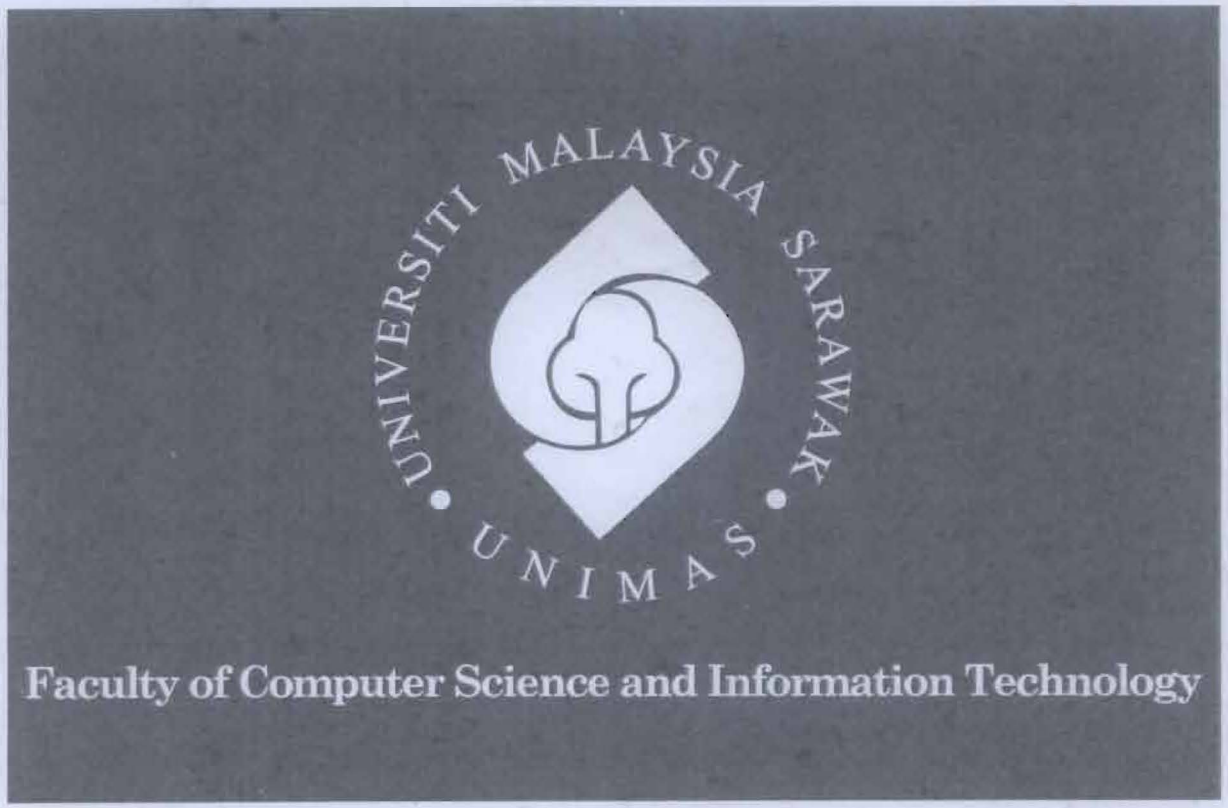
FSKTM
1501004g
Koleksi 2019



Ensemble Classifier on Malicious JavaScript Attacks Detection using Extended Features

Nayeem Ahmad Khan

Doctor of Philosophy
2018

Ensemble Classifier on Malicious JavaScript Attacks Detection using Extended Features

Nayeem Ahmad Khan

A thesis submitted

In fulfilment of the requirements for the degree of Doctor of Philosophy

(Computer Science)

Faculty of Computer Science and Information Technology
UNIVERSITI MALAYSIA SARAWAK
2018

DECLARATION

I hereby declare that the thesis is based on my original work except for quotations and citations, which have been duly acknowledged. The thesis has not been accepted for any degree and is not concurrently submitted in the candidature of any other degree.

Khanayeem

Name of the student: Nayeem Ahmad Khan

Date: 03-08-2018

DEDICATION

This work is dedicated to my spiritual father

Shaykh Engineer Zulfiqar Ahmad Naqshbandi Mujaddidi

ACKNOWLEDGEMENT

All praise is due to Allah, the Lord of the Worlds.

And peace & blessings be upon Prophet Muhammad- Last & Final Messenger of Allah.

Foremost, I am very grateful to my supervisor Assoc. Prof. Dr. Johari bin Abdullah, for his continuous help, guidance, and motivation throughout my Ph.D career without whom it won't have been possible to complete my Ph.D.

With profound reference and high regards, I am thankful to my co-supervisor Dr. Adnan Shahid Khan for his support and teaching me all the essential skills for scientific research.

My special thanks, Dr. Showkat Ahmad Bawani, Faculty of Resource Science & Technology, Universiti Malaysia Sarawak and my post-graduate fellows Raza Zaman and Navien Chandran for their support.

I am also thankful to my friends from Kuching especially Abang Habib, Abang Royhatas Kaloski, Abang Abdul Latif bin Sedik and Tuan Haji Ali Hussaini for making my stay comfortable in Kuching.

I would also like to thank my family for motivating me to do Ph.D and their unconditional support throughout my life.

I also wish to extend my appreciation to Universiti Malaysia Sarawak for the Zamalah award which provided me a fellowship and made pursuing my Ph.D degree possible.

ABSTRACT

The rapid expansion of the Internet and rich features of web application has led to many security flaws related to web application. Some flaws are due to poor programming practices while some are intentionally scripted by the attackers behind the scene. Among the reported vulnerabilities by Open Web Application Security Project has ranked cross-site scripting as the second most dangerous vulnerability among top ten vulnerabilities. Cross-site scripting attacks are conducted by using malicious JavaScript's on the client side. Due to the growing adoption of JavaScript's as the scripting language for the World Wide Web has led to the exploitation of cross-site scripting. Detecting these malicious JavaScript attacks with high accuracy on client side has thus becomes imperative. In this research, a detailed and systematic study was performed on the genesis of malicious JavaScript attacks and proposed a unified approach by combining static and dynamic analysis using minimalistic emulation of browser for performing comprehensive characterization of webpage features. This approach leverages features from URL and structural features of webpage to train classification model. A Parteo-optimised majority vote based multi-layer heterogeneous ensemble approach is also proposed and implemented for accurate classification of webpages in malicious and benign classes. Evaluation of the proposed approach on a corpus of webpages demonstrates that the approach is precise and effective enough in detecting malicious web pages with an accuracy of 98.74% along with sensitivity and specificity of 97.56% and 98.86%, respectively and also have minimal performance overheads in detection of malicious JavaScript code attacks.

Keywords: JavaScript, XSS, malware, detection, static analysis, dynamic analysis, ensemble

Pengklasifikasi Ensembel Terhadap Serangan JavaScript Berbahaya Menggunakan Ciri-ciri Diperluaskan

ABSTRAK

Internet berkembang pesat kerana ia memudahkan maklumat global, komunikasi, kepakaran, dan pengetahuan serta-merta. Jaringan Sejagat adalah pintu masuk kita ke dunia maklumat apabila melayari Internet. Pengguna Internet biasanya melawat banyak laman sesawang setiap hari baik untuk tujuan perniagaan, hiburan atau pendidikan. Walau bagaimanapun, kekayaan ciri-ciri sesawang juga membawa kepada beberapa kelemahan keselamatan sama ada disebabkan oleh amalan pengaturcaraan yang lemah atau skrip berniat jahat yang dilancarkan oleh penjenayah siber di belakang tabir. Oleh itu, sesawang menjadi terdedah kepada aktiviti berniat jahat yang mensasarkan pengguna yang tidak menyedarinya. Di antara kerentanan yang dilaporkan oleh Projek Keselamatan Aplikasi Terbuka Sesawang, pengskripan lintas tapak adalah kelemahan kedua paling berbahaya daripada sepuluh kelemahan. Serangan skrip lintas tapak dijalankan oleh penyerang melancarkan JavaScript yang jahat kepada pihak klien. Oleh kerana penerimaan JavaScript semakin meningkat sebagai bahasa skrip untuk Sesawang, ia telah membawa kepada eksploitasi skrip lintas tapak. Pengesanan serangan JavaScript yang berniat jahat dengan ketepatan yang tinggi pada pihak klien oleh itu menjadi satu cabaran yang besar. Dalam projek penyelidikan ini, kajian yang konsisten dan sistematik telah dilakukan terhadap serangan JavaScript yang jahat dan membangunkan pendekatan bersatu dengan menggabungkan analisis statik dan dinamik melalui emulasi minimalistik penyemak imbas Sesawang untuk melaksanakan ciri-ciri menyeluruh ciri laman sesawang. Pendekatan ini memanfaatkan ciri dari URL dan ciri struktur laman sesawang untuk melatih model klasifikasi. Penilaiar dan pelaksanaan pendekatan berkumpulan yang mempunyai pelbagai lapisan heterogen, dioptimumkan secara Parteo dan undi majoriti untuk pengkelasan tepat

halaman sesawang dalam rejim yang berbahaya dan jinak telah dijalankan. Penilaian prestasi pendekatan yang dikemukakan pada gugusan beberapa laman sesawang membuktikan bahawa pendekatan itu cukup tepat dan berkesan untuk mengesan laman sesawang berniat jahat dengan ketepatan 98.74%, kepekaan 97.56% dan spesifikasi 98.86%. Lagipun, pendekatan yang dibentangkan mempunyai kos prestasi yang minimum dalam pengesanan serangan kod JavaScript yang berniat jahat.

Kata kunci: *JavaScript, XSS, malware, pengesanan, statik analisis, dinamik analisis, berkumpulan*

TABLE OF CONTENTS

		Page
DECLARATION		i
DEDICATION		ii
ACKNOWLEDGEMENT		iii
ABSTRACT		iv
ABSTRAK		v
TABLE OF CONTENTS		vii
LIST OF TABLES		xi
LIST OF FIGURES		xii
LIST OF ABBREVIATIONS		xiii
CHAPTER 1: INTRODUCTION		1
1.1	Research Introduction	1
1.2	Problem Statement	5
1.3	Thesis Contribution	6
1.4	Objectives of the Study	7
1.5	Research Questions	7
1.6	Research Scope	8
CHAPTER 2: TECHNICAL BACKGROUND AND RELATED WORKS		9
2.1	Introduction	9
2.2	Technical Background	10
2.2.1	Stored XSS (Persistent XSS)	10

2.2.2 Reflected XSS (Non-Persistent XSS) 10

2.2.3 DOM-based XSS 11

2.3 Impact of XSS on Browsers 11

2.4 Origin of URL based Attacks 12

2.5 Understanding JavaScript 14

2.6 JavaScript Security Model 17

2.7 Malware Analysis Techniques 18

2.8 Malware Detection Using Machine Learning 18

2.8.1 Naïve Bayes Classifier 20

2.8.2 Support Vector Machines Classifiers 20

2.8.3 K-Nearest neighbor Classifier 22

2.8.4 Quadratic Discriminant Analysis Classifier 23

2.8.5 Logistic Regression 24

2.8.6Artifical Neural Network 25

2.8.7 Random Forests 25

2.9 Feature Selection 26

2.9.1 Chi-Square (χ^2) Feature Selection 27

2.10 The Ensemble 29

2.11 Related Works 31

2.11.1 Static Analysis 31

2.11.2 Dynamic Analysis 35

CHAPTER 3 : PROPOSED APPROACH 40

3.1 Introduction 40

3.2 Overview of Proposed Approach 40

3.3 Motivation for Hybrid and Ensemble Approach 42

3.4 Performance Evaluation 46

3.5 Software Tools Used 47

CHAPTER 4: MALICIOUS JAVASCRIPT ATTACK DETECTION USING HYBRID FEATURES 50

4.1 Overview 50

4.2 Hybrid Feature Approach and Methodology 51

4.2.1 Data Collection 52

4.2.2 Feature Extraction and Labelling 54

4.2.3 Feature Selection 56

4.2.4 Classification 57

4.3 Comprehensive Characterization of Webpage Features 57

4.3.1 Execution Time Features 58

4.3.2 Structural Features 61

4.3.2.1 URL Features 61

4.3.2.2 Webpage Source Features 65

4.4 Implementation and Experimental Procedure 71

4.4.1 Dataset Collection and Validation 72

4.4.2 Feature Effectiveness 73

4.4.3 Empirical Evaluation 74

4.4.4 Classification 76

4.5 Performance Overheads 77

4.6 Discussion on Results 77

4.7 Summary 79

CHAPTER 5: MALACIOUS JAVASCRIPT CLASSIFICATION BY ENSEMBLE 80

5.1 Overview 80

5.2 The New Ensemble Approach 81

5.2.1 Level 1: Selection of Classifier 82

5.2.2 Level 2: Selection of Classifier 83

5.2.3 Level 3: Selection of Classifier 84

5.2.4 Pareto Optimized weighted Voting Ensemble 84

5.2.5 Improved Bagging 86

5.3 Results and Discussion 88

5.4 Summary 92

CHAPTER 6: CONCLUSION AND FUTURE WORK 93

6.1 Conclusion 93

6.2 Future Works 94

REFERENCES 96

APPENDIX 116

LIST OF TABLES

		Page
Table 4.1	Summary of Execution Time Features	59
Table 4.2	Summary of URL Features	62
Table 4.3	JavaScript Features	67
Table 4.4	Summary of Extracted Features	69
Table 4.5	Results of generating Chi-Square values, p-values, and weight of features	74
Table 4.6	Classification Results	76
Table 5.1	Comparison of Present Ensemble Approach with Individual Classifiers	89
Table 5.2	Comparison of Present Ensemble Approach with State of the Art Ensembles	89
Table 5.3	Computational Time Comparison of Present Ensemble Approach Individual and Ensemble Classifiers	89

LIST OF FIGURES

		Page
Figure 2.1	XSS Attack Scenario	11
Figure 2.2	URL-based attack using obfuscated malicious code	13
Figure 2.3	Document Object Model	16
Figure 3.1	Schematic depiction of the Present Approach	41
Figure 3.2	The process of Hybrid Feature Extraction	43
Figure 4.1	Block diagram of Hybrid Feature Approach	52
Figure 4.2	Schematic representation of Data Collection Process	53
Figure 4.3	Outlining process of URL Feature Extraction	54
Figure 4.4	Illustrating process of JavaScript Feature Extraction	55
Figure 4.5	Explaining process of Feature Selection	57
Figure 4.6	Displaying process of Hybrid Feature Extraction	59
Figure 4 7	Statistical Performance of each Feature	75
Figure 5.1	Exposition of New Ensemble Approach for Detection of Malicious Code	81
Figure 5.2	Flowchart for Improved Bagging	88
Figure 5.3	Comparison of Accuracies between Present Ensemble Approach with Individual Classifiers and State of the Art Ensemble Approaches	90

LIST OF ABBREVIATIONS

AJAX	Asynchronous JavaScript and Extensible Markup Language
API	Application Programming Interface
BHO	Browser Handling Object
CERN	Conseil Européen pour la Recherche Nucléaire
CPU	Central Processing Unit
DOM	Document Object Model
DHTM	Dynamic HyerText Markup Language
FPR	False Positive Rate
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
JPG	Joint Photographic Experts Group
K-NN	K-Nearest Neighbor
MD5	Message Digest 5
MLP	Multilayer Perceptron
MOO	Multi-Objective Optimization
OWASP	Open Web Application Security Project
PDF	Portable Document Format
QDA	Quadratic Discriminant Analysis
RBF	Radial Basic Function
SVM	Support Vector Machines
SQL	Structured Query Language
TPR	True Positive Rate
TPU	Tensor Processing Unit

URL	Uniform Resource Locator
XSS	Cross-Site Scripting

CHAPTER 1

INTRODUCTION

1.1 Research Background

The invention of the Web by Tim Lenard-Bee in 1989, just a handful of pages then used by a small number of scientists at CERN Labs Geneva, has now evolved into a global platform consisting of trillions of pages used by millions of users (Berners & Fischetti, 2001). Pervasive influence of the Web has brought positive changes in the life patterns of people through executing business deals, establishing social contacts, disseminating information, sharing knowledge and exchanging ideas (Oyza & Edwin, 2015). The Web is evolving very rapidly and new features are being introduced for making human lives better. The human dependence on the Web is also increasing day by day in order to harness the power of the Web.

Regretfully, the Web is being exploited by the cyber criminals using malicious code attacks to attain personal, financial, military and political gains through damaging or disrupting the normal operation of computer systems, performing phishing attacks, displaying unwanted advertisements and extorting money (Khan et al., 2017). It has become a routine to visit numerous webpages daily for searching information, making communication, reading news and watching videos. It just takes a simple visit of a webpage to become the victim of these attacks. A common tactic employed by cyber criminals to perform these attacks is to lure the users who lack awareness to their webpages that are intentionally designed and coded to perform attacks while visiting such webpages (e.g. spam email and malicious advertisements) or to put at risk a legitimate webpage under the control

of cyber criminals (Li et al., 2012; Zarras et al., 2014). The targeted users are either tempted to give away sensitive information or their Web browser vulnerabilities are exploited to deliver the malware on their computers to perform an attack such as Plugins and extensions (Provos et al., 2007).

Earlier the design and development of such malicious code attacks were only limited to experienced attackers who could craft a malicious webpage but now inexperienced attackers can also launch attacks with special software called exploit kit (Kotov & Massacci, 2013). Exploit kits are developed by a group of experienced attackers and sold illegally in black and gray markets. These exploit kits are constantly updated to launch new attacks in webpages and to evade de-obfuscation of malicious code (Ablon et al., 2014).

As per Internet Security Threat Report released by Symantec in April 2017 more than 357 million new variants of malware were observed in 2016 (Symantec, 2017). A report from Open Application Security Project (OWASP) has ranked cross-site scripting (XSS) among top ten vulnerabilities affecting webpages (OWASP, 2017). XSS is a type of attack injecting malicious scripts around benign code in a legitimate webpage to access a cookie, a session, and other secret information. Web applications are exploited to transport malicious scripts to perform the attack. The target of XSS attack is on the client-side and the vulnerability at the application layer of the network hierarchy occurs by injecting malicious scripts to break the security mechanism (Gundy & Chen, 2009). About 70% attacks are reported to occur at the application layer and Web browsers are the most susceptible application-layer software for attacks (Mohan & Malik, 2017). The purpose of the Web browser is to get the requested resource from the server and display it in browser's windows. The format of the supplied resources is not restricted to HyperText Markup Language

(HTML) only but can also be applied to portable document format (PDF), joint photographic experts group (JPG), etc. Attackers run malicious JavaScript in a Web browser to target users whereas malicious and obfuscated URLs are used in servers as a carrier for XSS attacks (Nunan et al., 2012).

JavaScript is used for making webpages more interactive, to add more features, to improve end-user's experience and to help reduce server-side load by shifting some computation tasks to end user-side. Unfortunately, JavaScript also provides a strong base for carrying out attacks, especially drive-by-download attacks, which unnoticeably attacks client amid the visit to a webpage. In contrast to different sorts of network-based attacks, malicious JavaScripts are difficult to detect (Schütt et al., 2012). Javascript attacks are performed by looking into the vulnerability and exploitation by using JavaScript obfuscation technique to evade detection. Thus detection of such malicious JavaScript attacks in real time becomes imperative. Security solution providers who play a vital role in providing tools for defending attacks are fundamentally based on two correlative approaches: signature-based and heuristic-based detection approaches (Khan et al., 2017). The signature-base approach depends on the detection of unique string patterns in the binary code but fails when a new type of attack occurs. In this approach, the instance of new attack needs to be captured first in order to create a signature from it and subsequently update the definitions of the detection tool on the client-side (Cova et al., 2010). Since the time gap between the capturing of new attack instance and the updating at the client-side is high, it renders millions of client-side computers vulnerable to attacks. Cyber criminals take advantage of this time gap and vulnerability to simultaneously launch the attack. Consequently, this approach is unacceptable in an environment where new attacks are expected to venture. Another traditional approach for attack detection is heuristic-based detection rooted in a set of expert

decision stipulations to detect the attack. The advantage of using this attack detection approach is that it cannot only detect modified or variant of existing malware but can also detect the previously unknown attacks. However, the downside of using this approach is that it takes a long time in performing scanning and analysis, which drastically slows down the security performance. Another trouble with the approach is that it introduces many false positives (Liao et al., 2013). False positive occurs when a system wrongly identifies code or a file as malicious when actually it is not. In order to address these issues with signature and heuristic-based detection approaches, two complementary approaches namely static analysis and dynamic analysis were introduced (Laskov & Srndic 2011). In static analysis detection of malicious code inside a webpage is performed by quickly extracting features and without rendering the webpage in a browser. Dynamic analysis captures the malicious code by analyzing the behavior of a code when the webpage is in execution by the browser (Eshete et al., 2012).

The Web has become more and more vulnerable with the growing number of malicious code attacks regardless of a number of available defense mechanism (Thomas et al., 2011). To explore the status of malware on the Web, a study was conducted by investigators (Provos et al., 2012), who scrutinized about 4.5 million webpages indexed in Google and found that 1 in every 10 webpages may contain malicious code. In another study the investigators (Provos et al., 2007) identified about 3 million URLs on the Web that launch attacks against users. A study conducted by researchers (Canali & Balzarotti, 2013) employed about 500 honeypot sites to analyze the behavior of webpages prior and post compromise by an attack. The results elucidate that XSS attacks, phishing, drive-by-downloads, and defacement of sites are prevalent on the Web.

The fore-mentioned studies have established that malicious webpages act as a primary vector to transmit malicious activities on the Web. In this thesis we present a technique to improve upon the state of the art approach with the dual purpose to perform effective analysis and efficient detection of malicious activities on the Web. Hence, the main emphasis of this thesis is on the perspective of detecting malicious JavaScript-based XSS attacks and identifying malicious URLs.

1.2 Problem Statement

The ubiquitous revolution of modern-day computer systems, augmentation of networking infrastructural facilities and boosting up of internet services are primarily subservient to Web browsers and, therefore, these browsers have become the main target by attackers. Malicious JavaScript codes are injected around the benign code of a webpage usually in the obfuscated form to target client-side users. Once the user visits such an infected webpage, the malicious code gets executed on the client-side to compromise its information. The Web browsers are meek in capabilities in detecting such attacks and, therefore, pose great threat to end-user security.

A series of existing approaches based on machine learning have been put forth to detect such attacks (Canfora et al., 2014; Sirageldin et al., 2014; Jodavi et al., 2015; Aebersold et al., 2016). The flaw with these approaches is that they fail to detect attacks on three counts.

First, the weedy analysis and deficient characterization of malicious code results in a malicious page to escape the detection. Existing countermeasures are either based on static or dynamic analysis in detection of malicious JavaScript code attacks. The challenge with static analysis is that it cannot detect the complex and obfuscated malware. However through

dynamic analysis the behaviour of malicious JavaScript is known but has a challenge that it takes long time in processing and analysing the malware for the detection and works in a runtime environment which introduce performance overheads.

Second, the artefacts of a webpage evolve with time and old artefacts became less relevant due to emergence of new variants of malicious code that leads to poor detection. Two reasons through which artefacts are constantly evolving over time and presents a challenge for detection are: First the cybercriminals regularly keeps on changing their methods to craft attack payloads in malicious web pages with the purpose of making the attack more complex and avoid detection by existing countermeasures. Second New functionality is being introduced in benign web pages or change in the underlying technology which are employed in developing web pages.

Third, the classification error introduced by single classification algorithms cannot cope up with diverse malicious codes which limit the detection. Since there is no agreement yet which classifier produces the best prediction results. A particular classifier might be better than its counterparts for a particular dataset; nevertheless another classifier could achieve better results for some other dataset.

1.3 Thesis Contribution

The major contributions made in this thesis can be summed up as follows:

1. Developed and implemented a technique for comprehensive analysis, meticulous characterization, and effective detection of malicious webpages by performing an amalgamation of static and dynamic analyses by using minimal emulation.

2. Introduced and evaluated new features and improved several existing features to boost their discriminative power in categorizing malicious and benign webpages.
3. Presented and analysed a multilevel ensemble approach based on Pareto optimization to boost the detection rate of Web attacks.

