Analysis of JavaScript-based Attack Classification Using Support Vector Machine Learning Approach

By

Samuel Daba Sefissa

A Thesis Submitted to the Department of Computer Science and Engineering
School of Electrical Engineering and Computer Science

Presented in Partial Fulfilment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

February, 2021

Adama, Ethiopia

Analysis of JavaScript-based Attack Classification Using Support Vector Machine Learning Approach

By

Samuel Daba Sefissa

Advisor

Dr. Bahiru Shifaw

A Thesis Submitted to the Department of Computer Science and Engineering
School of Electrical Engineering and Computer Science

Presented in Partial Fulfilment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

February, 2021

Adama, Ethiopia

Declaration

I hereby declare that this Master Thesis is my original work and has not been presented for a degree in any other university, and all sources of material used for this thesis have been duly acknowledged.

Name: Samuel Daba Sefissa

Signature: _____

This MSc Thesis has been submitted for defense with my approval as a thesis advisor.

Name: Bahiru Shifaw(PhD)

Signature: _____

Acknowledgment

I would like to acknowledge God for his grace, strength, and good health as I undertook this research. My sincere gratitude to my advisor Dr. Bahiru Shifaw for his friendly approach, readiness and willingness to advise on the research, commitment to guide, and support greatly improved this thesis work.

Besides my advisor, I would like to thank Mr. Duresa Tamirat (Ph.D. Candidates at ASTU) for his constant help and he is always available to discuss and provide insightful comments on my work throughout my program. It was a pleasure working with him and also a wonderful learning experience. Also, my sincere thanks to Mr. Tafera Banti(PhD. Candidate at ASTU).

Last but not the least, I would like to thank my family: my parents Daba Sefissa and Chaltu Buta, for giving life to me in the first place and supporting me morally throughout my life and my elder brother Terecha, for always standing and guiding me in my academics and life in general and thanks to my classmates and friends for their support and love.

Table of Contents

List of Figures

List of table

List of Abbreviations and Acronyms

AST	Abstract Syntax Tree
CFG	Control Flow Graph
CNN	Convolutional Neural Networks
TF-IDF	Term Frequency — Inverse Document Frequency
SVM	Support Vector Machine
XSS	Cross-site scripting
DLL	Dynamic Link Library
JSON	JavaScript Object Notation
ML	Machine Learning
UFO	Unidentified Flying Objects
KNN	k-nearest neighbors
NB	Naïve Bayes
LR	Logistic Regression
DT	Decision Trees
GR	Gain ratio
HMM	Hidden Markov Model
IDS	Intrusion Detection Systems
KDD	Knowledge Discovery Dataset
CBoW	Continuous Bag-ofWords
APIv3	Application Programming Interface version three
CV	Cross-Validation
NAT	Network address translation
ROC	receiver operating characteristic curve
V8	JavaScript Engine

Abstract

A web application is a software or computer program which utilizes the web browser. JavaScript is an object-oriented scripting language that has experienced wide adoption as a client-side scripting language for web pages and makes websites dynamic and interactive which leading the less server interaction, immediate feedback as while as rich interfaces and it also provides a platform for attacks. Nowadays, JavaScript-based attack classification has essential significance, due to the high growth of Internet users. Javascript based attackers many times use obfuscation techniques to conceal their malicious content to avoid detection. And also, the dynamic nature and the syntax of JavaScript further increase the complexity of analysis.

The goal of this study is to design the analysis of javascript-based attack classification using the support vector machines. To address the current challenges of the analysis and feature extraction process for classification behavioural and statical analysis methods were used. The cuckoo sandbox was used to obtain the behavioural analysis for our datasets and an abstract syntax tree was used to acquire the static analysis. The output of both behaviour analysis and static analysis was text-based files. Hence, text-based feature extraction method such as TF-IDF, doc2vec, and word2vec has been used to extract the feature from the datasets. This experimental work was carried out using javascript dataset obtained from "javascript malware collection" and "Alexa" which contains malicious as well as benign javascript respectively. The dataset extracted is splited into 70% and 30% for training and testing respectively. Besides, K-fold cross-validation was used to evaluate the score mean as well as the accuracy of the SVM algorithm. Precision, Recall, and F1-score evaluation metrics are also used to measure the performance of the model. Although the prediction result of all the three feature extraction methods with the model is very high, support vector machine model with word2vec feature extraction has shown better performance in the classification of JavaScript-based attacks. Consequently, the accuracy of the support vector machine model with word2vec is 97.1 % to predict on the unseen test datasets and also has a low false-positive and false-negative rate.

Keywords: *Javascript, Machine learning, Support vector machine, Static analysis, Behavioural analysis, Javascript-based attack,*

CHAPTER ONE

1. Introduction

1.1.Background of the Study

A web application is a software or computer program which utilizes the web browser. It has an advantage in the world such as connect an individual to the internet. The web application can be used in society in different ways such as online tax filing, shopping, banking, social network, government office, and educational institution [1]. These technologies have consequently been adopted across various devices and browsers due to their innumerable advantages. However, the technologies to exploit vulnerabilities in servers, plugins, and other third-party systems have made websites an attractive arena for cyber-criminals such as phishing attacks, unwanted advertisements, malicious code, cyber warfare, and extortion.

As the number of users gradually increased using the world wide web, web pages have made JavaScript a popular attack for infecting user's machines with malware [2]. According to the report [3], there are over millions of victims attacked by JavaScript code attacks on the Internet per day.

JavaScript is an object-oriented scripting language that has experienced wide adoption as a client-side scripting language for web pages. JavaScript can be used to improve usability and extend functionality the web pages frequently use JavaScript. For example, it can help reduce the server-side load by conducting validation checks of filled forms before sending them to the webserver. Similarly, JavaScript is used in website development to change formatted data on the web page automatically, a linked page appears on a popup window, or to add some rollover effect. More than 95% of all websites use JavaScript for browser-side computation in Web applications [4].

It is a dynamic and lightweight scripting language used to add more functionality and enhance the user experience by allows the client-side script to interact with the user which makes websites dynamic and interactive which leading the less server interaction, immediate feedback, and rich interfaces [5]. Still, it also provides a platform for attacks, such as a drive-by download,

cross-site script, phishing, and any other code injection attack. Different security researchers, locating these JavaScript-based attacks, normally much more difficult than expected. For this reason, the attackers many times use obfuscation techniques to conceal their malicious content to avoid the detection of antivirus software. And also, the dynamic nature and flexible syntax of JavaScript further increase the complexity of analysis.

In contrast to other types of network attack, JavaScript-based attack particularly hard to detect [6], for example, JavaScript attack regular analysis of the browser environment, check for particular vulnerability and use dynamic exploiting techniques such as heap spraying, for computing a victim system. In addition, many of an attacker hide malicious attack in his code, and the code is immediately interpreted by the user's browser without pre-compilation, which means that without converting the script code into system-related machine code. Therefore, it is important to accurately detect JavaScript-based code attacks to proactively defend against JavaScript code attacks.

There is a different attack that uses the JS code as a bridge to attack the user. Some of them are: driven by download, cross-site scripting, phishing, and malicious advertising. *Driven by download* a victim is lured to a malicious web page. The page contains code written in the JavaScript language that exploits vulnerabilities and launches the automatic download and installation of malware without the permission of the user [7]. *Phishing* is one of the socially engineered cybersecurity attacks where the attacker impersonates a genuine and legitimate website source and sends emails with the intention of stealing sensitive personal information. *The cross-site script* is a known web attack. It occurs when malicious script web code is executed from the browser on the victim's computer by using their web applications. The execution could filter the personal information like steal the user cookies to hijacking the identity in a fraudulent session, and also it offers the attackers the possibility of stealing sensitive data or even being able to take control of certain devices [8]. *Malicious advertising* is one of the attackers that exploit JavaScript is malvertising. In 2019 mal-advertisers switch from almost entirely targeting mobile devices to the majority targeting desktop browsers [9].

There is a different existing classification and detection approach used over a year to identify the malicious JavaScript from the normal JavaScript. The *traditional approach* has used a list of malicious signatures that are previous identified as malicious which is the database for

malicious code is created on the host compute and malicious file is updated regularly. The *heuristic approach* has used security experts to define the rule for benign behaviour or malicious behaviour. The *static and dynamic approach* has been widely used in malware detection. The static analysis uses the static characteristics i.e. the structure of the scripts to identify malicious scripts. Dynamic analysis can easily detect the new malware by observing its behaviours activities at runtime.

The *machine learning approach* that has been put forward the automatic detection of JavaScript. Different researchers have used such detection frameworks in their works such that Likarish et al. [10] and as well as the detectors in the works of Curtsinger et al [11]. Similarly, Stokes et al. [12] proposed a new deep learning model to process the byte sequence of detecting malicious JavaScript. In particular, Fang et al. [13] used the V8 engine to generate JavaScript bytecode and trained deep learning models based on features extracted from the bytecode. Moreover, Hao et al. [14] extracted API symbol features from the abstract syntax tree and used the Naive Bayes algorithm to detect malicious JavaScript. And also Fass et al. [15] also proposed a fully static analysis method. Liang et al. [16] argue that the feature extraction method was used to extract the features from JavaScript-based attacks by using an abstract syntax tree (AST) and a control flow graph (CFG).

In this study, we build a JavaScript-based attack classification using a Support Vector Machine learning algorithm. To simplify the aforementioned process of tedious and time-consuming feature extraction, we use abstract syntax trees to get semantic information for static analysis. And also, we use a cuckoo sandbox for behaviour analysis. The output of both static and behaviour analysis is easy to transform into vectors by introducing methods such as word2vec, Doc2vec, and TF-IDF feature method, which can extract features for datasets. Based on these features, we propose the use of SVM classification algorithms to single out malicious code from benign code. We conduct evaluation studies on datasets to show the effectiveness of our proposed scheme.

1.2.Motivation of the Study.

The website and web application have an important role in society, educational institutions, governmental organizations, and non-governmental organizations. The web application is used everywhere and involves sensitive and personal data. This makes them a target for malicious exploiting their vulnerability to obtain unauthorized access to stored data. In the protection of the website security threat, there are lots of tools to detect the website vulnerability of the computer software but this tool has the drawback to detect all and new attacks. Therefore, conducting this study to increase the effectiveness and efficiency of classifying the JavaScript-based attack from the web application security perceptive is crucial.

This work is motivated based on the issues: the challenges of the analysis method and the feature extraction method used for classification. To alleviate the feature extraction method challenge we used three different methods (word2vec, doc2vec, and tf-idf) and for the analysis method, we used behaviour analysis method to overcome the JavaScript-based attack classification problem. The problem of static analysis does not correctly classify the JavaScript attack because malicious JavaScript code used the obfuscation method to hide their structure.

Hence, statistical analysis and traditional methods (Signature method, Heuristic method) have experimented but they have limitations; not appropriate to classify JavaScript attacks and difficult to extract features using statistical analysis methods. We are motivated to propose a combination of static and behavioural analysis.

1.3.Statement of the Problem

While the Internet has donated remarkably to the human society by enabling communication and providing easier access to services and information, guarding users and organizations against attackers remains a top priority. The attackers can exercise an excess of strategies against unknowing victim users through attack path in which attacker can gain access to the victim users including; forced redirection of a user to a webpage containing malware that automatically downloads and installs itself, manipulation of a legitimate webpage intended to obtain private user information those makes the users vulnerable for the attacker, and social engineering-based attacks e.g. phishing is the attacker impersonates a genuine and legitimate website source to

steal sensitive personal information, and pretexting is an attacker tries to convince a victim to give up valuable information or access to a service or system.

The JavaScript-based attackers often use obfuscation techniques to hide their malicious intent to evade the detection which made malicious JavaScript is usually much more difficult than expected. Besides, the dynamic nature and flexible syntax of JavaScript further increase the complexity of analysis. Javascript-based attackers dynamically generate attack instructions during script execution, which makes traditional static analysis techniques fail to perform well. Therefore, JavaScript attack detection is very difficult due to the complexity and flexibility of JavaScript in the client browser.

The other problem most antivirus software has been used signature-based detection which is based on the pattern matching that is commonly found in malicious scripts. The signature of the malware program is stored on the database. When scans the system, it finds programs having the signatures in the malware database. The advantage of such an approach is speed and accuracy but, its disadvantage of using such a detection method is only to compare the malware in the database. Any unknown signature that does not exist in the database is not detected. And again, the database is needed regularly update. Therefore, traditional approaches using the signature to detect malicious program fails for the new and unknown malware case, where signatures are not available. Another approach is the heuristic detecting approach, which is based on rules established by security experts who define benign behaviour or malicious behaviour to search for malicious code. However, its major drawback is that it can only detect known malicious code but cannot detect an unknown malicious code and slow detection rate due to lengthy analysis and scanning time.

In addition to the above, most researchers used features for their works. For example, *Dharmaraj R. Patil et al*. [17] has been used 77 JavaScript features from the script for detection purpose. *Likarishet et al*. [10] manually selected 65 malicious JavaScript features through statistical analysis of a large number of samples, and then test the effectiveness of these features. *Lwin Khin et al.* [18] have been used 32 features by using hybrid analysis for extract attributes. However, manual feature selection relies on the expertise and extensive experience which means cannot detect new malware variant. Again, identifying malicious JavaScript from benign for the expert by itself is an error-prone and time-consuming task.

Most researchers used the static analysis method for classifies malicious JavaScript code from benign since malicious JavaScript code used the obfuscation method to hide their structure. Besides JavaScript-based attacks creates the code at run time. In the case of dynamic analysis methods, irrelevant features are used for the classification of the attack. For example, [19] argues that a dynamic analysis method but uses the execution time only of the source code which is difficult to differentiate the malicious from a benign one.

So, the purpose of this research is to overcome the above problem by proposing the static and behavioural analysis of JavaScript code with different feature extraction methods and develop the machine learning model with a lowering rate of false-positive to increase the accuracy of the model.

1.4.Research Questions

In this study, to attain the objectives of the study effective attempt will be made to answer the following basic research questions.

RQ1: What are the analysis methods used in the JavaScript-based attack classification?

RQ2: What are the feature extraction methods used in the classification?

RQ2: How to design and enhance a better approach for classifies JavaScript-based attacks?

1.5.The Objective of the Study

1.5.1. General Objective

The overall objective of this study carried out an analysis to develop the approach for the classification of JavaScript-based attacks using the Support Vector Machine learning algorithm.

1.5.2. Specific Objective

The following is a catalogue of work done to achieve the objective of the study.

- Review literature that has relation with the concept of JavaScript classification framework and identifies gap analysis.
- Implement statical analysis by using an abstract syntax tree.
- Implement behavioural analysis by using the cuckoo sandbox.
- Apply the feature extraction method to statical and behavioural analysis results.
- Design the proposed framework for JavaScript-based attack classification.
- Implement the proposed design framework.
- Evaluate the classification model classifier on datasets using evaluation metrics such as accuracy, precision, recall, and F1-score.

1.6.Scope and Limitation

1.6.1. Scope of the Study

The scope of this study was to provide a JavaScript-based attack classification by applied static and behavioural analysis. For classification purposes, we select a support vector machine algorithm. The study focuses on the static and behaviour analysis of JavaScript code and improvement of detection of the automatic JavaScript-based attack classification model. To train and test these models, feature extraction methods are utilized on the result of both static and dynamic to determine the most discriminant features for the collected dataset. The study implements support vector machine learning classifiers for the classification model and evaluates the result of each classifier based on the classification accuracy of the K-fold cross-validation technique.

1.6.2. Limitation of the Study

This study comes across with limitations on a different phase of the research process. The study cannot accommodate all security of attack vulnerabilities. It was limited only to JavaScript-based attacks. The tools used, the cuckoo sandbox in the study depends on the computer we used which is a better computer with more processes, storage, and RAM sizes are selected for tools. We implement on three different computers, consequently, three computers have different analysis times to complete the analyses and generate the reports. The challenge we faced in the study of the computer we have is not preferable for the tools that it takes a lot of time to complete the analysis and generate the reports.

1.7.Significance of the Study

The study extends the effort exerted in solving the security problem of the web application especially web browser users. The proposed framework has tremendous advantages for developers and end-users because this study aimed to create a very secure environment for the end-user. Software security tester, ethical hacker, and developer will be beneficiaries. Specifically, the benefits of this research are:

- Classify JavaScript-based attacks using support vector machine learning.
- Maintain a stable working environment especially for end-user of the web application.
- Minimizing the cost of detecting website security-related problems.
- It can also be used as a reference for researchers and website developers.
- Increase the availability of detection framework.
- It provides information to evaluate web application security risks

1.8.Organization of the Study

The research paper organized into seven chapters in the following ways:

Chapter one: as discussed above, and It includes the introduction of the study, the motivation, and the statement of problems, the research questions that would be answered in the proposed solutions, the scope and limitation, the objective of the study, the methodology, and the application results of the study.

Chapter two: presents about literature review. Overviews of JavaScript-based attack. The method used in the JavaScript-based attack classification. Abstract syntax tree and cuckoo sandbox concept explained in detail. Feature extraction method, the basic concept of machine learning used in classifier, evaluation method used for the model, and its application of machine learning in the security perspective discussed. Finally, it discussed the related work.

Chapter three: discusses the research methodology used in this research work, methods used to build the dataset, methods used to develop JavaScripe-based attack models, feature extraction methods used in the study, and machine learning classifier algorithm, and finally, it presents the evaluation methods.

Chapter four: discusses the proposed solution for JavaScript-based attack classification, which is the proposed preprocessing, feature extraction, machine learning training, and Evaluation.

Chapter five: discusses the implementation and experimentation of the proposed solution. Describes working Environments, programming language, and implementation of the proposed framework.

Chapter six: discusses the result of the dataset. feature extraction, building the model. Also discusses the major results obtained by comparing the models based on features. The performance evaluation is also discussed and a comparison with previous work is carried out. Finally, it presents a discussion

Chapter seven*:* consists of a conclusion, recommendation, and identifies future works for further research direction on this research topic

CHAPTER TWO

2. Related and Literature Review

2.1.Overviews of JavaScript.

JavaScript is a browser scripting language initially created to enhance the display of web sites on the client-side, to enhance the interactivity of web sites, and to improve their user-friendliness. JavaScript is the most common language that is chosen by web developers, and it is supported by all web browsers such as Google Chrome, Firefox, and Internet Explorer. *Bichhawat et al.* [4] say that more than 95% of Web sites choose the JavaScript language for Web front-end development. However, JavaScript can be abused for malicious purposes e.g, exploit the vulnerabilities of the browser to cause private data leakage, it can be used to engage in malicious activities such as drive-by download attacks, cross-site scripting, malicious advertisements, and spam messages on blogs or social network sites.

Drive-by download attacks is an attack that refers to automatically downloading malware on a computer stealthily, without the user's awareness. malicious advertising attacks are caused by a click. Malicious advertisements caused when a user clicks on a malicious advertisement link, he will receive different types of attack that contains the response message is injected with malicious code this kind of advertising is a malicious advertisement. Cross-site scripting is an attack against web applications in which scripting code is typically injected into the output of an application that is then sent to client users. This scripting code executes in the browser with the privileges of the originating site, thereby circumventing the same-origin policy of the browser.

JavaScript may be used to redirect a user to a website hosting malicious software, to create a window recommending users download a fake codec, to detect what software versions the user has installed and selected a compatible exploit, or to directly execute an exploit [10].

JavaScript attack is a program that is specifically designed to perform different functions, including gaining access, encrypting, stealing, and damaging sensitive data without the knowledge and permission of a user.

2.2.Types of Classification Method of Attack

2.2.1. Static Analysis Method

Static analysis is used to discover security vulnerabilities in source code [22]. It is capable to detect all program paths along with the application by analysing and testing the source code of the web application. Nevertheless, the main drawback is the false positive rate, which makes the incorrect results and affects the final results of vulnerability detection. False-positive indicates detecting some paths as vulnerable paths, while in the real it is a safe path and there is no vulnerability in the paths.

In other words, this static analysis extracts lexical features from each source code to identify whether suspicious keywords or fragments exist in the code [23]. It is usually combined with the technique of classification learning. So, static analysis extracts static features source code to train a classifier that predicts whether the code is benign or malicious. There is a different approach used static analysis.

A static analysis method used by *Likarish et.al* [10] used the normalized frequency of each JavaScript keyword as a feature and builds the detection model with machine learning classifiers. The others, Convoluted Partitioning of Long Sequences (CPoLS) static analysis based model proposed by *Jack et.al* [12]. *Gupta et.al* [24] develop static analysis to predicted cross-site scripting by proposing a novel feature extraction algorithm to extract basic and context features from the PHP source code of web applications. The authors [25], [26], [27], and [28] use static analysis based on the detection of cross-site scripting attacks. Static analysis, performed just once, is used to reduce the run-time overhead of the dynamic monitoring phase.

2.2.2. Behaviour Analysis Method

Behaviour analysis method interrelated to the web application under examination, rather than using the source code in a similar way to the behaviour of a user with a web browser. It is a method that has fewer false positives which is an advantage over static analysis. However, this method cannot ensure that it will check every vulnerability in a web application, for this reason, it can identify vulnerability in the program path. The dynamic analysis puts web pages directly into a browser engine. By logging and analysing the execution information, the method can

provide extra details about any malicious behaviours. However, it requires more computing resources to emulate the execution environment. There are different approaches used for dynamic analysis.

Richards's et.al [29] propose JavaScript program behaviour by analysing execution traces recorded from a large corpus of real-world programs by using instrumented browser records. *Pietraszek*, and *Berghe* [28] develop an interpreter-based approach to identify untrusted data at the character level and to check vulnerabilities they use context-sensitive string. *Jim, Swamy, and M.Hicks* [30] propose a mechanism that modifies the browser so that it can execute only filter content to prevent injected script code from running in browsers that view the site. This was a good idea that allows only the scripts in the provided list to run because of this the browser can be used to filter the scripts and the developer of the web application knows scripts that should be executed for proper application functioning so the website can specify the legitimate scripts and filter the malicious scripts.

The main drawback of this mechanism is it requires modifications in the frameworks or installation of additional frameworks and only approved scripts have to be identified by the website. So, it suffers from scalability problems from the web application's point of view.

2.2.3. Hybrid Analysis Method

As understand from its name, hybrid analysis tools combine the functions of dynamic and static analysis. Given that both static analysis and dynamic analysis techniques have positive and negative features, a hybrid method is introduced that combines their positive features.

According, *Xincheng et al* [31] develop a hybrid analysis method for detecting malicious JavaScript code based on a machine learning algorithm. *Wang et al* [23] propose a detection method by combining static and dynamic analyses for detecting malicious web pages using the decision tree machine learning algorithm. *Balzarotti et.al* [32] develop WebSSARI, the aim is to find XSS, SQL injection, and general script injection, which combines static and runtime features and find security vulnerability by applying static taint propagation analysis by using the WebSSARI tool.

