MOBILE MALWARE ANOMALY-BASED DETECTION SYSTEMS USING STATIC ANALYSIS FEATURES

AHMAD FIRDAUS BIN ZAINAL ABIDIN

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

MOBILE MALWARE ANOMALY-BASED DETECTION SYSTEMS USING STATIC ANALYSIS FEATURES

AHMAD FIRDAUS BIN ZAINAL ABIDIN

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Ahmad Firdaus Bin Zainal Abidin

Matric No: WHA130060

Name of Degree: Degree of Philosophy

Title of Thesis: MOBILE MALWARE ANOMALY-BASED DETECTION SYSTEMS USING STATIC ANALYSIS FEATURES

Field of Study: Intrusion Detection System

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyrighted work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyrighted work;
(5) I hereby assign all and every right in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

MOBILE MALWARE ANOMALY-BASED DETECTION SYSTEMS USING STATIC ANALYSIS FEATURES

ABSTRACT

Presently, the rise of demand for Android gadget motivates the unscrupulous author to develop malware to compromise mobile devices for malicious and private purposes. The categories of mobile malware types are root exploit, botnet, and Trojan. Consequently, in order to classify an application either malware or benign, security practitioners conduct two types of analysis, namely dynamic and static. Dynamic analysis classifies an application as malware by executing it and monitors the behavior. However, it demands high computing requirements and monitors in a limited range of time. On the other hand, static analysis reverses engineer an application and examine overall code thoroughly, therefore further capable of examining the whole structure of the application. Furthermore, static analysis consumes low resources (for instance, CPU, memory, storage) and less time processing. As static analysis concentrates on the code, security practitioners face challenges to select the best features among thousand lines of it. Although they suggest several features, however, there are still provides many features available to be explored. Furthermore, less attention has been given to root exploit features specifically. It is one of the critical malware which compromises operating system kernel to obtain root privileges. When the attackers obtain the privileges, they are able to bypass security mechanisms and install other possible types of malware to the devices. Moreover, in order to achieve an efficient malware prediction in machine learning, it needs features in a minimal amount to enhance accuracy with fewer data, less time processing and reduces model complexity. Therefore, to achieve the aim of finding the best and minimal features to detect malware with root exploit, this study adopts bio-inspired Genetic Search (GS), conveys the range

of repeated features in similar application, and investigates root exploit to gain the best features to predict unknown malware using machine learning. The features categories involved in all these experiments are the permission, directory path, code-based, system command, and telephony. In detecting root exploit, the category involved is the novel features called Android Debug Bridge (ADB). By obtaining the best features derived from these experiments, this study applies it in machine learning to predict unknown malware. To demonstrate the results, this experiment evaluated six benchmarks (for instance, accuracy, True Positive Rate (TPR), False Positive Rate (FPR), recall, precision, and f-measure) to test the prediction and performance. From the outstanding results being collected, a website was established to validate the unique static features with machine learning mechanism to investigate its efficiency and practicality. Through the outcomes assembled, this research has verified that the unique static features capable of predicting unknown malware together with root exploit. The contributions of this study were investigated, selected, proposed, designed and evaluated the best features in detecting malware by using static analysis.

Keywords: Static analysis, Android, feature selection, root exploit, machine learning

SISTEM PENGESANAN PERISIAN PEROSAK BERASASKAN ANOMALI MENGGUNAKAN CIRI-CIRI ANALISIS STATIK

ABSTRAK

Pada masa ini, permintaan tinggi pada Android mendorong penyerang tidak beretika membangunkan perisian perosak yang berniat jahat untuk merosakkan peranti mudah alih dan tujuan persendirian yang lain. Jenis-jenis perisian perosak ialah pengeksploitasi kawalan, botnet, dan Trojan. Untuk mengesan semua jenis perisian perosak ini, para pengkaji menjalankan dua jenis analisis, iaitu dinamik dan statik. Analisis dinamik mengklasifikasikan perisian perosak dengan melaksanakan dan memantau tingkah lakunya. Walaubagaimanapun, analisis ini memerlukan kadar kiraan yang tinggi dan ia hanya memantau dalam masa lingkungan yang terhad. Sebaliknya, analisis jenis statik pula, ia mengubah perisian tersebut kepada kod asal dan berupaya untuk memeriksa keseluruhan kod dengan teliti, dan seterusnya mampu mengkaji keseluruhan struktur perisian. Tambahan lagi, analisis statik memerlukan sumber yang rendah (iaitu CPU, memori simpanan) dan masa pemprosesan yang kurang. Oleh itu, para pengkaji menghadapi cabaran memilih ciri yang terbaik antara ribuan garisan kod. Walaupun mereka mencadangkan beberapa ciri, bagaimanapun masih ada banyak lagi ciri untuk diterokai. Tambahan pula, kurang tumpuan diberikan dalam mencari ciri untuk pengeksploitasi kawalan secara spesifik. Ia adalah salah satu perisian perosak kritikal yang menceroboh sistem pengoperasian kernel untuk mendapatkan kawalan sepenuhnya. Apabila penyerang mendapat kawalan, mereka mampu melangkau mekanisme keselamatan dan memasang pelbagai jenis perisian perosak yang lain. Dengan itu, untuk mencapai ramalan pengesanan semua jenis perisian perosak (termasuk pengeksploitasi kawalan) yang cekap, jumlah ciri yang terbaik dalam kadar minima diperlukan untuk meningkatkan ketepatan ramalan walaupun dengan data yang

kurang, kurang masa pemprosesan dan mengurangkan kerumitan model ramalan. Oleh itu, untuk mencapai tujuan mencari ciri terbaik dengan kadar yang minimum, kajian ini menyiasat ciri menggunakan kaedah carian genetik (GS), mengambil kira ciri yang berulang kali dalam perisian yang sama dan ciri dalam perisian pengeksploitasi kawalan. Kemudian, ciri ini digunakan dalam ramalan mesin pembelajaran bio-inspirasi. Antara kategori ciri yang terlibat dalam eksperimen-eksperimen ini ialah kebenaran perisian, asas kod, laluan direktori, perintah sistem dan telefoni. Dalam mengesan pengekspolitasi kawalan, antara kategori ciri novel yang terlibat adalah Android Debug Bridge (ADB). Dengan mendapatkan ciri yang terbaik hasil daripada eksperimen-eksperimen ini, kajian ini memasukkan ia dalam mesin perisian ramalan untuk mengenalpasti perisian perosak yang masih belum dijumpai. Dalam mendemonstrasikan keputusan ekperimen, kajian ini menilai enam penanda aras (iaitu ketepatan, kadar positif benar (TPR), kadar positif palsu (FPR), penarikan balik, ketepatan dan ukuran-f) dalam menguji prestasi ramalan. Dari keputusan cemerlang dikumpulkan hasil dari menggunakan ciri yang dicadangkan, sebuah laman web dibangunkan dengan jentera mekanisme ramalan untuk mengesan perisian perosak. Melalui hasil yang dikumpulkan, penyelidikan ini telah mengesahkan bahawa ciri statik mampu mengesan jenis perisian perosak yang lain yang belum dijumpai lagi termasuk pengeksploitasi kawalan. Sumbangan utama kajian ini ialah menyiasat, memilih, mencadangkan, dan menilai ciri yang terbaik dalam mengesan perisian perosak dengan menggunakan analisis statik.

Kata kunci: Analisis statik, Android, pemilihan ciri-ciri, pengeksploitasi akar, pembelajaran mesin

ACKNOWLEDGEMENTS

A very special gratitude to my respectful supervisor, Dr. Nor Badrul Anuar Juma'at, who have provided me the uncountable knowledge, assistance, remarks, patience, and engagement for my learning period in the University of Malaya. His guidance helped me in all the time of my research, publication, and writing of this thesis.

I would like to express my heartfelt gratitude to my beloved mother, Noraini Md Zin for sacrificed her time raising me with endless love from a child to an adult. Without her continuous support and encouragement, I never would have been able to achieve my goals.

Also, I dedicated this study to my father, Zainal Abidin Baharudin who gave me motivations when I'm in depression situations. His showed me the true meaning of life and taught me to work hard and assist me through critical conditions.

A deep gratitude to my wife, Arniwaty Abdullah with our incoming third child. She has stopped her master education in the University of Malaya to give way for my study and understands me. Her dedication, devotion, and perseverance made my study goes this far.

My grateful thanks to my mother in law, Rusnah Abdul Hamid. She has selflessly given more time to take care of the kids (Ahmad Yusuf and Ainul Mardhiyyah) with patient when I'm not around. Her help and support to me are greatly appreciated indeed.

Lastly, thank you to everyone who assisted me in this study, Mohd Faizal Ab Razak, Hazim Hanif, Firdaus Afifi Md Isa, Kayode Adewole Sakariyyah, Ali Feizollah, and Ibrahim Abaker Targio Hashem and to all my friends in the lab. I wish them happiness and success.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

ADB	:	Android Debug Bridge
APK	:	Application Package
API	:	Application Programming Interfaces
ARFF	:	Attribute-Relation File Format
CPU	:	Central Processing Unit
CSV	:	Comma Separated Values
DT	:	Decision Tree
XML	:	Extensible Markup Language
FP	:	False Positive
FPR	:	False Positive Rate
FT	:	Functional Tree
GUI	:	Graphical User Interface
HTTP	:	Hypertext Transfer Protocol
IDS	:	Intrusion Detection System
JAR	:	Java Archive
KNN	:	K-Nearest Neighbors
ML	:	Machine Learning
MD5	:	Message Digest 5
MLP	:	Multilayer Perceptron
NB	:	Naïve Bayes
PC	:	Personal Computer
RBFN	:	Radial Basis Function Network

RF	:	Random Forest
SHA1	:	Secure Hash Algorithm 1
SVM	:	Support Vector Machine
TN	:	True Negative
TPR	:	True Positive Rate
URL	:	Uniform Resource Locator
VP	:	Voted Perceptron

LIST OF APPENDICES

CHAPTER 1: INTRODUCTION

In today's lifestyle, most people, whether young or old, regularly utilized mobile gadgets such as the smart phone. This is because people are using their mobile devices as their main gadgets to manage their daily activities (e.g. connecting, communicating, health management, synchronous data transfer, family communications, money transactions, business interactions and the world's updates). Due to the introduction of high technology mobile devices, the lives of human being throughout the world have become more convenient and communication among individuals in any part of the world have also become less effortless and more accessible (Union, 2016). Given this technology and ease of communication, manufacturers are producing even more mobile devices that are equipped with various produced many mobile devices with various types of Operating System (OS), such as Android, iOS, and Windows. As the world community advances in technology, they also become more dependent on mobile devices for various executions and duties. This has led malware creators to jump on the bandwagon by seizing this opportunity to develop various malware types that help them to execute malicious activities that harm mobile users and their mobile devices. Mobile device malware has been growing rapidly in scale and mobile users are being exploited by malware creators in various ways.

In the context of computer technology, malware is a kind of software designed by unscrupulous computer programmers whose aim to perform various diverse malicious actions on the mobile devices without the consent of the users. These activities are to the benefit of the malware creators. Some of the malware activities created include locating a user of victim's location, obtaining the victim's personal data (i.e., images, phone contacts, and messages), and compromising the Android OS kernel in the mobile

devices so as to gain power of root users. The various types of malware currently available in the market include root exploit, botnet, and Trojan.

In the second quarter of the year 2016, McAfee (2016) identified several new mobile malware samples. In September, the third quarter of the year 2016, Kaspersky Lab (Kasperksy, 2016) reported that the rate of users encountering mobile banking Trojan is almost eight times greater than it was in June, in the same year. Likewise, the number of malware for Android was also noted to have increased more than other mobile OSes.

Symantec discovered new Android malware families each month from February 2014 to January 2015 (Symantec, 2015). Similarly, Sophos Mobile Security also revealed that 610,389 of new Android malware samples had been detected outside of the Google Play market in the first 6 months of 2015 (Komili, 2016). In line with this, Google Play also observed that a new Android malware family called as Dresscode has been infecting between 500,000 and 2,000,000 users. This malware had automatically installed itself between 100,000 and 500,000 times in 2016 (Cimpanu, 2016).

Following this, security analyst also detected one specific type of malware called root exploit, known as the malware which modifies the kernel in an Android OS. Its aim is to gain super-user privileges. When malware gains root and these privileges increase, the attackers easily installed other malware types, such as botnets, worms, or Trojans into the OS. When this event occurs, the attackers capable of evading detection by modifying the OS code, running their malicious activities stealthily, and bypassing permission (Bickford *et al.*, 2010; Ma & Sharbaf, 2013; A. Schmidt *et al.*, 2009). Such types of malware able to take over the root privileges and performs malicious actions stealthily without the victim's knowledge (Wei *et al.*, 2015).

In June 2016, a root exploit called Godless was discovered (V. Zhang, 2016). It contains multiple types of malware exploits to be used later by the attacker and during the news in writing, it has affected almost 90% of devices that run on Android 5.1, the Lollipop version. Another recent discovery of root exploit known as Dirty COW was detected in October 2016 (Arghire, 2016). Researchers say that malware has the capability to compromise an entire mobile device system which runs a Linux kernel that is higher than 2.6.22. Due to this widespread malice, there is a need to study malware and to provide a detection system that able to identify all types of malware including the root exploit.

1.1 Malware detection

There are two types of malware detection system: signature-based (misuse-based) and anomaly-based. The signature-based approach is a traditional approach which utilizes antivirus that deployed in mobile devices. Over time, this approach has become less efficient because the antivirus signature needs to regularly update its database as a way to facilitating it to detect new malware variant (Nissim *et al.*, 2014) (Suarez-Tangil *et al.*, 2014). In contrast, the anomaly-based approach detects unknown malware by considering at the training set of feature rules which have been set forth by security analysts. This approach is able to intelligently measure the system features. Through the significant deviations noted from the extracted features, the anomaly-based approach is able to successfully detect it as unknown malware (Yerima, Sezer, & McWilliams, 2014). With this advantage, this thesis conducted malware analysis by adopting the anomaly-based detection by using machine learning (i.e. part of Artificial intelligence knowledge).

In the malware detection system, there are two types of malware analysis: dynamic and static. Dynamic analysis is a procedure that discovers malware by performing the

applications and monitoring the behavior. The necessity to monitor the behavior requires high specifications in both memory and CPU in supporting the applications to run. Dynamic analysis monitors at a certain range of time depending on the investigation period. Hence, it could possibly miss the malware activities that are beyond the time range of the investigation (i.e. attacker triggers malware actions in a certain time or whenever the attacker decides) (Feizollah, Anuar, *et al.*, 2013)(Yerima *et al.*, 2015). In contrast, static analysis is a procedure that inspects the applications codes without running it. The advantage of the static analysis is that it requires low resources (e.g. memory and CPU) and its processing rate is fast (Chess & McGraw, 2004). In this regard, it covers all possible activities without any time range.

1.2 Research motivations

Researchers conducted a study for practical or scientific purposes. In the same way, this study was also motivated by a number of reasons which are classified accordingly:

a) **Statistics of Android malware:** Of late, security analysts have discovered various types of hidden malware in more than 104 applications in the Google Play store. These applications have been downloaded by users over 3.2 million times and had caused numerous problems to user's mobile devices (Russon, 2016).

b) **Continuously conduct research:** As the owner of Android itself, Google is constantly looking for means to improve its systems and it is also constantly encouraging others to improve its systems by detect malware thereby, reducing the huge violations affecting mobile users (BBC, 2016). In this regard, security practitioners from both the industry and academia need to make their contributions by continuously conducting investigations pertaining to mobile malware.

c) **Research on Android:** Among all the mobile device OS, Android dominates the smartphone market, with a share worth of $366 billion (Thomas, 2015). Android also has the highest worldwide market share of 88% as noted in the third quarter of

2016 (Forni & Van der Meulen, 2016). Android prices are available on a wide range, starting from as low as $50 (eBay, 2016) and such low prices allow young children and teenagers to buy and own an Android mobile, as a part of their daily lifestyle. Nonetheless, many Android users including adults, teenagers and young children are exposed to malware attacks which affect their devices and ruin their personal data and information.

d) **The increases in root exploits:** The increasing number of root exploits is the evidence that malware creators and homebrew community attackers (smartphone users who break the OS kernel to obtain a customized version of an OS) are putting mobile users at risk. When a new version of the OS is released, malware creators tend to develop their own root exploits; alternatively, they wait for the homebrew community attackers to determine ways to break the OS (Felt *et al.*, 2011).

e) **Minimal features:** Selecting the relevant features in minimal amount for malware detection is crucial because this minimal features are able to reduce the runtime of the machine learning approach (Crussell *et al.*, 2012). It also removes noisy and irrelevant data that enhances the detection (Jensen & Shen, 2008)(Sabry *et al.*, 2015). According to studies, five out of 41 features received better detection when viewed through the experiments conducted. This is an important factor that motivated this thesis.

Overall, the five factors highlighted above are the main factors that have motivated this research to be conducted. For this effort to materialize, the problem statement is further elaborated.

1.3 Problem statement

In malware analysis, features reside in applications which consist of a thousand lines of code. Selecting the relevant features through those lines is rigorous because security

analysts need to inspect both the malware and benign applications in order to distinguish the difference between them before uncovering the characters or elements frequently used by the malware and benign applications. The issue of this problem increases when deals with a thousand applications particularly during the data collection phase.

The problem becomes bigger when the anomaly-based detection needs only minimal features for it to be better in classifying the applications as malware or benign (Feizollah *et al.* 2015). This is an important issue because minimal features have certain advantages such as reducing running time of the process (Crussell *et al.* 2012) and removing noisy and irrelevant data (Jensen & Shen 2008). The other advantage of the minimal feature is that the detection system takes lesser time to process when in a real environment (Chess & McGraw, 2004). As such, the experiments done in a simulation application would consume less processing time because the security analysts have already reversed engineering and extracted the features beforehand. Subsequently, the simulation only processes the extracted features and presents the results. In actual practical detection, once the user has uploaded the application, the detection system only needs to reverse the engineering, scan, identify application and extract features in the application. Given that these tasks are many; the system may also confront certain problems such as when the features exceed a suitable amount (e.g. 30, 40, 100, 200). Such problems include extracting the wrong features by mistake, facing a complicated and unwanted character/strings or the detection system may stop the process due to too many features. All these add on to the possibility of errors. In that regard, the detection system should only require minimal features to execute an easier and cleaner detection process.

As features consist of numerous lines of code, security analysts have to categorize these features into multiple categories such as permissions, string, function call, and API for easy identification. Given this situation, conducting an investigation on only one category of features would be insufficient and inadequate because precise malware detection in static analysis requires multiple categories of features to be examined.

Previous studies (Karim, Salleh, Khan, *et al.*, 2016; Seo *et al.*, 2014; Sheen *et al.*, 2015; Yerima *et al.*, 2015; Yerima, Sezer, & Muttik, 2014) had depended on only one file, for example AndroidManifest.xml for the features. Due to this, the detection system may achieve less accuracy. In the case where the application includes zero permission in the AndroidManifest.xml and where the features extracted are too few or if the application is damaged or unable to open the AndroidManifest.xml, the detection system then becomes less accurate. Moreover, the risk of using too few features causes the detection to mistakenly classify the malware application as benign. Given this situation, it is imperative to consider all files that exist in an application. In summary, there is a need to conduct research that looks for the relevant but minimal features which reside in numerous lines of application code. It is also important to consider features that exist among multiple categories of features in all files located in the applications to ensure detection of unknown malware in static analysis.

1.4 Aim and Objectives

The aim of this study, hereby also thesis, is to develop an intelligent anomaly-based detection system by using static analysis. In that regard, the main objectives of this thesis are as follows:

a) To review the domain of Android static analysis and its key issues
b) To establish the need for an intelligent intrusion detection system by using static analysis as well as methods to identify the best features in minimal amount

c) To design and develop a novel framework by applying the proposed features in the intelligent intrusion detection system

d) To evaluate the proposed features to detect unknown malware as well as the features specifically in root exploit in terms of accuracy and performance

1.5 Research methodologies

This study performed different methodologies in three experiments. The 1st experiment adopted Genetic Search (GS) in the features extraction phase so as to automatically search for the best features in detecting malware. The 2nd experiment nominated the best features by considering the utilization of similar features in the same application and applying this in range algorithm. The 3rd experiment investigated the best features of the malware that focusing on root exploit only.

The subsequent steps involved in these experiments include collecting the data, extracting the features and using these features in the intelligent prediction machine learning mechanism. Figure 1.1 depicts the general methodology in the experiments conducted.

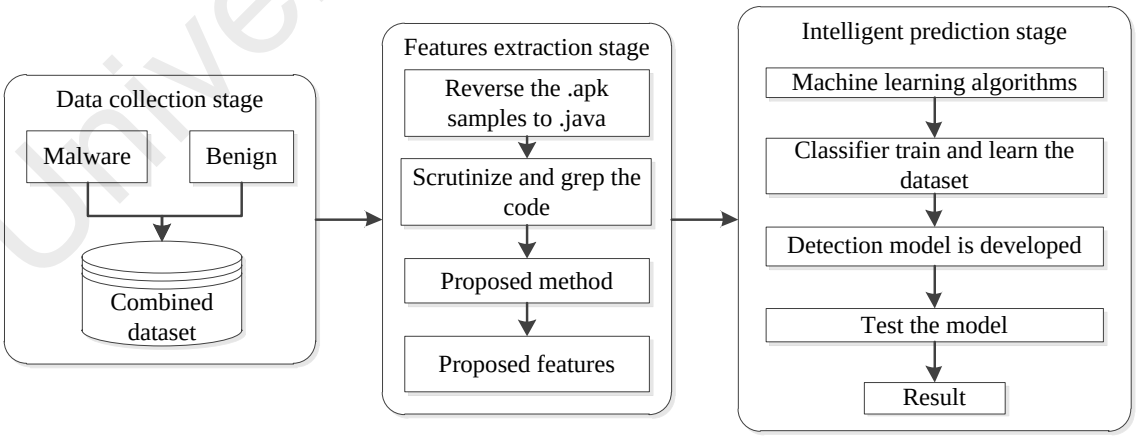


Figure 1.1: General methodologies

a) **Data collection stage:** The first step involves collecting the Android application package in the .apk extension. Since the aim of this study is to differentiate the malware and benign classes, it is important to have both types of applications for the experiment.

b) **Feature extraction stage:** The second step performs the basic static analysis approach which is also known as reverse engineering. This is to retrieve the Java code derived from the Android application package. Once the experiment obtains the code, the next step adopts a unique method to obtain the exclusive features from the multiple categories (i.e. permission, directory path, ADB, code-based, system command and telephony).

c) **Intelligent prediction stage:** The third step implements the anomaly-based detection in machine learning. It is used to conduct the intelligent prediction in detecting whether the classes of the applications are malware or benign.

However, these experiments are performed differently in the dataset, in the features extraction method and in the classification. Table 1.1 lists these differences in methodologies.

Table 1.1: Summary of methodologies

Experiment	Data collection stage		Features extraction stage		Classifier categories for intelligent prediction stage
	Benign	Malware	Method	Categories involved	
1	Google Play store	Drebin	Genetic Search (GS)	Permission, system command, directory path, and code-based	Bio-inspired, tree and bayes
2	Androzoo	Drebin	Range of similar features in same application, Information Gain (IG) and refer to official list	Permission, directory path, and telephony	Bio-inspired
3	Google Play store	Malgenome	Frequency investigation and IG	System command (with novel ADB), directory path, and code-based	Bio-inspired, tree and bayes

As can be seen, the 1st experiment adopts the Genetic Search (GS) to automatically select the features in a genetic way. Thus, this is the only experiment that involved four categories of features (i.e. permission, system command, directory path and code-based) as compared to other experiments. The purpose is to enable the GS to select the features based on a wide choice of features.

The 2nd experiment uses the novel range of repeated features in a similar application which aims to select the exclusive features. This is the only experiment that refers to the official list as a main source in order to avoid any missing features in the dataset of each category. In addition, this experiment also explores multiple types of bio-inspired machine learning algorithm in order to discover the potential of each classifier.

Unlike the other experiments, the 3rd experiment utilizes Malgenome as the malware dataset. The reason is because it provides various root exploit families more than Drebin. This experiment also explores the novel ADB features noted in the system command category of root exploit detection.

Finally, this study utilizes the proposed method to gain the best features noted in each experiment and implementing these in the web based environment prototypes to detect unknown malware including root exploit. This study develops the prototypes of malware detection system to test the accuracy and performance of these best features in a real environment.

1.6 Thesis outline

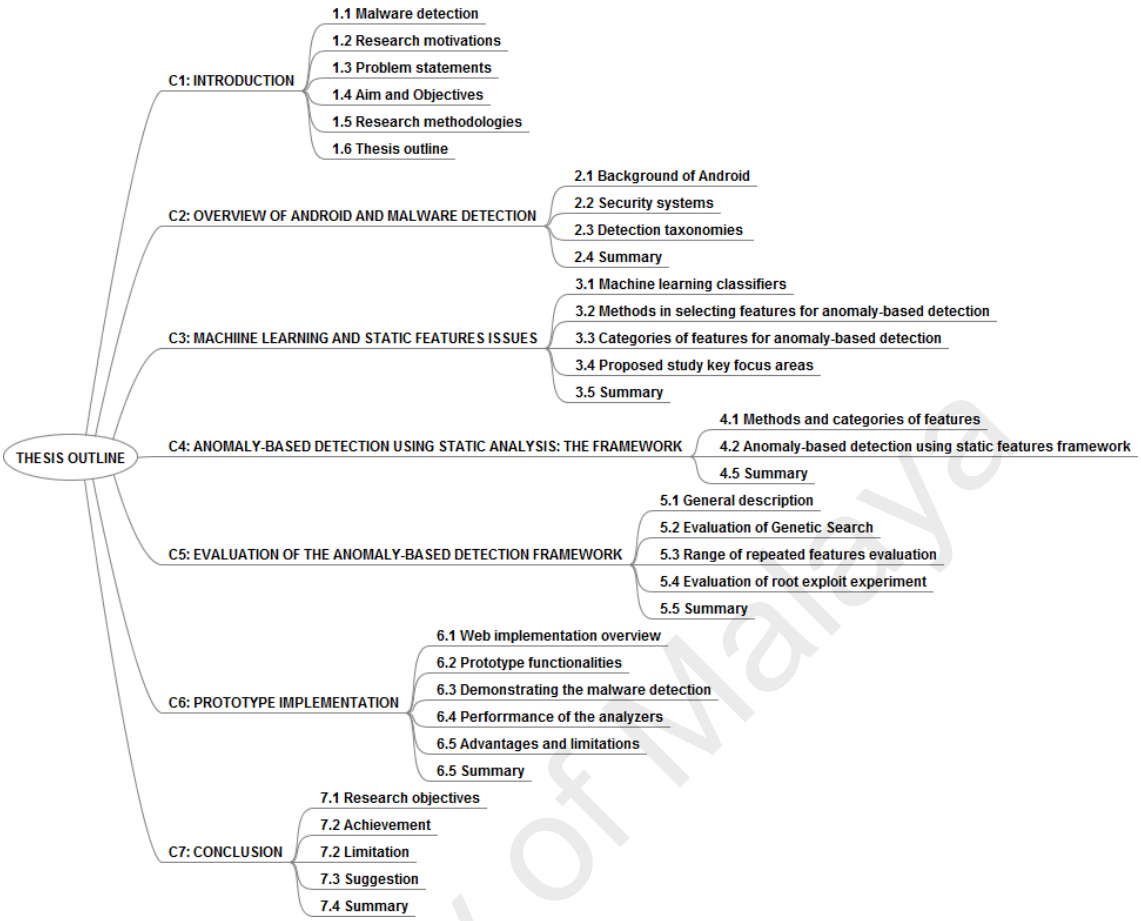


Figure 1.2: Thesis outline

As the thesis outline projects, there are seven chapters in this thesis. Chapter 1 introduces the motivations of this thesis. It outlines the problem statement leading to the study. This chapter also highlights the aim and objectives of the study before continuing with the methodologies of the research. It ends with the outline of the thesis provided in a graphical tree.

Chapter 2 introduces the background of Android and the security system it provides in detecting malware including Intrusion Detection System (IDS), Intrusion Prevention System (IPS) and Intrusion Response System (IPS). Specifically it also reviews their detection capabilities. A comprehensive taxonomy and the state-of-the-art IDS are assessed and presented, covering information concerning signature-based, anomaly-

based, dynamic and static analysis concepts as well as their differences, advantages and limitations.

Chapter 3 focuses on existing static analysis studies along with the machine learning classifiers. It then continues with the review of the relevant theories for the methods in selecting the static features and the categories of features. This chapter also highlights the categories of features involved in this study by summing up all the advantages of the studies. Finally, it also discusses how these can be combined to produce an effective anomaly-based detection system.

Chapter 4 presents the main contribution of this thesis: a novel framework with alternative approaches which can be used to select the best static features that are suitable for the machine learning detection system. In presenting the framework, this chapter begins by introducing the main rationale behind the framework as well as its operational characteristics. It also introduces the Genetic Search (GS), the range of repeated features and the Android Debug Bridge (ADB) used in the framework.

Chapter 5 extends the study by conducting multiple experiments to validate and evaluate the proposed framework. In order to demonstrate the progress of the results, the evaluation will comprise three experiments: a) 1st experiment is the evaluation of the features selected genetically from GS; b) 2nd experiment is the evaluation of the features derived from the investigation of the repeated features obtain from similar applications; c) 3rd experiment investigates and evaluates the root exploit features with the aid of the Android Debug Bridge (ADB). The evaluation consists of six benchmarks (i.e. accuracy, True Positive Rate (TPR), False Positive Rate (FPR), recall, precision, and f-measure). This chapter also provides an in-depth discussion of the implications of applying the proposed framework in practice whilst highlighting the advantages and limitations at the same time.

Chapter 6 presents the website development as a prototype which practically utilizes the proposed features to detect unknown malware including features which are specific to root exploit. It provides an overview of the system development consisting of upload and reverse engineering the application, identifying and extracting the proposed features and the machine learning prediction. In addition, this chapter uses different samples of malware extracted from a reliable source to test the efficiency of the prediction.

Chapter 7 provides the main conclusions derived from this study; it highlights the advantages and limitations of the study as well as suggestions for future research efforts.

CHAPTER 2: OVERVIEW OF ANDROID AND MALWARE DETECTION

In recent decades, the security researchers conducted studies to overcome the malware violation by focusing upon areas related to detection, prevention and response options. Over the years, the trend in these areas expands to various interest provides security researchers to explore and discover these options areas. Initially, to understand the detection for Android, this chapter begins by providing the background of Android information. Furthermore, this chapter provides the introduction of the security systems and the detection taxonomies. This chapter also introduces the detection options (i.e. signature-based and anomaly-based) as well as the analysis option (i.e. dynamic and static).

2.1 Background of Android

This section describes the Android architecture as well as its application package. It is to understand the basic of an Android mobile device and the elements inside the Android application itself. Android is an open source operating system that has its own unique architecture. It comprises of four layers: a) Application. b) Application framework. c) Libraries and Android runtime. d) Linux kernel.