1.4 Objectives of the Study

The main goal of this research work is to design and develop an approach convenient for automated detection of malicious JavaScript attacks at client-side. Specifically, this goal can be accomplished through the following objectives:

i. To develop a hybrid analysis approach for detection of malicious JavaScript attacks on the client-side.
ii. To investigate newly inducted features and to improve old features those are decisive in the detection of malicious code attacks.
iii. To formulate an algorithm for computing the effectiveness of extracted features.
iv. To probe supervised classifiers and to devise ensemble approach for proficient detection of malicious JavaScript code.
v. To execute the performance evaluation of the proposed mechanisms.

1.5 Research Questions

1. How can a hybrid analysis platform be built to facilitate the comprehensive analysis and proactive detection of malicious JavaScript attack?
2. What are the crucial features involved in the classification of malicious JavaScript? How to recognize the effectiveness of selected features?

3. How will a multilevel-based heterogeneous ensemble approach lead to elevated detection accuracy?

1.5 Research Scope

This study focuses on a hybrid analysis approach consisting of static and dynamic analysis for the detection of malicious JavaScript code attacks. For dynamic analysis, a minimalistic emulation on dynamic features was employed. For feature selection, an algorithm for calculating the effectiveness of the extracted features using Chi-square and feature weighting was introduced. For classification, several supervised classifiers were tested and an ensemble approach was introduced to improve the detection accuracy. The dataset used in this study were collected from the real word environment consisting of malicious and benign webpages and URLs.

CHAPTER 2

TECHNICAL BACKGROUND AND RELATED WORKS

2.1 Introduction

Malicious programs have become a serious problem that threatens the security of billions of Internet users. Attackers are constantly involved in developing techniques to steal sensitive data from computer systems through Web applications. Applications which are vulnerable to malicious users can break into the security and protection mechanisms of a system by gaining access to personal information or by taking control over system resources. The purpose of the attack to get access to personal information or system resources is to cause damage to digitized assets of individuals and organizations. Any individual or an organization which has its existence over the Web has some amount of exposure of being attacked and the level of risk varies depending on various factors. Among the reported vulnerabilities, OWASP has ranked in XSS the second most dangerous vulnerability among top ten vulnerabilities and the first attack of XSS was reported in early 90's. The target of XSS attack is client-side whereas SQL injections target the server (Gundy & Chen, 2009). Web browsers are considered as most vulnerable application layer software for attacks (Grobauer et al., 2011). The purpose of the Web browser is to acquire the requested Web resource from the server and to display it in browser's windows. The format of the supplied resource is not restricted to HTML only but can also be PDF and images. Attackers run malicious JavaScript in a Web browser to target users. XSS attack is vulnerability at the application layer of the network hierarchy by injecting malicious scripts to break the security mechanism. The drawback with some existing protection and security approaches including filtering, auditing system, black box testing, and content security policy is to degrade the

performance of the system with configuration overhead. Many researchers (Nadji et al., 2009; Pelizzi & Sekar, 2012; Xiao et al., 2014) have strongly backed an argument that there is no such server-side approach available which can completely stop these types of attacks against users via Web browsers. In this chapter the technical background and related works are presented.

2.2 Technical Background

XSS is a vulnerability that allows attackers to inject malicious code into the source-code of a webpage to be executed at the victim's browser. If the malicious code gets successfully executed into the victim's Web browser then the attacker captures the control of the victim's resources and sensitive data. Figure 2.1 outlines the principle of XSS attack. There are three types of XSS vulnerabilities: Stored, Reflected, and DOM-based.

2.2.1 Stored XSS (Persistent XSS) Attack

Stored XSS attack occurs when malicious Javascript is stored on the target-server in the database or guest book's message forum. In Persistent XSS attack, malicious scripts get executed when the user visits the malicious site; thereby passing the privileges of the user to the attacker who then undertakes unauthorized actions without user's knowledge (Baranwal, 2012).

2.2.2 Reflected XSS (Non-Persistent XSS) Attack

In reflected XSS attack, the attacker injects the malicious code into the server. In non-persistent XSS attack, the injected code is reflected back to the attacker in the form of error message or search result which may include some or part of inputs provided to the server as

a request. Then reflected XSS attacks are sent to target-victims through email or links embedded on the webpage to steal the confidential information or to take control over the victim's computer (Pelizzi & Serkar, 2012).

2.2.3 DOM-based XSS Attack

In Document Object Model (DOM)-based XSS attack, the entire tainted data flow from source to sink in the Web browser. The source of XSS can be any HTML element or the webpage's URL, while any method with the sensitive call can cause malicious code to be executed in the sink (Stock et al., 2014).

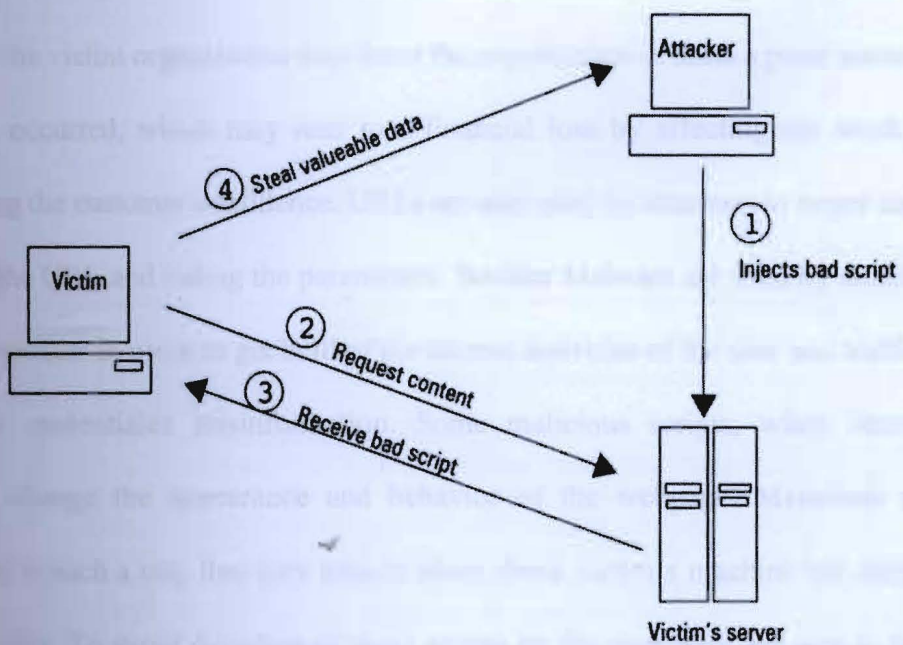


Figure 2.1: XSS Attack Scenario (Bajpai, 2010)

2.3 Impact of XSS Attack on Browsers

An XSS attack occurs due to the flaw in server-side applications and the reason behind the flaw is weak sanitization of HTML characters provided by the user as input. The browser is

considered as the most directly affected application from XSS attack on-client side (Pelizzi & Serkar, 2012). The browser interprets and displays HTML pages, executes Java's scripts, and other contents, which are sent from server to client. The content hosted on the Web server may be malicious with the intention to target users. Some of the common techniques by which attackers target users through a browser are cookie and session stealing, browser hijacking, browser-history sniffing, web-user behaviour tracking, buffer overflowing, format string attacking, pointer dangling, integer flowing, and drive-by downloading. A variety of other ways exists through which sensitive information maintained by the browser is either stolen or access to resources denied. The result of such attacks leads to cookie and session information leakage, and end-user documentation disclosure. Other consequences of such attacks on the victim organization may force the organization to issue a press statement about the attack occurred, which may lead to a financial loss by affecting the stock price and diminishing the customer confidence. URLs are also used by attackers to target end-users by encoding the URL and hiding the parameters. Besides Malware are used by attackers to spy on Web browsers in order to get hold of the current activities of the user and traffic statistics leading to credentialed misinformation. Some malicious scripts, when interrupted by browsers, change the appearance and behavior of the webpage. Malicious scripts are engineered in such a way that they remain silent about victim's machine but only make the impact visible. To avoid detection of these scripts by the user, they are sent to the victim's browser for execution in small batches.

2.4 Origin of URL-Based Attacks

Malicious URLs are created with an intention to attack users by downloading malicious code or malware into the target-machine, which can be contained in a spam or

phishing mail. Phishing with malicious URL is considered as one of the most common methods of launching a direct attack. Malicious URLs can also be found inside the source code of malicious webpage which operates by finding vulnerability in an application to perform the attack. The malicious code gets executed when a user clicks on a URL without having any knowledge of the attack. The innocent-user once clicks on the URL, the attacking party analyses the type of operating system, browser, and other plug-ins for vulnerability to carry out the attack which could lead to buffer overflow attack, format string attack, dangling pointer attack, and integer flow attack. In order to bypass data input validation filters and access restrictions for malicious code and malicious URLs, attackers use obfuscation of URLs to initiate an attack. Obfuscation is a mechanism to deliberately hide details by using encoding schemes such as Hexadecimal, Decimal, Octal, Unicode, and Base64 that are widely used in attacks (Likarish et al., 2009). Figure 2.2 illustrates the obfuscated URL with malicious code and has a size larger than URL without encoding.

```
1 http://www.trustedsite.com/search.html?type=<"<<sCrIpT>alert
2 (document.cookie)</ sCrIpT><"<< sCrIpT>alert(document.cookie)
3 </sCrIpT>
4 hhtp:// www.trustedsite.com/search.html?type=%3C%22%3C%3C
5 %73%43%72%49%70%54%3E%61%6C%65%72%74%28%64%
6 %6F%63%75%6D%65%6E%74%2E%36%6F%6F%6B%69%
7 %65%29%3C%2F%73%43%72%49%70%54%3E%3C%22%3C
8 3C%73%43%72%49%70%54%3E%61%6C%65%65%72%74%28
9 %64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%
10 69%65%29%3C%2F%73%43%43%72%49%70%54%3E
```

Figure 2.2: URL-based attack using obfuscated malicious code (Nunan et al., 2012).

Multiple domains in a single URL are also used to redirect the presented URL to malicious URL, shown as under:

www.trustedsite.com/redirect.php?url=http://www.malicioussite.com

Some malicious URLs are more likely to be advertisement-related. Such type of URLs contain certain keywords like *'ad'*, *'advert'*, *'popup'*, *'banner'*, *'sponsor'*, *'iframe'*, *'googlead'*, *'adsys'*, and *'adser'*. An example of ad URL is given here:

http://www.mycashkit.com/?from=googleads&click_id=ad1&gclid=CjwKEAjw1riwBRD6 1db6xtWTvTESJACoQ04QLHAtV1qDPK4WL2zHS2dDXvPLWUB3j1pvQJkvHcZWxoCRD fw_wcB

2.5 Understanding JavaScript

JavaScript is a cross-platform, dynamic, and object-based language. It was developed by Brendan Eich in 1995 with the aim to come up with the language that particularly runs in the Web browsers (Severance, 2012). Unlike Java language which is considered as heavyweight language chiefly used to create an application by programmers, JavaScript is lightweight in nature designed to be used even by non-programmers. JavaScript supports prototype-based inheritance which is used to add dynamism and make webpage interactive (Chellapilla & Maykov, 2007). In webpage, JavaScript code is embedded into the HTML source code by using tags which start with *<script>* and ends with *</script>* and can also be contained in special attributes such as *form onclick()" and alert('Good Morning!')"*. JavaScript can be embedded into a webpage as an external file between script tags. The most common and preferred way of using JavaScript is to use it in an external file. The JavaScript is defined in an external file with the extension .js and then called on the webpage. The example is given as:

<script type="text/javascript"scr=filename.js"> </script>

Another way of using JavaScript is to use it between script tags instead of specified source as shown in the following example:

```
<Script type "text"/javascript">
Var=myvar="Good Morning";
Function ShowAlert() {alert('you triggered an alert!');}</script>
```

One more way of using JavaScript in a webpage is to use it inline with HTML. An example of using JavaScript inline is:

```
<p><a href="#"onClick="alert"('Hello');> ClickMe</a></p>
```

In addition to foregoing way of using JavaScript inline with HTML code, JavaScript can also be used with some elements associated with an even attribute that calls JavaScript when an event occurs such as *OnClick, OnMouseOver,* and *OnMouseOut*. The features inside the webpage are manipulated and accessed by JavaScript code using DOM. DOM of a webpage is created once the webpage gets loaded into the web browser and represents various elements of the webpage in the form of a tree as shown in Figure 2.3. The document object has many properties that are essential to allow access and modify the content. The nodes of the tree are called as DOM objects in which window objects are at the root of the DOM tree and the document object represents the main HTML document. The location object represents the address link of the webpage whereas the cookie object represents the cookie associated with the webpage.

JavaScript is a versatile, powerful and dynamic language when used properly but can be potentially dangerous when used to target users (Rauschmayer, 2012; Guldamlasioglu, 2015). Every web browser has a feature of enabling and disabling the JavaScript, but disabling the JavaScript is not recommended as many of the features will not work. The websites can become slower to respond and even some website will not open due to the

disabling of JavaScript. Therefore, it becomes indispensable to have JavaScript on each machine and even modern-day smart mobile phones are JavaScript enabled. Keeping in view various access points of JavaScript that can be exploited by the cybercriminals to launch the attack, it becomes impracticable for the programmers to make JavaScript attack proof. The most general exploited vulnerability is cross-site script. Thus some additional security protection is necessitated to shield any attack on the client-side.

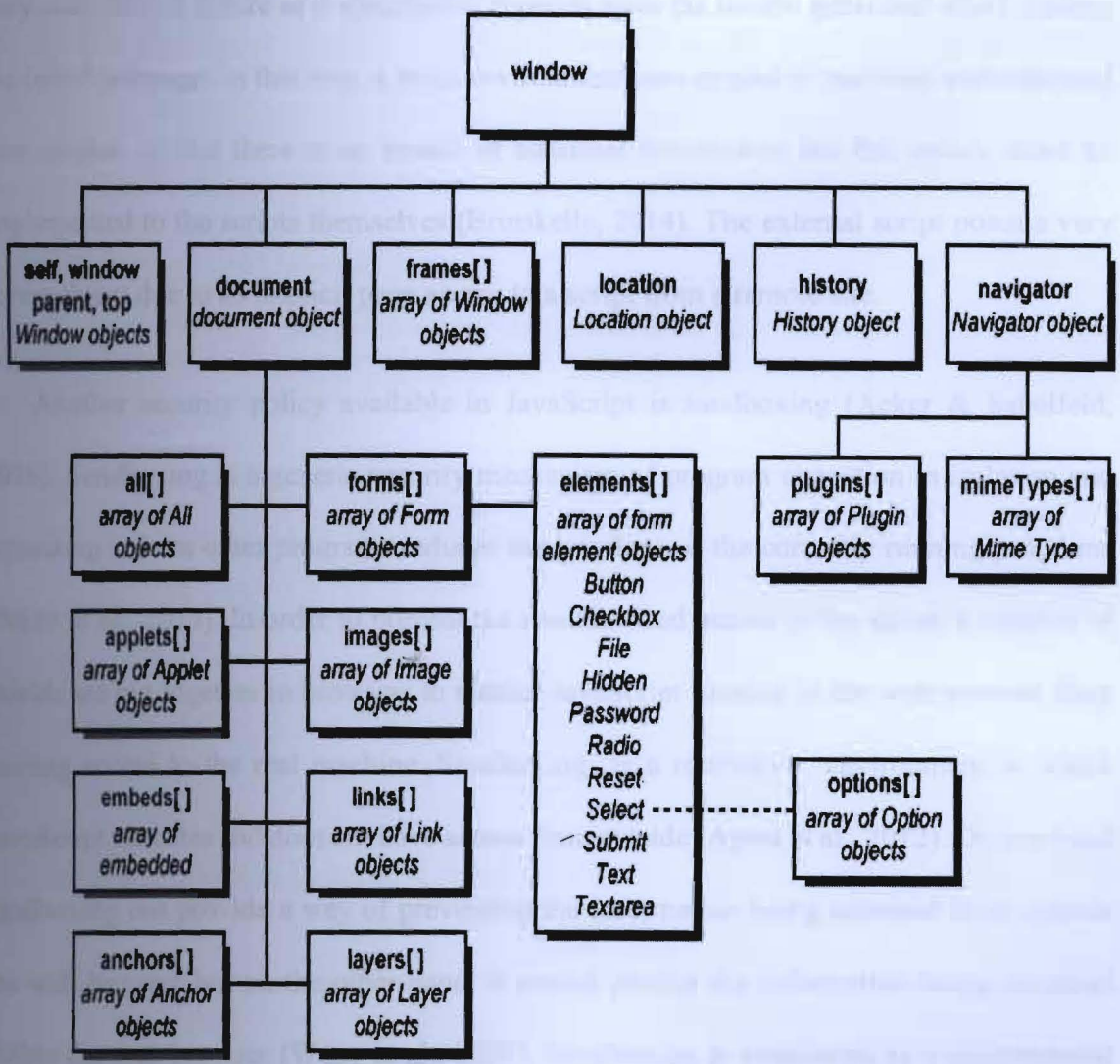


Figure 2.3: Document Object Model (Codingsec, 2016)

2.6 JavaScript Security Model

JavaScript security model is based on Java and there are two types of security policies in JavaScript: (1) same origin policy and (2) sandboxing policy. The same origin policy is believeed to be one of the crucial concepts in the security model for a Web application (Stefan et al., 2014). In this policy, the web browser permits accessing of the scripts from other webpage provided that webpage belongs to the same origin. The origin can be URL, port number, hostname or any combination of these (Barth et al., 2009). This mechanism is very restrictive in nature as it specifically depends upon the cookie generated while loading the initial webpage. In this way, a strict environment gets created to maintain authenticated user session so that there is no breach of sensitive information but this policy canot be implemented to the scripts themselves (Brinskelle, 2014). The external script poses a very severe threat due to its implicit page access to a script from a remote site.

Another security policy available in JavaScript is sandboxing (Acker & Sabelfeld, 2016). Sandboxing is a generic security mechanism of program execution in isolation and separating it from other programs reduces the privilege of the currently running programs (Politz et al., 2015). In order to prevent the unauthorized access to the client, a number of shields are put together in browsers to restrict JavaScript running in the web browser from gaining access to the real machine. Sandboxing is a restrictive environment in which JavaScript executes and does not have access from outside (Agten et al., 2012). On one hand sandboxing can provide a way of preventing the information being accessed from outside the web browser but on the other hand, it cannot protect the information being accessed within the web browser (Wang at al., 2007). Sandboxing is considered as a controversial approach because its complexity can cause more security problems (Agten et al., 2012).

Sandbox has to contain all the files inside which the applicant needs to execute, this can create a problem between two applications that needs to interact with each other (Acker & Sabelfeld, 2016).

2.7 Malware Analysis Techniques

Malware analysis is the systematic process of exploring the functionality, origin, and potential impact of a given malware sample (Rajeswari, 2017). Malware analysis involves two main techniques: static analysis and dynamic analysis. Static analysis involves reviewing, testing and examining the source code or bytecode of an application without executing the application to find the faults. Static analysis allows analyzing the data flow, checking the syntax and verifying if the states of application are finite (Pérez et al., 2011). Signature-based methods are primarily used to implement static analysis but rely on the identification of unique strings in binary code (Griffin et al., 2009; Cova, 2010). Static analysis has an advantage that it provides a rapid classification to detect a malicious file without executing it (Potashnik et al., 2011). In the dynamic analysis, the process of testing, evaluating and collecting of information from the operating system takes place in real time during program execution (Kruegel, 2014). The objective is to detect any malicious code or malware when a program is in running state rather than by evaluating the code offline (Rao & Hande, 2017).

2.8 Malware Detection Using Machine Learning

The signature-based methods for detection of malware are limited to only detection of previously known malware samples and have proven ineffective against unknown malicious code (Shabtai et al., 2012). One such method to potentially keep up with the

antivirus avoidance techniques used in modern malware is to augment existing detection systems with machine learning classifiers (Markel, 2015). The basic idea of any machine learning task is to train the model based on some algorithm and to perform a certain task: classification (Nagai et al., 2009), clusterization (Berkhin, 2006) and regression (Olson et al., 2007). Training is based on the input dataset, and the model built subsequently makes predictions. The output of such a model depends on the initial task and the implementation. The classification algorithms in machine learning enable the construction of classifier which automatically learns the characteristics of each class (Kotsiantis et al., 2007), such as malicious and benign code, by learning from example data to predict whether a given code sample is malicious or benign. Machine learning malware detection which stands introduced but still in developmental stage, provides an increased robustness in detection of newly launched marginally modified threats that hold some of the characteristics of the past malware (Markel, 2015; Pham & Hoang, 2016). To use machine learning to classify code segments as malicious or benign, one must first build labelled datasets for training. A set of records—one for each benign and malicious file comprises the dataset. Each record contains a set of features and one label (also referred to as the class). The features of each file are derived from some specific characteristics of the file, such as size of the file or frequency of a certain code snippet in the file; the label is a binary value indicating whether or not the file is malicious. Learning algorithms analyse records designated for training to generate a mathematical model that maps the relationship of file features and labels. The model, which is called the classifier, is used to predict the class of each record in the test data, or the records designated for testing. As the training data exists in the classification, it is referred as supervised learning (Ngai et al., 2009). The mathematical formalism for the supervised classifiers used in this thesis is given as:

2.8.1 Naïve Bayes Classifier

Naïve Bayes classifier technique also known as Bayesian classifier is based on Bayes theorem with independent assumption between predictors (Lewis, 1998). In the context of learning the process, the classifier tokenizes training data to some tokens c_j (j = 1 . . . n) and counts the number of occurrences of c_j in each class. Based on this process, the likelihood of each class is computed with some test data and that test data is then classified for the class which has the highest likelihood. Despite its simplicity, Naïve Bayes classifier is widely used as it surpasses more sophisticated classification methods. The Bayesian classifier is given by:

$$p(c_j|d) = \frac{p(d|c_j)p(c_j)}{p(d)}, \qquad 2.1$$

where

$p(c_j \mid d)$ = probability of instance d being in class c_j,

$p(d \mid c_j)$ = probability of generating instance d in given class c_j,

$p(c_j)$ = probability of occurrence of class c_j, and

$p(d)$ = probability of instance d occurring.

In this research study, there are only two classes: malicious and benign for classification, the range of j is, therefore, from 0 to 1only.

2.8.2 Support Vector Machines Classifier

Support Vector Machines (SVM) classifier was developed by Boser et al. in 1992. However, the original optimal hyperplane algorithm was introduced by Vapnik & Lerner in 1963. SVM classifier is considered as one of the most effective models for binary

classification of high dimensional data. An SVM performs classification by constructing an N-dimensional hyperplane that optimally separates the data into two categories to maximize the distance of the hyperplane. SVM can also be extended to the data that is not linearly separable by applying kernel to map the data into a higher-dimensional space and to separate the data on the mapped dimensions. For SVM, mostly linear, Gaussian and polynomial kernels are used depending upon the classification. Linear kernels are used when the number of features is larger than the observations whereas Gaussian kernels are used when the number of observations is larger than the features. Polynomial kernels are well suited for problems where all the training data is normalized.

The Gaussian kernel also known as Radial Basis Function (RBF) kernel is commonly used in the kernel classification. The RBF kernel is defined as: $K(x,y)=\exp\left[-\gamma\|x-y\|^2\right]$, where $\|x-y\|^2=<x-y,x-y>$ is the squared-norm (inner product) of the distance between space vectors (x, y) and γ the randomization parameter that gives the spread (width) of the Kernel. If the two vectors are close together then $\|x-y\|^2$ will be small and $-\gamma\|x-y\|^2$ will be larger for $\gamma>0$. Thus, closer vectors have a larger RBF kernel value than farther vectors. The polynomial kernel represents the similarity of vectors in a feature space over polynomials of the original variables, allowing learning of non-linear models. Polynomial kernel is represented and defined as $k(x,y) = (\alpha x^T y + c)^d$, where α is an adjustable parameter, c the constant term, d the degree of polynomial and x^T the transpose of x.