2.2.4. Heuristic-based Detection

Heuristic-based detection is an expert system-based analysis of a code segment to define whether the piece of code or file is suspicious based on the standards of knowledgeable decision rules [19]. The problem with using this methodology is a considerable amount of time in scanning and analyzing whether a piece of code is malicious or benign. An additional challenge with this methodology is the highest false-positive rate gained during analysis. False-positive arise in a situation when a detection apparatus classifies a code as suspicious but when they are not. To define whether code or a file is suspicious or not in a very short time with high accuracy and without isolating, the code segment or a file in sandbox machine learning has been introduced.

According to the previous literature show that static techniques have a high false-positive rate and extract lexical features while dynamic techniques have fewer false positives. Heuristic techniques have a high false-positive rate and needed an expert for decision rules.

2.3. Abstract Syntax Tree for Static Analysis

At present many software engineering methods, such as source code classification, code clone detection, defect prediction, and code summarization have been proposed to improve software development and maintenance [33]. The main trouble of the above method the way of representing source code to effectively encapsulate the syntactic and semantic evidence that is embedded in the source code.

The traditional code representation approach such *as Information Retrieval (IR)* usually treats code fragments as natural language texts and models them based on tokens [34], latent semantic indexing [35], and Latent Dirichlet Allocation (LDA) [36] to analyse source code. The problem of the traditional approach is that they assume the underlying corpus is composed of natural language texts they should not be simply dealt with text-based or token-based methods, which could miss important semantic information of source code.

Recent work comes up with strong evidence that syntactic knowledge contributes more to modelling source code and can obtain better representation than traditional token-based methods.

The approach is called the Abstract Syntax Tree (AST). AST is encapsulating both the lexical and syntactical information from source code.

The abstract syntax tree is a hierarchical program illustration that offerings source code structure [37] according to the grammar of a programming language, each AST node matches to an item of source cod. It has been broadly used by programming language and software engineering tools that are used for an accurate and abstract representation of a program.

AST is easy to process as a program than a source code or parse tree and it is used as an intermediate representation in the whole process of the compiler implementation. The syntax is abstract, so it does not represent every detail in the real syntax, but the structural or content related details [37]. For example, grouping parentheses are implicit in the tree structure, so these do not have to be represented as separate nodes. Similarly, a syntactic construct like an if-condition-then expression may be denoted using a single node with three branches.

The different researcher has been used AST to represent the source code as a tree structure. The author of Zozzole [11] uses an abstract syntax tree to hierarchical features extract from JavaScript source code. Similarly, [5], [38] and [39] use AST for their works to the representation code structure.

2.4.Cuckoo Sandbox for Behavioural Analysis

Cuckoo sandbox is an open-source used for dynamic analysis sandbox created by Guarnieri et al [40] in 2013 It is consists of central management software that handles sample execution and analysis. Cuckoo reports including all traced activity of any suspicious analysed files at run time environment[40].

Cuckoo can check any suspicious file in a few minutes. It provides a detailed report showing the behaviour of the file is executed in an isolated and realistic environment. Cuckoo Sandbox is a state of the art and open-source automated malware analysis system with endless application possibilities. It has two sub-components as shown in Figure 2-1. The first component is the machine used to start the analysis of malware, to store the results in the database, and to start the web service provided for the users. The other component is the analysis machine, virtual or physical machine, on which the malicious software is run, the actual analysis is performed,

similar to the real working environment of the malicious software. The following are the list of supported file formats:

- Generic Windows executables
- DLL files
- PDF documents
- Microsoft Office documents
- URLs and HTML files
- PHP scripts
- CPL files
- Visual Basic (VB) scripts
- ZIP files
- Java JAR
- Python files, and almost anything else.

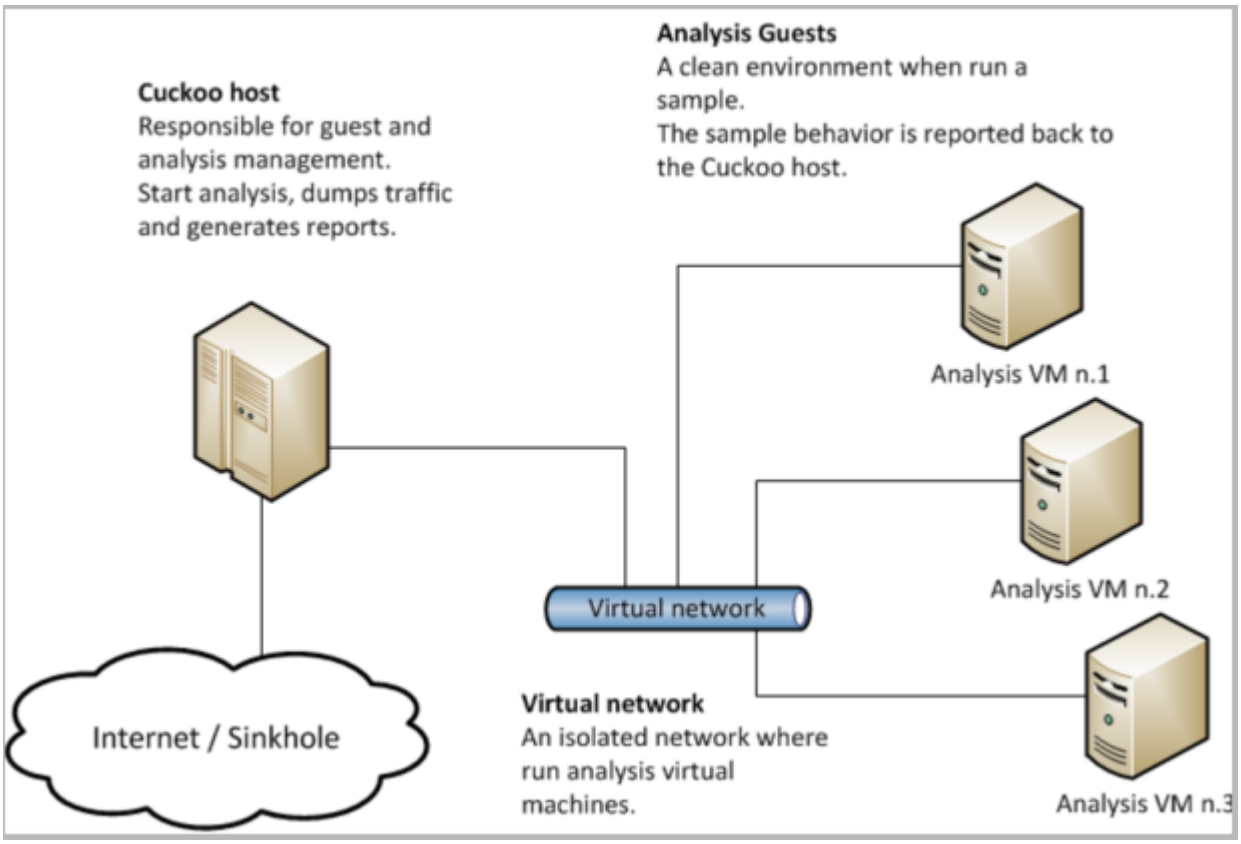


Figure 2-1 Cuckoo Sandbox architecture [40]

The sandbox generates the report which outlines all the behaviour of the file in the system. The report is represented as a JSON file, and currently, it is capable of detecting features like traces of activity performed by scan files like processes spawned, files being created, deleted, and downloaded during its execution memory dumps of the malware processes, screenshots that were taken during the execution of the malware and full memory dumps of the machines.

2.5.Machine Learning Concepts

2.5.1. Overview of Machine Learning and its Importance

Machine learning is the practice of programming computers to learn from data. The importance of machine learning is when you have a problem that requires many long lists of rules to find the solution. In this case, machine-learning techniques can simplify your code and improve performance. In other also when very complex problems for which there is no solution with a traditional approach. The following Figure 2-2 shows the overall activites exits in the machine learning process.

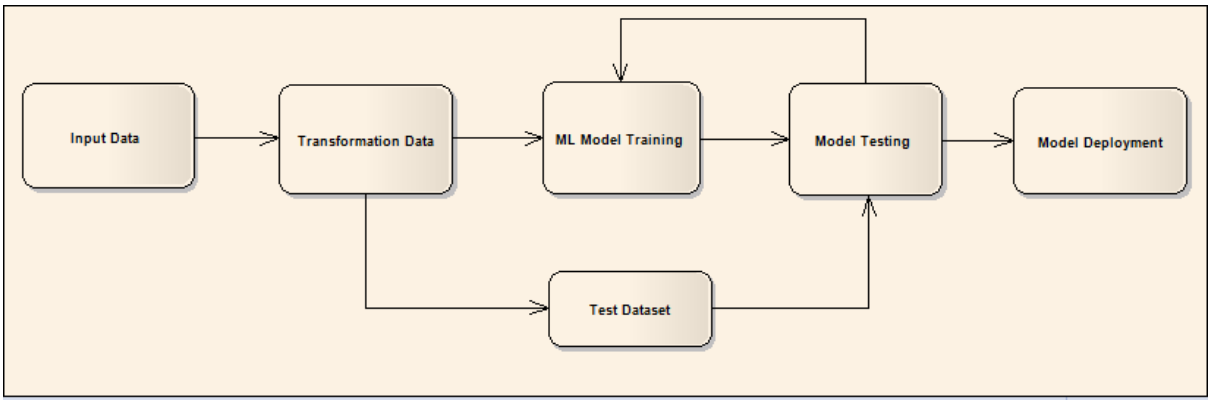


Figure 2-2 Workflow of the machine learning process

- **Input Data:** the dataset is loaded from the file and saved in the memory.
- **Transformation of data:** After importing the data, we need to transform, normalize, and clear data to be suitable for our ML model.
- **ML Model training:** The model is constructed using the selected algorithm.
- **Model Testing:** the model which is constructed is tested using the test dataset.
- **Model Deployment:** at this stage, an optimal model is selected.

2.5.2. Types of Machine Learning

Machine learning techniques are categorized into the taxonomy, based on the output provided. There are four common machine learning algorithm types such as supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning[41].

Supervised Learning

Supervised learning asks the machine to learn from our data when we specify a target variable [42]. The dataset in a supervised learning algorithm is divided into a training dataset and testing dataset, which helps us to predict outcomes for unforeseen data. To learn the model training dataset must be fed into the supervised algorithm. At a subsequent time, the test dataset has verified the model by estimating the predicted label with the test label of the data. Classification and regression are examples of *supervised learning*.

Classification is used when input data are divided into two or more classes that is predict the data into a specific class. It can be used in input training data to predict the likelihood that subsequent data will fall into one of the predetermined categories. A good example is spam filtering is an example of classification, where the inputs are messages and the classes are spam and not spam. It is used when the output variable is a category such as red or blue or disease and no disease. There are different classification machine learning such as SVM, decision tree, KNN, and Naive Bayes.

Regression is another type of supervised learning which is used when the outcome is a real or continuous value. E.g. Predicting age, the nationality of the person, predicting whether the stock price of a company will increase tomorrow, and predicting whether a document is related to the sighting of UFOs.

The following tables show that the advantage and drawback of machine learning algorithm used for classification.

Table 2-1 Classification machine learning algorithm advantage and drawbacks.

Algorithm	***Advantage***	***Drawback***
SVM	High accuracy, deal with high dimensional, handle imbalance class, and non-linear problems.	A large dataset required higher training time, does not perform very well when the dataset has more noise or overlapping classes.
NB	Perform well small dataset and multi class prediction	Zero conditional probability Problem, and a very strong assumption of independence class features
DT	Zero conditional probability Problem, and a very strong assumption of independence class features	Easy to overfitting; No ranking score as a direct result
KNN	High precision and accuracy, nonlinear classification, no assumption of features;	Sensitive to unbalanced sample set, heavy computing burden, poor interpretability.

The Table 2-1 shows that the advantage and draw back of most popular machin learning algorithm. SVM has high accuracy and deal with high dimensional when comparing with the other machine learning algorithm.

Unsupervised Learning

In unsupervised learning, the algorithm is provided with an unlabelled dataset and it predicts a pattern in the data. Unsupervised learning doesn't have a target variable as we did in classification and regression. Instead of telling the machine *Predict Y for our data X*, we're asking *what can you tell me about X* [42]. The machine learning algorithm receives only input data and the task to extract knowledge out of the data. In unsupervised learning contains two major types: Clustering and Association.

- **Clustering:** A clustering problem is where we group similar data according to a pattern in data, such as grouping customers by purchasing behaviour.

- **Association:** An association rule learning problem is where we want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Semi-supervised Learning

Semi-supervised learning is another type of machine learning which used both labelled and unlabelled data are combined in this learning type to improve the performance of generalization and to perform otherwise supervised and unsupervised learning. This type of learning used a small amount of labelled data with large unlabelled data.

Reinforcement Learning

Reinforcement learning is another type of machine learning which determines the interaction of the agent with the environment based on feedback that might be positive or negative. The objective of this learning in determining action within the specific environment is improving positive feedback called a reward. Unlike other machine learning or (supervised, unsupervised, and semi-supervised), reinforcement learning does not depend on the training data, rather it is based on action or interaction that happened between agent and environment. The interaction happens with the environment but not with data.

2.5.3. Most Popular Classification Algorithms

- **Naïve Bayes:** it is a simple and powerful algorithm for predictive modeling. It is considered a probabilistic classifier; hence, it is based on the Bayes probability theorem [43]. The model comprises two types of probabilities that can be calculated directly from the training data: (i) the probability of each class and (ii) the conditional probability for each class given each x value. Once the probability is calculated, the model can be used to make predictions for new data using Bayes theorem as denoted by.

$$P(C_i \mid O) = \frac{P(O|C_i)P(C_i)}{P(O)} \qquad \textit{Equation 2.1}$$

Where $P(C_i \mid O)$ is the posterior probability of being a class label C_i given the attribute value O, $P(O|C)$ is the probability of the attribute O when the given class label is C_i, $P(C_i)$ is the

prior probability of the class C_i label, and $P(O)$ is the prior probability of the attribute value O.

Naïve Bayes classifier is preferable when the dataset has a small size and noncorrelated or independent attributes. A dataset with many attributes would take a long time to calculate the posterior probabilities, that is $P(C_i \mid O)$

- **Support Vector Machine (SVM):** SVM works by constructing hyperplanes for linearly separable data in multidimensional space using a kernel function [44]. The hyperplanes separate the sample data into different groups. Objects are rearranged for optimal separation through the use of this kernel function. A kernel function provides a nonlinear mapping from inputs x to feature space φ(x). During mapping from the input space to the feature space, the mapped object can be separated by a straight line with the new space. All the instances and target classes are represented as vectors in high-dimensional space and the SVM finds the closest two points from the two classes that support the best separating line or hyperplane as shown in Figure 2-3.

  Although SVM was initially designed to classify two classes of linearly separable data, today it is used to classify data that are composed of more than two classes that cannot be separated linearly. In multi-class problems, datasets are classified using more than one binary SVM. SVM is capable of training very quickly and performs classification and regression tasks.
- **Logistic Regression (LR):** The logistic regression algorithm is considered as a linear model that measures the relationship between the target variable and other variables by estimating the probability using a logistic function [43]. In LR, input values (χ) are combined linearly using weights or coefficient values to predict an output value (y). The output of LR is a value between 0 and 1. Results closer to 1 indicate that the input variable more clearly fits within the category while results closer to 0 indicate that the input variable likely does not fit within the category. The ease of mathematical interpretation of the model makes this method suitable.

$$y = \frac{e^{B0+B1x\chi}}{1 + e^{B0+B1x\chi}} \qquad \textit{Equation 2.2}$$

Where y is the predicted output, $B0$s the bias or intercept term and $B1$ is the coefficient for the single input value (χ).

Making predictions with an LR model is as simple as plugging in numbers into the LR equation and calculating the result. However, one challenge with this algorithm is that we cannot understand the predictions as a linear combination of the inputs.

- **Random Forest:** Random forest is an ensemble classifier that fits a large number of DTs to a dataset and then combines the prediction from all trees. Ensemble methods are methods that combine multiple ML models to create more powerful models [42]. Random forest works on the voting mechanism and predicts the output class that received the maximum votes from all individual DTs [43]. It can be used for classification and regression. RF algorithm starts by randomly selecting a subset of training data and a single DT is generated with the sub-sample.

  The random selection of examples is repeated many times, and a single DT is generated with each sub-sample which finally forms a forest. After the forest is constructed, individual trees make a classification decision or prediction and then the final predicted class of observation is made by majority vote. One of the main advantages of RF is that it reduces the risk of overfitting.

  With RF, it is easy to build and train, and often it gives very good results with less chance of overfitting than DTs. This is due to the use of a random selection on subsets of variables and its voting scheme through which a decision is made. RF is appropriate for high-dimensional data modeling because it can handle missing values and can handle continuous, categorical, and binary data.

  The ensemble scheme of RF makes it enough to overcome the problems of overfitting and hence there is no need to prune the trees. Besides, its high prediction accuracy, efficiency, and interpretability make it preferable for various types of datasets.

- **Decision Trees (DT):** A DT is a supervised ML algorithm, that can be used to develop a learning model to predict a class or value of a target variable by learning the decision rule of training data [43]. This algorithm can be used to solve classification and regression problems. DTs are arranged in a hierarchal tree which is created by partitioning the training data into various groups to find homogenous subgroups. Knowing which attribute to be placed at the root and each level is the main challenge in constructing a DT. Entropy and

information gain are statistical measures that are used to construct a tree in the DT. Entropy deals with calculating the homogeneity of information. The total entropy of a given dataset is obtained by first splitting the dataset based on different attributes and calculate the entropy for each branch and adding each of the entropy for each branch proportionally.

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \qquad \textit{Equation 2.3}$$

Where $E(S)$ is the entropy of the dataset, and p_i is Probability of randomly picking an element of class i.

Information gain (IG) deals with calculating the expected reduction in entropy. IG for a split is calculated by subtracting the weighted entropies of each branch from the original entropy (entropy before split) by using the below equation. Attribute with maximum IG is considered to split.

$$Info(A) = E(S) - E(X) \qquad \textit{Equation 2.4}$$

Where $Info(A)$ is information gain of a specific attribute 'A', $E(S)$ is the entropy of the dataset given and $E(X)$ is the entropy of a specific attribute in the given dataset.

After the IG for each attribute is calculated, attributes are ranked in their order and the attribute with the largest IG is placed as the decision node in the DT. Although IG is a good measure for deciding attribute relevance, it has biasness problem when it is applied to attributes with a large number of values. One solution to this problem is using the gain ratio method.

Gain ratio (GR) reduces the bias on multivalued attributes by considering the intrinsic information of a split when choosing an attribute. The GR takes the number and size of branches into account when choosing an attribute. Intrinsic information is the entropy of the distribution of instances into branches (i.e., how much information do we need to tell which branch an instance belongs to).

$$Gain\ ratio(Atrribute) = \frac{Gain(Attributed)}{Intrinsic_{info(Attributed)}} \qquad \textit{Equation 2.5}$$

In general, DT algorithms are suitable for multiclass classification. But the problem with DT algorithms is overfitting.

2.5.4. Application of Machine Learning in Cybersecurity

Cybersecurity is the collection of policies, techniques, technologies, and processes that work together to protect the confidentiality, integrity, and availability of computing resources, networks, software programs, and data from attack [45]. Cybersecurity involves defensive key data and devices from cyber threats. Modern cyberattacks utilize sophisticated techniques to bypass security countermeasures. State-of-the-art security techniques are integrated with artificial intelligence to sense anomalies and model and detect threats. This is achieved by deploying diverse machine learning methods.

Solving cybersecurity complications with machine learning is the most state-of-the-art technique to identify defects and weaknesses in security breaches. This new approach includes turning to big data platforms to extend data accessibility and machine learning for the detection of advanced persistent threats. The application of machine learning techniques in cybersecurity is increasing than ever before.

Applying a machine-learning algorithm to give a solution to the cybersecurity problem in different security areas will be started a long time ago. As information and communications become gradually increase more data become available, many security risks arise as well as to manage and mitigate such risks. As a result, many researchers develop the techniques for applying and designing machine learning algorithms for security like intrusion detection systems (IDS), malware classification, security policy management (SPM), and information leak checking.

Earlier, the malware was detected and prevented by signature-based techniques like HMM have been applied for malware detection by using the method of signature scanning. For example, in [46], the authors proposed that HMMs can easily detect various classes of metamorphic malware. the paper [47] provides an outline for a metamorphic malware generator that is provably immune to signature detection. In [48], the paper has shown that HMMs can successfully recognize malware that is based on the technique in [47] the work. But signature techniques suffer from difficulties to detect zero-day malware or advanced malware. To overcome these limitations, machine learning techniques are gaining popularity in malware detection. The advantage of machine learning techniques is that it will not only detect known malware but also deliver knowledge for the detection of new or obfuscated malware.

[49] argue that IDS is based on machine learning. They use the KDD dataset to perform and evaluate to assess various machine learning classifiers depend on the KDD dataset. Their focus was on false (host intrusion detection system) or to monitor all network negative and false positive performance metrics to enhance traffics (network intrusion detection system) which is the detection rate of the intrusion detection system.

As [50] discussed in the work intrusion detection using machine learning by using rough set theory and support vector machine to detect network intrusions. Rough set theory is used to pre-process the data and reduce the dimensions and the features selected by rough set theory will be sent to the support vector machine model to learn and test respectively.

The author [49] noted the IDS which can effectively identify anomaly behaviours in the network. They propose an effective IDS by using hybrid data optimization. In data sampling, they use the Isolation Forest (iForest) to eliminate outliers, genetic algorithm (GA) to optimize the sampling ratio, and the Random Forest (RF) classifier as the evaluation criteria to obtain the optimal training dataset.

Machine learning techniques are a standard approach to spam detection. *M. Luckner et.al* [51] propose a novel web spam recognition system the system automatically rebuilds the learning set using external data from spam traps and popular web services. In [52] argues that the machine learning-based data analysis method accurately detects abnormal behaviours due to malware in large-scale network traffic in real-time. They develop an efficient algorithm for detecting malicious behaviours in a network environment and conduct experiments to verify algorithm accuracy.

2.5.5. Role of Machine Learning in Software & Application Security

Machine learning roles in software and application security mostly concentrate on web application security like filter unwanted resources and requests such as malicious advertisements, predict unwanted resources and requests, quantify web application vulnerabilities, detect web-based malware, and JavaScript (JS) code.