2.1.1 Android architecture

Figure 2.1 shows the Android operating system architecture in four layers:

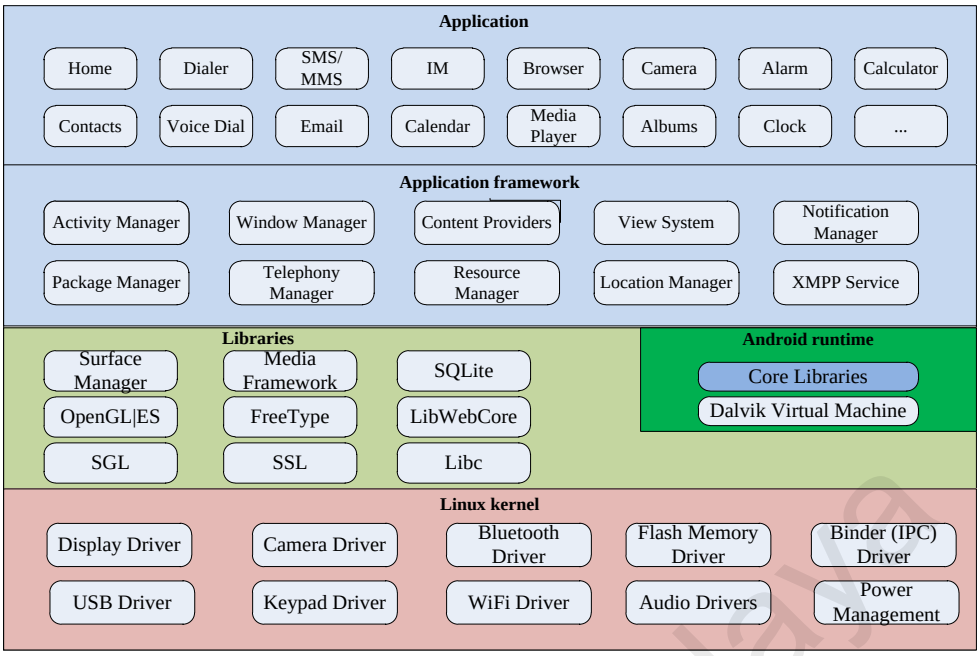


Figure 2.1: Android architecture (Android Developers, 2015)

According to Figure 2.1, each tier has its own task. The details of the layers are:

a) **Application layer:** It is the top layer in Android architecture which interacts with users directly. Each application performs different task depend on the logic of the application. Furthermore, each application has different set of permission that need to be granted during install-time in order the application to perform successfully.

b) **Application framework layer:** It provides the system server, which is a process containing the main modules for managing the device, (i.e. Activity Manager, Package Manager and Window Manager) and these components interact corresponding with Linux drivers.

c) **Libraries layer and Android runtime:** Library layer consists of a set of C/C++ libraries, which is assigned to invoke the basic kernel functionalities. The libraries are used by Application framework services to invoke protected Linux operations and to access data stored in the device. The libraries in this layer are the bionic Libc (i.e. a customized implementation of Libc for Android) and SQLite. While Android runtime comprises of Dalvik Virtual Machine (DVM), the core component that responsible to executes Dalvik Executable format (DEX) application. A mobile

device such as Android is a resource constraint environment where the battery fast depleted. DVM is chosen in Android architecture due to its efficient concurrent execution in a resource constrained environment. In the 4.4 version release, Google introduced Android Runtime (ART) which offers advanced features such as Ahead-Of-Time (AOT) compilation, improved garbage collection, development and debugging improvements.

d) **Linux kernel layer:** It is the lowest layer in the architecture. It stores drivers such as Wi-fi, camera, bluetooth, display, USB, binder (i.e. a driver that implements Inter-Process Communication (IPC)) and more necessary driver. It is built begin on Linux version 2.6 and forward.

2.1.2 Android application package

Android application is based on Android application package file (.apk) format and used to install application in android-based mobile devices. Basically, .apk consists of three elements: a) AndroidManifest.xml. b) Classes.dex. c) Resources.

a) **AndroidManifest.xml:** It is an essential file that contains the package name, components of the application (i.e. activities, services, broadcast receivers and content providers), declares permissions, instrumentation classes, minimum level of API and list of needed libraries (Android, 2015).

b) **Classes.dex:** It contains a complied source code of the application that has been converted from .java (i.e. application written in Java) to .dex extension.

c) **Resources:** It contains of all the necessary files for the application to execute, such as database, layout of the application, pictures, or graphics.

2.2 Security systems

There are three types of security systems in current research area that comprises of Intrusion Detection System (IDS), Intrusion Prevention System (IPS) and Intrusion Response System (IRS). The general information of these systems is as follows:

a) **Intrusion Detection System (IDS):** It is an application that automates the intrusion detection process to detect any possible intrusions from malware attacks (Patel *et al.*, 2013). The examples of the detection process are monitors network traffic for suspicious activity or classify the application either malware or benign and detects the possible intrusions. It is a compound process consists of identification and detection tasks.

b) **Intrusion Prevention System (IPS):** It shares the similarities with the IDS in terms of system deployment and detection method. However, it is different from IDS by one characteristics: it is designed to prevent or protect either host or network from malicious application or behavior from succeeding (Ghallali & El Ouahidi, 2012). It could adjust the security environment, such as reconfiguring the network device to protect from the malware attack.

c) **Intrusion Response System (IRS):** It is an approach to provide responses to administrator or user. The responses are from the basis of threat descriptions and attack symptoms. There are two types of response mode, namely passive and active (Anuar *et al.*, 2013). The passive response is to notify the administrator or user to activate other parties regarding the existence of the malware and depends on these parties to take further actions. The active response is to immediately execute an automated action to reduce the malware attacks without the human decision (Inayat *et al.*, 2016).

Between these systems, IDS is the main part to maintain the security system. It is the main indicator for the IPS and IRS to execute any action (Anwar *et al.*, 2017). By depending on IDS configurations and settings in detection, both IPS and IRS are capable to apply relevant countermeasures to potential incidents, hence decrease the malware violation.

2.3 Detection taxonomies

As this research focuses upon the detection, this section explains the detection taxonomies to improve the understanding of the IDS studies. Figure 2.2 depicts the taxonomy of detection (Alzahrani *et al.*, 2014) consists of scope of monitoring, detection approach and invasiveness of a technique.

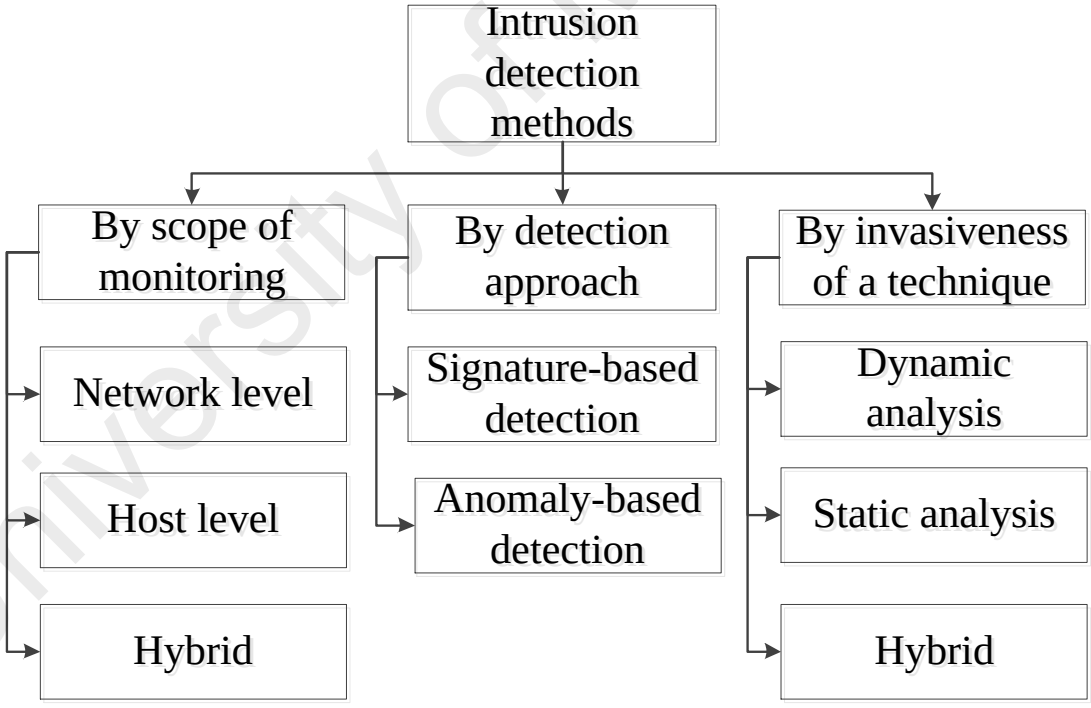


Figure 2.2: Taxonomy of detection (Alzahrani *et al.*, 2014)

a) **By scope of monitoring:** IDS spotted the malware activity by monitoring the network activities such as unusual keystroke dynamics or protocol transmissions. While host level is where the IDS stationed on a mobile device and monitor the

network from that device. While hybrid is the combination of both network and host level.

b) **By detection approach:** The signature-based technique (also known as misuse), is predominantly utilized by antivirus application that depends on detecting malware based on the constant unique signature. However, it is unable to predict unknown malware or threat because it requires consistent signature updates. For instance, Droidanalytics (Zheng *et al.*, 2013) detect malware by automatically collects, extracts and analyses the methods and classes in Android application file, which then employs it as signatures. Despite the condition that signature-based able to detect known malware, however, it is necessary to continuously update the database signature once a new malware is detected. On the other hand, anomaly-based is capable of detecting unknown (anomaly) malware by referring to classifiers prediction model (Yerima, Sezer, & McWilliams, 2014). It is perceived as a powerful due to its higher potential to address new threats.

c) **By invasiveness of technique:** There are two techniques of detection, dynamic and static. Dynamic is focuses on the behavior of the application and therefore only detects during the execution of the application. Hence, it has a limited focus because it detects the suspicious activity in a given running environment. On the other hand, static detection allows the security analysts to analyze the application without execute it by obtaining its source code. Thus, it is considered to be more thorough. While hybrid technique is the combination of both dynamic and static.

Figure 2.3 added more information in signature and anomaly-based detection:

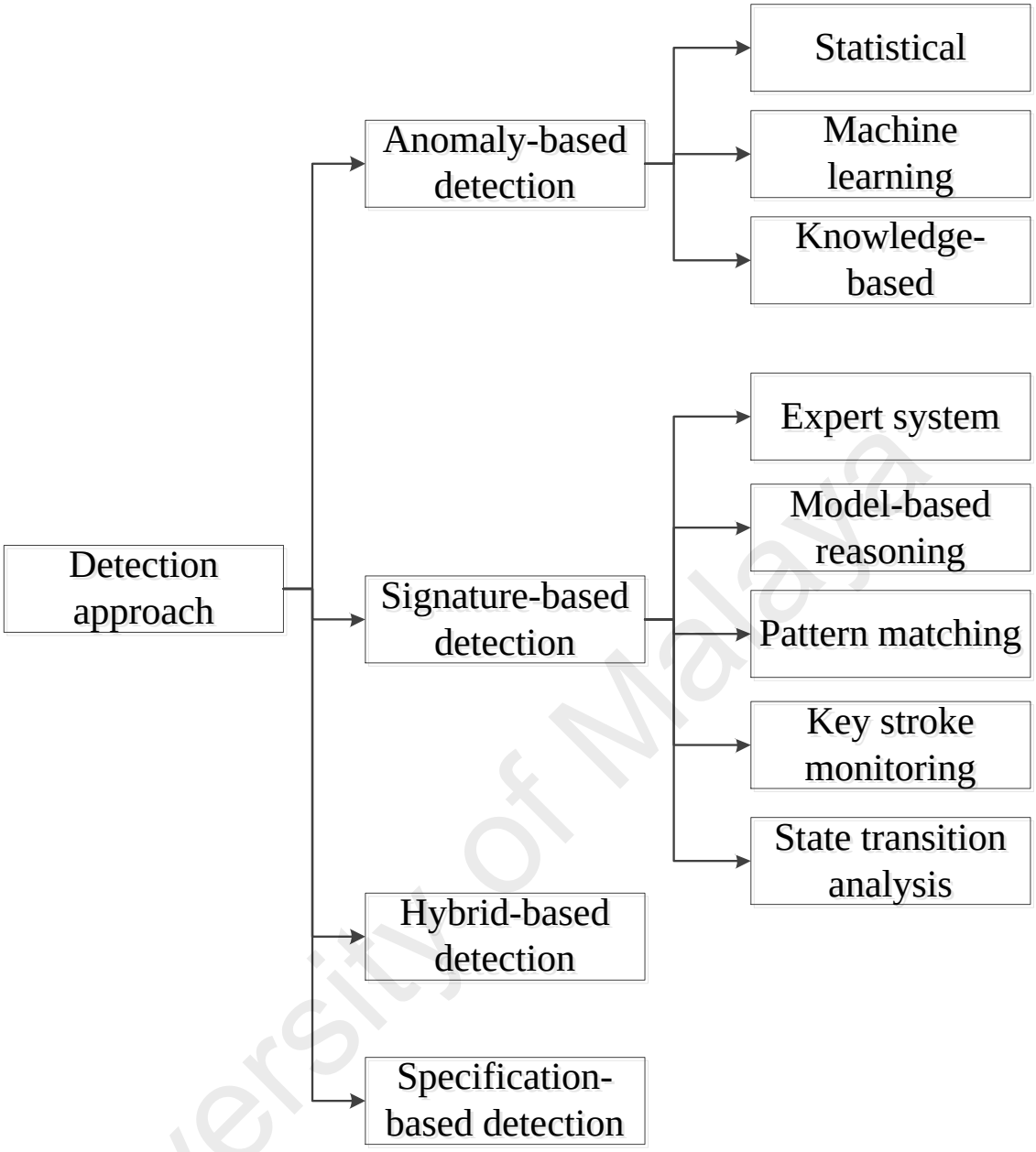


Figure 2.3: Taxonomy of detection (Inayat *et al.*, 2016)

In signature-based detection, the expert system is the knowledge regarding the attacks as if-then implication rules. Model-based reasoning is the combination models of the signature with evidentional reasoning. Pattern matching is used to store the known pattern of the malware to detect it. Static transition represents attacks as a sequence of state transition of the monitored systems. Key stroke detects the occurrence of malware by using the key stroke.

In anomaly-based detection, the statistical approach in detecting malware is by generating profiles by observing the behavior of the system activities. Machine learning generates an explicit or implicit model of the analyzed pattern, and knowledge-based is rely on the availability of the prior data of network parameter in normal and under malware attacks. Inayat *et al* (2016) claimed that the specification-based is similarly to anomaly detection with a difference in monitor the activity, where it monitors the system instead of users activity.

2.3.1 Detection options: signature vs. anomaly

Signature-based (also known as misuse), is predominantly utilized by antivirus application that depends on detecting malware based on the constant unique signature. It analyzes the activity or malware by comparing the collected information from a pattern that already defined that stored in a database. It has been used by the traditional antivirus long time ago. Zhou & Jiang (2012a) practiced signature-based and detect malware up to 79.6%. Despite the condition that signature-based able to detect known malware, however, it need to continuously update the database signature once a new malware is detected. On the other hand, anomaly-based is capable to detect unknown (anomaly) malware by referring to classifiers prediction model (Yerima, Sezer, & McWilliams, 2014).

Unlike signature-based, anomaly-based detection does not require any signatures. It differentiates the normal and malware attack by training the machine learning approaches (i.e. Artificial Neural Network (ANN), Decision Trees (DT) and Bayesian) (Feizollah *et al.*, 2017). It is a scientific discipline that is capable to predict future decisions and outputs based on the experiences gained through past input features (learning set) to predict anomalies or unknown instances (Kotsiantis *et al.*, 2006)(Feizollah, Shamshirband, *et al.*, 2013). The learning set is based on given dataset;

furthermore, intelligent decisions are made according to certain algorithms. This technique has been widely used for classifying which applications fall in which classes (normal or malware). Furthermore, machine learning belongs to the Artificial Intelligence (AI) field that allows the computer to reason and to decide based on datasets (Kotsiantis *et al.*, 2006). Table 2.1 tabulates the advantages and disadvantages between signature-based and anomaly-based detection.

Table 2.1: Signature-based and anomaly-based advantages and disadvantages

Signature-based	Anomaly-based
Advantages	
Minimum false alarm	Able to detect new and unknown malware
	Does not require signatures
Limitations	
Unable to detect new and unknown malware	High false alarm
Need signatures from the database	
Need to constantly update the signature in the database	

The advantage of signature-based is it generates minimum false alarm as it depends on the continuously update signature from the database. However, it is unable to detect new and unknown malware if the database is outdated (Feizollah *et al.*, 2015). In contrast, anomaly-based is able to detect unknown malware without require any signatures from the database. Nevertheless, it generates high false alarm (Anuar *et al.*, 2008). Therefore, in the interest for a research to achieve minimum false alarm for anomaly-based detection, it depends on the type of experiments that have been conducted.

2.3.2 Analysis options: dynamic vs. static

There are two types of malware analysis: dynamic and static. Dynamic analysis investigates the behavior of the running processes by executing the application. For instance, a study observed network system activities to monitor the applications (Narudin *et al.*, 2014). In network frames, packets, and port numbers activities, two

studies (Afifi *et al.*, 2016) and (Narudin *et al.*, 2014) examined these behaviors in detecting malware. While the HTTP features (i.e. establish Transmission Control Protocol (TCP) connection to send data from client to server), two studies (Narudin *et al.*, 2014) and (Karim, Salleh, & Khan, 2016) contemplated these features to detect malware. As opposed to dynamic, static analysis is another method which scrutinizing application codes in unexecuted condition.

Static analysis is a type of analysis which investigates the malware application code, hence covers over all the possible activities in an application within an unlimited range of time because the analysis is unexecuted (Chess & McGraw, 2004). The main step of static analysis procedure is the reverse engineer process. It is the process to retrieve the whole code and further scrutinize the structure and substance within the application (Aafer *et al.*, 2013; Chang & Hwang, 2007; Sharif *et al.*, 2008). Therefore, it is able to examine the overall code, requires low memory resources, minimal CPU processes and the analysis process is fast because the application is unexecuted. Additionally, static analysis is capable to discover unknown malware with enhanced detection accuracy with machine learning approaches (Narudin *et al.*, 2014)(Feizollah, Anuar, *et al.*, 2013). Table 2.2 compares the advantages and disadvantages of dynamic and static analysis.

Table 2.2: Static and dynamic advantages and disadvantages

Dynamic	Static
Advantages	
Able to detect unknown malware	Low resources (e.g. CPU, memory, network, and storage). Relevance in mobile devices equipped with low specifications
Detection of normal applications that change to malware on-the-fly	Fast processing in conducting reverse engineering the application
	Examining overall code and further, discover entire possible action
	Able to detect unknown malware with the aid of machine learning
Limitations	
High resources (e.g. CPU, memory, network, and storage)	Inability to detect normal application that changes to malware on-the-fly
Possibly misses the malware activities that beyond the analysis range	Investigation continues in finding minimal features (e.g. permission, function call, and strings) to detect malware
Difficulty in detecting applications that able to hide malicious behavior while it runs	
Investigation continues in finding minimal features (e.g. traffic, memory) to detect malware	

In Table 2.2, both types of analyses share the similar limitations, whereas selecting the best features in minimal amount. In detecting malware, features refer to attributes or elements to differentiate an application is either malware or benign. Security practitioners face obstacles in investigating various features in all types of categories (e.g. permission, API, directory path, and code-based) along with the need to decrease these features at the same time. Finding best features in minimal amount is crucial because it enhances accuracy (i.e. accurate predictive model) with fewer data and reduces model complexity (Feizollah *et al.*, 2015).

2.4 Summary

This chapter introduced the Android information in its architecture as well as the application package. It also highlighted the types of detection by providing the detection taxonomies, comparing their unique characteristics and operations through different types of detection and analyses that include signature-based, anomaly-based, dynamic and static analysis. Then it discusses this information to underline the advantages and disadvantages between them. The subsequent chapter presented a review of machine learning and the static features issues in previous studies to discover aspects that receive less attention and to facilitate the feature selection issues.

CHAPTER 3: MACHINE LEARNING AND STATIC FEATURES ISSUES

As detailed in the previous chapter, this study adopts static analysis because of its rapid processing attribute, its overall code coverage and its low resource requirements. In the machine learning approach, finding the best features in minimal amount is crucial because with fewer data, accuracy (i.e. accurate predictive model) is enhanced thereby, reducing model complexity (Feizollah *et al.*, 2015). However, searching for the malware features which reside in thousands of lines of code located in each application can be a sophisticated, difficult and complicated process that requires multiple experiments to be conducted.

In most malware analysis studies, permissions have been utilized as a feature and these features normally reside in the AndroidManifest.xml file. It is a confinement which limits access to an application to utilize the part of the code or information located in the Android mobile devices. Studies which use only permission as features include (C.-Y. Huang *et al.*, 2012; Peng *et al.*, 2012; Sahs & Khan, 2012; Samra *et al.*, 2013; Walenstein *et al.*, 2012). However, there are numerous features, besides permission, that are also ready to be explored.

Since there are numerous static features and many malware are zero-permission (X. Zhou *et al.*, 2013)(Adrian, 2012), security analysts have also resorted to investigating other categories of features so as to increase detection accuracy. Karim *et al.* (2016) covered two categories of features such as API and permission while Arp *et al.* (2014) included categories of API, permission, URL and hardware components. These studies proved that by adopting other features besides permission, they improve the detection accuracy in malware detection.

In addition, there are studies that searched the features in one particular file only. For instance, Sanz *et al.* (2013), Talha *et al.* (2015) and Sarma *et al.* (2012) focused on the permission features extracted from only one particular file, the AndroidManifest.xml.

In some studies, features were extracted manually and then inspected without referring to any reliable and completed list of resources to verify their features. (Arp *et al.*, 2014; Karim, Salleh, Khan, *et al.*, 2016; Yerima *et al.*, 2015; Yerima, Sezer, & McWilliams, 2014) are some examples. Comparatively, fewer studies have concentrated on root exploit features. The current study aims to fill that gap by first attempting to search for the features in the overall files. It will then conduct the experiments based on the main and reliable list of features as a reference before investigating the features that root exploit frequently uses.

Besides mentioning the categories of features involved, it is also necessary to mention the method used to select the static features. In their work, Azhagusundari and Thanamani (2013) utilized Information Gain (IG) as a method, while Priyadarsini *et al.* (2011) used Gain Ratio (GR) as a method to search for the optimized features. Arp *et al.* (2014) adopted a method called joint vector space to identify the typical patterns of the features geometrically.

Apart from the methods mentioned above, there are also opportunities for research to include methods that utilize evolutionary algorithms for inspecting the repeated features located in the same application. Stein *et al.* (2005) and Middlemiss and Dick (2003) utilized evolutionary algorithms to detect malware by using dynamic analysis.

This chapter begins by reviewing the machine learning classifier issues, the methods involved in selecting the best features which are ideal for machine learning and the categories of the features involved.

3.1 Machine learning classifiers

This study attempts to investigate and discover the different utilizations of the different machine learning types (i.e. function, tree and bayes) which have been frequently and infrequently used in past research efforts. Table 3.1 provides the machine learning classifiers extracted from previous static analysis studies for Android.

Table 3.1: Machine learning classifiers in static analysis studies

References	Machine learning classifiers involved	Machine learning classifiers used in this thesis						
		NB	FT	J48	RF	MLP	VP	RBFN
(Shabtai *et al.*, 2010)	DT, NB, BN, Part, boosted bayesian network, boosted decision tree, RF, and voting feature interval (VFI)	NB			RF			
(Sanz, Santos, Laorden, Ugarte-Pedrero, Bringas, *et al.*, 2013)	Simple logistic, NB, BN, sequential minimal optimization, instance-based learning with parameter k, J48, random tree, and RF	NB		J48	RF			
(Peiravian & Zhu, 2013)	SVM, J48, bagging. prism and KNN			J48				
(Yerima, Sezer, & Muttik, 2014)	NB, part, ridor, DT, and simple logistic	NB						
(Yerima *et al.*, 2015)	RF, random tree, NB, DT, and simple logistic	NB			RF			
(Chan & Song, 2015)	NB, SVM with sequential minimal optimization (SMO), radial basis function network (RBFN), MLP, liblinear, DT, and RF	NB			RF	MLP		
	Total	5	0	2	4	1	0	0

Legends: NB = Naïve Bayes, FT = Functional Trees, RF=Random Forest, MLP = Multilayer Perceptron, VP=Voted Perceptron, RBFN= Radial Basis Function Network

Table 3.1 lists the type of classifiers which have been applied in static analysis by previous research. The table shows that security analysts used NB and RF more than the other classifiers (i.e. FT, J48, MLP, VP and RBFN). Furthermore, these security analysts also used MLP once only while FT was excluded from their analyses. Based on this, it is useful to prefer NB and RF because security analysts had utilized these classifiers regularly. Moreover, they have also received huge acknowledgments in the intrusion detection system area Thus, there is an opportunity for studies to discover the classifiers of FT, J48, MLP, VP and RBFN because previous studies (Y. Lu *et al.*, 2013)(Díaz-Uriarte & Alvarez de Andrés, 2006)(Caruana *et al.*, 2008) seldom used these three classifiers of FT, NB and RF as a part of static analysis investigation. Hence,

it is beneficial to observe the distinctive results noted in frequently as well as infrequently used machine learning classifiers (i.e. RF, NB, FT, J48, MLP, VP and RBFN). Figure 3.1 depicts these classifiers in categories. Table 3.2 displayed their advantages.

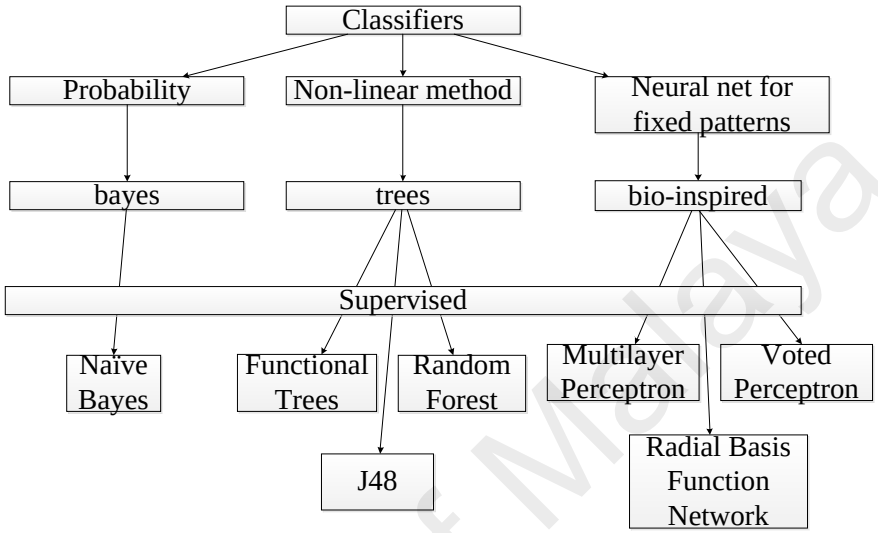


Figure 3.1: Machine learning types

Table 3.2: Machine learning classifier advantages

Categories	Machine learning classifiers	Advantages
Bayes	NB	1) Operates on the (naïve) assumption (i.e. a fruit considered an apple if it is red, round, and about 3 inches). 2) Performs well in certain real-life applications (i.e. file classification and spam filtering). 3) Learning and classifying in an extremely rapid manner.
Tree	RF, FT, and J48	1) A model is constructed from root until reaching the leaf. 2) Generally known as "divide and conquer" algorithms (Kotsiantis, 2013).
Bio-inspired	MLP, VP, and RBFN	1) Stable learning algorithm (Lippmann, 1987). 2) ANN based on the biological neural network, which consists of three layers (input, hidden, and output).

The MLP structure is based on a feed-forward (also known as error backpropagation) neural system that has at least one or more layers situated between the input and output layer (Lippmann, 1987). The feed-forward indicates the information streams in one

direction, forward heading from the input to the output layer. The network learns the training information by altering the synaptic weight of the neurons conferred to the error existing on the output layer. It has been noted that Artificial Neural Network (ANN) is used in numerous fields due to its powerful and stable learning algorithm. Analysts have generally utilized the MLP for pattern classification (making inferences from perceptual data), recognition (focuses on regularities in data), prediction and approximation. Figure 3.2 depicts the MLP feedforward concept.

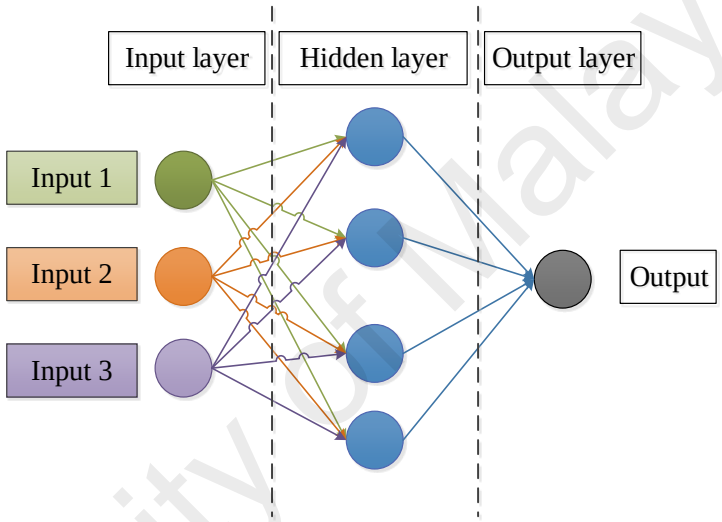


Figure 3.2: MLP concept (Lippmann, 1987)

Here, the figure shows the three input layers which consist of input, hidden and output layer.

In the subsequent section, Figure of 3.3 indicates the Voted Perceptron (VP) which is a classification algorithm that was introduced by Freund and Schapire (1999). The VP works on a vote system. Here, the algorithm keeps a list of all the forecast vectors which are created after each misclassified component. It then calculates the number of iterations each vector endures. By utilizing the total number of iterations as votes, the models which survived the most (i.e. fewer mistakes are made using this model) have a greater majority on the vote. This means that the information it keeps during training makes up the list of all the prediction vectors that were created after every mistake.

Each vector ascertains the quantity of the cycles it "survives" until the subsequent mistake is made. In this regard, the VP refers this count as the "weight" of the prediction vector. Figure 3.3 details the VP algorithm from the training until the prediction phase.

Training

Input: a labeled training set $\langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\rangle$
number of epochs T

Output: a list of weighted perceptrons $\langle(\mathbf{v}_1, c_1), \ldots, (\mathbf{v}_k, c_k)\rangle$

- Initialize: $k := 0$, $\mathbf{v}_1 := \mathbf{0}$, $c_1 := 0$.
- Repeat T times:
  - For $i = 1, \ldots, m$:
    - Compute prediction: $\hat{y} := \text{sign}(\mathbf{v}_k \cdot \mathbf{x}_i)$
    - If $\hat{y} = y$ then $c_k := c_k + 1$.
      else $\mathbf{v}_{k+1} := \mathbf{v}_k + y_i \mathbf{x}_i$;
      $c_{k+1} := 1$;
      $k := k + 1$.

Prediction

Given: the list of weighted perceptrons: $\langle(\mathbf{v}_1, c_1), \ldots, (\mathbf{v}_k, c_k)\rangle$
an unlabeled instance: $\mathbf{x}$

compute a predicted label $\hat{y}$ as follows:

$$s = \sum_{i=1}^{k} c_i \,\text{sign}(\mathbf{v}_i \cdot \mathbf{x}); \quad \hat{y} = \text{sign}(s).$$

Figure 3.3: VP algorithm (Freund & Schapire, 1999)

Following the VP algorithm, the ANN types of the applications used are also discussed.

Here, the Radial Basis Function Network (RBFN) is one of the ANN types for the application of supervised learning. It uses radial basis functions which is also known as activation function to calculate the derived features in the neural networks. The application gives value to each point noted on the distance taken from its origin. Figure 3.4 depicts the RBFN architecture which comprises input layer, hidden layer and output layer with bias.