2.8.3 K-Nearest Neighbour Classifier

The K-Nearest Neighbour Algorithm (K-NN) is the simplest machine learning algorithm (Altman, 1992). To determine the category of the test data, K-NN performs a test to check the degree of similarity between documents and k training data to store a certain amount of classified data. In this research work, for K-NN classifies instances, the malicious and benign code instances are nearest to the training space. The classification of unknown instances is performed by measuring the distance between the training instance and unknown instance. Sine instances are classified on the basis of majority vote of neighbor; the most common neighbor is measured by a distance function. If $k = 1$, then the instance is assigned for the class of its nearest neighbor. In N-dimensional normed vector space distance between two points (x and y), is achieved by using any distance function. For K-NN, we have used Euclidean, Manhattan, and Minkowski distance functions to measure the value of k.

Euclidean Distance Function:

$$d(x,y)=\sqrt{\sum_{i=1}^{k}(x_i-y_i)^2} \qquad 2.2$$

Manhattan Distance Function:

$$d(x,y)=\sum_{i=1}^{k}|x_i-y_i| \qquad 2.3$$

Minkowski Distance Function:

$$d(x,y)=\left(\sum_{i=1}^{k}\left(|x_i-y_i|\right)^q\right)^{1/q}, \quad q\geq 1. \qquad 2.4$$

For $q=1$ and $q=2$, the Minkowski distance function becomes equal to the Manhattan and the Euclidean distance functions, respectively.

2.8.4 Quadratic Discriminant Analysis Classifier

Quadratic discriminant analysis (QDA), is a machine learning classifier that uses the quadratic surface to separate two or more classes. It assumes that each class has a normal distribution and does not require any parameter to tune the algorithm. QDA allows each class to have its own covariance matrix and tends to fit the data best as compared to linear discriminant analysis (LDA). Following formula is used to determine quadratic discriminant function $\delta_k(X)$ (Narsky & Porter, 2013):

$$\delta_k(X)=-\frac{1}{2}\log|\Sigma_k|-\frac{1}{2}(X-\mu_k)^T\Sigma_k^{-1}(X-\mu_k)+\log\pi_k \qquad 2.5$$

Here μ_k represents mean of each class for k number of classes, Σ_k the covariance matrix for k class, π_k the priori probability of class k, $|\Sigma_k|$ the determinant of Σ_k, and T the transpose of the vector. Following classification rule is used to classify given datasets:

$$G(X)=\arg\max_k \delta_k(X), \qquad 2.6$$

where arg max is the set of points, X, for which $\delta(X)$ attains the function's largest existing value. A class which maximizes the quadratic discriminant function will be considered as output class. The discriminant analysis has reduced error rate and uses multiple dependent variables to determine output class. Moreover, the interpretation between groups is easier to interpret and calculate.

2.8.5 Logistic Regression

Regression is a statistical technique that determines the best model to relate an independent variable (predictor or explanatory) to various independent variables (Cox, 1958). Logistic regression is the type of regression that estimates the probability of a given value of the dependent variable (response or outcome variable), instead of estimating the value of the variable. The outcome is measured with a dichotomous variable (having two possible outcomes), the probability of the presence of the characteristic of interest (p) ranges from zero to one, i.e. $p \in [0,1]$.

The *logit* transformation of a dichotomous variable is defined as:

$$logit(p) = ln\frac{p}{1-p}, \text{ where} \qquad 2.7$$

$$odds = \frac{p}{1-p} = \frac{\text{probability of presence of characteristic}}{\text{probability of absence of characteristic}}. \qquad 2.8$$

$ln\frac{p}{1-p}$, the natural logarithm of the odds, is called *logit* and can be any real number from minus infinity to plus infinity, i.e. $logit = ln\frac{p}{1-p} \in [-\infty, +\infty]$.

The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest and a set of independent variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a logit transformation of p. The logistic regression equation that results in is (Best & Wolf, 2015):

$$logit(p)=ln\frac{p}{1-p}=b_0+b_1X_1+b_2X_2+b_3X_3+.....+b_kX_k, \quad 2.9$$

where the response variable (the probability) p is a function of the coefficients $(b_0,b_1,b_2,.....b_k)$, that correspond to the predictor variables $(X_1,X_2,X_3,.....X_k)$.

Rather than choosing parameters that minimize the sum of squared errors (like in ordinary regression), estimation in logistic regression picks parameters that maximize the likelihood of observing the sample values.

2.8.6 Artificial Neural Network

A neural network is a network of simple processing elements (neurons) that can exhibit complex global behavior, determined by the connections between the processing elements and element parameters. It can be used to model the complex nonlinear relationship between inputs and outputs by learning from experiential knowledge expressed through the connections between the processing elements (Eluyode & Akomolafe, 2013). The multi-layer perceptron (MLP) is a type of artificial neural network (ANN) consisting of a network of neuron layers inspired by the human brain. It has been widely employed by researchers in various fields such as banking, defense, electronics and has a medium level of complexity (Swingler , 1996).

2.8.7 Random Forests

A Random Forest consists of a collection or ensemble of simple tree predictors, each capable of producing a response when presented with a set of predictor values. For classification problems, this response takes the form of a class membership, which associates, or classifies, a set of independent predictor values with one of the categories present in the dependent variable (Ho, 1995). Alternatively, for regression problems, the tree response is an estimate of the dependent variable given to the predictors. A Random Forest consists of an arbitrary number of simple trees, which are used to determine the final outcome. For classification problems, the ensemble of simple trees votes for the most popular class. In the regression problem, their responses are averaged to obtain an estimate of the dependent variable (Gerard & Erwan, 2016). Using of tree ensembles can lead to significant improvement in prediction accuracy that is better able to predict new data cases (Seni & Elder, 2010). The predictions of the Random Forest are taken to be the average of the predictions of a set of k trees with individual weight functions K.

$$\text{Random Forest Predictions} = \frac{1}{k}\sum_{k=1}^{k} K^{th}, \qquad 2.10$$

where the index k runs over the individual trees in the forest.

2.9 Feature Selection

Feature selection also known as variable or attribute selection is the method of selecting a subset of relevant feature used for developing classification model (Hall, 1999). The primary objective of feature selection is to remove the irrelevant and redundant features without incurring much loss of information (Cao et al., 2014). Feature selection techniques

are used for model simplifying (James et al., 2013), training-time shortening (Fan et al., 2005), dimensionality avoiding (Hong et al., 2014), generalization enhancing (Reis, 2016), variance reducing (James et al., 2013; Bermingham et al., 2015), and performance improving (Li & Ho, 2016) in order to gain predictive accuracy (Ladha & Deepa, 2011). A number of feature selection approaches have been introduced including Chi-Square (Ferguson & Schneier, 2003), T-Test (Guyon & Elisseeff, 2003), Euclidian Distance (Dash & Liu, 1997), Information Gain (Yang & Pedersen, 1997), Correlation-Based Feature Selection (Hall, 1999), Markov Blanket Filtering (Xing et al., 2001), Simulated Annealing (Lin et al., 2008) and Randomized Hill-Climbing (Aha & Bankert, 1996). In this thesis, we have used Chi-Square method for feature selection. The advantages of using Chi-Square feature selection are that of being fast, scalable and independent of classifiers (Ladha & Deepa, 2011). The mathematical formalism for Chi-Square feature selection method is given as follows:

2.9.1 Chi-Square (χ^2) Feature Selection

Chi-Square (χ^2) test measures the lack of independence between a term and the category. Chi-Squared, the common statistical test that determines divergence from the distribution expected if one assumes that the feature occurrence is actually independent of the class value. As a statistical test, it is known to behave erratically for very small expected counts, which are common in text classification both because of having rarely occurring word features, and sometimes because of having few positive training examples for a concept (Ladha & Deepa, 2011). In statistics, the χ^2 test is applied to test the independence of two events. The two events A and B are defined to be independent if the probability of A and B, that is, the probability of their intersection, $P(AB) = P(A)P(B)$ or, equivalently, $P(A|B) = P(A)$ and $P(B|A) = P(B)$; that is, the probability of A given B is the probability of

A, alternatively $P(B|A) = P(B)$. In other words, occurrence or lack of it of one does not influence the occurrence or lack of it of the other. In general, any set of events is said to be independent if the probability of the intersection of any (finite) number of them is the product of their respective probabilities.

In feature selection, the two events are the occurrence of the term and occurrence of the class. Feature selection using χ^2 statistics is analogous to performing a hypothesis test on the distribution of the class as it relates to the values of the feature in question. In null hypothesis there is no correlation; each value is likely to have instances in any one class as in any other class. Under the null hypothesis, if p of the instances have a given value and q of the instances are in a specific class, then $(p \cdot q)/n$ instances have a given value and are in a specific class (n is the total number of instances in the dataset). This is because p/n instances have the value and q/n instances are in the class, and if the probabilities are independent (i.e. the null hypothesis) their joint probability is their product. Given the null hypothesis, the χ^2 statistics measures how far away the actual value is from the expected value:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}. \qquad 2.11$$

In Equation 2.11, r is the number of different values of the feature in question, c the number of classes in question (in this work, $c = 2$), $O_{i,j}$ the number of instances with value i in class j, and $E_{i,j}$ the expected number of instances with value i in class j, based on $(p \cdot q)/n$. The larger this Chi-Squared statistics, more unlikely the distribution of values and classes are independent; that is, they are related, and the features in question are relevant to the class.

2.10 The Ensemble

One issue associated with machine learning is generalization and generalization error gives a measure of how accurately a learning algorithm can predict outcome values for previously unseen data (Abu et al., 2012). A number of research studies (Shabtai et al., 2009; Zhang et al., 2014; Dai et al., 2016; Yang et al., 2017) have been conducted in recent past to enhance the prediction capabilities of learning models by combining the output generated from each base learner and this combining process is known as ensemble. Several studies (Brown et al., 2005; Tumer & Ghosh, 1996) have shown theoretically and empirically that ensembles bear the ability to generate more accurate prediction results than an individual learner on following some given criteria. Any member of a good ensemble must perform better than random guessing (accuracy requirement) that make errors on different parts of the input spaces (diversity requirement), and the trade-off between these two criteria is essential. It has been shown that diversity in base learners must decrease when they approach the highest level of accuracy (Kuncheva at al., 2002). From the optimization point of view, the objective of enhancing accuracy and the objective for increasing diversity are likely to be conflicting with each other. By taking into account the conflicting nature of accuracy and diversity, many researchers (Chandra & Yao, 2004; Jin & Sendhoff, 2008; Li et al., 2012; Minku & Yao, 2013) attempted to evolve a set of base learners using multi-objective optimization (MOO) approach. MOO is a more natural and straightforward machine learning method that can generate a set of solutions (learners) to be selected for constructing an ensemble. In Chapter 5 of this thesis, an MOO based multi-level ensemble approach is proposed. The common type of ensemble methods include Bagging, Boosting, and Stacking.

Bootstrap aggregating, often abbreviated as bagging, involves each model in the ensemble to vote with an equal weight. In order to promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set. For instance, the random forest algorithm combines random decision trees with bagging to achieve very high classification accuracy (Breiman, 1996).

Boosting involves incremental building of an ensemble by training each new model instance to emphasize the training instances that previous models have mis-classified (Li & Wang, 2013). In some cases, boosting has been shown to yield better accuracy than bagging, but it also tends to be more likely to over-fit the training data (Dietterich et al., 2000). By far, the most common implementation of boosting is AdaBoost (Freund & Schapire, 1997), although some newer algorithms have been reported to achieve better results (Hu et al., 2008).

Stacking (sometimes called stacked generalization) involves training of a learning algorithm to combine the predictions of several other learning algorithms (Kotsiantis et al., 2007). Primarily, all other algorithms are trained using the available data, then a combiner algorithm is trained to make a final prediction using all the predictions of the other algorithms as additional inputs. Stacking typically yields better performance than any single one of the trained models (Wolpert, 1992). It has been successfully used in all supervised learning tasks (regression, classification, distance learning and unsupervised learning (density estimation) (Ting & Witten, 1999; Ozay & Vural, 2012). It has also been used to estimate Bagging's error rate (Rokach, 2010; Wolpert & Macready, 1999).

2.11 Related Works

Broadly speaking, two complementary approaches have been proposed to analyze malicious web activities with a focus on detection of attacks on webpages. These are static analysis and dynamic analysis. The static analysis uses the source code of a webpage where from some static features are extracted such as URL structure and host details for detection of malicious code. Dynamic analysis invokes the evaluation of a code behavior exhibited when a webpage gets loaded and executed in real environment or using an emulated environment. A common procedure employed in both analyzing techniques is that the extracted features allow the identification of malicious code attack patterns in the webpage based on which the classification model is designed by using machine learning techniques. Ensemble methods of classification are applied to build a predictive model by combining multiple models to improve the predictive performance. In the following section, the most notable works related to the approaches presented in this thesis are discussed.

2.11.1 Static Analysis

Tokenising of URL in the form of query strings, path strings, hostname and artefacts related to HTML and JavaScript such as iframes, remote links, and function calls are found thriving in detection of malicious webpage attacks (Ma et al., 2009; Canali et al., 2011; Choi et al., 2011; Ma et al., 2011). An assumption following static facets of the webpage is that the statistical distribution of features from the malicious webpage rather than benign webpage. When these features are extracted, their values convert to train machine learning classifiers for classification.

The importance of using static analysis for detection of attacks is the fast extraction of features from the source code without loading it into the web browser. However, one limitation of using static analysis for malicious code detection is that it cannot detect those attacks which exploit execution of webpage. Precisely, the limitation of using features extracted from webpage source is to introduce a huge risk of code obfuscation and overlooking malicious JavaScript which exploits the browser vulnerabilities. The extracted features from static analysis by tokenizing URLs and JavaScript for detection may be used by an attacker for code obfuscation to evade these features from detection by making them look statically identical to benign ones.

Garera et al. (2007) have developed an approach for detection of phishing attacks. The present authors have developed certain heuristics to be used for training the logistic regression classifier to classify between phishing and benign URLs. In addition to heuristics pertinent to obfuscation, several general heuristics based on page rank, the word-based feature has been used by leveraging the Google index infrastructure. The results obtained from this approach show that an accuracy of 97.30 % can be achieved. In the present study, the authors have also conducted a large-scale measurement using Google Toolbar URLs for 12 days with the purpose of exploring the prevalence of phishing attacks on the Web. The results obtained from obtained measurements reveal that an average of 777 unique phishing URLs were active and an average of 8.24% of the users were observed to visit such webpages which are likely to become prey to the phishing attacks.

Seifert et al. (2008) presented an approach based on heuristics for generating signatures of attacks. These generated signatures of known attacks are used by Anti-Virus engines or Intrusion Detection Systems to perform a scan on the webpage to validate it as malicious

provided its heuristic pattern signatures match with the database. One limitation of this approach is that attackers can evade signatures using obfuscation so that the heuristics fail to detect novel attacks. Another limitation of this approach is that the time of signature database of heuristic-based system may be slower than the time used by attacker to exploit the vulnerability which leads to a zero-day exploit.

Ma et al. (2009) have developed an approach based on lightweight extraction for the analysis of lexical aspects and host information from URLs in order to build a classification model that learns from a dataset composed of mostly spam and phishing URLs. The main assumption of this approach is that URL tokens and host-based values of malicious URLs are different from the benign URLs. They used three classifiers such as Naïve Bayes, SVM and Logistic Regression on a large dataset of 30,000 URLs. Their experimental results show that Logistic Regression achieved a detection accuracy of 99% with 0.1% FPR. In Chapter 4 of this thesis, the hybrid feature approach by introducing 3 new URL features and reusing 8 URL features from their work. The proposed analysis, combines static analysis and lightweight dynamic analysis by using an emulated browser to visit and render the webpage, and execute client-side code upon webpage load. In addition to this, an ensemble technique for classification is used which is showcased in the Chapter 5.

Ma et al. (2011) in their follow-up work, improved the approach discussed in the previous Section by proposing an online learning algorithm to be applied on extracted features from URL and host-based via live feeds of URL. The assertion is that batch learning fails to cope up with a continuous distribution of features over the period of time. In this approach Perceptron, Logistic Regression with Stochastic Gradient Descent, Passive-Aggressive Algorithm, and Confidence Weighted Learning were used for classification.

Results obtained show that an accuracy of 99% can be achieved by confidence weight algorithm from a dataset obtained from a webmail provider.

Hou et al. (2010) put forth a machine learning-based obfuscation resilient approach obtained from detection of malicious code with a focus on detecting malicious DHTML. The feature has been extracted from webpage contents such as JavaScript functions and objects. This approach has used Naive Bayes, Decision Trees, Support Vector Machines, and Boosted Decision Trees for classification. Experimental results show that Boosted Decision Trees achieved an accuracy of 96% with 7.6 FRR on a dataset consisting of 176 malicious and 965 benign samples. In Chapter 4 of this thesis, some of the webpage source features have been reused from this study.

Canali et al. (2011) came up with a machine learning based approach called Prophiler which is a pre-filtering mechanism to optimize the resource consumption of a dynamic analysis and detection backend system. This approach deems a webpage into two categories which are likely malicious and likely benign. Prophiler uses four classifiers such as Random Tree, Random Forest, Naive Bayes, and Logistic Bayes Net for classification. Experimental results illustrate that a low false positive rate was obtained using a large URL testing set by using 78 features extracted from URL, HTML, and JavaScript. This approach uses features from different studies (Hou et al., 2010; Ma et al., 2011) and 48 new features have been introduced demonstrating the effectiveness of using Prophiler of likely malicious webpages. In Prophiler only one classifier is chosen for classification whereas in proposed approach an ensemble for classification as described in Chapter 5 of this Thesis has been used. The approach presented in Chapter 4 and Chapter 5 uses only 30 features obtained from a statistical evaluation for detection of malicious JavaScript code in comparison to 78 features

of Profiler. While the proposed approach judges unknown webpage either malicious or benign, Prophiler considers each unknown webpage as likely malicious or likely benign.

Xu et al. (2013) proposed a technique mainly based on static analysis by the name JStill for detection of obfuscated malicious JavaScript. This technique uses function invocation analysis in order to detect essential characteristics of a malicious code. JStill also uses a lightweight runtime analysis to detect and prevent the execution of obfuscated JavaScript code. The investigators claimed that JStill achieved high detection accuracy with low false positive rate based on a dataset consisting of malicious JavaScript samples obtained from real-world and benign data from top Alexa websites.

2.11.2 Dynamic Analysis

Code execution and behavior monitoring have proven effective in the path of analysis and detection of a malicious webpage (Moshchuk et al., 2007; Ikinci et al., 2008; Cova, 2010; Cova et al., 2010; Rieck et al., 2010; Kim et al., 2011; Qassrawi & Zhang, 2011; Kolbitsch et al., 2012). These approaches can be deployed at a proxy level (Moshchuk, 2007) to intercept the request and to observe the response of a user to visit the URL in a sandboxed environment for analyzing the execution dynamics to decide the safety of webpage to be opened at the user's Web browser.

Among dynamic analysis techniques, Honeyclients (Qassrawi & Zhang, 2011) are mainly employed systems that imitate a human visitor and dedicate a sandbox environment to visit a webpage. The execution dynamics is taken and analyzed on webpage rendering to seek a clue for malicious attack payloads. There are two types of Honeyclients: Low-interaction and high-interaction. Honeyclients which use emulated browsers and utilize less

operating system resources for webpage rendering are called as low-interaction Honeyclients. Whereas, Honeyclients that use real Web browsers and utilize more operating system resources for webpage rendering are termed as high-interaction Honeyclients. Low-interactions Honeyclients such as HoneyC (Seifert, 2007), Monkey-Spider (Ikinci et al., 2008), and PhonneyC (Nazario, 2009) are usually restricted in monitoring the traces of the activities through interaction with predefined attack signatures. With this limitation they are unable to find zero-day exploit due to static nature of signatures. High-interaction Honeyclients such as Capture-HPC (Hes et al., 2009), MITRE Honeyclient (MITRE, 2011), and Microsoft HoneyMonkey (Wang et al., 2006) check integrity changes in system states that require monitoring system files, processes, network connection, anomalies in CPU and memory consumption. The benefit of using Honeyclients is to provide an in-depth understanding of the malicious activities implanted inside the malicious webpage to perform the attack. Dynamic analysis has proven to be effective in the detection of malicious webpage but its downside is that it requires more resources as the webpage under analysis needs to be loaded and executed.

Wang et al. (2006) proposed a technique known as Strider HoneyMonkey which is based on a collection of many programs running the potentially vulnerable browsers in a sandbox environment with dissimilar patch level. The objective of the system is to patrol the Web and classify that website which explores the Web browser vulnerabilities and classification performed by correlating unusual changes to the system.

The approach presented in Chapter 4 of this thesis uses an emulated browser where a script is executed and certain information, script execution time and function calls are extracted and correlated with the malicious scripts. From dynamic aspect, hybrid feature

extraction can be categorized into low-interaction Honeyclient where a minimum sandbox environment is utilized for categorizing webpage into malicious or benign.

Moshchuk et al. (2007) implemented an approach termed as SpyProxy, which performs the analysis on execution of Web content to safeguard end-user from web-based malicious code. SpyProxy intercepts all the requests and responses from the client browser to serve as webpage, and retrieves the root webpage to analyze it statically. While performing static analysis, a root webpage found to be unsafe is forwarded to the virtual machine to perform further analysis. Once a root webpage is confirmed to be malicious, it is blocked. If the webpage is found safe by virtual machine, it is forwarded to the Web browser of the user. The evaluation of SpyProxy reports that it can detect any threat to which it was exposed. SpyProxy introduced a computation overhead of 600 milliseconds in analyzing and deciding whether a webpage is malicious or benign.

Ratanaworabhan et al. (2009) presented an approach known by Noozle for detecting heap spray attacks. Noozle captures calls made to browser's memory manager and attempts to detect any possible heap spraying attack by considering local objects as they were code and interprets them. Noozle uses a lightweight emulator to conduct a scan on local objects for the valid x86 code. The report claims that the Noozle was able to detect all types of heap spray attacks with low false positive rate. The limitation of this study is that it deals with heap spray attacks only although there are other types of attacks for which users require protection. The issue of heap spray attacks has been in Chapter 4 of this thesis by introducing execution time features intended for measuring the execution time of each script. Scripts that perform heap spray attacks require more execution time than benign scripts in order to all allocate heap memory.

Cova et al. (2010) suggested an approach named as Wepawet for analyzing malicious JavaScript for detection of any possible drive-by-download attack. Wepawet is based on emulation, anomaly detection, and machine learning. The Wepawet framework is an emulation-based dynamic analysis for detection of malicious contents especially JavaScript. They report to have achieved a low false positive rate in detection of drive-by-download attacks and that Wepawet being available online as a public service. But on visiting the Wepawet website, we found the service discontinued while filing this report.

Dewald et al. (2010) developed a client-side approach termed as ADSandbox. ADSandbox detects malicious webpages based on sandboxing and heuristics. JavaScript embedded in a webpage is extracted and analyzed in sandbox environment with logs generated for each event. Using Internet Explorer Web browser, this approach is implemented through Browser Helper Object (BHO). At any time when a malicious code is detected, BHO blocks it and notifies the user whether to block it completely or want to load it anyway. This approach attains low false positive rate with very high performance overheads.

Rieck et al. (2010) introduced another client-side approach namely Cujo, a web proxy for detection of driver-by-download chiefly focusing on the detection of malicious JavaScript using SVM classifier for dynamic features. The results show that Cujo achieved 95% accuracy with some low false rate. Cujo introduced a performance overhead of 0.5 seconds per page in deciding whether a webpage is malicious or benign.

Curtsinger et al. (2011) inducted a technique called Zoozle based on a combination of static and dynamic analysis for malicious JavaScript detection. In Zoozle features are extracted from hierarchical aspects of Abstract Syntax Table and classification done by the

Bayesian classifier. Their report indicates that Zoozle is effective in detecting heap spray attacks and low fast positive rate of 0.003% achieved by detecting malicious JavaScript using static analysis.

Seifert et al. (2012) offered a technique by the name Rozzle which is a JavaScript multi-execution sandbox environment to trace multi-execution paths of a code that reveals environment specific malicious code. Rozzle claims to increase the detection rate for offline runtime detection by seven times, for online detection by three times using a large scale data and suffers no performance overheads.