The *Hainsworth* et.al [53] propose the detects and blocks adversarial advertisements that may attempt to profit from the creation of low-quality or harmful advertisements. Their approaches form the basis of a system that quickly and reliably identifies adversarial advertisements and

blocks them from serving. *C. Dunn et.al* [54] propose an alternate approach that leverages machine learning to bootstrap a superior classifier for ad-blocking with less human intervention. Their classifier can simultaneously maintain an accuracy similar to the hand-crafted filters while also blocking new ads which would otherwise necessitate further human intervention in the form of additional handmade filter rules.

The authors [55] is implemented, and evaluate a general methodology to identify webservers that are at a high risk of becoming malicious before they become malicious. In the [56] introduce *MEERKAT*, a defacement detection system that requires no prior knowledge about the web site's content or its structure, but only its URL, by automatically learns high-level features from screenshots of defaced websites by combining recent advances in machine learning, like stacked auto-encoders and deep neural networks, with techniques from computer vision.

As shown by. *Evans et.al* [57] develop a black-box tool for detecting and quantifying the severity of side-channel vulnerabilities by analysing network traffic over repeated crawls of a web application. They develop a methodology to more thoroughly measure the distinguishably of network traffic for a variety of classification metrics.

P. Chapman and D. Evans [57] have noted that the novel approach to automatically detect evasive behaviour in malicious JavaScript. Uses efficient techniques to identify similarities between a large number of JavaScript programs despite their use of obfuscation techniques, such as packing, polymorphism, and dynamic code generation, and to automatically interpret their differences to detect evasions.

2.6.Machine Learning for Attack Classification.

Machine learning is a subfield of computer science that uses a statistical method that enables the computer the ability to learn with data without being explicitly programmed.

2.6.1. Feature Extraction Techniques

The objective is to extract a set of features that can be used to classify samples between classes, within a specified percentage error using a feasible decision scheme. There is a different feature extraction method in malicious attack classification. The following are some methods used in the previous work.

Word2Vec Techniques: it is a word vector feature training tool open-source by Google in 2013 [58]. Word2vec feature extraction is used especially for nature processing language but also it has been used in malware classification in the past decades. This kind of feature extraction has mainly two main methods: bag-of-word and word embedding.

Bag of Words is a representative model of document data, which simply counts how many times a word appears in a document [59]. Bag-of-Words is commonly used in clustering, classification, and topic modelling by weighing special words and relevant terminologies.

Word embedding is one of the document representations in the vector space model [59]. It captures contexts and semantics of words, unlike the Bag-of-Words model. Bag-of-Words only represents the number of occurrences of words in a document without any relationships and contexts. On the other hand, it is preserves contexts and relationships of words so that it detects similar words more accurately [60].

Word embedding has different implementations such as word2vec, FastText, GloVe. word2vec is the most popular implementation of word embedding. Word embedding is always a vector associated with each word. It has two different algorithms: CBoW (Continuous Bag-of-Words) and Skip-gram model [58]. CBoW is to predict a target word from context. Skip gram is the reverse of bag-of-words that is to predict content from the target word.

The previous researcher used word2vec for malicious activity classification. *Popov* [60] proposed applying word2vec techniques for extracting vector embedding of machine code instruction and evaluate a convolutional neural network-based classifier that uses extracted vectors for malware detection. Fang et.al [13] propose word2vec feature extraction for malicious detection. They use word2vec to create the feature vectors from the semantic level of the bytecode of JS source code.

TF-IDF Techniques: it is a statistic widely used in information retrieval and text mining domains for calculating frequency-based weights for terms in a document to assess their relative importance. Term frequency-inverse document frequency-based text dataset vectorization is a traditional method to create a numerical dataset.

$$tf_{(t,d)} = f(t,d) \qquad \text{Equation 2.6}$$

Whereas *t* is the term in document *d* is assigned a weight assignment according to the frequency of occurrence in that document. This process is called term frequency, *tf (t, d)*. The vector, thus formed, can be considered as a digitized summary of a document. Let the term to be represented by *f (t, d)* of the raw frequency.

However, a dataset generated only by the term frequencies assigns equal importance to each term. The inverse document frequency (IDF) is defined as the logarithm of the division of the number of documents in the, *N*, to the document frequency, *dft*. Thus, if the *dft* in the documents in the collection is *low*, the *idft* value will be *high*, and in the *high dtf* frequency, the *idft* value will *below*. To calculate the weights of the terms contained in each document, the term frequency, tft;d, and inverse document frequency, *idft*, are combined to form the term frequency-inverse document frequency (TF-IDF) matrix.

N-gram Techniques: it is a subsequence of *N* consecutive tokens in a stream of tokens. N-gram analysis has been applied in many tasks and is well understood and efficient to implement. By converting a string of data to n-grams, it can be embedded in a vector space to efficiently compare two or more streams of data.

Zhang et.al [61] use n-gram analysis for detecting malicious code by using signature-based techniques. The other, JAST [15] also use n-gram techniques for feature extraction purpose from the abstract syntax tree with a random forest classifier. It is based on a frequency analysis of specific patterns, which are either predictive of benign or of malicious samples. *Kolter et al*. [62] used n-grams to develop detecting malicious executables files using the decision tree model.

Binary feature extraction: The baseline method of extracting features from a sequence is to identify all the distinct elements found in the sequence [63]. The sequence can then be represented as a binary vector of the same length as the term dictionary S such that each feature in the vector signifies the presence or absence of the corresponding dictionary term in the sequence.

The resulting feature vector can be represented as $V_{Sb} = (b_{S1}, b_{S2}, b_{S3,\ldots,b_{Sn}})$, where b_{Si} is 1 if S contains at least one instance of s_i and 0 otherwise, and n is the size of S. Tain R .et.al [64] proposed a binary feature technique for malware detection and classification using API calls.

The authors also experimented with the frequency-based methods on the same data but no improvement was observed over the binary representation.

Doc2vec Techniques: The doc2vec algorithm for learning neural network-based document embedding is widely used in text classification, sentiment analysis, information retrieval, spam filtering, malware detection. *Ndichu et.al* [5] used doc2vec techniques for detecting malicious code by using the static analysis method. They used doc2vec to learn context information of texts with a variable length of the documents. The experiment results show that doc2vec feature extraction has better performance and a fast detection rate.

2.6.2. Machine Learning Techniques

Machine learning can learn from the patterns and is capable of detecting a new variant of malware with a very low false-positive rate [19]. Machine learning is very beneficial especially used as the polymorphic nature of malware is ever increasing. In particular, machine learning-based malware detection approaches can lead to fast detection of malware and do not require the regular updating of the anti-malware tool on the client-side. Several studies have been conducted using machine learning for the detection of malicious JavaScript's such as [5], [6], [15], [65]–[67].

Support vector machine algorithm has been used in a variety of applications such as text classification, facial expression recognition, gene analysis, and malicious analysis [44]. It is a classification and regression prediction tool that uses machine learning theory to maximize predictive accuracy while automatically avoiding over-fit to the data. It can be defined as systems that use hypothesis space of linear functions in a high dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory [68].

Support Vector Machines building a classifier by creating a boundary between two classes which enables the prediction of labels from one or more feature vectors, this boundary is known as the hyperplane or decision plane. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier.

A support vector machine model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the *kernel trick*, implicitly mapping their inputs into high-dimensional feature spaces.

Suppose a given labelled training dataset:

$$(x_1, y_1), \ldots, (x_n, y_n), x_i \in R^d \; and \; y_i \in (-1, +1) \qquad \text{Equation 2.7}$$

Whereas x_i is a feature vector representation and y_i the class label (negative or positive) of a training compound *i*. The optimal hyperplane can then be defined as:

$$wx^T + b = 0 \qquad \text{Equation 2.8}$$

Where *w* is the weight vector, *x* is the input feature vector, and *b* is the bias. The w and b would satisfy the following inequalities for all elements of the training set:

$$wx_i^T + b \geq +1 \; if \; y_i = 1 \qquad \text{Equation 2.9}$$

$$wx_i^T + b \leq -1 \; if \; y_i = -1 \qquad \text{Equation 2.10}$$

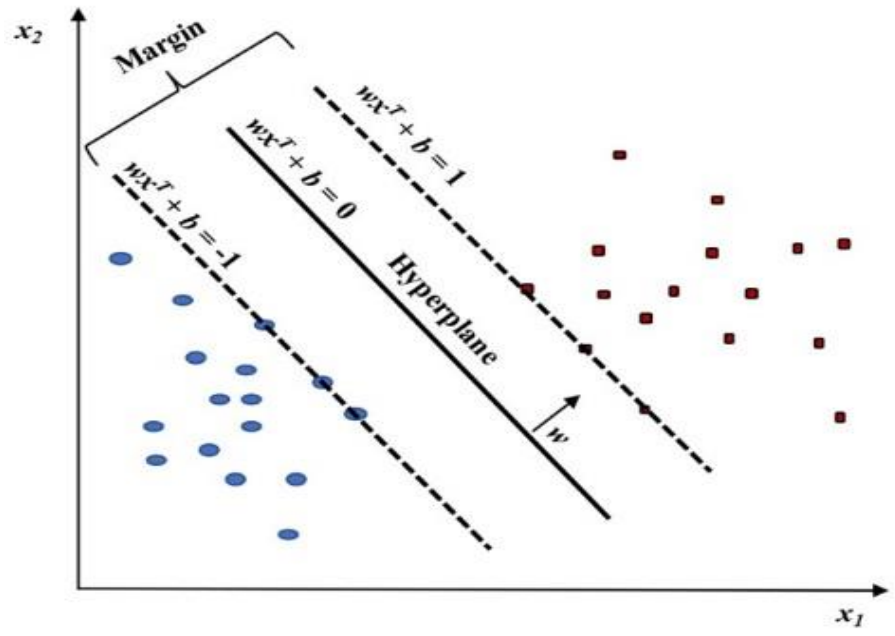


Figure 2-3 Linear SVM model [69].

As shown in Figure 2-3 linear SVM models the objective of training an SVM model is to find the *w* and *b* so that the hyperplane separates the data and maximizes the margin $\frac{1}{||w||^2}$1. Vectors x_i for which $|y| = (wx_i^T + b) = 1$ will be termed support-vector.

2.6.3. Classification Algorithms Evaluation Metrics

Evaluation metrics are tied to machine learning tasks. The metric is used to consider the goodness of fit between model and data, to associate different models, in the context of model selection, and to predict how predictions are expected to be accurate [70].

In the evaluation of classification models, basic concept is the notion of fault. If the application of the classification models in the selected case leading to the prediction of a class that is different from the actual class examples then there is a classification error. If any mistake is equally important, then the total number of errors in the observed set can be an indicator of work a classifier.

The most common metrics used in most machine learning classification applications are the accuracy, confusion matrix, precision, recall, and fl-score. All the classification metrics that we mentioned here are based on actual labels and predicted labels of samples.

Train-Test Split Evaluation

The train-test split is a technique for evaluating the performance of a machine learning algorithm. The procedure involves taking a dataset and dividing it into two subsets called training datasets and test datasets. The training datasets are used to fit the model. The testing datasets are delivered to the model then predictions are made and compared to the expected values.

The objective is to assess the performance of the machine learning model on new data that not used to train the model. The train-test evaluation method used when there is a sufficiently large dataset available.

Cross-validation evaluation

Cross-validation is the most common evaluation method used to evaluate machine learning models on a limited data sample. It has a only parameter *k*, called *k-fold cross-validation,* that raises the number of groups that a given data sample is to be split into. When a specific value for *k* is chosen, it may be used in place of *k* in the reference to the model, such as *k=10* becoming *10-fold cross-validation*.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

Confusion Matrix

The confusion matrix is also known as the Error matrix and is represented by a table that describes the performance of a classification model on a set of test data in machine learning. A confusion matrix is a table that summarizes how successful the classification model is at predicting examples of various classes.

Table 2-2 Confusion matrix

		Actual Values: Positive (1)	Actual Values: Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

As observed from Table 2-2 shows individual cells of the confusion matrix in the table with binary classes. TP (True Positive) means the observation is positive and is predicted as positive, FP (False Positive) means observation is positive but is predicted as negative, TN (True Negative) means the observation is negative and is predicted as negative and FN (False Negative) means the observation is negative but it is predicted as positive.

Accuracy

Accuracy is the percentage of correct predictions made by the model. This metric is commonly used for assessing classification models. It measures how often the classifier makes the correct predictions. It represents how the classifier predicts benign data as 'Benign' and as 'Malicious' for malicious data. This metric is useful when there are equal numbers of observations in each class and all predictions are important. Accuracy will take value in the range of [0, 1]. If accuracy is equal to 1 that means all samples in the dataset are correctly classified. In contrast, if accuracy is equal to 0 that means none of the samples in the dataset is classified correctly. Accuracy is computed using the following Eq. The accuracy is the proportion of the total number of correct predictions. It is determined using the equation.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \qquad \text{Equation 2.11}$$

Precision

Precision measures the number of actual positive cases out of all the positive cases predicted by the model based on the positive class using the following equation. This is also called positive predictive value. Out of the items that the classifier predicted to be true, how many are actually true? In our case, how correct enough is the classifier to recognize the benign samples? The formula for calculating precision is as follows.

$$Precision = \frac{TP}{TP + FP} \qquad \text{Equation 2.12}$$

This metric is more important in cases where we are more concerned with finding the maximum number of positive classes.

Recall

A recall is the percentage of actual positive cases that the model can predict correctly out of the total number of positive cases. In our case, how correct is the classifier to classify all available malicious instances as 'Malicious'. The formula for calculating recall is denoted by Eq.

$$Recall = \frac{TP}{TP + FN} \qquad \text{Equation 2.13}$$

F1- score

F1-score is denoted by the following equation. It is the harmonic mean of the values of precision and recall. F1- score close to 1 means a perfect model. However, f1- score close to 0 shows a low model's predictive capability. This metric is important for unbalanced classes. F1 - score of a model is calculated as:

$$F1\ score = \frac{2 * Precision * Recall}{Precisoin + Recall} \qquad \text{Equation 2.14}$$

FPR also called false alarms or specificity; determines the total number of incorrect predictions among all negative samples in the dataset. FPR is defined by Eq. below.

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \qquad \text{Equation 2.15}$$

AUC-ROC curve

AUC-ROC (*Area Under Curve - Receiver Operating Characteristics*) is a measurement technique used for binary classification to determine how well a classifier can distinguish between classes at various threshold settings. The higher the AUC rate is, the better the model is predicting the different classes.

To plot a ROC curve, the calculations of TPR (True positive rate) and FPR (False positive rate) is used, equations. $TPR = \frac{TP}{TP+FN}$, and $FPR = \frac{FP}{FP+TN}$

2.7.Related Works

Due to the thoughtful harm caused by malicious JavaScript code, it has fascinated expansive attention to the security research area. At present, several network security scholars and experts have proposed a variety of methods in research articles to detect malicious JavaScript code. Therefore, this subsection introduces related and most recent scientific research with a focus on the classification of JavaScript-based attacks. The paper uses a different analysis approach and machine learning techniques.

Patil et.al. [17] developed a methodology for the detection of malicious JavaScript on the web page with a low false-positive rate and false-negative rate. From collected the benign and malicious JavaScript's from the benchmark sources of Web pages, the authors performed the static analysis of every benign and malicious script with the help of feature extraction. They used seven machine learning algorithms for classifies JavaScript using the WEKA tool. The experimental results show that machine learning classifiers have achieved a good detection rate between 97- 99% with very low FPR and FNR. Because they use static analysis still attackers use different obfuscation techniques to hide the real structure of the JavaScript.

According to [16] point out a novel framework to detect JS malware called JSAC by combines deep learning and program analysis techniques to capture the syntactic and semantic features of JS programs. They use abstract syntax tree and control flow graph to get the syntactic and semantic information from the JS codes respectively. The information extracted is finally fused

to the final detection. They use the CNN model for the detection. Experiment results show that JSAC outperforms 4 machine learning-based tools and CNN in detecting JS malware and better prediction with 98.73% F1-score. They focus on the semantic and syntax of the JS code but they don't include the behaviour part of JS code.

As shown by, *Wang et.al* [71] developed a machine learning-based malicious JavaScript detection method, which is based on fully static analysis of scripting language, to extracts the static information of the script, and improves the detection efficiency and safety of the system, without parsing and compiling the script. They extracted 27 features from collected JavaScript codes using static analysis. Experimented on the four machine learning algorithms, according to their results, SVM is excellent both on accuracy and detection efficiency which, obtains more than 90% both on accuracy and recall, and the accuracy on the training set even raised to 93.8%. They focus on static analysis only and also feature extraction mechanism they used is manual and needs an expert to extract the feature from the datasets. Behavioural analysis wasn't included in their study.

A study by *Kolter et al*. [62], developed detecting malicious executables in the wild. The extracted feature from malicious executable using static analysis. The algorithm they used is Naïve Bayes, SVM, and Decision Trees for the classification of n-grams. For their study, they use benign and malicious executables datasets and encoded each as a training example using n-grams of byte codes as features. The experimental results show that the J48 classifier outperformed other classifiers with a ROC of 0.966. They focused on static analysis of the executable files and compared Naive Bayes, Support Vector Machine, and Decision Tree based on the *n*-grams.

According to *Kan and Thi* [72], first probe the use of machine learning for the detection of malware. In their study, the webpage was analysed for any malicious activity based on the URL. The authors performed lexical analysis on the URL of the webpage and features are extracted features from it for classification purposes. The experimental results obtained show that their approach has a high false-positive rate. In this study, they did not perform any analysis on code for any suspiciousness.

Suga et.al. [73] proposes a method to detect malicious JavaScript snippets with Paragraph Vector, an unsupervised algorithm to generate vectors for documents with neural networks.

They extract lexical features by using Doc2Vec, and mitigate the class imbalance problem with a clustering-based under-sampling technique and evaluate the practical performance on an imbalanced dataset. The experiment result shows that the proposed method achieves an F-measure of 0.71. They used lexical analysis only but it has drawbacks on the representation of the source code. They use imbalanced datasets and the proposed method has low accuracy.

Nayeem et.al. [19] proposed deep learning-based malicious JavaScript detection using a hybrid analysis approach. The proposed architecture consists of an interceptor between the client's browser and a web server which is works as an agent for all the HTTP traffic. They use an interceptor as intercepting the request generated from the client which is responded to by a web server to perform the scanning on the webpage for any possible attack. The authors collect both malicious and benign datasets. The feature extraction method is based on execution time, the page source, and the URL features of the web page. Among the page source features and URL, features are extracted by using static analysis whereas the execution time features are extracted by using dynamic analysis. In this way, the proposed approach extracts 30 features in total. They use 10-fold cross-validation. The experimental results show that this approach was able to detect 99.01% of new malicious code variants. The disadvantage of this approach is that it is concerned only with the known feature extracted only. However, manual selection requires great expertise in the field and it's also unable to detect new kinds of malware.

JAST, developed by *Fass et al*. [15] a system used as a static feature-based mechanism to detect malicious JavaScript by combines the extraction of features from the abstract syntax tree with a random forest classifier to detect malicious JavaScript instances. Even though they were using static analysis only, it yields a high detection accuracy of almost 99.5% and has a low false-negative rate of 0.54%.

Richards et.al. [29] JavaScript program behaviour by analysing execution traces recorded from a large corpus of real-world programs. To collect the trace they used an instrumented version of the WebKit web browser engine integrated into Apple's Safari browser. The instrumented browser records a trace containing most operations performed by the interpreter (reads, writes, deletes, calls, and defines.) as well as events for garbage collection and source file loads. From traced records, they perform offline analysis tools to extracted behavioural information from the traces. From the information, they made nine assumptions about the JavaScript program. Source

code captured when the programs were loaded was analysed to yield static metrics. They compare every JavaScript program to either confirm or invalidate these assumptions.

According, *Xincheng et al* [31] propose a hybrid analysis method for detecting malicious JavaScript code based on a machine learning algorithm. The approach extracts four categories of features for static analysis detection: Code obfuscation, URL redirection, Specific behaviors, and Specific characters. Again, for dynamic analysis, the approach uses three categories of features detection: Function invocation, Data manipulation, and Code structure. Their experiment shows that the machine learning classifier model is based on random forests, and the accuracy is 94.76%.

According to the author, *Wang et al.* [23] propose a detection method that consists of static and dynamic analyses for detecting malicious web pages. They propose the detection method contains two stages. In the first stages, the static feature of the webpages is extracted and according to the pre-set threshold value of prediction, possible webpages are classified into benign and malicious, for those below threshold values preceding the second stage. In the second stage, dynamic analysis stages analysis to determine whether it contains a piece of shellcode which is a malicious webpage. They use the decision tree algorithm used to classify the webpage. 1369 instance webpage collected as the dataset from these 787 instances of benign webpages and 682 instances of the malicious webpage.

Table 2-3 Summary of related works

Title(author)	***Methodology***	***Analysis method***	***Tools***	***Gap***
JSAC: A Novel Framework to Detect Malicious JavaScript via CNNs over AST and CFG () 2019	Uses syntactic and semantic features of JS program. Uses AST and CFG	Static analysis & CNN	Control flow & Esprima	Only feed a portion of AST and CFG It cannot detect the behaviour malicious JS
A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors (*Ndichu et.al*) 2019	Used abstract syntax tree of JS code and plain JS codes. Use LDA (Latent Dirichlet Allocation), LSA (Latent Semantic Analysis), Doc2vec, and N-gram for feature extraction to judge the maliciousness of JavaScript code.	Static analysis & ML method	*AST*	They use plain JS code (i.e removed symbol and special character from JS code) which doesn't represent the structure of the JS code. It's only a static analysis method. It can't detect for behaviour malicious script.
Hybrid Feature Classification Approach for Malicious JavaScript Attack Detection using Deep Learning	Uses execution time, the page source, and URL features (number features used 30). Uses Chi-Square for feature selection.	Static and behavio ur	*HtmlUnit*	Unrealistic behaviour analysis i.e. only execution time feature.

(*khan et.al*) *May 2020*				
Research on Malicious JavaScript Detection Technology Based on LSTM (Fang) 2018	The method utilizes JS engine V8 to parse JS code into bytecode and features are extracted from the semantic level of bytecode. Optimizes Word2Vec model to better train real-valued vectors.	Static analysis & LSTM	V8(Java Script engine)	Bytecode code only Doesn't include the behaviour of JS code
Detecting malicious JavaScript code based on semantic analysis *(Fang et.al*) *February 2020*	Use syntactic unit sequences of JS code with FastText algorithm to create training vectors.	Static analysis	*AST*	Use AST only i.e behaviour analysis doesn't include.

Summary of related work:

Based on the research works reviewed we have summarized all JavaScript-based attack classification to identify their gaps and to suggest a solution that enhances them. A lot of research works have been done which can be applied to JavaScript-based analysis. But the related works we reviewed showed the analysis method, feature extraction method, and difficulty for behaviour malicious script is the major issue.