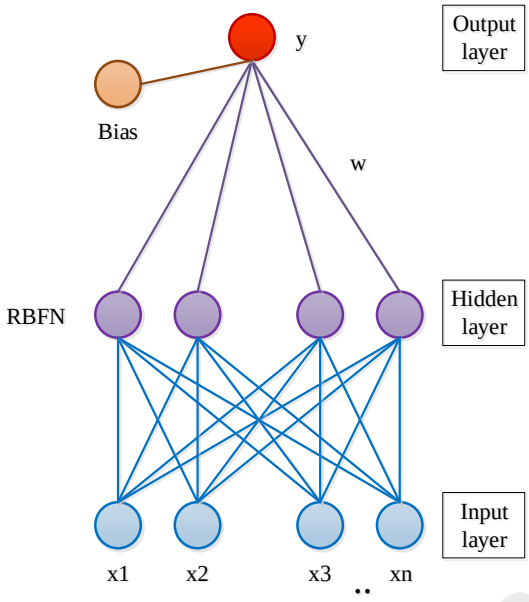


Figure 3.4: RBFN architecture (Walczak & Massart, 2000)

Here, the RBFN is a three-layer feedforward structure with points as indicated below:

a) The input layer only serves as the input distributor to the hidden layer.

b) Every node in the hidden layer is a radial function. Its dimensionality is similar to the dimensionality of the input information.

c) A linear combination then calculates the output layer, particularly the weighted sum of the radial basis function including the bias. In the context of machine learning algorithm, it can be referred to the following equation:

$$y(x_i) = \sum_{j\,=\,1:K} w_{j\phi}\left(||x_i - c_j||\right) + w_0 \qquad 3\text{-}1$$

As machine learning is being connected with more complex tasks, it is important to recognize the most suitable data noted from a large amount of information. This data may contain an extensive number of features which need to be reduced through careful selection. The method employed for doing so is further explained.

3.2 Methods in selecting features for anomaly-based detection

Theoretically, the more the features, the better the ease in accomplishing accurate prediction in machine learning. Nonetheless, too many features may hinder the learning phase and confuse the learning algorithm. Such an occurrence causes the classifiers to over-fit the training data (Yu & Liu, 2004). Consequently, the results of the prediction become inaccurate.

Over the years, the ability to select optimized features has received attention. This process has been used in numerous research areas encompassing machine learning, statistical pattern recognition and data mining (Blum & Langley, 1997). Security analysts (Appavu *et al.*, 2011) have also highlighted that feature selection increases the accuracy of machine learning detection as well as reduces the complexity in machine learning classifiers' learning results. Other advantages of features selection (i.e. selecting the best features) include the use of less measurement and less storage requirements, less learning, training and testing processes. The method enhances data visualization and data understanding and it deals with the effect of dimensionality by increasing prediction accuracy (Guyon & Elisseeff, 2003).

It has been noted in machine learning prediction that as the capacity of the data rises, the difficulties of the classification problem and data analysis also increases significantly. Powell (2011) claims that this situation is known as the curse of the dimensionality. Generally speaking, selecting minimal features of data is necessary for the improvement of the prediction's accuracy as well as for the achievement of a faster processing rate. Thus, it acts as cost-effective predictors. Moskovitch *et al.* (2008) and Koller and Sahami (1996) adopted the feature selection practice for their machine learning predictions. The practice improved their accuracy rate showing low false positive rate. Noting its ease, the common feature selection methods utilized in this study and its

inclusion of Information Gain (IG), Gain Ratio (GR) and Chi Square (CS) are further presented.

3.2.1 Information Gain (IG)

Information Gain (IG) is one of the feature selection methods proposed by Clause Shannon in 1948 for information theory research. This method aims to find fundamental limits on communication operation and signal processing (Ueltschi, 2006). It is done by measuring the amount of data (i.e. bits), regarding the prediction of the class and the corresponding class distribution (Han *et al.*, 2001). The IG method considers feature to be more important if its normalized information gain is larger and if it treats all features as independent (Duch *et al.*, 2003).

$$Gi = I(S) - Ei; \quad G'i = Gi/Ii \tag{3-2}$$

Table 3.3 tabulates the IG utilizations which were extracted from previous studies.

Table 3.3: IG utilization

References	Utilization
(Roobaert *et al.*, 2006)	Adopted for feature correlation and variable selection with SVM for induction purposes.
(Appavu *et al.*, 2011)	Adopted with Bayes theorem by searching out the dependent features and eliminating the redundancy among them.
(Azhagusundari & Thanamani, 2013)	Searched the optimized features by utilizing IG as well as discernibility matrix.

In this table, it can be seen that the IG utilization was applied. Based on previous studies, it was noted that Roobaert *et al.* (2006) had successfully improved the generalization performance of the Support Vector Machine (SVM) models in five classification datasets of the competition. Hence, the IG is seen as one of the filter approaches used in the selecting features' process.

Appavu *et al.* (2011) adopted the Bayes theorem and the IG method to enhance the accuracy of the prediction of their work. Likewise, it was proven that the IG is able to

improve the NB machine learning classifier prediction by three percent, gaining from 83.93% to 86%.

Azhagusundari and Thanamani (2013) also produced an outstanding result in relation to the number of features selected. Their prediction accuracy also improved when the method was distinctly applied.

3.2.2 Gain Ratio (GR)

Relating this to previous studies, the gain ratio (GR) is the modification of the IG method. It is used for selecting the best features. It takes the number and the size of the branches into account when choosing the features. It considers the intrinsic information of a split as a means to correct the IG. The intrinsic information is the entropy of the distribution of instances in the branches. Here, the value of the features decreases as the intrinsic information increases (Han *et al.*, 2001). The GR is defined in 3-3 as:

$$GainRation(A) = \frac{Gain\ (A)}{SplitInfo\ _A\ (D)} \qquad 3\text{-}3$$

The feature containing the maximum GR value is designated as the splitting feature. Once the split information reaches 0, then the ratio becomes unbalanced. Therefore, a constraint is added to hinder this situation whereby the IG of the selected essential test then becomes enormous (Han *et al.*, 2001).

In their work, Priyadarsini *et al.* (2011) increased the dimensionality so as to search for a subset of seven features from the original dataset which consist of ten features. They compared the reduced dataset with the original dataset to detect the accuracy of the prediction. Their results indicate that the level of the accuracy remains similar. This proves that dimensionality reduction does not affect the accuracy level.

3.2.3 Chi Square (CS)

Another common feature selection method used in information technology research is the Chi Square (CS) where the x^2 test is used to test the independence of two events. The test measures the divergence from the conveyance expected in the event. When it accepts the element event, it is accepted to be really autonomous of the class value. As a statistical test, the Chi Square is known to behave erratically for little expected counts, which are regular in text classifications. This is because it occurs once in a while with word features, and at times when having a few positive preparing cases for a concept (Foreman, 2003).

The Chi Square (CS) is particularly used to test the occurrence of a specific class and specific terms which are independent. Therefore, it can estimate the following quantity for each term and rank them by scores with the equation of:

$$X^2(D,t,c) = \sum_{e_I \in \{1,0\}} \sum_{e_C \in \{1,0\}} \frac{(N_{e_I e_C} - N_{e_I e_C})}{E_{e_I e_C}} \quad 3\text{-}4$$

The high scores seen in the Chi Square of x^2 indicate that the null hypothesis (H_0) of independence should be rejected. This also means that the occurrence of the term and class is dependent. If they are both dependent, then the researcher needs to select the feature for the classification. Other than the IG, GR and CS, there are additional methods used by others in selecting features. They are further mentioned below.

3.2.4 Other feature selection methods

In order to accomplish considerable results from the machine learning system, it is vital to select minimal features. Table 3.4 tabulates the information of the methods used in selecting the features. It also provides the information extracted from previous works.

Most studies (Arp *et al.*, 2014) use Drebin to analyze joint vector space and to identify the typical patterns of the features in geometric form. For example, Karim *et al.* (2016) utilized the element tree xml Application Programming Interface (API) including regular expression to identify strings or the number of attributes noted in obtaining features.

Table 3.4: Other methods in selecting features

References	Method to select features	Information
(Arp *et al.*, 2014)	Joint vector space	Used to identify patterns of features
(Karim, Salleh, & Khan, 2016)	Element tree xml API and regular expression	Used to wrap an element structure or string
(T. Zhang, 2009)	Forward greedy algorithm	Greedily selecting another feature in each iteration
(Lai *et al.*, 2011)	Forward-backward greedy algorithm	Greedily selecting and removing another feature in each iteration

Further to using Drebin, the forward greedy algorithm (T. Zhang, 2009) has also been used to select features. This algorithm is part of an investigation for sparse approximation (Tropp, 2004). The fundamental awareness of the forward greedy algorithm is that it greedily selects another feature per iteration. It also has the means to forcefully decrease the loss objective. This strategy is generally, capable of locating the best features. Forward greedy algorithm follows the problem-solving heuristics of making local optimal decisions at every phase through the inspiration of searching for global optimum. The forward greedy algorithm generalizes the case of measurement noise. It is able to identify features in a sparse eigenvalue condition. This strategy's capability continues for as long as each non-zero coefficient is larger than the constant time of the noise level. However, forward greedy algorithm is unable to correct the mistakes it has previously made. Hence, Lai *et al.* (2011) combined the forward greedy algorithm strategy with the backward greedy algorithm to resolve the downside of the strategy. Unlike the forward greedy algorithm strategy which selects feature at every iteration, the backward greedy algorithm removes each feature per iteration. Therefore,

the features that have minimum contributions are removed and this can further decrease the cost function.

Different from the joint vector space and forward greedy algorithm, the evolutionary algorithm (i.e. population, generation, crossover, mutation) can be used to search for optimal and relevant features that are contained in multiple categories.

a) Genetic Algorithm (GA) and Genetic Search (GS)

Genetic algorithm (GA) is the inspiration extracted from evolutionary biology. It is a technique that automatically improves parameters or features. Table 3.5 lists past related works which had deployed the GA in selecting features. The strategy has been widely used in numerous fields such as soil classification, colon cancer identification, gene expression and malware detection.

Table 3.5: Studies of GA in selecting features

References	Objectives	Year
(Punch *et al.*, 1993)	Classify soil to three environments; a) near the roots of a crop (rhizosphere). b) away from the influence of the crop roots (non-rhizosphere). c) from a fallow field (crop residue)	1993
(Fröhlich *et al.*, 2003)	Classify colon cancer and gene expression	2003
(Middlemiss & Dick, 2003)	Detect malware intrusion (dynamic analysis)	2003
(Stein *et al.*, 2005)		2005

In their work, Punch *et al.* (1993) utilized feature selection and data classification on soil classification by using GA combined with the K-Nearest Neighbor (KNN). The KNN assists the study in measuring the similarity of the samples. However, it is unable to inform the relative importance of the features in discriminating known samples. Henceforth, GA, along with the KNN, as the main part of the evaluation stage, is applied to optimize the features. The dataset for their study were rhizosphere data, the world's natural dataset. This dataset formed part of the soil ecosystem where plant roots, soil and soil biota communicate with each other. These connections were advantageous

to plants in that the soil fruitfulness was enhanced and harmful chemical debasement was upgraded. In this regard, it can be said that the study attempted to classify three rhizosphere classes namely: the rhizosphere, the non-rhizosphere and the crop residue.

Likewise, Fröhlich *et al.* (2003) also applied the GA in their work to make three classifications (i.e. toy data, colon cancer, and gene in a yeast dataset). The genetic evolutionary process in the GA enabled them to switch the different factors and to optimize parameters in the Support Vector Machine (SVM) algorithm. The analysts also used theoretical bounds on the generalization error for the SVM by proposing a decimal encoding that is much more efficient than the binary encoding. If the number of features to be selected was unfixed beforehand, the usual binary encoding is preferred. Furthermore, kernel parameters such as the regularization parameter C of the SVM were able to be optimized by the GA for selecting a feature subset, given that the choice of the feature subset influences the appropriate kernel parameters and vice versa.

Middlemiss and Dick (2003) utilized the GA to select optimal features via weighted feature extraction and machine learning. The study conducted was a malware detection study that applied the dynamic analysis approach. To accomplish their aims, the analysts implemented the GA to calculate the weights for the dataset features. Thereafter, the KNN classifier was applied to the fitness function through the GA so as to assess the new weighted feature set performance. Their results showed that the weighted set of features for the class classification of data increased the accuracy of the intrusion detection.

Another study that attempted a malware detection by applying the GA is (Stein *et al.*, 2005). They used the GA algorithm as a method to select a subset of features which were put into the DT classifiers through static analysis. They then utilized the KDDCUP

99 as a dataset to train and test the tree classifiers. Through the assistance of GA, the studies were able to successfully select the best features.

From the previous studies noted in Table 3.5, it appears that none had adopted the GA for selecting the best features in malware detection using static analysis with machine learning. In this regard, the gap provides an opportunity for others to explore the advantages of using the GA in selecting features genetically. Apart from discussing the features' selection methods, the opportunity in using repeated features is also revealed below.

3.2.5 Repeated features in similar application

As mentioned in chapter 1, this thesis was motivated by the aim of conducting an experiment that calculate the repeated features noted in a similar application and its efficiency in detecting malware. This novel feature extraction method was inspired by the fact that some features are exist and manifest themselves multiple times in other files within one application. For instance, one of the API features, getSubscriberId, was discovered to exist multiple times in the same application. In this regard, this thesis explores a similar factor by adopting the novel method of selecting features for anomaly-based detection through the GA.

On the other hand, as root exploit has received lesser attention in research hence following section explains the methods in detecting this type of malware.

3.2.6 Methods in detecting root exploit

One type of malware called root exploit is a malicious application which modifies the kernel in an Android OS so as to gain super-user privileges. When the attackers gain root and the privileges increase, they are able to install other malware types such as botnets, worms or Trojans. When this occurs, the attackers are also capable of evading detection by modifying the OS code, execute stealthily and bypassing over the

permission (Bickford *et al.*, 2010; Ma & Sharbaf, 2013; A. Schmidt *et al.*, 2009). The number of root exploits has increased because of malware creators and homebrew community attackers (i.e. smartphone users who break the OS kernel to obtain a customized version of the OS). When a new version of the OS is released, malware creators develop their own root exploits or they wait for the homebrew community attackers to determine ways to break the OS (Felt *et al.*, 2011). In this context, homebrew community is a group of people who change the OS default structure in mobile devices for their own benefit (e.g. to customize the graphic @ outlook, accelerate hardware capabilities). Thus, one way to deter these attacks is to conduct an investigation for root exploit features.

Despite all the investigations that have been conducted, fewer studies have given focus to discussing root exploit malware, particularly for Android mobile devices, except for Droidanalyzer (Seo *et al.*, 2014) and Droidexec (Wei *et al.*, 2015). This justifies the investigation conducted on root exploit is rare especially involves Android. Unlike Droidanalyzer and DroidExec, this thesis aims to adopt machine learning as a means to detect it.

The Droidanalyzer (Seo *et al.*, 2014) used an algorithm to calculate the MD5 hash value which was cross-referenced in the database of signatures. In comparison, the similarity recognition applied by Droidexec (Wei *et al.*, 2015) used a structural graph constructor (i.e. function–relation graph extraction and opcode component graph constructor). Clearly, both had also overlooked detecting root exploit with the machine learning strategy.

Due to this lack of attention, this thesis uses the anomaly-based technique which adopts machine learning to detect unknown root exploit. Having discussed the methods in

detecting malware, further section explains the categories of features involves for the anomaly-based detection system.

3.3 Categories of features for anomaly-based detection

Figure 3.5 depicts the taxonomy of the mobile malware features (see (Feizollah *et al.*, 2015). The taxonomy divides the features into four categories: static, dynamic, hybrid and application metadata.

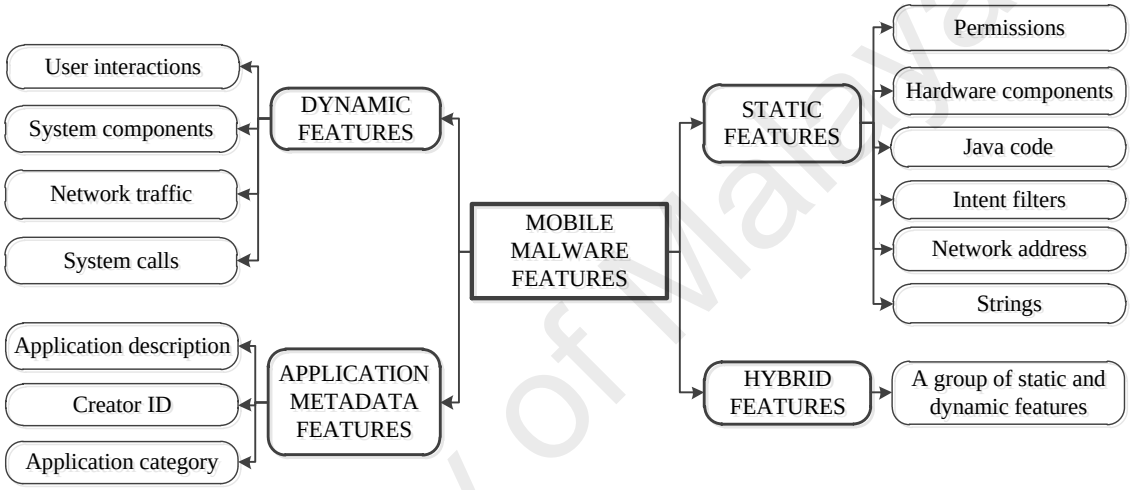


Figure 3.5: Taxonomy of mobile malware features (Feizollah *et al.*, 2015)

From the figure, it is noted that the most features are dynamic and static. Unlike the dynamic option, the static option seems to be carrying more features. The advantage of using static analysis is its rapid processing and low resources which attract security analysts to spend their time in exploring and searching for the best features in-depth. Table 3.6 tabulates the categories of features. Here, it can be seen that multiple categories of features (e.g. permission, API, function call, code structures, sources and sink, strings) were combined so as to detect malware.

Table 3.6: Series of features in static analysis studies

References	Features	Information about the features
(Aafer *et al.*, 2013; Deshotels *et al.*, 2014; S.-H. Lee & Jin, 2013)	API	Contain codes of an application consist of classes, methods, functions, and parameters
(Gascon *et al.*, 2013; A.-D. Schmidt *et al.*, 2009)	Function call	In application code, it is a declaration in an argument. It either contains any number of names either separated by commas or empty.
(Suarez-Tangil, Tapiador, Peris-Lopez, & Blasco, 2014)	Code structures	Comprises of a line or set of programming codes in an application.
(Gordon *et al.*, 2015; L. Lu *et al.*, 2012)	Sources and sinks	Sources and sinks are related terms. The sources in computing area are where the data enter the program, whereas sinks are where the data flows to leave the program (Rasthofer *et al.*, 2014).
(Faruki *et al.*, 2013)	Bytes	Referring to code in an application.
(Junaid *et al.*, 2016)	Reverse-engineered Life Cycle Model	Android application consists of essential building blocks called application components (activity, service, broadcast receiver, and content provider) which follow a life cycle model during execution.
(Aung & Zaw, 2013; C.-Y. Huang *et al.*, 2012; Peng *et al.*, 2012; Sahs & Khan, 2012; Samra *et al.*, 2013; Walenstein *et al.*, 2012; Wu *et al.*, 2012)(Sanz, Santos *et al.*, 2013)(Talha *et al.*, 2015)(Sarma *et al.*, 2012)	AndroidManifest.xml	One of the files in Android application is AndroidManifest.xml. It is an essential file, containing the package name, the application components (activities, services, broadcast receivers and content providers), the permission declarations, the instrumentation classes, the API minimum level, and the list of necessary libraries (Android, 2015).
(Feizollah *et al.*, 2017)	Intent	Intent objects deliver an abstract definition of the operations in an application which plans to accomplish.
(Aafer *et al.*, 2013; Arp *et al.*, 2014; Arzt *et al.*, 2014; Bartel *et al.*, 2012; Feng *et al.*, 2014; M. Grace *et al.*, 2012; J. Huang *et al.*, 2014; Liang *et al.*, 2013; Peiravian & Zhu, 2013; Sheen *et al.*, 2015; Wu *et al.*, 2012; Yerima *et al.*, 2014; W. Zhou *et al.*, 2013)	API and permission	In the Android OS process, permission and API are depend on to each other. Parts of API calls in the code needs permission to execute (Wu *et al.*, 2012).
(Apvrille & Strazzere, 2012; Luoshi *et al.*, 2013; Yerima *et al.*, 2014)	API, permission, and others	Combined features consist of API, permission, and others.
(Seo *et al.*, 2014)	API and other features (exclude permission)	Combined features consist of API and others, except permission.
(M. Grace *et al.*, 2011; Sarma *et al.*, 2012; Shabtai *et al.*, 2010; Yang & Yang, 2012) (Kang *et al.*, 2015; W. Zhou *et al.*, 2012) (Yerima *et al.*, 2014) (Yerima *et al.*, 2013)	Permission and other features	Apart from API combinations, security analysts also combine permission with other features.
(M. C. Grace *et al.*, 2012; J. Lee *et al.*, 2015)	Other features except for API and permission	Features used other than API and permission.

Table 3.6 shows that the previous security analysts had investigated features either by extracting the existence of the features or by manually inspecting these features without referring to a reliable and complete list of studies as a reference. This is a disadvantage, as seen in some studies (Aafer *et al.*, 2013; Arp *et al.*, 2014; Karim *et al.*, 2016; Sanz *et al.*, 2013; Sanz *et al.*, 2013; Yerima *et al.*, 2014; Yerima *et al.*, 2014) because without referring a complete list as guidance, the method of extracting features is likely to miss some important features that significantly to detect malware during the experiment. Due to this, the list of features stated in the Android official website such as the list of permission (Android, 2015) and telephony (Android, 2016), is important.

Apart from this, certain existing studies as noted in Table 3.6, appear to have extracted the permission features from only one particular file such as the Androidmanifest.xml although many other applications in the Androidmanifest.xml have zero permission (Adrian, 2012). In this regard, it is hard to differentiate between the malware and benign applications. In addition, this approach may further increase the false positive detection value. Consequently, investigating features that reside in all the files located in an application including the Androidmanifest.xml file need to be explored in order to fully discover the potential of the features in malware detection.

In addition, previous studies less discussed the root exploit features with machine learning approach. Given this scarcity, there is a need to conduct investigations to find the best in a minimal quantity of features to detect unknown root exploit as well as other types of malware by utilizing anomaly-based intelligent machine learning prediction system. Particularly, at this time of writing, none of the existing studies concerns root exploit features except Droidanalyzer (Seo *et al.*, 2014) and Droidexec (Wei *et al.*, 2015).

3.3.1 Categories of root exploit features

In looking at the categories of root exploit features, the Droidanalyzer (Seo *et al.*, 2014) combined the API, rooting and botnet command as features for detecting root exploit and mobile botnet. The analysts analyzed the risky API and the strings which identified malware by using particular features and keywords. The system included some features taken from those categories of root exploit. In their work, the security analysts only listed the keywords as an example without revealing the exact list of the features. Therefore, the features to detect root exploits are still unavailable. In contrast, the Droidexec (Wei *et al.*, 2015) adopted a graph constructor which used opcode components as features for detecting root exploit. Clearly, these two studies employ different approaches than machine learning to detect root exploit.

a) Android Debug Bridge (ADB)

One of the novel features noted in this thesis is ADB. It is a tool that permits the local computer to connect to the Android mobile device or an emulator. It connects the mobile device or other personal wireless component with the local computer therefore, users are able to interact with the mobile device through the command line of the local computer (Android Developer, 2017).

In comparison to previous machine learning studies as shown in Table 3.6, the features which were precluded in this thesis are *startservice -n* and *adb_enabled*. The features being observed are of the ADB type which was included in the system command category. According to literature review, the ADB command is one of the novel types of features which have not been exploited through static analysis and machine learning. Moreover, this study combines the ADB with other categories of features (i.e. system command, directory path and code-based) to discover unknown root exploit. Although certain directory path is discussed (Yerima, Sezer, & McWilliams, 2014), the current

study attempts to identify specific directory paths that detect root exploit only instead of detecting all types of malware. In this chapter, after the root exploit features are discussed, the opportunities in searching for the features in the overall files by referring to a complete list as guidance is further explained.

3.3.2 Searching for features in overall files and complete list as guidance

In Section 3.3, it was mentioned that most studies had inspected the features without making reference to all the previous studies as a guide. Section 3.3 also mentioned the importance of searching and counting the features in the overall files in each application. This is done prior to the detection accuracy test which may decrease if features have been taken from one particular file only. To overcome that weakness, it is crucial to have the complete list of features noted in each category so as to prevent any features from being missed out in the experiment. The information provided in Table 3.7 compares the similarities and differences of features extracted from the manifest file, the overall files as well as the complete list of guidance.

Table 3.7: Similarities and differences in extracting features

References	Extracting features
(Apvrille & Strazzere, 2012; Aung & Zaw, 2013; Karim, Salleh, Khan, *et al.*, 2016; Peiravian & Zhu, 2013; Peng *et al.*, 2012; Sanz, Santos, Laorden, Ugarte-Pedrero, Nieves, *et al.*, 2013; Sarma *et al.*, 2012; Seo *et al.*, 2014; Shabtai *et al.*, 2010; Sheen *et al.*, 2015; Yerima *et al.*, 2013, 2015; Yerima, Sezer, & Muttik, 2014; W. Zhou *et al.*, 2012)	• In manifest file only • Without referring to a complete list (manual inspection, investigation, observation)
(Arp *et al.*, 2014; Feizollah *et al.*, 2017; Kang *et al.*, 2015; Wu *et al.*, 2012; Yerima, Sezer, & McWilliams, 2014)	• Search in overall files
(Talha *et al.*, 2015)	• According to the complete lists as guidance
This thesis, (C.-Y. Huang *et al.*, 2012)	• In overall files including manifest • According to the complete lists as guidance

In the comparison of the overall features, it shows that none of the previous studies shown had computed similar features which were repeated multiple times in each application. In addition, previous studies did not calculate the repeated features extracted from both the malware and benign applications so as to obtain the relevant features. Keeping in mind these limitations, advantages and opportunities in selecting features through static analysis and machine learning, the following section proposes the key focus areas to overcome the weaknesses identified.

3.4 Proposed study key focus areas

Based on the issues in previous studies, the proposed study is to conduct an evolutionary algorithm to genetically search the features, conveys the range of repeated features in similar application to gain the best features and investigate the features specifically on root exploit.

3.4.1 Evolutionary algorithm in selecting features

Features refer to the elements or characteristics which are applied to mark an application as being malware or benign. Machine learning classifiers normally use these features as an input to make a decision. Minimum features are desirable because they offer enhanced accuracy (i.e. accurate predictive model) with fewer data; minimum features also reduce the complexity of the detection model by decreasing noisy and irrelevant data (Feizollah *et al.*, 2015; Sarip *et al.*, 2016; Zia *et al.*, 2015). The current study adopts an evolutionary algorithm method called Genetic Search (GS) which is based on Genetic Algorithm (GA), to search and improve the parameters so as to provide minimal quantity of features in multiple categories of features.

GA is an algorithm that mimics the natural evolutionary process which consists of the crossover process that combines multiple generations. It then continues to loop until the best generations are achieved. Frohlich *et al.* (2003), Middlemiss and Dick (2003),

Punch *et al.* (1993) and Stein *et al.* (2005) applied the GA to achieve the best parameter of features for their detection algorithms when trying to increase accuracy. However, as mentioned in Section 3.1.6, none of these studies had adopted the GA method in selecting features using static analysis and machine learning in detecting Android malware. This gap will be filled by the current study which adopts the GS to perform a search based on the GA as described by (Goldberg & Holland, 1988). Figure 3.6 illustrates the basic GA processes which consists of the crossover and mutation.

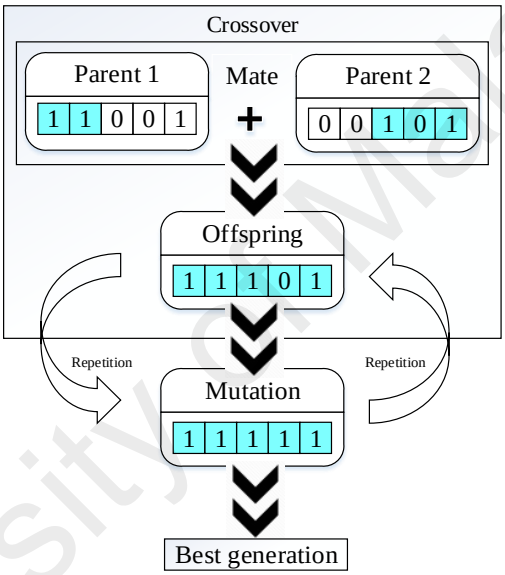


Figure 3.6: Basic GA process

To further illustrate the GA, the following analogy is offered. Consider a company requiring advice in designing a good vehicle characteristic that is able to enter and successfully exit from a high hill forest that is filled with obstacles. In this case, the fitness is set according to the multiple checkpoints noted in the high hill forest. First, the vehicle consists of small tires that have low performance engine. The vehicle keeps going and keeps changing the characteristics or features simultaneously. Once the vehicle has successfully reached the first checkpoint, the first fitness is satisfied and the characteristics of the vehicle are saved. The GA process then continues (the characteristics of the vehicle are gradually changing) until the vehicle achieves all the checkpoints (save all the required characteristics). Finally, the set of features of a

vehicle that is necessary to enter and survive the forest is successfully achieved and this includes the big tires, the high acceleration engine, the big tank and good quality brakes. The GA equation (refer to 3-5) is defined according to the f function and features string of length 1 known as fitness. The new features are created from the current population and the probability that a parent string Hj is selected from the N strings such as H1, H2, H3 until HN appears in the system equation is illustrated in the equation below.

$$(Hj) = f(Hj) \div \sum_{n=1}^{N} f(Hn) \quad 3\text{-}5$$

Apart from the GS, another method for selecting the best features in detecting malware is also explained.

3.4.2 Repeated features in similar application

In the aim to discover which features are frequently used by malware, the current study inspects the existence of each feature in both the malware and benign applications. It also inspects similar features which are repeatedly utilized in each application. Accordingly, this method calculates the frequency of the similar features that exist, do not exist as well as those features which are repeatedly used in similar samples. To obtain the results showing which features in malware are frequently used, the frequency identified between the malware and benign applications are subtracted so as to obtain the range. The subsequent section explains how root exploit is detected.

3.4.3 Root exploit

It is important to investigate the exclusive features that are best suited for machine learning classifiers in detecting unknown root exploit. Hence, investigating features to detect root exploit is included in this section as the key focus area.

3.5 Summary

This chapter has addressed the challenges faced in selecting features through static analysis. These challenges are further summarized as follows:

a) **Minimal features:** There is a need to identify and select minimal amount of features in detecting malware. This is because excessive features may weaken the machine learning phase and confuse the learning algorithms and thereby, causing the classifiers to over-fit the training data.

b) **Investigation and method in selecting features:** Using static analysis to search for malware features that reside in a thousand lines of code located in each application are sophisticated, difficult, and complicated. Thus, there is a need to identify an approach that select minimal amount but the best features in detecting malware.

c) **Complete list of features:** Many studies have investigated features by extracting their existence either manually or inspecting them without referring to a reliable and complete list of reference. Therefore, there is a need to refer complete list of features to prevent any significant features that are capable of detecting malware from being missed.

d) **Search features in overall files:** Although thousands of applications in the Androidmanifest.xml contain zero permission, many studies seem to be extracting their data from only one particular file (i.e. AndroidManifest.xml) to detect malware. As this may increase the false positive value, thus, it is important to investigate the features that reside in all the files located within an application including the Androidmanifest.xml file.

e) **Root exploit features:** From literature review, it appears that lesser attention has been given to the investigation of specific features using static analysis and machine learning to detect root exploit. Thus, it is important to specifically investigate the multiple categories of features on root exploit.