Thomas et al. (2011) prepared a system called Monarach based on static analysis partially from the earlier works (Ma et al., 2009) and introduced dynamic analysis for crawling URLs. The crawled URLs are submitted to Monarach which in turn analyzes each URL and determines whether the URL directs to a phish or a spam webpage. Experimental results show that this approach has high false positive rate and introduces computation overheads.

Xu et al. (2013) proposed a method for detection of malicious webpage based on URL features, host-based features, and features extracted from the network -track based on execution. Their method is similar to have being presented in Chapter 4 of this thesis with the aim of combining static and dynamic aspects for detection of malicious webpage. The proposed approach used browser emulation whereas they used network track with honeyclient.

CHAPTER 3

PROPOSED APPROACH

3.1 Introduction

In this chapter, a client-side approach is develop, demonstrate and implemented for the detection of malicious JavaScript attacks, prediction of unknown JavaScript programs, and mitigation of XSS attacks prior to execution by the client's browser. An overview of the present approach is given in Section 3.2. Section 3.3 provides the details about the evaluation metrics employed in this study. In Section 3.4 the software tools utilized in the implementation of the proposed approach are presented.

3.2 Overview of Proposed Approach

In view of the crucial need for the detection of malicious JavaScript code in webpages with high accuracy and low performance overheads before being interpreted by the web browser, the approach has been developed for the detection of such attacks on the client-side. In this approach, all the webpages sent from server to client are analysed for the detection of any possible malicious JavaScript code attacks. This approach is developed to protect the client-side from being attacked or delivered with a malicious code. The evolved approach can be installed as a plugin on the client's web browser. The envisaged approach employs a hybrid of static and dynamic analyses coupled with machine learning algorithms for detection of malicious JavaScript attacks and during the detection process, any malevolent JavaScript found in the source code of the webpage gets obstructed. This approach is projected to have better detection rate as compared to manually crafted rules which have low detection accuracy with the high false-positive rate. The benefit of using the client-side solution for detection of

malicious JavaScript is its effectiveness and proactiveness in mitigating attacks owing to proximity with web browsers interpreting the JavaScript. The client-side solution also protects the cookie and session from getting attacked besides reducing the server load. Although several server-side countermeasures do exist for XSS attacks yet such techniques have not been adopted globally because of their deployment overheads and the poor understanding of XSS problems. A schematic representation of the developed approach is depicted in Figure 3.1. The three vital components such as data collection, feature extraction and feature selection used within the framework of the envisaged approach are discussed in greater detail in Chapter 4. The classification process using ensemble technique is discussed and pressed into operation in Chapter 5.

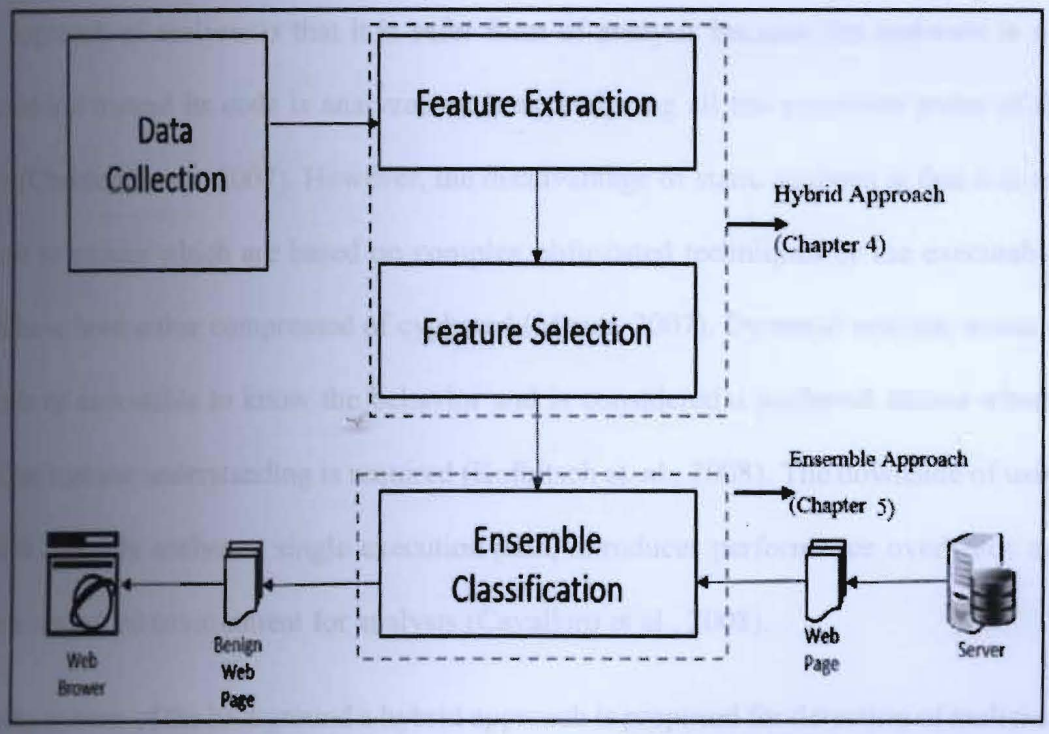


Figure 3.1: Schematic depiction of the Present Approach.

3.3 Motivation for Hybrid and Ensemble Approach

Machine learning approaches for malware detection such as (Schultz et al., 2001; Kolter et al., 2004; Moskovitch et al., 2008; Santos et al., 2009) are generally dependent on the dataset which includes characteristic features from both malicious and benign samples which are needed for the classification. These features are pivotal in deciding which sample is malicious or benign. To detect obfuscated code and previously unknown malware, two types of features can be used which are extracted statically and dynamically. The static features are extracted by performing static analysis in which features are extracted from the source code or executable without actually executing. While as dynamic features are extracted by performing dynamic analysis in which features are extracted when the code or an executable is executed in a controlled environment also called as sandboxing. The advantages of using static approach of analysis is that it is safer form of analysis because the malware is not executed but instead its code is analyzed such as analyzing all the execution paths of the binary (Christodorescu, 2007). However, the disadvantage of static analysis is that it is not resilient to attacks which are based on complex obfuscated techniques or the executables which have been either compressed of cyphered (Moser, 2007). Dynamic analysis executes the code or executable to know the behavior and is considered a preferred choice when a complete runtime understanding is required (Kolbitsch et. al., 2008). The downside of using is that it can only analyze a single execution path, introduces performance overheads and needs a controlled environment for analysis (Cavallaro et al., 2008).

Keeping in view of the background a hybrid approach is proposed for detection of malicious JavaScript code that employs a feature set which is composed of static and dynamic features. The static features are based on structural features which include URL features and webpage source features. On the other hand, the dynamic features are based on execution time.

Therefore, combining the strength of both the static and dynamic approaches as a hybrid approach is proposed. The process is depicted in Figure 3.2 and is further detailed in Chapter 4.

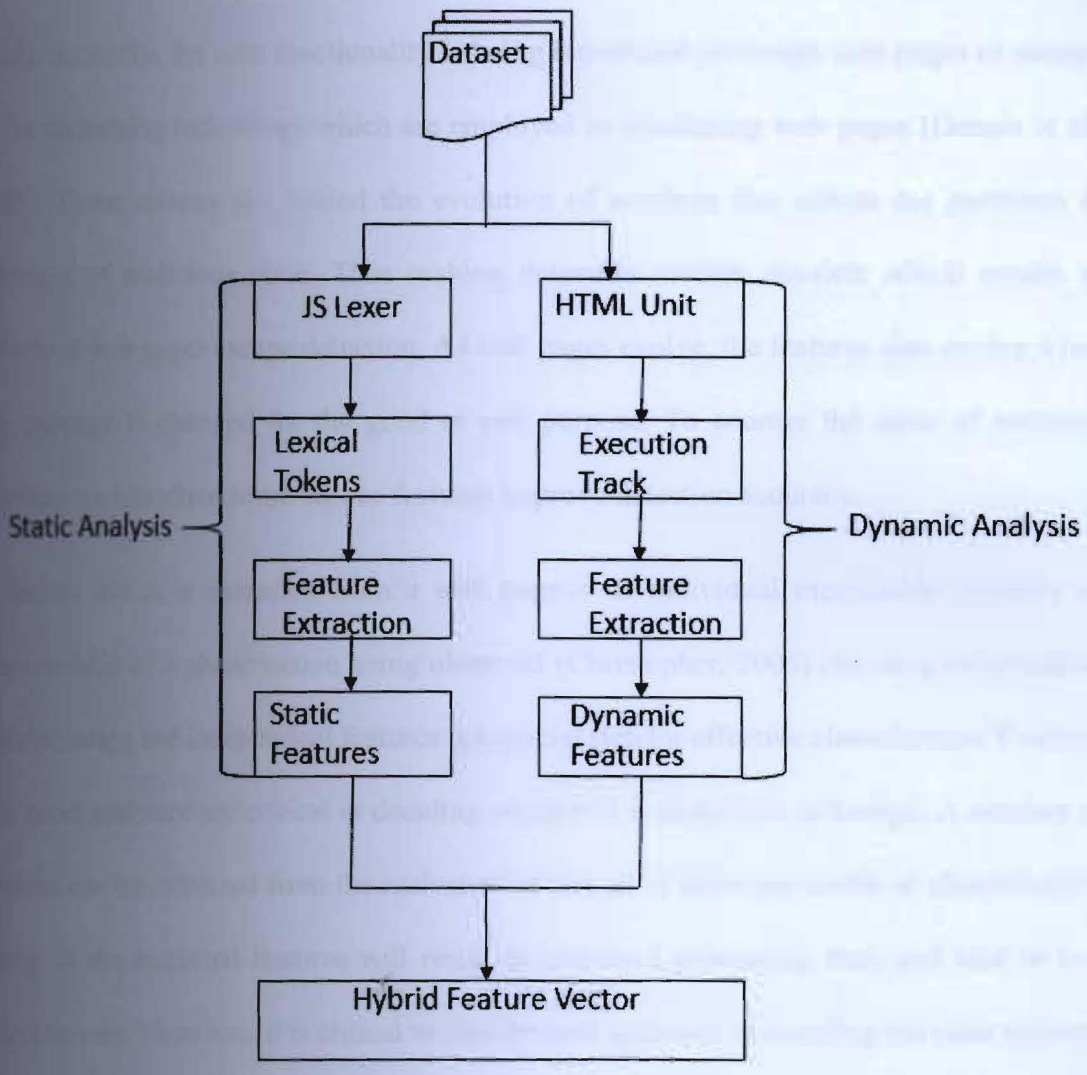


Figure 3.2: The process of Hybrid Feature Extraction

To develop a classification model for the detection of malicious JavaScript, distinct artefacts of malicious and benign web pages are analyzed. These artefacts are constantly evolving over time and presents a challenge for detection in current anti-malware software (Dennis et

al., 2003; Rajab et al., 2011; Kapravelos et al., 2013). The evolution takes place because of two reasons: firstly, the cybercriminals regularly keeps on changing their methods to craft attack payloads in malicious web pages with the purpose of making the attack more complex and avoid detection by existing countermeasures (Rajab et al., 2011; Kapravelos et al., 2013). Secondly, the new functionality is being introduced in benign web pages or change in the underlying technology which are employed in developing web pages (Dennis et al., 2003). These reasons are behind the evolution of artefacts that affects the precision in detection of malicious code. Thus making detection models obsolete which results in malicious web pages escape detection. As web pages evolve, the features also evolve when the webpage is changed for the good or evil purpose. To counter the issue of evolving artefacts we introduce some unique features improve detection accuracy.

A feature which is extracted from a web page is an individual measurable property or characteristic of a phenomenon being observed (Christopher, 2006) choosing informative, discriminating and independent features is a crucial step for effective classification. Features of a given malware are critical in deciding whether it is malicious or benign. A number of features can be extracted from the malware but not all of them are useful in classification. Using all the extracted features will result in increased processing time and lead to low detection rate. Therefore, it is critical to find the best approach in deciding the most relevant features for a given malware and feature selection is another main component of hybrid approach to select the relevant features. The feature space in this study is created from samples by the tokenizing process. The tokens generated are refined and grouped in a similar type of feature set which the attributes of JavaScript are such as the number of JavaScript keywords, number of DOM functions, and number of event-driven methods. The obtained features are then passed to Countvectroizer matrix which reveals the number of occurrences

of a particular feature in a sample. The large feature space will have irrelevant and less informative features. In order to make classification accurate, the feature space will be reduced to only such features which will contribute towards higher accuracy by performing feature selection. To reduce the dimensionality and enable classification models to work accurately and effectively, feature selection algorithm Chi-Square and designed an algorithm to weight each feature were employed.

Detection of malicious code using machine learning methods has been explored extensively to enable fast detection of newly released malware. The performance of these classifiers depends on the induction algorithms being used. In order to benefit from multiple different classifiers, and exploit their strengths ensemble of these classifiers is introduced for classification. The strength of ensemble methods lies in their ability to correct errors made by some of their members (Polikar, 2012). Therefore, ensemble members have to be diverse in terms of the errors they make, so that their combination can reduce the total prediction error (Brown et al., 2005). In order to make the ensemble more effective, there should be some sort of diversity between the classifiers (Kuncheva, 2005). Ensembles with greater diversity among their members tend to result in higher predictive accuracy (Kuncheva et al, 2003). Diversity among the members of an ensemble can be achieved in different ways, resulting in homogeneous or heterogeneous ensembles. Diversity between classifiers can be calculated by the level of disagreement about the probability distribution for the test datasets. In Multi-Inducer strategy, diversity is obtained by using different types of inducers. Each inducer contains an explicit or implicit bias that leads it to prefer certain generalizations over others. Even more ambitiously, there is hope that this combination of paradigms might produce synergistic effects, leading to levels of accuracy that neither atomic approach by itself would be able to achieve. Keeping in view the strengths of an ensemble of classifiers

for classification, an efficient heterogeneous multi-level ensemble using enhanced bagging and multi-objective optimized weighting is introduced.

3.4 Performance Evaluation

Performance evaluation acts as a multi-purpose tool to measure the actual values within the system against expected values. The purpose of the present evaluation is to study and measure the performance of the developed approach in detecting the malicious code attacks. The following metrics are used for the evaluation:

1. Accuracy = $\frac{True\ positive+True\ negative}{True\ positive+True\ negative+False\ positive+False\ Negative}$ 3.1

2. Sensitivity (Recall)= $\frac{Number\ of\ true\ positivies}{Number\ of\ true\ positivies+Number\ of\ false\ negatives}$ 3.2

3. Specificity = $\frac{Number\ of\ true\ negativies}{Number\ of\ true\ negativies\ +Number\ of\ false\ positivies}$ 3.3

4. Precision= $\frac{Number\ of\ true\ positivies}{Number\ of\ true\ positivies\ +Number\ of\ false\ positivies}$ 3.4

5. F-Measure =2 * $\frac{Precision*Recall}{Precison\ +Recall}$ 3.5

A False-positive scenario occurs when the attack detection process mistakenly treats a normal code as a malicious one. The other situation in which the implemented approach is unable to detect the malicious code despite its illegal behavior leads to a False-negative. True-negative gives the evidence of a number of negative samples correctly identified, False-negative pretends a number of malicious samples identified as negative, False-positive poses a number of negative samples identified as malicious, and True-positive designates a number of malicious samples correctly identified as malicious (Nunan et al., 2012). The performance of the developed detection approach is the rate at which the malicious scripts get processed. The performance is calculated by the latency time taken in presence and absence of the

developed approach to display a webpage or by calculating the system resource consumption in both scenarios.

3.5 Software Tools Used

In this section, the details of various software tools employed in the implementation of the developed approach for the detection of the malevolent JavaScript code attacks.

3.5.1 Weka

Weka (Waikato Environment for Knowledge Analysis) is a Java-based open source machine learning workbench developed by the University of Waikato, New Zealand in 1993 [*https://www.cs.waikato.ac.nz/ml/weka/*]. Weka is a user-friendly tool with a graphical interface including all the standard methods of data mining problems such as classification, clustering, regression, associations, and attribute selection (Bouckaert et al., 2010). In addition, Weka includes a vast variety of packages for data prepossessing and visualization (Witten et al., 2016). Explorer is the foremost user interface of Weka; however, the same functionality can be accessed through Knowledge Flow and Command Line interfaces. Experimenter is yet another interface that permits an organized evaluation of the predictive performance of Weka machine learning algorithms on a group of datasets.

Weka works on the basic premise of availability of data in a flat file where each data point is represented by a specific number of numeric, nominal or other types of attributes. Although it allows access to SQL databases and Deep Learning [*http://weka.sourceforge.net/packageMetaData/*], nevertheless it does not work for manifold data mining without the support of other software. Weka is also helpful in sequence

modeling. Considering the foregoing characteristics of Weka we got motivated to use the tool in this study.

3.5.2 HTMLUnit Browser

HTMLUnit is a Java-based browser without a graphical user interface but implemented through a command-line interface which offers automated control of a webpage or using network communication in a situation analogous to common Web browsers [*http://arch.net/2009/10/what-is-headless-browser.html*]. The specific task of this browser is webpage testing as well as fashioning items such as webpage layout, color choice, and font selection besides execution of JavaScript and AJAX generally not accessible in other testing techniques [*http://phantomjs.org/quick-start.html*]. HTMLUnit browser permits higher scale management of other Java code websites like filling and submitting of forms and clicking of hyperlinks. It has the capability of emulating components of browser behavior and the capacity of dealing with HTTP headers such as security, authentication, automatic page redirection, etc [*http://htmlunit.sourceforge.net/*]. HTMLUnit embeds Rhino, an open source implementation of JavaScript can imitate and operate like Internet Explorer and Firefox browser [*https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino*]. Having fairly good JavaScript Support, it can work easily and flexibly with complex libraries and supports both HTTP and HTTPs protocols. The motivation of using this emulated browser in the present research work is to simulate the browsers in an arbitrary environment for execution of web pages.

3.5.3 ParseHub

ParseHub (2017) is a web crawling tool that supports complicated data extraction from static as well as dynamic data websites, using AJAX, JavaScript, redirects, and cookies. It is equipped with machine learning technology that can read and analyze documents on the Web to yield relevant data (Milev, 2017). ParseHub can extract the data and turn it into an Application Programming Interface (API) which is a bunch of well-defined means of communication among different software components. The API can be used to recall tasks in progress, cancel active tasks, execute illustrations, revisit task conditions, and alter executions.

This web-based tool can be exploited by the individuals in promptly submitting e-forms, website logins, and clicking interactive operations. It saves time, specifically while performing works such as looping during dropdowns and managing web pages with countless scrolling. The tool operates on JavaScript, AJAX, cookies, sessions, and redirects, rendering it a dominant choice for ordinary users concerned in data mining from websites but lacking latest soft skills. The motivation for using this tool in this investigation is to scrap the web pages from web to create a corpus of benign web pages.

CHAPTER 4

MALICIOUS JAVASCRIPT ATTACK DETECTION USING HYBRID FEATURES

4.1 Overview

Studies have shown that new web-based attacks are ever increasing, getting more and more complex as the attackers use unified techniques to set aside corrective measures (Rajab et al., 2011; Stringhini et al., 2013). A number of approaches have been proposed over the years for the detection of malicious JavaScript attacks based on features extracted by static or dynamic analysis. However, these approaches are limited in their utility as they perform only partial inspection of malicious webpages. To overcome this limitation, an effective approach is proposed and implemented in terms of high accuracy and low computational cost for detecting malicious JavaScript payload attacks in webpages. A distinguished character of this approach is an in-depth inspection of webpages for detection of malicious code using hybrid features obtained by performing static and dynamic analysis on program structure and URL metadata. On one side, the proposed approach have reused and improved some of the features carried out by earlier researchers (Cova et al., 2010; Canali et al., 2011; Choi et al., 2011; Ma et al., 2011; Eshete et al., 2012; Mohammad et al., 2014; Boire et al., 2015; Xiong et al., 2015; Wang et al., 2015; Patil et al., 2017) while on the other, some unique features are introduced in order to detect malicious JavaScript attacks in webpages.

The major characteristics of this hybrid feature approach are highlighted as under:

1. A unified approach has been developed that combines static and dynamic analysis for producing hybrid features to detect malicious webpages.

2. Eight unique features have been inducted and several existing features enhanced to boost their discriminative power in categorizing malicious webpages from benign ones.
3. The effectiveness of the approach has been envisaged via implementation and evaluation of the extracted features for characterizing webpages for any possible malicious code.

The rest of this Chapter is organized as follows: Section 4.2 describes the methodology employed in the hybrid feature approach, Section 4.3 provides the details about the comprehensive characterization of webpages, Section 4.4 offers the modus operandi of the approach, Section 4.5 evaluates the performance overhead of the approach, Section 4.6 discusses the efficiency of the approach and, Section 4.7 summarises the Chapter.

4.2 Hybrid Feature Approach and Methodology

With the primary focus on analyzing and detection of malicious webpages containing attack payloads, this approach combines static and dynamic methods of analysis for extracting hybrid features. The extracted hybrid feature set consists of some unique and several existing features for characterizing a webpage for any malicious code. The extracted features are studied using the proven statistical approach as feature selection for accurate classification of malicious and benign webpages. This approach has four vital components: Data Collection, Feature Extraction, Feature Selection, and Classification, which are diagrammatically shown in Figure 4.1. The following Section provides an elaborate discussion on these components.

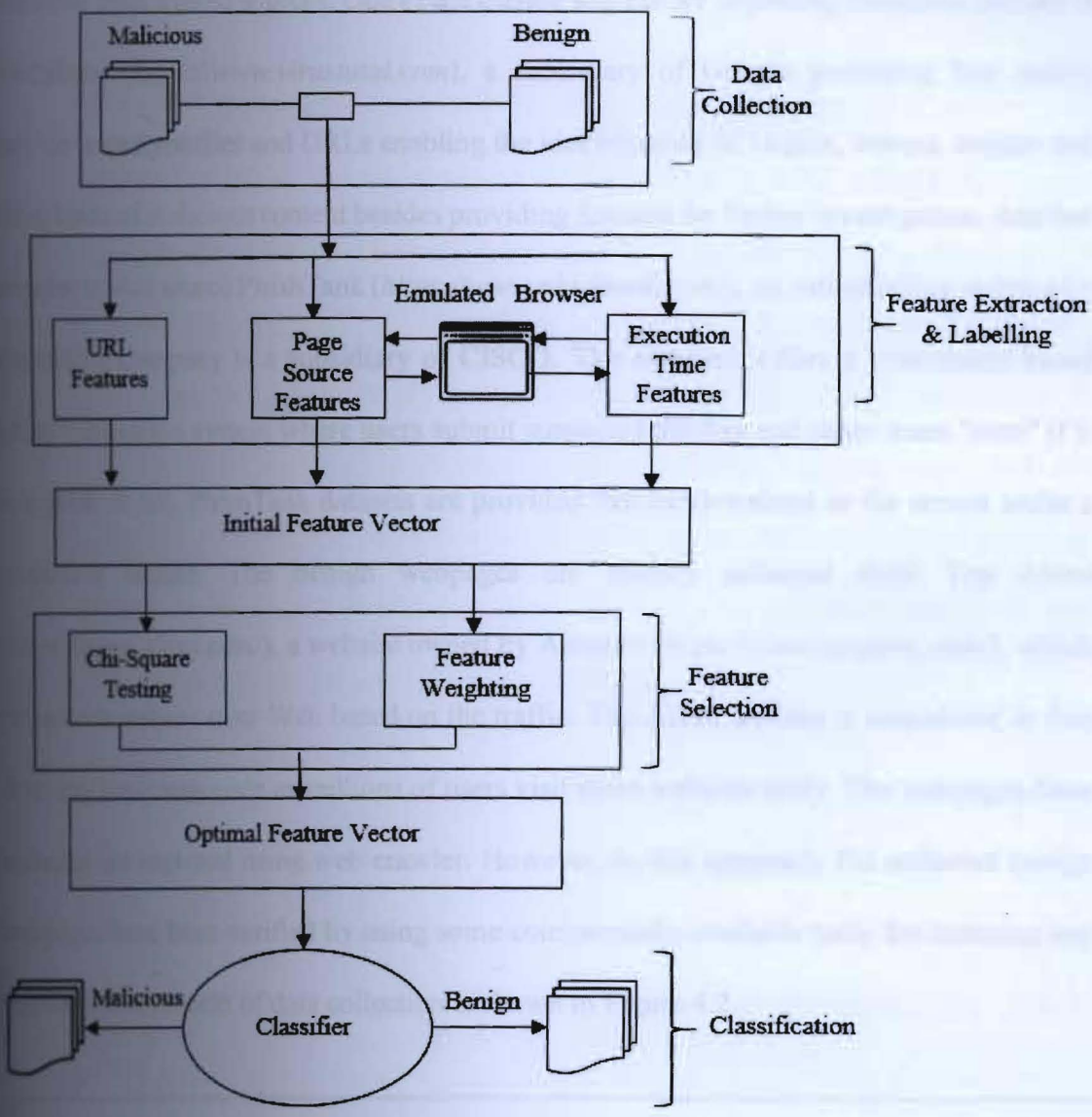


Figure 4.1: Block diagram of Hybrid Feature Approach

4.2.1 Data Collection

In order to perform the classification of webpages for malicious code detection using machine learning approach, data collection is the first step in this process. The quality of data is important which must possess a blend of malicious and benign instances so that the distribution of samples emulates distribution on the Web. An impurity in the dataset will impact the classification results. In this step, the malicious and benign data has to be

collected from trusted sources. One of the trusted sources for collecting malicious dataset is VirusTotal (*https://www.virustotal.com*), a subsidiary of Google providing free online service to analyze files and URLs enabling the identification of viruses, worms, trojans and other kinds of malicious content besides providing datasets for further investigation. Another popular trusted source PhishTank (*https://www.phishtank.com/*), an anti-phishing website by OpenDNS Company is a subsidiary of CISCO. The company offers a community-based phish verification system where users submit suspected phishes and other users "vote" if it is a phish or not. PhishTank datasets are provided free for download or for access under a restrictive license. The benign webpages are usually collected from Top Alexa (*https://www.alexa.com/*), a website owned by Amazon (*https://www.amazon.com/*), which ranks each website over Web based on the traffic. Top Alexa website is considered as free from any malicious code as millions of users visit these websites daily. The webpages form websites are captured using web crawler. However, in this approach, the collected benign webpages have been verified by using some commercially available tools for scanning any malware. The process of data collection is shown in Figure 4.2.