Besides, most of the works have been done using a static analysis-based classification approach that can only detect known attacks. Based on this observation, we select a JavaScript-based attack classification by using the behaviour and static analysis method. As a feature extraction, we used the word2vec method to create the feature vector of the datasets used for the machine learning model. Using those methods we enhance the performance, behaviour malicious script classification, and feature extraction methods. Table 2-3 the summary of the related work we addressed in the literature reviews and also the gap we found in the previous works.

CHAPTER THREE

3. Research Methodology

3.1.Introduction

This chapter discusses the research methodology and the process of developing a JavaScript-based attack classification. It begins with a methodology for how the data were collected and processed alternatively. The following section in this chapter explains and justifies the methodology used in conducting the study.

To attain the objectives of the research, an in-depth literature review on malicious JavaScript-based attack classification and related works was done. Books, articles, thesis papers, electronic materials, and different open-source tools were used to collect the necessary information for this study. Based on the information obtained from the reviewed literature, the input analysis, underlying representation, and output representation techniques and tools are selected. The techniques and tools are chosen by considering various factors like their effectiveness in similar previous works.

The data collection starts with determining what kind of data are needed to be followed by techniques of information collection. In this research, the following data categories collected:

- ***Malicious JavaScript code***: the malicious term is used to show a desire to cause harm to someone [74]. Malicious JavaScript code is used to describe code in any part of a web script that is intended to cause undesired effects, security breaches, or damage to a system [75]. Therefore, malicious JavaScript is any code that has the intention of install malware, exploiting a browser security hole, capturing sensitive data, or impersonating something without the user's knowledge, consent, or realization. For that reason, in this study, we use malicious JavaScript code to point out the harmful JavaScript code.
- ***Benign JavaScript code*:** the benign term is used to showing something good, kind, or not dangerous [76] which means it is not likely to be harmful. Therefore, benign JavaScript code in this study strongly suggests that JavaScript code does not cause damages.

In general, to accomplish the above objectives and to answer the research questions, the following methods and techniques are followed. The following steps are mentioned as a research methodology.

- Formulating the problem by reviewing different literature and documents.
- Collecting data related to JavaScript-based attacks.
- Mention the research methodologies such as literature reviewing, data collection method, the proposed framework design method, and implementation tools used for the research work.
- Implementing the proposed framework.
- Finally, evaluating and testing the functionality and usability of the proposed framework.

3.2.Building Dataset

The main objective of this study is to analysis JavaScript based-attack classification using a support vector machine algorithm. Therefore, the dataset is required to achieve the objective of the study. To do this the first thing is to collect the JavaScript code by crawling different webpages using python programming languages. The dataset contains both malicious and benign JavaScript samples and the below Figure 3-1 shows the overall processes of building the datasets for this study.

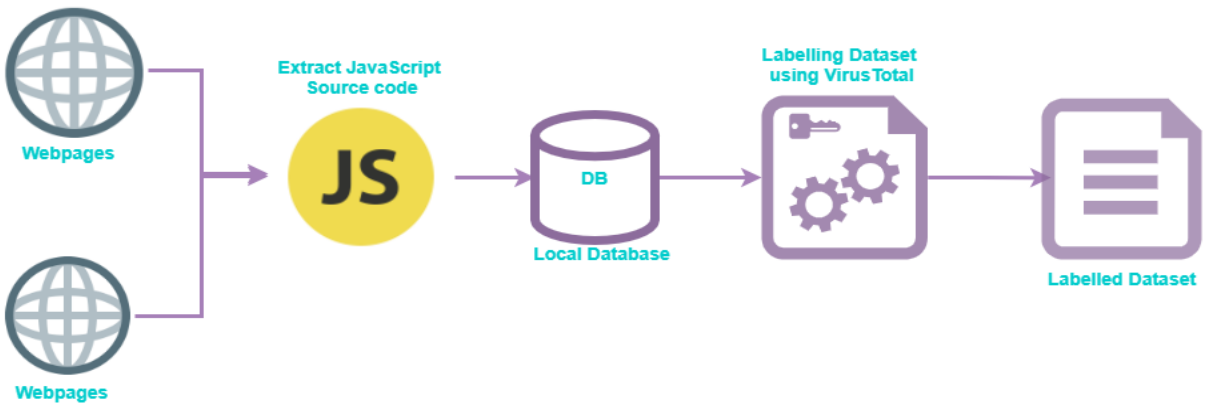


Figure 3-1 Method for building malicious and benign JavaScript datasets

3.2.1. Data Collection

In this study, the dataset is composed of both malicious and benign JavaScript code sample, which is labeled as *1* and *0* respectively. That instance sample is collected by using python IDE JavaScript code is extracted from the websites. For a normal instance(benign) datasets collected from Alexa [77] top website store a collection of non-blacklist URL websites. Alexa Internet, Inc. is an American web traffic analysis company. Its toolbar collects data on Internet browsing behaviour and transmits them to the Alexa website, where they are stored and analysed. According to its website, Alexa provides web traffic data, global rankings, and other information on 30 million websites. The malicious samples were collected from GitHub [78] and the publicly available data [79] that is the most widely used and freely available dataset.

After the dataset is collected, labelling the dataset as malicious and benign is essentially important. To label the dataset we use VirusTotal an antivirus product and scan engine. In this study, we use VirsuTotal APIv3 to label the dataset as malicious or benign JavaScript using the python programming language.

3.2.2. Dataset Preprocessing

The main reason why we need data preprocessing arises from the fact that redundant data and insignificant features may be unclear to the classification algorithm and this may lead to inaccurate and less generalizable models. Usually, the data collected from different sources may not be in a format that is suitable for a machine that may contain invalid and/or missed values. Machines need specific representations of the input data for both training and testing which are termed features before a feed to an ML algorithm.

Consequently, after we collect the datasets and the labelling process completed the next step we did was preparing the dataset for the model. The preparation process follows cleaning and removes duplicate data from the collected dataset. This makes the dataset ready for the feature extraction method. In this study, we use static analysis and behavioural analysis methods for feature extraction. We apply Abstract Syntax Tree (AST), which is a static-based analysis, on the dataset to prepare the dataset for the model. ASTs convert the JavaScript source code into a tree format through the JavaScript parser. To parsing JavaScript code, Esprima open source tools were used. Esprima parser takes a string representing a valid JavaScript program and

produces a syntax tree, an ordered tree that describes the syntactic structure of the program. By using that method converts JavaScript source code into a tree structure in JSON format. The resulting syntax tree is useful for the feature extraction method. The whole process is called static code analysis.

The other data preparation for the models is pre-processing datasets with cuckoo sandbox analysis to get the behavioural analysis of JavaScript. Cuckoo Sandbox is the open-source used for behavioural analysis and generating analysis reports by executing the sample files in the virtual (isolated) environment. Using this method we got the runtime information of JavaScript source code in the JSON formats. Therefore, we use Cuckoo Sandbox to observe the runtime behaviour for the datasets.

In general, We select the above method to depend on the advantages of the method and previous literature. We used static analysis because it can analyse the source code without executing and also uncomplicated to implement the method. For behaviour analyses, we select it because it reduces the drawback of static analysis i.e. static method has a high false-positive rate as we have said in the literature reviews. We select cuckoo sandbox because it is open source code and trace all activity of source code at run time. This is used especially when JavaScript has hidden their activities by using obfuscated techniques.

3.3.Feature Extraction Method

Machine learning algorithms learn the patterns from fixed-length feature vectors [19], and therefore feature extraction is the first step to get the feature vectors. Making use of more features to run machine learning algorithms tends to make models more complex and difficult to interpret. Besides this, it can often lead to the model's overfitting on the training data. This led models to perform very poorly on new, previously unseen data. In addition to this, having irrelevant features in a dataset can decrease the accuracy of models and takes much amount of time to train models. Therefore, we have to select an optimal number of features to train and build models that can generalize very well on the data, with less time of computation.

Consequently, after the dataset preprocessed which describe under the data processing section, the following methods are used to extract features from preprocessed data that would help to identify malicious and benign JavaScript code from the dataset and could be used in the

modeling of the detection problem. The following feature extraction methods are used for modelling malicious JavaScript-based attack classification.

Word2Vec: *it* is one of the most popular training tools for extracting a feature vector from the datasets by transforms the text content into a vector. The vector is representing in the k-dimensional space and uses the vector space to represent the semantic similarity of the text.

Word2vec models have reached modern issues for text classification and clustering tasks on various datasets. As such, since a preprocessed JavaScript datasets also contain textual data, we use these models for constructing the feature vector for both static and behavioral results that are used for training the machine learning models to classify malicious and benign JavaScript.

The word2vec primarily has two components training models: the Continuous Bag of Words (CBOW) and the Skip-gram model shown in Figure 3-2 below. Both Skip-gram and CBOW models are creating a feature vector from a given dataset. CBoW is to predict a target word from context. Skip gram is the reverse of bag-of-words that is to predict content from the target word. CBOW is trained to predict a single word from a fixed window size of context words, whereas Skip-gram does the opposite, and tries to predict several context words from a single input word.

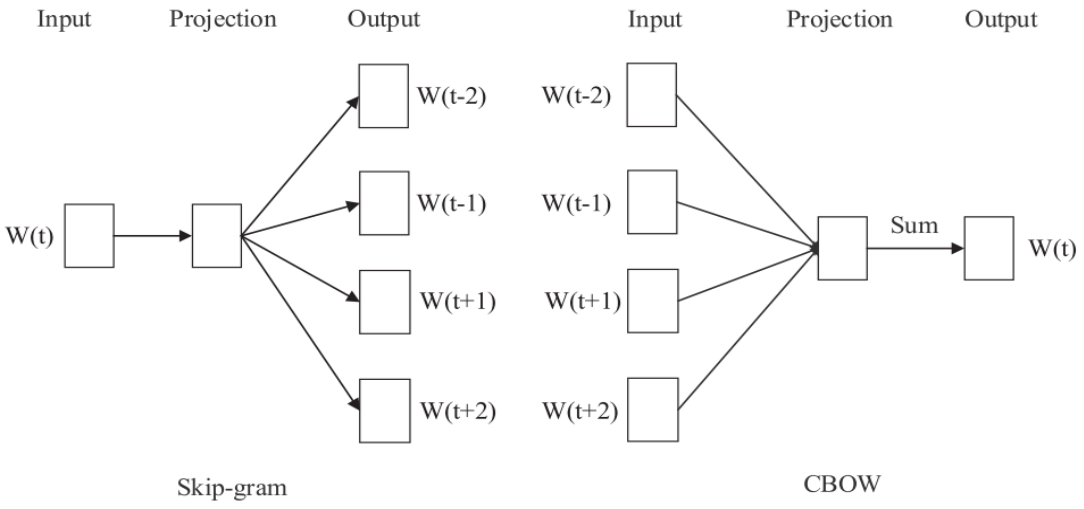


Figure 3-2 Skip-gram Model (left) and CBOW Model (right) [38]

***Doc2Vec*:** it is the most known feature extraction method especially for text documents which means converts the document into feature vectors. The input of word per document can be varied while the output is fixed-length vectors. It is used for many purposes like document retrieval,

web search, spam filtering, and topic modelling. Doc2vec is creating a numeric representation of a given document. As such the result of both analysis have text format we used this feature extraction method to create a feature vector for a given dataset. Doc2vec contains two important models as a Distributed bag of word and Distributed memory.

In the distributed bag of word each text is mapped to a unique paragraph vector and each word is mapped a unique word vector in a continuous space. It uses a paragraph vector to classify entire words in the document. In distributed-memory, each word vector and paragraph vector is concatenated to create feature vectors as shown in below Figure 3-3.

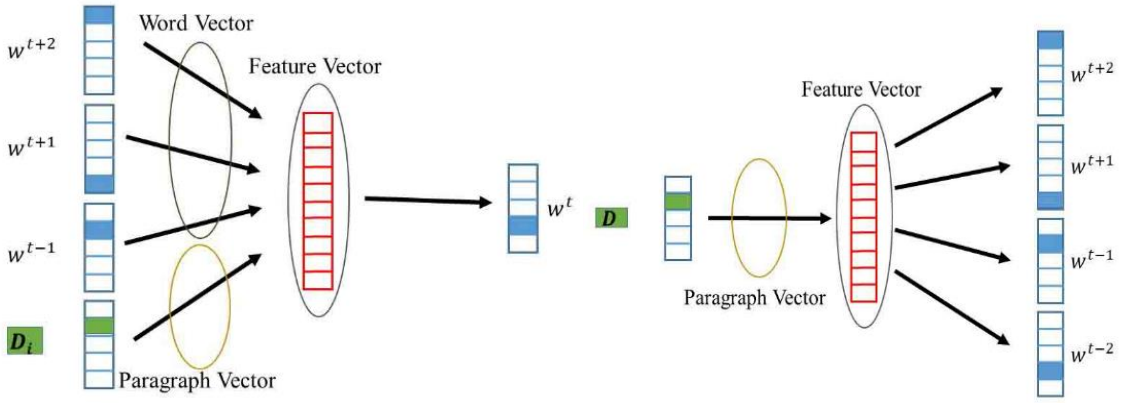


Figure 3-3 (a) Distributed bag of words (b) Distributed memory model [66]

***TF-IDF*:** it is one the text feature extracted method which works by determining the relative frequency of words in the document compared to the inverse proportion of that word over the document corpus. In our case, TF-IDF constructs the feature vector from both analysis methods we raised in the preprocessed section.

3.4.Machine Learning Algorithm

As a known machine learning model learns from data and can give an accurate prediction for a given input after correctly conducting learning. Usually, raw data need to be transformed into a proper feature representation for inputs to attain high-performance prediction.

To achieve the goal the training dataset is supplied to the machine learning model. In this study, we used *a Support Vector Machine* (SVM) for classification. The algorithm is selected because of it is used for classification problems and solving malicious script detection problems on the recommendations of the previous research work on the cybersecurity area [81]–[84].. Another primary advantage of SVMs is the low expected probability of generalization errors [80]. SVM algorithm performs classification by finding hyperplane in n-dimensional space that separates the classes or data points in feature space. Hyperplanes are decision boundaries that help classify the data points. In SVM, the main aim is to find the optimal hyperplane that maximizes the margin. SVMs classify data by determining a set of support vectors, which are members of the set of training inputs that outline a hyperplane in the feature space [80].

SVMs are relatively unaffected by the number of data points and the classification complexity does not depend on the dimensionality of the feature space, so it can potentially learn a larger set of patterns and thus be able to scale better than neural networks. Once the data is classified into two classes, a suitable optimizing algorithm can be used if necessary, for further feature identification, depending on the application.

3.5.Evaluation Techniques and Model

This subsection deals with evaluating a machine learning approach and improving the used model by optimizing parameters. The machine learning classifier was evaluated by applying both datasets. This technique aims to forecast and estimate how accurate a classification model will work in practice. To evaluate the accuracy and performance of the developed machine learning model this study uses different model evaluation metrics appropriate like F-score i.e. recall, and precision. Also, k-fold cross-validation techniques increase the evaluation reliability of the classifier by partitions the original whole dataset into *k* equally-sized folds, from which *k-1* folds are operated together as a training dataset and *k* fold is used as a test dataset. Again, use the confusion matrix as a metric to evaluate classifier performance. The matrix enables the assessment of the result of false negatives and false positives.

After all evaluation results will analyse and a conclusion will draw for suggesting a better approach to classify accurately. These values are the primary metrics for binary classification and can be depicted in a called confusion matrix.

$$\begin{bmatrix} True\ Negative(TN) & False\ Posative(FP) \\ False\ Negative(FN) & True\ Posative(TP) \end{bmatrix} \quad \text{Equation 3.1}$$

- **True Positives (TP)—**are the cases when the actual class of the data point was true and the predicted is also true.
- **True Negatives (TN)—**are the cases when the actual class of the data point was False and the predicted is also false.
- **False Positives (FP)—**are the cases when the actual class of the data point was False and the predicted is true.
- **False Negatives (FN)—**are the cases when the actual class of the data point was true and the predicted is False.

The actual accuracy measurement can be defined by using the following equation.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$ Equation 3.2

Precision measures how many positive predictions are positive. The recall is used as measurements if the avoidance of false negatives is required. The definition of both measurements depicted in the following equation.

$$precision = \frac{TP}{TP + FP}$$ Equation 3.3

$$recall = \frac{TP}{TP + FN}$$ Equation 3.4

A harmonic mean of the precision and recall is called the f1 score. It is defined as

$$f1 = 2 * \frac{precision * recall}{precision + recall}$$ Equation 3.5

CHAPTER FOUR

4. Proposed JavaScript-based Attack Classification

4.1.Overviews

In this chapter, the proposed javaScript- based attack classification was discussed in different ways and the high-level architecture of the proposed solution is presented in detail to show each component that exists within the proposed framework.

4.2.The Proposed System Architecture.

The proposed architecture has includes five components: behavioural analysis, static analysis, Feature Engineering, model build, and the last is model evaluation. The proposed solution experimented on both malicious and benign JavaScript source code. In the behavioural analysis, behaviour analysis is done and the result is used for the next modules. We used the cuckoo sandbox system to get the behavioural analysis for our datasets. The function of the cuckoo sandbox in this proposed solution is to generating the behavioural report of the dataset at runtime environment which to easily understand the JavaScript source code activities at execution. Finally, the proposed solution architecture is detailly pointed out in Figure 4-1.

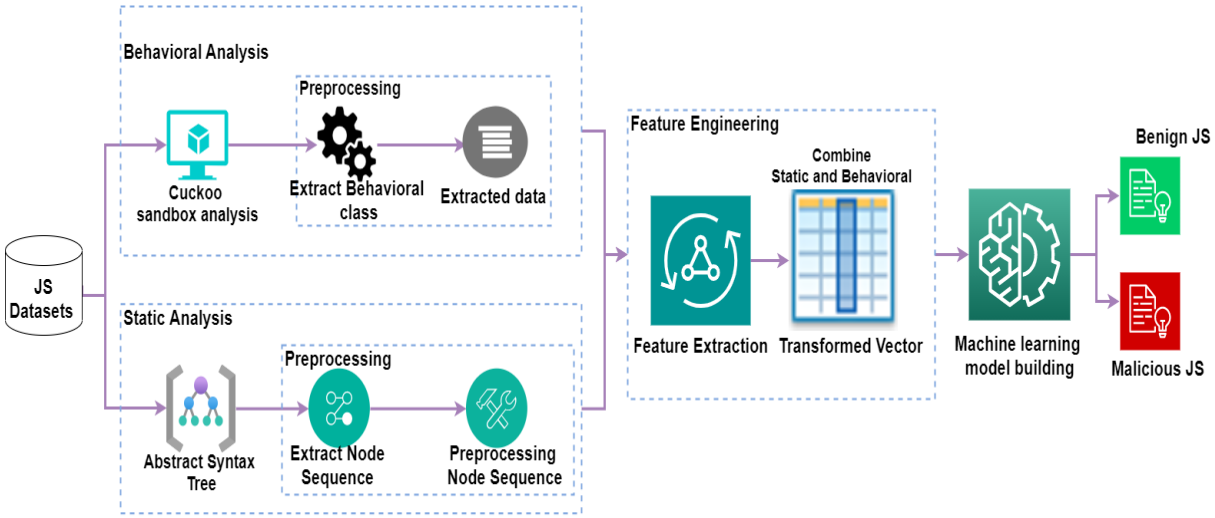


Figure 4-1 Overviews of the proposed architecture

The above figure 4.1. shows that in the static analysis, the static analysis of source code is done. We used an Abstract syntax tree (AST) which represents source code in the tree formats, for static analysis modules. The other works done in the module is extracting the node sequence of AST. For this purpose, we used Nodejs module packages called AST-Traverse, it is used for traverses an abstract syntax tree and performs transformation at each of its Nodes of AST. So, AST-Traverse returns the node sequence of the AST.

The results of both static analysis and behaviour analysis needed to pre-process to increase the performance of our model. The main reason why we need data pre-processing the result of both static and behaviour analysis results is to arise from the fact that redundant data and insignificant features may be unclear to the classification algorithm and this may lead to inaccurate and less generalizable models. We used the *Python* programming language for static and behaviour analysis pre-processing.

In the next feature engineering is carried out. In this component work done is about extracting relevant features for our classification machine learning model. We apply the feature extraction on the datasets separately for static and dynamic analysis. After feature shape, we combine both features by using *scipy.sparse.hstack()* of sklearn modules. The main aim of combined feature vectors is to increase the accuracy of the model.

The next component in the proposed system architecture is building the model with the relevant feature we get in the previous feature extraction methods. So, to build the model first we trained the model over a set of features extracted under the feature engineering module to recognize a certain type of pattern. Once the model is building we can use it to reason over data that it hasn't seen before, and make predictions about those data.

Finally, the evaluation component is done. With this evaluation component, we evaluate the performance of the model by using evaluation metrics and methods.

4.3.The Proposed Analysis and Feature Engineering Method

As discussed in detail in the literature part(see section 2.2), there are different methods we mentioned for the analysis of JavaScript code. So, we select the two most popular method for JavaScript code analysis which is behaviour and static analysis module.

4.3.1. Static Analysis Method

Static analysis is a method that is used for debugging by examining source code before execution. To implement the static analysis we used the abstract syntax tree. The abstract syntax tree is a tree representation of the abstract syntactic structure of source code [37]. It represents the necessary and functional structure of source code such as scopes, expressions, or declarations while at the same time avoiding redundancy by omitting unnecessary syntactic details [5] such as whitespaces, punctuation marks, or comments which do not affect the original function of code even after the parsing process.

The abstract syntax tree is often used to generate intermediate code by applying some transformations. The simplest way to generate intermediate code using an abstract syntax tree is to traverse the source code and generate a node sequence which may be used as the intermediate code.