As a conclusion, this chapter has highlighted some related and important issues and some strategies to reduce the limitations. Addressing the limitations and the advantages is important because the outcomes of this study can be used as a guideline to construct a valuable framework or to further improve existing studies and their methods used. Furthermore, the outcome may be used to enhance the anomaly-based malware detection through static analysis for the intrusion detection system.

CHAPTER 4: ANOMALY-BASED DETECTION USING STATIC ANALYSIS: THE FRAMEWORK

This chapter entails the methodologies developed for the proposed framework. The aim of the framework is to select the best features in minimal amount for the anomaly-based detection using static analysis with the selected methods. In order to have an effective anomaly-based machine learning prediction in static analysis, the process in selecting the minimal features is crucial. To support the process, this chapter details the strategies in Genetic Search (GS), range of repeated features and investigation on root exploit features.

GS is based on Genetic Algorithm (GA). It is a bio-inspired mechanism inspired by biological evolutionary concepts. This experiment used it to select a minimal number of features in multiple categories of it to detect malware using machine learning predictions.

The following strategy is an effort to inspect the best features by considering the range of repeated features in each application, including benign and malware. This range algorithm is inspired by the existence of similar feature multiple times in each application.

Another strategy is to investigate which features are the best to detect particularly on root exploit. It is a type of malware which capable of evading detection and escalate privileges (Li & Clark, 2013). Hence, there is a need to conduct an experiment to investigate and identify exquisite features to detect unknown root exploit malware. Next section discusses all the methods in selecting features in detail.

4.1 Methods and categories of features

Table 4.1 tabulates the methods in selecting features for the best in minimal amount. It also includes the categories of features and techniques that include in the experiment. In order to observe different results, each experiment has different approaches. Table 4.1 provides the detail regarding this information.

Table 4.1: Methods and techniques for the proposed framework

Experiment	Method in selecting features	Categories of features and technique involved			Machine learning classifier categories
1	a) Genetic Search (GS)	a) Permission b) System command c) Directory path d) Code-based	Search features in overall files		Bio-inspired, tree and bayes
2	a) Range of similar features in same application b) Information Gain (IG)	a) Permission b) Directory path c) Telephony	Search features in overall files	Refer features from the complete list	Bio-inspired
3	a) Frequency investigation b) IG	a) System command with Android Debug Bridge (ADB) b) Directory path c) Code-based	Search features in overall files		Bio-inspired, tree and bayes

4.1.1 Genetic Search (GS)

In order to discover the efficiency of the features selected by the GS genetically, this first experiment only adopts GS method alone without the aid of additional methods. Furthermore, this experiment covered four categories of features, which exceed than other experiment. This is to facilitate the GS to have a wide of choices of features during the evolutionary process.

4.1.2 Range of repeated features

In second experiment, this research subtracts the frequency of the repeated features in both malware and benign. This is to obtain the features of malware frequently used compare to benign. Next, this experiment applies the range of the features according to

the equation below. In order to obtain the $Range$, the frequency of repeated features in malware (fm) subtracts the frequency of repeated features in benign, fb. This algorithm then applied on all the categories of features in the experiment. In addition, this research is the only experiment that refers to the reliable list as a guidance to explore the differences between the other experiments that execute without it.

$$Range = fm - fb \quad (4\text{-}1)$$

4.1.3 Root exploit features

As this experiment focuses on selecting the features to detect specifically on root exploit, there is a need to add the novel features such as ADB. Furthermore, the other addition categories of features (i.e. system command, directory path and code based) are also included to identify the need of these categories in root exploit. Although, these features are already included in other experiment in this thesis, however, this research attempt to discover how much critical the root exploit use these features compare to other types of malware. As an example, in directory path category, root exploit utilized certain directory path (i.e. /system/xbin/su) more than other types of malware. Therefore, this experiment is conducted to answer this question. Next section provides the proposed framework that depicts all the proposed methods and techniques.

4.2 Anomaly-based detection using static features framework

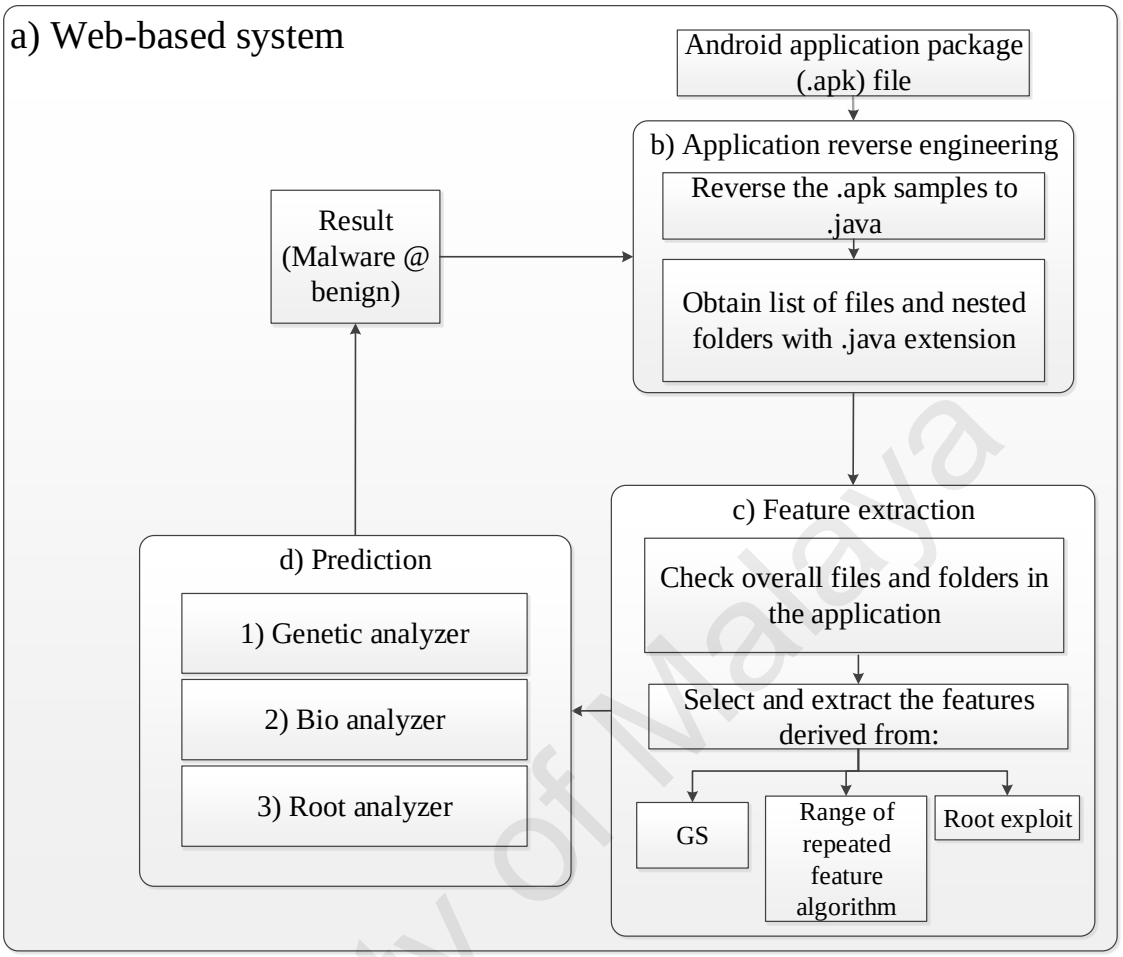


Figure 4.1: Anomaly-based Detection Using Static Features Framework

The framework comprises of four main elements as follows:

a) **Web-based system**: This is the first part in framework that consists of application reverse engineering, feature extraction and prediction. It provides the graphical user interface functions to facilitate the end-users to predict their desired .apk files.

b) **Application reverse engineering:** This is the second part of the proposed framework and it aims to convert the Android application package (.apk) to .java extension files to retrieve the code. This is done by reverse engineering the .apk file and subsequently obtains all the files and folders.

c) **Feature extraction:** This is the third part in the framework that extracts the features from the application. These features are derived from the following strategies:

i. **Evolutionary method:** It is a method to search the best features by adopting bio-inspired GS. Initially, this research collects all the malware and benign features and the GS algorithm search the best generation of features from the collection.

ii. **Range of repeated features**: As many features are repeatedly used in similar application, this research takes this opportunity to calculate how many features have been used in each application in both categories, malware and benign. By scrutinizing the frequencies of these repeated features, this research capable of identifying the features that malware frequently used with this novel method.

iii. **Root exploit investigation**: In order to discover the best features that root exploit used, this research inspects this type of malware by investigating multiple categories of features including the novel Android Debug Bridge (ADB).

d) **Prediction:** In fourth part, the system used these features as input for the machine learning classifiers to predict the class of the .apk file either malware or benign. As to discover the different results in different machine learning types as well as frequently and infrequently used in past research effort, the type of machine learning classifiers involved are function, tree, and bayes. More information can be obtained in Table 3.1 in Section 3.1. In addition, the final selection of machine learning classifiers will be selected according to the evaluation on the following section, which is Section 5.

4.2.1 Operational Characteristics

The proposed framework offers the following operational characteristics:

a) **Multiple strategies:** The proposed framework applies multiple strategies in selecting features in malware detection process as well as thoroughly searches the

features in all files in each application. By implication, minimal features and deep searches allow a comprehensive detection result.

b) **Reduce the complexity of machine learning detection:** The used of methodical approaches (i.e. GS and range algorithm of repeated features) in selecting the best features in minimal amount reduces the complexity of machine learning predictive model. It allows the anomaly-based detection to confront the effect of dimensionality to produce an efficient prediction score.

c) **Rapid detection process:** The proposed framework adopts the static analysis method that decreases the prediction process and further achieves much faster detection. This type of analysis is significantly faster because it classifies the application without executing them.

d) **User friendly interfaces:** The proposed framework allows a simple prediction of the results of the framework by providing a friendly graphical user interface system. Moreover, it also allows the security analysts to thoroughly assess the features in CSV and ARFF files from the prediction results to conduct further analysis.

e) **Lightweight:** the proposed method adopts static analysis technique, where only reverse engineers the application and retrieves the code. It is done without execute and monitor the behavior of the application.

f) **Intelligent:** this methodology employs machine learning prediction to detect malware, instead of signature-based method that needs to continuously update the malware signature in the database.

4.3 Summary

This chapter has focused on the conceptual framework for selecting the best features by conducting investigations and methods for anomaly-based machine learning in detecting malware. This is to increase the efficiency in malware detection process. Its description has included an introduction of the main models, strategies, frameworks and the

rationale behind their implementation, as well as their operational characteristics. In conclusion, this chapter highlighted the main point of this study and gave the detail of the framework. It is important to understand the interrelationship between those strategies and the model in compiling the overall process in detecting malware in order to achieve an outstanding result from this process.

Having established the proposed framework using multiple strategies and methods, the next chapter presents the methods of the experiment, evaluations of the framework followed by a detailed discussion of them. It is important to understand that the results provide a verification of the usefulness and suitability of the framework in facilitating the anomaly-based static analysis in detecting malware process.

CHAPTER 5: EVALUATION OF THE ANOMALY-BASED DETECTION FRAMEWORK

The novelty of this study is to propose a framework to have the mobile malware anomaly-based detection systems by conducting experiments to identify the best features, in order to facilitate the static analysis detection. Thus, in order to highlight the feasibility and suitability of the framework, this evaluation study is significant.

This chapter provides the multiple experiments and the evaluation part based upon the proposed framework that aims to evaluate it in terms of its effectiveness and performances in relation to the static features selected. It is important to this evaluation study to investigate the effectiveness and performance of the proposed framework in order to satisfy its feasibility and suitability, in particular the ability of the framework to facilitate anomaly-based malware detection.

The first experiment investigates the feasibility of the features selected from the evolutionary GS search method. To evaluate and compare with the results from other studies, the first experiment analyses the effectiveness of the GS process. With the first experiment results, the second experiment extends the evaluation study by analyzing the effect of adopting the novel range of repeated features and referring to the official list as the main source, in order to satisfy the process enhancement. Furthermore, instead of detecting all types of malware, the third experiment investigates and evaluates the suitability of using the proposed features, especially for root exploit. The third experiment also evaluates the categories of features that root exploit frequently used only, as opposed to the features applied in the first and second experiments that detects general types of malware. Finally, the chapter concludes with a summary.

5.1 General Description

This chapter provides the results of the evaluations and each with its own set of results, discussion, and conclusion. Nonetheless, they do share a similar requirement in their experimental procedures. This section discusses the similarities in order to avoid any repetition in this chapter. In conducting the evaluation measure, the following section provides the description of the dataset used in the following experiments.

5.1.1 Dataset

Primarily, the dataset is a collection of related data to initiate the experiment in the initial phase. It consists of all the samples required for research activities. Typically, in malware detection study, there are two types of classes involved as the dataset, which are benign (also known as normal) and malware applications.

a) Benign Dataset

Benign, also known as normal or clean, is a status of an application that contains non-malicious activities. It is considered safe for the user to utilize and install it in mobile devices for daily purposes. The following subsection provides the benign dataset applied in this study.

i) **Google Play store:** This thesis used the benign dataset obtained from Google Play store market (Google, 2014). This market provides Android applications of various categories, such as business, books, comics, communication, education, entertainment, family, lifestyle, medical, music, shopping, transport, tools, and social. These applications are provided for users of Android-powered phones, tablets, and Android TV devices. In order to ensure the applications are in pristine condition, this study included the samples which are only available from the Google Play store. The reason for this is that, Google introduced Bouncer, a security service that scans the application, its developer account, reputation engine and cloud

infrastructure automatically. Google is continuously improving and updating the Bouncer detection system and is responsible for dropping the number of malicious applications by about 40% in the Google Play store (Hou, 2016). In addition, this study also conducted a scan in VirusTotal (VirusTotal, 2016). It is an online website which provides button for users to upload their Android package file (.apk) file and retrieve the results from more than 50 online antivirus results, either it malicious or non-malicious. This experiment uploaded the benign applications in VirusTotal and only included applications that received 0/50 scores, which indicates that 50 antiviruses claimed the .apk is clean or non-malicious.

ii) **Androzoo:** In the benign dataset collection phase, this study downloaded 7000 benign applications from Androzoo (Allix *et al.*, 2016). It is the dataset searched by the University of Luxembourg which contains 5 million Android application files from the several markets. There are Google Play store (Google, 2014), Anzhi (Anzhi, 2017), AppChina (AppChina, 2017), 1mobile (1mobile, 2017), AnGeeks (Angeeks, 2017). Slideme (Slideme, 2017), FreewareLovers (Freewarelovers, 2017), ProAndroid (ProAndroid, 2017), HiApk (HiApk, 2017) and F-Droid (F-Droid, 2017). Among all of these markets, this thesis only included application from Google Play store. This is to ensure the applications are in pristine condition. The reason is, Google introduced Bouncer (Hou, 2016) to double check the clean status of applications. Furthermore, this thesis also examined the dataset in VirusTotal to confirm its benign status. After the benign dataset, the following section describes the malware used in this thesis.

b) Malware Dataset

The word *malware* is an abbreviated term for "malicious software." Unethical authors design an application known as malware for harmful purposes, such as damaging the operating system of computer and gaining access to steal private data without the user's

consent. The following subsections present the information of malware used in this thesis.

i) **Malgenome:** As the experiments require malware dataset, a publicly available dataset called Malgenome is thus utilized and extracted, which contains 1,260 samples. These samples consist of 49 different malware families (Y. Zhou & Jiang, 2012b) and have been used in many studies (Afifi *et al.*, 2016; Feizollah, Anuar, *et al.*, 2013; Narudin *et al.*, 2014). The identification includes several malware types, such as botnet, root exploit, and Trojan.

ii) **Drebin:** Another malware dataset is Drebin (Arp *et al.*, 2014). It is the experiment collaboration in University of Gottingen and Siemens, Germany. They obtain the malware by conducting static analysis whereas the features are embedded in a joint vector space to identify the application as either malware or benign. The overall total amount of Drebin is 5560, which consists of 179 different families. The subsequent section provides the information of the tools utilized throughout the experiment in this thesis.

5.1.2 General tools

This section provides the information of the tools used throughout the experiments in this thesis. The tools involved are Jadx, Weka, Apktool and R.

i) **Jadx:** In order to gain the application code of the dataset, Jadx (Skylot, 2015) tool reverse engineered the Android .apk files. It is used to reverse the .dex file in .apk Android file to .java extension file. Once the process is accomplished, this study obtained all the files in the nested folders which consist of .xml and .java file extensions. The common file found after the reverse process is Androidmanifest.xml, and the common folder is called res. The Androidmanifest.xml file contains the Android permission, intent filters, libraries,

and the folder res contains the application layout with various .xml file types. The remaining folders contains .java file types, in which the file names differ depending on the application. This thesis chose Jadx for this task because it provides the deobfuscation option that performs best in handling the obfuscated code with minimal error.

ii) **Weka:** It is an acronym which stands for Waikato Environment for Knowledge Analysis. It is the machine learning platform for machine learning (Hall *et al.*, 2009). Figure 5.1 depicts the Weka logo which represents the bird found in New Zealand, which is also called Weka.

Figure 5.1: The logo of Weka

Weka is released as an open source application that provides a graphical user interface (GUI) and a command line interface. The GUI facility eases the user to complete their machine learning projects easily. Meanwhile the command line interface features is very useful for scripting projects. Moreover, it is written in Java and provides an API that is well documented and promotes integration into the user's application.

iii) **Apktool:** It is a reverse engineering application that decodes the Android application package (.apk) file to nearly original form (.java) and able to rebuilds it to .apk back after certain modifications have been made (Wiśniewski, 2015).

iv) **R:** R is an open source programming language for statistical computing and graphics that is supported by the R Foundation for Statistical Computing. It is generally utilized among statisticians and data miners to conduct data analysis and

statistical applications. It is capable to conduct numerical computations through additional packages that are available (Chambers, 2017).

This section has defined the dataset (i.e. malware, benign) and general tools used for the experiments in this thesis. The next section provides the evaluation measure that is used in the experiment in this section.

5.1.3 Evaluation measure

In order to evaluate the effectiveness of the features, this thesis assessed the performance matrix of the machine learning classifiers. Table 5.1 lists each evaluation in terms of accuracy, True Positive Rate (TPR), recall, precision, f-measure and False Positive Rate (FPR). It further lists the benchmark performance evaluation and its descriptions. The following section explains the experiment that involve the dataset, tools and evaluation measure in detecting malware

Table 5.1: List of evaluation measure

	Evaluation measure	Descriptions	Equation
Higher value indicates better performance	Accuracy	Correctly predicts instances as either malware or benign	$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$
	True Positive Rate (TPR)	Correctly predicts instances as malware	$TPR = TP/(TP + FN)$
	Recall (similar to TPR)	Measures the algorithm performance in identifying malicious samples	$Recall = TP/(TP + FN)$
	Precision	Measures whether the prediction is true or otherwise	$Precision = TP/(TP + FP)$
	F-measure	Measures the weighted harmonic mean of precision and recall	$F - measure = \frac{2 \times precision \times recall}{precision + recall}$
Lower value indicates better performance	False Positive Rate (FPR)	Incorrectly predicts the sample as malware, when it is actually benign	$FPR = FP/(FP + TN)$

5.2 Evaluation of Genetic Search (GS)

This section provides the evaluation study that investigates the use of GS in selecting the features genetically and to satisfy the following points:

a) To propose an evolutionary method known as GS to select the best features in minimal amount.

b) To validate the results of GS, this thesis compared it to existing anomaly-based experiments that utilized static features.

5.2.1 Experiment and procedure description

This section presents the flow of the experiment. Figure 5.2 illustrates the four phases experiment process. The beginning phase is data collection, which includes the reverse engineering process that retrieves the code of the application. The following phase is string identification, which consists of permission, the words in the double quote, function, intent, Linux command, directory path, and system command. The third phase applies the GS process to select the best features among all the extracted strings obtained in the strings identification phase. Subsequently, the fourth phase involves the machine learning classifier, whereas the machine learning classifier trains the information in the dataset to construct a detection model which is capable of predicting an application either benign or malware. The following subsections describe each phase in detail.

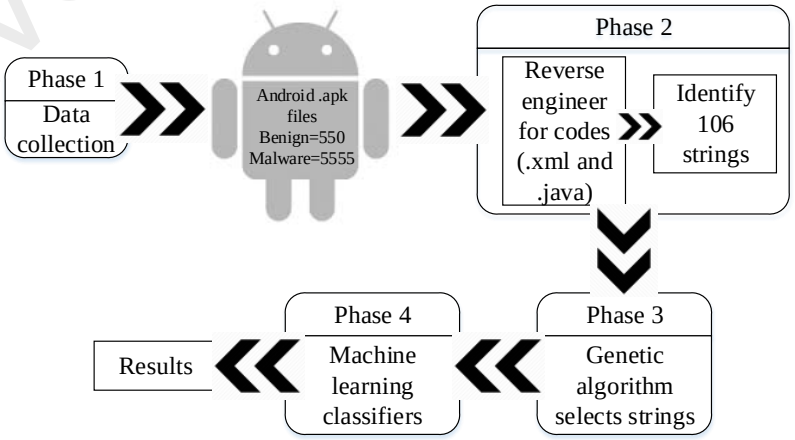


Figure 5.2: Structure of the experiment phases

a) Data collection

This section explains the data collection phase, which is the initial step of the experiment. Table 5.2 lists the summary of the dataset in the experiment. Followingly, a reverse engineering tool called Jadx (Skylot, 2015), reverses engineered the Android .apk files to obtain the code of the application.

Table 5.2: Dataset summary

Dataset	Source	Total downloaded	Total used in the experiments
Malware	Drebin	5560	5555
Benign	Google Play store	1209	550
		Total =	6105

b) Strings identification

After obtaining the code of the application, the following step is to investigate and to identify the strings for features. Figure 5.3 illustrates the second phase for strings identification. To differentiate the colon, quote, and bracket in the code, Natural Language Toolkit (NLTK) (Bird *et al.*, 2009) tokenizes the line and pull out the strings. Afterward, the existence of each feature (where 1 indicates existence and 0 indicates as non-existence) in both benign and malware takes place. In some cases, the strings are confused with one another. For example, *cat*, one of the Linux commands, is confused with other words such as concatenate, locate, and other similar words which contain *cat* strings in it. In the interest to observe the entire exact line of codes and to pull out the *cat* command, this part used the grep command in Ubuntu terminal. It is a command provided for the Linux platform.

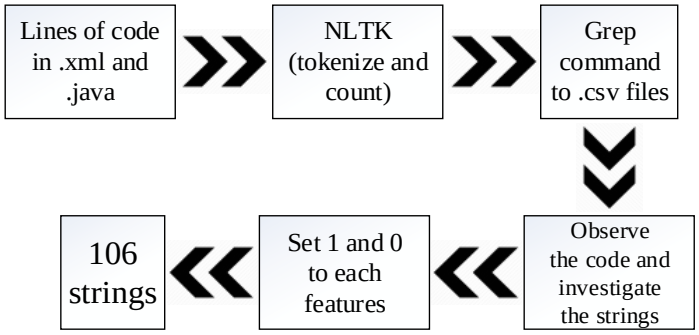


Figure 5.3: Strings identification

Other than the Linux commands, the features listed in the next section are selected from other studies and additional investigation from this experiment. The permission features consists of 13 dangerous permissions declared in the Androguard (Desnos, 2015) open source tool. Furthermore, the additional features are the proposed set in the root exploit study (Firdaus & Anuar, 2015) that consists of the system command, directory path, and code-based. This is included due to the reason that this study attempts to detect all malware types including root exploit. It is among the malware types that exploit the vulnerable Android kernel to gain root and to further execute malicious actions such as installing botnet, spreading Trojan, providing fake antivirus results, and executing root privileges.

i) **Code-based**: Table 5.3 lists the 36 code-based features along with their existences in both classes (malware and benign). For instance, createSubprocess is a code string exists in 307 malware samples and does not appear in benign samples. As the malware dataset is 5,555, and benign is less than malware which is 550; therefore, it is necessary to calculate the percentage by dividing the existence, and multiplying it by 100. Figure 5.4 depicts the code-based features in percentage form. The vertical red line in the middle significantly distinguishes the benign class with only three malware features included within the range of 50% to 100%.

This reveals that benign features exceed the malware existence in the code-based category.

Table 5.3: Code-based features

No	Code-based	Existence		No	Code-based	Existence	
		Malware	Benign			Malware	Benign
1	android.util.Log	4607	543	19	get(os.version)	531	5
2	android.os.Handler	3878	539	20	getDefaultHost()	489	38
3	DefaultHttpClient	3413	469	21	createSubprocess	307	0
4	setReadTimeout	2719	406	22	com.google.update.RU.U11	272	0
5	UrlEncodedFormEntity	2589	311	23	Forked	95	1
6	getResourceAsStream	2386	261	24	ImageTestActivity	74	0
7	getNetworkInfo	2157	348	25	InstallApk	73	1
8	vnd.android.package-archive	2034	90	26	ACTION_CostInfo	72	0
9	getExternalStorageState	1954	438	27	ACTION_SaveID	72	0
10	checkPermission	1893	428	28	AndroidThemeService	72	0
11	Cipher.getInstance	1738	408	29	CostNumberConfirm	72	0
12	Math.random	1590	346	30	CostParms	72	0
13	getLaunchIntentForPackage	1413	284	31	getResponseContentGet	72	0
14	getSimSerialNumber	1363	36	32	getResponseContentStream	72	0
15	setWifiEnabled	772	46	33	KeyCostTips	72	0
16	process.waitFor	738	4	34	KeyPIDIntent	72	0
17	getField(MANUFACTURER)	641	29	35	TASK_Network_Setting	72	0
18	getWifiState	559	25	36	config.dat	72	3

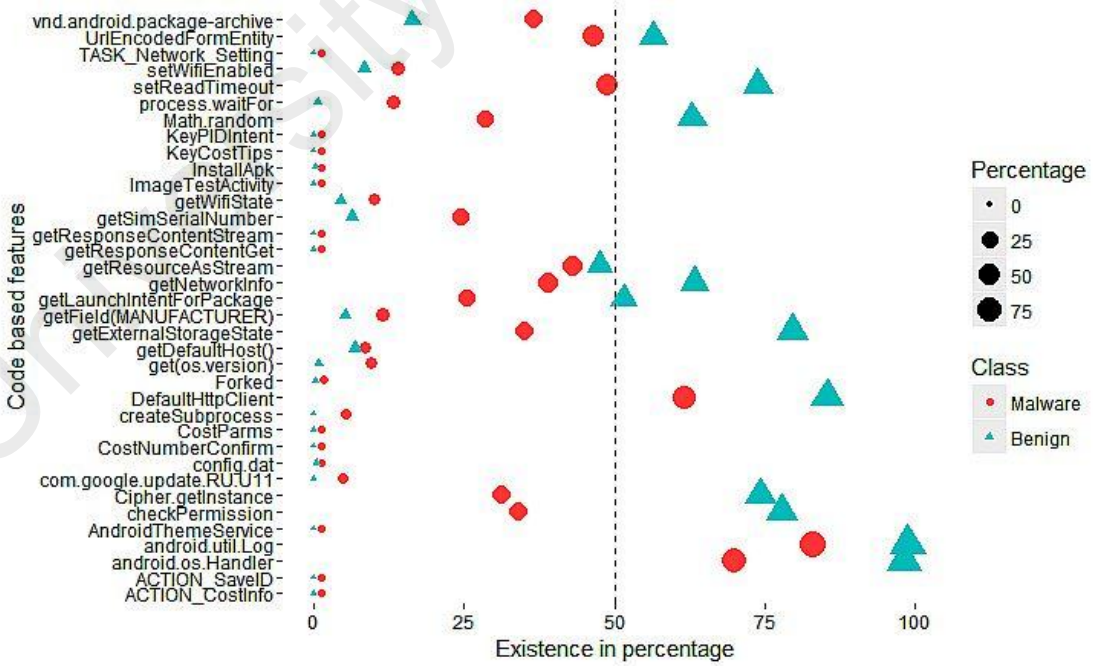


Figure 5.4: Code-based in percentage values

ii) **Permission:** The second category is the permission features, included in the AndroidManifest.xml file. It is a file required in every Android application main directory. It represents the essential information regarding the application package name, permission, activities, services, broadcast receivers, and content providers. Table 5.4 lists the permissions included in the experiments before the GS feature selection phase. For example, android.intent.action.MAIN is a permission that exists 5,355 and 541 times in the malware and benign samples, respectively. Some of these permissions were taken from the Androguard (Desnos, 2015) application which is declared as dangerous, whereas the others are added from this investigation. Figure 5.5 depicts the existences of these features in percentage form. The plot in the graph shows that both malware and benign classes are placed in the similar area from 0% to 100%, in which the areas are similar yet insignificant to each other. Meanwhile, the next section presents the directory path features.

Table 5.4: Permission features

No	Permission	Existence		No	Permission	Existence	
		Malware	Benign			Malware	Benign
1	android.intent.action.MAIN	5355	541	22	android.intent.extra.shortcut.INTENT	1352	66
2	android.content.Context	5320	548	23	android.intent.extra.shortcut.NAME	1352	66
3	android.permission.INTERNET	5234	527	24	android.permission.READ_CONTACTS	1314	59
4	android.intent.category.LAUNCHER	5150	530	25	android.permission.WRITE_SMS	1213	6
5	android.telephony.TelephonyManager	4902	481	26	com.android.launcher.action.INSTALL_SHORTCUT	1197	52
6	android.intent.action.VIEW	4844	544	27	android.permission.CHANGE_WIFI_STATE	983	85
7	android.permission.READ_PHONE_STATE	4838	388	28	android.intent.action.SCREEN_ON	912	107
8	android.permission.WRITE_EXTERNAL_STORAGE	3644	447	29	android.intent.extra.shortcut.ICON_RESOURCE	762	55
9	android.intent.action.BOOT_COMPLETED	3539	143	30	android.intent.action.SIG_STR	671	1
10	android.webkit.WebView	3453	494	31	android.intent.action.BATTERY_CHANGED_ACTION	581	0

11	android.content.IntentFilter	3077	504	32	com.google.update.UpdateService	406	0
12	android.permission.SEND_SMS	2976	72	33	com.google.update.Receiver	398	0
13	android.webkit.WebSettings	2425	396	34	com.android.packageinstaller	335	5
14	android.permission.RECEIVE_SMS	2127	17	35	android.permission.READ_EXTERNAL_STORAGE	323	82
15	android.permission.ACCESS_COARSE_LOCATION	2119	297	36	com.google.map.apk	275	0
16	android.permission.ACCESS_FINE_LOCATION	2109	306	37	android.intent.action.NEW_OUTGOING_CALL	241	3
17	android.permission.WAKE_LOCK	2098	334	38	android.intent.extra.PHONE_NUMBER	173	3
18	android.permission.READ_SMS	2037	12	39	android.provider.Telephony.WAP_PUSH_RECEIVED	173	2
19	android.intent.action.DIAL	1729	184	40	android.settings.WIRELESS_SETTINGS	158	46
20	android.provider.Telephony.SMS_RECEIVED	1446	24	41	android.provider.Telephony.MMS_RECEIVED	72	0
21	android.intent.action.SCREEN_OFF	1407	286	42	com.android.browser.application_id	45	28

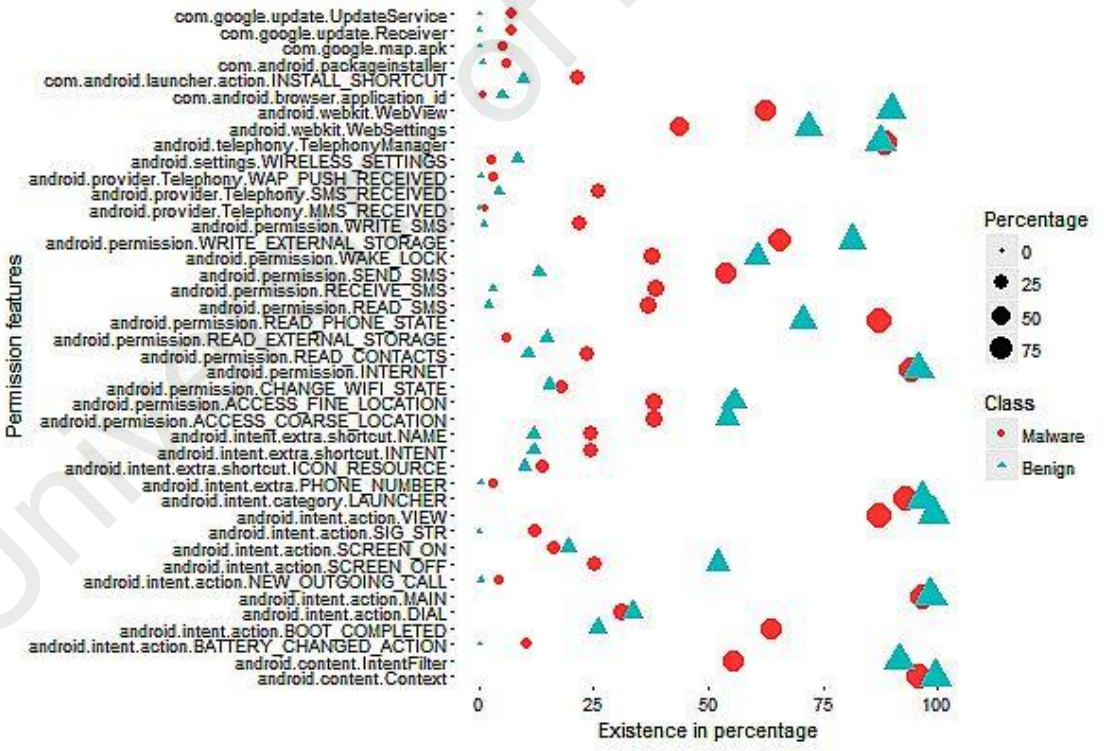


Figure 5.5: Permission features in percentages

iii) **Directory path:** Table 5.5 lists the directory path features included in this experiment, in which some paths are taken from the root exploit study (Firdaus & Anuar, 2015) and some added from additional investigations. One of the

directories, system/bin/su, exists 633 times in malware and none in benign samples. It stores the super user information of the Android OS and attracts unscrupulous authors to gain root in the victim's mobile devices. Figure 5.6 shows the percentage of the directory path features. The figure depicts the critical directory, where system/bin/secbin appears 44% in the graph, leaving the other features.