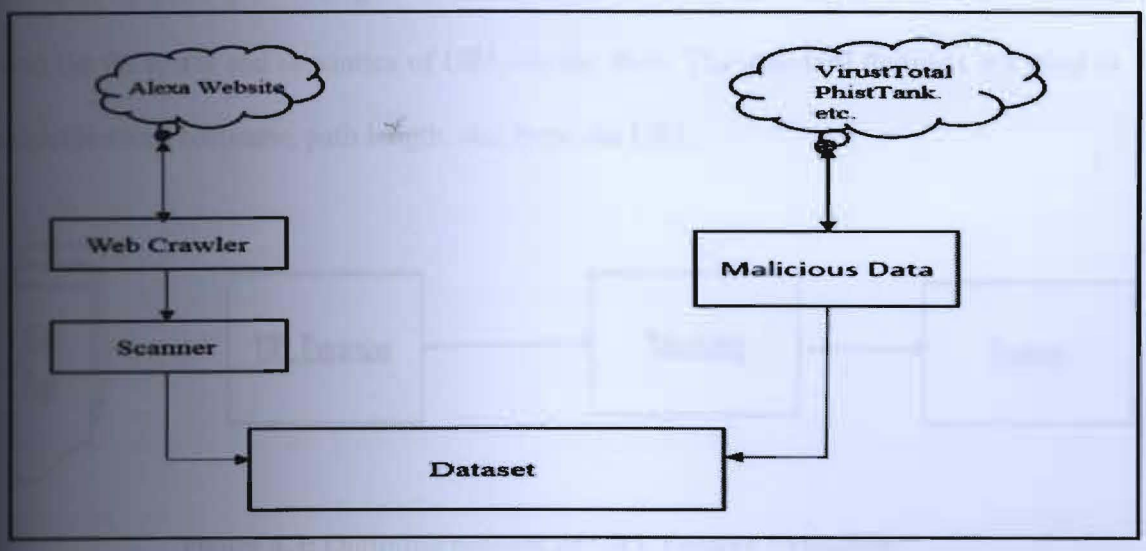


Figure 4.2: Schematic representation of Data Collection Process

4.2.2 Feature Extraction and Labelling

Next step is to extract features from the obtained dataset for detection of malicious JavaScript code. The static features are extracted by performing URL tokenizing whereas dynamic features are extracted by an emulated browser to get the execution details of the features. For this purpose, an HTMLUnit which is an emulated browser has been used for rendering and executing JavaScript. In this approach, HTMLUnit was customized to consider each script as a new instance to ensure unique session for each script. Both the features extracted by static analysis and dynamic analysis are combined and are called as hybrid features. To be precise, hybrid feature approach consists of features extracted from URL lexical analysis, structural analysis, and runtime analysis.

Tokenizing is performed on the URL to generate a set of tokens from which features are extracted. Figure 4.3 shows the process of feature extraction using URL tokenizing. The parser reads and analyses the input stream, and then splits into component pieces such as keywords, string numbers, and punctuations without any further division. The URL is tokenized by parsing according to Remote Function Call (RFC-1738) Standard which describes the syntax and semantics of URL on the Web. The standard delimits are used to extract protocol, hostname, path length, etc. from the URL.

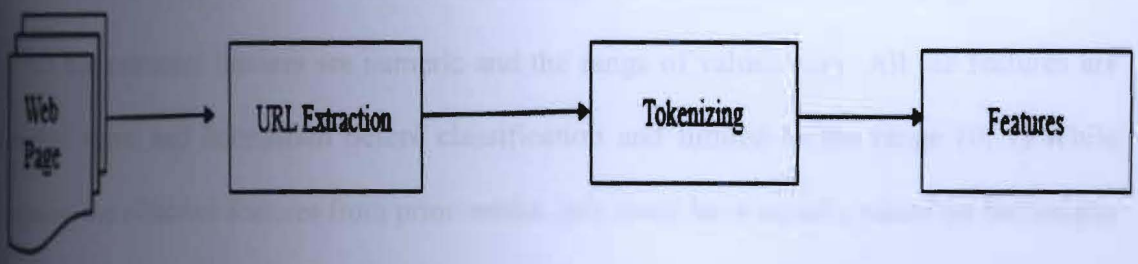


Figure 4.3: Outlining process of URL Feature Extraction

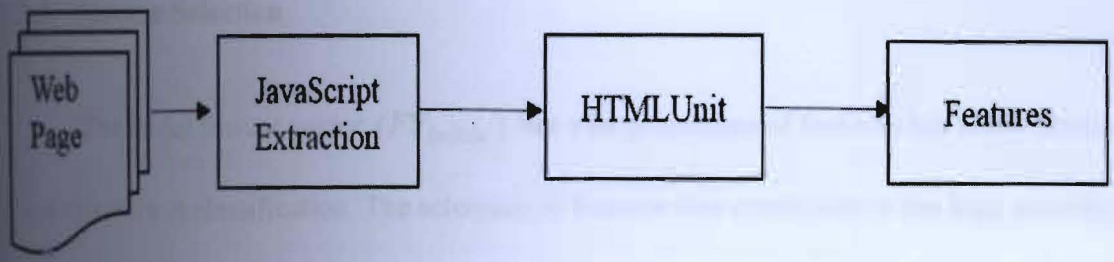


Figure 4.4: Illustrating process of JavaScript Feature Extraction

Similarly, JavaScript from HTML file has been extracted for classification and the process of feature extraction is depicted in Figure 4.4. To obtain the JavaScript from the main body of HTML and external JavaScript by finding the start <script> tag. JavaScript code segment is considered until </script> tag comes across. Subsequently, external JavaScripts are extracted when tag "src" is found. There are certain extracted features from JavaScript which can be tested only using the emulated browser. Such features which need emulated browser session are required in detection of obfuscation, redirection and clocking. The extracted hybrid features are represented in a feature vector as:

$$FV_{Initial}=\{f_1^{(i)}, f_2^{(i)} \ldots\ldots f_{n-1}^{(i)}, class^{(i)}\}, \quad 4.1$$

where $FV_{Initial}$ is an initial feature vector, $f_k^{(i)}$ the feature values $(k=0,1,\ldots.n)$, n the number of features, and $class^{(i)} \in (malicious, benign)$ the class label of a webpage. The value of all the extracted features are numeric and the range of values vary. All the features are treated equal and normalised before classification and limited to the range (0, 1).While reusing the effective features from prior works, this study have equally relied on the unique features introduced in this study to improve the detection accuracy of malicious webpages.

4.2.3 Feature Selection

The initial feature vector ($FV_{Initial}$) has a large number of features but every feature is not effective in classification. The selection of features that contribute to the high accuracy is needed from the initial feature vector while ignoring rest of the features. In order to calculate the effectiveness of each feature, Chi-Square testing to experimentally evaluate the performance of individual features has been used. As Chi-Square works on features independently, feature weighting at classifier level for calculating the effectiveness of each feature to obtain optimal feature vector was also used. The advantage of using Chi-Square test on features is its robustness on distribution of data with fast and easy computation. The objectives of performing selection both at feature level as well as at classifier level are to maximise the classification performance, minimize the number of features and optimize the training time by removing noisy features. The weight w_i for each feature F_i is calculated as:

$w_i = \frac{e_i}{\sum e_i}$, where an effect e_i is calculated for each feature through: $e_i = TP_i - FP_i$ while TP_i and FP_i are calculated by using Naïve Bayes classifier. To get a solution in the feature selection sense, the threshold value will indicate whether a feature F_i is selected or neglected according to its P-value and weight w_i. Figure 4.5 shows the process of feature selection.

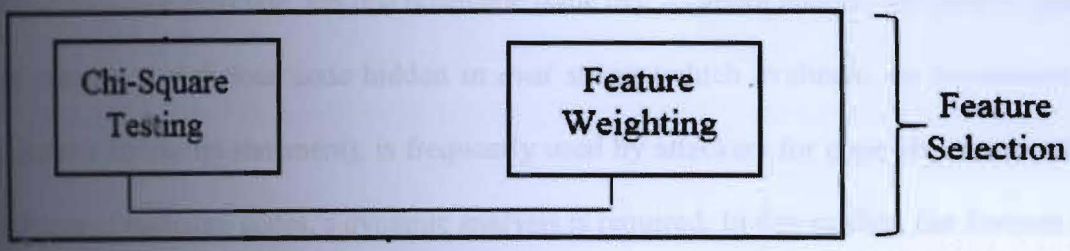


Figure 4.5: Explaining process of Feature Selection

4.2.4 Classification

In machine learning, supervised learning is the task of inferring a function from a labeled training data consisting of a set of training examples (Anzai, 2012). Each example is a pair of an input vector and the desired output value (also called a class label). A supervised learning algorithm analyzes the training examples and produces an inferred function, which is called a classier for discrete output or a regression function for continuous output. The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to unseen instances in a way that avoids overfitting (Mohri et al., 2012). In this studies, the five trained supervised learning algorithms which were use are: Naïve Bayes, J48 Decision Trees, SVM, Random Forests, and K-NN in order to evaluate the efficiency of the existing and unique features.

4.3 Comprehensive Characterisation of Webpage Features

For any given webpage, the process of analyzing and classifying of a malicious JavaScript attack is performed by extracting features from webpages under analysis. In the present research work, the features are extracted from static and dynamic aspects of webpages. Static analysis, considered as a powerful way of exploiting the nature of the code

in malicious code detection, has one downside issue that it cannot analyze all sorts of codes. For example, a malicious code hidden in *eval* string (which evaluates an expression or executes a JavaScript statement), is frequently used by attackers for code obfuscation. For such type of malicious codes, a dynamic analysis is required. In this studies, the features are categorized into execution time features and structural features. The structural features are further categorized into URL meta-features and webpage source features. For training the learning algorithms, a known mix of benign and malicious JavaScripts and URLs are used. The corpus of malicious JavaScript used in training consists of scripts capable of performing shellcode attacks, drive-by-download attacks, event generators and injection attacks. In the next section, the features that were extracted to build this approach in order to classify malicious webpages from benign ones are detailed. Figure 4.6 depicts the technique exploited for hybrid feature extraction. For the sake of URL feature extraction 500 malicious URLs were randomly selected, and for the utilization of webpage source, arbitrarily 4000 malicious scripts in JavaScript where choosen.

4.3.1 Execution Time Features

Three features which have been extracted by performing dynamic analysis of webpages include: execution time, call to JavaScript function, and number of function calls made by JavaScript. These extracted features are based on the assumption that malicious JavaScripts require more resources than benign JavaScripts and that malicious JavaScripts call a limited number of JavaScripts. Extracted execution time features are summarised in Table 4.1.

A. Maximum JavaScript Function Execution Time

The maximum execution time of a JavaScript function during runtime is computed as:

$$MT=M(t_i), \qquad 4.2$$

where M is the maximum execution time of JavaScript function and t_i the execution time of the i^{th} JavaScript function. During investigation it was found about 475 such malicious scripts which have memory leaks to exhaust the system memory. The maximum time of such scripts was found to be 132 seconds.

Table 4.1: Summary of Execution Time Features

S.No.	Feature Name	Type
1	Maximum JavaScript Function Execution	Unique
2	Average JavaScript Function Execution Time	Unique
3	Number of Function Calls	Reused

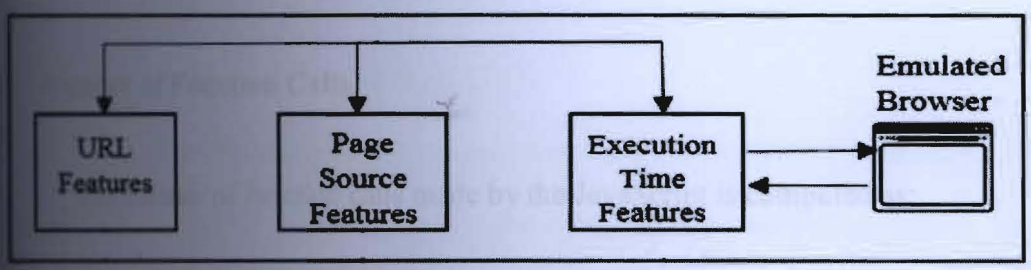


Figure 4.6: Displaying process of Hybrid Feature Extraction

Since HTMLUnit, is used which is an emulated browser for execution of scripts, the browser became non-responsive after 132 seconds and generated warning "Unresponsive Script"

while no time limit was set for execution of JavaScript. Different web browsers have experimented different techniques to limit the JavaScript maximum execution time. Firefox uses a limit of 10 seconds and Safari requires 5 seconds limit whereas Chrome does not put any time limit but detects when a browser crashes or becomes unresponsive. These time constraints are necessary to minimize the processing time at client-side.

B. Average JavaScript Function Execution Time

The average JavaScript execution computes the average time taken by JavaScript functions at runtime. The time taken is calculated as:

$$AT = \frac{1}{n} \sum_{k=1}^{n} f_i , \qquad 4.3$$

where AT is the average time, n the number of JavaScript functions, and f_i the execution time of the i^{th} JavaScript function. Generally malicious code requires more execution time than benign code and the average of all the executed 4000 malicious JavaScripts was found to be 3.4 seconds.

C. Number of Function Calls

The number of function calls made by the JavaScript is computed as:

$$FC = \sum_{i=1}^{n} f_i , \qquad 4.4$$

where FC is the number of function calls and f_i the number of JavaScript calls made for the i^{th} function.

4.3.2 Structural Features

The structure of a webpage consists of HTML code, JavaScript and other embedded objects. HTML code containing the description of the webpage, has only a small amount of information in the form of text, image, and hyperlink visible to the user while rest is kept hidden in the form of source code. The information about the structure of a webpage is obtained from HTML tags in the form of HTML code. These HTML tags when counted provide information that how many times a particular tag was used in the structure. The count number of tags indicates that if the webpage has forms, embedded objects and JavaScripts which require attention. Based on this logic, it is assumed that such information may be useful in detecting malicious JavaScript attack when converted into features for predicting. The structural unique features, which were extracted, are categorized into URL features and webpage source features.

4.3.2.1 URL Features

A number of studies have proven that lexical elements of URL strings are highly effective in detecting malicious and benign URLs particularly in the rapid detection of phishing and spam URLs (Darling et al., 2015; Mamun et al., 2016). In this study, 3 unique features introduced and reused 6 URL-based features from previous studies summarized in Table 4.2. Features from URL string are extracted by performing lexical scanning considered to be effective in detecting malicious URLs (Mamun et al., 2016). Algorithm 4.1 extracts features from URL by tokenizing.

Table 4.2: Summary of URL Features

S. No	Feature Name	Type
1	Length of Query	Unique
2	Missing Top Level Domain	Unique
3	Suffixing/ Prefixing of Dash	Unique
4	Length of URL	Reused
5	Number of Forward Slashes	Reused
6	Number of Dots	Reused
7	Special Characters	Reused
8	Number of Client-Server Words	Reused
9	Number of Equal Signs	Reused

Algorithm 4.1: URL Tokenizing

```
Input: URL
Output: Extracted URL Features
1: lower case URL
2: split URL string of letters, until
   Punctuation mark, number, non-letter
   characters
3: remove http://
4: Tokenize, generate bag of words
5: Generate URL Features
```

A. Length of Query

The query string is composed of a series of field-value pairs. Within each pair, the field name and value are separated by an equals sign, '='. The series of pairs are separated

by the ampersand, '&' for URLs embedded in HTML. The query strings in malicious URL are generally large and complex consisting of a number of the ampersands, '&' in the URL query string than in benign URL. While investigating this feature, it was found that the average length of query strings is 18 in malicious URLs while as the average length of query string in benign URL is 8. The heuristic for this feature is expressed as:

```
if (Value of l > α) → Malicious

else, benign
```

Here l represents length of query and $\alpha = 18$.

B. Missing Top Level Domains

The hostname of a URL includes the domain, secondary level domain, and top level domain. The top-level domain part of the URL is checked for existence. While investigating this feature, it was found that the top level domain of a malicious URL was missing in about 80% of studied malicious URLs.

```
if(TLD=Missing) → Malicious

else, benign
```

C. Suffixing/ Prefixing of Dash "-"

Dashes "-" in legitimated websites are very hard to find. Malicious URLs suffix or prefix "-" to the domain names so that users feel the website is trusted. Dash, when used in a URL, is ignored by the user as it does not catch the user's attention due to its small size

and the user believes the site as trusted. In this investigation, it was found that 76% of URLs in this dataset which mimic the benign ones use dash.

D. Length of URL

Length of malicious URLs are usually long and their long length is used to disguise the suspicious part in the address bar. To ensure the accuracy, the average length was calculated for malicious and benign URLs and found that the average length of about 92% malicious URLs in this dataset were 70 characters whereas in benign URLs the average number of characters were 23.

```
if (length of URL> β) → Malacious
 else, benign
```

E. Number of Forward Slashes "//"

The number of forward slashes in a URL shows the presence of subfolders that are used to hide the data in a webpage. Malicious URLs usually have more forward slashes than benign. In this studies, it was found that 96% of studied malicious URLs contain an average of 5 forward slashes whereas benign URLs contain an average of 3 forward slashes.

```
if (number of forward slashes> Υ) → Malacious
 else, benign
```

F. Number of Dots "."

The presence of more dots in the hostname of a URL is to make users believe that they are visiting a genuine webpage. From measurements, it was found that 84% of malicious URLs in dataset have an average of 5 dots in a URL while as only 2 dots were found in a benign URL.

```
if (number of dots> μ) → Malacious
else, benign
```

G. Special Characters

Every webpage or website over the Web has its unique URL made of alphanumeric characters. The special characters are infused into the URL to perform a malicious activity. The special characters used are: { , }< , >, \ , ~ , [,] , ?. While examining, it was found that 88% of malicious URLs contain these special characters and an average of 3 such characters exist in a malicious URL at a time.

```
If (special Character∈ URL>3)
else, benign
```

3.2.2 Webpage Source Features

Webpage source features are extracted by downloading the entire webpage for analysis. In comparison to the features extracted from URL, webpage source features are a bit heavyweight as the vast information needs to be extracted simultaneously for the detection of any malicious code. As more information is available on a webpage for which detection is to be performed, it is logical to assume that it will lead towards an accurate

model of prediction. One drawback of detecting malicious webpages using URL features is that if URL feature fails to detect a malicious URL it will lead to a very high false positive rate. The in-depth analysis of webpage source is, therefore, indispensable to reduce such false positive rate and to perform accurate classification. The webpage source feature can be extracted from the HTML and JavaScript present in a webpage. Algorithm 4.2 extracts webpage source features from JavaScript, and the summary of these features is given in Table 4.3.

Algorithm 4.2: JavaScript Feature Extraction

```
Input: Webpage
Output: Extracted JavaScript Features
1: Get webpage
2: search for <Script> tag
3: if <Script> = found Get JavaScript Until
   </script> tag found endif
4: if "src"= found load external JavaScript endif
5: tokenize all obtained JavaScripts
6: Generate bag of words, extract features
7: Input features → HTMLUnit browser
8: Get features from HTMLUnit browser
```

Table 4.3: JavaScript Features

S. No	Feature Name	Type
1	Set Interval	Unique
2	Hiding Script	Unique
3	Error Message	Unique
4	eval()	Reused
5	iframe()	Reused
6	document.write()	Reused
7	setTimeout()	Reused
8	escape()	Reused
9	Link()	Reused
10	search()	Reused
11	substring()	Reused
12	exec()	Reused
13	attachEvent()	Reused
14	ParseInt()	Reused
15	dynamicUsageContentLen	Reused
16	Changesrc	Reused
17	printSnapshot	Reused

A. Set Interval

The setInterval() method, a recursive process, calls a function or evaluates an expression at specified time intervals (in milliseconds). The setInterval() method continues to call the function until clearInterval() method (cancels a timed repeating action which was previously established by a call to setInterval() method) is called, or the windows is closed.

B. Hiding Script

Along with using a JavaScript function, attackers also use JQuery functions for carrying the malicious script attacks. Detection approaches based on features extracted from JavaScript are not accurate in detecting malicious scripts which use JQuery functions. A hide

method (hide()) using a JQuery function is used to hide some parts of HTML and JavaScript to carry out attack such as shown in below code segment:

```
iframe src ="test.com"

<Script> $ ("iframe").hide();</script>
```

An iframe() is an inline frame HTML document, once embedded inside another HTML document can be used to insert a piece of content from another source into a webpage. As shown in the above code segment, an attacker can use the hide function to hide malicious code. In legitimated webpages, hide function is used to hide some selected HTML elements while having no impact on the layout of the webpage. From dataset of the malicious webpages, it was found that 71% of JavaScripts have used hide function to hide the malicious code usually in obfuscated form.