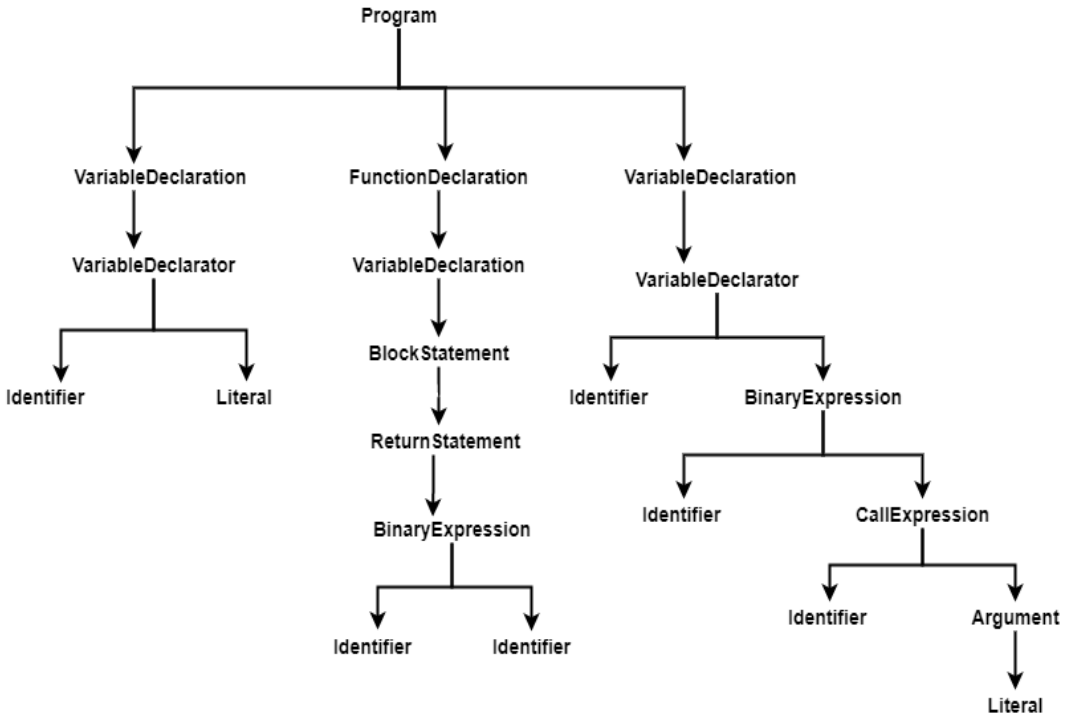
In this study, we used Esprima to parse the JavaScript source code into an abstract syntax tree i.e to generate ASTs from JS code. Esprima is a high-performance JavaScript parser that takes a string representing the JavaScript program and produces a syntax tree. Esprima can produce 67 different syntactic entities, which range from *ArrayExpression* to *YieldExpression*.

Figure 4-2 Representing the dataset using AST

Figure 4-2 shows that how can convert the malicious and benign JavaScript datasets into the abstract syntax tree by using the esprima parser techniques and it's output is a node sequence. The output of Esprema perser were saved in the JSON (JavaScript Object Notation) format and the esprima parser generator is required to produce an abstract syntax tree. The following figures show that how we implement the static analysis method on the sample code.

```
1  var a = 42;
2
3  function addA(d) {
4      return a + d;
5  }
6  var c = addA(2) + a;
```

(a)



(b)

Figure 4-3 JS source code sample (a) and AST Produce form source code sample (b)

Figure 4-3 shows the sample source code and its abstract syntax tree. When we apply the esprima parser on the source code (a) it immediately generates the tree form (b). Since the abstract syntax tree is tree format we can traverse over the node tree easily by using the *ast-traverse()* and extract node sequences from the tree. After the node sequence is traversed it is output saved in the local disk as '.txt' formats that are used for the next feature engineering modules.

4.3.2. Behaviour Analysis Module

Behavioural analysis is the method used to analyze the source code under virtual (isolated) environments for tracing all activities that happened during the run time environments. We use the cuckoo sandbox for behaviour analysis, hence, using the cuckoo sandbox we create the virtual environment to executes and monitor the JavaScript datasets to extract the behaviour information about the executed datasets.

Cuckoo sandbox can analyse many other different files not only JavaScript as discussed in chapter two section 2.4 and also cuckoo can analyse malicious web-sites in a virtualized environment. Therefore, the cuckoo can trace all activities and general behaviour of our dataset files. The reports generated by the cuckoo sandbox, describing the behavioural data of each dataset sample. The report generated by the cuckoo sandbox needs to be pre-processed, and the features were extracted from there by using text-based techniques.

Cuckoo sandbox is configured to execute the file and produce the output in the form of a JSON report. These reports contain the behavior of the program executed in a controlled environment of the cuckoo sandbox. In this study, a collection of both malicious and benign JavaScript source code is executed on the cuckoo sandbox and the result of the cuckoo sandbox is used as the input for the next feature extraction module.

Figure 4-4 shows how the beahvioural analysis is operated in the proposed system. We use python IDE to start the cuckoo sandbox on the virtual machine, later the execution is completed and the report generated is preprocessed. Finally, the extracted data is stored in the '.txt' formats.

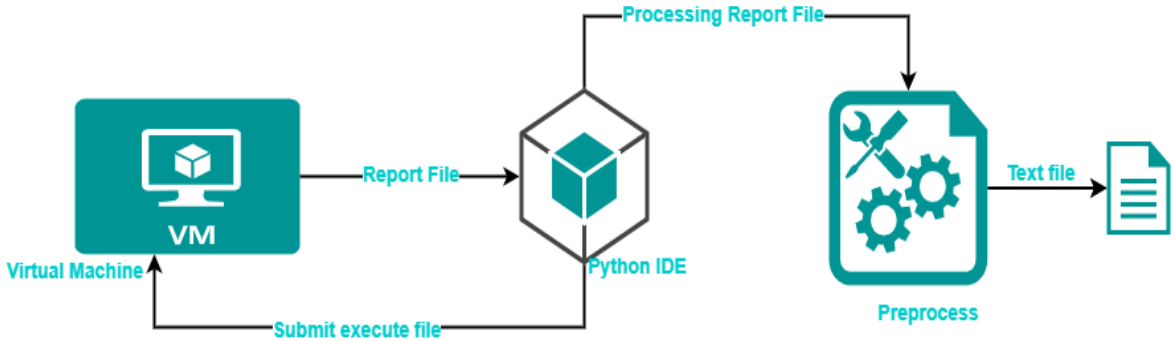


Figure 4-4 Overview of behavioural analysis.

The reports of the cuckoo sandbox contain multiple sections as discussed below in Figure 4-5 and each section has different information about analyzed JavaScript files. From those sections of the report, we select the *behaviour section* because we focus on the behavioral analyses only. The behaviour section has different five-section such as *generic section, apistat section, processes section, processtree section, and summary section.*

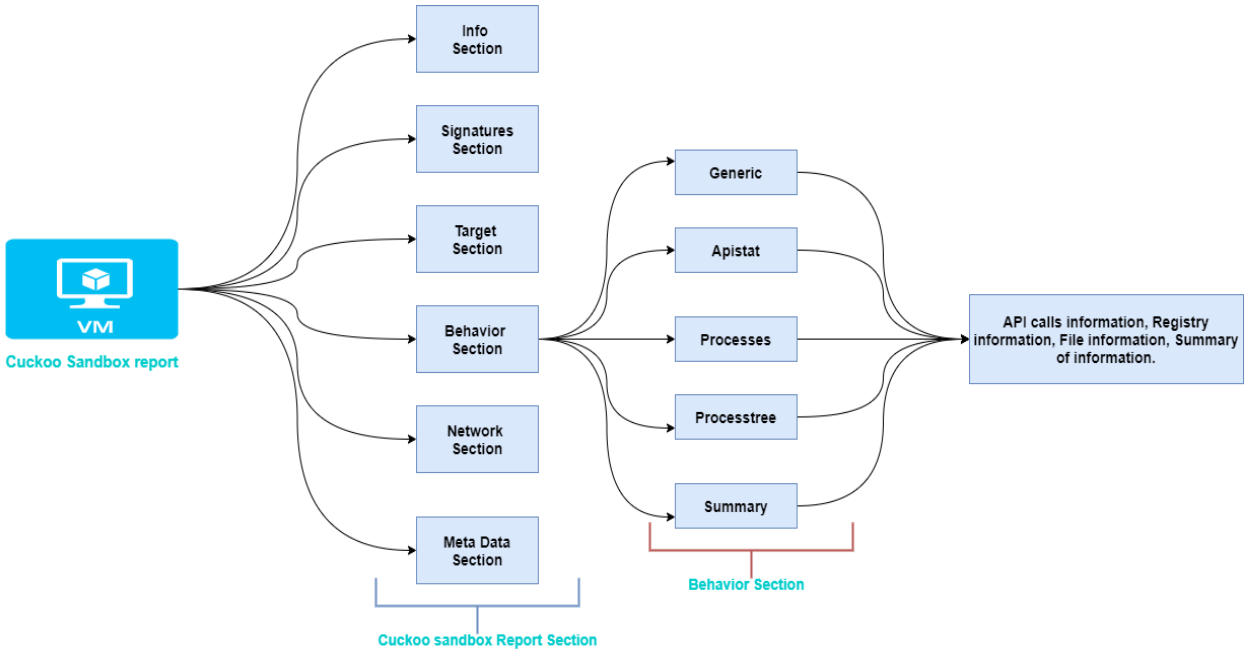


Figure 4-5 Cuckoo Sandbox report section

4.3.3. Feature Engineering Methods

The pre-process is very important to enhance model performance during feature learning. The big problem of malicious classification is extracting the relevant feature for malicious activities from the datasets. The most researcher uses manually method for extracting the feature but this needs the skill and takes a lot of time to process the source code line by line from a long code of line to get the features. To address the problems in the existing feature extraction methods such as low execution efficiency and complex feature extraction in the behaviour analysis and static analysis respectively. In this study, we used the text-based feature extraction method such as word2vec, doc2vec and tf-idf, because the output of static and behaviour analysis is text formats.

The output of both behaviour analysis and static analysis is text-based files, so, this is followed by using a text-based feature extraction method to extract information from the file. We used TF-IDF, Doc2Vec, and Word2Vec (discussed in section **4.4.1**) as feature engineering methods. The sample of text datasets we used is not a natural language but it is a list of AST nodes sequence and reports of the behaviour analysis. Therefore, there is no stop word like the natural language environment. That is, each word in the dataset provides useful information. Each method gives feature vectors used to train machine learning classifiers.

4.4.Machine Learning Model Build

A explained in proposed architecture for JavaScript-based attacks designed in Figure **4-1**, the proposed system architecture consists of different modules. The first activity to be processed is to prepare the datasets for model building by loading the datasets from the local disk and the next, feature engineering carries out on both static and dynamic output results to extract the feature in the form of the features vector. Thus Each JavaScript source code is represented as feature vectors. Finally, the classification machine learning model is trained on all feature vectors constructed by feature engineering that maps the JavaScript code with their features to the target class by using learning algorithms.

Machine learning uses feature vectors created in future engineering and machine learning learns the patterns from the feature vectors of the dataset to classify JavaScript source code. finally the model ready for classification hence, this study builds a classification model mainly by using a

machine learning classifier algorithm that would be selected for JavaScript-based attack detection like the SVM algorithm.

4.4.1. Feature Extraction

Feature extraction was performed for obtaining the feature data to predict the class of datasets. The behaviour and static analysis return the result in the form of text formats. And as we have raised in the literature review section (see section **2.6.1**) there is a different method to extract the feature vector. From those methods, we select the Word2Vec, Doc2Vec, and TF-IDF feature extraction methods.

4.4.1.1. TF-IDF Feature Extraction

The TF-IDF feature extraction method was used in the experiment to get the word frequency in the dataset by applying the term frequency and the importance of the word in the datasets that represented by measuring the inverse document frequency of the word in the dataset. This featured model gives the classifiers of frequency and the importance of the word in the dataset as a feature vector for training the machine learning model.

The TF-IDF score for the word t in document d from the document set D is calculated as the term frequency (tf), the number of times a term occurs in a given document, is multiplied with the inverse document frequency (idf) component, which is computed as follows:

$$tfidf(t,d,D) = tf(n,d) * idf(t,D) \qquad \textit{Equation 4.1}$$

Whereas tf(n,d) is the term frequency and idf(t,D) if the inverse document frequency.

TF-IDF features are created for both static and behaviour analysis separately. Later the feature is shaped by using the *NumPy* package to concatenate the feature into single feature vectors that are used for the machine learning model.

4.4.1.2. Word2Vec Feature Extraction.

The word2vec method performs word to vector modeling on a larger amount of data that come from static and behaviour analysis to build the model. Word2Vec produces word embeddings based on the contextual semantics of words in a text (based on the context that the word occurs

in). These models are used to extract features from the text in the dataset by calculating the average of all vectors using the model.

Word2vec creates vectors that are the numerical representation of word features in the JavaScripts datasets. After preprocess static and behaviour analysis output we apply word2vec to create the vector space model for each datastes. Each distinctive word in the corpus is allowed with a corresponding vector in the space. Subsequently, the feature space is created for each static and behaviour analysis to build the model, hence the feature space is needed to combine to get the single vectors. We use the *Numpy* library to concatenates both feature vectors created separately by word2vec.

4.4.1.3. Doc2Vec Feature Extraction

Doc2Vec is an extension of Word2Vec that is applied to a document as a whole instead of individual words. It operates on the logic that the meaning of a word also depends on the document that it occurs in. The vectors generated by Doc2Vec can be used for finding similarities between documents. Later the vector is created for each static and behaviour dataset we mixed using *NumPy* library to get a single vector that is used for model builds.

4.4.2. Build Model

A support vector machine was used as a classifier, we select SVM because it is more accurate and more robust i.e. due to the optimal margin gap between separating hyperplanes, it could do predictions better with test data. In addition to that, it is also computationally more efficient. To builds the model we training on all feature vectors constructed by the feature extraction components methods. This study builds a classification model mainly by using a machine learning classifier algorithm SVM on the dataset. The process of modeling is to find patterns in the training set of the dataset that maps the dataset with their features to the target class by using learning algorithms. The output of these modeling is a trained model that can be used for the classification of javascript-based attacks, making predictions on new javascript-based attacks. In the experiments, we use training and testing datasets: train-test split with 0.3 values i.e. 70 for training and 30 for testing, and ten-fold cross-validation to evaluate our model as a shown Figure 4-6.

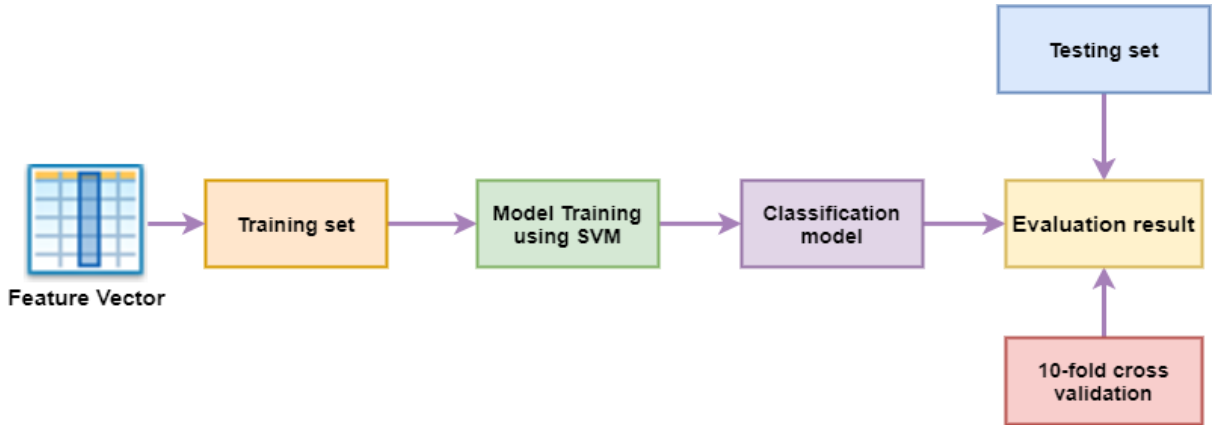


Figure 4-6 Model building flow diagram

4.4.3. Models Evaluation and Validation

Evaluating a model is a very important step that is performed throughout the development of the model. However, we still evaluate a model through various methods. It is important to note that models can be evaluated successfully when working in a supervised learning environment as the actual values are available for the evaluation methods to function.

To test and estimate the learning ability of machine learning models. The basic concern of the machine learning model is to find an accurate estimation of the generalization error of the trained models on finite datasets. Hence we used train-test split methods and K-fold cross-validation (CV) methods. In train-test split the dataset is splited into two parts: training dataset used to train the model and test datasets used to evaluates the model. In cross validation we used 10 folds i.e. the data sets are randomly partitioned into 10 equal-size parts. At each iteration, one single part is taken as the testing data, and the other parts are used as training data. The final result is the overall average of each iteration. For performance evaluation, we used different metrics that appropriate for models such as confusion matrix, accuracy, precision, recall, and F-Measure, as we discussed in chapter two section 2.6.3.

CHAPTER FIVE

5. Implementation of the Proposed Framework

5.1.Overview

This chapter describes the implementation of the proposed framework for the analysis of JavaScript-based attack classification using the Support Vector Machine Learning Approach and further modules were discussed in the following sections.

5.2.Working Environment

This study used several development tools and packages to address the proposed solution for the analysis of JavaScript-based attack classification using the Support Vector Machine Learning Approach. We use the *python programming* language for implementing and experimenting with each proposed solution from the data pre-processing to the model building phase. Also, to evaluate the implemented proposed classifiers model. Hence python is the programming language of choice for developers, researchers, and data scientists who need to work in machine learning models. The other programming language used in this study was *Node.js* to processes the JavaScript source code. In general, the following are the tools and package/library we are used in this study.

Table 5-1 Tools and library used in study

Tools		
Tools	Version	Description
Anaconda Navigator	1.9.6	Allows us to launch a development application and easily manage conda packages, environment, and channels without the need to use command-line commands.
Python	3.7	Easy to learn, powerful programming language to develop a machine learning application.

API	2.0	VirusTotal API is used to check the dataset for the process of verification.
ECMAScript		Open source tools which allow various code analysis and operation such as editing, transformation, visualization, and validation. It can be used to perform lexical analysis (tokenization) or syntactic analysis (parsing) of a JavaScript program.
Cuckoo Sandbox	2.0.7	Open source tools are used for malware analysis in an isolated environment. We used the cuckoo sandbox for behavioural analysis module.
Node JS	14.15.0	Node.js is an open-source platform that executes JavaScript code outside the web browser. We used for abstract syntax tree from the dataset.
Python Packages		
Scikit-learn	0.20.1	A set of python modules for machine learning and data mining. This study uses it for feature extraction and training and testing model.
Pandas	0.24.2	High-performance, easy-to-use data structures, and data analysis tools. This study uses it for data reading, manipulation, writing, and handling the data frame.
Genism	3.4.0	Python library for topic modelling document indexing and similarity retrieval with large corpora. This study uses it for the constricting word2vec model.
Matplotlib	3.0.3	Publication quality figures in python. This study uses it for data and results visualization.

5.3.Deployment Environment

The tools which are discussed above in section 5.2 have been deployed on a desktop computer equipped with a processor Intel® Core™ i5-6500 CPU @ 3.20 GHz 2.0 GHz(Giga Hertz), 8 Gigabytes of physical memory. The operating system is Windows 10 Pro, 64 bits system type x64-based processor.

5.4.Analysis Implementation

5.4.1. Implement Abstract Syntax Tree

To implement the pre-processing method, we used Node.js programming language and Esprima modules. To perform the whole implementation of the methods, we perform the following common activities, which are importing important libraries packages and load the dataset file to the disk using the code. *Figure 5-1*. Shows below, the dataset is loaded using file operation of Node.js, then after esprima parser is applied on loaded dataset and store the output of esprima parser in the form of the JSON format. After the abstract syntax tree is extracted from the dataset, the next step is to traverse over AST. Since the AST is tree format we can traverse over the tree easily by using the *ast-traverse* node js module. The function of the ast-traverse is to traverse over the AST node.

```
var content
const path = require('path')
const fs = require('fs')
var esprima = require('esprima')
const directoryPath = path.join(__dirname, 'Data/test')
fs.readdir(directoryPath,function(err,files)
{
    if(err)throw err
    files.forEach(function(file){
        fs.readFile('Data/test/'+file,'utf8',function(err,data){
            if(err)throw err
            content =esprima.parseScript(data)
            fs.writeFile(file, JSON.stringify(content), function(err){
                if (err) throw err;
            });
        });
    });
});
```

Figure 5-1 Abstract syntax tree implementation using esprima

5.4.2. Implement Cuckoo Sandbox Setup

To implement the cuckoo sandbox configuration we used a Desktop computer as an analysis machine using VirtualBox. The Ubuntu operating system was installed on the analysis machine. The Cuckoo Sandbox application has been installed host machine. Configuration setup for Cuckoo sandbox in the host is done accordingly to the Virtual machine which will be used for the execution of the sample. The analysis machine is run as a virtual server, where malware will be run and analyzed. Windows operating system is installed on this server. The Cuckoo agent is installed in the guest machine in the start-up menu. The firewall has been turned off, UAC (User Account Control) turned off, and operating system updates have not been applied to prevent any obstacles during the operation of malicious software.

Virtual Box only Adapter (Vboxnet0) were used for communication between Virtual machine and Cuckoo host and again NAT adapter is used for communication between Cuckoo guest and the internet. To start analysis on any file the cuckoo agent called Python script *agent.py* is executed with root privilege on the cuckoo host. Once the Cuckoo host started, we can submit files for analysis in a Virtual machine as specified in Cuckoo configurations. Cuckoo sandbox executes the files in the Virtual machine in the clean state and monitor each action happening in the virtual environment to the Cuckoo host and generates a report for each sample whenever a sample is submitted.

To implement the analysis activities on a virtual machine different pre-requirement necessary to relive the cuckoo sandbox. Cuckoo requires a minimum of two computer systems to function: one acting as host and the other one as guest. In this study, we used a virtual machine as a guest computer.

Table 5-2 List of tools and package for cuckoo sandbox

No.	Tools and Packages	Version
1.	Ubuntu	18.04
2.	Virtual Box	5.1.22
3.	VBoxGuestAdditions	5.1.18
4.	Cuckoo Sandbox	2.0.7
5.	Python	2.7.1
6.	Pillow	5.1.0
7.	Windows 7 SP(Guest)	-
8.	MongoDB as a web reporting interface	
9.	PostgreSQL as database	
10.	Yara	
11.	Distorm3	3.6.3
12.	ssdeep	2.14
13.	Pydeep	0.2
14.	TCPDump for packet analysis	-

Table 5-2 Shows that the tools and package with thear version used for analyzing the dataset on the cuckoo sandbox.

Configure VirtualBox.

The above tools and packages are installed from the Ubuntu terminal. Now we have to configure cuckoo after installing it. But also VirtualBox and Networking part need to correctly setups. Before configuring any VirtualBox network interface. Now we create a *Host-Only Adapter* by running *vboxmanage hostonlyif create.* After create successfully set the IP address for the *vboxnet0* interface by running:

vboxmanage hostonlyif ipconfig vboxnet0 --ip 192.168.56.1

Therefore, *vboxnet0* configure with the above *ip* address. Subsequently, the *vboxnet0* interface was configured successfully. Next, create the VirtualBox with a Window 7 SP OS image

because Window 7 SP works best for malware analysis and configure the VM with a *Host-Only Adapter*. The below figure shows the configuration of the VM. Now it is time to customize the Window 7 SP1 VM and make it ready for the cuckoo. The following steps we used to configure the Windows guest virtual machine.