Table 5.5: Directory path features

No	Directory path	Existence	
		Malware	Benign
1	system/bin/secbin	2434	0
2	system/bin/su	633	17
3	content://telephony/carriers/preferapn	625	19
4	system/xbin/su	621	38
5	system/bin/chmod	516	12
6	system/etc/dhcpcd	449	0
7	system/etc/rild/cfg	441	0
8	content://telephony/carriers	376	3
9	system/bin/sh	287	2
10	application/vnd.wap.mms-message	280	7
11	application/vnd.wap.sic	151	3
12	DES/CBC/PKCS5Padding	134	7
13	system/bin/mount	41	0
14	data/local/tmp/rootshell	24	0
15	system/bin/rm	23	0
16	system/bin/profile	19	0

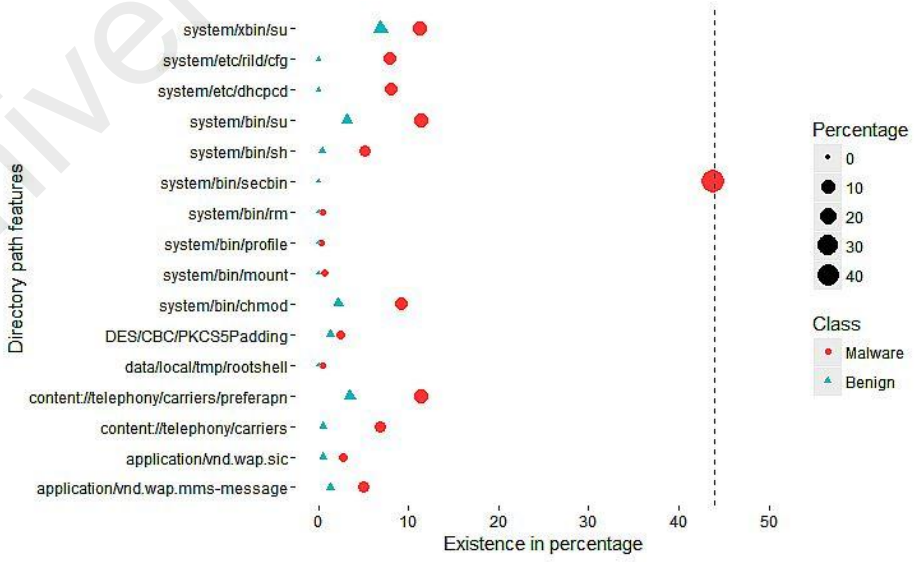


Figure 5.6: Directory path features in percentages

iv) **System command:** The final category is the system command feature listed in Table 5.6. As the objective is to detect all types of malware including root exploit, this section included the system command from the root exploit study (Firdaus & Anuar, 2015). These features originated from two categories, namely, the Unix command and the ADB commands. Examples of Unix commands are chmod, chown, pm install, cat, cp -rp, and mount -o remount, and ADB commands include startservice -n and adb_enabled. Figure 5.7 displays these commands in percentage form. The commit command has the most existence in both benign and malware classes.

Table 5.6: System command features

No	System command	Existence	
		Malware	Benign
1	commit	4529	536
2	chmod	1034	49
3	startservice -n	474	0
4	buffer.listFiles	272	0
5	buffer.mkdir	272	0
6	mount -o remount	247	2
7	chown	195	0
8	pm install	183	0
9	stdin	179	8
10	cat	123	1
11	adb_enabled	80	2
12	cp -rp	72	0

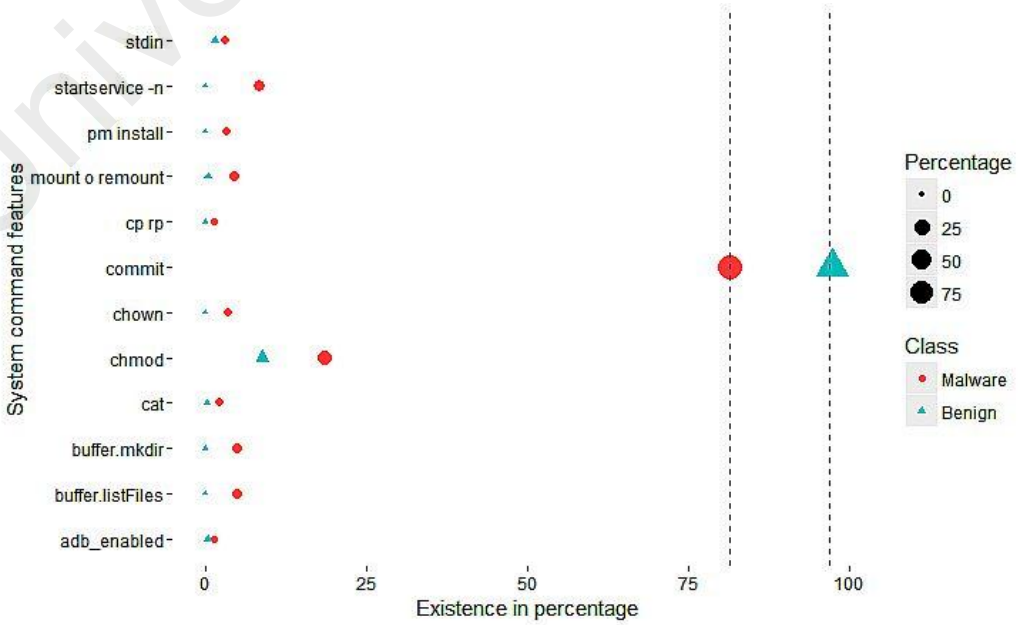


Figure 5.7: System command features

Figure 5.8 shows the regression line to discover the relationship of significant features in the malware and benign samples. The three lines (indicated as code-based, permission, and system command) that rise from below to the top show a positive relationship. The lines indicate that whenever benign features increase, malware and benign expand as well. This finding proves that the malware features are suitable attributes for detecting malware. However, the line of directory path category is minimal, by reason of percentage is small compared with other categories. Thus, in the succeeding section, the GS method search which features is the best among all the features in code-based, directory path, permission and system command categories.

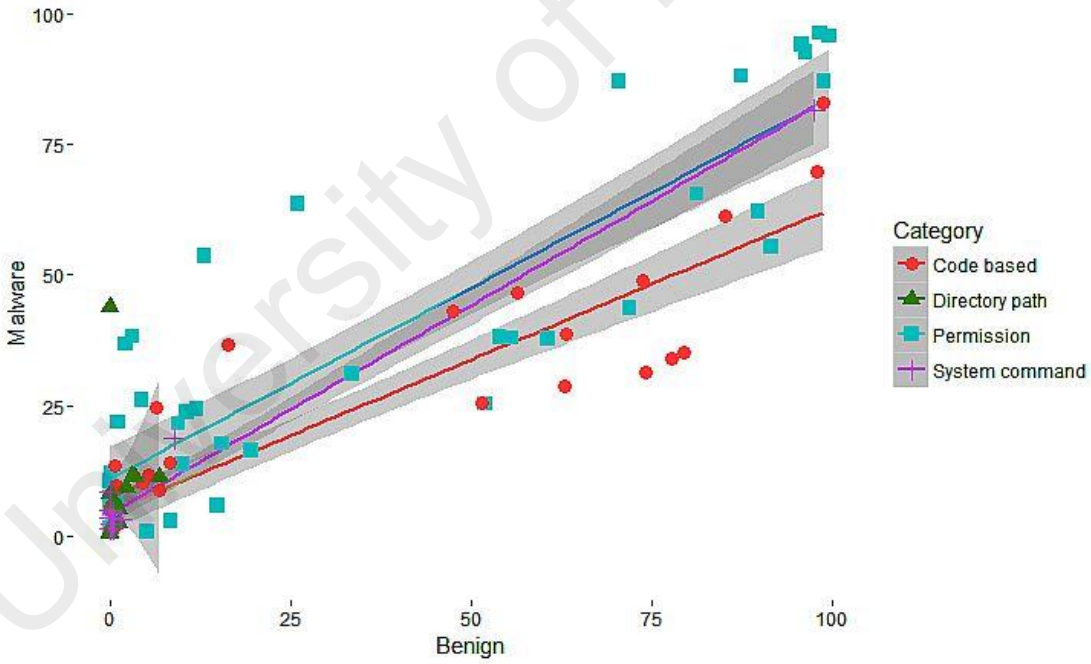


Figure 5.8: Regression lines of all categories

c) Proposed GS and features

The GS is based on the GA method which is inspired and based on the evolutionary biology in nature (e.g. crossover and mutation). This evolutionary process provides a technique to automatically improve characteristics or features in order to generate the

best generation. The evolution is able to transform the worst generation to form a better generation. This is done by referring to the fitness requirement. This method includes a set of requirements, which is when one generation is unmatched according to certain characteristics, the GA excludes that generation and this process repeats until the best generation is achieved. Furthermore, the GA is unable to solve any research problem by providing a final ultimate solution; instead, it serves a method to select at least the optimal features.

Hence, the next process is to select the best features among the 106 features mentioned in the previous section. The GS process takes place in the Waikato environment knowledge analysis (Weka) (Hall *et al.*, 2009) tool. As the total dataset consists of 6,105 applications (both benign and malware), the parameter settings are 400 for populations in each generation, which amount to a total of 15 generations. For crossover probability, it is set as 0.6, which crossover the 106 features to produce the offspring list of features. To maintain the value of the features (0 and 1) in each application, this study set zero for the mutation process. At the end of the process, GA successfully produces the best generations of features and selected the six features as listed in Table 5.7.

Table 5.7: Six GS-selected features

Features	Existence in malware (%)	Existence in benign (%)
android.permission.READ_SMS	36.7	2.2
android.permission.RECEIVE_SMS	38.3	3.1
android.permission.WRITE_SMS	21.8	1.1
checkPermission	34.1	77.8
system/bin/sec/bin	43.8	0
com.android.browser.application.id	0.8	5.1

These features are included in three types of categories; first is android permission (read, receive, and write SMS), the second is code-based (checkPermission and com.android.browser.application.id), and the third is directory path (system/bin/secbin). In Table 5.7 the existence of malware in permission and directory path is more than

benign. Nevertheless, it is different for the code-based category, with the malware existence found to be fewer that of benign. This demonstrates that GS search the optimal features not according to malware existence; merely it is according to the evolutionary process in GS. Subsequently, in order to evaluate the effectiveness of these features, this study used three types of machine learning classifiers in the following section (bayes, trees, and function).

d) Machine learning classification

In constructing the machine learning model, the classifiers are run in the Weka (Frank *et al.*, 2016). In this tool, it is fundamental to prepare a Comma Separated Values (.csv) file. As GS selected the six features and addition to the class label, this file therefore, contains seven columns. Furthermore, this file contains 6105 lines of rows, which represents the total of both malware (5555) and benign (550). Given that this work uses static analysis, each row comprises of 1 or 0 only. Each row represents an application, which shows 1 (if the feature exists @ occur) or 0 (if the feature is non-existent @ non-occur).

Once the .csv file application is completed, the following step is to convert it to Attribute-Relation File Format (.arff) file. This conversion is done by Weka. The reason is, ARFF is an ASCII text file format, which Weka introduced specifically to loads faster (Williams, 2010). To achieve natural and acceptable results, the evaluation process applies the randomize option to randomly shuffle the order of both classes (malware and benign) in the datasets. Therefore, the class categories in each application are arranged randomly to provide a natural order. Figure 5.9 displays the captured screen of some part in the ARFF file after the randomize option. Eventually, the three machine learning classifiers, namely, MP, RF, and NB, utilized this ARFF file for evaluation.

```
@relation genetic

@attribute android.permission.READ_SMS numeric
@attribute android.permission.RECEIVE_SMS numeric
@attribute android.permission.WRITE_SMS numeric
@attribute checkPermission numeric
@attribute system/bin/secbin numeric
@attribute com.android.browser.application.id numeric
@attribute Class {B,M}

@data
0,0,0,1,1,0,M
0,0,0,1,0,0,B
1,1,1,1,1,0,M
1,0,1,0,1,0,M
0,0,0,0,0,0,B
```

Figure 5.9: Part of ARFF file

In constructing the machine learning predictive model, it is necessary to conduct *k*-fold cross validation methods, which runs repeatedly in *k*-fold times. In the experiment, the *k* subsets serve as the test set, whereas *k*-1 subsets are used as the training set. Moreover, the average of all *k* trials is computed for the evaluation (Schneider, 2016). This study used the 10-fold method, which repeatedly runs for 10-fold times for feature effectiveness. Particularly, the dataset is randomly divided into ten subsets of equal size and repeated ten times. In each repetition, one subset is used as the test set and the other nine subsets are combined to form the training set. Accordingly, the test set is excluded from the training set, which is used to detect unknown malware in this step. The following section provides the results from the six features.

5.2.2 Results

In the interest to evaluate the effectiveness of the six features which are selected by GS, it is crucial to address the machine learning classifiers performance matrix. The following section provides the results of this experiment in two aspects, namely, cross validation as well as training and testing section.

a) Cross validation

Table 5.8 lists the results derived from the experiments. In cross validation, FT accomplished magnificent results across four categories; highest accuracy (94.22%), highest correctly classified instances (5752), the lowest percentage of incorrectly classified instances (353), and lowest FPR (23.8%). In the true positive value, all classifiers gain the same mark except for J48, which achieved the lowest at 95.8 %. Additionally, in the ROC category, MLP obtained the highest value at 0.950. In the next category, precision, three classifiers shared an identical value of 97.6%, whereas the others recorded 97.5%. Moreover, all classifiers share a similar result for the recall and f-measure categories except for J48.

Table 5.8: Classifiers results in cross validation

		Classifiers				
Categories		Bayes	Function	Trees		
		NB	MLP	FT	RF	J48
Results	Accuracy	94.17%	94.19%	**94.22%**	94.20%	93.89%
	Correctly classified instances	5749	5750	**5752**	5751	5732
	Incorrectly classified instances	356	355	**353**	354	373
	FPR	24.5%	24.2%	**23.8%**	24%	25.1%
	TPR	**96%**	**96%**	**96%**	**96%**	95.8%
	ROC	0.936	**0.950**	0.947	0.946	0.905
	Precision	97.5%	**97.6%**	**97.6%**	**97.6%**	97.5%
	Recall	**96%**	**96%**	**96%**	**96%**	95.8%
	F-measure	**96.8%**	**96.8%**	**96.8%**	**96.8%**	96.6%

b) Training and testing

On the other hand, Table 5.9 enlists the results retrieved from 80% training and 20% testing section. This step trains the classifiers with 80% of the dataset, while the remainder is used for testing the model in detecting malware, which is excluded in the training set. As the total dataset is 6105, therefore, 4884 of it is used for training, while 1221 is for testing part. In the table, two classifiers, NB and FT share the best value in accuracy (95%), correctly (1160) and incorrectly (61) classified instances, and f-

measure (97.2%). However, NB holds the best value among other classifiers in TPR and recall which is 96.8%. Overall, FT is the outstanding classifier which jotted the highest value in all categories except TPR and recall in detecting unknown malware.

Table 5.9: Classifiers results in training and testing

Categories		Classifiers				
		Bayes	Function	Trees		
		NB	MLP	FT	RF	J48
Results	Accuracy	**95%**	94.9%	**95%**	94.9%	94.7%
	Correctly classified instances	**1160**	1159	**1160**	1159	1157
	Incorrectly classified instances	**61**	62	**61**	62	64
	FPR	22.1%	22.1%	**21.2%**	22.1%	23.9%
	TPR	**96.8%**	96.7%	96.7%	96.7%	96.7%
	ROC	0.94	0.953	**0.956**	0.954	0.916
	Precision	97.7%	97.7%	**97.8%**	97.7%	97.5%
	Recall	**96.8%**	96.7%	96.7%	96.7%	96.7%
	F-measure	**97.2%**	**97.2%**	**97.2%**	**97.2%**	97.1%

c) Before and after GA

In the interest to observe the differences before and after the GA process, this subsection compares the result in predicting the class samples in Weka. This part first prepares the benign samples which excluded from this experiment dataset to test the prediction effectiveness. Table 5.10 lists ten benign samples downloaded from the Google Play store and further scanned in VirusTotal to confirm its benign status. In the beginning, this study predicts these applications using 106 features. In the following, the machine learning predicts these samples using the GA features. Table 5.11 details the differences before and after the GA process with the best prediction results highlighted in bold.

Table 5.10: Benign samples information

Package name	Md5	Size (byte)
com.schoenmueller.wiesbadenPlus	cd217fd9a73b5660175acf5282e7f83a	27491923
com.incisivemedia.erandroid	300ae64c25457ed12d9d4afadd90424b	3610532
com.tester.wpswpatester	0e7c2d0422290eef1af6a80ad67e26b3	907795
com.appsbuilder4593	2c0e53592afdf430f1052ed7873e7124	536882
com.komfo.komfo	a8f5008f60c169e67feb93cfeb2f3368	4085095
cc.leet.free	66b129b07a3c4c8f82da428fa6809528	2191099
com.mgz.bmpkiosk	f49c0c987e0187dea84033a9fcaf55a2	27181791
com.aor.droidedit	55f51015242995f88cc059aa368a58af	1465402
com.bigint.writermd	b45b49c5369fa351a54130124675ee03	3942002
com.aflower.weightlosssmoothies	53939541cf6439979f154966ab0109e6	1403709

Table 5.11: Before and after GA

Classifier	NB	MLP	FT	J48	RF
Package name	Before GA				
com.schoenmueller.wiesbadenPlus	B	B	B	B	B
com.incisivemedia.erandroid	B	B	B	B	B
com.tester.wpswpatester	B	B	B	B	B
com.appsbuilder4593	B	B	B	M	B
com.komfo.komfo	B	B	B	B	B
cc.leet.free	B	B	B	B	B
com.mgz.bmpkiosk	B	B	B	B	B
com.aor.droidedit	B	M	M	M	M
com.bigint.writermd	B	B	B	B	B
com.aflower.weightlosssmoothies	B	B	B	B	B
Prediction/total benign	**10/10**	9/10	9/10	8/10	9/10
Time taken to predict (second)	0.1	175.57	1.81	0.36	0.36
Classifier	NB	MLP	FT	J48	RF
Package name	After GA				
com.schoenmueller.wiesbadenPlus	B	B	B	M	B
com.incisivemedia.erandroid	B	B	B	B	B
com.tester.wpswpatester	B	B	B	B	B
com.appsbuilder4593	B	B	B	M	B
com.komfo.komfo	B	B	B	B	B
cc.leet.free	B	B	B	B	B
com.mgz.bmpkiosk	B	B	B	M	B
com.aor.droidedit	B	B	B	B	B
com.bigint.writermd	B	B	B	B	B
com.aflower.weightlosssmoothies	B	B	B	B	B
Prediction/total benign	**10/10**	**10/10**	**10/10**	7/10	**10/10**
Time taken to predict (second)	0.01	2.5	0.31	0.02	0.09

Table 5.11 compares the accuracy as well as the time taken in machine learning before and after the GA course. Figure 5.10 depicts these differences in graph manner. Before GA selects the features, the machine learning classifiers incorrectly predicted the benign applications as malware for 5 times. One of the classifiers, J48 mistakenly predict for two times, compare to MLP, FT, and RF which forecast wrong for one time only. However, Figure 5.10 places NB in the top value which shows the only one classifier which successfully predicts the ten applications as benign. Dissimilarly, after the GA selects the best features among 106 features, all the classifiers accurately predict the applications, except J48 which incorrectly predict three times, which is the lowest place in the figure. It proves that by adopting GA in decreasing the number of features, machine learning classifiers are capable of enhancing the accuracy in distinguishing between benign and malware classes effectively. Meanwhile, Figure 5.10 illustrates another significant aspect of the GA process which is the time consumes for the classifier in conducting the intelligent machine learning prediction system.

Figure 5.10: The accuracy and time comparison in machine learning prediction

Before the GA course, by using 106 features, the total time taken to predict by machine learning classifiers is 178.2 seconds, with MLP jotted the longest time in the figure, which is 175.57 seconds. In the following, by using features derived from the GA

method, the classifiers successfully predicted 47 correct classes as benign with a satisfactory short time interval, 2.93 seconds in total. This demonstrates that by applying 106 features, the prediction took a long time to predict the samples as the classifiers need to consider all the features in each sample, which leads to the detection being more complex. By decreasing the features with GA, the classifiers are capable of decreasing the features complexity in the machine learning intrusion detection system. Having defined the number of feature differences, the next issue is the reverse engineering process in static analysis.

These machine learning steps were done in Weka whereas this study reverses engineers the samples and count the frequency before assigning it to the simulation. However, this situation is different when this prediction takes place in the actual IDS. Practically in the static analysis initial step, the IDS automatically reverse engineer each sample to obtain the code and further count each feature in machine learning classifiers. If in the situation where IDS needs to recognize 106 features in overall code, which include all types of strings including numbers, characters (e.g. double apostrophe, bracket), and symbols, this would possibly increase the noise of the strings and lead to irrelevant data. Figure 5.10 illustrates the worst accuracy in prediction and time taken in the simulation before the GA process. The predictions would possibly reach less accuracy and the time taken would be a great deal longer in real IDS. MLP takes 175.57 seconds which is equal to 2.93 minutes in the simulation, with the reverse engineering and counting features processes being excluded from the simulation process. Therefore, in actual IDS, MLP needs more than 2.93 minutes to detect malware. Henceforth, the features noise may increase when machine learning needs to process more than ten samples. Therefore, by reducing the features with the GA method, this work is able to decrease the noise of the string including unnecessary data, which contribute to the effectiveness of machine learning intelligent prediction system.

In addition, FT jotted promising scores in cross validation as well as training and testing phases, which surpasses other tree-based machine learning classifiers (i.e. RF and J48), despite the fact that past analysts precluded FT in their static analysis research. Meanwhile, the following section compares this study with previous works which apply similar malware dataset, Drebin.

5.2.3 Discussion

Every type of analysis, such as static and dynamic, applies a different method in examining malware and benign datasets. Correspondingly, different analyses present distinct advantages and disadvantages. To substantiate the effectiveness of the results, this section compares the results of this experiment with previous studies that used static and dynamic analyses with Drebin as the malware dataset. Table 5.12 compares the results in the accuracy category. As for the classifiers shown in the table, this section prefers the classifier which achieved only the best accuracy. These studies are chosen based on two reasons; 1) The dataset used is similar to this experiment's study, which is Drebin. 2) The selected studies are published in good quality journals. 3) The research objective is to decrease the features to derive its relevance, which is similar to the aim of this experiment.

Table 5.12: Comparison with other studies

References	Type of analysis	Classifier	Result	Dataset		Method for selecting features	Features
			Accuracy	Benign	Malware		
Ours	Static	FT	94.22%	550	5555	Genetic search	6 features (permissions, code-based, and directory path)
Drebin (Arp *et al.*, 2014)		Support vector machine	94%	123,453	5560	Joint vector space	Set to a vector space
Smartbot (Karim, Salleh, & Khan, 2016)	Dynamic	Simple logistic regression	99.49%	n/a	36 for learning, 4891 for testing	Element tree xml API of python and regular expression	16 network features (file system activities, network connections, information leakage, started services, SMS, cryptographic operations, DNS request, HTTP traffic parameters and unknown TCP and UDP conversations)

In the static analysis comparison, this study method in selecting features is different from another Drebin study (Arp *et al.*, 2014). GS is based on an approach which models the natural processes of the inheritance of multiple generations and the survival by fitness or adaptation to the environment. In contrast, joint vector space analyzed and identified the typical patterns and combinations of the features in geometric form. However, in the accuracy comparison, this experiment's accuracy exceeded 0.22% more with Drebin. This proved that GS performed much better in selecting relevant features compared to joint vector space. However, the frequency of the feature in their study remained unknown; therefore, this section is unable to discuss which and how many features are used for their Linear Support Vector Machine training.

Smartbot (Karim, Salleh, & Khan, 2016) outperformed this experiment's accuracy value of 94.22% with their 99.49% using dynamic analysis. This indicates that their method of using element tree xml in selecting features is convenient for the dynamic analysis. Nevertheless, their benign dataset is unknown, and they mentioned only the malware dataset in their study. In addition, although the accuracy is high in detecting malware, dynamic analysis consumes a much longer time compared with static as it classified the samples by executing it and by monitoring its behavior further. Moreover, this study aims to detect all types of malware (e.g. trojan, botnet, and root exploit), whereas their study particularly focused on detecting botnet.

Meanwhile, in FPR comparison for both static and dynamic analyses, this experiment's score is left behind compared to Drebin and Smartbot. This demonstrates that benign samples play an important role in producing beneficial results, whereas more benign samples contribute to achieving fewer FPR value (lower value indicates better performance). However, in Drebin, the FPR value is corresponding to one false alarm which is limited to 100 applications and precluded the exact list of features utilized in their experiment. On the other hand, this investigation implements two types of evaluation, the 10-fold cross validation and training and testing which include all the dataset. Furthermore, this study listed the six features for the machine learning classifier for this experiment. Meanwhile, in the Smartbot study, the experiment practices dynamic analysis that monitors application activities which consumes a lot of time, effort, performance and hardware resources. In comparison to static analysis study, this experiment only needs low resources (e.g. memory, CPU) than a dynamic analysis which consumes more hardware requirement and time. Having discussed the GS method, the next section conducts the range of repeated features evaluation.

5.3 Range of repeated features evaluation

This section provides the evaluation study that investigates the novel range of repeated features method in selecting the best features that malware frequently used as well as to satisfy the following points:

a) To propose the range of repeated features method to select the best features in minimal amount.
b) To validate the result from the range of repeated features method, this thesis compared it to existing anomaly-based experiment that utilized static features.

5.3.1 Experiment and procedure description

This section presents the overall workflow of the experiment. Figure 5.11 illustrates the methodology process. The first phase begins with data collection, which includes the reverse engineering process that retrieves the code of the application. The following phase is the feature investigation to select the exquisite of it among three categories (i.e. permission, directory path, and telephony). In the objective to detect unknown malware, the third phase applies the neural network-based classifiers (MLP, VP, and RBFN). The following subsections describe these phases in detail.

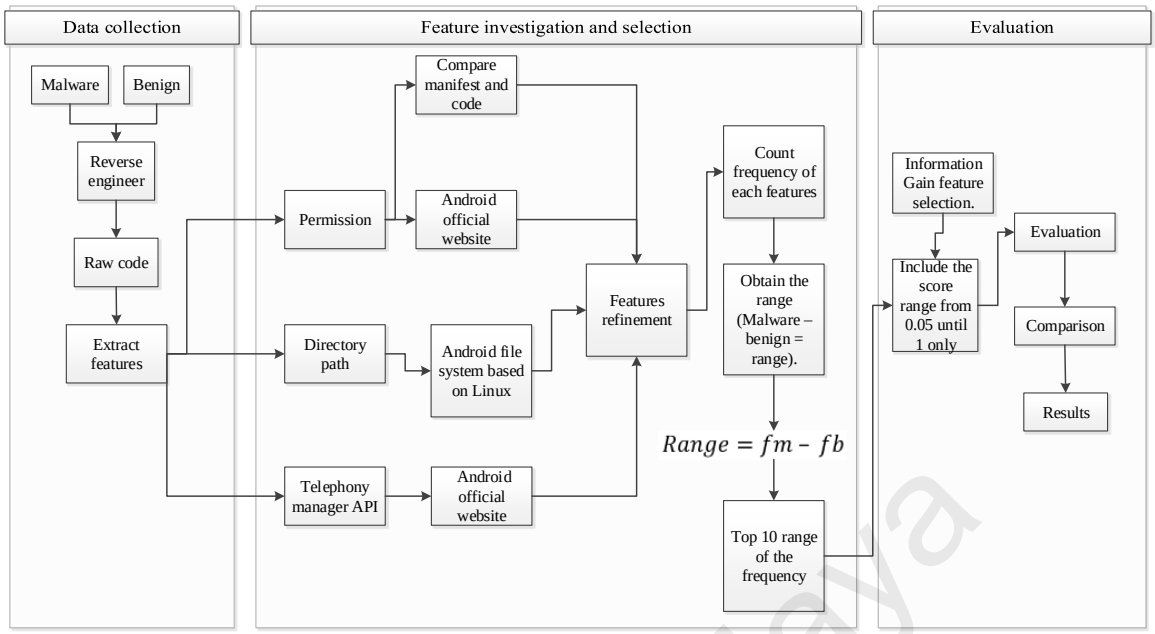


Figure 5.11: Methodology of the experiment

a) Data collection

Primarily, the dataset is a collection of related data to initiate the experiment in the initial phase. It consists of all the information required for research activities. Table 5.13 lists the summary of dataset information in this experiment. The following subsections describe the dataset in detail.

