3.4 Support vector machine.....	21
2.4 Machine learning evaluation methods.....	22
2.4.1 Confusion matrix .....	23
2.4.2 Performance indicator .....	23
2.5 Python Libraries.....	24
2.6 OWASP ZAP .....	25
2.7 Related works.....	25
<b>Chapter 3 .....</b>	<b>28</b>
<b>Requirement specification and system design.....</b>	<b>28</b>
3.1 Requirement specification .....	28
3.1.1 Functional requirement .....	28
3.1.2 None- functional requirements .....	29
3.2 System design.....	29

<b>Chapter 4 .....</b>	<b>32</b>
<b>Implementation.....</b>	<b>32</b>
<b>4.1 Classifier Module .....</b>	<b>32</b>
<b>4.1.1 Data collection.....</b>	<b>32</b>
<b>4.1.2 Feature selection .....</b>	<b>35</b>
<b>4.1.3 Data preprocessing .....</b>	<b>35</b>
Data preprocessing for comprehensive features .....	37
Data preprocessing for JavaScript code .....	38
<b>4.1.4 Machine learning models training.....</b>	<b>40</b>
Random forest.....	40
Naïve Bayes .....	41
KNN .....	41
SVM .....	42
<b>4.1.5 Classifier evaluation and comparative analysis .....</b>	<b>43</b>
Confusion matrix (Comprehensive features dataset) .....	43
Confusion matrix (JavaScript code dataset) .....	43
Precision (Comprehensive features dataset) .....	44
Precision (JavaScript code dataset) .....	45
Accuracy (Comprehensive features dataset) .....	45
Accuracy (JavaScript code dataset) .....	46
Recall (Comprehensive features dataset) .....	46
Recall (JavaScript code dataset) .....	47
F1 score (Comprehensive features dataset).....	47
F1 score (JavaScript code dataset) .....	48
<b>4.2 Crawler module .....</b>	<b>49</b>
<b>4.3 Vulnerability scanning module.....</b>	<b>50</b>
<b>Chapter 5 .....</b>	<b>52</b>
<b>System testing and comparative analysis .....</b>	<b>52</b>
<b>5.1 Crawler module test .....</b>	<b>52</b>
<b>5.2 Classifier module test .....</b>	<b>52</b>
<b>5.3 Vulnerability scanning module test.....</b>	<b>54</b>
<b>5.4 Integration test.....</b>	<b>54</b>
<b>5.5 Test result.....</b>	<b>54</b>
<b>5.6 Comparative analysis.....</b>	<b>55</b>
<b>Chapter 6 .....</b>	<b>56</b>
<b>Limitation and future work.....</b>	<b>56</b>
<b>6.1 Limitation .....</b>	<b>56</b>
<b>6.2 Future works .....</b>	<b>56</b>
<b>Chapter 7 .....</b>	<b>58</b>
<b>Conclusion .....</b>	<b>58</b>
<b>Chapter 8 .....</b>	<b>59</b>

<b><i>Reflection .....</i></b>	<b><i>59</i></b>
<b><i>References.....</i></b>	<b><i>61</i></b>

## List of figures

Figure 1: Machine learning process.....	15
Figure 2: Machine learning training.....	16
Figure 3: Decision tree.....	18
Figure 4: Random forest.....	19
Figure 5: Naïve Bayes.....	20
Figure 6: KNN.....	21
Figure 7: SVM.....	22
Figure 8: Program structure .....	29
Figure 9: Program processes.....	30
Figure 10: Code block of data checking.....	36
Figure 11: Result of data checking .....	36
Figure 12: Code block of under sampling .....	36
Figure 13: Code block of data encode .....	37
Figure 14: Code block of get URL depth.....	38
Figure 15: Code block of splicing column .....	38
Figure 16: Code block of get JavaScript code .....	39
Figure 17: Code block of code feature extraction.....	39
Figure 18: Code block of random forest model training.....	40
Figure 19: Code block of naïve Bayes model training .....	41
Figure 20: Code block of 4 folds cross validation .....	42
Figure 21: Result of validation .....	42
Figure 22: Code block of SVM model training.....	42
Figure 23: Confusion matrix (Comprehensive features).....	43
Figure 24: Confusion matrix (JavaScript code).....	43
Figure 25: Precision (Comprehensive features) .....	44
Figure 26: Precision (JavaScript code) .....	45
Figure 27: Accuracy (Comprehensive features) .....	45
Figure 28: Accuracy (JavaScript code) .....	46
Figure 29: Recall (Comprehensive features).....	46
Figure 30: Recall (JavaScript code).....	47
Figure 31: F1 score (Comprehensive features) .....	47
Figure 32: F1 score (JavaScript code) .....	48
Figure 33: Code block of crawler.....	49
Figure 34: Code block of data preprocessing .....	50
Figure 35: Code block of load classifier .....	50
Figure 36: Code block of extract malicious URL.....	50
Figure 37: Code block of ZAP Class.....	51
Figure 38: Code block of vulnerability scanning .....	51
Figure 39: Code block of vulnerability exploiting .....	51

## List of tables

Table 1: Confusion matrix.....	23
Table 2: Features in dataset [45].....	34
Table 3: Test case 1 .....	52
Table 4: Test case 2.....	53
Table 5: Test case 3.....	53
Table 6: Test case 4.....	53
Table 7: Test case 5.....	54
Table 8: Test case 6.....	54
Table 9: Test case 7 .....	54



# Chapter 1

## Introduction

### 1.1 Preface

In today's world, with the rapid development of the Internet, all walks of life have gradually entered the age of information. Especially in recent years, due to the rapid spread of COVID-19 in the world, there are many companies and organizations have begun to migrate their business to online. As a relatively flexible network software, web application is very convenient. Web applications allow their users to access a variety of network services by browser without installing any additional software. Therefore, web application is widely used in different kind of industry. Enterprises and even government agencies can use it to provide services and processing their works. Although the popularity of these web applications has greatly improved work efficiency and provided a lot of convenience for daily life, it is also accompanied by some data security risks. Statistics show that network security problems caused by web application vulnerabilities have become more common in recent years. According to the report released by CDNetworks, the number of attacks on web application have reached approximately 4.2 billion which have boosted over 9 times compared to 2019. It is also worth noting that with the rapidly development of artificial intelligence, there are many attackers are applying machine learning on malicious activities. The report shows that more than 90% web application attacks are from automated scanners [1].

### 1.2 Research problem

Once the web application is attacked, it may cause a series of serious problems. Hackers may insert malicious scripts into web pages through vulnerabilities to secretly carry out destructive activities. Especially for small businesses or personal websites, their web pages are likely to become malicious websites if existing vulnerabilities are exploited by attackers. Common attacks against web applications include DDoS, XSS injection, SQL injection and remote code

execution (RCE). DDoS attacks cause network congestion by generating a large amount of traffic, making the target website inaccessible, which destroys the availability of data. It is reported that New Zealand stock exchange was forced to suspend spot market trading due to DDoS attacks [2]. In 2016, because DNS server provider dyn was attacked by DDoS, many well-known websites such as Amazon, twitter and spotify could not be accessed [3]. XSS attack may have a certain impact on the confidentiality and integrity of data. The attacker tampers with the website through XSS insertion and lures the victim into the changed website, so as to steal cookies and redirect users to malicious websites. British Airways has faced a fine of £ 20m for violating gdpr due to data disclosure caused by XSS vulnerability [4]. EBay's XSS vulnerability has caused a large number of users to be cheated [6]. SQL injection is also a dangerous attack. It can insert specific SQL statements into the page request, so as to access the database without authorization and obtain the information in the database, resulting in serious information disclosure. In 2014, security researchers found SQL injection vulnerability in Tesla website, which will cause user information disclosure [7]. Cisco has been reported that there could be SQL injection vulnerability in Cisco data center manager [8]. If it is exploited, it will allow attackers to gain administrative privileges. The reason for the RCE vulnerability is that the code of the web application uses functions that can execute system commands, and the user's input is not filtered, so the user's input can be executed on the server as a command statement. Through this vulnerability, an attacker can easily obtain the privileges of the server, resulting in more serious damage.

## **1.2 Research motivation**

In order to repair vulnerabilities and find malicious code as soon as possible after the web application get compromised or showing signs of being attacked, to avoid more losses from serious web security incidents caused by malicious exploitation by hackers, it is increasingly necessary for the owners of web applications to conduct penetration testing on their own websites. However, the traditional penetration testing process is often cumbersome and has high technical requirements for testers, it is usually taking a certain time to find potential vulnerabilities as much as possible. For the penetration test of web application,

there are many automatic scanning tools, such as Nmap, Nikto, SQLmap, w3af, etc. [9]. However, these tools often require testers to have some experience to find the location of vulnerabilities and select the appropriate payload to launch attack. Although those automatic scanning tools could be a powerful automatic scanning tool which allows users to find and verify vulnerabilities, for large-scale web applications, its complete scanning function will take a lot of time. Facing so many network threats, how to improve the efficiency of penetration testing could be a long-term problem that worth to explore. As mentioned above, machine learning is used by some hackers to carry out malicious activities, which can be said to be the negative impact of the rapid development of machine learning in recent years. However, machine learning is also widely used to ensure cybersecurity. For example, spam classification, malicious website identification, malicious traffic detection, etc. [10]. However, these applications could be regard as passive security measures, and the research of machine learning in active security detection is just emerging. In order to face the increasingly serious network threat, the combination of penetration testing and machine learning to improve work efficiency and improve the accuracy of vulnerability scanning could be a direction worth to explore.

### **1.3 Project aims and objectives**

The aim of this project is to develop a lightweight Python3 program based on machine learning to detect the web pages which could be malicious, then to scan and exploit potential vulnerabilities in compromised web pages.

The scope of this project is to train several different machine learning models through the training set, which are random forest, naive Bayes, KNN and SVM, then compare the performance of several different machine learning models in predicting malicious websites in the test set and select a model with the best performance to be applied to the automatic penetration test program. The program consists of three parts. The first part is the crawler, which will crawl the content of the target web page and convert it into the certain data format that required by the prediction model. The second part is the classifier based on selected machine learning model, which will predict whether the target web site may have

vulnerabilities. The third part is the exploit module, which will scan the corresponding target web page according to the prediction results and try to exploit. The innovation of this project is that the combination of machine learning and traditional penetration test scanning tools can effectively shorten the scanning time of traditional scanning tools. The dataset used in this project comes from A.K. Singh. The development language is Python, which is widely used in machine learning, and the test environment of the program is Kali Linux.

#### **1.4 Intended audience**

The audience of this project are researchers and relevant practitioners in the field of Cybersecurity, as well as anyone interested in effectively applying machine learning to penetration testing.

#### **1.5 Dissertation outline**

Chapter 1 gives a short description of the project context, main aims, scope and intended audience of the project. Chapter 2 introduces the related technical background of the project from three aspects. Firstly, it briefly introduces the penetration test and its process. Secondly, it provides machine learning and lists some classical machine learning models that can be used for vulnerability web page prediction, as well as some methods to evaluate the performance of models. Thirdly, it lists the main Python libraries used in this project. Finally, it gives some relevant research status. In the Chapter 3, the requirements of the project are explained and analyzed, and the structure and process design of the project are given. The Chapter 4 is the main section of this paper. In this chapter, the implementation process is described in detail according to the main structure of the project. Chapter 5 will give some system test case of this project and comparative analysis. Chapter 6 will summarize the limitation of the project and give some potential work in the future. Chapter 7 will summaries this project and draw conclusion. Chapter 8 will present a reflection on learning result from this project.

## Chapter 2

### Background and literature review

#### 2.1 Introduction of penetration testing

Penetration testing usually refers to a set of activities to ensure the security of the target system by launching ethical attacks. This process includes dynamical analysis of technical vulnerability or weakness of the system. The vulnerability analysis is carried out from the possible injection point of an attacker, and conditionally take the initiative to exploit the vulnerabilities of this location using various payload.

Penetration testing is usually divided into seven stages, including clear requirements, information collection, threat modeling, vulnerability analysis, vulnerability verification, deep attack, written report, etc. [9]. Clarifying the requirements stage means that before the penetration test, the tester shall communicate with the customer to unify the opinions of both parties on the penetration test, determine the objectives of the penetration test, etc. After that, enter the information collection stage [9]. The main work at this stage is to collect some public information of the target, such as the URL of the web app, the network protocol used (HTTP or HTTPS), web page content, etc. After the information is collected, the work can enter the threat modeling stage. It is to say that use the collected information to formulate some possible attack schemes to achieve the purpose of penetration test [9]. The next stage is the vulnerability analyzing. The main work of this stage is to scan the vulnerabilities of the target system with the help of some software tools [9]. After the vulnerability is scanned, the vulnerability verification phase can be carried out, in which some vulnerability exploit tools are usually used [9]. If the vulnerability is verified, the deep attack stage can be carried out. The deep attack stage often means that the security protection measures of the penetration target have been broken. The work of this stage is to show the consequences of the security protection being broken [9]. The last stage is the written report stage. The main goal of this stage is to provide customers with a well

understood document, explain a series of problems found in the penetration test, and provide repair suggestions [9].

## **2.2 Introduction of machine learning**

Machine learning is a kind of technology that explore how to use computers to obtain new knowledge or skills, rebuild the existing knowledge structure and iteratively improve its performance by simulate or realize the learning behavior of human.[11][12] It could be seen as a subject which is interdisciplinary, for example, it includes statistics, probability theory, approximation theory, convex analysis and algorithm complexity theory and so on. As a popular subject in recent years, the history of machine learning could be track back to decades or centuries ago. Dating back to the 17th century, Bayesian and Laplace's derivation of least squares and Markov chain has been widely used, it could be the foundation and tool of modern machine learning. Since 1950 when Alan Turing has proposed to create a learning machine, there is a great development has been made in the research of machine learning.[13]

With the rapid progress of Internet and computer technology in the last few years, there is an explosion of data increases, and the demand for data analysis in various industries continues to increase, how to collect information efficiently through machine learning has gradually become a main motive force for the research of machine learning technology. How to comprehensively analyze complicated and various information based on machine learning methods and make more efficient use of digital data has become the major direction of study of machine learning in the current stage which is in a big data environment [14]. Now, machine learning is widely used in various fields, for example, computer vision, speech recognition, natural language processing, spam recognition and malicious website recognition and so on.