Step1—Configure Ethernet Network Adapter IPv4 setting with the following parameter

IP address – 192.168.56.1

Subnet-mask –255.255.255.0

Default gateway –192.168.56.1

DNS Servers –8.8.8.8/8.8.4.4

Step2—Disable windows update, UAC (User Account Control), and Windows firewall.

Step3—Install *python 2.7* for windows.

Step4—Upload the *agent.py* file from the cuckoo host which can be found in the *cuckoo/agent* directory.

Configure Cuckoo

Cuckoo relies on a couple of main configuration files. The following table shows the main necessary file for the cuckoo sandbox which exists under the */opt/cuckoo/conf* folder.

Table 5-3 Cuckoo configuration file

Filename	***Description***
cuckoo.conf	This configuration file contains information about the general behaviour and analysis options in Cuckoo Sandbox.
processing.conf	This file is used for enabling and configuring the processing of modules.
reporting.conf	This file contains information about reporting methodologies.
auxiliary.conf	For enabling and configuring auxiliary modules.
virtualbox.conf	For defining the options for your virtualization software.
memory.conf	Volatility configuration.

The following are the cuckoo file configuration we used in this study.

Table 5-4 Cuckoo file configuration used in this study

Filename	***Configuration part***
cuckoo.conf	memory_dump = yes machinery = virtualbox [resultserver] ip = 192.168.56.1
auxiliary.conf	[sniffer] enabled = yes
virtualbox.conf	[virtualbox] mode = gui machines = cuckoo [cuckoo] label = cuckoo platform = windows ip = 192.168.56.101 snapshot = Snapshot1
processing.conf	[memory] enabled = yes
memory.conf	[basic] guest_profile = “Win7SP1x64”

reporting.conf	[singlefile]
	# Enable creation of report.html?
	enabled = yes
	[mongodb]
	enabled = yes

Finally, the cuckoo is ready to use. Update cuckoo scoring signature by typing the command *cuckoo community*, then to start *cuckoo* and a *Django* web interface by typing *cuckoo* and

cuckoo web in two different Ubuntu terminal windows. Next, go to the *localhost:8000* in the browser to get the web interface of a cuckoo. This is where we can start uploading files and using cuckoo to analyze the suspicious file. The below figure shows the web interface for the cuckoo.

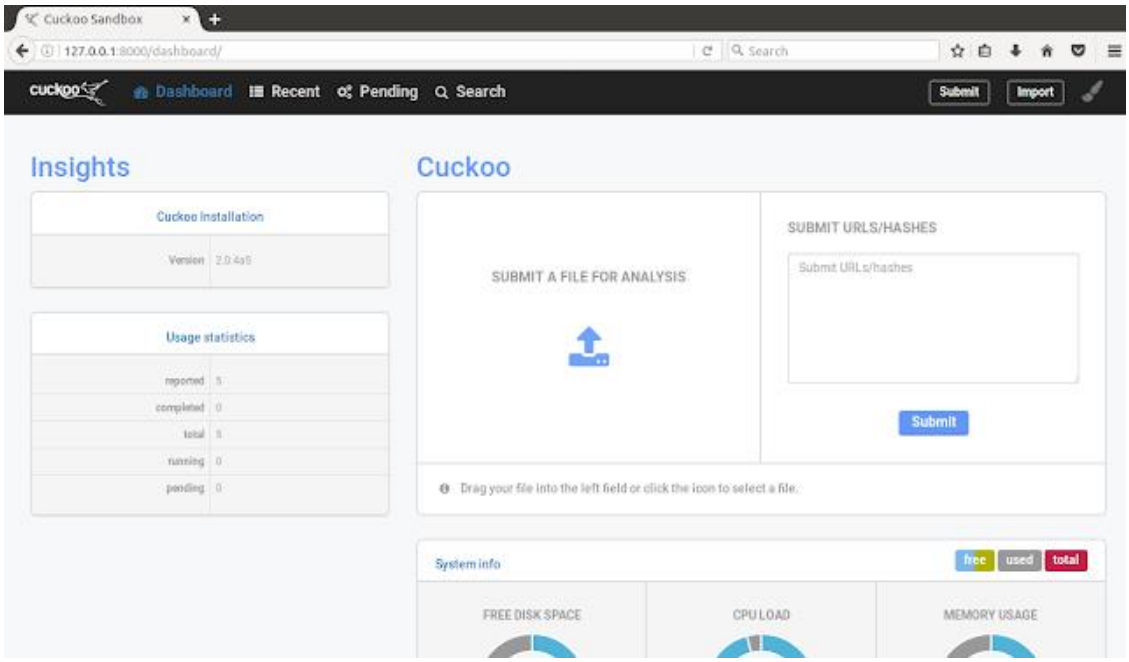


Figure 5-2 cuckoo sandbox web interface

After the cuckoo sandbox is finished it works the analysis results stored in the reporting folder of cuckoos in the form of JSON formats. Cuckoo reports have nine categories of suspicious file analysis outputs such as *info*, *procmemory*, *target*, *network*, *signatures*, *behavior*, *debug*, *strings*, and *metadata* categories. The below figures show that the output of the cuckoo sandbox report categories. Therefore, in this study, we restrict to the *behavior* categories and i*nfo* categories because this category contains the raw behavioral logs for each process running by the analyzed files including logs of the complete processes tracing, a behavioural summary, and a process tree. Info categories include the information about the executed file like started & ended time, machine name, and version. of the analysis file. Also, info categories contain the duration attributes which indicate the total execution time of the suspicious files.

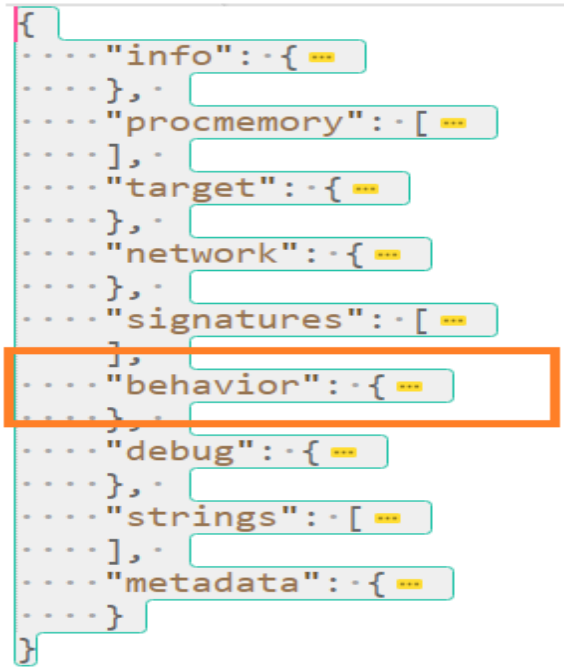


Figure 5-3 cuckoo report categories example

After the cuckoo sandbox configuration is completed the generated report the next step applies to pre-process the results by using python. From behaviour categories, we extract the categories, API calls, and arguments by using a python programming language to prepare the dataset for both training and testing.

5.4.3. Preprocessing Static and Behaviour

In this sub-topic, we preprocess the output from both *abstract syntax tree* in static analysis and *Cuckoo sandbox* in behavioral analysis results. hence the AST and cuckoo sandbox has different special characteristics that need preprocesses. For preprocessing the data we have used python programming language.

We use *content_to_wordlist()* function to split the datasets into wordlists and remove unnecessary special characters from the datasets. Again, *content_to_wordlist* use *lower()* function to convert individual word into lower case format and return the list of words. The followings are the sample code for the word into list function.

```
def Content_to_Wordlist(data_sent):
    """
    Convert a content of datasets in to a list of words.
    """
    # remove non-letters
    content_text = re.sub("[^a-zA-Z0-9]"," ", data_sent)

    # convert to lower case and split at whitespace
    words = content_text.lower().split()

    words = [ words ]

    return words
```

Figure 5-4 Sample code for the word to list

5.5.Feature Extraction Implementation

After finished the pre-processing module of the dataset the next step is to extract the feature from the pre-processed dataset. To extract features that are used for training the model. We used the python Scikit-learn module for *TF-IDF* and the python Genism module for *word2vec* and doc2vec. Each method is implemented with vector transformation methods. Because machine learning models operate on numbers (vectors) instead of words to train models.

5.5.1. Implementation of Word2Vec

To implement word2Vec, this study uses a python Gensim module which is used to implement different embedding methods. Feature modeling with gensim word2Vec is straightforward. First import and instantiate word2Vec class with necessary parameter and builds the vocabulary, and training the Word2Vec model using the preprocessed datasets. We identify the parameter of the model which affect the performance of the model such as vector size, min_count, and the windows. So, vector_size is representing the dimension of the vectors. Min_count is the frequency of the words. Windows is the context of the words.

5.5.2. Implementation of Doc2Vec

The other method we used in the feature extraction is Doc2vec. doc2vec aims to learn the documents into the feature vectors. In this study, we use the python genism module of doc2vec to implement doc2vec feature vectors modelling with different parameters.

5.5.3. Implementation of TF-IDF

The other method we implement is TF-IDF feature extraction methods. To implement it we used sklearn packages class called *TfidfVectorizer*. This class converts the datasets in to a matrix of tf-idf features vector contains the word frequency and their importance in the datasets. We create the *TfidfTransformer* instance to *fit_transform* to get a matrix of tf-idf features vector. And finally, we used the feature vector with the matrix for training our model.

5.6.Machine Learning Module

In this study, we used the python scikit-learning library package to build a machine learning model. Different package libraries necessary for modelling and metrics model evaluation are imported Figure 5-5 shows that the necessary library for build the model.

```
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve, f1_score
import gensim
from gensim.models import Word2Vec, word2vec
from tqdm import tqdm
from numpy import random
import pandas as pd
import numpy as np
import os
import re
import matplotlib.pyplot as plt
```

Figure 5-5 Important packages for model

After the feature extraction process, both static and behaviour datasets with the feature extraction vector used as training belong to *X_train* and the label belongs to *y_train*. The classifier training with *X_train* and *y_train*. To train using the *fit()* method with the appropriate set parameter for each classifier instantiate the class.

To implement the SVM classifier, we use sklearn package to build the SVM model. SVM is a suitable optimizing algorithm for classification data with binary class and that instantiating with different parameter.

5.7.Model Evaluation and Validation

The training that is performed is validated using the K-fold CV technique implement using the sklearn *cross_val_score()* and *cross_val_predict()* methods using K value ten. These methods use the models and dataset to validate the learning skill of each model. We use a 10-fold CV, which means that the dataset is divided into ten different sets and nine sets use to train and one set used to test models each 1 to k iteration as shown in Figure 5-6 below.

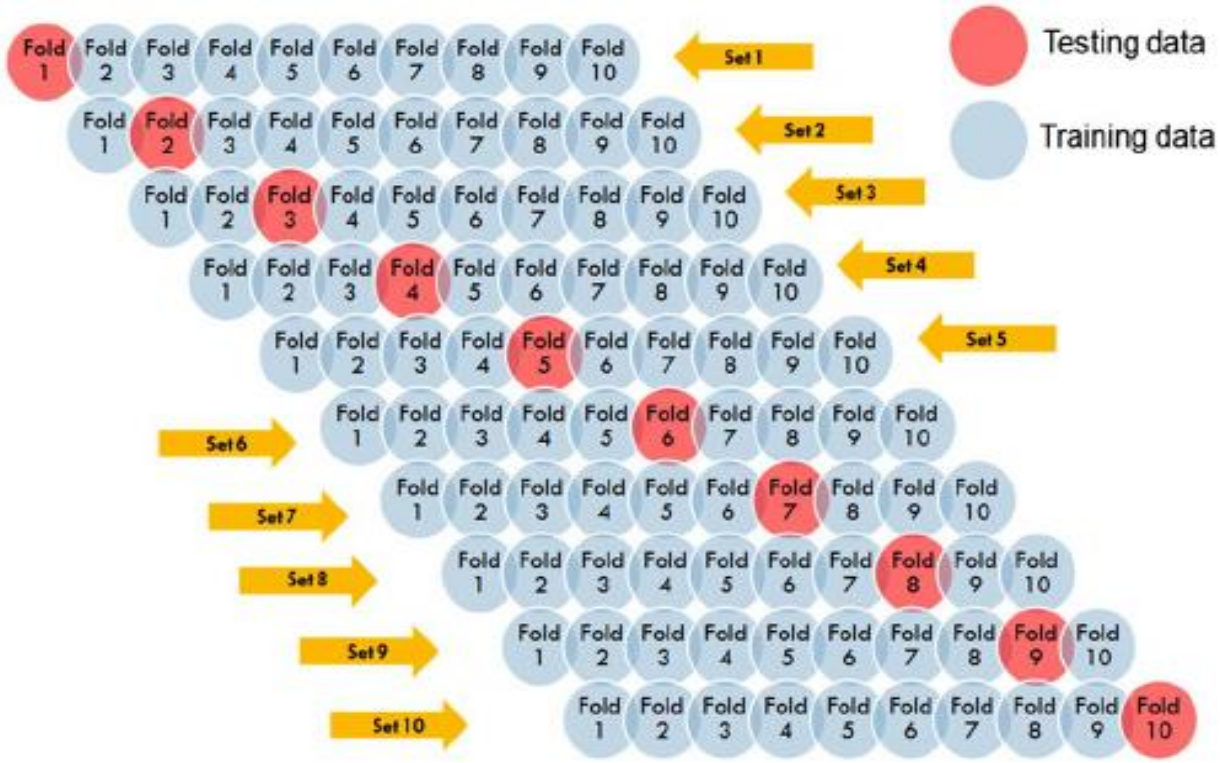


Figure 5-6 10-fold cross-validation procedure [85]

The study used a *classification_report (), accuracy_score(),* and *confusion_matrix()* methods, which are used to evaluate the performance of the built models using predict the value of 10-fold CV for the entire dataset. These methods give a result of evaluation metrics such as accuracy, precision, recall, and F1-score value of the model under testing.

CHAPTER SIX

6. Evaluation, Result, and Discussion

This chapter presents an overview of this study, discusses the evaluation and results of the research. First, it provides the result of this research in the form of a discussion. Then, it reflects the evaluation process, which emerged during this study.

6.1.Result of Study

6.1.1. Dataset Results

In this study, we used 5071 JavaScript code datasets in our experiments. The datasets have 2565 malicious JavaScript and 2506 benign JavaScript. Labelling the dataset is very important since we used a supervised machine learning model. So, we used VirsuTotal APIv2 for labelling the datasets.

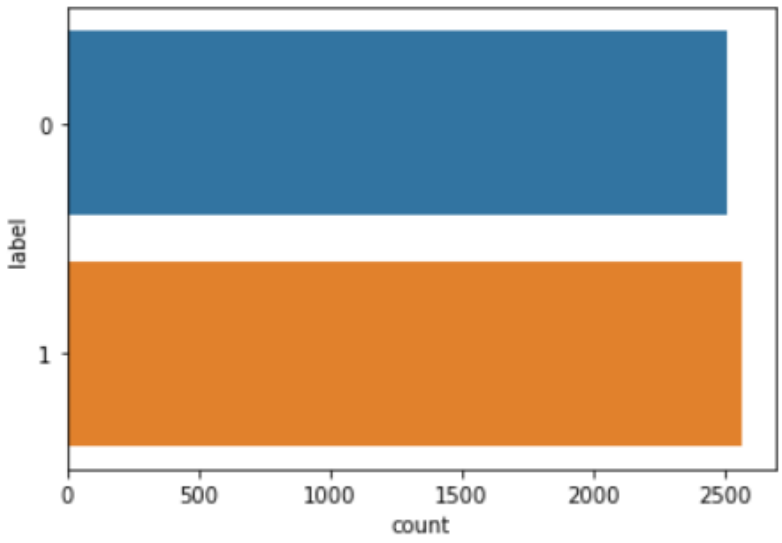


Figure 6-1 Distribution of datasets

In the distribution datasets in Figure 6-1 the y-axis is the number of classes whereas the x-axis is the number of data in each class. 1 represents the malicious class and 0 represents the benign class. We used a train-test split with a 70/30 ratio meaning 70% of the data is used to train the model, and the remaining part, 30%, is used to test the model in the train test model.

6.1.2. Feature Extraction Result

We use the three feature extraction method to create feature vectors from the datasets that were used to train our SVM model. So, we use three different feature extraction methods as Word2vec, TF-IDF, and Doc2Vec method.

In the feature extraction method, we create the feature vector separately from each static and behaviour dataset. After each feature vector is shaped, we use *scipy.sparse.hstack()* to combine the feature vectors. In TF-IDF we used *TfidfVectorizer to* instantiate the class to create the feature vectors. For the word2vec feature extraction method, we use *word2vec* of *genism_model* to generate the feature vectors. In the case of doc2vec, we used *doc2vec* of *genism_model*. After the single feature vector shaped we used it for the model.

Using TF-IDF feature method two feature vector is created from the static and behaviour analysis. For static analysis, the method is to create the 70 unique words from static analysis datasets and 9446 unique words from behaviour analysis datasets.

Using the word2vec feature extraction method we instantiate the genism word2vec model with a parameter such as a number feature, minimum word count, and window sizes. In the static analysis we use 70 number_features and in behaviour analysis 300 number_feature. When combine both features using NumPy we got the 370 feature vector. Generally, word2vec creates the feature vector having 5071 rows with 370 columns (5071 x 370 matrix).

For the Doc2vec feature extraction method, we used the genism model doc2vec function to create the instance of doc2vec. We use 300 number_feature as vector size to train the tagged document to create the feature vectors. Hence, we create a matrix of 600 columns for both the static and behaviour method. Generally, doc2vec creates the feature vector having 5071 rows with 600 columns (5071 x 600 matrix).

6.2.Evaluation Result

6.2.1. Performance Evaluation Result

To evaluate the performance of the proposed model, we used different metrics that appropriate for models such as confusion matrix, accuracy, precision, recall, and F-Measure. Additionally, we used a support vector machine learning algorithm for classification with different feature vector methods. The model has different performance on classification with each feature vector. The algorithm with high performance is selected for classification of JavaScript-based. The accuracy and precision calculated using datasets for SVM algorithms are presented in the following table.

Table 6-1 Result of SVM model evaluation

Feature Extraction Method	Accuracy	Average precision Score
TF-IDF	0.950	0.998
Doc2Vec	0.955	0.990
Word2Vec	0.973	0.997

Table 6-2 shows the results of the evaluation metric for the model based on the features extracted method. It shows the main classification metrics i.e. precision, recall, and f1-score on all classes.

Table 6-2 Classification report of SVM model with a train-test split approach

Feature Extraction Method	Performance evaluation metrics		
	Precision (%)	Recall (%)	F1 (%)
Doc2Vec	0.95	0.95	0.95
TF-IDF	0.96	0.95	0.96
Word2Vec	0.98	0.98	0.98

The results indicate the effectiveness of the proposed approach in terms of precision, recall, and f-measure. From this, we observe that the SVM model with word2vec achieved the highest result of precision, recall, and f1-score than the other two feature vector methods. The SVM

classifier with tf-idf feature vectors is the second-best performer, whereas the SVM model with doc2vec feature vectors is the least performer on all metrics.

Table 6-3 Performance Evaluation of class-wise classification using SVM with Doc2Vec

Class	Performance measures		
	Precision (%)	**Recall (%)**	**F1 (%)**
Malicious class	0.94	0.96	0.95
Benign class	0.96	0.94	0.95

Table 6-3, we conclude that the effectiveness of the proposed approach in terms of class-wise classification. In terms of F-measures, we observe the performance of the proposed approach for the classification of benign class as compare to malicious class almost similar (F1 = 95 %). From this, we understand that the SVM model with Doc2Vec feature extraction has 95 % of accuracy to classify the datasets into benign and malicious classes.

Table 6-4 Performance Evaluation of class-wise classification using SVM with word2vec

Class	Performance measures		
	Precision	**Recall**	**F1**
Malicious class	0.98	0.98	0.98
Benign class	0.98	0.98	0.98

Table 6-4 shows that the SVM model with word2vec feature vectors in class-wise classification. From the table, here the performance of the proposed approach for the classification of both benign class and malicious class the same in terms of F-measures (F1 = 98 %).

Table 6-5 Performance Evaluation of class-wise classification using SVM with TF-IDF

Class	Performance measures		
	Precision	**Recall**	**F1**
Malicious class	1.00	0.91	0.95
Benign class	0.92	1.00	0.96

From Table 6-3, we can observe the effectiveness of the proposed approach of SVM with TF-IDF in terms of class-wise classification. In terms of F-measures, we recognize that the performance of the proposed approach for the classification of benign class (F1 = 96 %) as compare to malicious class (F1 = 95 %).

Also, we use the confusion matrix of the models using the prediction result shown in Figure 6-2, Figure 6-3, and Figure 6-4 below. These confusion matrices are used to calculate the final metrics for the results of all approaches.

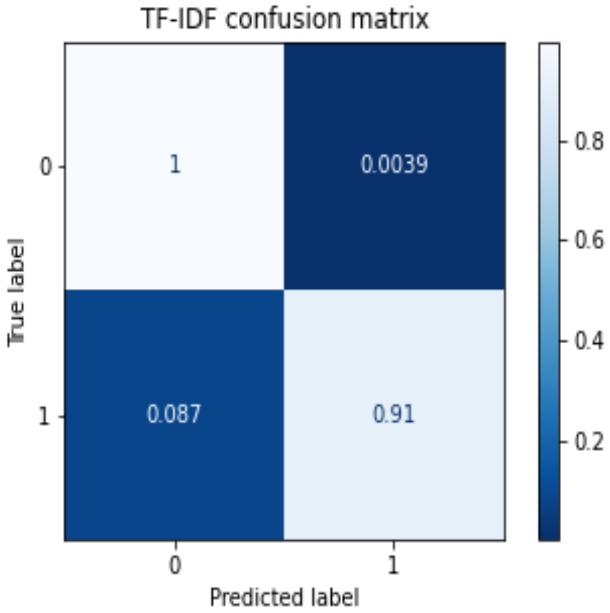


Figure 6-2 Normalized confusion matrix SVM based on TF-IDF with train-test split.

Figure 6-2 shows that the normalized confusion matrices of the SVM model with TF-IDF extracted feature vectors. SVM on TF-IDF classifies 100% of benign and 91% of malicious class correctly; misclassification between benign and malicious is 8.7%. Regarding this count, the model gained a larger number of true positive and true negative results with a false-positive rate (8.7%) and a false-negative (0.39%).