Table 5.13: Dataset summary

Dataset	Source	Total downloaded	Total used in the experiments
Malware	Drebin	5560	5551
Benign	Google Play store	7000	5551
		Total overall=	11102

b) Feature investigation and selection

In malware analysis, finding relevant features in minimal amount is crucial to establish an accurate predictive model, hence enhancing the detection accuracy of limited data and reducing the complexity of the predictive model (Feizollah *et al.*, 2015). This phase scrutinizes and selects which features are suitable for numerous lines of codes. This section applied the grep command in Ubuntu terminal to observe the entire codes and

pull out the features. Later, this experiment used R tool to clean the data and count the features. Figure 5.12 displays overall features in all categories.

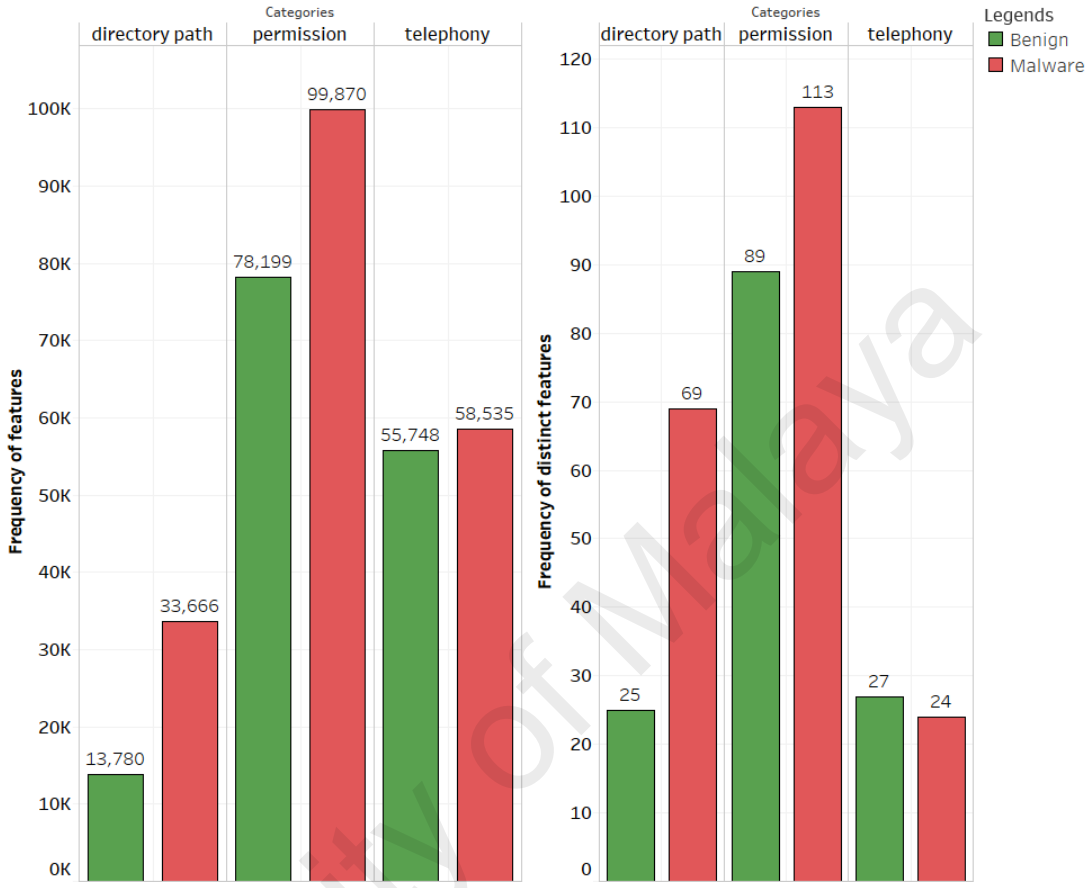


Figure 5.12: Frequency of features in all categories

The categories of features in this experiment are permission, directory path, and telephony. This study acknowledged the frequency of each feature that is existing or non-existing as well as those repeatedly used in each application. For instance, android.permission.WRITE_SMS existed in one application, and this similar application also utilized the same feature repetitively. In Figure 5.12, the left side shows the total number of features frequency which the applications repetitively used. It describes the malware utilized permission features found to be more than benign. However, in the telephony category, the range of features between malware (58535) and benign (55748) is small with only 2787. Meanwhile, the right-side view in Figure 5.12 depicts the number of distinct frequency.

In this study, the maximum total of distinct frequency in both malware and benign in directory path is 113, permission is 378, and telephony is 52. In this aspect, certain features exist in malware but are excluded in benign, and some of it exist in benign, but are non-existing in malware. Therefore, in the figure, by combining the distinct features of malware and benign in each category, the amount will not exceed the maximum total of distinct frequency. For instance, in the directory path, the total distinct features of malware (69) and benign (25) is 94, which is lower than the maximum total (113).

In the distinct features of view, Figure 5.12 describes that malware used more amount of features than benign, except for the telephony category. It shows that benign used 27 telephony features, exceeding malware which only used 24. It is worth noting that dataset total amount, for both benign and malware are similar, which is 5551. Therefore, this investigation discovers that, if this study adds more benign samples, it derives the probability that benign possibly used more amounts of telephony features than malware, which is different than the directory path and permission categories.

After this experiment obtained all the features frequency including the repetition, the subsequent step is to expose which features are most used by malware. In this step, this study subtracts the malware frequency with benign and obtain the range algorithm according to the equation in Section 4.1.2. The following section provides the top range features obtained from the equation in the permission, directory path, and telephony categories.

i) **Permission:** The first category is the permission features, which is encoded in the AndroidManifest.xml file. Fundamentally, every Android application includes AndroidManifest.xml in its directory. It represents the information that consists of the application package name, its permissions, activities, services, broadcast receivers, and content providers. In this category, this experiment includes the

permission in the Android reliable official website (Android, 2015), as well as other permissions obtained from this investigation. This exploration realized that the number of permissions is increased because the developer is capable to create permission in their own desired name upon creating an Android application. Hence, this study stopped the process and gained 378 permissions both in malware and benign. As this experiment included the imperative permissions from reliable sources in the Android official website, this experiment considers the total as sufficient. Figure 5.13 depicts the top 10 permission range and the frequency. The highest range in the figure is android.permission.READ_PHONE_STATE which appear between 5000 and 7000. It shows that malware used this permission more regularly than benign. Moreover, the same permission is included in the top ten range in Figure 5.14, which is the top ten malware permission enclosed in overall code (including manifest) and in the manifest file only. The permission listed in both Figure 5.13 and Figure 5.14 are android.permission.READ_PHONE_STATE, com.android.launcher.permission.INSTALL_SHORTCUT, android.permission.ACCESS_COARSE_LOCATION and android.permission.READ_CONTACTS. On the other hand, the next section provides the directory path feature.

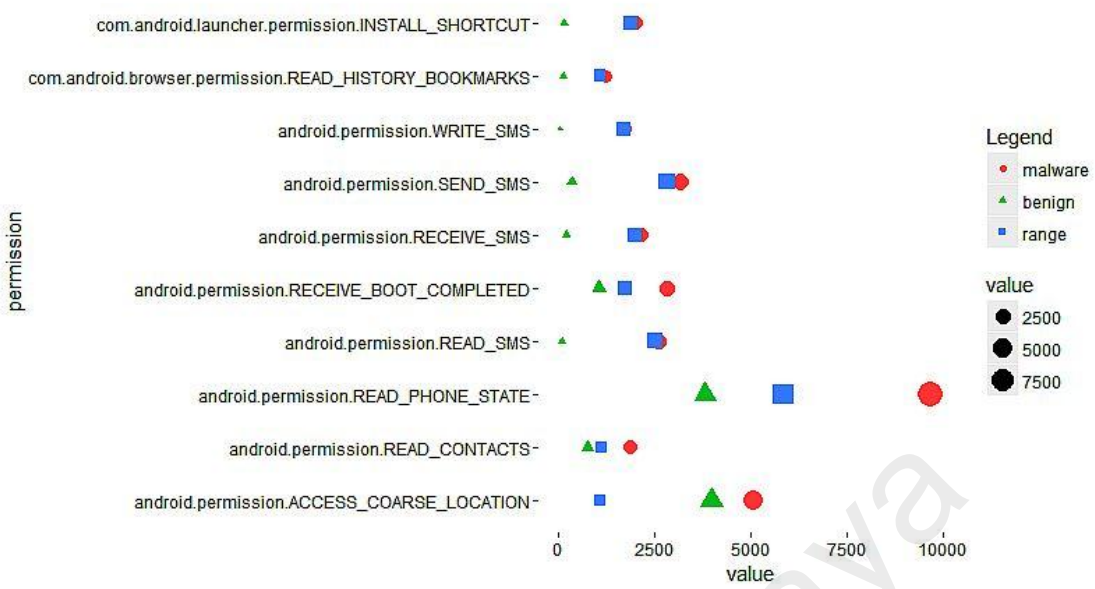


Figure 5.13: Top 10 permission range with frequency

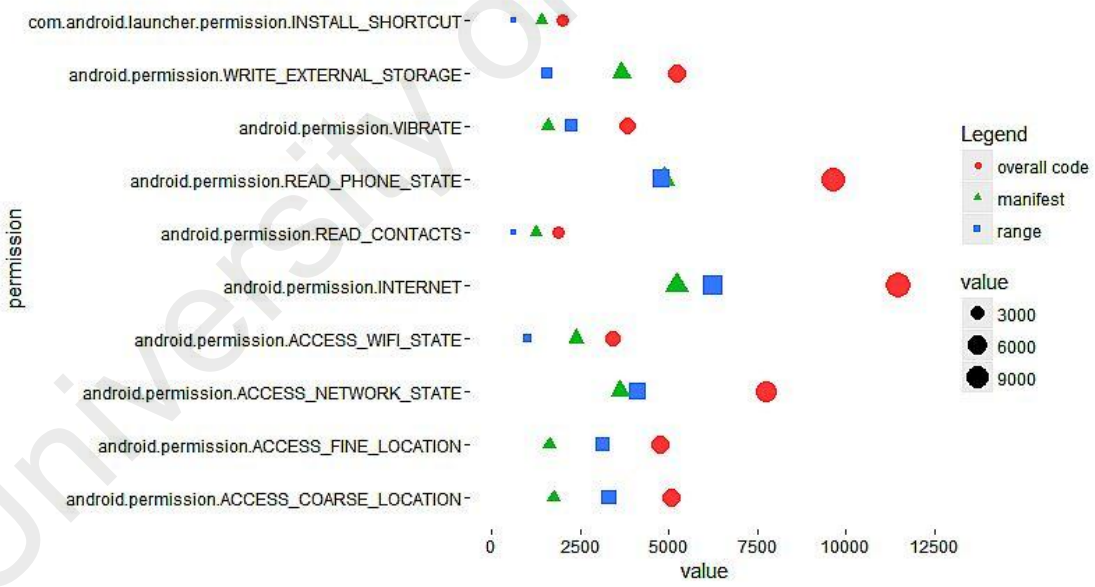


Figure 5.14: Top 10 malware permission range (code and manifest) with frequency

ii) **Directory path:** The second category included in this study is the directory path. This experiment investigates all the possible directory paths based on the Linux file system (Anderson, 2016). The total features in this category are 113. Figure 5.15 shows the top ten directory path frequency and the range. The highest range in this

figure is /data/data which is allocated between 6000 and 9000. The other features received a smaller range and are slightly similar to each other. Afterward, this investigation continues with the following category, which is telephony.

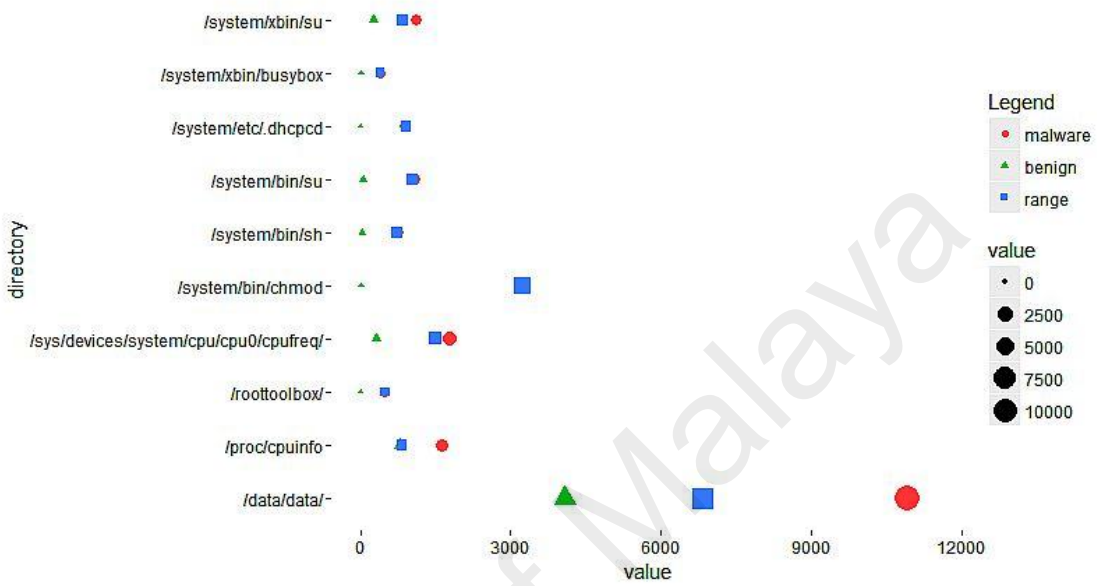


Figure 5.15: Top 10 directory path range with frequency

iii) **Telephony:** The final category is the telephony features, also known as telephony manager. It is one of the Application Programming Interface (API) in the Android system. In order to discover the features, this investigation is based on the reliable Android official website (Android, 2016), which the total is 52. Figure 5.16 depicts the top ten telephony range including the frequency. Accordingly, the features among the higher ranges are the getLine1Number, getSubscriberId, and getSimSerialNumber. Once this experiment calculates the range of malware and benign, there is a need to observe the relationship of malware and benign features as either positive or negative.

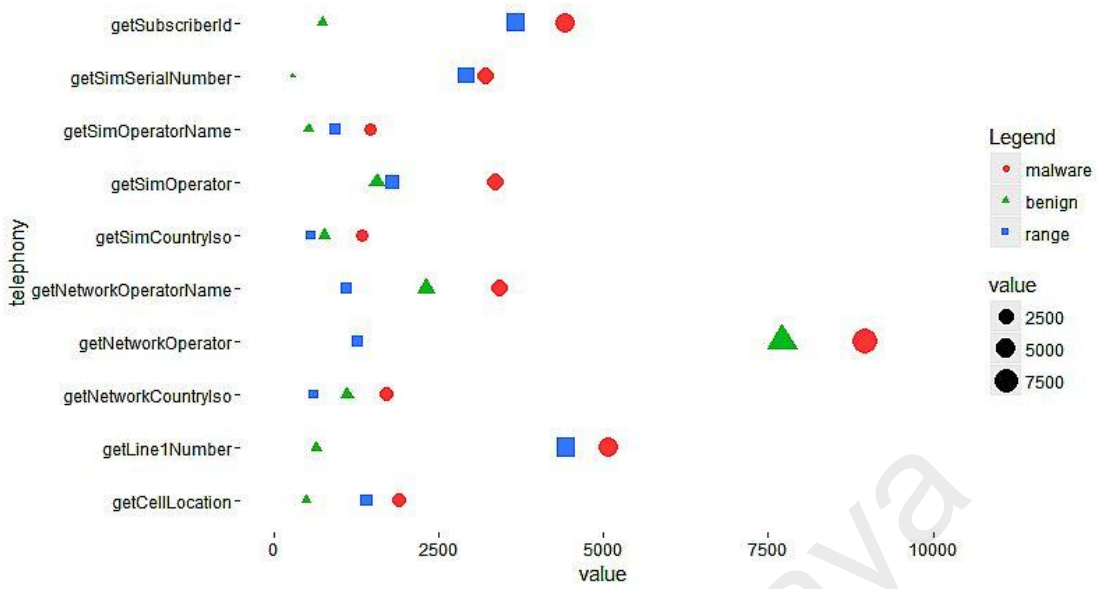


Figure 5.16: Top 10 telephony range with frequency

To discover the features relationship in malware and benign samples, this study provided the regression line depicted in Figure 5.17. The three lines (indicated as directory path, permission, and telephony) that rise from below to the top show a positive relationship. The lines indicate that whenever benign features increase, malware and benign expand. This finding proves that the malware features in this experiment are relevant features in detecting malware. The two significant lines, telephony (blue square) and permission (green triangle) depict that these two features are significant than directory features. In addition, this experiment used Information Gain (IG) value to ensure the features effectiveness in machine learning detection accuracy.

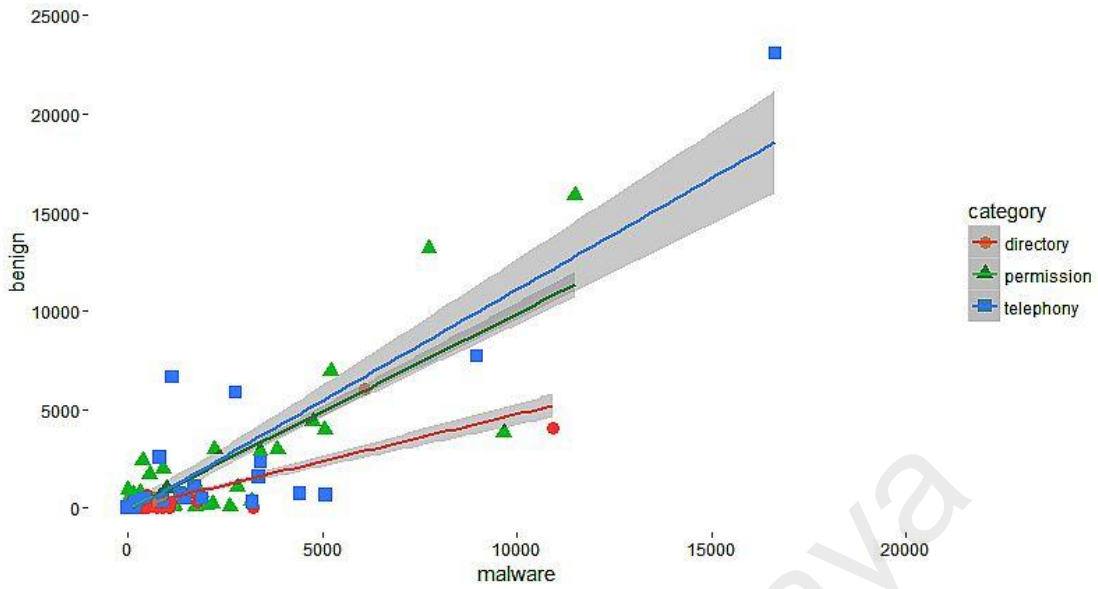


Figure 5.17: Regression lines of features

IG (Shannon, 1948) decides the amount of data by measuring how well it isolates the training samples according to the objective. This study inclines toward IG because of its compelling measuring features, generalization capability, accuracy enhancement, and short execution time (Kent, 1982). The highest IG value demonstrates the most effective for machine learning recognition. Table 5.14 records the twelve features in IG value beginning from 0.05 onwards.

Table 5.14: Features in IG value from 0.05 onwards.

Features	Info Gain	Category
android.permission.SEND_SMS	0.2273	permission
android.permission.READ_SMS	0.1772	permission
android.permission.READ_PHONE_STATE	0.1591	permission
android.permission.RECEIVE_SMS	0.1538	permission
getSubscriberId	0.1528	telephony
getLine1Number	0.1466	telephony
android.permission.WRITE_SMS	0.0992	permission
com.android.launcher.permission.INSTALL_SHORTCUT	0.0939	permission
android.permission.RECEIVE_BOOT_COMPLETED	0.0885	permission
getSimSerialNumber	0.0727	telephony
android.permission.ACCESS_NETWORK_STATE	0.0708	permission
com.android.browser.permission.READ_HISTORY_BOOKMARKS	0.0525	permission

For the final decision in selecting the best features, this study considers these twelve features for the following reasons. First, the features in this table are included in best IG

value. Second, the positive relationship in regression line proves the permission and telephony are the significant categories which are similar to the categories in best IG value in Table 5.14. Third, these twelve features are included in the top ten range frequencies. Therefore, this experiment utilized these selected twelve features for the bio-inspired ANN classification in machine learning.

c) Artificial Neural Network (ANN) evaluation

In machine learning, the classifier's prediction is based on the predictive model. In building the model, this study used Weka (Waikato, 2017). In this tool, it is fundamental to prepare a Comma Separated Values (.csv) file. As this investigation selected the twelve features and addition to the class label, this file, therefore, contains thirteen columns. Furthermore, this file contains 11102 lines of rows, which represents the total of both malware (5551) and benign (5551). Given that this work utilizes static analysis, each row comprises of 1 or 0 only. Each row represents an application, which shows 1 (if the feature exists @ occur) or 0 (if the feature is non-existing @ non-occur). ing

Once the .csv file application is complete, the following step is to convert it to Attribute-Relation File Format (.arff) file. This conversion is done by Weka. The reason is, ARFF is an ASCII text file format, which Weka introduced specifically to loads faster (Williams, 2010). To achieve natural and acceptable results, the evaluation process applies the randomize option to randomly shuffle the order of both classes (malware and benign) in the datasets. Therefore, the class categories in each application are arranged randomly to provide a natural order. Figure 5.18 displays the captured screen of some part in the ARFF file after the randomize option. Eventually, the three machine learning classifiers, namely, MLP, VP, and RBFN, utilized this ARFF file for constructing the machine learning model.

```
@relation dataset

@attribute android.permission.READ_PHONE_STATE numeric
@attribute com.android.launcher.permission.INSTALL_SHORTCUT numeric
@attribute android.permission.ACCESS_NETWORK_STATE numeric
@attribute android.permission.SEND_SMS numeric
@attribute android.permission.READ_SMS numeric
@attribute android.permission.RECEIVE_SMS numeric
@attribute android.permission.WRITE_SMS numeric
@attribute android.permission.RECEIVE_BOOT_COMPLETED numeric
@attribute com.android.browser.permission.READ_HISTORY_BOOKMARKS numeric
@attribute getSubscriberId numeric
@attribute getLine1Number numeric
@attribute getSimSerialNumber numeric
@attribute Class {M,B}

@data
1,1,1,0,0,0,0,1,0,1,1,1,M
1,1,1,1,0,0,0,1,0,1,1,0,M
0,0,1,0,0,0,0,0,0,0,0,0,B
1,0,1,1,1,1,1,1,0,0,0,0,M
0,1,1,0,0,0,0,0,0,0,0,0,B
```

Figure 5.18: Sample screenshot of arff file

In constructing the machine learning predictive model, it is necessary to conduct *k*-fold cross validation methods, which runs repeatedly in *k*-fold times. In the experiment, the *k* subsets serve as the test set, whereas *k*-1 subsets are used as the training set. Moreover, the average of all *k* trials is computed for the evaluation (Schneider, 2016). This study used the 10-fold method, which repeatedly runs for 10-fold times for feature effectiveness. Particularly, the dataset is randomly divided into ten subsets of equal size and repeated ten times. In each repetition, one subset is used as the test set and the other nine subsets are combined to form the training set. Accordingly, in this step, the test set is excluded from the training set, which is used to detect unknown malware.

5.3.2 Results

This section evaluates the effectiveness of the twelve features derived from the third experiment investigation. The evaluation information for the results was explained in Section 5.1.1.

In this section, the beginning result is MLP, followed by VP and finally RBFN. In the aspiration to discover results in different parameter value beginning from 0.1 to 1.0, these three classifiers utilized these similar values in different parameters. For MLP, the

parameter is the learning rate, the exponent is for VP and the minimum standard deviation is for RBFN.

a) MLP

Table 5.15 enlists MLP results with different learning rate values. It refers to the update network weights amount for each training period. This learning rate is for the backpropagation algorithm with the default value being 0.3. The value of this parameter should be between 0 and 1. In this table, the best parameter is the 0.1, which achieved the best score in four metrics, whereas the accuracy is 90%, FPR is 7.2%, precision is 92.3% and f-measure is 89.6%. This result justifies that the evaluation value is changed to a better score whenever the amount of weight in learning rate is decreasing.

Table 5.15: MLP evaluation results

Learning rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Evaluation	MLP (%)									
Accuracy	**90**	**90**	89	89	89	89	89	89	89	**90**
True positive rate (TPR)	87	86.8	87.7	88.2	**88.3**	87.3	87.9	88.1	**88.3**	87.2
False positive rate (FPR)	**7.2**	7.5	8.7	9.3	9.3	8.3	8.7	9.3	9.8	7.9
Precision	**92.3**	92.1	91	90.5	90.5	91.3	91	90.5	90	91.7
Recall	87	86.8	87.7	88.2	**88.3**	87.3	87.9	88.1	**88.3**	87.2
F-measure	**89.6**	89.4	89.4	89.3	89.3	89.3	89.4	89.3	89.1	89.4

b) VP

On the other hand, Table 5.16 lists the VP evaluation value with the exponent as the parameter. The exponent refers to the value of the polynomial kernel in VP algorithm, which the default is 1. For this neural network classifier, the evaluation value changes to better results whenever the exponent value is increasing. The best parameter is 1.0 which jotted 89% in accuracy, 11.3% in FPR, 88.7% in precision and 88.7% in f-measure. This demonstrates that the increment of exponent contributes to better evaluation results.

Table 5.16: VP evaluation results

Exponent	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Evaluation	VP (%)									
Accuracy	86	84	84	83	84	85	86	87	88	**89**
True Positive Rate (TPR)	86.9	85.3	85.1	84.6	84.8	85.1	87	87.4	**89.6**	88.7
False Positive Rate (FPR)	13.7	16.9	17.9	17.5	16.4	15.9	15.4	13.4	13.3	**11.3**
Precision	86.4	83.5	82.6	82.9	83.8	84.3	85	86.7	87.1	**88.7**
Recall	86.9	85.3	85.1	84.6	84.8	85.1	87	87.4	**89.6**	88.7
F-measure	86.6	84.4	83.9	83.7	84.3	84.7	86	87	88.3	**88.7**

c) RBFN

Table 5.17 shows the RBFN evaluation results with the best score highlighted in bold. The parameter is the minimum standard deviation, which is set for the clusters in the neural network. In this table, the lowest parameter receives good results with 87% in accuracy, 86.1% in TPR, 86.1% in recall and 87.1% in f-measure. This result describes that the most minimal standard deviation, which is 0.1, records the best evaluation results. So far this section has focused on the outcomes of accuracy, TPR, FPR, precision, recall and f-measure, while Figure 5.19 depicts all these evaluations in graph manner.

Table 5.17: RBFN evaluation results

Minimum standard deviation	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Evaluation	RBFN (%)									
Accuracy	**87**	**87**	86	86	**87**	**87**	**87**	**87**	**87**	**87**
True positive rate (TPR)	**86.1**	85.9	83.5	83.9	83.8	83	82.8	82.4	81.5	81.2
False positive rate (FPR)	11.7	11.6	10.9	10.9	10.6	9.6	8.9	8.2	7.6	**7.3**
Precision	88	88.1	88.5	88.5	88.8	89.6	90.3	91	91.5	**91.7**
Recall	**86.1**	85.9	83.5	83.9	83.8	83	82.8	82.4	81.5	81.2
F-measure	**87.1**	87	85.9	86.2	86.2	86.2	86.4	86.5	86.2	86.1

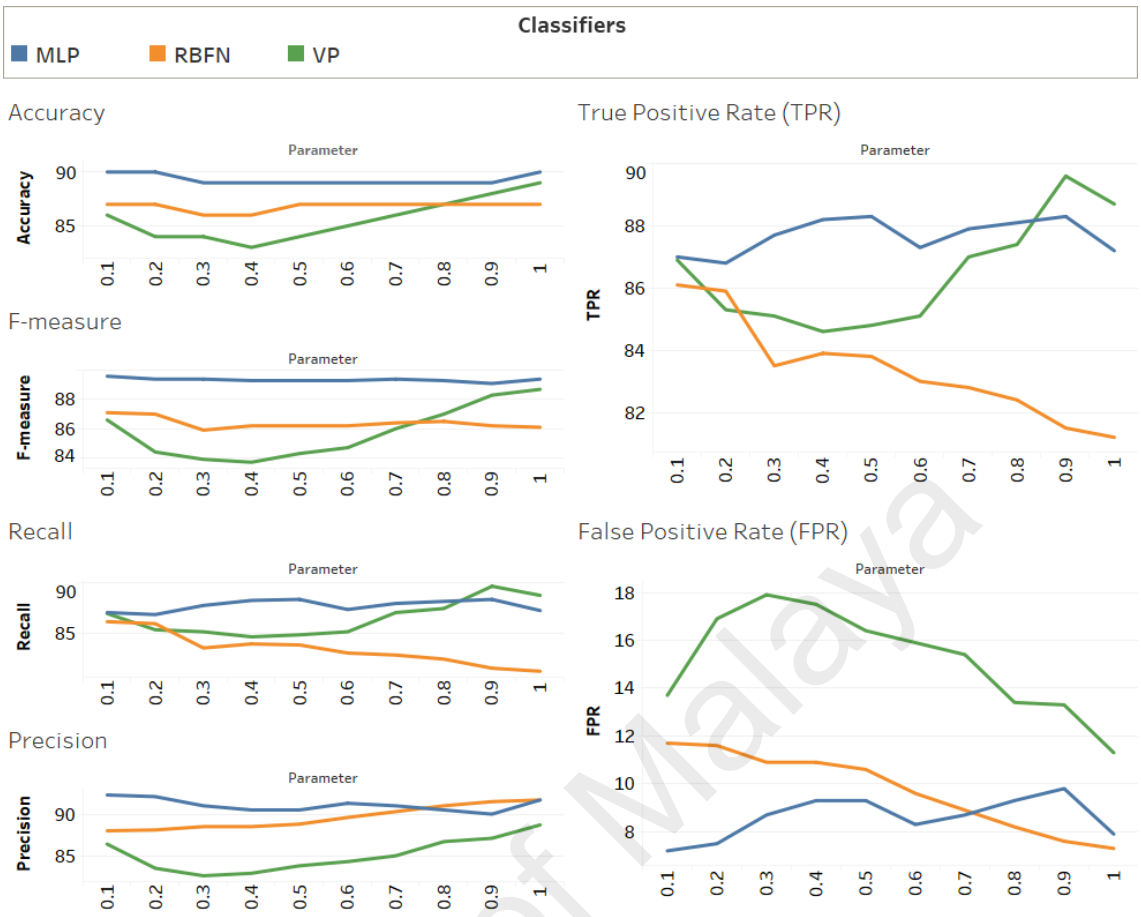


Figure 5.19: The evaluation results in graph manner

Figure 5.19 demonstrates that by using this experiment's proposed features and method, MLP classifier recorded the best prediction in this simulation that surpasses other classifiers. It marks the highest scores in accuracy, TPR, f-measure, recall, and precision. Furthermore, MLP achieves the low scores in incorrect prediction in classifying malware. Although VP seems to have the worst score compared to other classifiers, however, it still achieves better scores than RBFN in recall and TPR. So far, this experiment has focused on the outcomes in accuracy, TPR, FPR, precision, recall, and f-measure, while the following part describes the Receiver Operating Characteristic (ROC) values from different parameters.

d) Receiver Operating Characteristic (ROC)

In this section, this study utilized the ROC value to discover the tradeoff between the TPR and FPR values. It is a fundamental indicator for diagnostic test evaluation. TPR (sensitivity) is plotted in the function of FPR for different cutoff points of a parameter. The ROC value closer to 1 indicates good classifier performance and high classification accuracy. Table 5.18 shows the ROC value in each parameter with the leading value highlighted in bold. In ROC value comparison, MLP obtains the closest value to 1 at 0.958. This value is higher than VP and RBFN (0.888 and 0.93). This demonstrates that MLP performs better in comparing ROC value between VP and RBFN. On the other hand, Figure 5.20 displays the ROC value in graph form to reveal the pattern line when the parameter values are changing from 0.1 to 1.0.