According to different learning methods, machine learning can be classified into three different methods in general: supervised learning, unsupervised learning and reinforcement learning [14].

### 2.2.1 Supervised learning

Supervised learning could be regarded as a broad category of machine learning theory that generate a function from labeled training data after a series of calculations. The training data should include a set of samples with characteristics parameters. In supervised learning progress, every instance should be constituted with an input object (usually a set of vector matrix) and a desired output value (also known as a supervised label). Supervised learning methods is generally used to analyze the sample features in training data to produce a function, which could be used to map out new samples to a predicted label. The main task of supervised learning is to train the potential relationship between eigenvalues and labels in learning samples, and finally make correct predictions for the labels of new samples. Supervised learning is generally used to solve classification problems and regression problems [15]. In the regression problem, it is often necessary to predict continuous specific values, while in the classification problem, the main goal of supervised algorithm is to classify materials, which is the prediction of discrete values. Classification problems are very common in the practical application of network security, such as spam classification, malicious traffic classification, malicious URL detection and so on [10]. The machine learning algorithms applied in this project belong to supervised learning.

The general process of using supervised learning to solve classification problems is shown in the figure below:

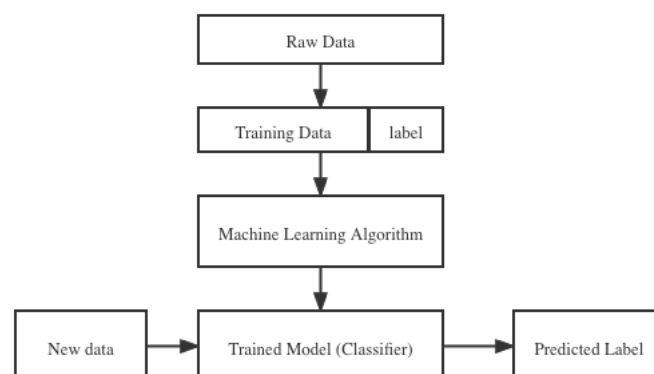


Figure 1: Machine learning process

Firstly, a large number of data will be preprocessed to extract appropriate features, labeled according to different categories, then transformed into the format required by machine learning algorithm. Secondly, the dataset used for training will be handed over to the machine learning algorithm. The algorithm will continue to use it to calculate the training data and try to find the model with the least loss. Its training process can also be said to be a trial-and-error process [14]. The training process could be simply shown as the figure:

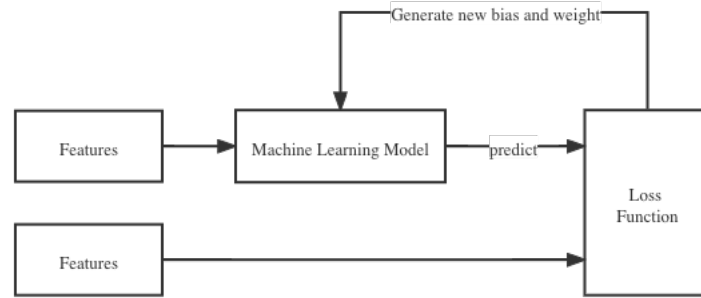


Figure 2: Machine learning training

- Model: take the feature set (  $X$  ) in the dataset as the input and return a predicted value (  $y'$  ) as the output, which can be simplified and understood as the following formula (  $b$  is bias,  $w$  is weight,  $n$  is the number of features, and  $x \in X$  ):

$$y' = b + w_1x_1 + w_2x_2 + \dots w_nx_n$$

- Calculate loss: calculate the loss under this parameter (bias, weight) through the loss function. Loss is an important indicator in machine learning algorithms that is used to evaluate the prediction accuracy of a classifier for each sample. The value of loss will be small if the predicted result is accurate, otherwise the loss will be large. The loss function is necessary for the classification model. The linear regression model usually adopts the mean square error function, while the logical regression model uses the logarithmic loss function [15].
- Calculate parameter update: detect the value of the loss function and generate new bias and weight to reduce the value of the loss function.



The model obtained through training can make more accurate prediction of new data.

In practical applications, it is often necessary to divide the dataset into two parts. One part should be used for the training task of machine learning model, that is, the training set, and the other part should be utilized as the test dataset to evaluate the prediction accuracy of the trained model [15].

### **2.2.2 Unsupervised learning**

In practical problems, it is often difficult to classify the data manually because lack of sufficient experience and prior knowledge, or the cost of manual classification could be too high. Unsupervised learning is to solve these problems. The task of unsupervised learning is to analyze the data from rows according to the training samples without marked categories, and finally distinguish the observed values [14].

### **2.2.3 Reinforcement learning**

The process of reinforcement learning could be regarded as a reward guiding behavior that agents interact with the digital environment and improve in the means of "trial and error". The object of reinforcement learning agent is to maximize the reward from action space in the specific environment. The fields reinforcement learning being distinguished from supervised learning could be the way that agents learning. The former mainly reflected in given labels, but in reinforcement learning, the label could be seen as reinforcement signal provided by the environment which is an reward or penalty of the selected action. In this way, agents constantly learn new experiences, iterate from the environment, and finally have the ability to quickly make the best action in the environment [14].

## **2.3 Machine learning algorithm used in this project**

The machine learning classifiers used in this project include random forest, naive Bayes, k-nearest neighbor (KNN) and support vector machine (SVM). These

algorithms are classical methods for solving classification problems, which could be helpful to achieve the final goal of the project.

### 2.3.1 Random forests

Random forest is a kind of classification method containing multiple decision trees. Decision tree algorithm is a classical method with tree structure, which is easy to understand and implement. Take a simple example to illustrate the decision tree, as shown in the figure:

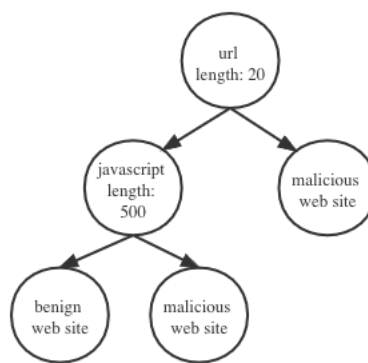


Figure 3: Decision tree

At the root node, the decision tree obtains a characteristic URL length value of 20. In general, this is normal for a web application, so it cannot be determined as a malicious website at this step. However, it is still uncertain whether it is a benign website, so continue to judge the next feature. The length of JavaScript is 500, while the length of JS in normal web pages generally does not exceed 300. Therefore, it can be judged that this web page is a malicious website. (in practical application, the situation is much more complex [14]. This example is only a brief description of the general process of decision tree)

Random forest is a combination of multiple such decision trees (Figure). There are many studies have shown that the combined classifier could have a better efficiency on classification than that of single classifier. There is a useful characteristic of Random forest classification method that it can rank the importance and evaluate the role in classification of each feature of the when it is processing the classification work [16].

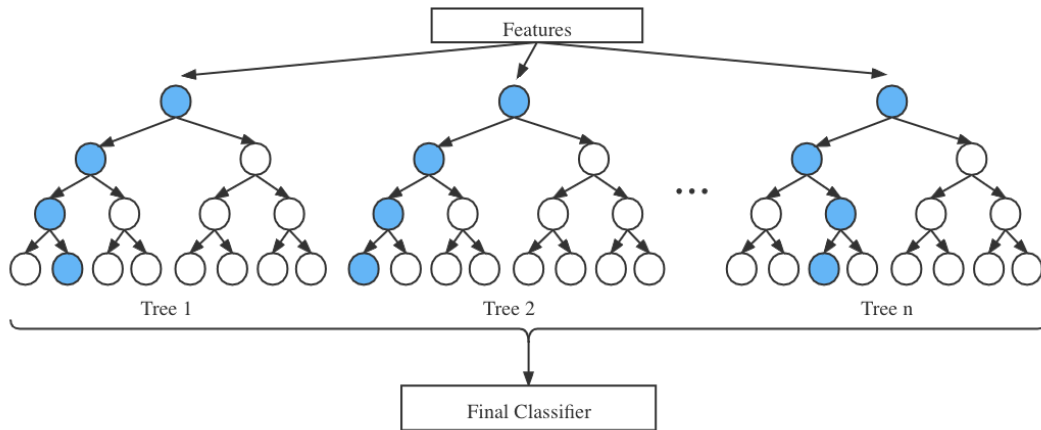


Figure 4: Random forest

### 2.3.2 Naive Bayes

Naive Bayes algorithm evolved from Bayes theorem of classical mathematics, which belongs to probability theory. The classifier based on naive Bayes algorithm is not only easy to implement, but it also has a stable performance on classification works. It can also perform well in the case of small samples. Moreover, it is not sensitive on abnormal samples, which could be a good characteristic for classification model. However, it also has disadvantages. In naive Bayes, the characteristics of samples are assumed to be independent of each other. Although this can avoid the excessive weight of a specific feature, in practical problems, there may be correlation between features, which will affect the effect of classification to a certain extent [17].

In order to illustrate the principle of naive Bayesian classification algorithm, first give a simple example. As shown in the figure, suppose there is a data set, which is composed of two types of samples: normal web application (blue) and vulnerable web application (red). Each sample has been labeled, that is, classified. For the new sample  $S(x, y)$ , Use  $P_{normal}(x, y)$  to represent the probability that the point is divided into normal web application, and Use  $P_{vul}(x, y)$  to represent the probability that the point is divided into vulnerable web application [17].

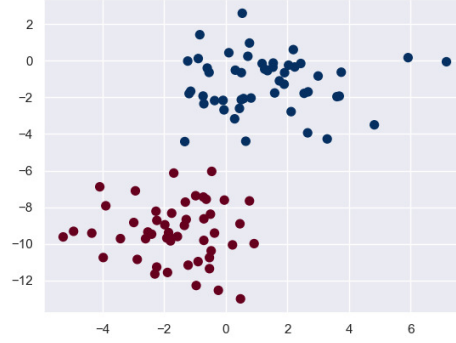


Figure 5: Naïve Bayes

Then, if  $P_{normal}(x, y) > P_{vul}(x, y)$ , the  $S(x, y)$  will be classified as the normal web site; if  $P_{normal}(x, y) < P_{vul}(x, y)$ , the  $S(x, y)$  will be classified as the vulnerable website.

The above is the case of two-dimensional features. Expand the features to d-Dimension, its mathematical principle can be shown as follows:

With sample data set  $D = \{d_1, d_2 \dots, d_n\}$ , the feature set of the corresponding sample data is  $X = \{x_1, x_2 \dots, x_d\}$ , the label set is  $Y = \{y_1, y_2 \dots, y_m\}$ , that is,  $D$  can be classified into  $m$  kind of different class. Where,  $x_1, x_2 \dots, x_d$  is independent and random, then the priori probability of  $Y$ ,  $P_{prior} = P(Y)$ ; the posteriori probability of  $Y$ ,  $P_{post} = P(Y|X)$ . According to Bayesian formula, the posteriori probability  $P_{post}$  can be calculated from  $P_{prior} = P(Y)$ ,  $P(X)$  and  $P(X|Y)$ :

$$P(Y|X) = \frac{P(Y)P(X|Y)}{P(X)}$$

Since naive Bayes is based on the independence of each feature, the above equation can be written as:

$$P(X|Y = y) = \prod_{i=1}^d P(x_i|Y = y)$$

The posteriori probability can be calculated from the above two equations:

$$P_{post} = P(X|Y) = \frac{P(y_i) \prod_{j=1}^d P(x_j|y_i)}{\prod_{j=1}^d P(x_i)}$$

Since the size of  $P(X)$  is fixed, it is only necessary to compare the molecular part of the above formula when comparing the a posteriori probability. Therefore, the formula of the label  $y_i$  to which the sample data belongs can be obtained:

$$P(y_i|x_1, x_2, \dots, x_d) = \frac{P(y_i) \prod_{j=1}^d P(x_j|y_i)}{\prod_{j=1}^d P(x_j)}$$

### 2.3.3 K-nearest neighbor

K-nearest neighbor (KNN) can be traced back to 1951, it was proposed by Fix and Hodges as a statistical algorithm, then it was expanded by Cover. After years of development, KNN algorithm is a relatively mature and simple method to solve classification problems. Its main idea is to find k nearest samples in the feature space for a new sample. If most of the K samples belong to a category, the new sample can be classified into this category. The advantage of KNN algorithm is that it is relatively easy to understand and implement, and in principle, it does not need the process of training. However, its disadvantages are also obvious. In the condition of samples with high-dimensional features, its calculation pressure will be very high. It also fails to perform well in unbalanced samples [15].

The following is an example to illustrate the principle of KNN. As shown in the figure, it is assumed that in a two-dimensional feature space, square samples represent vulnerable website, triangular samples represent normal website, and circular samples represent new samples. It can be intuitively observed from the figure that if the value of K is set to 3, the new sample will be classified as normal website. If K value is set to 5, the new sample will be marked as vulnerable website.

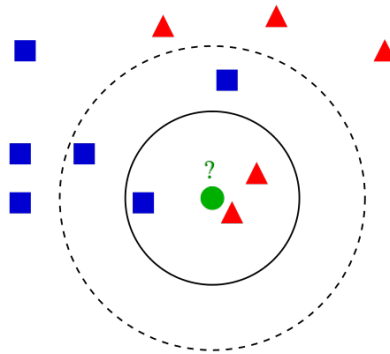


Figure 6: KNN

### 2.3.4 Support vector machine

Support vector machines (SVM) is a set of supervised machine learning method which is commonly used to binary classification problems. Its decision boundary is

a maximum-margin hyperplane which is the optimal solution to solve the feature matrix of training samples [18]. It has many advantages. For example, when the dimension of the feature matrix is relatively high, it can also classify the data set better. Moreover, SVM is quite effective even when the dimension of feature matrix is higher than the number of samples [19]. However, it also has some disadvantages. For example, fitting may occur when the dimension of the characteristic matrix is much larger than the number of samples. In addition, SVM will facing too much computation when the sample size is very large, so it could be not suitable for large data sets [20][21].