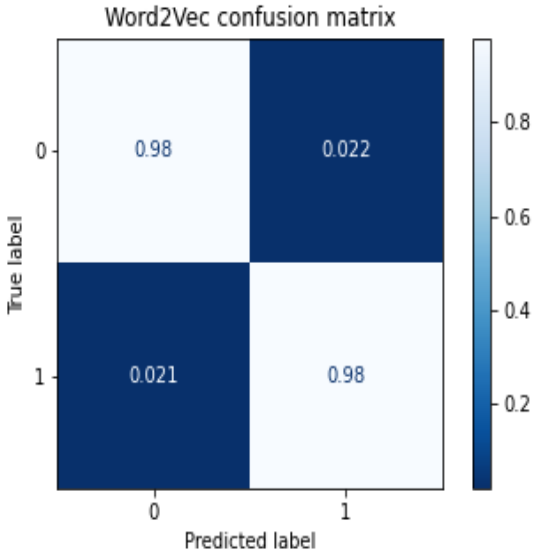


Figure 6-3 Normalized confusion matrix SVM based on Word2Vec feature vectors

Figure 6-3 shows that the normalized confusion matrices of the SVM model with Word2Vec extracted feature vectors. SVM on word2vec classifies 98% of benign and 98% of malicious class correctly; misclassification between benign and malicious is 2.1% only. With Word2Vec feature vectors the model classifier obtained 98% true positive that means among the total number of positives in the test set the model correctly classified 98 % of samples as positive. Among the total number of negative samples in the test set, 98% of samples are correctly classified as negative. Regarding this count, the model gained a larger number of true positive and true negative results with minimum false positive (2.3%) and false-negative (2.1 %).

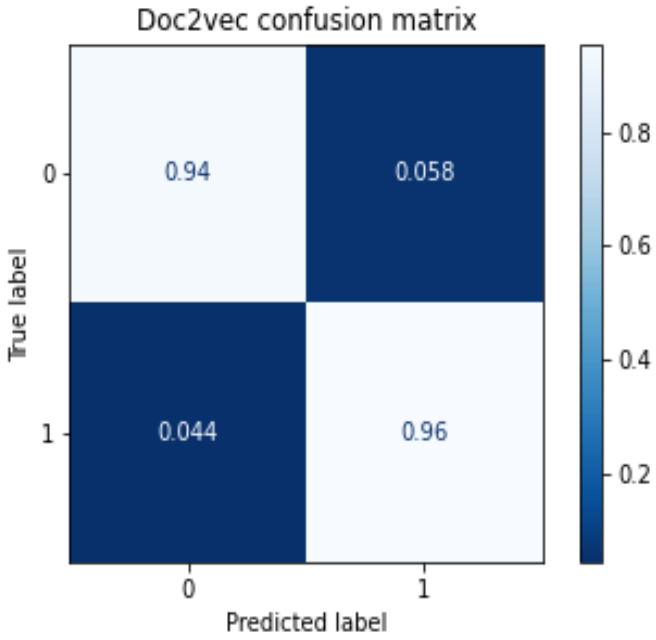


Figure 6-4 Normalized confusion matrix SVM based on Doc2Vec feature vectors.

Figure 6-4 shows that the normalized confusion matrices of the SVM model with Doc2Vec extracted feature vectors. SVM on doc2vec classifies 94% of benign and 96% of malicious class correctly; misclassification between benign and malicious is 4.4%. With Doc2Vec feature vectors the model classifier obtained 96% true positive that means among the total number of positives in the test set the model correctly classified 96 % of samples as positive. Among the total number of negative samples in the test set, 94% of samples are correctly classified as negative. Regarding this count, the model gained a larger number of true positive and true negative results with false positive (5.8%) and false-negative (4.4%).

Generally from the above three figures, we concluded that the normalized Confusion matrix for proposed classifier models based on the feature extracted using the prediction result of the train-test split method for the models shows that the model classifiers with different feature vectors have a greater number of true negative and true positive samples than the other categories of samples. This shows all models are good at realizing the normal samples that they are “Benign” and the malicious samples that they are “Malicious”. SVM with TF-IDF has the greatest coverage of false positives than the others this tells that predicting normal samples as “Malicious”

too much than the others. According to this normalized confusion matrix result, the SVM model with word2vec features extraction obtained better overall accuracy than the others.

To evaluate the performance of machine learning classifiers, k-fold cross-validation was used. For each classifier that has been used, we apply k-fold cross-validation with k =10, i.e., the dataset is randomly split ten times into ten different equal size sets. The following table shows the cross-validation accuracy results of the model based on extracted feature vectors. The method we used *cross_val_score* of sci-kit-learn is used in combination with the estimator with actual data with label parameter.

Table 6-6 SVM model cross-validation accuracy for each feature vectors

Feature	**10 fold accuracy(%) for SVM model**										**Average Accuracy**
	1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**	
Doc2Vec	0.939	0.949	0.966	0.937	0.939	0.955	0.955	0.947	0.931	0.901	0.941
Word2Vec	0.813	0.997	0.981	0.971	0.993	0.997	0.990	0.990	0.996	0.998	0.971
TF-IDF	0.724	0.990	0.891	0.934	0.952	0.930	0.900	0.996	0.763	0.921	0.925

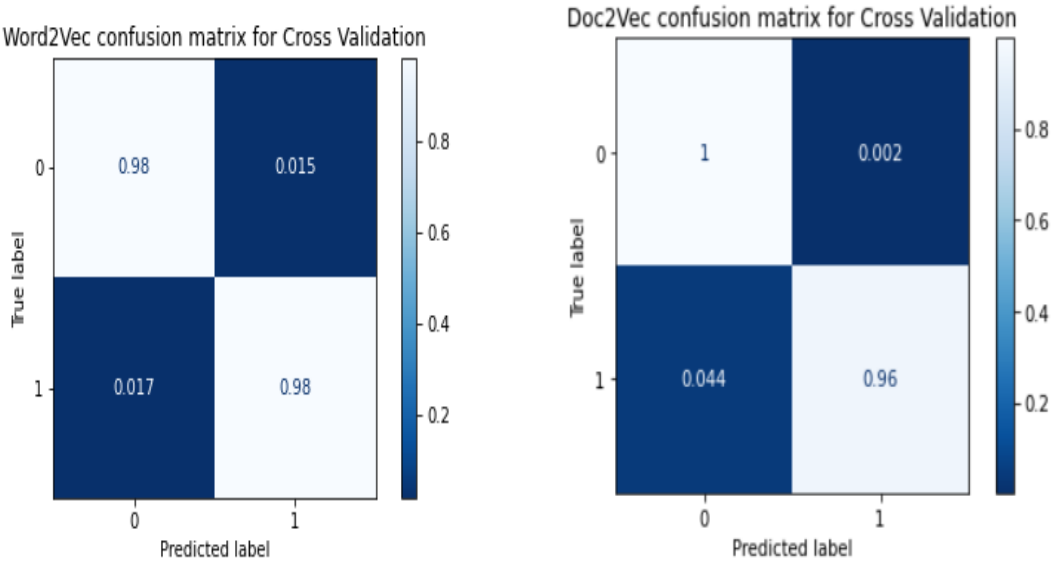
Table 6-6 shows the Cross-Validation accuracy scores for the SVM model based on each feature vector corresponding to each fold test. The lower average accuracy of 90.5% results recorded on TF-IDF the feature model and the higher accuracy 97.1% resulted using the word2vec feature.

Precision, recall, and F1-scores of the model with the cross-validation are summarized in the following table.

Table 6-7 Results of cross-validation approach

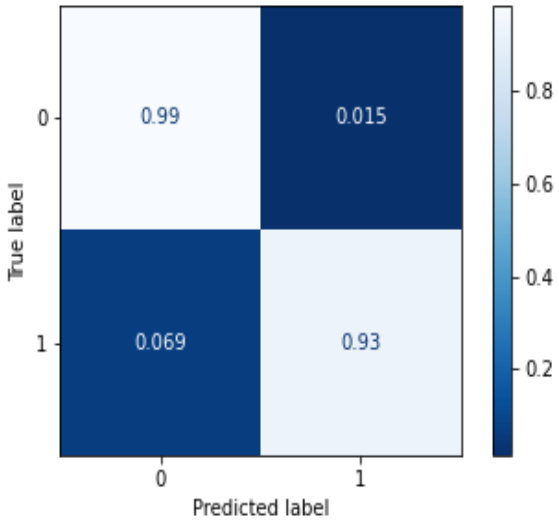
Feature Extraction Method	**Score mean**	**Error**
Doc2Vec	0.941	0.059
TF-IDF	0.925	0.075
Word2Vec	0.971	0.029

Also, the following figures show the normalized confusion matric for the cross-validation evaluation methods. The result of cross-validation is improved accuracy compared to the train-test approaches evaluation.



(a)

(b)



(c)

Figure 6-5 Normalized confusion matrix SVM based on each feature with cross-validation

Figure 6-5 (a) explains the normalized confusion matrix of word2vec with cross-validation. With word2vec feature vectors the model classifier obtained 98% true positive that means among the total number of benign in the train set the model correctly classified 98 % samples as benign. Among the total number of malicious samples in the train set, 98% of samples are correctly classified as malicious. Regarding this count, the model gained a larger number of true positive and true negative results with minimum false positive (1.5%) and false-negative (1.7%). Unlike train-test split, the cross-validation approach has a small number of false positives with false negatives and has more accuracy. In doc2vec, (b) observe that 96% classifies true positives i.e. from the total number of benign the model correctly classified 99.8 % samples as benign and 96% as true negative which is malicious. In terms of false negative and false positive, we observe that doc2vec has 4.4 % false-negative and 0.2% false positive. In TF-IDF (c) has a number 6.9% false positives and 1.5 % of false negatives.

6.2.2. Validation Result

To validate the model we used the validate metrics and to shows which model with feature, extraction has better results. So, we use the ROC curve since it is used to illustrate the performance of the binary classifier system. ROC curves summarize the trade-off between the true positive rate and false-positive rate for a predictive model. A perfect classifier's ROC curve passes through (0, 1), meaning it can classify all positives correctly without a single false positive. Therefore, the following figure shows the ROC curves of the model classifier with different feature vectors.

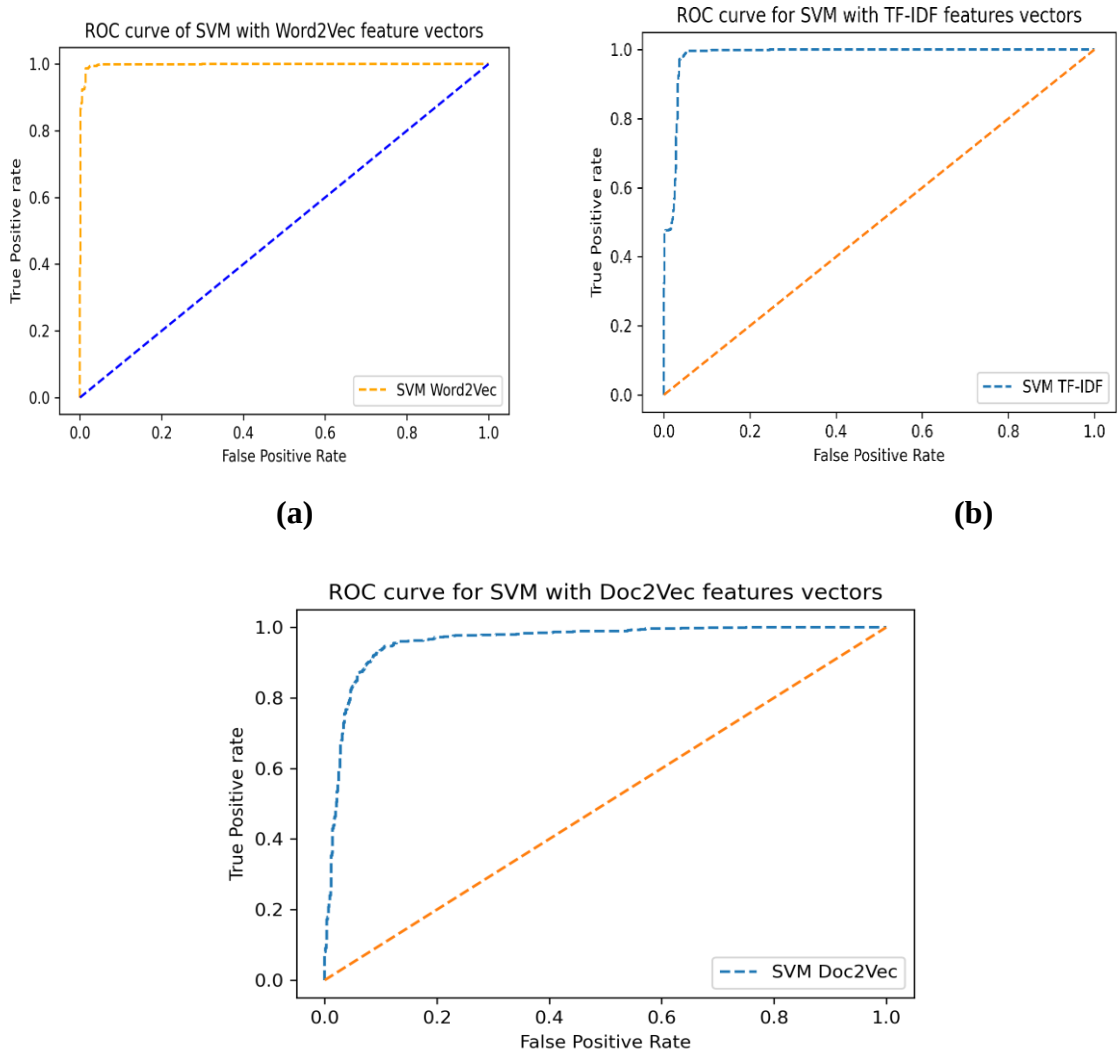


Figure 6-6 (a) ROC curve for SVM with W2V, (b) ROC curve for SVM with TF-IDF, and (c) ROC curve for SVM with D2V (c)

The above Figure 6-6 represents the ROC curve SVM model with each feature vector. The x-axis can be interpreted as a measure of liberalism of the classifier, depicting its false-positive rate (1-specificity). The y-axis represents how well it is at detecting positives, depicting the classifier's true positive rate (sensitivity). The diagonal (45°) line connecting (0,0) to (1,1) is the ROC curve corresponding to random chance. This can be explained by noting that the true positive rate equal to the false positive rate at any cutoff point.

Figure 6-6 (a) shows that the ROC curve for the SVM model with the word2vec feature extraction method typically shows a better ROC curve (AUC ≈ 0.99) and a diagonal line corresponding to random chance (AUC=0.5). ROC curve for doc2vec is 0.89 and for TF-IDF is 0.96. The word2vec ROC curve moves toward (0, 1) to (0, 1) and the AUC approaches 1. Therefore, the ROC curve for word2vec performs much better when we compare it with the other feature extraction methods such as doc2vec and TF-IDF.

6.2.3. Comparison

To compare with the existing JavaScript-based attack detection techniques in references [5], [19], [13]. In reference [5], the author proposed static analysis based was explored. The approach used AST and plain JS optimize with Doc2vec feature vector method were applied to the dataset. Since they used the static analysis method the proposed approach difficult to identify in the case of obfuscation techniques. In reference [19] the authors proposed both static and dynamic methods by using execution time, the page source, and URL features. Execution time is used as dynamic analysis features and the other feature is used as static analysis i.e. page source and URL features. But the proposed study is unrealistic behaviour analysis i.e. only execution time feature. In our case, we used all run time traced activities as features.

In reference [13] the authors proposed a static analysis method only. The authors used V8 to collect the byte code of the datasets and optimize the Word2Vec model to better train real-valued vectors on the byte codes. But in our case, we used both static and dynamic methods. In reference [16] the authors proposed syntactic and semantic features by using abstract syntax tree and control graph respectively.

Generally, in our proposed study we used both static and behaviour analysis. The static analysis only parses the contents in the input and does not execute the program. Even though static

analysis requires fewer resources and cost-effective but difficult to identify in case of obfuscation techniques which allows for false-negatives of results. The behaviour analysis, on the other hand, analyses JavaScript code behaviours during their executions in a real or virtual environment and observe the behaviour of the code during execution time. Behaviour analysis is also not affected by obfuscation techniques as it observes the actual behaviour of the malware and can cover all the paths of the executed trap.

Table 6-8 Comparison with the previous study

#Ref	**Methodology**	**Analysis method**		**Feature Vector method**
		Static	**Behaviour**	
[5]	AST and plain JS	√	*X*	Doc2Vec
[19]	Execution time, the page source, and URL features Number features used 30. Uses Chi-Square for feature selection.	√	√	----------
[13]	V8 JS engine and Byte code	√	*X*	Word2Vec
[16]	Uses syntactic and semantic features	√	*X*	Tree-based CNN and Graph-based CNN
Our approach	AST and Cuckoo sandbox	√	√	Doc2Vec, Word2Vec, TF-IDF

6.3.Discussion

The study proposed the classification JavaScript-based attack using a support vector machine learning algorithm by using the feature extraction method the feature vector is shaped from the analysed datasets. The datasets have consisted of both malicious and benign JavaScript datasets for the models. In the experiment, both static and behaviour analysis are used as some of the special keys for JavaScript classification. After using this two analysis method we use the text-based feature extraction method to get the feature vector which is used to build the model.

Feature extraction methods an important process to capture patterns from the dataset which is used in machine learning algorithms to recognize the patterns. This study relied on three different feature vector methods with a support vector machine learning algorithm. During the experiential testing, word2vec base feature extraction has high performance when we compare with the other two feature vector methods i.e. doc2vec and TF-IDF.

To evaluate the model build classifier we used train-test split which is used to shuffle the data and split it into 70% training data and 30% test data. For implementation, sci-kit learn library method train_test_split from model_selection is used. To evaluate the performance of the model cross-validation with 10-fold used. ROC is used to show how the classifier can separate true positive rate and false-positive rate for a predictive model using different probability thresholds. We used a parameter optimizer for the support vector machine model which has a great influence on the accuracy and complexity of the classification models. As we can see from the Figure 6-6 model train and test with word2vec have more classify all positives correctly with a small false positive. The following are the research question stated in this study is addressed based on the result and evaluation metrics that have been done.

RQ1. *What are the analysis method used in the JavaScript-based attack classification?* In order to reach to answer the research question, two analysis method is implemented: static and behaviour analysis method. Static analysis is a method that is used for debugging by examining source code before execution. Behavioural analysis is the method used to analyze the source code under virtual (isolated) environments for tracing all activities that happened during the run time environments.

RQ2: *What are the feature extraction method used for the classification of JavaScript-based attack?* Section **6.1.2** illustrates the results of the feature extraction method in this study. Hence, the proposed solution used three feature extraction methods like TF-IDF, word2vec, and doc2vec, to classify the JavaScript-based attacks.

RQ3: *How to design and enhance a better approach for classifies JavaScript-based attacks?* The proposed solution (see section **4.2**) we design by the combination of both static and behaviour analysis results we can enhance the classification performance with the word2vec feature extraction method. As shown in Table 6-2 SVM with word2vec achieves high performance. Accordingly, accuracy is reached 97%, which shows that most dataset sample is classified accurately found during the test. Simultaneously, the model achieved a recall rate of 97% which means that it has low false-positive results when compared with the TF-IDF and doc2vec method.

CHAPTER SEVEN

7. Conclusion and Feature Work

This chapter contains recommendations for additional and useful tasks in the future. It covers important thoughts and ideas, which were not included in this study, due to time and scope restrictions.

7.1.Conclusion

A framework to classify the JavaScript-based attack using support vector machine learning is constructed. The main aim of this research work is to build an efficient model for the classification of JavaScript-based attacks into malicious and benign. To address the current challenges of the analysis method and the feature extraction method for classification we used both behaviour and static analysis to analyze the JavaScript datasets. We used the cuckoo sandbox to get the behavioural analysis for our datasets and an abstract syntax tree is used for static analysis. The output of both behaviour analysis and static analysis is text-based files. So, this is followed by using a text-based feature extraction method such as TF-IDF, doc2vec, and word2vec to extract the feature from the datasets.

A support vector machine was used as a classifier. Support vector machine is constructing hyperplanes for linearly separable data in multidimensional space using a linear kernel function. The hyperplanes separate the sample data into different groups. Objects are rearranged for optimal separation through the use of this kernel function. We build support vector machine model by using the feature vector created by the feature extraction methods. The process of modelling is to find patterns in the training set of the dataset that maps the dataset with their features to the target class by using learning algorithms. The results of the classification show a slightly better performance of the static and behaviour analysis when comparing the accuracy and f1 measurement.

The machine learning techniques are validated in two ways. In the first experiment, the models are trained with their default parameters and tested on a separate test set. In the second experiment, 10-fold cross-validation techniques are applied to the machine learning techniques

to find the optimal value for each hyper parameter to avoid overfitting and improve the accuracy of the models.

The implementation started with abstract syntax and the cuckoo sandbox is applied to the dataset and the feature extraction method is applied to both abstract syntax and cuckoo sandbox output. The extracted feature vectors are used for training and testing the classifiers. The models are implemented in the python scikit-learning library package.

Based on the evaluation results of the model with different feature extraction methods presented in chapter 6, SVM with word2vec feature extraction has a high-performance model. SVM score classification accuracy 97.1 %, Precision 98 %, and Recall 98 % with f1-score of 98 %; SVM with TF-IDF feature extraction method, SVM model has score classification accuracy 90.4 %, Precision 96 %, and Recall 95 % with f1-score of 96 %. And SVM with Doc2Vec feature extraction method, SVM model has score classification accuracy 95.5 %, Precision 96 %, and Recall 95 % with f1-score of 96 %.

From this, we conclude that the SVM model with word2vec feature extraction has better performance in the classification of JavaScript-based classification and also has a low false positive and false negative.

7.2.Recommendations

As far as this research result is concerned, support vector machine with word2vec feature extraction method has shown the highest accuracy in classifying attacks related to JavaScript. So, developers, programmers, students as well as websites and data server owners can extend the prototype and use the system for experiencing and controlling javascript-based attacks of software. However, this does not mean that the result obtained by applying other feature extraction methods tried in this paper is discouraging. They can also be used for similar purposes as their prediction accuracy measures show high.

7.3.Feature Work

In this research 5071, JavaScript codes as a dataset have been used and a support vector which is a supervised machine learning algorithm has been implemented with a text-based feature extraction method to build models. Even though, good performance has been achieved through experimentation, it is better to see the difference in the performance results by increasing the dataset as well as through applying unsupervised or deep learning algorithms models.

Special Acknowledgement

This master thesis research is funded by Adama Science and Technology University (ASTU) under the grant number.

ASTU/SM-R/095/19

Adama Ethiopia

Reference.

[1] G. Deepa and P. S. Thilagam, “Securing web applications from injection and logic vulnerabilities : Approaches and challenges,” *Inf. Softw. Technol.*, vol. 74, pp. 160–180, 2016, doi: 10.1016/j.infsof.2016.02.005.