Table 5.18: ROC value in each parameter

Classifier	ROC									
Parameter	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
MLP	**0.958**	**0.958**	0.957	0.**958**	0.957	0.956	0.957	0.955	0.954	0.955
VP	0.866	0.843	0.836	0.837	0.844	0.849	0.858	0.87	0.882	**0.888**
RBFN	0.92	0.922	0.918	0.918	0.92	0.921	0.921	0.926	0.928	**0.93**

In the MLP classifier, the ROC values are declining when the learning rate parameter increases from 0.1 to 1. It demonstrates that the MLP reaches the best ROC value in a lower learning rate. Unlike MLP, the ROC value of VP gradually rises whenever the parameters are increasing. Similarly, RBFN receives akin pattern line compared to VP. These lines verify that these two classifiers are experiencing better ROC value when the parameter values are expanding. This section thus far discusses results derived from this study; hence the next section presents the confusion matrix.

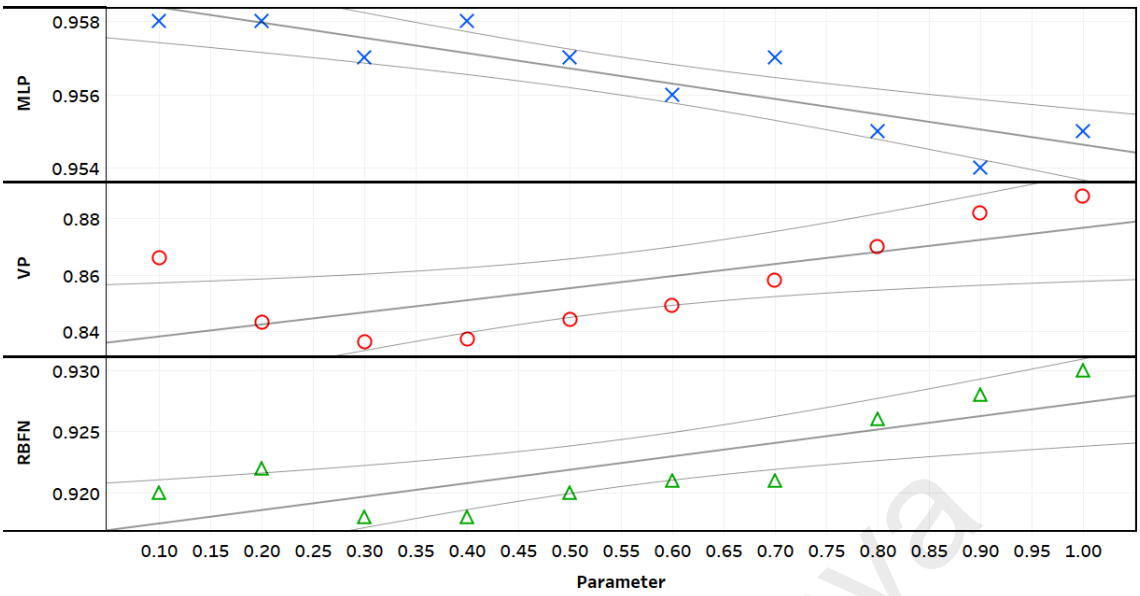


Figure 5.20: ROC value in graph form

e) Confusion matrix

A confusion matrix is a table to explain the performance of the machine learning classifier model**.** It provides the information of correct and incorrect prediction that have been done from the testing phase. Table 5.19 lists the confusion matrix of MLP, followed by Table 5.20 from VP, and Table 5.21 from RBFN classifiers. To clearly describe the correct prediction in detecting malware, Figure 5.21 depicts the pattern of this information between the classifiers.

Table 5.19: Confusion matrix of MLP classifier

Parameters	Actual	Predicted	
		Predicted malware	Predicted benign
0.1	Actual malware	4827	724
	Actual benign	402	5149
0.2	Actual malware	4820	731
	Actual benign	414	5137
0.3	Actual malware	4871	680
	Actual benign	481	5070
0.4	Actual malware	4896	655
	Actual benign	515	5036
0.5	Actual malware	4899	652
	Actual benign	517	5034
0.6	Actual malware	4845	706
	Actual benign	461	5090
0.7	Actual malware	4877	674
	Actual benign	482	5069
0.8	Actual malware	4891	660

	Actual benign	514	5037
0.9	Actual malware	4901	650
	Actual benign	543	5008
1	Actual malware	4840	711
	Actual benign	437	5114

Table 5.20: Confusion matrix of VP classifier

Parameters	Actual	Predicted	
		Predicted malware	Predicted benign
0.1	Actual malware	4822	729
	Actual benign	760	4791
0.2	Actual malware	4737	814
	Actual benign	939	4612
0.3	Actual malware	4725	826
	Actual benign	992	4559
0.4	Actual malware	4694	857
	Actual benign	969	4582
0.5	Actual malware	4709	842
	Actual benign	908	4643
0.6	Actual malware	4725	826
	Actual benign	882	4669
0.7	Actual malware	4830	721
	Actual benign	853	4698
0.8	Actual malware	4853	698
	Actual benign	746	4805
0.9	Actual malware	4974	577
	Actual benign	737	4814
1	Actual malware	4921	630
	Actual benign	626	4925

Table 5.21: Confusion matrix of RBFN classifier

Parameters	Actual	Predicted	
		Predicted malware	Predicted benign
0.1	Actual malware	4780	771
	Actual benign	649	4902
0.2	Actual malware	4767	784
	Actual benign	645	4906
0.3	Actual malware	4634	917
	Actual benign	603	4948
0.4	Actual malware	4659	892
	Actual benign	605	4946
0.5	Actual malware	4649	902
	Actual benign	589	4962
0.6	Actual malware	4606	945
	Actual benign	533	5018
0.7	Actual malware	4594	957
	Actual benign	494	5057
0.8	Actual malware	4575	976
	Actual benign	454	5097
0.9	Actual malware	4524	1027
	Actual benign	420	5131
1	Actual malware	4508	1043
	Actual benign	407	5144

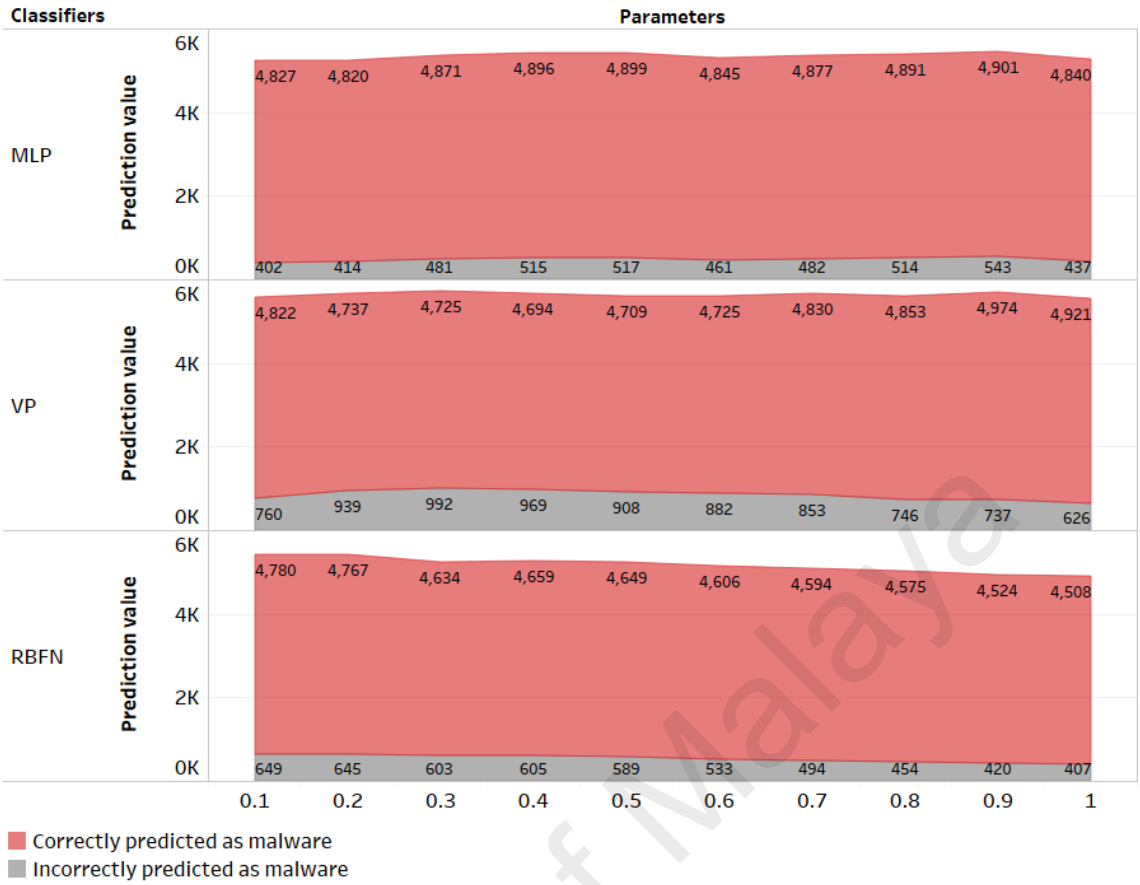


Figure 5.21: Comparison between correct and incorrect predictions of malware

Between other classifiers, Figure 5.21 demonstrates that MLP jotted magnificent scores in predicting unknown malware with this experiment's proposed features. In terms of incorrect prediction perspective, MLP collected the minimal value which is only 402 in the parameter of 0.1. However, RBFN nearly followed the MLP value by scoring 407 in the 1.0 parameter. Meanwhile, VP collected the highest incorrect prediction which is 992 in the 0.3 parameters.

5.3.3 Discussion

In the enthusiasm to substantiate the adequacy of the results of this experiment, this section compares and discuss it with the previous studies in accuracy category. The previous studies are chosen based on three reasons; 1) The previous studies utilized similar malware in the dataset in this experiment – Drebin. 2) The selected papers were published in good ranking journals, which were indexed in "Thomson Reuters Institute

of Scientific Information" (ISI), Web of Science (WoS) database (Razak *et al.*, 2016). 3) The studies utilize static analysis, which is similar to this study. Table 5.22 compares the studies that used Drebin.

Table 5.22: Comparison between the simulation results

References	Classifier/system	Result	Dataset		Features
		Accuracy	Benign	Malware	
This study	MLP	90%	5551	5551	12 (permission and telephony)
	VP	89%			
	RBFN	87%			
(Karim, Salleh, Khan, *et al.*, 2016)	DeDroid	90%	14865	5064	18 (permission and API calls)
(Arp *et al.*, 2014)	Drebin	94%	123,453	5560	Set to a vector space

In Table 5.22 comparison, DeDroid method is different with this experiment. Dedroid adopted the comparative system, dissimilarly with the study which adopted machine learning method to detect unknown malware. By adopting intelligent machine learning, this third experiment is capable to classify classes between benign and malware with 90% accuracy from MLP classifier. This accuracy value between DeDroid and this study is similar, which is 90% as well. Meanwhile, other classifiers (i.e. VP and RBFN) recorded slightly low accuracies than DeDroid. By considering the benign dataset, DeDroid (14865) utilized more than this study (5551). This derived the probability that, by increasing the benign samples, this study is capable of increasing the accuracy value by more than 90%.

On the other hand, Drebin used SVM machine learning classifier and jotted 94% in accuracy, which surpassed this study of 90%. However, they excluded the exact list of features used in their paper and used 123,453 as the benign dataset. As DeDroid and Drebin jotted higher accuracies by using more benign than malware samples, these situations provide convincing evidence that by using more benign, security analyst are

capable of increasing the accuracy. Other than accuracy and dataset samples, features selection is one of the crucial attentions in malware detection as well.

In detecting malware, feature selection provides significant effects on experimental results. Minimum features are desirable because it offers enhanced accuracy with fewer data, reduces the complexity of the detection model, decreases noise and irrelevant data (Feizollah *et al.*, 2015; Sarip *et al.*, 2016; Zia *et al.*, 2015). Hence, in features comparison between DeDroid (18 features) and this study (12 features), MLP in this study gains similar accuracy (90%) with fewer features. The next study, Drebin used vector space in selecting features and achieved higher accuracy than this experiment. However, their study precluded the exact list of features and therefore is unable to compare a number of features they have used. After having conducted the range of repeated features method, the subsequent section is the evaluation of the root exploit experiment.

5.4 Evaluation of root exploit experiment

This section provides the evaluation study that investigates the root exploit features and selects the best of it for the anomaly-based detection and to satisfy the following points:

a) To investigate the root exploit as well as the novel features of ADB.
b) To validate the result from the selected features, this thesis compared it to existing anomaly-based experiment that utilized static features.

5.4.1 Experiment and procedure description

This section describes the experiment for root exploit feature identification. It consists of four stages: data collection, application reverse engineering, feature extraction, and machine learning. Figure 5.22 shows malware and benign applications are collected in the data collection stage. The following step is to reverse engineer the application to

retrieve the codes. Subsequently, this experiment investigates further by extracting and identifying the relevant features. The final stage evaluates these features using machine learning classifiers for evaluation of the machine learning prediction.

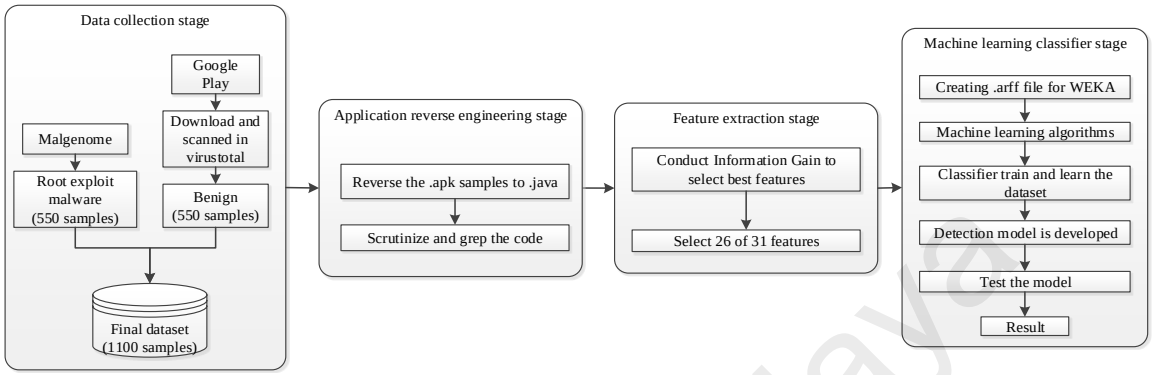


Figure 5.22: Detecting root exploit malware methodology

a) Data collection

This first experiment focuses on root exploit malware, which is present in Malgenome. Hence, this experiment considers all the samples of the corresponding types and obtained a total of 550 samples (Y. Zhou & Jiang, 2012a). Table 5.23 tabulates the samples family and its descriptions.

Table 5.23: List of root exploit malware

Root exploit	Frequency	Descriptions
Asroot	8	Asroot is similar to the word in Unix terminal, that is, login *as root*. Asroot is a standalone program that capable to execute without OS service and installation procedure.
BaseBridge	120	BaseBridge conducts a silent installation of additional applications without user approval.
DroidDream	16	Whenever the user clicks the application icon on the home screen or when an intent ACTION_MAIN is received by the application, DroidDream directly hijacks the entry activity of the host application.
DroidDeluxe	1	Without querying the user to grant the root privileges, DroidDeluxe leverages known root exploits to bypass the built-in security sandbox.
DroidCoupon	1	DroidCoupon obfuscates the file names that are associated with root exploits (e.g., impersonate as picture file with .png file type).
DroidKungfu 1	34	DroidKungfu contains encrypted root exploit scripts and decrypts these scripts during runtime conditions. It remotely downloads and updates a new version via a network.
DroidKungfu 2	30	
DroidKungfu 3	309	
DroidKungfu 4	20	
zHash	11	zHash contains *exploid* file names, which are exactly the same as the publicly available file names.
Total	**550**	

On the other hand, this study collected benign applications from the Google Play store (Google, 2014). Table 5.24 lists the benign samples with the frequency. To achieve an equal condition and obtain unbiased content, this work downloaded 25 applications in each of the 34 categories.

Table 5.24: List of benign applications

Category	Frequency	Category	Frequency
Books and Reference	15	Games (Puzzle)	11
Business	20	Games (Racing)	11
Comic	21	Games (Role Playing Games)	23
Communication	23	Games (Simulation)	12
Education	11	Games (Sports)	9
Entertainment	16	Games (Strategy)	16
Finance	24	Games (Word)	15
Games (Action)	18	Health and Fitness	19
Games (Adventure)	12	Live Wallpaper	17
Games (Arcade)	10	Media and Video	18
Games (Board)	14	Medical	17
Games (Card)	15	Music and Audio	11
Games (Casino)	18	News and Magazine	23
Games (Casual)	13	Personalization	16
Games (Education)	15	Photography	11
Games (Family)	15	Productivity	17
Games (Music)	20	Shopping	24
		Total	**550**

To avoid malware in benign applications, this research conducted a scan using VirusTotal (VirusTotal, 2016). A total of 850 applications is downloaded. However, 300 applications are discarded because of the following reasons. First, this step considers only the applications with VirusTotal scan result of 0, which means the application is malware-free. Second, certain applications are placed in multiple categories. For instance, one application exists in books, references and comic categories. Accordingly, this second step excludes similar applications in any category to avoid duplicates. Third, this step is to set the total target frequency of samples as 550 for each malware and benign applications. The reason is to observe the result by using the similar number of samples. By combining both malware and benign datasets, thereby the total are 1,100 samples (i.e. 550 benign, 550 malware).

b) Reverse engineering

The general process in static analysis is reverse engineering, which involves reversing the application compilation to reveal its programming codes. To analyze malware and benign applications, this phase applies this process to the dataset. Figure 5.23 illustrates the reverse engineering procedure. This method applies the files, reverses them to Java programming codes, and selects the features from the codes. To reverse .apk to Java codes, this step used a tool called Jadx (Skylot, 2015). This engineering tool is able to reverse compiled .apk to .java extension files (java code).

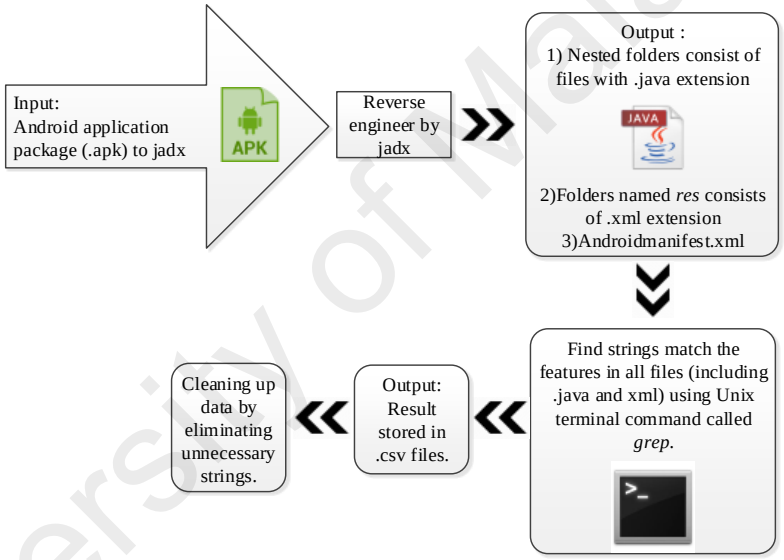


Figure 5.23: Reverse engineering process

The following steps involve extracting and selecting the features in the code. Given that many lines of code are involved, the keywords used in the malware and benign datasets must be determined. These processes are conducted manually using the "grep" command. The output in the .csv file is saved via Ubuntu's terminal. The command is normally used by Unix users to find and grab any desired keyword according to user demand. This study uses this command to find malicious strings and keywords for the features. Figure 5.24 depicts a sample screenshot of the extracted information and shows a malware application and its Secure Hash Algorithm (SHA) name, folder name,

and java file. The string contains "/system/bin/chmod." The figure also shows five malware samples that acquire one java file containing the "/system/bin/chmod" string.

```
5f7a40015a1f3f42802424ec776799f5e096/com/ UpdateService.java: String str_d = "/system/bin/chmod";
3ac954c94c9c420578777c4aa878b12cd89/com/google1.java: str = "/system/bin/chmod";
4f1cbb091dcde0cd0e8fe0d4bd27134750b/com/Service.java:str5 = "/system/bin/chmod";
5fb8cabd3dd8d4caca3f626f96419ea70e4f/update/Update.java: dir = "/system/bin/chmod";
6a1431f60a704ca729d36c7edf2b8142b03/com/google/UpdateService.java: TCP.execute("/system/bin/chmod");
```

Figure 5.24: Example screenshot of chmod directory feature

After the step grabbed the strings, it is essential to clean the data. In this case, certain strings are often confused with one another. For example, cat is one of the Linux commands. This word may be confused with other words, such as concatenate, locate, and other similar words containing cat strings. Thereafter, this experiment identified and observed the full exact line of codes. The cat command, which is specifically used for the Linux platform, is pulled out.

c) Feature extraction

This section describes the 31 features from the dataset samples. The types of features are system command, directory path, and code-based features. The amounts of the first, second, and third types of features are 12, 10, and 9, respectively. The following subsection describes these features briefly.

i) **System command:** The system command consists of a terminal, process, and ADB command. Examples of terminal and process commands are adb_enabled and cat. As an illustration, *cat* is a maintain command used although the kernel version and API call are regularly updated. The *cat* is an abbreviation for "concatenate." This command is the most frequently used in Unix-type OS and allows the user to view files, create single or multiple files, concatenates files, and redirects output via a terminal.

On the other hand, ADB command is a terminal command line tool that is allowed communication between the user and the Android emulator to connect to the Android-powered device (Android Developer, 2017). This communication tool allows users to easily connect to their own mobile devices via desktop computers or notebooks. In consequence, malware practitioners misuse this tool to gain malicious actions, particularly gaining root. These system commands are unique elements because they are unchanged and similar to other Linux-based OS commands globally. Furthermore, the architecture of Android depends on the Linux layer. Hence, this feature increases the reliability of future detection methods. Figure 5.25 depicts the system command features existence. For instance, cat appears 21 times in malware samples, but only appears once in benign samples. Startservice –n appears 359 times in malware samples, but do not appear in benign samples.

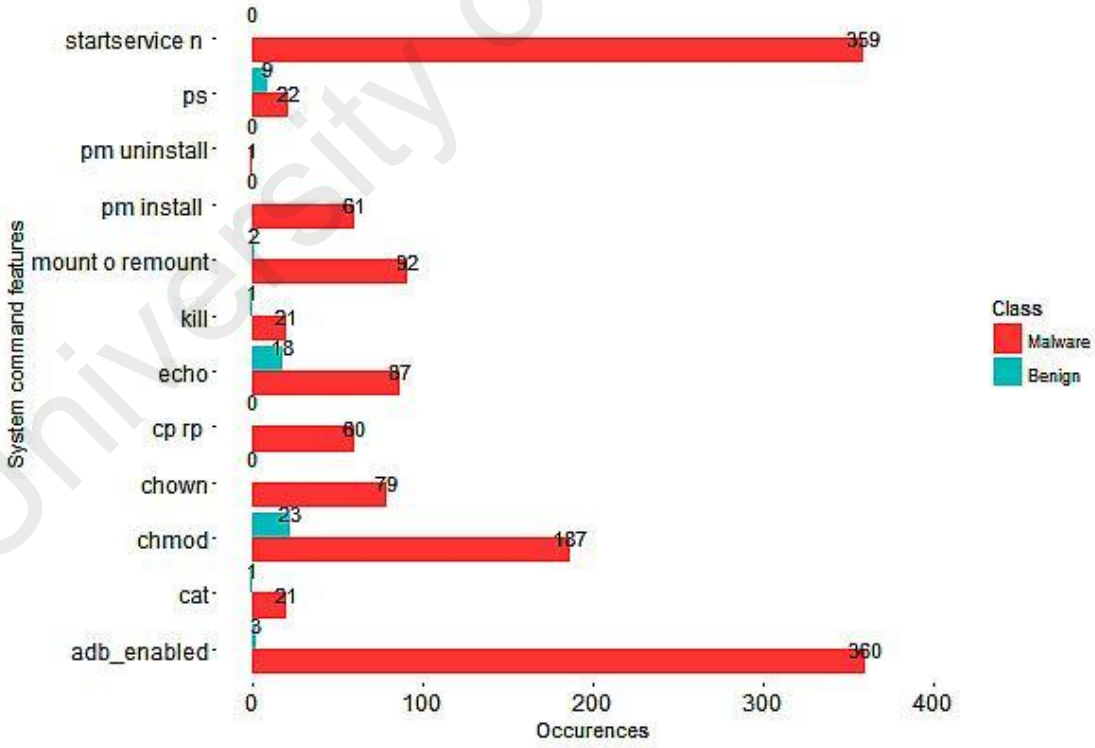


Figure 5.25: System command occurrences

i) **Directory path:** Android has its own OS directory path whereas its architecture is similar to Linux kernel. During feature selection, this part discovered the list of the directory path. The examples are /system/bin/mount and /proc, which are paths that unscrupulously authorize an attempt to enter and gain access to the kernel directories and obtain root privileges without user consent. This study includes these sensitive directory paths in the current experiment as features. Figure 5.26 shows one of the directory paths (/system/xbin/su) appear 361 times in the malware samples. This result is higher than benign samples, which only appear 38 times.

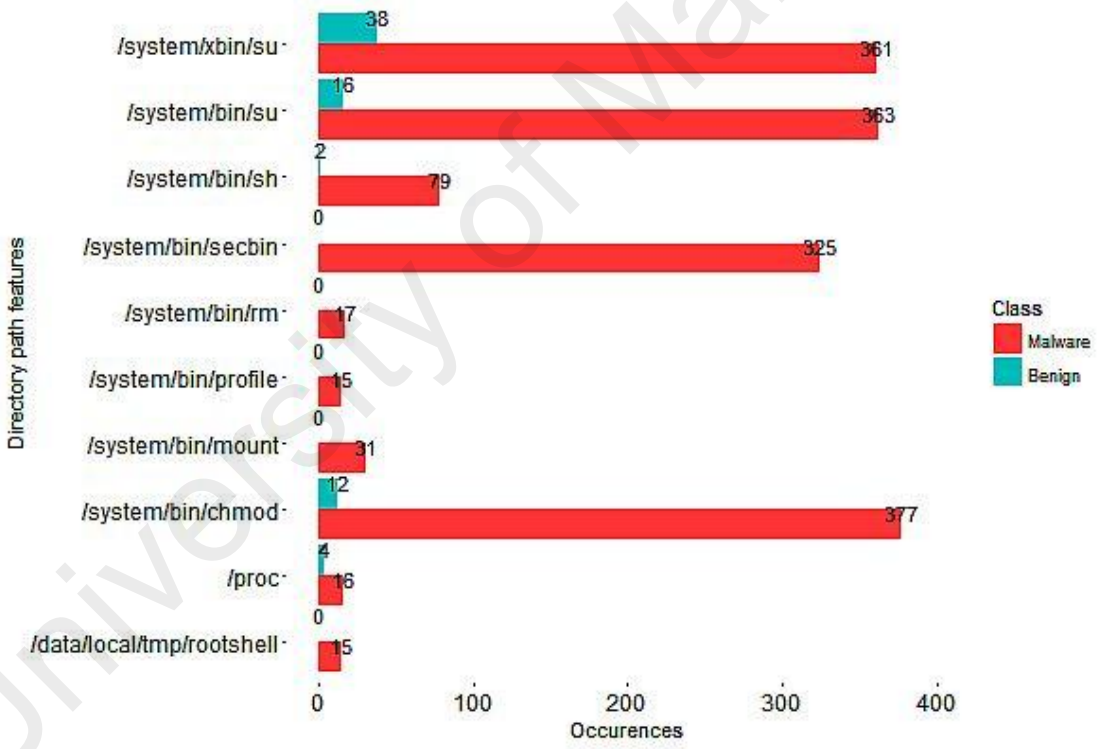


Figure 5.26: Directory path occurrences

ii) **Code-based:** In this study, the type of feature other than system command and directory path is the code-based features. Figure 5.27 illustrates the code for executing the command is createSubprocess, which appears 83 times in malware samples but do not appears in benign samples. Meanwhile, another feature that does not appear in the benign sample is Forked. This feature is the string in an

argument or a parameter, which occurs 76 times in malware samples. Other code-based features that are included in the static analysis approach are as follows: setPtyWindowSize (code to execute process), three code execution processes (exec(), exec("sh"), exec("su")), stderr (to detect standard error), stdin (standard input), and stdout (standard output).

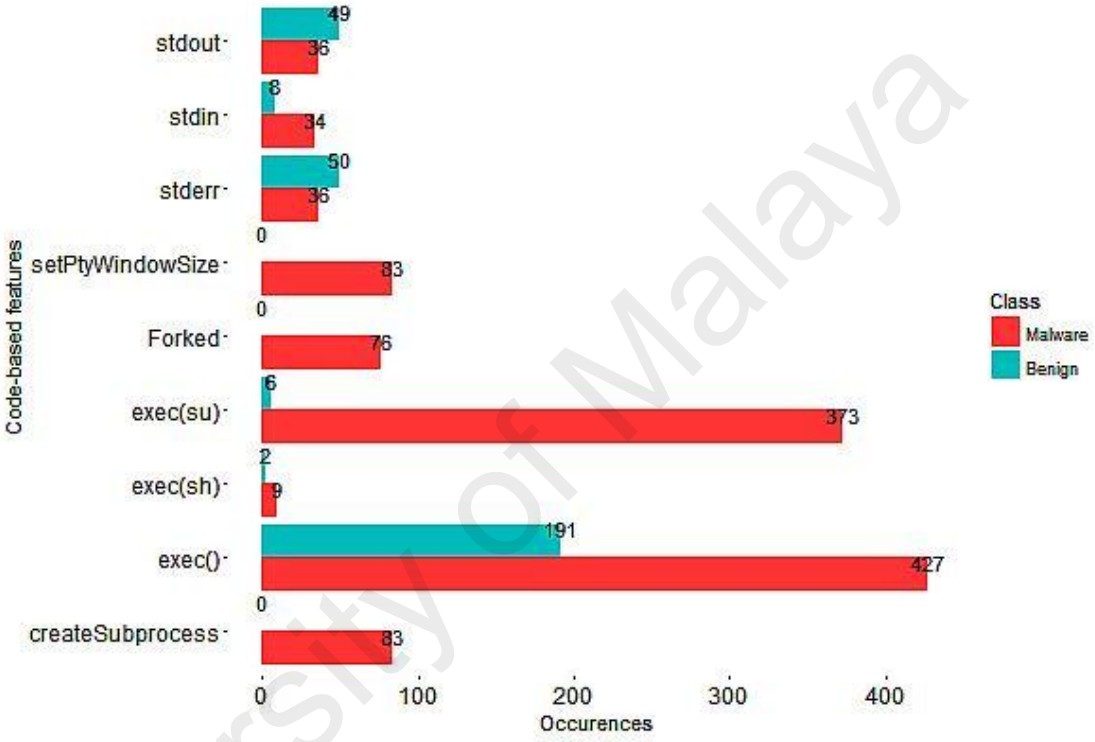


Figure 5.27: Code-based occurrences

Figure 5.28 depicts a graph that combines malware and benign samples, their occurrences, and their categories. The figure indicates the vertical lines from 60 to 500, malware class elevates the area, except one feature (exec ()). Among the 31 features, exec () has the highest occurrences in both benign and malware categories. Moreover, only malware class exists in vertical lines from 300 to 500. In this instance, the directory path type has four features, which is more than system command (two features) and code-based (two features). In this root exploit investigation; this graph demonstrates that directory path is more significant than system command and code-based.