The basic idea of SVM is as follows:

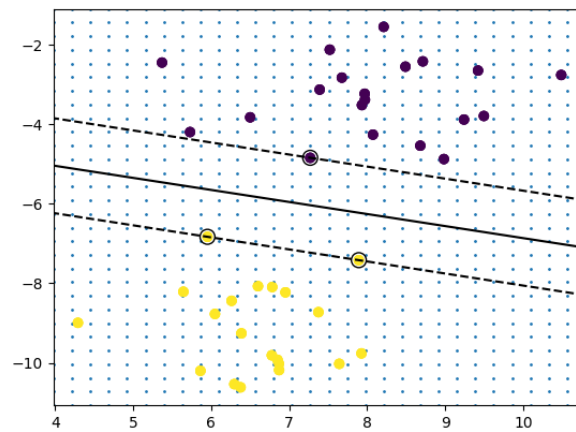


Figure 7: SVM

Take a simple example to illustrate the basic principle of SVM, as shown in the figure. Suppose that in the two-dimensional feature space, dark points represent the sample as vulnerable web application, and the light points represent normal web application. What SVM needs to do is to find a line that can divide the two-dimensional space into two parts, which contains two different web applications. And this line should be optimal, that is, even if a new sample is added to this space, the line can successfully divide the new sample into the correct area [18].

## 2.4 Machine learning evaluation methods

After machine learning model training, an essential step is to evaluate the established model and evaluate whether it meets the needs of use.

### 2.4.1 Confusion matrix

The classification judgment result of the classifier for an instance may be positive or negative. Similarly, the judgment may be true or false. There are 4 conditions that could be combined as a table, namely, confusion matrix [11].

- True positive: It means that the positive result which predicted by classifier is correct.
- False positive: It suggests that the positive result predicted by classifier is wrong.
- True negative: It indicates that the negative result predicted by classifier is correct.
- False negative: It means that the negative result predicted by classifier is wrong.

	Predicted positive	Predicted negative
Actually positive	True positive (TP)	False negative (FN)
Actually negative	False positive (FP)	True negative (TN)

Table 1: Confusion matrix

### 2.4.2 Performance indicator

Accuracy is used to represent the percentage of correct prediction in all prediction results of the classifier [11].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision is used to indicate how many of the results predicted as positive by the classifier are correct [11].

$$Precision = \frac{TP}{TP + FP}$$

Recall is used to indicate how many samples with positive labels are correctly classified by the classifier [11].

$$Recall = \frac{TP}{TP + FN}$$

F1 score is an evaluation index combining precision and recall [11].

$$F1\ score = \frac{2(Recall \times Precision)}{Recall + Precision}$$

## 2.5 Python Libraries

This project utilizes python as programming language. The syntax of Python is relatively simple and easy to learn for starters, so it is widely welcomed by developers come from various fields and becomes one of the popular programming languages in recent years. Another powerful advantage of python is that its community provides a large number of third-party libraries, which provide a variety of useful functions, covering the fields of scientific computing, web development, web crawling and machine learning, and most of them are mature and stable. These third-party libraries are very important for Python development. They can greatly improve the efficiency of development. The main libraries used in this project are as follows:

**requests:** It is a HTTP library which is widely used in crawling. In this project, it is used to send request to the target web application and receive its response. [22]

**BeautifulSoup:** It contains some functions that can conveniently extract the content from web pages. [23]

**whois:** It is mainly used to query whether the domain name has been registered, which is one of the features required by the classification model in this project. [24]

**geoip2:** It is used to query the geographical location of IP, which is used as a feature in the prediction model. [25]

**tld:** It can easily extract top-level domain names from complex URLs. The TLD is a feature that will be used in the classifier in this project. [26]

**re:** It is a basic library of python, which is mainly used to string matching.



**numpy:** It is a powerful mathematical function library. It supports multi-dimensional array operation and matrix operation. It is an important library in the field of machine learning. [27]

**pandas:** It is a library built on Numpy, which is mainly used to easily operate Numpy data. [28]

**sklearn:** Its full name is scikit-learn. It is one of the most popular Python modules in the field of machine learning. It includes a variety of machine learning methods, such as classification, progression, clustering, dimensional reduction, model selection and preprocessing. The utilization of scikit-learn makes the development of machine learning more simple and convenient, and reduces the amount of code to a great extent. [29]

**matplotlib:** It is a powerful Python module which is used to data visualization. [30]

**imblearn:** It is a library contains some methods to process unbalanced dataset. [31]

## 2.6 OWASP ZAP

ZAP is a tool for web penetration test produced by OWASP company. It supplies the functions to scan and test a variety of vulnerabilities in the web page [32]. ZAP provides a set of API for Python [33]. Its functions can be easily called through API in Python programs. In this project, ZAP + API is used as a vulnerability exploitation tool to complete the final work of penetration test.

## 2.7 Related works

Alam et al. Proposed a method called NMPREDICTOR to predict vulnerabilities in web pages. This method is actually based on the white box test in penetration testing, that is, using machine learning to review the code, and abstracting vulnerability prediction as a text feature extraction and classification problem of

machine learning. The precision of the prediction results of vulnerabilities in the website framework Drupal reached 84.9%. [34]

Aldwairi and Alsalman proposed a method to identify malicious websites based on URL features. Their method is based on Naive Bayesian classifier and uses genetic algorithm. They complete the training of this prediction method in a short time with a small data set and low memory consumption. The prediction accuracy of this method for malicious websites reaches 87%. [35]

Calzavara et al. proposed a method called Mitch to predict CSRF vulnerabilities in web pages. They trained a machine learning model to judge the target through some characteristics of web page requests. The prediction accuracy of their method for CSRF vulnerabilities is about 74%. It is said that they successfully found CSRF vulnerabilities in the actual website test, and several vulnerabilities were not found in the traditional vulnerability analysis software. [36]

Bauer, Fung and Jia proposed a lightweight method to detect DOM XSS vulnerabilities in web pages. They use crawlers to collect a large number of web pages and extract more than 18 billion JavaScript function codes, including secure code and potentially vulnerable code. Using these data, they trained a classifier based on deep neural network to predict whether there may be DOM XSS vulnerabilities by analyzing the characteristics of JavaScript code. According to their conclusion, the prediction accuracy of this method for DOM type XSS vulnerabilities is 94.5%. [37]

Shar and Tan proposed a machine learning based method to predict SQL injection vulnerabilities and XSS vulnerabilities in web applications. They use supervised learning and unsupervised learning methods to mix static analysis and dynamic analysis of the code features of web applications, and finally predict the results. According to their experiments, they have a supervised learning method to predict vulnerabilities with an accuracy of 85. [38]

Kamtuo and Soomlek studied SQL injection vulnerability prediction based on machine learning. In their work, they used about 1000 samples to train a variety of

machine learning classification models (SVM, boosted decision tree, artificial neural network, decision jungle) to conduct text analysis on the server code of web application. After comparison, the accuracy of vulnerability prediction of the decision jungle model they trained is the highest among several models, Most SQL vulnerabilities can be accurately predicted. [39]

Govil, Gupta and Singh specifically studied the prediction of XSS vulnerabilities in web applications. They established some machine learning models to analyze the context by extracting the features in the web source code, so as to predict whether there are XSS vulnerabilities in web pages. They used SVM, Nb, Bagging, J48 and JRip classifiers to test on public data sets. Their experiments show that Bagging performs best among these classifiers and can accurately predict XSS vulnerabilities in web pages. [40]

Fang et al. have proposed a method based on semantic analysis for static detection of JavaScript code. They use JavaScript code to generate a syntax tree, format it into a sequence of syntax units, and then use text processing algorithms to convert it into the form of word vectors. Finally, the dataset is used to train the machine learning model. According to their experiments, the detection accuracy of this model for malicious JavaScript code can reach 97.7%. [41]

## **Chapter 3**

### **Requirement specification and system design**

#### **3.1 Requirement specification**

In order to achieve the final goal of the project, it is necessary to analyze the needs of the project based on the goal. Requirement analysis is to more clearly point out what functions need to be implemented in the project, which is very helpful for subsequent development.

##### **3.1.1 Functional requirement**

The final goal of this project is to complete a penetration testing tool that can automatically predict whether there may be injected malicious code in web pages, and scan and exploit web pages which may be compromised. This goal can be divided into three modules: crawler module, machine learning classifier module and vulnerability scanning module.

Crawler module: This module mainly needs to realize two functions: traversing the website and reading website information. First, in this module, the user is allowed to enter a URL of the target web application. This module needs to be able to traverse the target and find all the URLs under the target site as much as possible. Secondly, the module needs to read the information of the target site and page, save and format the data, and prepare to submit it to the second module.

Machine learning classifier module: The module should mainly realize two part in function. Firstly, users can select the trained model in this project to complete the subsequent web page classification, and the performance of the model should be given. Secondly, users can also choose to use their own data set to train the model and choose the most appropriate one according to the performance of the model which could be used to predict compromised webpages.

Vulnerability scanning module: The main function of this module is to scan the web pages predicted as risky by the classifier and verify the existing vulnerabilities. In addition, the results of this web application penetration test shall also be displayed.

### 3.1.2 None- functional requirements

Accuracy: the final program should be able to identify the compromised web pages accurately.

Efficiency: it should be more efficient than the traditional web application vulnerability scanning tool.

Usability: it should be convenient to use and have the necessary user interaction.

## 3.2 System design

According to the above requirements analysis, this project is mainly divided into three functional modules, namely crawler module, classifier module and vulnerability scanning module. According to the tasks of each module, the main structural design of the project is as follows:

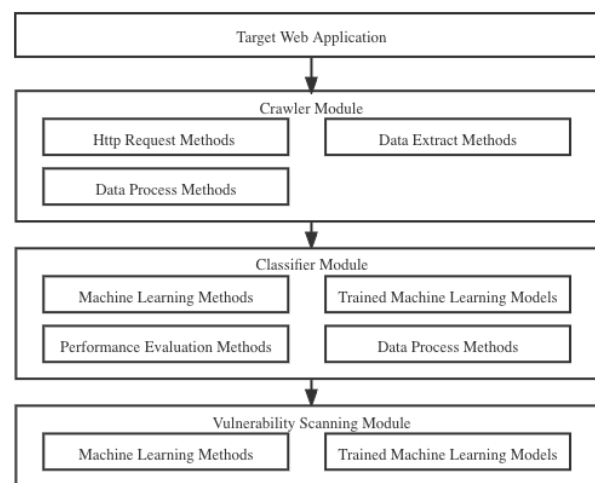


Figure 8: Program structure

In the crawler module, there are three main functions. HTTP request methods represents a group of methods used to send and receive requests to the target web application, and it also has the ability to automatically traverse the whole web

application to obtain and the content of all web pages in the target. Data extract methods contains some methods to extract the contents of web pages and save them in a list. Data process methods is used to process those data and convert them into the data format supported by the machine learning model. Next, the processed data will be submitted to the second module, named, classification module.

The main components of the classification module include machine learning methods, trained machine learning models, performance evaluation methods and data processing methods. Machine learning methods includes the implementation of some machine learning algorithms (Random forest, Naïve Bayes, KNN and SVM). These algorithms need to be trained with training dataset. After training, the corresponding trained machine learning models can be obtained. These trained models can be used to classify the data submitted in the crawler module and predict whether there may be malicious vulnerabilities in the target web application. The performance evaluation method includes some evaluation methods on the performance of classification models (precision, accuracy, recall, F1 score), which are used to compare the performance differences between trained classifiers. The data processing method in this module is used to preprocess the training data to meet the needs of model training.

In the last part, the vulnerability scanning module is mainly composed of OWASP zap and its API. According to the judgment results of the classifier, web pages with vulnerability risk can be submitted to OWASP zap through API for vulnerability scanning and verification and the result will be shown.

In general, the main processes of the project are as shown as below:

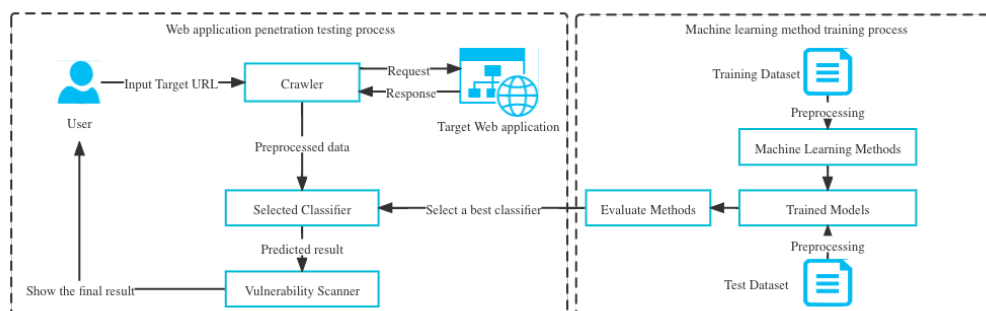


Figure 9: Program processes

In the web application penetration testing process, first, the user inputs the URL of a target web application, and the crawler module will crawl the content of the web application from the web page pointed to by the URL link. After traversing, each web page will be regarded as a sample, and the data corresponding to each page will be preprocessed and converted into the form of characteristic matrix, then submit to the classifier. The classifier will classify these samples, that is, predict whether there may be malicious code in the samples. Then submit the URL corresponding to the sample which is predicted as compromised to the vulnerability scanning module for vulnerability verification, and finally print the scanning results.

In the process of machine learning training, first preprocess the training data set and test data set, and then use the processed training set to train the machine learning model (random forest, naive Bayes, KNN and SVM). After obtaining the trained model, use them to classify the samples of the test data set, Then the evaluation method is used to test their classification results. By comparing the precision, accuracy, recall and f1score of these models, the best model is selected as the classifier in the penetration test process.

## **Chapter 4**

### **Implementation**

This chapter will introduce the implementation of the project in detail according to the three modules mentioned in the system design (classifier module, crawler module and vulnerability scanning module).

#### **4.1 Classifier Module**

Firstly, the work of classifier module is carried out because this module plays a very important role in the whole project. It not only processes the sample data obtained by crawlers from web pages and classifies them according to whether there may be vulnerabilities, but also calls the vulnerability scanning module according to the classification results. The work completed by this module is similar to the threat modeling and vulnerability analysis stage in the process of penetration testing. Different from the traditional methods, in this project, machine learning model is used to assist testers to complete these works, in order to improve the efficiency of penetration testing to a certain extent.