[2] M. F. Der, “Detecting Malicious JavaScript,” University of Richmond, Richmond, VA, 2010.

[3] Kamesh and N. Sakthi Priya, “A survey of cyber crimes Yanping,” *Secur. Commun. Networks*, vol. 5, pp. 422–437, Feb. 2012, doi: 10.1002/sec.

[4] C. Bichhawat, A.; Rajani, V.; Garg, D.; Hammer, “Information flow control in WebKit’s JavaScript bytecode,” in *In Proceedings of the International Conference on Principles of Security and Trust*, 2014, pp. 159–178.

[5] 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,” *Appl. Soft Comput. J.*, vol. 84, Aug. 2019.

[6] K. Schütt, M. Kloft, A. Bikadorov, and K. Rieck, “Early Detection of Malicious Behavior in JavaScript Code,” in *Proceedings of the 5th ACM Workshop on Artificial Intelligence and Security (AISec12)*, 2012, pp. 15–24.

[7] M. Cova, C. Kruegel, and G. Vigna, “Detection and analysis of drive-by-download attacks and malicious JavaScript code,” in *Proceedings of the 19th International Conference on World Wide Web, WWW*, Apr. 2010, pp. 281–290.

[8] G. E. Rodríguez, J. G. Torres, P. Flores, and D. E. Benavides, “Cross-site scripting (XSS) attacks and mitigation : A survey,” *Comput. Networks J.*, vol. 166, Oct. 2020.

[9] M. Louie, “Expanding Threats Demand New Protections Executive,” THE MAGECART AD THREAT CONNECTION, Rep. no 1, Jun. 2019. [Online]. Available: https://static1.squarespace.com/static/5c4752bd7c9327814573cd04/t/5df804d32b72246fbefce106/1576535268232/2019HolidayThreatReport-DEVCON.pdf.

[10] P. Likarish, E. Jung, and I. Jo, “Obfuscated malicious javascript detection using

classification techniques," in *4th International Conference on Malicious and Unwanted Software, MALWARE*, 2009, pp. 47–54.

[11] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "ZOZZLE : Fast and Precise In-Browser JavaScript Malware Detection," *USENIX Secur.*, pp. 33–48, Feb. 2016.

[12] J. W. Stokes, R. Agrawal, G. McDonald, and M. Hausknecht, "ScriptNet: Neural Static Analysis for Malicious JavaScript Detection," *Proc. - IEEE Mil. Commun. Conf. MILCOM*, 2019.

[13] Y. Fang, C. Huang, L. Liu, and M. Xue, "Research on malicious JavaScript detection technology based on LSTM," in *IEEE Access*, Oct. 2018, vol. 6, pp. 59118–59125.

[14] Y. Hao, H. Liang, D. Zhang, Q. Zhao, and B. Cui, "JavaScript malicious codes analysis based on naive bayes classification," in *Proceedings - 2014 9th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2014*, Feb. 2014, pp. 513–519.

[15] A. Fass, R. P. Krawczyk, M. Backes, and B. Stock, "JAST : Fully Syntactic Detection of Malicious (Obfuscated) JavaScript," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Feb. 2018, vol. 1, pp. 303–325.

[16] H. Liang, Y. Yang, L. Sun, and L. Jiang, "JSAC: A Novel Framework to Detect Malicious JavaScript via CNNs over AST and CFG," in *Proceedings of the International Joint Conference on Neural Networks*, Jul. 2019, pp. 1–8.

[17] D. R. Patil and J. B. Patil, "Detection of Malicious JavaScript Code in Web Pages," *Indian J. Sci. Technol.*, vol. 10, pp. 1–12, Jun. 2017.

[18] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning," *IEEE Trans. Dependable Secur. Comput.*, vol. 12, no. 6, pp. 688–707, Aug. 2015.

[19] N. A. Khan, M. Y. Alzaharani, and H. A. Kar, "Hybrid Feature Classification Approach for Malicious JavaScript Attack Detection using Deep Learning," vol. 18, no. 5, pp. 79–90, May 2020.

[20] P. Raman, D. Howe, and A. Somayaji, “Jaspin: Javascript Based Anomaly Detection of Cross-Site Scripting Attacks,” *Carlet. Univ.*, pp. 1–122, Sep. 2008.

[21] A. W. Marashdih and Z. F. Zaaba, “Cross Site Scripting: Removing Approaches in Web Application,” in *4th Information Systems International Conference*, Nov. 2017, vol. 124, pp. 647–655.

[22] G. Díaz and J. Ramón, “Static analysis of source code security : Assessment of tools against SAMATE tests,” *Inf. Softw. Technol.*, vol. 55, no. 8, pp. 1462–1476, Aug. 2013.

[23] R. Wang, Y. Zhu, J. Tan, and B. Zhou, “Detection of malicious web pages based on hybrid analysis,” *J. Inf. Secur. Appl.*, vol. 35, pp. 68–74, Jun. 2017.

[24] M. K. Gupta, M. C. Govil, and G. Singh, “Predicting Cross-Site Scripting (XSS) security vulnerabilities in web applications,” in *12th International Joint Conference on Computer Science and Software Engineering, JCSSE 2015*, 2015, pp. 162–167.

[25] A. Duraisamy; M.Subramaniam, “Detection and Prevention of Cross Site Scripting Attacks using Concolic Testing and Pattern Filtering Approach in Web Application,” *Aust. J. Basic Appl. Sci.*, vol. 10, pp. 66–73, Dec. 2016.

[26] Y. Minamide and D. S. Engineering, “Static Approximation of Dynamically Generated Web Pages,” in *International World Wide Web Conference Com- mittee (IW3C2).*, May 2005, pp. 432–441.

[27] F. Yu, M. Alkhalaf, and T. Bultan, “STRANGER : An Automata-based String Analysis Tool,” *TACAS*, Jun. 2010.

[28] S. Wassermann, Gary; Zhendong, “Static Detection of Cross-Site Scripting Vulnerabilities,” in *In Proceeding of the 30th International Conference on Software Engineering*, May 2008, pp. 171–180.

[29] G. Richards, S. Lebresne, B. Burg, and J. Vitek, “An analysis of the dynamic behavior of JavaScript programs,” in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Jun. 2010, pp. 1–12.

[30] T. Jim and M. Hicks, “Defeating Script Injection Attacks with Browser-Enforced

Embedded Policies," in *In Proceedings of the 16th International World Wide Web Conference, ACM*, May 2007, pp. 601–610.

[31] X. He, L. Xu, and C. Cha, "Malicious JavaScript Code Detection Based on Hybrid Analysis," in *25th Asia-Pacific Software Engineering Conference (APSEC)*, 2018, pp. 365–374.

[32] D. Balzarotti, M. Cova, V. V Felmetsger, and G. Vigna, "Multi-Module Vulnerability Analysis of Web-based Applications," in *14th ACM Conference on Computer and Communications Security*, Nov. 2007, pp. 114–126.

[33] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A Novel Neural Source Code Representation Based on Abstract Syntax Tree," *41st Int. Conf. Softw. Eng.*, pp. 783–794, May 2019.

[34] S. MacDonell, G. Frantzeskou, E. Stamatatos, and S. Gritzalis, "Examining the significance of high-level programming features in source code author classification," *J. Syst. Softw.*, vol. 18, no. 3, pp. 447–460, May 2008.

[35] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *J. Am. Soc. Inf. Sci.*, vol. 41, p. 391, 2005.

[36] D. M., A. Blei, Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Feb. 2003.

[37] D. KUNDEL, "Introduction to Abstract Syntax Trees - Twilio," *ASTs - What are they and how to use them*, 2020. https://www.twilio.com/blog/abstract-syntax-trees (accessed Aug. 19, 2020).

[38] Y. Fang, C. Huang, Y. Su, and Y. Qiu, "Detecting malicious JavaScript code based on semantic analysis," *Comput. Secur.*, vol. 93, Feb. 2020.

[39] Z. Cai, L. Lu, and S. Qiu, "An Abstract Syntax Tree Encoding Method for Cross-Project Defect Prediction," *IEEE Access*, vol. 7, pp. 170844–170853, 2019.

[40] G. Claudio, S. Mark, T. Alessandro, and S. Mark, "The Cuckoo sandbox," *Cuckoo*, 2013. https://cuckoosandbox.org/ (accessed Dec. 26, 2020).

[41] S. Ale and Vishwanathan, *Introduction to machine learning*. the press syndicate of the university of cambridge, 2008.

[42] P. Harrington, *Machine Learning in Action*. Manning Publications C, 2012.

[43] R. Das and T. H. Morris, "Machine learning and cyber security," in *2017 International Conference on Computer, Electrical and Communication Engineering, ICCECE 2017*, Dec. 2018, pp. 1–7.

[44] D. Fradkin and I. Muchnik, "Support vector machines for classification," *Discret. Math. Theor. Comput. Sci.*, Jan. 2014.

[45] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, "A Survey of Deep Learning Methods for Cyber Security," *Information*, vol. 10, no. 4, p. 122, Apr. 2019.

[46] W. Wong and M. Stamp, "Hunting for Metamorphic Engines ∗," *J. Comput. Virol.*, vol. 10, no. 5, pp. 211–220, Nov. 2014, doi: 10.1007/s11416-006-0028-7.

[47] J. B. L. Mé, "Code obfuscation techniques for metamorphic viruses," *J. Comput. Virol.*, vol. 4, pp. 211–220, Feb. 2008, doi: 10.1007/s11416-008-0084-2.

[48] S. Kazi and M. Stamp, "Hidden Markov Models for Software Piracy Detection," *Inf. Secur. J. A Glob. Perspect.*, vol. 22, no. 3, pp. 140–149, 2011.

[49] M. Almseidin, M. Alzubi, S. Kovacs, and M. Alkasassbeh, "Evaluation of Machine Learning Algorithms for Intrusion Detection System," in *15th International Symposium on Intelligent Systems and Informatics (SISY)*, Jan. 2018, no. 4, pp. 000277–000282.

[50] V. Das, V. Pathak, S. Sharma, and M. Srikanth, "NETWORK INTRUSION DETECTION SYSTEM BASED ON MACHINE LEARNING," *Int. J. Comput. Sci. Inf. Technol.*, vol. 2, no. 6, pp. 138–151, 2010.

[51] M. Luckner, "Practical Web Spam Lifelong Machine Learning System with Automatic Adjustment to Current Lifecycle Phase," *Secur. Commun. Networks*, no. 2, pp. 1–16, Feb. 2019.

[52] S. Park and J. Choi, "Malware Detection in Self-Driving Vehicles Using Machine Learning Algorithms," *J. Adv. Transp.*, vol. 2020, no. 4, pp. 1–9, Jan. 2020.

[53] D. Sculley, M. E. Otey, M. Pohl, B. Spitznagel, and J. Hainsworth, "Detecting Adversarial Advertisements in the Wild," in *17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011, pp. 1–9.

[54] C. Dunn, C. Kanich, and B. Ziebart, "Leveraging Machine Learning to Improve Unwanted Resource Filtering," *14 Proc. 2014 Work. Artif. Intell. Secur. Work.*, pp. 95–102, Nov. 2014.

[55] K. Soska, N. Christin, K. Soska, and N. Christin, "Automatically Detecting Vulnerable Websites Before They Turn Malicious," *USENIX Secur.*, pp. 625--640, Aug. 2014.

[56] K. Borgolte, C. Kruegel, G. Vigna, K. Borgolte, C. Kruegel, and G. Vigna, "Meerkat : Detecting Website Defacements through Image-based Object Recognition This paper is included in the Proceedings of the," *USENIX Secur. Symp.*, pp. 615--625, Aug. 2015.

[57] P. Chapman and D. Evans, "Automated Black-Box Detection of Side-Channel Vulnerabilities in Web Applications," in *18th ACM Conference on Computer and Communications Security, Chicago*, Oct. 2011, pp. 263–274.

[58] M. M. Truşcă, "Efficiency of SVM classifier with Word2Vec and Doc2Vec models," in *13th International Conference on Applied Statistics*, Aug. 2019, vol. 2019, no. 1, pp. 496–503.

[59] D. Karani, "Introduction to Word Embedding and Word2Vec," *Towards Data Science*. https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa (accessed Nov. 16, 2020).

[60] Eiki, "Feature Extraction in Natural Language Processing with Python," *Medium*. https://medium.com/@eiki1212/feature-extraction-in-natural-language-processing-with-python-59c7cdcaf064 (accessed Nov. 16, 2020).

[61] Z. Boyun, Y. Jianping, H. Jingbo, and W. Shulin, "New Malicious Code Detection Based on N- Gram Analysis and Rough Set Theory," Oct. 2007, no. 4, pp. 626–633, doi: 10.1007/978-3-540-74377-4.

[62] J. Zico Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," in *In Proceedings of the tenth ACM SIGKDD international conference on*

Knowledge discovery and data mining (KDD '04), Aug. 2006, vol. 7, pp. 470–478.

[63] M. Imran, M. T. Afzal, and M. A. Qadir, "A comparison of feature extraction techniques for malware analysis," *Turkish J. Electr. Eng. Comput. Sci.*, vol. 25, no. 2, pp. 1173–1183, 2017, doi: 10.3906/elk-1601-189.

[64] M. Beraha, A. M. Metelli, M. Papini, A. Tirinzoni, and M. Restelli, "Feature selection via mutual information: New theoretical insights," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–9.

[65] X. Song, C. Chen, B. Cui, and J. Fu, "Malicious javascript detection based on bidirectional LSTM model," in *Applied Sciences*, 2020, vol. 10, no. 10, doi: 10.3390/app10103440.

[66] S. Ndichu, S. Ozawa, T. Misu, and K. Okada, "A Machine Learning Approach to Malicious JavaScript Detection using Fixed Length Vector Representation," in *18th International Joint Conference on Neural Networks*, 2018, pp. 1–8, doi: 10.1109/IJCNN.2018.8489414.

[67] D. R. Patil and J. B. Patil, "Detection of Malicious JavaScript Code in Web Pages," *Indian J. Sci. Technol.*, vol. 10, no. 19, pp. 1–12, 2017, doi: 10.17485/ijst/2017/v10i19/114828.

[68] V. Jakkula, "Tutorial on Support Vector Machine (SVM)," 2011, pp. 1–13, [Online]. Available: http://www.ccs.neu.edu/course/cs5100f11/resources/jakkula.pdf.

[69] C. Manitoba and M. Genetics, "Applications of Support Vector Machine (SVM) Learning," *Mach. Learn.*, vol. 51, no. 1, pp. 41–51, 2018, doi: 10.21873/cgp.20063.

[70] "Model Evaluation Techniques." https://www.datasciencecentral.com/profiles/blogs/7-important-model-evaluation-error-metrics-everyone-should-know (accessed Sep. 30, 2020).

[71] W.-H. WANG, Y.-J. LV, H.-B. CHEN, and Z.-L. FANG, "A Static Malicious Javascript Detection Using SVM," in *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*, 2013, pp. 214–217, doi: 10.2991/iccsee.2013.56.

[72] M. Y. Kan and H. O. Nguyen Thi, “Fast webpage classification using URL features,” in *International Conference on Information and Knowledge Management, Proceedings*, Nov. 2005, pp. 325–326, doi: 10.1145/1099554.1099649.

[73] M. Mimura and Y. Suga, “Filtering Malicious JavaScript Code with Doc2Vec on an Imbalanced Dataset,” in *2019 14th Asia Joint Conference on Information Security (AsiaJCIS)*, 2019, pp. 24–31, doi: 10.1109/AsiaJCIS.2019.000-9.

[74] “Malicious | Definition of Malicious by Merriam-Webster.” https://www.merriam-webster.com/dictionary/malicious (accessed Aug. 20, 2020).

[75] “Malicious Code.” https://www3.safenet-inc.com/csrt/malicious-code-more.aspx (accessed Aug. 20, 2020).

[76] “Definition of benign virus | PCMag.” https://www.pcmag.com/encyclopedia/term/benign-virus (accessed Aug. 20, 2020).

[77] Alexa, “Alexa static.” http://s3.amazonaws.com/alexa-static/top-1m.csv.zip (accessed Feb. 20, 2020).

[78] HynekPetrak, “Javascript malware collection,” 2017. https://github.com/HynekPetrak (accessed Jan. 30, 2020).

[79] C. Curtsinger, C. , Livshits, B. , Zorn, B.G. , Seifert, “Zozzle: Fast and precise in- -browser JavaScript malware detection,” *USENIX Secur. Symp. San Fr.*, pp. 33–48, Feb. 2016.

[80] S. Mukkamala, G. Janoski, and A. Sung, “Intrusion detection using neural networks and support vector machines,” in *Proceedings of the International Joint Conference on Neural Networks*, Sep. 2002, vol. 2, no. 1, pp. 1702–1707.

[81] S. Santhosh and S. R, “N-Gram based malicious code detection using Support Vector Machine learning approach,” in *Fourth International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom2012)*, Feb. 2012, pp. 237–239.

[82] S. Ranveer and S. Hiray, “SVM Based Effective Malware Detection System,” *Int. J.*

Comput. Sci. Inf. Technol., vol. 6, no. 4, pp. 3361–3365, 2015.

[83] B. Sanjaa and E. Chuluun, "Malware detection using linear SVM," *8th Int. Forum Strateg. Technol. 2013, IFOST 2013 - Proc.*, vol. 2, pp. 136–138, 2013, doi: 10.1109/IFOST.2013.6616872.

[84] H. S. Ham, H. H. Kim, M. S. Kim, and M. J. Choi, "Linear SVM-based android malware detection," *Lect. Notes Electr. Eng.*, vol. 301, pp. 575–585, 2014, doi: 10.1007/978-94-017-8798-7_68.

[85] J.-S. Chou, C.-K. Chiu, M. Farfoura, and I. Al-Taharwa, "Optimizing the Prediction Accuracy of Concrete Compressive Strength Based on a Comparison of Data-Mining Techniques," *J. Comput. Civ. Eng.*, vol. 25, no. 3, pp. 242–253, 2011, doi: 10.1061/(asce)cp.1943-5487.0000088.

Appendixes

Appendix A: Node JS Programming for Abstract Syntax Tree and ast-traverse

```
var esprima = require('esprima');
var traverse = require("ast-traverse");
const path = require('path')
const fs = require('fs')
var Filequeue = require('filequeue');
var fq = new Filequeue(11000);
const directoryPath = path.join(__dirname, 'Data')
fq.readdir(directoryPath,function(err,files)
{
        if(err)throw err
        files.forEach(function(file){
                fs.readFile(directoryPath+'/'+file,'utf8',function(err,data){
                        if(err){
                                return console.log(err)
                        }
                        try {
                                        var ast = esprima.parseScript(data);
                                        var indent = 0;
                                        var content
                                        file = path.basename(file,'.js')
                                        var logStream =
fs.createWriteStream('PreAST/'+file+'.txt', {flags: 'a'});
                                        traverse(ast, {
                                        pre: function(node) {
                                           content = Array(indent + 1).join(" ") + node.type;
                                           content = content.replace(/  +/g, ' ')
                                           indent += 4;
```

```
                                    logStream.write(content)

                                },
                                post: function() {
                                    indent -= 4;
                                }
                                });
                            } catch(e){
                                    }
                    });

            });
});
```

Appendix B: Sample Source Code Extracting Behavioural Class of Cuckoo sandbox Report

```
Malicious_result = []
count = 1
for Filename_path in os.listdir (Malicious_path):
    Filename = os.path.basename(Filename_path)
    with open(Malicious_Static_Folder+'/'+ Filename,'r', encoding='utf-8' ) as Static_File_Read:
        Malicious_result.append(Static_File_Read.read())
    with open(Malicious_Behaviour_Folder+'/'+Filename,'rb') as Behaviour_File_Read:
        Malicious_result.append(Behaviour_File_Read.read())
    with open(Combined_File +'Malicious/' + Filename,'w', encoding='utf-8') as
Write_Both_File:
        Write_Both_File.write(str(Malicious_result) )
    Malicious_result = []
    # print("Processing Malicious " + str(count))
    # count = count + 1
print('Malicious Succesfully')
```

```
for Filename_path in os.listdir(Data_With_Tag_Folder):
   try:
      with open(Data_With_Tag_Folder +'/'+ Filename_path,'r',encoding='utf-8') as fread:
         data = fread.read()
         p =re.compile(r'<.*?>')
         data = p.sub('', data)
      Filename = os.path.basename(Filename_path)
      Filename = os.path.splitext(Filename)[0]
      with open(Removed_Script_Tag_Training_Folder + Filename + '.js', 'w' ,encoding='utf-
8') as wfile:
         wfile.write(data)
      print(Filename+ "-->Success" + str(count))
      count = count + 1
   except IOError:
      print('Error' + IOError)
```

Appendix C: Sample Code for Word2vec Feature Extraction

```
model_name = 's_new_train_model'
num_features = 70    # Word vector dimensionality

min_word_count = 1    # Minimum word count

num_workers =50       # Number of threads to run in para
llel
context = 10          # Context window size
downsampling = 1e-
3   # Downsample setting for frequent words
model_w2v = word2vec.Word2Vec(s_reviews,workers=num_work
ers,size=num_features,min_count=min_word_count,window=co
ntext,sample=downsampling)
model_w2v.init_sims(replace=True)
model_w2v.save(model_name)
print('success')
```

```
#the following is used to create the feature vectors
w2v_words=list(model_w2v.wv.vocab)
listof_sent_vec=[]
for sent in tqdm(s_reviews):
    sent_vec = np.zeros(70)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = model_w2v.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    listof_sent_vec.append(sent_vec)
Label = list(s_df.label)
list_col=tuple(range(num_features))
W2v_data_train=pd.DataFrame(data=listof_sent_vec, column
s=list_col)
W2v_data_train["label"] = Label
print(W2v_data_train.head(10))
print(W2v_data_train.shape)
```

Appendix D: Sample Code for Building and Evaluating Models

```
SVM = svm.SVC(kernel='linear',gamma='auto',probability=T
rue)
SVM.fit(X_train,y_train)
_pred = SVM.predict(X_test)
print('Testing accuracy %s' % accuracy_score(y_test, y_p
red))
print('Testing F1 score: {}'.format(f1_score(y_test, y_p
red, average='weighted')))
#calculate the average precision-recall
y_score = SVM.decision_function(X_test)
average_precision = average_precision_score(y_test, y_sc
ore)
print('Average precision-recall score: {0:0.3f}'.format(
```

```
      average_precision))
#Accuracy of model with cross validation
accuracy = cross_val_score(SVM_cv, X_train_allFeature, y
_train_allFeature,scoring='accuracy', cv=10)
print(accuracy)
print("Accuracy of Model with Cross Validation is:",accu
racy.mean() * 100)
```