C. Error Message

Error function (error()) belongs to the category of event handler functions and gets invoked when an error is encountered. This function is employed by the attackers to run their malicious code deliberately on occurrence of an error in order to draw the attention of the user toward the error while a malicious code is being executed in the background. The basic objective of using error function in legitimated webpages is to inform the user about the error or perform some action after the encounter of error. Usage of error function is given here:

```
<img alt="car" id="car" scr =www.noadrees.com/>
</script>
$(#car).error(function()
{
window.location.href = http://malware.com/xss.exe;
}
```

Table 4.4 continued

16	eval()	Source Code	Counts number of eval() functions. eval() executes code passed with privileges of caller.	Livshits et al. 2015
17	iframe()	Source Code	Counts number strings containing iframe(). iframes are used by attackers to serve up drive-by internet attacks, silently & invisibly.	Selim et al. 2016
18	document.write()	Source Code	Counts number of document() functions, writes document stream, calls document.write on closed (loaded) document & automatically calls document.open to clear document. Attackers take advantage of document.write() function to send malicious code.	Xu et al. 2013
19	setTimeout()	Source Code	Counts number of setTimeout() functions.	Ilyushin et al. 2015
20	escape()	Source Code	Counts number of escape() functions. escape() function encodes a string and attackers use it to encode strings to avoid detection.	Corona et al. 2014
21	link()	Source Code	Counts number of link() functions. link() method is used to display a string as a hyperlink. This method returns embedded string that may be malicious.	Hou et al. 2010; Canali et al. 2011
22	search()	Source Code	Counts number of search() functions. search() takes user input as search string & returns its occurrence. As user-provided string is not filtered for escape characters, attackers exploit this vulnerability to attack if a user posts a search string from clipboard.	Hou et al, 2010; Canali et al. 2011
23	substring()	Source Code	Counts number of substtring() function. substring() method extracts characters from a string, between two specified indices & returns new sub string.	Hou et al. 2010; Canali et al. 2011
24	exec()	Source Code	Counts number of exec() functions. exec() replaces program in current process with a brand new program. It is used to inject malicious code during runtime.	Hou et al. 2010; Canali et al. 2011
25	attachEvent()	Source Code	Counts number of attachEvent() functions.	Hou et al. 2010; Canali et al. 2011
26	parseInt()	Source Code	Converts a string into its corresponding numeric value.	Patil et al. 2017
27	dynamicUsageContentLen	Source Code	Stores length of contents that are passed as arguments to APIs which support dynamic execution of JavaScript.	Wang et al. 2015

Table 4.4 continued

28	domAndDynamicUsageCnt	Source Code	Counts number of invocations for APIs that change DOM structure or support dynamic execution of JavaScript code.	Wang et al. 2015
29	Changesrc	Source Code	Counts number of changing src attributes.	Patil et al. 2017
30	PrintSnapshot	Source Code	Allows an attacker to take "screenshots" of web pages or parts of it, directly on the user's browser.	Patil et al. 2017

4.4 Implementation and Experimental Procedure

The static features extracted after performing tokenizing by Feature Extractor were implemented by using JAVA. Feature Extractor was built through a Java class library called JSoup. The dynamic features have been extracted by employing emulated browser HTMLUnit (also Java-based) and implemented by using Java. The classification and training have been performed through a Java-based machine learning tool known as WEKA. The experimental technique is elucidated in the following subsections. The Algorithm 4.3 shows the experimental procedure.

Algorithm 4.3: Experimental Procedure

```
Input: Web page
Output: Malicious or Benign
1: Collect Benign, Malicious Data
2: Extract Static Features using Tokenizing
3: Extract Dynamic Feature using HTMLUnit
4: Create Initial Feature Vector
5: Calculate Chi-square value on each Feature
```

```
6: Calculate Feature Weight for each Feature
7: Remove Feature with Threshold
8: Get Optimal Feature Vector
9: Perform Classification
```

4.4.1 Dataset Collection and Validation

The collected dataset consisting of 32,000 benign webpages and 12,900 malicious pages. The benign webpages were obtained from 50 Top Alexa websites whereas malicious webpages were obtained from websites of VirusTotal and DNS-BH (*http://www.malwaredomains.com/*). Moreover, a dataset of 5,000 benign and 2,000 malicious URLs was used. The malicious URLs were obtained from PhishTank whereas the benign URLs were collected using ParseHub web crawler *(https://www.parsehub.com/)*. 97,430 benign JavaScripts were extracted from 32,000 benign webpages whereas 25,097 malicious JavaScripts were extracted from 12,900 malicious webpages. Validation of a dataset is vitally important in order to have a trustworthy dataset. In particular, the careful validation of the maliciousness of webpages that were used in dataset was performed. For dataset validation, a MD5 hash was generated for each webpage and then compared it with online hash search service provided by VirusTotal. All the obtained MD5 hashes from benign webpages were found clean and similar. On the contrary, the MD5 hashes generated for malicious webpages were found in the database of VirusTotal. Furthermore, benign webpages were scanned for any possible malware by using anti-virus Bitdefender *(https://www.bitdefender.com/)*.

4.4.2 Feature Effectiveness

The purpose of performing this investigation is to classify the features that are effective in characterization of malicious and benign webpages whereas the effectiveness is calculated experimentally and evaluated. The structure of URL and JavaScript are defined to a level of granularity (the level of detail in a dataset) appropriate for classification. An initial feature vector consisting of unique and studied features are evaluated for effectiveness in two ways. The first way is by applying a Chi-Square statistical test with feature weighting, and the second one is by measuring the accuracy of unique & studied features using a classifier. The Algorithm 4.4 performs the feature selection as:

Algorithm 4.4: Feature Selection

Input: Initial Feature Vector

Output: Optimal Feature Vector

1: for f_i=1, 2, ... n do

2: compute χ^2, P-value→ $f_i \in FV_{Initial}$

3: compute TP_i, FP_i $\forall$ $f_i \in FV_{Initial}$ Using Naïve Bayes

4: compute $e_i=TP_i-FP_i$ $\forall$ $f_i \in FV_{Initial}$

5: compute $w_i = e_i/(\sum e_i)$

6: if P-value>T && w_i>T // T=Threshold

7: Put f_i →$FV_{Optimal}$

8: else remove f_i

9: end if

10: end for

4.4.2.1 Empirical Evaluation

The χ^2 test is the process of experientially evaluating the performance of obtained initial feature vector and removing the features not effective in distinguishing malicious and benign codes. The score and p-value of the χ^2 test indicate the strength and statistical significance of a feature's ability to determine the class of a sample. The obtained values are exhibited in Table 4.5. The process was carried out through a 10-fold cross-validation; the dataset was split into 10 parts where 9/10 partitions were used for training and remaining partition was used for testing. Each fold uses a different partition as the testing set. The scores from each combination of partitions were subsequently averaged. Similarly, the calculated weight for each feature is also shown in Table 4.5.

Table 4.5: Results of generating Chi-Square values, p-values, and weight of features.

S. No.	Feature	Rank	χ^2	P-Value	TP	FP	w_i
1	eval()	1	17841.528	0.00	0.954	0.348	0.09454
2	Average function execution time	2	17412.322	0.00	0.982	0.383	0.093448
3	iframe()	3	16664.957	0.00	0.98	0.417	0.087832
4	Number of function calls	4	16470.735	0.00	0.963	0.413	0.085803
5	dynamicUsageContentLen	5	16159.853	0.00	0.979	0.479	0.078003
6	document.write()	6	15589.101	0.00	0.97	0.584	0.060218
7	Maximum script execution time	7	14744.945	0.00	0.966	0.621	0.053822
8	setTimeout()	8	14700.207	0.00	0.931	0.62	0.048518
9	escape()	9	14542.828	0.00	0.96	0.654	0.047738
10	Length of URL	10	13760.829	0.00	0.94	0.634	0.047738
11	exec()	11	13657.709	0.00	0.922	0.623	0.046646
12	Hiddenscript	12	11643.394	0.00	0.963	0.601	0.056474
13	Errormessage	13	7082.294	0.00	0.967	0.693	0.042746
14	search()	14	5670.615	0.00	0.952	0.682	0.042122
15	Missing top level domain	15	5515.699	0.00	0.941	0.671	0.042122
16	Number of Dots	16	4355.018	0.00	0.954	0.792	0.025273
17	Length of query	17	3663.018	0.00	0.953	0.811	0.022153
18	substring()	18	2694.692	0.00	0.959	0.881	0.012168

Table 4.5 continued

19	Number of forward slashes	19	1334.2113	0.00	0.946	0.94	0.000936
20	Special characters	20	907.914	0.00	0.955	0.95	0.00078
21	Suffixing/prefixing of dash	21	842.337	0.00	0.954	0.932	0.003432
22	Setinterval()	22	579.345	0.00	0.956	0.955	0.000156
23	link()	23	537.135	0.00	0.966	0.955	0.001716
24	Number of Client/Server words	24	363.392	0.00	0.942	0.94	0.000312
25	Number of equal signs	25	333.301	0.00	0.955	0.954	0.000156
26	AttachEvent	26	210.3	0.2673	0.955	0.955	0
27	parseInt()	27	59.299	0.1344	0.955	0.955	0
28	PrintSnapshot	28	0	0.1804	0.955	0.955	0
29	Changesrc	29	0	0.3632	0.955	0.955	0
30	domAndDynamicUsageCnt	30	0	0.6232	0.955	0.955	0

Features having a p-value of greater than zero and weight w_i less than zero ($p > 0$, $w_i < 0$), are selected as a threshold. Figure 4.7 depicts the performance of each feature against χ^2 and feature weight w_i.

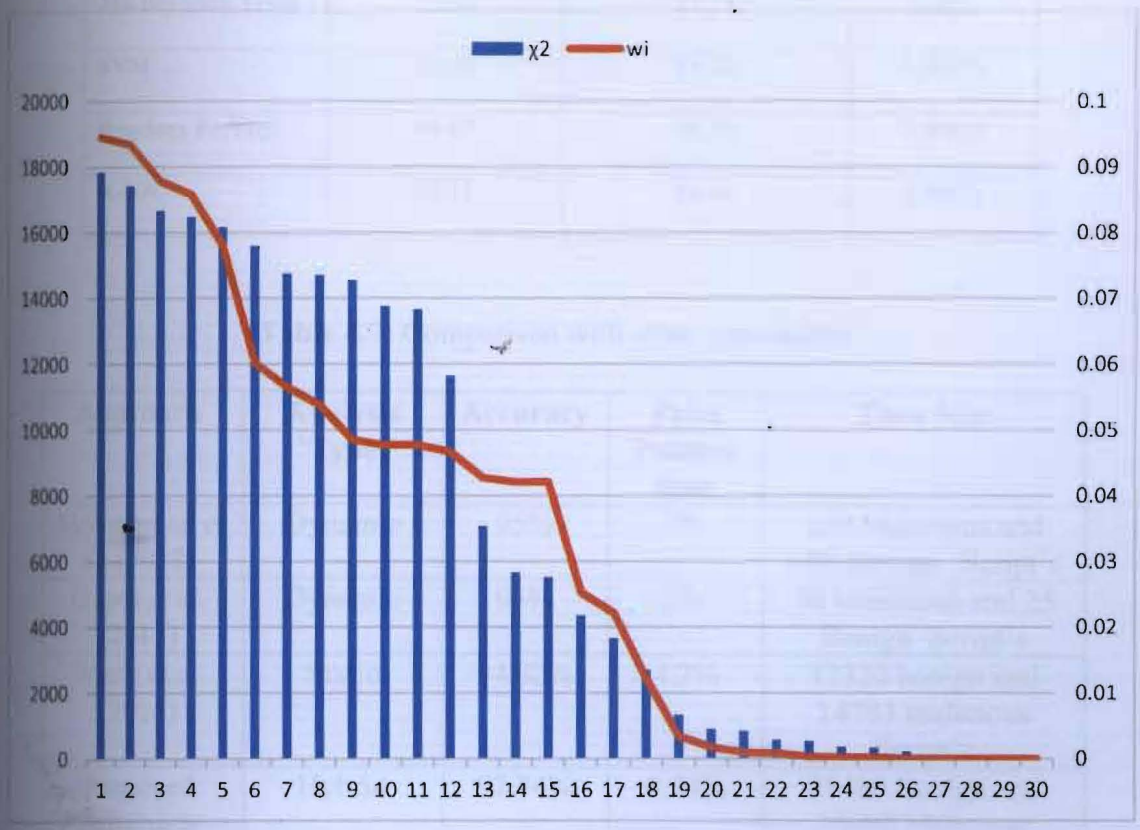


Figure 4.7: Statistical Performance of each Feature

4.4.2.2 Classification

In order to verify whether the unique features introduced in the present study are predictive important and leads to high accuracy in detection of malicious webpages, five classifiers were used in evaluating the performance. The classification results obtained by using existing and unique features are depicted in Table 4.6. In both cases of classification with and without unique features, features were selected by maintaining the threshold, $p>0$ and $w_i<0$.

Table 4.6: Classification Results

Classifier	Accuracy Without Unique Feature (%)	Accuracy with Unique Feature (%)	Change (%)
Naïve Bayes	94.84	96.03	1.19(↑)
J48 Decision Tress	95.34	97.74	2.4(↑)
SVM	93.49	95.32	1.83(↑)
Random Forests	94.67	96.36	1.99(↑)
K-NN	95.11	96.01	0.90(↑)

Table 4.7: Comparison with other approaches

Approach	Analysis Type	Accuracy	False Positive Rate	Data Size
Mohammad et al. (2012)	Dynamic	95%	3%	100 Malicious and 100 Benign Script's
Gupta et al. (2013)	Dynamic	96%	2%	50 Malicious and 25 Benign Script's
Wang et al. (2016)	Static	94.82%	4.2%	12320 benign and 14783 malicious Script's
Proposed Approach	Hybrid	97.74%	0.54%	97430 Benign and 25097 Malicious

Comparison with other approaches depicted in Table 4.7 shows that proposed approach performed better in accuracy with low false positive rate than others using single classifiers. In this approach J48 Decision tree achieved 97.74% accuracy whereas a study by (Mohammad et al., 2012) achieved by using decision tree classifier, study by Gupta et al., 2013 achieve 96% accuracy with 2% false positive rate using logistic regression similarly study by Wang et al., 2016 achieved 94.82% with a high false positive rate of 4.2% using SVM classier.

4.5 Performance Overhead

The experiment was conducted using an Intel Core i5-7300HQ processor having 64-bit Windows 7 with memory of 4 GB. With this infrastructure, the average time to train a classifier was recorded as 1.54 seconds. Under variable load conditions, this approach took an average of 2.3 seconds to detect whether a webpage is malicious or benign. This is an acceptable performance overhead as the part of webpage analysis is performed by the emulated browser. However, the performance overhead with malware analysis tools could not be verified as they take the processes in batch form and introduce delay due to queuing.

4.6 Discussion on Results

The use of extracted features from static and dynamic aspects has proven to be effective in detecting malicious and benign webpages. This study relied on 30 features out of which 8 features were unique and rest selected from previous studies. During experiential testing, it was found that some features are ineffective that were considered in the literature as effective. The Chi-Square and feature weight played an important role in selecting features which contributed towards high accuracy by putting threshold between effective and non-effective features. From the experiential testing using Chi-Square and feature weight, the

threshold selected as the p-value started diverging away from 0 and feature weight w_i converging towards 0. Both the tests confirm the effectiveness of the features.

All the newly introduced 8 features performed very well as none of them were found to be ineffective during empirical testing and classification. Features such as average execution time, maximum execution time, and function calls seem to be more effective as they obtained a rank of 2, 4 and 7, respectively as shown in Table 4.5. This corroborates the assumption that malicious code requires more execution time than benign code. Malicious code requires more computation resources for performing memory scanning, altering system settings, and creating heap overflow once the attack is successful. In addition, some malicious scripts require more time to carry drive-by-download attack and call other malicious third parties. The feature function call suggests that a legitimate function calls more functions in order to implement the business logic of the website while as the malicious function calls fewer functions as their objective is to attack. The business logic of such malicious websites are poor.

The obfuscated code is contained inside an eval() function which evaluates the JavaScript code represented in a string. An eval() is considered dangerous function because it provides a way to access the content to the caller and acts as a primary carrier in the delivery of malicious code in obfuscation. During obfuscation, JavaScript can be split into multiple variables and substrings, and are combined during obfuscation using document.write() function. This function is used for printing the text value and can also be exploited to inject the malicious code when the page is loaded. Similarly, iframe() tag enabling other documents to be in line with the current document can also be exploited during obfuscation to carry the malicious attack. These three features viz.eval(),

document.write(), and iframe() have achieved a rank of 1, 3, and 6, respectively during empirical evaluation; thus become very important for classification. Classification results shown in Table 4.6 reveal that newly introduced features help in improving the performance accuracy from previously studied features with Naïve Bayes (1.19 % ↑), J48 Decision Tress (2.4 % ↑), SVM (1.83 % ↑)), Random Forests (1.99 % ↑), and K-NN (0.90 % ↑). The newly introduced features and their implementation in classification successfully demonstrated that classifiers such as Naïve Bayes, J48 Decision Tress, SVM, Random Forests, and K-NN were able to detect more 298, 602, 459, 499, and 255 attacks, respectively against earlier features.

4.7 Summary

In this Chapter, an approach for defending malicious JavaScript attacks efficiently is presented by leveraging static and dynamic analysis coupled with machine learning. The experimental results displayed here establish that this approach can effectively detect malicious webpages with high accuracy. The newly introduced features are very efficient in improving the classification accuracy. This approach incurs acceptable performance overhead in characterizing a webpage in real time due to effective features used in previous studies and due to the unique introduced features.

CHAPTER 5

MALICIOUS JAVASCRIPT CLASSIFICATION BY ENSEMBLE

5.1 Overview

Detection of malicious code attacks using machine learning techniques has been explored largely to facilitate the fast and accurate detection of new variants of malicious code using single classifiers. Conversely, there is no agreement yet which classifier produces the best prediction results. A particular classifier might be better than its counterparts for a particular dataset; nevertheless another classifier could achieve better results for some other dataset. The ensemble of machine learning classifier has demonstrated its effectiveness in improving the classification accuracy. In order to harness the power of multiple distinct classifiers and explore their strengths, an efficient heterogeneous multi-level ensemble using enhanced bagging and multi-objective optimised weighting is proposed. The fundamental objective of this method is to find the best ensemble for detection of malicious JavaScript in terms of accuracy, specificity, sensitivity and computational cost.

The new approach has two main characteristics which are portrayed as:

1. An efficient heterogeneous multi-level ensemble has been developed by exploiting improved bagging and multi-objective optimised weighting techniques for the detection of malicious JavaScript code attack.

2. Augmentation of performance has been achieved via resampling of the dataset by employing the improved bagging method.

The remaining of this chapter is organized as follows: Section 5.2 presents a meticulous description of the approach along with the methodology for instituting the ensemble, Section 5.3 provides results and discussions, and Section 5.4 recaps the chapter findings.

5.2 The New Ensemble Approach

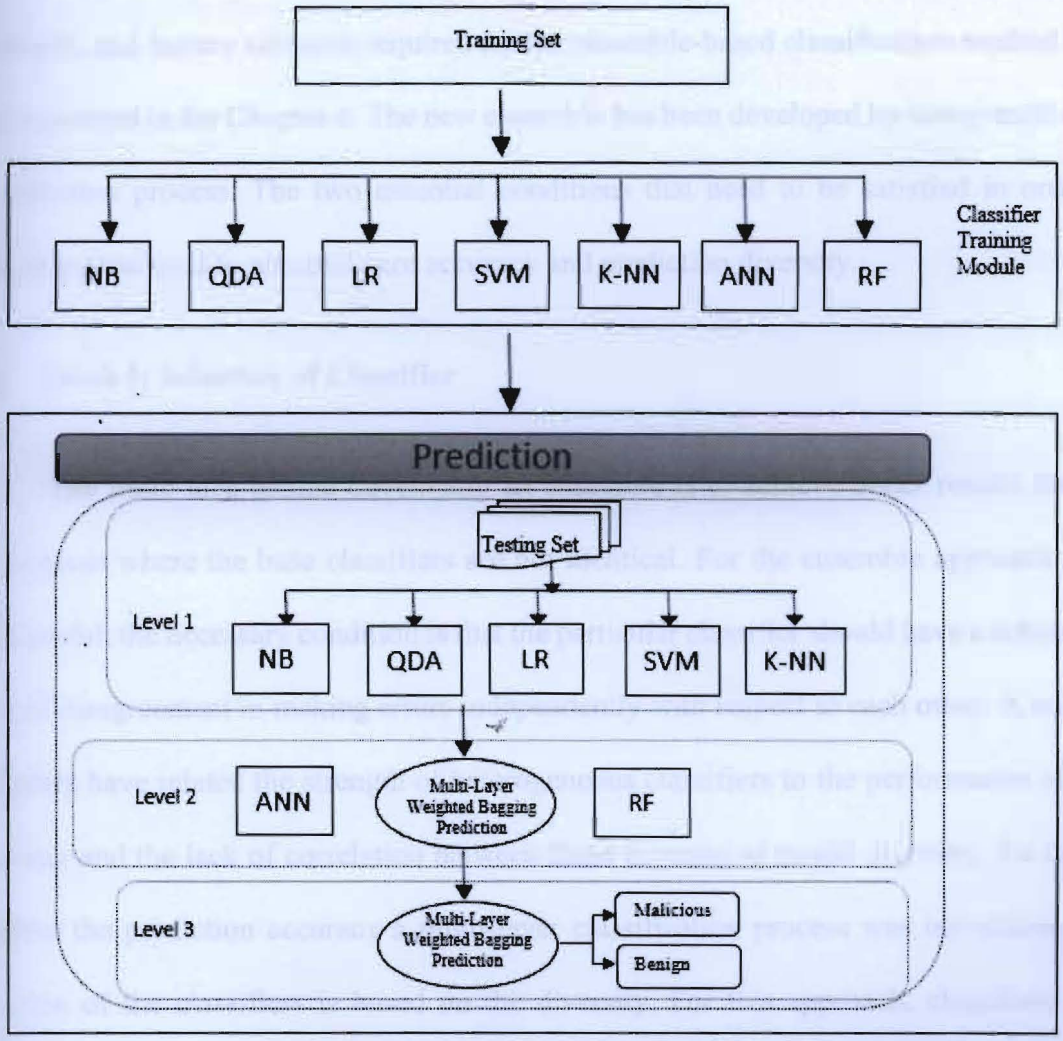


Figure 5.1: Exposition of New Ensemble Approach for Detection of Malicious Code.

Figure 5.1 illustrates the entire course of action for the detection of malicious code using the proposed classifier ensemble. The primary emphasis of the ensemble-based approach is to boost the detection accuracy by creating a set of classifiers that differ on their decision by changing the training process resulting in a different classification. In order to achieve heterogeneity, seven classifiers namely Naïve Bayes, QDA, LR, SVM, Random Forest, K-NN, and ANN has been used. The mathematical formalism of these supervised classifiers has been presented in Chapter 2. The data collection process, hybrid feature extraction, and feature selection required for the ensemble-based classification method have been presented in the Chapter 4. The new ensemble has been developed by using multi-level classification process. The two essential conditions that need to be satisfied in order to achieve a good quality ensemble are accuracy and prediction diversity.

5.2.1 Level-1: Selection of Classifier

The basic idea behind developing an ensemble is to achieve better results than its components where the base classifiers are not identical. For the ensemble approach to be more useful, the necessary condition is that the particular classifier should have a substantial level of disagreement in making errors independently with respect to each other. A number of studies have related the strength of heterogeneous classifiers to the performance of base classifier and the lack of correlation between them in terms of model diversity. To further improve the prediction accuracy a multi-layer classification process was introduced. The selection of the classifiers is based on the diversity. For this approach, classifiers were selected from the recently conducted studies which produced the highest classification and prediction accuracy for detection of malicious code. Five classifiers used at this level are: Naïve Bayes, SVM, QDA, LR, and K-NN. The new ensemble method employs classifiers

having a diverse set of qualities complementing each other to compose an accurate ensemble for the detection of malicious JavaScript code attacks. The diversity between classifiers can be calculated by the level of disagreement about the probability distribution for the test datasets. All the selected classifiers for this ensemble method complement each other. Each individual classifier predicts instances in one class either 0 or 1, and the weight for each classifier is calculated based on F-Measure.

5.2.2 Level-2: Selection of Classifier

The output produced at level-1 is combined at this level using weighted bagging. To achieve more diversity, at this level two more classifiers are used: Random Forest and Artificial Neural Networks (ANN). At level-1, no classifier from the category of Random Forests was included; instead of using a single tree classifier at this level random forests was introduced. The property of random forests is that it operates by developing a number of decision tress at training time and generates the class which is the mode of classes or mean prediction of the individual trees. One main advantage of using Random Forests is that it has low classification errors and deals very well with uneven dataset possessing missing values. Radom Forests also have better F-score and are fast to build. ANN is based on iterative network learning with the property of solving multiple problems without finding and describing methods. Since ANN has a capability to implicitly detect complex nonlinearity between variables, this capability makes ANN ideal for low-error prediction. The weight for each classifier is computed based on the F-Measure of the training dataset.