In this project, the implementation of the classifier module can be mainly divided into the following steps: data collection, feature selection, data preprocessing, training several machine learning models, using the test set to verify the model and compare the performance of different trained models to choose the best classifier. Next, this paper will introduce the specific implementation methods and processes of these steps one by one.

##### **4.1.1 Data collection**

In order to train machine learning model, a large number of high-quality samples are needed. Because the principle of machine learning is to use a series of mathematical theories to calculate the characteristics of samples, the quality of samples often affects the performance of the machine learning model it trains. However, this is also one of the difficulties in the implementation of the project,



because it is a very arduous task to obtain many high-quality malicious web pages with vulnerability. Due to time constraints, it is unrealistic to obtain enough samples independently during the implementation of the project. Therefore, the data set provided by Singh is used in the project, this is a data set containing samples of malicious websites and benign websites. The number of samples is about 1.5 million. This data set is provided in CSV format and can be easily processed using pandas.

In this project, the task of the machine learning model is to accurately identify malicious web pages (web pages implanted with malicious code because vulnerabilities are exploited by hackers) according to several characteristic data. Therefore, this data set provided by Singh is very suitable for machine learning training of this project. It is reasonable to use malicious web pages to locate vulnerabilities more effectively because there are many studies have shown that the way of malicious web page attack is often through the utilization of web page vulnerabilities. For example, according to Liu and Zhong's research, a web page with vulnerabilities is likely to be broken by hackers, and then secretly used to spread web malware [42]. Christin and Soska have provided a method to predict whether vulnerable web pages will become malicious web pages [43]. Eshete et al. Also pointed out in the research that malicious web pages are closely related to vulnerability exploitation [44]. The above research shows that there are some common characteristics between malicious web pages and web pages with vulnerabilities. In summary, it is feasible to use malicious web pages to predict whether web pages have vulnerabilities.

In this dataset, the features contained in the sample are shown in the following table:

Feature	Description
url	Uniform resource locator (URL) string of web application.
ip_add	Host IP address of the web application.

geo_loc	The geographic location corresponding to the IP address of the web application.
url_len	The number of characters in the URL.
js_len	The length of JavaScript code, which is calculated by dividing the total number of characters contained in the string of JavaScript code in the web page by 1000.
js_obf_len	The length of obfuscated JavaScript code, which is calculated by dividing the total number of characters contained in the string of obfuscated JavaScript code in the web page by 1000.
tld	Top level domain of the web application.
who_is	To Indicate whether the domain name of the web application has been registered.
https	Used to indicate whether the HTTPS protocol is enabled for the web application.
content	Text content and JavaScript code of web pages.
label	Classified tags, malicious web application is marked as bad, and benign websites are marked as good.

Table 2: Features in dataset [45]

In addition to the original features above in the dataset, a new feature, path\_depth, is added in this project according to the depth of the URL.

#### 4.1.2 Feature selection

Feature selection is a very important work in machine learning, because the principle of machine learning is to continuously iterate and learn through the operation of the features of the sample, so whether the feature selection is appropriate or not often affects the classification accuracy of the trained machine learning model. In this project, two different schemes are used for feature extraction. One scheme is to select comprehensive features, another scheme is to extract text features from JavaScript code. The reasons for using two methods for feature extraction in this project will be described in detail below.

*Comprehensive features:* This scheme is mainly based on the idea that there are some similar characteristics between malicious websites and vulnerable webpages. To select appropriate features, the difficulty of data preprocessing was considered, `geo_loc`, `url_len`, `js_len`, `js_obf_len`, `tld`, `who_is`, `https` and `path_depth` is selected as the features used to train machine learning models.

*JavaScript code:* Javascript code is used for text feature extraction, and the JavaScript code in the sample is transformed into the form of feature vector that can be recognized by machine learning model. This method is used because vulnerabilities in web applications may be caused by not paying attention to filtering some inputs in JavaScript code, such as XSS vulnerability, SQLInjection and remote command execution. Therefore, we can analyze the JavaScript code in the web page through machine learning to find out the potential characteristics of the codes which causes the vulnerability, so as to predict whether there may be a vulnerability in the web page.

#### 4.1.3 Data preprocessing

In machine learning, data preprocessing could be an essential work, because the data in the raw dataset could not be directly recognized by the machine learning model, and there may be some abnormal data in the dataset. Direct use will affect the training effect of machine learning, and even some program errors may occur.

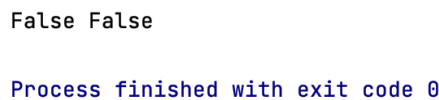
The data set used in this project has been divided into training set and test set. Firstly, check the data set to see if there are abnormal null values. The main python codes are shown in Figure 10.



```
import pandas as pd
import numpy as np
# Load Datasets
def loadDataset(file_name):
    df = pd.read_csv(file_name)
    return df
data_train = loadDataset("Webpages_Classification_train_data.csv")
data_test = loadDataset("Webpages_Classification_test_data.csv")
# Ensuring correct sequence of columns
data_train = data_train[['url', 'ip_add', 'geo_loc', 'url_len', 'js_len', 'js_obf_len', 'tld', 'who_is', 'https', 'content', 'label']]
data_test = data_test[['url', 'ip_add', 'geo_loc', 'url_len', 'js_len', 'js_obf_len', 'tld', 'who_is', 'https', 'content', 'label']]
# Check the abnormal value
print(np.any(data_train.isnull()), np.any(data_test.isnull()))
```

Figure 10: Code block of data checking

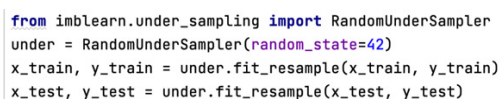
The *loadDataset()* function includes *pandas.read\_csv()*, which is a function used to load csv files, *numpy.any()* can be used to traverse the whole dataset. *Pandas.isnull()* will check whether there are null values in the data. If there are null values, it will return True, otherwise it will return False. The result printed in the terminal are as below, it means that there is no null value in training set and test set.



```
False False
Process finished with exit code 0
```

Figure 11: Result of data checking

The second point to note is to check whether the proportion of positive and negative samples in the data set is balanced, because too unbalanced samples will affect the training results of machine learning and lead to the problem of over fitting in prediction. According to the description document of the dataset, there are approximately 2% samples which are labeled as bad, while the proportion of samples marked as good are almost 98%. Therefore, corresponding processing should be made for this data set to balance the proportion of positive and negative samples. There are two ways to deal with unbalanced samples, under sampling and oversampling. Under sampling makes the data balanced by reducing the number of positive examples. In contrast, oversampling uses some methods to add negative sample data to balance the data [46]. Under sampling has been adopted in this project, the main codes are as follows (Figure 12):



```
from imblearn.under_sampling import RandomUnderSampler
under = RandomUnderSampler(random_state=42)
x_train, y_train = under.fit_resample(x_train, y_train)
x_test, y_test = under.fit_resample(x_test, y_test)
```

Figure 12: Code block of under sampling

The code above uses *RandomUnderSampler()* to create an object which contains *fit\_resample()* to under sampling. The parameter *random\_state* in *RandomUnderSample()* is random seed which is used to ensure that the results of each random under sampling are the same. There are two parameters in *fit\_resample()*, *x\_train* should be the feature columns and *y\_train* should be the label column. After processing, the ratio of positive and negative samples in the data set is balanced to 1:1.

According to the two different feature selection methods, the subsequent data preprocessing work is also different. The two processing processes are described in detail below.

### Data preprocessing for comprehensive features

Firstly, because the machine learning models used in this project are based on mathematical functions, some discrete text data in the dataset cannot be processed by mathematical methods. For example, for the column *https* in the dataset, its original data is a string with a value of "yes" or "no", which could not be recognized by the machine learning model, so a method should be used to convert it into digital data of 1 or 0. Sklearn provides a convenient data conversion method in its data preprocessing package, which is *sklearn.preprocessing.LabelEncoder*. The main codes for data format conversion are as follows (Figure 14).

```
# transfer data into discrete value
def encodeLabel(col):
    labelEn = LabelEncoder()
    return labelEn.fit_transform(col.values)
```

Figure 13: Code block of data encode

In this function, *labelEn* was instantiated as a *LabelEncoder* object, then it calls the data formatting method through *fit\_transform()* and return the result. The parameter *col* should be a feature column which need to be digitized and the *col.value* is the data for each row.

Secondly, define a function to calculate URL depth of the sample (Figure 13). The parameter *col* should be the column of *url* in the dataset. It will traverse the data of this column and return a list containing the information of path depth.

```

# get path_depth
def getPathDepth(col):
    path_depth = []
    url = ""
    depth = 0
    if(len(col) > 0):
        for ele in col:
            if(ele.startswith("http://")):
                url = ele.strip("http://")

            elif(ele.startswith("https://")):
                url = ele.strip("https://")
            else:
                pass
            depth = url.count("/")
            path_depth.append(depth)
    return path_depth

```

Figure 14: Code block of get URL depth

After calculating the *path\_depth* value of each row of data, the result matrix should be spliced into the original dataset. Pandas provides the method *concat()*, which can easily realize the splicing of multiple matrices (Figure 15). In the *concat* parameter, the matrix list to be spliced is required. The parameter *axis* is used to set the splicing mode, *axis = 1* means horizontal splicing, namely, adding columns.

```

# add path_depth to dataset
path_depth = pd.DataFrame({"path_depth": getPathDepth(datas_train["url"])})
datas_train = pd.concat([datas_train, path_depth], axis=1)

```

Figure 15: Code block of splicing column

## Data preprocessing for JavaScript code

In malicious website, the main way to harm visitors could be regarded as the exploitation of the vulnerabilities in the JavaScript code of web application. Therefore, analyzing the JavaScript code in these malicious websites could be a solution to find out the potential compromised web pages [47].

In the dataset, the *content* column is the raw content extracted from each web page sample, which contains JavaScript code. In order to extract these codes, the processing method is as Figure 16. In *getJs(col)* function, the parameter *col* should be the *content* column of the dataset. the list *js\_arr* is used to save the JavaScript code extracted from content, the function will traverse *col* and use *re.findall(R '<script(.\*)</script>', ele, re.DOTALL)* to match the JavaScript code in each sample *ele* using regular expressions of the first parameter. The third parameter *re.DOTALL* means to match all characters.

```

def getJs(col):
    js_arr = []
    if(len(col) > 0):
        for ele in col:
            str_arr = re.findall(r'<script(?:.*?)>',ele, re.DOTALL)
            js_str = ""
            for ele in str_arr:
                js_str += ele
            if(js_str != ""):
                js_arr.append(js_str)
            else:
                js_arr.append("Null Javascript Code")
    return js_arr

```

Figure 16: Code block of get JavaScript code

The analysis of JavaScript code belongs to the problem of text classification in machine learning applications. In order to solve the problem of text classification, the next step should be extracting the features of the text and convert the features into the form of matrix for the calculation of machine learning model. There are two main ideas for text transformation, Bag of Word model and the word vector model. The method of bag model is to put all the words in the text into one bag, regardless of word order and grammar [48]. For example, for the two texts "it is a vulnerable web application" and "it is a benign web application", their word bags are expressed in array form as ["it", "is", "a", "vulnerable", "benign", "Web", "application"]. Then, the two texts above can be expressed in vector form as [1,1,1,1,0,1,1] and [1,1,1,1,0,1,1]. The position of the element in the vector corresponds to the subscript of the word bag array, and the value of the element represents the number of occurrences of the corresponding word. Word vector model maps each word in the data set into a high-dimensional spatial vector through a large number of calculations. The relationship between words can be obtained by calculating cosine [41].

Considering that the principle of word bag model is relatively simple and easy to implement, word bag model is adopted in this project. The main code is as below.

```

# JavaScript code feature extraction
from sklearn.feature_extraction.text import CountVectorizer
transfer = CountVectorizer()
js_train = transfer.fit_transform(x_train["js"])
js_test = transfer.fit_transform(x_test["js"])

```

Figure 17: Code block of code feature extraction

*CountVectorizer* is a class in sklearn, which provides a text feature extraction method based on word bag model. As shown in the Figure 16, *The contavectorizer ()* is used to instantiate an object *transfer*, and then call the *fit\_transform ()* method

to vectorize the text in the parameters ( $x\_train[\"js\"]$  and  $x\_test[\"js\"]$ , which are the *js* columns in the training set and the test set, respectively).

#### 4.1.4 Machine learning models training

After preprocessing the data, we can use the data to train the machine learning model. In this project, for the two feature extraction methods, four machine learning algorithms, random forest, naive Bayes, KNN and SVM, are used to create classifiers to evaluate the performance of different classification algorithms and the effect of the two feature extraction methods. The following describes the main codes according to these four different algorithms (Because several codes for machine learning model training are the same whether using comprehensive features or text features, only the codes of comprehensive features are listed as instructions).

##### Random forest

The class *sklearn. Ensemble. Randomforestclassifier* is provided in sklearn, which contains the implementation of the random forest algorithm. The code is as follows.

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=10)
classifier.fit(x_train, y_train)
y_predict = classifier.predict(x_test)

import pickle
file = open("randomForests_js.model", "wb")
pickle.dump(classifier, file)
file.close()
```

Figure 18: Code block of random forest model training

*RandomForestClassifier(n\_estimators=10)* is used to instantiate a *classifier* object, where the parameter *n\_estimators* represents the number of decision trees in random forest. *classifier.fit(x\_train, y\_train)* means to start training the model, parameter *x\_train* is the feature matrix of the training set, and *y\_train* is the label column of the training set. After the training, the samples in the training set *x\_test* can be classified through *classifier.predict(x\_test)*. In the end, *pickle* is used to save the trained model to a file for later use in comparison work.



## Naïve Bayes

For naive Bayes classifier, sklearn provides a variety of algorithm implementations, namely GaussianNB, MultinomialNB and BernoulliNB. MultinomialNB is used in this project because this algorithm is suitable for discrete features. The main codes for training naive Bayes classifier are as follows (Figure 19). Similar to the code above, first use MultinomialNB () to instantiate a model object classifier, then call fit (x\_train, y\_train) to train the model, and finally save the classifier obtained after the training to the file.

```
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
classifier.fit(x_train, y_train)
y_predict = classifier.predict(x_test)

import pickle
file = open("naiveBayes.js.model", "wb")
pickle.dump(classifier, file)
file.close()
```

Figure 19: Code block of naïve Bayes model training

## KNN

The process of creating a KNN classifier is similar to the above work. It is realized by training the instantiated objects of the *KNeighborsClassifier*, which is the class of sklearn. However, for the *KNeighborsClassifier* class, it is necessary to determine the value of the parameter *n\_neighbors* to obtain a KNN classifier with better performance. In order to determine *n\_neighbors*, k-cross validation is a common method. Its principle is: divide the original data into *k* groups (generally evenly), make every subset dataset as a verification set respectively, and the other *k-1* subset data as the training set. In this way, there are *k* models will be obtained, and the average of the classification accuracy of the final verification sets of *k* models will be used as the performance index of the KNN classifier. This project used 4 cross validation; the code is as Figure 20. After 4 cross validation, it is finally obtained that among the values of 15, 20, 25, 30, 35 and 40, When *n\_neighbors* = 15, KNN has the highest accuracy, as shown in Figure 21.

```

folds = 4 # it means 4 cross validation
k_choices = [15, 20, 25, 30, 35, 40] # the n_neighbor value being validated

X_folds = vsplit(x_train, folds) # split dataset in vertical
Y_folds = hsplit(y_train, folds) # split dataset in horizontal
accuracy_k = {}
for k in k_choices:
    accuracy_k[k] = [] # create a matrix to save accuracy of each KNN model
for i in range(folds):
    X_train = vstack(X_folds[:i] + X_folds[i + 1:]) # generate a new sub training set
    X_val = X_folds[i] # generate test set
    Y_train = hstack(Y_folds[:i] + Y_folds[i + 1:]) # generate the label of sub training set
    Y_val = Y_folds[i] # generate the label of test set
    for k in k_choices:
        knn = KNeighborsClassifier(n_neighbors=k) # instantiate KNN classifier object
        knn.fit(X_train, Y_train) # start to train
        Y_val_pred = knn.predict(X_val) # predict the label of test set
        accuracy = mean(Y_val_pred == Y_val) # calculate the accuracy of KNN classifier with n_neighbors=k
        accuracy_k[k].append(accuracy) # record accuracy to a list

```

Figure 20: Code block of 4 folds cross validation

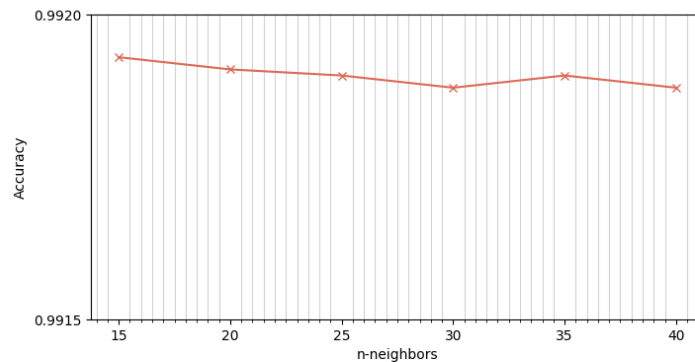


Figure 21: Result of validation

## SVM

A variety of kernel functions are provided in the SVM class of sklearn, which can be roughly divided into linear and nonlinear. According to the study that compared supervised classification models, [49] the linear kernel performs better than several other nonlinear kernels in the task of text classification. The SVM model with linear kernel is adopted in this project, and the code is as follows:

```

from sklearn import svm
classifier = svm.SVC(kernel="linear") # select linear kernel function
classifier.fit(js_train, y_train)
y_predict = classifier.predict(js_test)