5.2.3 Level-3: Selection of Classifier

At this level, for each instance the final predictions are generated. All the results generated at previous levels are combined here using weighted bagging ensemble and the output so produced represents the final result.

5.2.4 Pareto Optimized Weighted Voting Ensemble

The ensemble approach for the detection of malicious JavaScript code uses Pareto multi-objective optimization. Efforts has been to optimise multiple objectives simultaneously as the traditional learning and optimization algorithms are limited to single objective optimization only. In Pareto multi-objective optimization, the objective function is no longer a scalar value but a vector with multiple of Pareto optimal solutions rather than a single solution which can formally be expressed as:

Find the number of vectors V_k *, where* $V_k = \{V_1, V_2, V_3, \ldots . V_n, \}$ *for each classifier* C_k *and* $C_k = \{C_1, C_2, C_3, \ldots . C_n\}$ *, such that N objective criterion is optimized simultaneously while satisfying the constraints, if any.*

In this ensemble approach, Precision and Recall as two objective functions have been used to increase the number of benign samples and to achieve the correct number of benign samples, respectively to the best possible manner. These objective functions are able to attain two different classification qualities as there is an inverse relationship between the two objectives; the quality of one objective enhances at the expense of reducing the quality of other objective. For each classifier, F-Measure is computed for the training data which produces values (weights) based on precision and recall combination. The resultant weight

is then normalized by using Min-Max normalization to the F-Measure values translating each result into either 0 or 1. The Min-Max normalization is performed by using the following expression (Jain et al, 2013)

$$V_i' = \frac{V_i - min_A}{max_A - min_A}(new_{max_A} - new_{min_A}) + new_{min_A} \quad 5.1$$

Here V_i is the value needed to be normalized, $\min_A$ the minimum value for a given dataset, $\max_A$ the maximum value for a given dataset, and V_i' the normalised output. Following example further explains the working of the propsoed ensemble:

Example: Suppose a set of heterogeneous five classifiers C_1, C_2, C_3, C_4 and C_5 on training data and subsequent application of F-Measure give results:

$$C_1 = 60\%, C_2 = 70\%, C_3 = 80\%, C_4 = 85\%, \; C_5 = 90\%$$

1. Now, the votes are computed by the formula given in Eq. 5.1 having values of $new_{max} = 1$, $new_{min} = 0.1$, $max = 100$, and $min = 0$. The calculated weights obtained are:

$$C_1 = 0.6, C_2 = 0.7, C_3 = 0.8, C_4 = 0.85, C_5 = 0.9$$

2. Suppose that prediction results generated by each classifier classes the test instance as:

$$C_1 = \text{Class } 0, C_2 = \text{Class } 0, C_3 = \text{Class } 1, C_4 = \text{Class } 0, \; C_5 = \text{Class } 1$$

3. Now the ensemble, when the weights for each classifier are added that have voted for a particular class, will produce the following predictions:

$$\text{Class } 0: \; C_1, C_2, C_4 \rightarrow 0.6 + 0.7 + 0.85 = 2.15$$

$$\text{Class } 1: \; C_3, C_5 \rightarrow 0.8 + 0.9 = 1.7$$

Based on the results obtained by weighted vote, Class 0 has the highest weight than Class 1, thus the test instance will be classified as Class 0.

In multi-objective optimisation, a number of non-dominated solutions are produced by algorithm on the final Pareto optimal front. The Pareto front is a set of non-dominated solutions, being chosen as optimal if no objective can be improved without sacrificing at least one other objective. A solution x^* is referred to as dominated by another solution x if, and only if, x is equally good or better than x^* with respect to all objectives. All the solutions produced are equally important from the algorithmic point of view. For every solution on the final Pareto optimal front, the overall average F-measure value using tenfold cross-validation is calculated on the training data. The solution with the maximum F-measure value is selected as the best solution. Final results on the test data are reported using the classifier ensemble corresponding to this best solution.

5.2.5 Improved Bagging

Bagging method has an objective of sampling datasets to be used by an ensemble of classifiers. In this approach, an improved bagging method is presented for re-sampling of subsets drawn from an original dataset used by classifiers in order to aggregate the individual predictions to form the final method. The main objective has been to reduce the variance of base estimators by bringing randomisation into it and then forming an ensemble. Algorithm 5.1 shows the improved bagging. The improved bagging approach can be divided into two stages. At the first stage, the original training set is segregated into multiple bootstrap sets with replacement as explained in flowchart given in Figure 5.2. Each bootstrap created from training set has a size, s and i iterations are performed in which

one of the m examples is drawn from each iteration and the probability of drawing each example from iteration is $1/m$. The ensemble picks up a sample f from 1 to m, and f sample is added to new bootstrap training. Furthermore, there is a possibility that some of the training examples may be selected more than once. At the second stage, the training is performed by classifiers on the bootstrap training sets produced in the first stage. The individual classifiers were considered in the bagging as they should not be of same weight with the rationale that each classifier achieves a different performance level. Instead of using simple voting multi-objective optimisation has been introduced. It is estimated that the ensemble will return a function h(d) that classifies the instance into class 0 or class 1 based on the highest weight obtained from base learned models $h_1, h_2, h_3, \ldots h_n$. The proposed ensemble model is anticipated to perform better than any individual classifier if the difference among bootstrap training sets induces a noticeable difference among M individual classifiers; thereby generating reasonably good performance by each of them.

Algorithm 5.1: Improved Bagging

```
Let k be the number of bootstraps with size m
1. For i=1 to k do
2. Draw f from ki with P(f)=1/f // probability of
   selecting sample
3. pick sample f from 1 to k, add f to new bootstrap s
4. Input s for classification
```

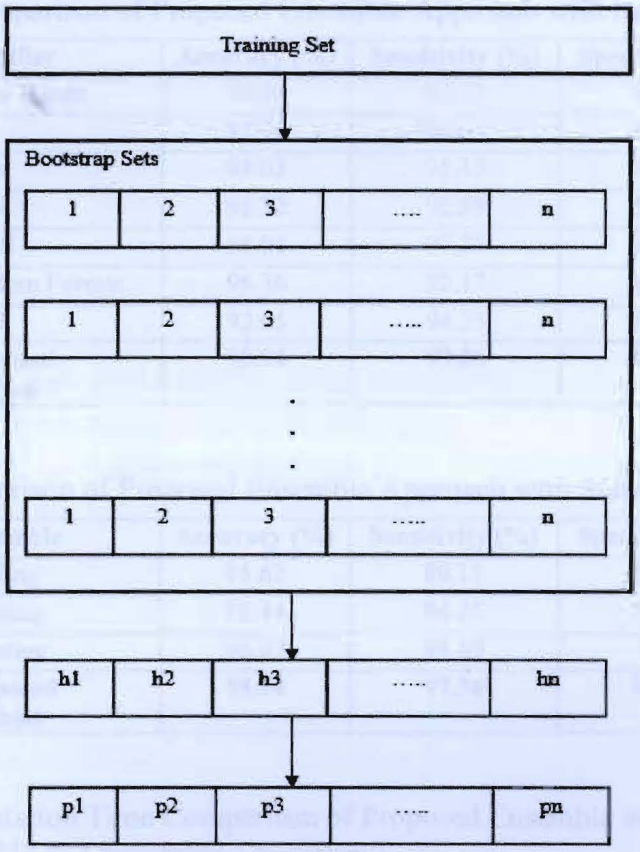


Figure 5.2: Flowchart for Improved Bagging

5.3 Results and Discussion

The present ensemble approach is executed on the dataset obtained for the detection of malicious JavaScript code attacks. The dataset description and optimal features used here have been discussed in detail in the Chapter 4. Table 5.1 shows the comparison of accuracy, sensitivity, and specificity of the proposed ensemble with individual classifiers. Table 5.2 presents the comparison of accuracy, sensitivity, and specificity of the proposed ensemble with state-of-the-art ensembles. The analyses of the results demonstrate that high accuracy of 98.74%, sensitivity 97.56% and specificity 98.86% were obtained using present ensemble approach.

Table 5.1: Comparison of Proposed Ensemble Approach with Individual Classifiers.

Classifier	Accuracy (%)	Sensitivity (%)	Specificity (%)
Naïve Bayes	96.03	92.23	92.61
LR	92.56	93.65	87.54
QDA	94.03	95.15	97.07
SVM	95.32	92.59	97.38
K-NN	96.01	97.37	95.42
Random Forests	96.36	92.17	95.78
ANN	93.05	94.25	88.21
Proposed Method	**98.74**	**97.56**	**98.86**

Table 5.2: Comparison of Proposed Ensemble Approach with State of the Art Ensembles.

Ensemble	Accuracy (%)	Sensitivity (%)	Specificity (%)
Grading	95.62	89.15	98.44
Stacking	96.14	94.35	97.18
Boosting	96.07	94.65	9.22
Proposed Method	**98.74**	**97.56**	**98.86**

Table 5.3: Computation Time Comparison of Proposed Ensemble with Individual and Ensemble and Ensemble Classifiers.

	Classifier/Ensemble	Time (ms) per Instance
Individual Classifier	Naïve Bayes	.06
	LR	.03
	QDA	.05
	SVM	.04
	K-NN	.06
	Random Forests	.08
	ANN	.06
Ensemble	Grading	104.83
	Stacking	96.94
	Boosting	107.88
	Proposed Method	**113.06**

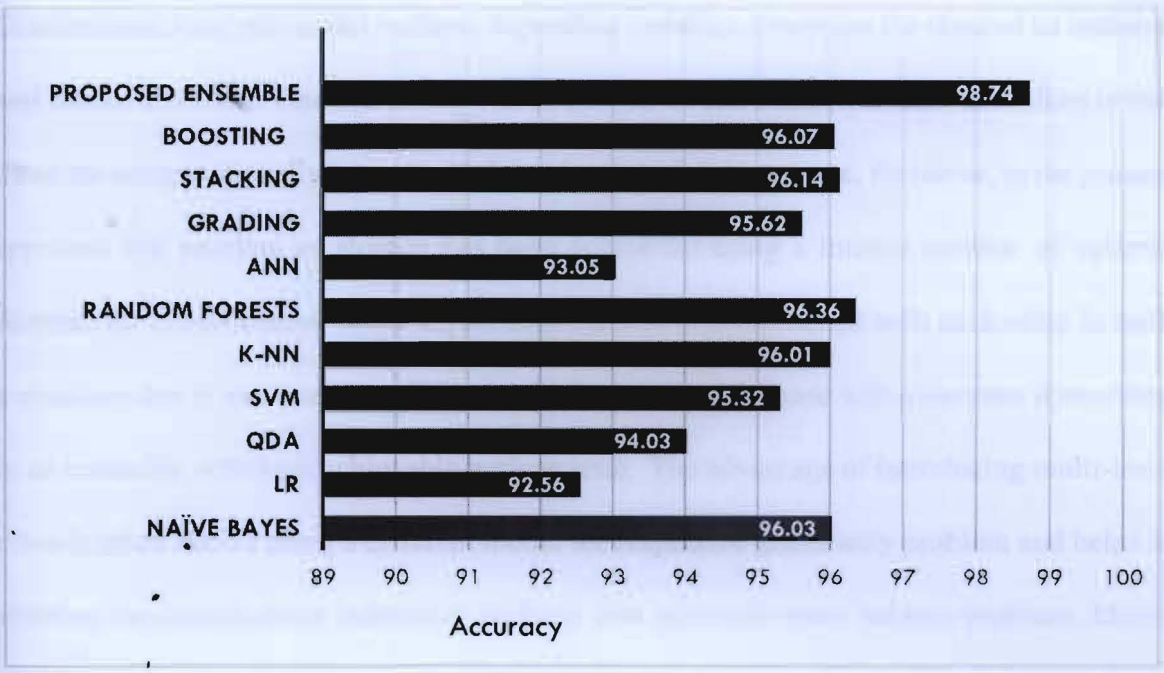


Figure 5.3: Comparison of accuracies between Present Ensemble Approach with Individual Classifiers and State of the Art Ensemble Approaches.

The comparison of accuracies is presented in Figure 5.3 which shows that the proposed ensemble approach has achieved highest performance accuracy. This best performance report among compared individual classifiers and ensembles can be attributed to the utilization of an optimal model of class diversity in the present ensemble approach. The present ensemble consists of heterogeneous supervised classifiers that belong to the groups such as Naïve Bayes, Support Quadratic, Discriminant Analysis, Linear Regression, Instance-Based Learning, Support Vector Machines, Random Forests and Artificial Neural Networks. The property of using Naïve Bayes is that it treats every attribute individually without taking into the consideration the relation among them. The proposed ensemble approach handles the dependency and relations among attributes by using K-NN classifier where the neighbour decides the class of an instance. The property of using Linear Regression model is that it statistically determines the relationship that exists between dependent and independent variables and achieves optimal results. In Quadratic

Discriminant Analysis model multiple dependent variables determine the class of an instance and reduce error rate. One drawback with K-NN, ANN and Random Forest classifiers is that these are computationally intensive and requires lot of storage space. However, in the present approach the problem of storage has been solved by using a limited number of optimal features for classification. Since all the used classifiers complement with each other in such a situation that if any one classifier has a limitation the other one will overcome it resulting in an ensemble with high achievable performance. The advantage of introducing multi-level classification allows using a different model for respective granularity problem and helps in splitting the classification imbalance problem into relatively more balance problem. Multi-level classification provides a great flexibility of using a different set of features by a different classifier. However this study is confined to using of same features to all classifiers. This multi-level hierarchical-structure delivers better performance results than the single-level flat-structure because of the computational errors at each level, and facilitates in reducing the risk of errors in top-down classification process.

The training time consumed by the present ensemble approach is compared with individual classifiers and state of the art ensemble and is shown in Table 5.3. Results acquired demonstrate that the present ensemble approach has a higher time complexity of 113.06 ms per instance as compared to the individual as well as state of the art ensemble approaches. The reason behind the higher time complexity is that the present ensemble uses heterogeneous classifiers with enhanced bagging. However, this time complexity does not affect the overall performance of the ensemble approach as off-line learning model commonly used in expert systems has been used. This time complexity is restricted to training phase alone because at runtime only learned model is used for prediction.

5.4 Summary

In this Chapter, an ensemble approach for classification of malicious JavaScript attacks was presented. This ensemble approach was developed by using multi-layer and uses multi-objective optimisation for classification. To resample the dataset an improved bagging Algorithm 5.1 was presented. The experimental results reveal that the present ensemble approach offers best performance as compared to individual classifiers and the state of ensemble approaches for detection of malicious webpages with high accuracy. Results also demonstrate that this approach incurs negliable performance overheads in the classifcation process.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

Internet has indeed revolutionized the world we live in through its multitasking features. The Web constitutes the technological infrastructure of the global network society. Unfortunately, the Web is being misused by the cybercriminals through launching of malicious code attacks to manage individual, pecuniary, clandestine and unethical gains via damaging or disrupting the normal functioning of computer systems, conducting phishing attacks, exhibiting superfluous advertisements and bullying money. Cybercriminals manoeuvre the convenience and flexibility of Web by exploiting the vulnerabilities through malicious activities targeting naive users. Malicious code attacks crafted inside the webpages are launched on a mere visit by the user resulting in loss of the confidential information. The attackers also use the victim's devices to infect other devices and carry complex attacks. Among the most exploited vulnerabilities include XSS which is generally considered simple to operate but difficult to detect owing to the flexible nature of HTML encoding schemes.

Existing countermeasures for mitigating malicious JavaScript-based XSS attacks on webpages are commendable but their success pathway is obstructed due to partial analysis and poor characterization of webpages, emergence of artefacts in webpages, and induction of classification error by single classifiers. The research work presented in this thesis addressed these challenges and developed a unified approach with a distributed goal of performing the effective analysis and carrying out the comprehensive characterization with high accuracy of detecting malicious JavaScript code attacks in webpages.

To address the challenge of deficient analysis and inadequate characterization as well as evolving artifacts in webpages, we implemented the envisaged approach which coalesce the static and dynamic analyses with simplistic imitation for feature extraction. In addition to this, 8 unique features were introduced in this approach. The evaluation results reveal that the new approach performed better for the effective detection and the improved performance is attributed to the induction of unique features in the approach. For addressing the challenge of classification errors introduced by single classifiers, a multilevel heterogeneous classifier ensemble technique based on Pareto optimization was implemented. The evaluation results demonstrate that high accuracy, enhanced sensitivity, and improved specificity were achievable with low-performance overheads by this method.

Thus, it is anticipated to a greater extent, the attackers would not have an easy walk in successful launching of such malicious attacks and the end-users may heave a sigh of relief by effective detection of malicious JavaScript code attacks on the implementation of the present approach though detection of covert malicious attacks altogether still looks tough.

6.2 Future Works

The classification algorithms implemented in this study for the detection of malicious JavaScript code attacks were mainly limited to those supervised classifiers whose performance was found good in literature. Very recently, on 30-December-2017, a new classification algorithm called XGBoost (Extreme Gradient Boosting) was introduced in classification domain designed for speed and performance. An extensive effort may find better-suited algorithms with high performance yield than traditional classification algorithms used in this approach. Further, the algorithms used in this study were

implemented on Central Processing Unit (CPU), which has less processing speed than Graphics Processing Unit (GPU). GPU can be used in the model for making a precise prediction in extremely short time. Recent Google invented Tensor Processing Unit (TPU), specifically for artificial intelligence and machine learning, can be used for fast prediction which will minimize the computational cost in complex models.

REFERENCES

Aebersold, S., Kryszczuk, K., Paganoni, S., Tellenbach B. & Trowbridge T. (2016). Detecting obfuscated JavaScripts using machine learning. *Proceedings of 11th International Conference on Internet Monitoring & Protection (ICIMP)*, 11-16.

Abu, M., Yaser, S., Malik, M. I. & Hsuan, T. L. (2012). *Learning From Data-A Short Course*. AMLBook.com: New York, USA, 1-197.

Ablon, L., Libicki, M. C. & Golay A. A. (2014). Markets for Cybercrime Tools and Stolen Data: *Hackers' Bazaar. Rand Corporation Document*, U.S., 3-83.

Acker, S. V. & Sabelfeld, A. (2016). JavaScript Sandboxing: isolating and restricting client-side JavaScript. *Foundations of Security Analysis and Design VIII (FOSAD),* Springer International Publishing Switzerland, 32-86.

Agten, P., Van, A. S., Brondsema, Y., Phung P. H., Desmet, L. & Piessens, F. (2012). JSand: complete client-side sandboxing of third-party JavaScript without browser modifications. *Proceedings of 28th ACM Annual Computer Security Applications Conference*, 1-10.

Aha, D. W. & Bankert, R. L. (1996). A comparative evaluation of sequential feature selection algorithms. *Learning From Data*. Springer: New York, U.S. 199-206.

Altman, N. (1992). Introduction to kernel and nearest-neighbour nonparametric regression. *The American Statistician*, 46, 175–184.

Anzai, Y. (2012). *Pattern Recognition and Machine Learning*. Elsevier, Amsterdam, Netherlands, 1-403.

Bajpai, A. (2010). Securing Apache, Part 2: *XSS Injections*. http://opensourceforu.com/ 2010/09/ securing-apache-part-2-xss-injections/, Accessed on 26/12/2017.

Baranwal, A.K. (2012). Approaches to detect SQL injection and XSS in web applications. *Term Survey Paper, EECE 571B*, University of British Columbia 1-12.

Barth, A., Weinberger J. & Song D. (2009). Cross-origin JavaScript capability leaks: detection, exploitation, & defence. *USENIX Security Symposium*, 187-198.

Berkhin, P. (2006). A survey of clustering data mining techniques. *Grouping Multi-Dimensional Data*, Springer: Berlin, Heidelberg, 25-71.

Berners, L. T. & Fischetti, M. (1999). *Weaving the Web: the original design and ultimate destiny of the World Wide Web by its inventor*. HarperOne, San Francisco, U.S, 1-226.

Bermingham, M. L., Pong W. R., Spiliopoulou, A., Hayward, C., Rudan I., Campbell, H. & Haley, C. S. (2015). Application of high-dimensional feature selection: evaluation for genomic prediction in man. *Nature Scientific Reports*, 5, 10312, 1-12.

Best H., & Wolf, C. (2015). Logistic regression. *The SAGE Handbook of Regression Analysis and Causal Inference*. Sage: Los Angeles, CA, U.S., 153-171.

Boser, B. E., Guyon I. M. & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of 5th ACM Annual Workshop on Computational Learning Theory*, 144-152.

Boire, M. M., Marschalek, M. & Guarnieri, C. (2015). Big game hunting: the peculiarities in nation-state malware research. *Proceedings of BlackHat USA Security Conference*, 1-12.

Breiman, L. (1996). Bagging predictors. *Machine Learning,* Springer, 24, 123-140.

Brinskelle, J. E. (2014). Enforcement of same origin policy for sensitive data. U. S. Patent No. 8,856,869.

Brown, G., Wyatt J., Harris, R., Yao, X. (2005). Diversity creation methods: a survey and categorisation. *Information Fusion*, 6, 5–20.

Canali, D., Cova, M., Vigna, G. & Kruegel, C. (2011). Prophiler: a fast filter for the large-scale detection of malicious web page. *Proceedings of 20th International Conference on World Wide Web*, 197-206.

Canali, D. & Balzarotti, D. (2013). Behind the scenes of online attacks: an analysis of exploitation behaviours on the Web. *Proceedings of 20th Annual Network & Distributed System Security Symposium*, 5-12

Canfora, G., Mercaldo, F. & Visaggio, C. A. (2014). Malicious JavaScript detection by features extraction. *e-Informatica Software Engineering Journal*, 8, 65-87.

Cao, B., He L., Kong, X., Philip ,S. Y., Hao, Z., & Ragin, A. B. (2014). Tensor-based multi-view feature selection with applications to brain diseases. *Proceedings of IEEE International Conference on Data Mining (ICDM)*, 40-49.

Cavallaro, L., Saxena, P., & Sekar, R. (2008). On the limits of information flow techniques for malware analysis and containment. *International conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Berlin, Heidelberg, 143-163.

Chandra, A. & Yao, X. (2004). DIVACE: diverse and accurate ensemble learning algorithm. *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, Berlin, Heidelberg, 619–625.

Christopher, M. B. (2006). Pattern recognition and machine learning. *Information Science and Statistics*, Springer-Verlag New York, 7-15.

Chellapilla, K., & Maykov, A. (2007). A taxonomy of JavaScript redirection spam. *Proceedings of 3rd ACM International Workshop on Adversarial Information Retrieval on the Web*, 81-88.

Choi, H., Zhu, B. B. & Lee, H. (2011). Detecting malicious web links and identifying their attack types. *Proceedings of 2nd USENIX Conference on Web Application Development*, 125-136.

Christodorescu, M. (2007). *Behavior-based malware detection*. Doctoral Dissertation, University of Wisconsin at Madison.

Codingsec (2016). https://codingsec.net/2016/07/document-object-model-javascript/ Accessed on 19/12/2017.

Corona, I., Maiorca, D., Ariu, D. & Giacinto, G. (2014). Luxor: Detection of malicious pdf-embedded JavaScript code through discriminant analysis of api references. *Proceedings of ACM Workshop on Artificial Intelligent & Security Workshop*, 47-57.

Cova, M. (2010). *Taming the malicious web: avoiding and detecting web-based attacks*. Doctoral Thesis, Department of Computer Science, University of California, Santa Barbara, U.S.

Cova, M., Kruegel, C. & Vigna, G. (2010). Detection and analysis of drive-by-download attacks and malicious JavaScript code. *Proceedings of 19th ACM International Conference on World Wide Web*, 281-290.

Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society*, Series B (Methodological), 20, 215-242.

Curtsinger, C., Livshits, B., Zorn, B. G. & Seifert, C. (2011). ZOZZLE: fast and precise in-browser JavaScript malware detection. *Proceedings of USENIX Security Symposium*, 33-48.

Dai, C., Pang, J., Yue, F., Cui P., Sun, D. & Zhu, L. (2016). A detection framework of malicious code based on multi-classifiers ensemble. *International Journal of Security & Its Applications*, 10, 75-86.