import pickle
file = open("svm_linear_js.model", "wb")
pickle.dump(classifier, file)
file.close()

```

Figure 22: Code block of SVM model training

### 4.1.5 Classifier evaluation and comparative analysis

In this part, for the two classification methods, four different classifiers are used to classify the samples of training set and test set respectively, and the corresponding conflict matrix, precision, accuracy, recall and f1score are calculated, and then the performance of classifiers is compared and analyzed through these indicators.

#### Confusion matrix (Comprehensive features dataset)

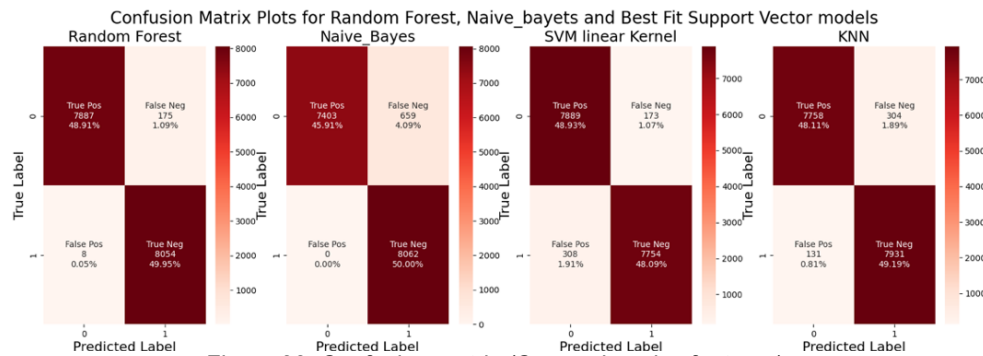


Figure 23: Confusion matrix (Comprehensive features)

Firstly, for the training results of data sets using comprehensive features, the prediction results of different machine learning models for samples can be seen intuitively from the display of the conflict matrix (Figure 23, 0 indicates malicious websites and 1 indicates benign websites). Among the prediction results of Naive Bayes classifier, the proportion of False Neg is 4.09%, which is the highest among these models, which means that it is more likely to mistakenly identify malicious web pages as benign web pages than other classifiers. For random forest, KNN and SVM, there is not a big gap in the performance of malicious web page recognition.

#### Confusion matrix (JavaScript code dataset)

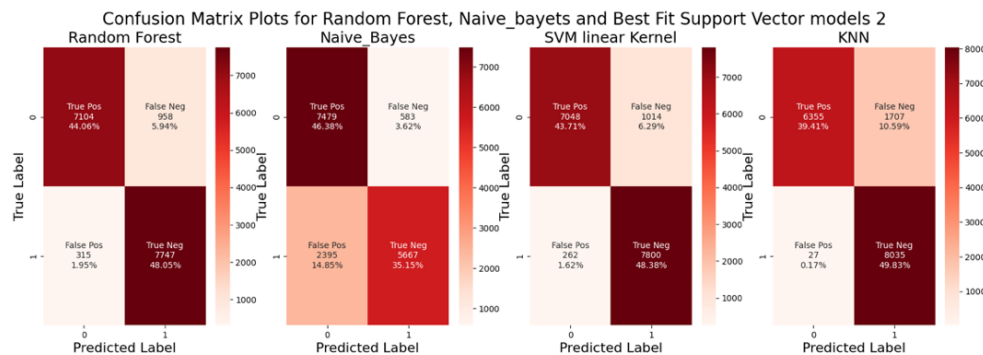


Figure 24: Confusion matrix (JavaScript code)

For the training results of this dataset, it can be seen from the conflict matrix (Figure 24) that, contrary to the above experiments, Naive Bayes has the best effect on identifying malicious websites in this dataset, and the proportion of True Pos is 46.38%. Although it identifies too many benign websites as malicious websites (the proportion of false neg is 14.85%), to some extent, it can avoid ignoring some potential malicious pages, which may be beneficial for vulnerability detection.

**Precision (Comprehensive features dataset)**

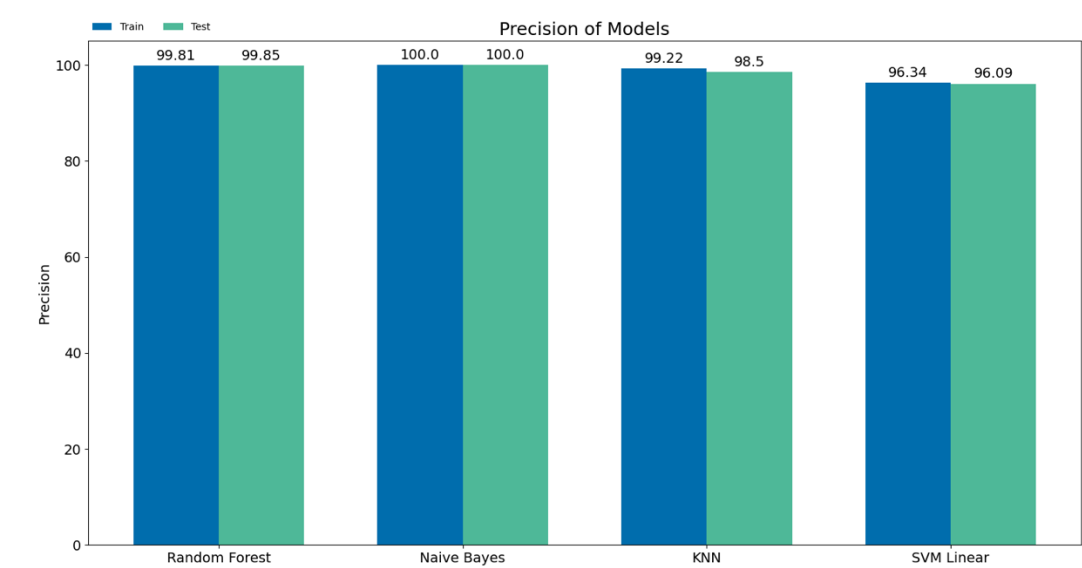


Figure 25: Precision (Comprehensive features)

It is shown in the above Figure 25 that in terms of precision, the four machine learning models perform well. Among them, Naive Bayes classifier has the highest accuracy, and the prediction precision of training set and test set reaches 100%. This means that if a sample is judged as a malicious web page by Naive Bayes classifier, the probability that the sample is actually a malicious web page is almost 100%.

## Precision (JavaScript code dataset)

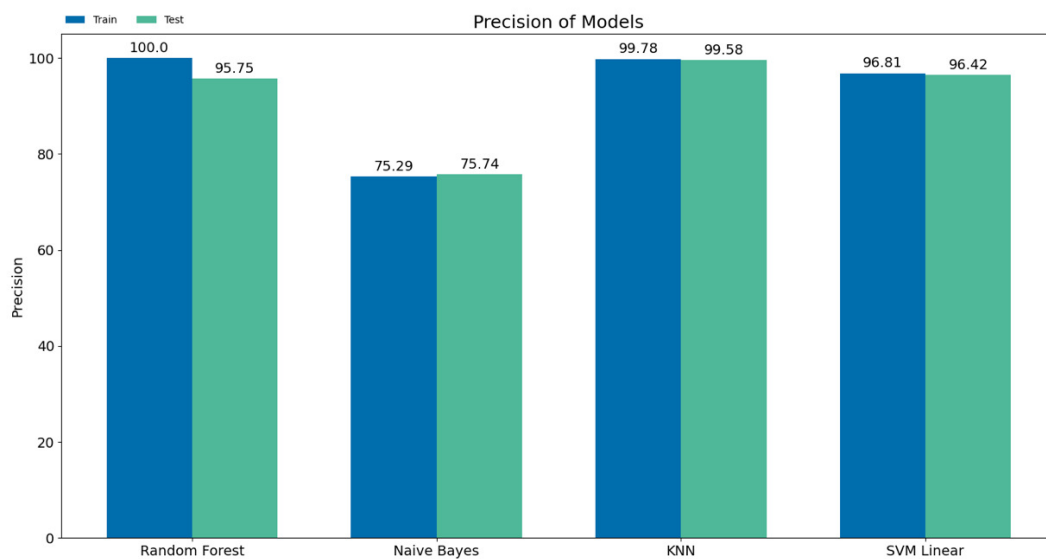


Figure 26: Precision (JavaScript code)

In the work of classifying JavaScript code, Naive Bayes is still the best classifier in precision. Its precision of prediction for training set and test set is 90.06% and 90.67%. Although the prediction accuracy of random forest in the training set is 93.9%, its performance is not stable. In the test set, its precision is reduced to 88.99%.

## Accuracy (Comprehensive features dataset)

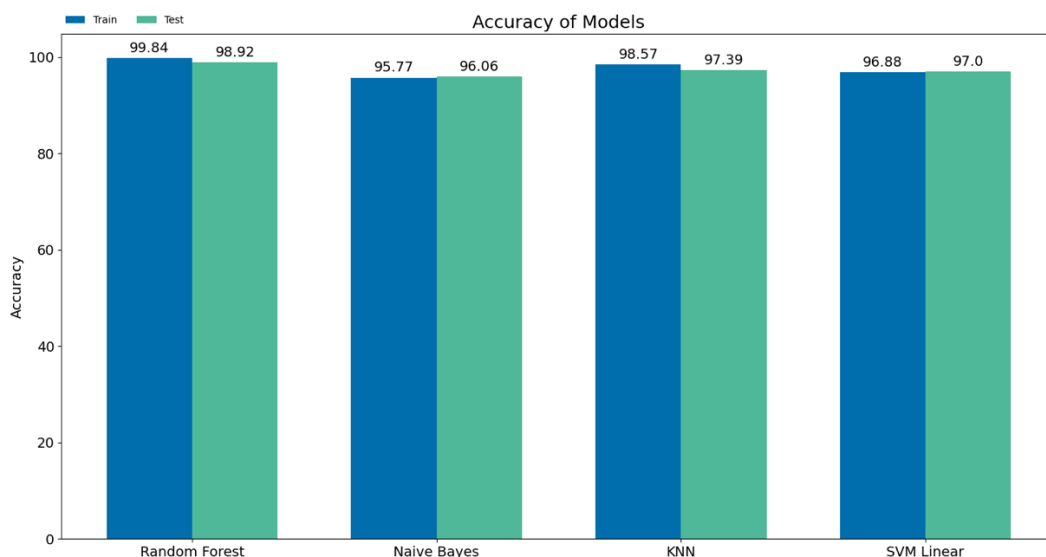


Figure 27: Accuracy (Comprehensive features)

In terms of accuracy, random forest is the best of the four models. Its accuracy of prediction results in the training set is 99.84%, and its score in the test set is 98.92%. This means that it can correctly classify most web pages.

### Accuracy (JavaScript code dataset)

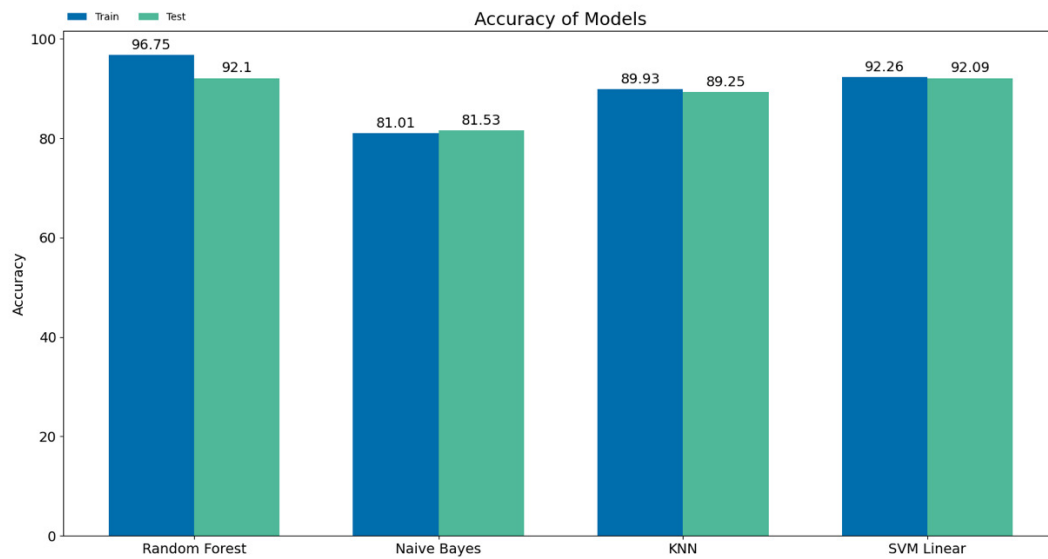


Figure 28: Accuracy (JavaScript code)

Classifiers using JavaScript as features do not perform as well as classifiers using comprehensive features in terms of accuracy. As shown in Figure 28, random forest is still the best model. Its prediction accuracy for the training set is 96.75%. Although the prediction accuracy for the test set has decreased to 92.1%, it is still the highest among several models.

### Recall (Comprehensive features dataset)

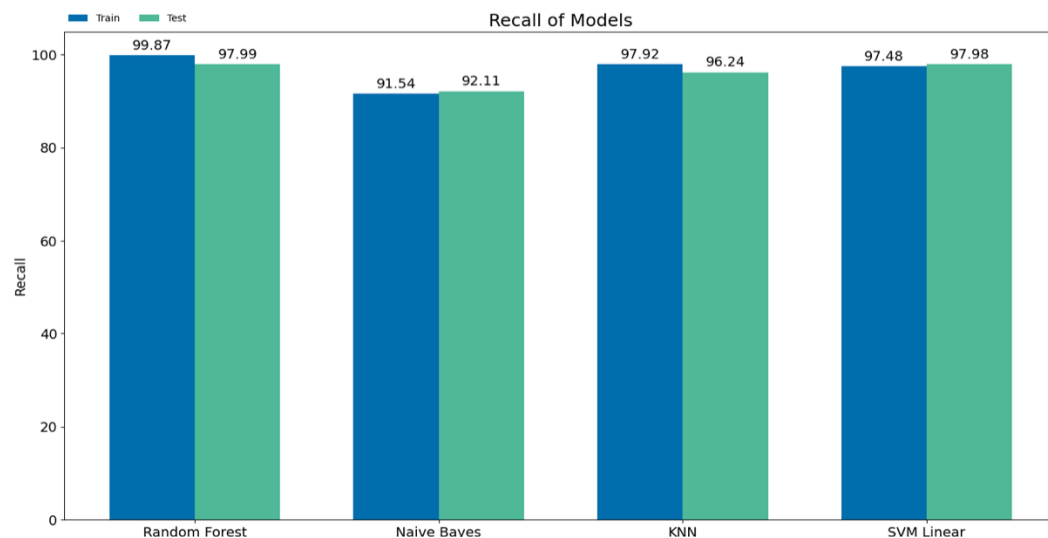


Figure 29: Recall (Comprehensive features)

Figure 29 shows the performance of recall of several different machine learning models in the case of training using data sets with comprehensive features. It can be seen that the performance of random forest classifier is the best. In the

classification of training set and test set, recall is 99.87% and 97.99% respectively. The higher recall means the classifier can identify the malicious web pages more accurately.

**Recall (JavaScript code dataset)**

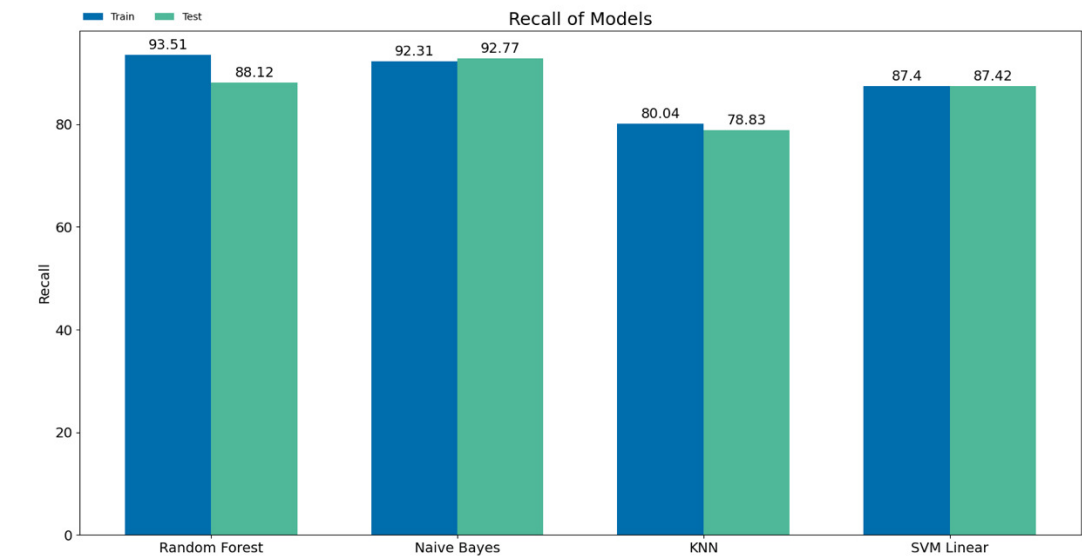


Figure 30: Recall (JavaScript code)

In this data set, the recall of random forest model in the classification of training set is the highest among the four models, which is 93.51, while the recall of test set is reduced to 88.12%. Naive Bayes performs well in the classification of test sets, which is 92.77%, and its classification of test sets and training sets is relatively stable in recall.

**F1 score (Comprehensive features dataset)**

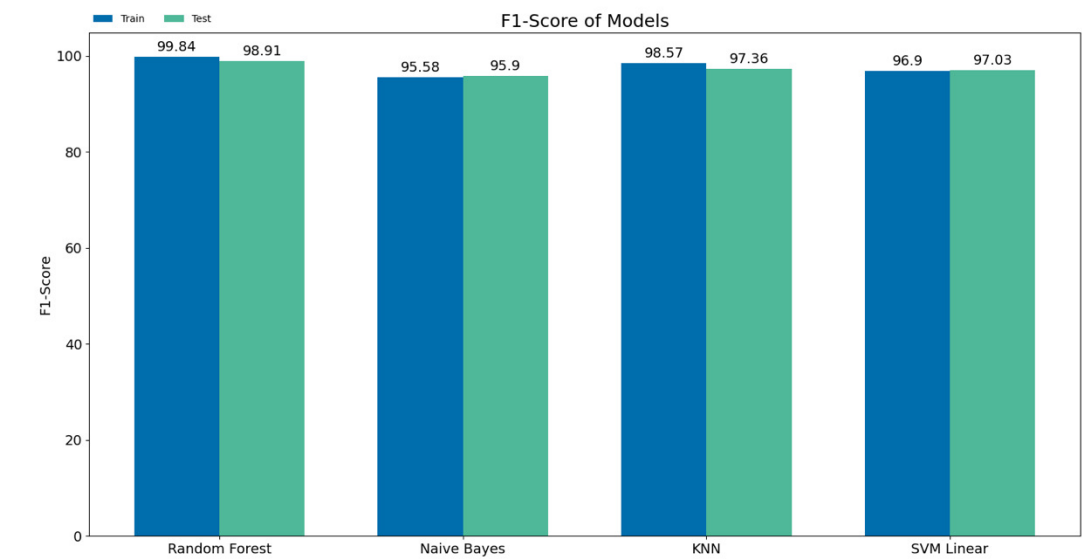


Figure 31: F1 score (Comprehensive features)

F1-score is used to balance precision and recall. It is widely used to compare the performance of several different models [50]. As can be seen from Fig. 31, the F1 scores of random forest for both training set and test set are the highest compared with other models, which are 99.84% and 98.91% respectively.

**F1 score (JavaScript code dataset)**

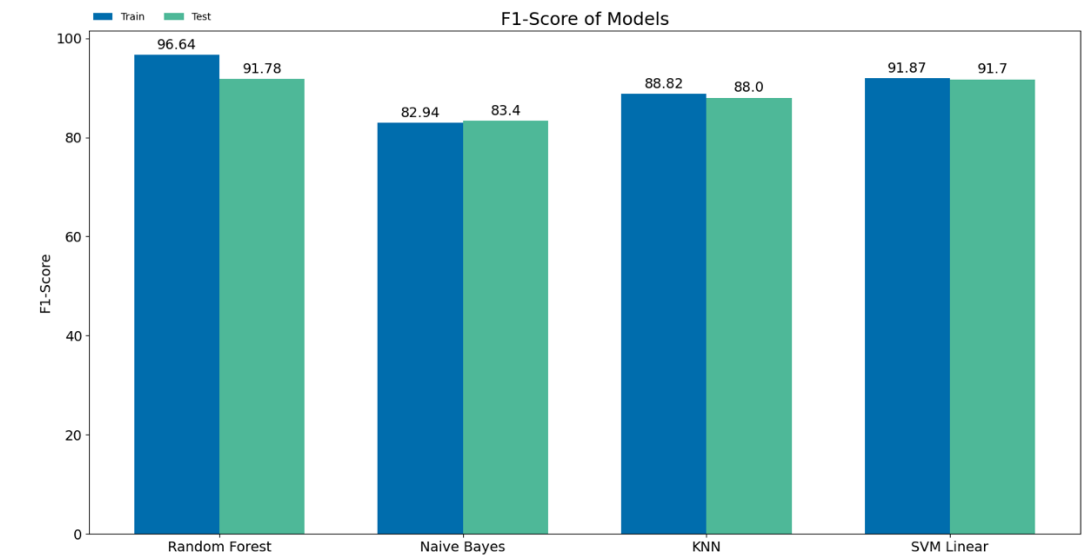


Figure 32: F1 score (JavaScript code)

When using JavaScript code dataset, the F1 score of random forest classifier is still the best among the four classifiers. The F1 score in the training set is 96.64%, and the performance in the training set decreases, with f1score of 91.78%.

**Comparison of machine learning classifiers**

It can be seen from the above data that the classifier trained by using the dataset of comprehensive features has a good performance in identifying malicious websites. In this training mode, random forest performs well. A higher accuracy indicates that it can classify websites well in most cases, and a high recall indicates that it has a relatively high accuracy in identifying malicious websites. Although Naive Bayes achieves 100% performance in precision, this means that it may miss some malicious web pages. In general, the random forest model can identify malicious websites relatively well.



## 4.2 Crawler module

There are three main objects to be realized in this module:

1. Automatically traverse all pages under the web application.
2. Save and extract some features of web pages, which are needed by machine learning classifier
3. The extracted data should be processed and formatted into the data set for training and testing machine learning above.

The main Python code of crawler function in this module are as follows.

```
def crawler():
    session = requests.Session()
    start_html = session.get(uri)
    soup = BeautifulSoup(start_html.content, features='lxml')
    all_href = soup.find_all('a')
    all_href = [l['href'] for l in all_href]
    uri_list = []
    for idx, href in enumerate(all_href):
        try:
            print(f">>> Collecting web pages data. >>> {round(idx / len(all_href) * 100, 1)}% >>>")
            r = session.get(uri + '/' + href)
            url_len = len(uri + '/' + href)
            soup = BeautifulSoup(r.content, features='lxml')
            path_depth = len(href.split("/"))
            all_js = soup.find_all('script')
            js_str = getJS(all_js)
            js_obf = getObfJS(all_js)
            js_len = len(js_str)/1000
            js_obf_len = len(js_obf)/1000
            uri_list.append({'path_depth':path_depth, 'geo_loc':getHostLocation(),
                            'url_len':url_len, 'js_len':js_len, 'js_obf_len':js_obf_len,
                            'js_str':js_str, 'url': uri + '/' + href,
                            'tld':getTld(), 'who_is':getWhoIs(), 'https':getIfHttps()})
            time.sleep(2)
        except:
            print("fail to connect target.")
    print(uri_list)
    return uri_list
```

Figure 33: Code block of crawler

In this function, the requests library is used to complete the task of sending and receiving requests for web pages. Compared with urllib2 module, using requests can make the code more concise. It supports HTTP connection retention and connection pooling, session retention using cookies, and encoding of automatic response content. The main idea to implement crawler is to use beautiful loop to find all <a> tags in the web page, because a tag is often used to render the navigation bar and link to other pages of the target web application. After finding the <a> tag, filter out the target with href attribute, and you can traverse the website according to the links found in href.

The code of data preprocessing function is as below:

```
def preProcessing(rawData):
    geo_loc = []
    who_is = []
    tld = []
    https = []
    url_len = []
    path_depth = []
    js_len = []
    js_obf_len = []
    for idx, ele in enumerate(rawData):
        geo_loc.append(ele["geo_loc"])
        who_is.append(ele["who_is"])
        tld.append(ele["tld"])
        https.append(ele["https"])
        url_len.append(ele["url_len"])
        path_depth.append(ele["path_depth"])
        js_len.append(ele["js_len"])
        js_obf_len.append(ele["js_obf_len"])
    web_data = pd.DataFrame({"geo_loc":geo_loc, "who_is":who_is, "tld":tld, "https":https,
                           "url_len":url_len, "path_depth":path_depth, "js_len":js_len, "js_obf_len":js_obf_len})
    return web_data
```

Figure 34: Code block of data preprocessing

The implementation of this part is relatively simple. It mainly standardizes each row of data by traversing the data list returned by the crawler, and finally converts the data into the format of DataFrame through the DataFrame() function of pandas. Then, the data could be submitted to the classifier module (Figure 35).

```
f = open("randomForests_under.model", "rb")          # read trained model file
classifier = pickle.load(f)                          # load trained model as classifier
f.close()                                            # close file

prediction_result = classifier.predict(dataset)       # save the prediction result
```

Figure 35: Code block of load classifier

After the classifier completes the identification of malicious web pages, the program will call the function `extractVulnerableURL(predict_list, raw_data)` to map the URL of the predicted malicious web pages according to the corresponding relationship of the number of rows, which is used as the target of the vulnerability exploitation module.

```
# extract URL
def extractVulnerableURL(predict_list, raw_data):
    target_list = []
    for idx, ele in enumerate(predict_list):
        if ele == 0:
            target_list.append(raw_data[idx]['url'])
    if(len(target_list) > 0):
        return target_list
    else: return 0
```

# collect url of the page which is predicted as malicious  
# return url list if there are malicious pages  
# return 0 if there is no malicious pages

Figure 36: Code block of extract malicious URL

### 4.3 Vulnerability scanning module

In this module, the main function should be scanning and exploiting the web pages which are predicted as malicious by the machine learning classifier. In order to realize the function above, this project uses the ZAP API for Python to call the

corresponding methods to verify the vulnerabilities. The main Python code are as follows:

```
class ZAP:
    target = ''
    apikey = ''
    zap = ZAPv2(apikey=apikey, proxies={'http': 'http://127.0.0.1:8080', 'https': 'http://127.0.0.1:8080'})

    def __init__(self, target, apikey, proxies_http, proxies_https):
        self.target = target
        self.apikey = apikey
        self.zap = ZAPv2(apikey=apikey, proxies={'http':proxies_http, 'https':proxies_https})
```

Figure 37: Code block of ZAP Class

ZAP is defined as a class with 3 main member variables: *target*, *apikey* and *zap*. The *target* is the url of web page which may be malicious, *apikey* should be a string which used to match the ZAP application installed. It could be found at the menu of ZAP: “ *Preferences / Options / API / API key* ”. *zap* is a ZAPv2 object which is used to call ZAP API functions.

```
def spider(self):
    print('Spidering target {}'.format(self.target))
    scanID = self.zap.spider.scan(self.target)
    while int(self.zap.spider.status(scanID)) < 100:
        print('Spider progress %: {}'.format(self.zap.spider.status(scanID)))
        time.sleep(1)
    print('Spider has completed!')
    print('\n'.join(map(str, self.zap.spider.results(scanID))))
```

# ZAP spider  
# start the scanning task  
# set a condition to avoid endless loop  
# print the progress of spider  
# set a pending time  
# join an element as exploitation target

Figure 38: Code block of vulnerability scanning

The method *spider()* above (Figure 37) is used to launch web page scan on specific target by ZAP. It will send various http request to the target to find potential injection point.