Darling, M., Heileman, G., Gressel, G., Ashok, A. & Poornachandran P. (2015). A lexical approach for classifying malicious URLs. *Proceedings of IEEE International Conference on High Performance Computing & Simulation (HPCS)*, 195-202.

Dash, M. & Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis*, 1, 131-156.

Dewald, A., Holz, T. & Freiling, F. C. (2010). ADSandbox: Sandboxing JavaScript to fight malicious websites. *Proceedings of ACM Symposium on Applied Computing*, 1859-1864.

Dietterich, T. G. (2000). Ensemble methods in machine learning, *First International Workshop on Multiple Classifier Systems, MCS2000*, 1-15.

Eluyode, O. S. & Akomolafe, D. T. (2013). Comparative study of biological and artificial neural networks, *European Journal of Applied Engineering & Scientific Research*, 2, 36-46.

Eshete, B., Villafiorita, A. & Weldemariam K. (2012). BINSPECT: Holistic Analysis and Detection of Malicious Webpages. *Proceedings of International Conference on Security & Privacy in Communication Systems*, 149-166.

Fan, R. E., Chen, P. H., & Lin, C. J. (2005). Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6, 1889-1918.

Ferguson, N. & Schneier, B. (2003). *Practical Cryptography*. Wiley, New York, U.S.

Freund, Y. & Schapire, R. E (1997). A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer & System Sciences*, 55, 119-139.

Garera, S., Provos, N., Chew M. & Rubin, A. D. (2007). A framework for detection and measurement of phishing attacks. *Proceedings of Association for Computing Machinery Workshop on Recurring Malcode*, 1-8.

Gerard, B. & Erwan, S. (2016). A random forest guided tour. *Journal of the Spanish Society of Statistics and Operations Research*, 25, 197–227.

Grobauer B., Walloschek T., & Stocker E. (2011). Understanding cloud computing vulnerabilities. *IEEE Security & Privacy Magazine*, 9, 50-57.

Griffin, K., Schneider, S., Hu X. & Chiueh, T. C. (2009). Automatic generation of string signatures for malware detection. *Proceedings of Workshop on Recent Advances in Intrusion Detection*, 101-120.

Guldamlasioglu, S. (2015). *Web-based Information Visualization using JavaScript*, M.Sc. Thesis. School of Information Sciences, University of Tampere, Finland.

Gundy, M. V. & Chen, H. (2009). Noncespaces: using randomization to enforce information low tracking and thwart cross-site scripting attacks. *Proceedings of Network & Distributed System Security Symposium (NDSS)*, 1–18.

Guyon, I. & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3, 1157-1182.

Hall, M. A. (1999). *Correlation-Based Feature Selection for Machine Learning*. Ph. D. Thesis, Department of Computer Science, Hamilton, New Zealand.

Hes, R., Komisarczuk P., Steenson, R. & Seifert, C. (2009). *The Capture-HPC client architecture*. Technical Report, Victoria University of Wellington, ECSTR09-11.

Ho, T. K. (1995). Random decision forests. *Proceedings of 3rd IEEE International Conference on Document Analysis & Recognition*, 278-282.

Hong, R., Wang, M., Gao Y., Tao, D., Li X. & Wu, X. (2014). Image annotation by multiple-instance learning with discriminative feature mapping & selection. *IEEE Transactions on Cybernetics*, 44, 669-680.

Hou, Y. T., Chang, Y., Chen, T., Laih, C. S. & Chen C. M. (2010). Malicious web content detection by machine learning. *Expert Systems with Applications*, 37, 55-60.

Hu, W., Hu, W. & Maybank, S. (2008). AdaBoost-based algorithm for network intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics*, Part B (Cybernetics), 38, 577-583.

Ilyushin, E. & Namiot, D. (2015). On JavaScripts Memory Leaks. *International Journal of Open Information Technologies*, 3, 27-31.

Ikinci, A., Holz, T. & Freiling, F. C. (2008). Monkey-Spider: Detecting malicious websites with low-interaction Honeyclients. *Lecture Notes in Informatics*, Sicherheit, 8, 407-421.

Jain, M, Dua, P. & Lukiw, W. J. (2013). Data adaptive rule-based classification system for Alzheimer classification. *Journal Computer Science System Biology*, 6, 291-297.

James, G., Witten, D., Hastie T. & Tibshirani R. (2013). *An Introduction to Statistical Learning*, Springer: New York, U.S.

Jin, Y. & Sendhoff, B. (2008). Pareto-based multiobjective machine learning: an overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics*, Part C (Applications and Reviews) 38, 397-415.

Jodavi, M., Abadi, M. & Parhizkar, E. (2015). JSObfusDetector: a binary PSO-based one-class classifier ensemble to detect obfuscated JavaScript code. *Proceedings of IEEE International Symposium on Artificial Intelligence & Signal Processing (AISP)*, 322-327.

Kapravelos, A., Shoshitaishvili, Y., Cova, M., Kruegel, C., & Vigna, G. (2013). Revolver: An Automated Approach to the Detection of Evasive Web-based Malware. *USENIX Security Symposium*, 637-652.

Khan, N., Abdullah, J. & Khan, A. S. (2017). Defending malicious script attacks using machine learning classifiers. *Wireless Communications & Mobile Computing*, 17, 1-9.

Kim, B. I., Im, C. T., & Jung, H. C. (2011). Suspicious malicious web site detection with strength analysis of a javascript obfuscation. *International Journal of Advanced Science & Technology*, 26, 19-32.

Kolbitsch, C., Livshits, B., Zorn, B. & Seifert, C. (2012). Rozzle: De-cloaking internet malware. *Proceedings of IEEE Symposium on Security & Privacy (SP)*, 443-457.

Kolbitsch, C., Holz, T., Kruegel, C., & Kirda, E. (2010). Inspector gadget: Automated extraction of proprietary gadgets from malware binaries. *IEEE Symposium on Security and Privacy (SP)*, 29-44)

Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: a review of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering*, 60, 3-24.

Kotov, V. & Masaccio, F. (2013). Anatomy of Exploit Kits. *Proceedings of International Symposium on Engineering Secure Software & Systems*, Springer: Berlin, Heidelberg, Germany, 181-196.

Kruegel, C. (2014). Full system emulation: achieving successful automated dynamic analysis of evasive malware. *Proceedings of BlackHat USA Security Conference*, 1-7.

Kuncheva, L. I., Skurichina, M. & Duin, R. P. W. (2002).An experimental study on diversity for bagging and boosting with linear classifiers. *Information Fusion*, 3, 245–258.

Kuncheva, L. I., Whitaker, C. J., Shipp, C. A., & Duin, R. P. (2003). Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis & Applications*, 6(1), 22-31.

Kolter, J. Z., & Maloof, M. A. (2004,). Learning to detect malicious executables in the wild. *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 470-478.

Livshits, B., Sridharan, M., Smaragdakis, Y., Lhoták, O., Amaral, J. N., Chang, B. Y. E. & Vardoulakis D. (2015). In defense of soundiness: A manifesto. *Communications of the ACM*, 58, 44-46.

Ladha, L., & Deepa, T. (2011). Feature selection methods and algorithms. *International Journal on Computer Science & Engineering*, 3, 1787-1797.

Laskov, P. & Šrndić, N. (2011). Static detection of malicious JavaScript-bearing PDF documents. *Proceedings of 27th ACM Annual Conference on Computer Security Applications*, 373-382.

Lewis, D. D. (1998). Naive (Bayes) at forty: the independence assumption in information retrieval. *Proceedings of European Conference on Machine Learning (ECML)*, 4-15.

Likarish, P., Jung E. & Jo I. (2009). Obfuscated malicious JavaScript detection using classification techniques. *Proceedings of 4th IEEE International Conference on Malicious & Unwanted Software (Malware)*, 47-54.

Li, K., Wang, Z., Ma, C., & Song, S. (2013). SNS privacy protection based on the ELM integration and semi-supervised clustering. *Journal of Software*, 8, 160-167.

Li, N., Yu Y., & Zhou, Z. H. (2012). Diversity regularized ensemble pruning. *Machine Learning and Knowledge Discovery in Databases*, Springer: Berlin, Heidelberg, 7523, 330–345.

Li, S. & Oh, S. (2016). Improving feature selection performance using pairwise pre-evaluation. *BMC Bioinformatics*, 17, 312-325.

Li, X. & Wang, X. (2013). Response adaptive designs with misclassified responses. *Communications in Statistics -Theory & Methods*, 42, 2071-2083.

Li, Z., Zhang K., Xie Y., Yu F. & Wang X. (2012). Knowing your enemy: understanding and detecting malicious advertising. *Proceedings of ACM Conference on Computer & Communications Security*, 674-686.

Liao, H. J., Lin C. H. R., Lin, Y. C. & Tung, K. Y. (2013). Intrusion detection system: a comprehensive review. *Journal of Network & Computer Applications*, 36, 16-24.

Lin, S. W., Lee Z. J., Chen, S. C. & Tseng, T. Y. (2008). Parameter determination of support vector machine and feature selection using simulated annealing approach. *Applied Soft Computing*, 8, 1505-1512.

Ma, J., Saul, L. K., Savage, S. & Voelker, G. M. (2009). Beyond blacklists: learning to detect malicious web sites from suspicious URLs. *Proceedings of 15th ACM International Conference on Knowledge Discovery & Data Mining*, 1245-1254.

Ma, J., Saul, L. K., Savage, S. & Voelker, G. M. (2011). Learning to detect malicious URLs. *ACM Transactions on Intelligent Systems & Technology (TIST)*, 2, 30.1-30.24.

Ma, J., Saul, L. K., Savage, S. & Voelker, G. M. (2009). Identifying suspicious URLs: an application of large-scale online learning. *Proceedings of 26th ACM Annual International Conference on Machine Learning*, 681-688.

Mamun, M. S. I., Rathore, M. A., Lashkari, A. H., Stakhanova, N. & Ghorbani, A. A. (2016). Detecting malicious URLs using lexical analysis. *Proceedings of International Conference on Network & System Security*, 467-482.

Markel, Z. A. (2015). Machine learning based malware detection. *Naval Academy Annapolis MD*, No. USNA-TSPR-440, 1-24.

Minku, L. L. & Yao, X. (2013). An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation. *Proceedings of 9th International Conference on Predictive Models in Software Engineering*, 8, 1–8.

MITRE. (2011). The MITRE HoneyClient Project. http://search.cpan.org/~mitrehc. Accessed on 26/12/2017.

Mohammad, R. M., Thabtah, F. & McCluskey, L. (2014). Intelligent rule-based phishing websites classification. *IET Information Security*, 8, 153-160.

Mohan, V. & Malik, S. (2017). Debunking of common attacks in applications. *International Journal of Current Research*, 9, 47370-47373.

Mohri, M., Rostamizadeh, A. & Talwalkar, A. (2012). *Foundations of Machine Learning*. MIT Press, Cambridge, Massachusetts, U.S. 1-113

Moshchuk, A., Bragin, T., Deville D., Gribble, S. D., & Levy, H. M. (2007). SpyProxy: execution-based detection of malicious web content. *Proceedings of USENIX Security Symposium*, 1-16.

Moskovitch, R., Stopel, D., Feher, C., Nissim, N., & Elovici, Y. (2008). Unknown malcode detection via text categorization and the imbalance problem. *IEEE International Conference on Intelligence and Security Informatics*, 156-161.

Moser, A., Kruegel, C., & Kirda, E. (2007). Limits of static analysis for malware detection. *23rd IEEE Annual Conference on Computer Security Applications,* 421-430.

Nadji, Y., Saxena, P. & Song, D. (2009). Document structure integrity: a robust basis for cross-site scripting defense. *Proceedings of Network & Distributed System Security Symposium (NDSS)*, 1-20.

Nazario, J. (2009). PhoneyC: A virtual client Honeypot. *Proceeding of 2nd USENIX Workshop on Large-Scale Exploits & Emergent Threats, (LEET)* 9, 911-919.

Nunan, A. E., Souto, E., Santos, E. M. D. & Feitosa, E.(2012). Automatic classification of cross-site scripting in web page using document-based & URL-based features. *Proceedings of* IEEE Symposium on Computers & Communications (ISCC), 702-707.

Nagai, E. W., Xiu L. & Chau, D. C. (2009). Application of data mining techniques in customer relationship management: a literature review and classification. *Expert Systems with Applications*, 36, 2592-2602.

Narsky, I. & Porter, F. C. (2013). *Statistical Analysis Techniques in Particle Physics: Fits, Density Estimation and Supervised Learning.* John Wiley & Sons: U.S., 221-227.

Olson, D. L., Shi, Y. & Shi Y. (2007). *Introduction to business data mining.* Englewood Cliffs: McGraw-Hill, Irwin, 10, 2250-2254.

Oyza, I. & Edwin, A. M. (2015). Effectiveness of social media networks as a strategic tool for organizational marketing management. *The Journal of Internet Banking & Commerce*, 21, 1-19.

Ozay, M. & Vural, F. T. Y. (2012). A new fuzzy stacked generalization technique and analysis of its performance. Cornell University, *arXiv*: 1204.0171.

OWASP (2017). *OWASP Top 10 2017. The Ten Most Critical Application Security Risks*. Available at https://www.owasp.org/images/b/b0/OWASP_Top_10_2017_RC2_Final .pdf. (Accessed on 3-Januaray, 2018).

Patil, D. R. & Patil, J. B. (2017). Detection of malicious JavaScript code in webpages. *Indian Journal of Science & Technology*, 10, 1-12.

Pelizzi, R. & Sekar, R. (2012). Protection, usability and improvements in reflected XSS filters. *Proceedings of 7th ACM Symposium on Information, Computer & Communications Security (ASIACCS)*, 5-16.

Pérez, P. M., Filipiak, J. & Sierra, J. M. (2011). LAPSE+ static analysis security software: vulnerabilities detection in java EE applications. *In Future Information Technology*, Springer: Berlin, Heidelberg, Germany, 148-156.

Pham, T. S. & Hoang, T. H. (2016). Machine learning techniques for web intrusion detection-a comparison. *8th IEEE International Conference on Knowledge & Systems Engineering (KSE)*, 291-297.

Politz, J. G., Eliopoulos S., Guha, A. & Krishnamurthi, S. (2015). *ADsafety: type-based verification of JavaScript Sandboxing*. Cornell University, *ArXiv:* 1506.07813.

Polikar, R. (2012). Ensemble learning. *In Ensemble machine learning*, Springer, Boston, MA, 1-34.

Potashnik, D., Fledel, Y., Moskovitch, R. & Elovici, Y. (2011). Monitoring, analysis, and filtering system for purifying network traffic of known and unknown malicious content. *Security & Communication Networks*, 4, 947-965.

Provos, N., McNamee, D., Mavrommatis, P., Wang K. & Modadugu, N. (2007). The ghost in the browser: analysis of -based malware. *Proceedings of I st Conference on Hot Topics in Understanding Botnets (HotBots)*, 1-9.

Provos, N. et al. (2012) *Safe Browsing: Protecting Web Users for 5 Years and Counting*. Available at: http://googleonlinesecurity.blogspot.it/2012/06/safe-browsing-protecting-users-for.html, Accessed on 14-Januarary, 2017.

Qassrawi, M. T. & Zhang, H. (2011). Detecting malicious web servers with Honeyclients. *Journal of Networks (JNW)*, 6, 145-152.

Rajab, M., Ballard, L., Jagpal, N., Mavrommatis, P., Nojiri D., Provos, N. & Schmidt, L. (2011). Trends in circumventing web-malware detection. *Google Technical Report*, 1-12.

Rajeswari, B. (2017). A study of techniques and tools used in malware analysis. *World Journal of Engineering Research & Technology*, 3, 133 -140.

Ratanaworabhan, P., Livshits, B. & Zorn, B. (2009). Nozzle: A defense against heap-spraying code injection attacks. *Proceedings of USENIX Security Symposium*, 169-186.

Rauschmayer, A. (2012).*The Past, Present, and Future of JavaScript*. O'Reilly Media, Incorporation, 1-55.

Rao, V. & Hande, K. (2017). A comparative study of static, dynamic and hybrid analysis techniques for android malware detection. *International Journal of Engineering Development & Research*, 5, 1433-1436.

Reis, E. R., Castro, P. A., Sichman, J. S. & Gomes P. M. E.(2016). Enhancing classification accuracy through feature selection methods. *Proceedings of 13th SBC National Encounter of Artificial & Computational Intelligence (ENIAC)*, 1-12.

Rieck, K., Krueger, T. & Dewald, A. (2010). Cujo: efficient detection and prevention of drive-by-download attacks. *Proceedings of 26th ACM Annual Computer Security Applications Conference*, 31-39.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, 33, 1–39.

Santos, I., Penya, Y. K., Devesa, J., & Bringas, P. G. (2009). N-grams-based File Signatures for Malware Detection. *11th International Conference on Enterprise Information Systems (ICEIS)* 317-320.

Schütt, K., Kloft, M., Bikadorov, A. & Rieck, K. (2012). Early detection of malicious behaviour in JavaScript code. *Proceedings of 5th ACM Workshop on Security & Artificial Intelligence*, 15-24.

Schultz, M. G., Eskin, E., Zadok, F., & Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. *Proceedings IEEE Symposium on Security and Privacy*, 38-49.

Seifert, C., Welch, I. & Komisarczuk, P. (2007). HoneyC-the low-interaction client honeypot. *Proceedings of 14th New Zealand Computer Science Research Student Conference (*NZCSRSC*)*, 1-9.

Seifert, C., Welch, I. & Komisarczuk, P. (2008). Identification of malicious web page with static heuristics. *Proceedings of IEEE Conference on Telecommunication Networks & Applications (ATNAC)*, 91-96.

Selim, H., Tayeb, S., Kim, Y., Zhan, J., & Pirouz, M. (2016). Vulnerability analysis of iframe attacks on websites. *Proceedings of 3rd AMC Multidisciplinary International Social Networks Conference on Social Informatics (MISNC), Data Science*, 45, 1-6

Seni, G. & Elder, J. F. (2010). Ensemble methods in data mining: improving accuracy through combining predictions. *Synthesis Lectures on Data Mining & Knowledge Discovery*, 2, 1-126.

Severance, C. R. (2012). JavaScript: Designing a language in 10 days. *IEEE Computer* 45, 7-8.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss Y. (2012). Andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38, 161-190.

Shabtai, A., Moskovitch, R., Elovici, Y., & Glezer C. (2009). Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey. *Information Security Technical Report*, 14, 16-29.

Stefan, D., Yang, E. Z., Marchenko, P., Russo, A., Herman, D., Karp, B. & Mazieres, D. (2014). Protecting users by confining JavaScript with COWL. *Proceedings of 11th USENIX Symposium on Operating Systems Design & Implementation (OSDI)*, 131-146.

Stringhini, G., Kruegel, C. & Vigna, G. (2013). Shady paths: Leveraging surfing crowds to detect malicious web pages. *Proceedings of ACM SIGSAC Conference on Computer & Communications Security*, 133-144.

Sirageldin, A., Baharudin, B. B. & Jung, L. T. (2014). Malicious Webpage Detection: A Machine Learning Approach. *Advances in Computer Science and its Applications*, Springer: Berlin, Heidelberg, Germany, 217-224.

Stock, B., Lekies, S., Mueller, T., Spiegel, P., & Johns, M. (2014). Precise client-side protection against DOM-based cross-site scripting. *Proceedings of 23rd USENIX Security Symposium*, 655-670.

Symantec (2017). Internet Security Threat Report - April 2017. *ISTR* - Volume 22. [Online] Symantec. Available at: https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf (Accessed on 2-November, 2017).

Swingler, K. (1996). *Applying neural networks: a practical guide*. Morgan Kaufmann., California, U.S.

Thomas, K., Grier C., Ma, J., Paxson V. & Song, D. (2011). Design and evaluation of a real-time URL spam filtering service. *Proceedings of IEEE Symposium on Security & Privacy (SP)*, 447-462.

Ting, K. M. & Witten, I. H. (1999). Issues in stacked generalization. *Journal of Artificial Intelligence Research (JAIR)*, 10, 271-289.

Tumer, K. & Ghosh, J. (1996). Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8, 385–404.

Tumer, K. & Ghosh, J. (1996). Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition*, 29, 341-348.

Vapnik, V. & A. Lerner. (1963). Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 774–780.

Wang, J., Xue Y., Liu, Y. & Tan, T. H. (2015). JSDC: a hybrid approach for JavaScript malware detection and classification. *Proceedings of 10th ACM Symposium on Information, Computer & Communication Security*, 109-120.

Wang, H. J., Fan, X., Howell, J. & Jackson, C., (2007). Protection and communication abstractions for web browsers in MashupOS. *ACM SIGOPS Operating Systems Review*, 41, 1-16.

Wang, Y. M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen S. & King S. (2006). Automated web patrol with strider HoneyMonkeys: finding websites that exploit

browser vulnerabilities. *Proceedings of Symposium on Network & Distributed System Security (NDSS)*, 35-49.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5, 241-259.

Wolpert, D. H. & Macready, W. G. (1999). An efficient method to estimate Bagging's generalization error. *Machine Learning Journal*, 35, 41-55.

Xiao, W., Sun, J., Chen, H. & Xu, X. (2014). Preventing client-side XSS with rewrite based dynamic information flow. *Proceedings of 6th International Symposium on Parallel Architecture Algorithms Program*, 238–243.

Xiong, C., Li P., Zhang, P., Liu, Q. & Tan, J. (2015). MIRD: Trigram-based malicious URL detection implanted with random domain name recognition. *Proceedings of International Conference on Applications & Techniques in Information Security*, 303-314.

Xing, E. P., Jordan, M. I. & Karp, R. M. (2001). Feature selection for high-dimensional genomic microarray data. *Proceedings of International Conference on Machine Learning. (ICML)*, 601-608.

Xu, L., Zhan, Z., Xu, S., & Ye, K. (2013). Cross-layer detection of malicious websites. *Proceedings of 3rd ACM Conference on Data & Application Security & Privacy*, 141-152.

Xu, W., Zhang, F. & Zhu, S. (2013). JStill: mostly static detection of obfuscated malicious JavaScript code. *Proceedings of 3rd ACM Conference on Data & Application Security & Privacy*, 117-128.

Yang, T., Cui, H., & Niu, S. (2017). Dynamic loading vulnerability detection for android applications through ensemble learning. *Chinese Journal of Electronics*, 26, 960-965.

Yang, Y. & Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. *Proceedings of International Conference on Machine Learning (ICML)*, 412-420.

Zarras, A., Kapravelos, A., Stringhini, G., Holz, T., Kruegel, C. & Vigna, G. (2014). The dark alleys of Madison Avenue: understanding malicious advertisements. *Proceedings of ACM Conference on Internet Measurement*, 373-380.

Zhang, B. Y., Yin, J. P., Wang, S. L., & Yan, X. A. (2014). Research on virus detection technique based on ensemble neural network and SVM. *Neurocomputing*, 137, 24-33.

APPENDIX

List of Publications

1. Khan, N, Abdullah, J., & Khan, A. S. (2017) Defending Malicious Script Attacks Using Machine Learning Classifiers, *Wireless Communications and Mobile Computing*, doi:10.1155/2017/5360472, Article ID 5360472, 9-Pages

2. Khan, N., Abdullah, J., & Khan, A. S. (2017). A Dynamic Method of Detecting Malicious Scripts Using Classifiers. *Advanced Science Letters*, 23(6), 5352-5355.

3. Khan, N., Abdullah, J., & Khan, A. S. (2015). Towards vulnerability prevention model for web browser using interceptor approach. *IEEE 9th International Conference on IT in Asia (CITA)*, Kuching, Sarawak, Malaysia 1-5.

4. Khan, N., Abdullah, J., & Khan, A. S. (2017). A Taxonomy Study of XSS Vulnerabilities. *Asian Journal of Information Technology*, 16(2), 169-177.

5. Khan, N., Abdullah, J., & Khan, A. S. (2017) A Framework for Vulnerability Prevention Model for Web Browsers Using Interceptor Approach. *Indian Journal of Science & Technology*, 11(16), 1-12