```
def attack(self):
    print('Active Scanning target {}'.format(self.target))
    scanID = self.zap.ascan.scan(self.target)
    while int(self.zap.ascan.status(scanID)) < 100:
        print('Scan progress %: {}'.format(self.zap.ascan.status(scanID)))
        time.sleep(5)
    print('Active Scan completed')
    print('Hosts: {}'.format(', '.join(self.zap.core.hosts)))
    print('Alerts: ')
    pprint(self.zap.core.alerts(baseUrl=self.target))
```

# ZAP attacker  
# start vulnerability scanning  
# set a condition to avoid endless loop  
# print the progress of spider  
# set a pending time  
# print the result of exploitation

Figure 39: Code block of vulnerability exploiting

The method *attack()* above is used to verify vulnerability by trying different kind of attack with corresponding payloads.

## Chapter 5

### System testing and comparative analysis

In order to ensure that the program can run correctly and meet the requirements, testing the program is a necessary link. In this project, the functions in three modules (crawler module, classifier module and expansion module) are tested respectively, and finally the process test is carried out.

#### 5.1 Crawler module test

Test Case ID: 1	
Description: To verify whether the crawler module can traverse the web page normally, extract the web page information and format it.	
Test Steps	1. Add three sets of <i>print()</i> functions to the code of crawler module to print the processing results of Http Request Methods, Data Extract Methods and Data Process Methods respectively in terminal.
	2. Run the crawler program at the terminal.
Expected Result	1. The crawler can send network requests normally.
	2. It can show the progress of the crawler.
	3. Successfully complete the web page crawling and output the data in the correct format.
Pass / Fail	Pass

Table 3: Test case 1

#### 5.2 Classifier module test

Test Case ID: 2	
Description: To ensure that data preprocessing can successfully convert data to the specific format required.	
Test Steps	1. Transform training set
	2. Transform test set
	3. At the same time, traverse the transformed training set and test set, and compare the coded lines to ensure that the coding rules are the same.

Expected Result	1. The training set and test set were successfully converted to a specific format
	2. The coding rules of the lines to be coded in the training set and the test set are consistent.
Pass / Fail	Pass

Table 4: Test case 2

Test Case ID: 3	
Description: Ensure that the machine learning training can operate normally.	
Test Steps	1. Train random forest model.
	2. Train Naïve Bayes model.
	3. Train KNN model.
	4. Train SVM model.
	5. Run test set on random forest model.
	6. Run test set on Naïve Bayes model.
	7. Run test set on KNN model.
	8. Run test set on SVM model.
Expected Result	1. No error during training.
	2. The test set samples can be predicted normally.
Pass / Fail	Pass

Table 5: Test case 3

Test Case ID: 4	
Description: To ensure that the evaluation method and the trained classifier works properly.	
Test Steps	1. Run precision evaluation on 4 classifiers.
	2. Run accuracy evaluation on 4 classifiers.
	3. Run recall evaluation on 4 classifiers.
	4. Run F1 score evaluation on 4 classifiers.
	5. Run confusion matrix evaluation on 4 classifiers.
Expected Result	1. The machine learning model can run successfully
	2. Precision evaluation, accuracy evaluation, recall evaluation, F1 score evaluation and confusion matrix evaluation can run normally and generate charts.
Pass / Fail	Pass

Table 6: Test case 4

Test Case ID: 5	
Description: Ensure that the data processing module can match the URL to the sample according to the prediction results.	
Test Steps	1. Extract 50 pieces of data from the test set, use random forest classifier to predict the label, and output the results.
	2. Run the data processing function to match the prediction results with the test data and output the results.
Expected Result	1. The classifier operates normally.
	2. The program can successfully match the predicted label and the URL of the sample.
Pass / Fail	Pass

Table 7: Test case 5

### 5.3 Vulnerability scanning module test

Test Case ID: 6	
Description: To ensure that the vulnerability scanning module can operate normally	
Test Steps	1. Run OWASP ZAP
	2. Run vulnerability scanning python program.
Expected Result	1. The program can run normally and successfully establish a connection with OWASP ZAP.
	2. Scanning and exploiting functions can run normally.
Pass / Fail	Pass

Table 8: Test case 6

### 5.4 Integration test

Test Case ID: 7	
Description: In order to verify that each module can cooperate normally, the whole program can run normally.	
Test Steps	1. Run the main program.
Expected Result	1. There is no unexpected error.
	2. The console can print results normally.
Pass / Fail	Pass

Table 9: Test case 7

### 5.5 Test result

After unit test and integration test, the results show that the program can run normally and complete the planned tasks. In the test, the crawler module can

successfully send requests and receive feedback to the web page entered by the user. The information extraction function can run normally, traverse the web application and extract the relevant information of the web page, and finally convert these information into the specified format. The training method in the machine learning module can run normally, and can generate a classifier to identify malicious web pages by training according to the training set. The evaluation function is available. After using the test set to test and evaluate these classifiers, it can be found that random forest can accurately predict malicious web pages, and its accuracy can reach 98.92%. The vulnerability scanning and verification module can also run normally. For several simple web pages used in the test, the scanning module has successfully scanned and verified them.

## **5.6 Comparative analysis**

The innovation of this project is to combine malicious web page detection and penetration testing based on machine learning. By judging the pages that have been maliciously tampered with in the whole web application, we can scan and these pages first, so as to quickly find the location of the problem. In this project, the comprehensive feature training machine learning model is relatively good for the accuracy of malicious website detection, and the precision can reach 99.85%. Compared with other similar studies, Aldwairi and Alsalman extracted the features in the URL to train the model in their project. The precision of malicious website recognition is 87% [35]. In Yan and Xu's research, based on the feature analysis of malicious code embedded in the URL, the precision of recognition is about 90% [51]. Xuan, Dinh and victor in the research of using URL features to detect malicious web pages, the random forest model of 100 decision trees achieved a precision of 98.75% [52]. Vinayakumar, Soman and Poornachandran compared several deep learning URL detection methods, of which the best precision is 98.88% [53]. From the comparison, it can be concluded that the model of this project can be better suitable for finding malicious tampered web pages, which can help the penetration test become more targeted for finding problems.

## **Chapter 6**

### **Limitation and future work**

#### **6.1 Limitation**

First of all, due to the limited time, the implementation of this project is still relatively basic for the application of machine learning in penetration testing, so the goal is also more targeted. It mainly focuses on screening the objectives of penetration testing by detecting potential malicious code in web pages, The purpose is to help web application owners quickly find out the vulnerabilities in the tampered web pages, so as to restore the web pages to normal as soon as possible and prevent further losses. According to the test results, this work is meaningful. It can accurately screen the samples of malicious web pages from the test set. Second, due to the limited hardware conditions, the problem of memory overflow was encountered in the process of machine learning model training of this project. Finally, the way of under sampling was used to randomly screen the samples.

#### **6.2 Future works**

In the future work, the following points can be improved on the research of this project:

1. Try to use the technology related to deep learning to mine the vulnerabilities of web applications. Compared with the traditional machine learning which needs to manually extract the features from the training data set, deep learning can extract the features by itself, which has stronger learning ability and even surpasses human performance in some tasks.
2. Use natural language processing combined with deep learning to synthesize a variety of data in the sample, conduct semantic analysis, and detect potential vulnerabilities in a way similar to code review.



3. In the process of data preprocessing, make more detailed classification of samples, such as distinguishing the types of malicious code and vulnerabilities in web pages and the threat degree of vulnerabilities.
4. Improve the code structure, reduce the coupling between different modules of the program, and make the work of data preprocessing more flexible.
5. Create a crawler to obtain more targeted information for vulnerability mining, collect the data of web applications with potential vulnerabilities that have not been injected with malicious code or tampered with, and turn the work goal to find vulnerabilities before web applications are attacked.

## Chapter 7

### Conclusion

This project aims to create a program to quickly judge whether a website may have been implanted with malicious code based on machine learning, and then give priority to the vulnerability scanning and verification of the tampered web pages according to the judgment results, so as to help testers locate vulnerabilities in a short time. In order to achieve the above objectives, three main parts are created in this program: crawler module, machine learning module and vulnerability scanning verification module. First, the main function of the crawler module is to collect the necessary information contained in the target web page, and preprocess it after the collection, so as to meet the sample format supported by the machine learning prediction model. Second, in the machine learning module, firstly, the data set containing comprehensive features is used to train and test the four machine learning models, and then the data set containing JavaScript code is used to train and test the four machine learning models again. Finally, a total of eight classifiers are created and their performance is evaluated. Thirdly, the vulnerability scanning verification module mainly realizes the function of scanning specific web pages. After testing, these three parts can complete the expected tasks well. The main result of this study is that the created program can give priority to scanning the web pages predicted by the classifier as likely to have been attacked, which can greatly shorten the time to find problems and improve the efficiency of penetration testing to a certain extent. The second achievement is that after comparison, it is found that the random forest classifier trained with the data set containing comprehensive features has the best performance for predicting malicious web pages. It performs well in the test set with the ratio of positive and negative samples of 1:1, and the recognition rate of web pages inserted with malicious code can reach 98.92%.

## Chapter 8

### Reflection

This project has improved me in both academic and technical aspects. The whole research process is a very meaningful experience for me. There are many valuable lessons in the main stages of this research process. In the first stage, that is, the stage of investigating and learning the relevant research of the project, I have done a lot of investigation on the application of machine learning in the field of network security at this stage. By consulting a large number of relevant academic papers, I have a deeper understanding of the intersection of these two disciplines and have a lot of new ideas for the research of this project. Through the study of these papers, I also have a clearer understanding of the general process and standards of academic research, which will be a valuable asset on my academic road in the future. The second stage is to analyze my project objectives and seek technical methods to meet the target requirements. This stage is a great challenge for me, and it also gives me a lot of very useful research experience. Because my previous work did not involve machine learning related experience, and my understanding of penetration testing is relatively limited, how to combine these two unfamiliar fields is a difficult but worthwhile task. Because of my limited understanding of machine learning in the early stage, I tried some inaccurate or unrealistic directions when looking for solutions. For example, I excessively pursue the use of reinforcement learning related solutions, but it is unrealistic for my current knowledge reserve to use it in my project in a limited time. At this stage, the summary and guidance on my project objectives offered by my supervisor made me gradually find the right direction. Finally, I chose the traditional machine learning scheme to be applied to this project, and made some attempts in common methods, such as using sample features different from which are used in related research to train the machine learning model. In addition, in the process of exploring the scheme, I also learned various performance evaluation methods for machine learning. The third stage, that is, the stage of project implementation and testing, is relatively smooth at this stage because I have been engaged in software development before. During this stage, I have learned a lot of academic and technical experience and lessons. For example, in terms of learning, I have learned

relevant methods of data preprocessing, such as data cleaning, unbalanced sample processing, etc; Methods and basic processes related to machine learning model training and performance evaluation, such as cross validation of machine learning model to obtain the model parameter value with the best performance; In terms of technology, I have gained some experience in error and debugging in the process of development and testing, which is very valuable for future learning and research. In addition, another attempt I think is more useful is that I added process visualization code to the model training program, because the training time of machine learning model is sometimes long. By visualizing the process, the step of the program can be intuitively shown, so as to know whether it is running normally or crashing. The last stage is the writing stage of the thesis. At this stage, I learned the lesson that the problems and achievements arising from the research process should be recorded in time, which will facilitate the sorting work in the future. Because I did not save the results in the process of implementation, I had to run the program again and record their results, which took a lot of time. The above are my reflections on the project. By summarizing the experience of the project, I have deepened my understanding of the project, clarified my strengths and weaknesses, and stimulated my enthusiasm for academic research. I will make full use of these favorable experiences and correct my shortcomings, Invest in the research on the cross field of machine learning and network security with a more professional attitude.

## References

- [1] "State of the Web Security 2020 - CDNetworks", *CDNetworks*, 2021. [Online]. Available: <https://www.cdnetworks.com/news/state-of-the-web-security-2020/>. [Accessed: 28- Oct- 2021].
- [2] "New Zealand stock exchange halted by cyber-attack", *BBC News*, 2021. [Online]. Available: <https://www.bbc.com/news/53918580>. [Accessed: 25- Oct- 2021].
- [3] "Dyn says cyberattack 'has been resolved' after services shut down", *NBC News*, 2021. [Online]. Available: <https://www.nbcnews.com/tech/tech-news/dyn-says-cyberattack-resolved-after-services-shut-down-n670926>. [Accessed: 25- Oct- 2021].
- [4] "British Airways fined £20m over data breach", *BBC News*, 2021. [Online]. Available: <https://www.bbc.co.uk/news/technology-54568784>. [Accessed: 25- Oct- 2021].
- [5] "British Airways breach: How did hackers get in?", *BBC News*, 2021. [Online]. Available: <https://www.bbc.com/news/technology-45446529>. [Accessed: 25- Oct- 2021].
- [6] "eBay under pressure as hacks continue", *BBC News*, 2021. [Online]. Available: <https://www.bbc.com/news/technology-29310042>. [Accessed: 25- Oct- 2021].
- [7] "Agent Tesla Panel Remote Code Execution", *Rapid7*, 2021. [Online]. Available: [https://www.rapid7.com/db/modules/exploit/multi/http/agent\\_tesla\\_panel\\_rce/](https://www.rapid7.com/db/modules/exploit/multi/http/agent_tesla_panel_rce/). [Accessed: 28- Oct- 2021].
- [8] "Cisco Data Center Network Manager SQL Injection Vulnerability", *Tools.cisco.com*, 2021. [Online]. Available: <https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-dcnm-sql-inject-8hk6PwmF>. [Accessed: 28- Oct- 2021].
- [9] G. Weidman, *Penetration testing*, 1st ed. San Francisco: No Starch Press, 2014. pp. 2-6, 317-337.
- [10] T. Thomas, A. Vijayaraghavan and S. Emmanuel, *Machine learning approaches in cyber security analytics*. pp. 37-47.
- [11] E. Machines, T. Author(s) and C. Apress, "Efficient Learning Machines | SpringerLink", *Apress.com*, 2021. [Online]. Available: <https://www.apress.com/gp/book/9781430259893>. [Accessed: 30- Oct- 2021].
- [12] M. Mohri, A. Rostamizadeh and A. Talwalkar, *Foundations of machine learning*. pp. 2-13.
- [13] A. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE", *Mind*, vol., no. 236, pp. 433-460, 1950. Available: 10.1093/mind/lix.236.433.

- [14] S. Raschka and V. Mirjalili, *Python Machine Learning - Third Edition*. pp. 2-15.
- [15] N. Nilsson, "INTRODUCTION TO MACHINE LEARNING", *Robotics.stanford.edu*, 2021. [Online]. Available: <http://robotics.stanford.edu/~nilsson/MLBOOK.pdf>. [Accessed: 28- Oct- 2021].
- [16] G. Biau and E. Scornet, "A random forest guided tour", *TEST*, vol. 25, no. 2, pp. 197-227, 2016. Available: 10.1007/s11749-016-0481-7.
- [17] S. Raschka, "Naive Bayes and Text Classification I - Introduction and Theory", *arXiv.org*, 2021. [Online]. Available: <https://arxiv.org/abs/1410.5329>. [Accessed: 31- Oct- 2021].
- [18] W. Noble, "What is a support vector machine?", *Nature Biotechnology*, vol. 24, no. 12, pp. 1565-1567, 2006. Available: 10.1038/nbt1206-1565.
- [19] D. Meyer, F. Leisch and K. Hornik, "The support vector machine under test", *Neurocomputing*, vol. 55, no. 1-2, pp. 169-186, 2003. Available: 10.1016/s0925-2312(03)00431-4.
- [20] P. Liang, W. Li and J. Hu, "Fast SVM training using data reconstruction for classification of very large datasets", *IEEE Transactions on Electrical and Electronic Engineering*, vol. 15, no. 3, pp. 372-381, 2019. Available: 10.1002/tee.23065.
- [21] A. Abdiansah and R. Wardoyo, "Time Complexity Analysis of Support Vector Machines (SVM) in LibSVM", *International Journal of Computer Applications*, vol. 128, no. 3, pp. 28-34, 2015. Available: 10.5120/ijca2015906480.
- [22] "requests", *PyPI*, 2021. [Online]. Available: <https://pypi.org/project/requests/>. [Accessed: 28- Oct- 2021].
- [23] "beautifulsoup4", *PyPI*, 2021. [Online]. Available: <https://pypi.org/project/beautifulsoup4/>. [Accessed: 28- Oct- 2021].
- [24] "pa-whois", *PyPI*, 2021. [Online]. Available: <https://pypi.org/project/pa-whois/>. [Accessed: 28- Oct- 2021].
- [25] "geoip2", *PyPI*, 2021. [Online]. Available: <https://pypi.org/project/geoip2/>. [Accessed: 28- Oct- 2021].
- [26] "tld", *PyPI*, 2021. [Online]. Available: <https://pypi.org/project/tld/>. [Accessed: 28- Oct- 2021].
- [27] C. Harris et al., "Array programming with NumPy", *Nature*, 2021. [Online]. Available: <https://www.nature.com/articles/s41586-020-2649-2>. [Accessed: 31- Oct- 2021].
- [28] W. McKinney, "pandas: a Foundational Python Library for Data Analysis and Statistics", *ResearchGate*, 2021. [Online]. Available:

- [https://www.researchgate.net/publication/265194455\\_pandas\\_a\\_Foundational\\_Python\\_Library\\_for\\_Data\\_Analysis\\_and\\_Statistics](https://www.researchgate.net/publication/265194455_pandas_a_Foundational_Python_Library_for_Data_Analysis_and_Statistics). [Accessed: 31- Oct- 2021].
- [29] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python", Jmlr.org, 2021. [Online]. Available: <https://jmlr.org/papers/v12/pedregosa11a.html>. [Accessed: 31- Oct- 2021].
- [30] [49]W. McKinney, "Data Structures for Statistical Computing in Python", *ResearchGate*, 2021. [Online]. Available: [https://www.researchgate.net/publication/265001241\\_Data\\_Structures\\_for\\_Statistical\\_Computing\\_in\\_Python](https://www.researchgate.net/publication/265001241_Data_Structures_for_Statistical_Computing_in_Python). [Accessed: 31- Oct- 2021].
- [31] "imbalanced-learn", *PyPI*, 2021. [Online]. Available: <https://pypi.org/project/imbalanced-learn/>. [Accessed: 28- Oct- 2021].
- [32] H. Abdullah, "Evaluation of Open Source Web Application Vulnerability Scanners", *Academic Journal of Nawroz University*, vol. 9, no. 1, p. 47, 2020. Available: 10.25007/ajnu.v9n1a532.
- [33] J. Fan, P. Gao, C. Shi and N. Li, "Research on Combine White-Box Testing and Black-Box Testing of Web Applications Security", *Advanced Materials Research*, vol. 989-994, pp. 4542-4546, 2014. Available: 10.4028/www.scientific.net/amr.989-994.4542.
- [34] M. Khalid, H. Farooq, M. Iqbal, M. Alam and K. Rasheed, "Predicting Web Vulnerabilities in Web Applications Based on Machine Learning", *Communications in Computer and Information Science*, pp. 473-484, 2019. Available: 10.1007/978-981-13-6052-7\_41 [Accessed 31 October 2021].
- [35] M. Aldwairi and R. Als Salman, "MALURLS: A Lightweight Malicious Website Classification Based on URL Features", 2021. [Online]. Available: [https://www.researchgate.net/publication/237067589\\_MALURLS\\_A\\_lightweight\\_malicious\\_website\\_classification\\_based\\_on\\_URL\\_features](https://www.researchgate.net/publication/237067589_MALURLS_A_lightweight_malicious_website_classification_based_on_URL_features). [Accessed: 28- Oct- 2021].
- [36] S. Calzavara, M. Conti, R. Focardi, A. Rabitti and G. Tolomei, "Mitch: A Machine Learning Approach to the Black-Box Detection of CSRF Vulnerabilities", *leeexplore.ieee.org*, 2021. [Online]. Available: <https://leeexplore.ieee.org/document/8806728/>. [Accessed: 22- Oct- 2021].
- [37] W. Melicher, C. Fung, L. Bauer and L. Jia, "Towards a Lightweight, Hybrid Approach for Detecting DOM XSS Vulnerabilities with Machine Learning", *Users.ece.cmu.edu*, 2021. [Online]. Available: <https://users.ece.cmu.edu/~lbauer/papers/2021/www2021-dom-xss-dnn.pdf>. [Accessed: 28- Oct- 2021].
- [38] L. Shar, H. Tan and L. Briand, "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis | Proceedings of the 2013

- International Conference on Software Engineering", *DI.acm.org*, 2021. [Online]. Available: <https://dl.acm.org/doi/10.5555/2486788.2486873>. [Accessed: 28- Oct- 2021].
- [39] K. Kamtuo and C. Soomlek, "Machine Learning for SQL injection prevention on server-side scripting", *IEEE Explore. IEEE.org*, 2021. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7859950/>. [Accessed: 28- Oct- 2021].
- [40] M. Gupta, M. Govil and G. Singh, "Predicting Cross-Site Scripting (XSS) security vulnerabilities in web applications", *IEEE Explore. IEEE.org*, 2021. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7219789>. [Accessed: 28- Oct- 2021].
- [41] Y. Fang, C. Huang, Y. Su and Y. Qiu, "Detecting malicious JavaScript code based on semantic analysis", 2021. [Online]. Available: [https://www.researchgate.net/publication/339356123\\_Detecting\\_Malicious\\_JavaScript\\_Code\\_Based\\_on\\_Semantic\\_Analysis](https://www.researchgate.net/publication/339356123_Detecting_Malicious_JavaScript_Code_Based_on_Semantic_Analysis). [Accessed: 28- Oct- 2021].
- [42] W. Liu and S. Zhong, "Web malware spread modelling and optimal control strategies", *Scientific Reports*, vol. 7, no. 1, 2017. Available: 10.1038/srep42308.
- [43] K. Soska and N. Christin, "Automatically Detecting Vulnerable Websites Before They Turn Malicious", *Usenix.org*, 2021. [Online]. Available: <https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-soska.pdf>. [Accessed: 28- Oct- 2021].
- [44] B. Eshete, A. Villafiorita, K. Weldemariam and M. Zulkernine, "EINSPECT: Evolution-Guided Analysis and Detection of Malicious Web Pages," 2013 IEEE 37th Annual Computer Software and Applications Conference, 2013, pp. 375-380, doi: 10.1109/COMPSAC.2013.63.
- [45] A. Singh, "Malicious and Benign Webpages Dataset", 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2352340920311987>. [Accessed: 28- Oct- 2021].
- [46] A. Liu, "The Effect of Oversampling and Undersampling on Classifying Imbalanced Text Datasets", 2021. [Online]. Available: <https://www.semanticscholar.org/paper/The-Effect-of-Oversampling-and-Undersampling-on-Liu/cade435c88610820f073a0fb61b73dff8f006760>. [Accessed: 28- Oct- 2021].
- [47] S. Ndichu, S. Kim, S. Ozawa, T. Misu and K. Makishima, "A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors", *Applied Soft Computing*, vol. 84, p. 105721, 2019. Available: 10.1016/j.asoc.2019.105721.
- [48] Y. Zhang, R. Jin and Z. Zhou, "Understanding Bag-of-Words Model: A Statistical Framework", 2021. [Online]. Available:



- [https://www.researchgate.net/profile/Rong-Jin-4/publication/226525014\\_Understanding\\_bag-of-words\\_model\\_A\\_statistical\\_framework/links/554b968f0cf29752ee7cc15b/Understanding-bag-of-words-model-A-statistical-framework.pdf](https://www.researchgate.net/profile/Rong-Jin-4/publication/226525014_Understanding_bag-of-words_model_A_statistical_framework/links/554b968f0cf29752ee7cc15b/Understanding-bag-of-words-model-A-statistical-framework.pdf). [Accessed: 28- Oct- 2021].
- [49] B. Hsu, "Comparison of Supervised Classification Models on Textual Data", 2021. [Online]. Available: [https://www.researchgate.net/publication/341619075\\_Comparison\\_of\\_Supervised\\_Classification\\_Models\\_on\\_Textual\\_Data](https://www.researchgate.net/publication/341619075_Comparison_of_Supervised_Classification_Models_on_Textual_Data). [Accessed: 28- Oct- 2021].
- [50] D. Miyamoto, H. Hazeyama and Y. Kadobayashi, "An evaluation of machine learning-based methods for detection of phishing sites | Proceedings of the 15th international conference on Advances in neuro-information processing - Volume Part I", *DI.acm.org*, 2021. [Online]. Available: <https://dl.acm.org/doi/10.5555/1813488.1813559>. [Accessed: 29- Oct- 2021].
- [51] X. Yan, Y. Xu, B. Cui, S. Zhang, T. Guo and C. Li, "Learning URL Embedding for Malicious Website Detection", *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6673-6681, 2020. Available: 10.1109/tii.2020.2977886.
- [52] C. Xuan, H. Dinh and T. Victor, "Malicious URL Detection based on Machine Learning", *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 1, 2020. Available: 10.14569/ijacsa.2020.0110119.
- [53] R. Vinayakumar, K. Soman and P. Poornachandran, "Evaluating deep learning approaches to characterize and classify malicious URL's", *Journal of Intelligent & Fuzzy Systems*, vol. 34, no. 3, pp. 1333-1343, 2018. Available: 10.3233/jifs-169429.