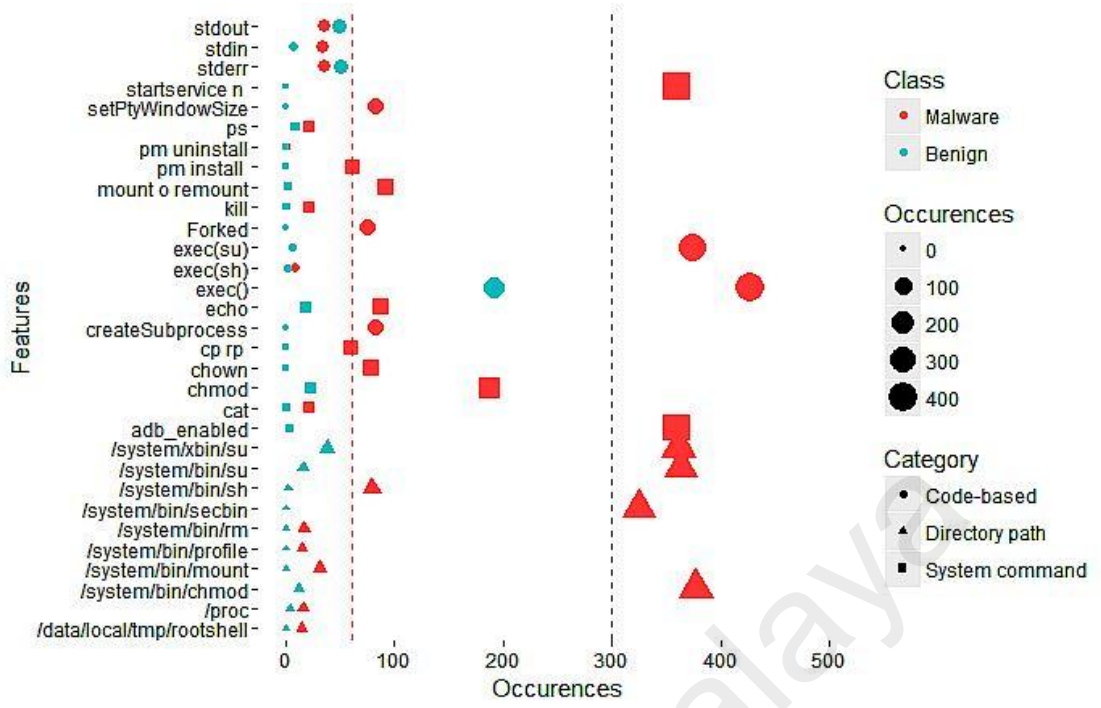


Figure 5.28: The 31 features in categories

d) Feature selection

The process of feature selection involves searching for any suspicious string in all samples (malware and benign) and gathering these strings into one list. This selection process involves a month-long search for each feature in the 1,100 samples. This investigation managed to discover only 31 features. According to the time constraint, this search stopped the investigation and began to select the relevant features. This process is vital because it helps to remove noise and irrelevant data, thereby increasing the accuracy of the results of the machine learning algorithms (Jensen & Shen, 2008).

To select the most relevant features to enhance the machine learning detection accuracy (Spolaôr *et al.*, 2013), this study adopted the feature selection algorithms, such as Information Gain (IG) (Shannon, 1948), Chi-Square (CS) (Imam *et al.*, 1993), and Fisher Score (FS) (Golub *et al.*, 1999). IG determines the amount of information by measuring how well it separates the training examples according to their target classification. CS is used as a test of independence to assess whether the assigned class is

independent of a particular variable. FS expresses the difference between two classes that are relative to a specific feature and considers the mean and standard deviation of the feature values in different classes. This experiment selects IG because of its effective measuring features, generalization capability, accuracy enhancement, and short execution time (Kent, 1982). Table 5.25 shows the IG result. The higher gain ratio indicates the feature's relevance in a classification model for a machine learning classifier.

Table 5.25: Information gain value

No	InfoGain Value	Features	Categories
1	0.44536	startservice –n	System command
2	0.43255	.exec("su")	Code-based
3	0.42553	adb_enabled	System command
4	0.41233	/system/bin/chmod	Directory path
5	0.38765	/system/bin/secbin	Directory path
6	0.37179	/system/bin/su	Directory path
7	0.29953	/system/xbin/su	Directory path
8	0.13976	.exec()	Code-based
9	0.11568	chmod	System command
10	0.0799	setPtyWindowSize	Code-based
11	0.0799	createSubprocess	Code-based
12	0.07805	mount -o remount	System command
13	0.07583	chown	System command
14	0.07279	Forked	Code-based
15	0.06516	/system/bin/sh	Directory path
16	0.0578	pm install	System command
17	0.05682	cp –rp	System command
18	0.0355	echo	System command
19	0.02877	/system/bin/mount	Directory path
20	0.01563	/system/bin/rm	Directory path
21	0.01491	kill	System command
22	0.01491	cat	System command
23	0.01377	/data/local/tmp/rootshell	Directory path
24	0.01377	/system/bin/profile	Directory path
25	0.01178	Stdin	Code-based
26	0.00514	/proc	Directory path
27	0	Ps	System command
28	0	.exec(sh)	Code-based
29	0	Stderr	Code-based
30	0	Stdout	Code-based
31	0	pm uninstall	System command

Table 5.25 indicates only 26 of 31 features are relevant. Thus, this part only considers the features from 1 to 26 and excludes the remaining 5 features. The three most relevant features are startservice –n, exec ("su"), and adb_enabled. The most relevant feature is startservice –n, which is one of the shell commands that initiated the ADB. Meanwhile,

the second most relevant feature is exec("su"), which is a code that gains the super-user privileges in Linux kernel. This feature is the low layer of the Android OS. The third most relevant feature is adb_enabled, which is a code that enables the ADB option to provide ways for a root exploit to enter a device. It is worth noting that in these three top relevant features are the novel elements undiscovered in previous studies. Once this step gains the best 26 features, machine learning used it for evaluation phase in the preceding section.

e) Machine learning classifier

The steps in this section build the machine learning predictive model to expose unknown root exploit malware. In constructing the machine learning model, the classifiers are run in Weka (Hall *et al.*, 2009). In building the model, the initial step is to prepare the Comma Separated Values (.csv) file with the static features (0 and 1). This file contains 27 columns and 1,101 rows. The total of 27 columns is consist of 26 attributes (features), followed by one class column at the end (M for malware and B for benign). The 1,101 rows represent the samples used in this experiment (1,100 samples) and addition with one feature header names, thereby the total number of row is 1,101. Given that this experiment uses static analysis, each sample takes 1 or 0 only. Each row represents an application, which shows 1 (if the feature exists @ occur) or 0 (if the feature is non-existing @ not-occuring).

After setting the total number of features, it is necessary to convert the .csv files to Attribute-Relation File Format (.arff) file using Weka. The reason is, ARFF is an ASCII text file format, which is developed specifically for Weka. As compared with .csv file loads, .arff file loads faster (Williams, 2010). Once the conversion is done, the subsequent step is to randomizes the instances in the file using the randomize option. It is used to constitute a natural data for a machine learning classifier. Figure 5.29 shows

the information in ARFF after the randomize option. Both classes (malware and benign) were crossed to each other repeatedly.

```
@attribute /system/bin/sh numeric
@attribute /system/bin/su numeric
@attribute /system/xbin/su numeric
@attribute /system/bin/chmod numeric
@attribute Class {M,B}

@data
0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,B
1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,1,1,M
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,B
1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,1,1,M
1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,1,1,M
1,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,1,1,M
```

Figure 5.29: Arff file

The subsequent process is to apply 10-fold cross validation, wherein Weka randomly selects the parts of data for training and the remainder for testing. These actions (training and testing) are repeated 10 times to achieve significant results. Particularly, the dataset is randomly split into ten subsets of equal size and repeated ten times. In each repetition, nine subsets are combined to form the training set for constructing the predictive model, while the remainder one subset is used as the test set. To note, this test set is excluded from the training set, which is used to detect unknown root exploit malware in this study. For evaluation, this process executed three machine learning classifiers, namely, MP, RF, and NB. This evaluation conducts the classifier on a desktop computer equipped with Intel Core i7-4770 CPU of 3.40 GHZ, 16 GB of RAM, and Microsoft Windows 7 Professional as an operating system.

5.4.2 Results

In order to evaluate the effectiveness of the proposed 31 root exploit features (system command, directory path, and code-based features), this part of the experiment assessed the performance matrix of the machine learning classifiers according to Table 5.1.

The result in Table 5.26 shows that MLP exhibits the best TPR value at 86.2%. While in FPR value, RF obtains the value of 0.9%, demonstrates that RF is more effective in minimizing mistakes than MP and NB, which obtain FPR values of 1.1% and 2.7%, respectively. In a high precision aspect, it indicates that the model of the classifier is effective, whereas RF obtains the best value (98.9%) as compared with MP and NB. For the next three benchmarks (i.e. recall, f-measure, and accuracy), MLP surpasses other classifiers by achieving 86.2%, 92% and 92.5% respectively. However, NB consumes a short time to build classifier model which indicates that it is useful whenever a situation needs to update a model regularly in a fast manner. This section describes all the aspects of the study, except for the ROC curve value. The next section briefly explains the relevant case.

Table 5.26: Classifier Result

Classifier	**MLP**	**RF**	**NB**
True Positive Rate	**86.2%**	85.3%	83.1%
False Positive Rate	1.1%	**0.9**%	2.7%
Precision	98.8%	**98.9**%	96.8%
Recall	**86.2**%	85.3%	83.1%
F-Measure	**92**%	91.6%	89.4%
Accuracy	**92.5%**	92.2%	90.2%
Time taken to build model (second)	2.81	0.11	**0.02**
Receiver operating characteristic curve	**0.941**	0.936	0.901

a) ROC Performance

In this area, this study used the ROC curve to illustrate the graphical representation of the tradeoff between the TPR and FPR values. This curve is a fundamental indicator for diagnostic test evaluation. TPR (sensitivity) is plotted in the function of FPR for different cutoff points of a parameter. The Area Under the Curve (AUC) is the total area under the ROC curve. An AUC value closer to 1 indicates good classifier performance and high classification accuracy. Figure 5.30 shows the ROC value in the three machine learning classifiers, namely, MP, NB, and RF. As shown, MP obtains the closest value

to 1 at 0.941. This value is higher than those obtained by RF and NB (0.936 and 0.901). The figure indicates MP performs better than NB and RF. Specifically, the line graph of MP is significantly higher and closer to 1. NB has the lowest value, which is far from 1.

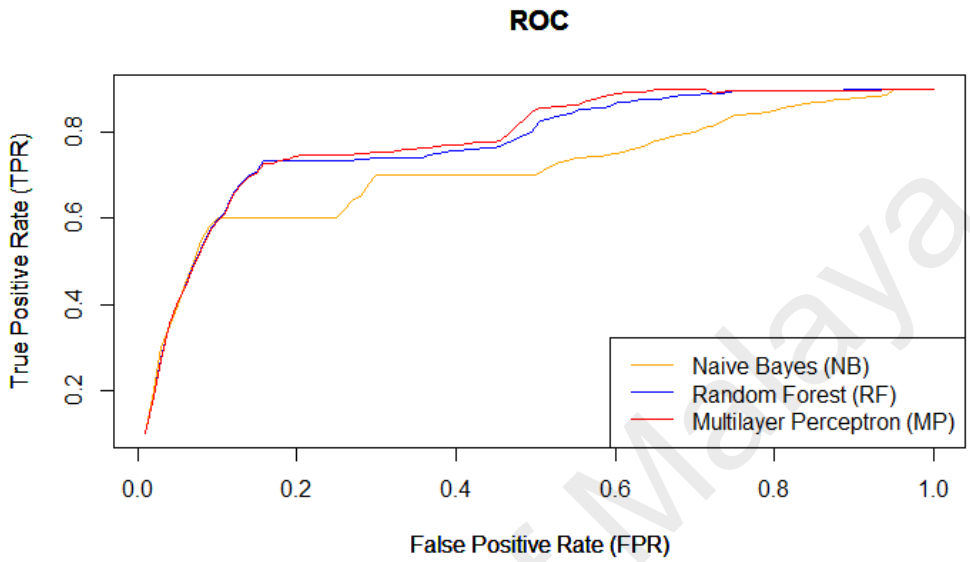


Figure 5.30: ROC curve

5.4.3 Discussion

This section compares and discusses the experimental results. Table 5.27 presents the comparison between TPR and FPR values. It is to investigate the feature performance in distinguishing malware and benign samples, as well as to obtain the capability of the machine learning classifiers. Correspondingly, this table compares these results with the results obtained in previous studies on static and dynamic analyses. The best result value is highlighted in bold. This study selects these previous works for comparison because of two main reasons. One is that they adopt Malgenome as malware dataset, which is similar to this dataset; the other is that these selected studies are published in reputable journals.

Table 5.27: Result comparison

Type of analysis	References	Classifiers	Result		Dataset		Features
			TPR	FPR	Benign	Malware	
Static	Ours	MLP	86.2%	1.1%	550	550	10 System commands, 10 directory paths, 6 code-based features (a total of 26 features)
		RF	85.3%	**0.9%**			
		NB	83.1%	2.7%			
	(Yerima, Sezer, & McWilliams, 2014)	Bayesian	**90.9%**	5.1%	1000	1000	15 selected mixed features (permission and code-based features)
Dynamic	(Narudin *et al.*, 2014)	MLP	94.83%	5.17%	20	1000	11 network traffics
		RF	**99.96%**	**0.04%**			
		Bayesian	99.88%	0.12%			
		KNN	98.73%	1.27%			
		J48	99.9%	0.1%			

The static analysis results show that the TPR result of the bayesian classifier in (Yerima, Sezer, & McWilliams, 2014) is higher than the results when amounts of features and dataset malware samples differ. Specifically, their classifier obtains 90.9%, whereas this thesis classifiers only obtain 86.2%, 85.3%, and 83.1%. This finding proves that this bayesian classifier obtains high TPR values when the features are less and the dataset is high. However, this FPR result is significantly better. In this benchmark, a low value indicates few mistakes in classifying a benign sample as malware sample. This result demonstrates that this feature is more effective in detecting root exploit in Malgenome dataset. Yerima *et al* (2014) focused mostly on general malware types, whereas this research focuses on root exploits only.

The dynamic analysis results indicate that this experiment's TPR result is low because of malware samples. The TPR values in (Narudin *et al.*, 2014) are higher than this experiment TPR values. Notably, this study used 550 samples whereas 1,000 samples are used in (Narudin *et al.*, 2014). This limitation suggests that this investigation is able to obtain better results by increasing root exploit malware samples. In features comparison, this part used 26 features whereas their study used only 11 features. This

distinction indicates that few features yield accurate results in static and dynamic analyses. In future work, this research attempts to decrease the 26 features to obtain the promising results.

Nonetheless, in MLP comparison, this work's value is better than (Narudin *et al.*, 2014) study. In their study, MLP obtains a value of 5.17%, which is higher than this experiment's study (only 1.1%). Furthermore, MLP is better than another classifier in the static analysis in both TPR and FPR. This comparison verifies the MLP works better in static analysis than in dynamic analysis.

To summarize, this experiment adopts the static analysis method which reverses engineer the application and examines the sample of a code without executing it. In detecting the malware process, this step consumes lesser time than dynamic analysis, which runs the samples first and then observes its behavior. Although the results are lower than those of dynamic analysis, this experiment proposed analysis is significantly faster because it classifies the samples without executing them. Furthermore, certain malware samples are able to avoid dynamic analysis by acting as normal samples during the monitor operation. Furthermore, covering all possible activities and executing each sample are time-consuming.

5.5 Summary

This chapter has discussed the evaluation study of the selected static features derived from the investigations and methods used in the proposed framework. The useful results from the experiments have demonstrated a combination of different aspects of evaluation, and they highlighted their unique findings and conclusions.

The key objective of describing the evaluation at different experiments of studies is to investigate the unique objectives at each experiment. The result presented has shown

strong evidence to support the ability of the proposed framework to work robustly based upon its operational characteristics. Furthermore, the comparison study in the evaluation studies also strengthens the framework and its suitability to facilitate the anomaly-based malware detection using static analysis. In conclusion, the analysis made of the studies clearly defined their contribution as well as stating their limitations.

To further investigate the usefulness and feasibility of the proposed framework in a practical mode, the following chapter presents the prototype of the proposed framework and evaluates it using different datasets to the one used in this chapter, in order to test the efficiency in predicting unknown malware.

CHAPTER 6: PROTOTYPE IMPLEMENTATION OF MOBILE MALWARE DETECTION SYSTEM

Once the validating and evaluating the approaching framework are done, the following stage of the research is to design and achieve a prototype program that verifies its main operations and bring to light how these able to be achieved practically. This chapter describes the prototype implementation of the proposed framework for mobile malware detection using static features. The main features of the malware detection have been personified in a web interface, which can be used to upload the Android application and predict it either malware or benign. Several modeling languages, including use case diagrams and state diagrams, are used to provide a visual illustration of the prototype. Finally, this chapter used different malware dataset from the machine learning training phase to test the efficiency of the models.

6.1 Web implementation overview

Figure 6.1 illustrates the three modules in the web development with three analyzers. This implementation is based on Java, Javascript, and HTML. The stages are reverse engineering, feature extraction, and prediction. The details of the stages are:

a) **Reverse engineering:** This module reverses engineers the Android application package (.apk) file to obtain the entire folders consist of files that end with Java extension (.java). This module used an open source tool known as apktool to execute this process.

b) **Feature extraction:** To extract the proposed features, this module searches the features in overall files include the files in the nested folders.

c) **Prediction:** This prediction module used the proposed features as input for the machine learning classifier to predict the class of the uploaded file either malware or

benign. This stage consists of three analyzers with three models. First analyzer is Root analyzer, second is Genetic analyzer and third is Bio analyzer.

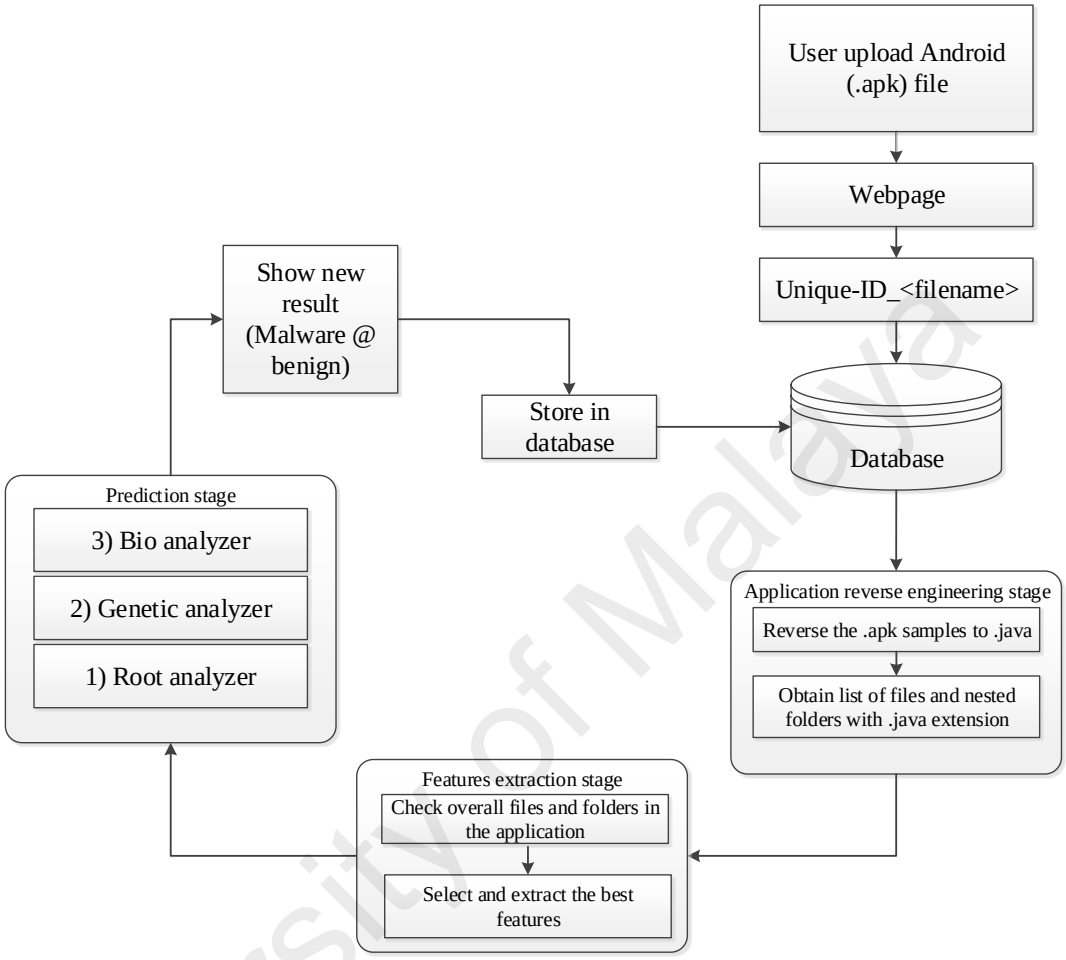


Figure 6.1: Web development

6.2 Prototype functionalities

In order to gain understanding into the main functionality of the proposed framework, this section presents the modeling languages such as use case diagrams and state diagrams.

6.2.1 Use case diagram

Use case modeling has been commonly adopted to plot a graphical functional explanation of the interaction between external entities and systems, in addition to their cooperation. The diagrams are utilized to determine the characteristics of the developed systems, without the necessity to mention how those characteristics are implemented.

Figure 6.2 demonstrates the system level and explains the relationship between external systems. Following explanation provides the role of the user in the figure:

a) Users are able to manage the web modules that represent the run application and download file modules. This includes the ability to upload the .apk file and download the csv and arff files consist of the information of features in each application uploaded by the user.

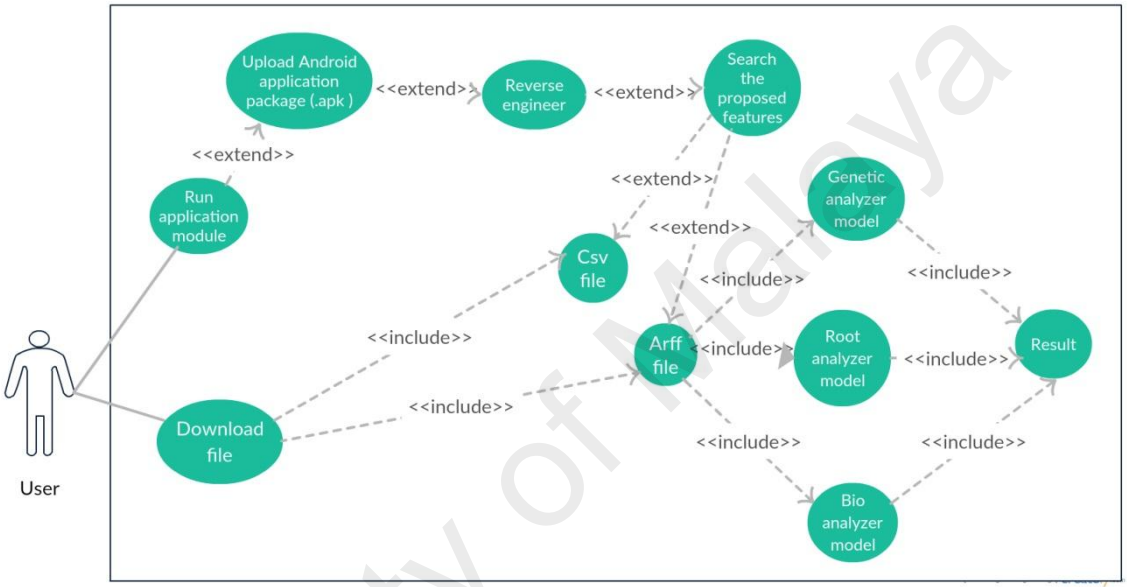


Figure 6.2: Use case diagram

The use case diagram has obtained a short-term summary of the modules' functionality. However, it lacks clarification on how those modules are operated. Hence, the state diagrams are utilized in the subsequent section.

6.2.2 State diagram

A state diagram designates all the possible states of an object as an event occurs, and is used to show the characteristics of an object by using many use cases of a system, in addition, to focusing on the flow of control one state to another. Figure 6.3 reveals all the possible states in the proposed framework, and it summarizes the characteristics of the running system.

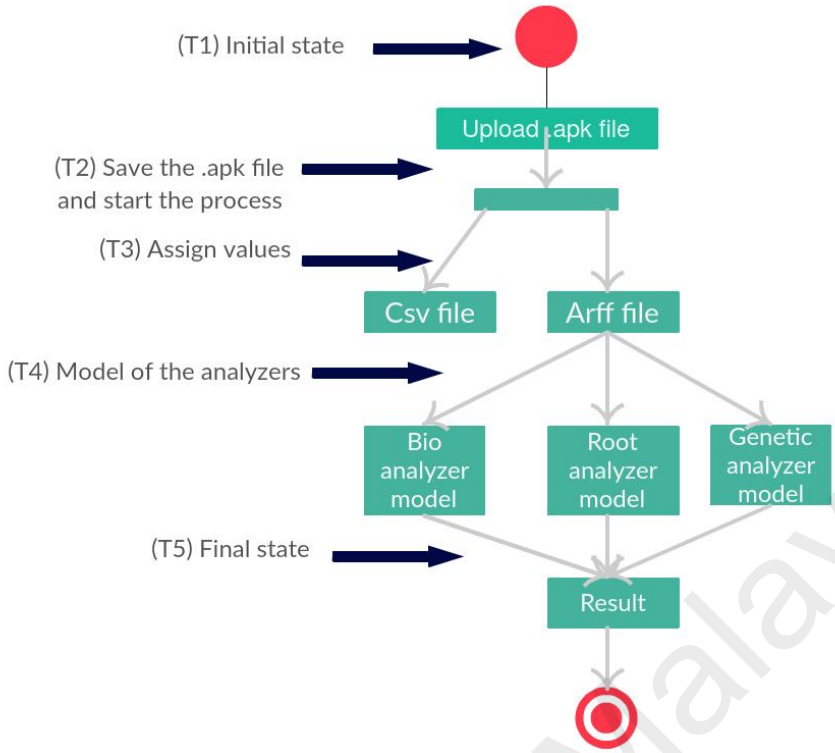


Figure 6.3: Prime-state diagram

As prime-state, there are three sub-states and short-term summary of it are provided below:

a) **Save the .apk file:** The initial state is T2.1 when the user uploads the .apk file. There are four sub-states in this specific state, as shown in Figure 6.4: *Database saved the .apk file, Identification of .apk file, Reverse engineer command and Search the proposed features.* In T2.2, the system identifies the extension file, either it is .apk or other types of file. The process continues only the uploaded file is .apk only. The T2.3 state starts the reverse engineer command to obtain the code and search the proposed features in overall files (T2.4). In the end of this state, the system sampled the uploaded .apk file with the proposed features.

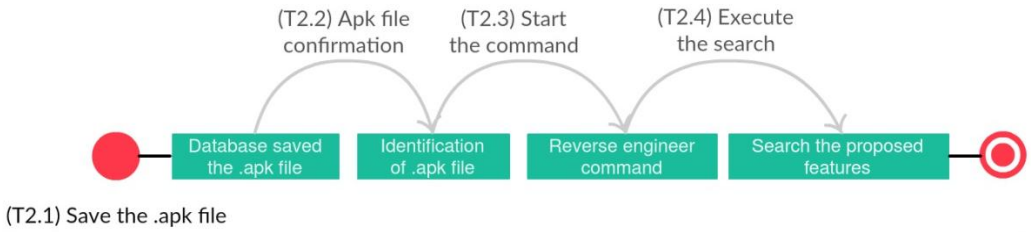


Figure 6.4: Save the upload file state

b) **Assign value:** This state assigns the feature vectors by creating the csv and arff files. These files contain the information of each feature and there are important for the models of the analyzers to predict the uploaded .apk file.

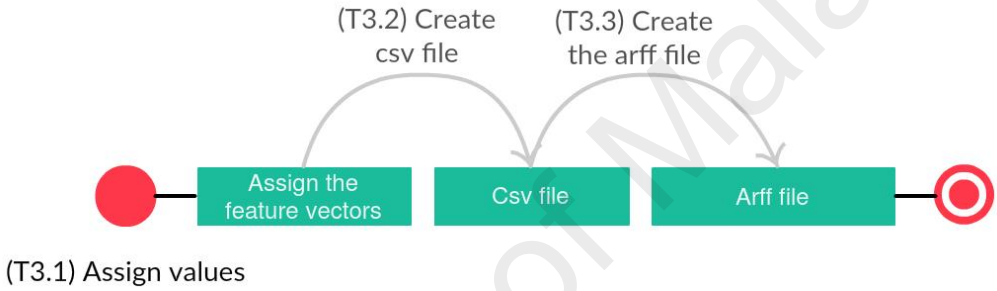


Figure 6.5: Assign value state

c) **Model of the analyzers:** This state receives the arff file and predicts according to the analyzers (Root, Genetic and Bio). Root analyzer predicts the unknown root exploit, Genetic analyzer predicts by using the proposed features from the GS bio-inspired feature selection method, while Bio analyzer predicts by using the proposed features and bio-inspired classification.

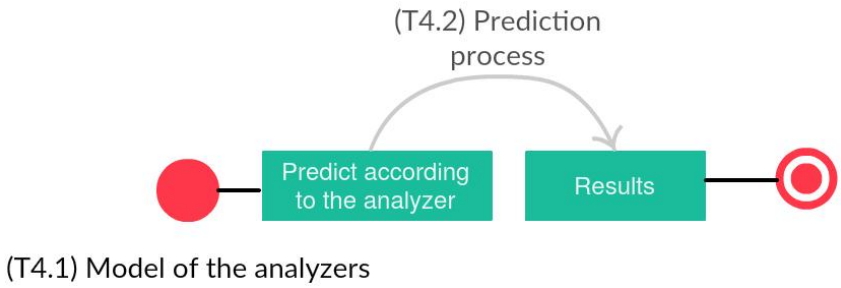


Figure 6.6: Model of the analyzers state

6.3 Demonstrating the malware detection

The main functionalities of the proposed framework and its modules are presented in Section 6.2. However, this section demonstrates the three analyzers in predicting unknown malware. The similar stages in these analyzers are application reverse engineering, features extraction and prediction.

a) **Application reverse engineering:** After the user uploaded their desired Android application package file (.apk) on the first page of the website, the system will grant a unique identification number followed by the name of the file to avoid duplication.
b) **Feature extraction:** Afterward, the system reverses engineer the file to obtain the entire code that ends with Java extension (.java). This system continues the process by searching overall files including in the nested folders in each application to extract the proposed features.
c) **Prediction:** Finally, the system used the features as input for the machine learning classifier to predict the class of the uploaded file either malware or benign.

The differences between these analyzers are the proposed features and machine learning classifiers. For demonstrations of the analyzers, these prototypes used similar hardware specification consist of desktop computer equipped with Intel Core i7-4770 CPU of 3.40 GHZ, 16 GB of RAM, and Microsoft windows 7 professional as an operating system. This section begins with Genetic analyzer in the following section.

6.3.1 Genetic analyzer prediction system

In Section 5.3, FT is the outstanding classifier that achieves the best prediction in detecting unknown malware in the simulation by adopting the best features selected by GS. Therefore, in order to test the genetic-selected features as well as FT machine learning classifiers in detecting unknown malware, this study developed an intelligent

prediction system in website environment called as Genetic Analyzer. Figure 6.7 depicts the architecture of the proposed prediction system.

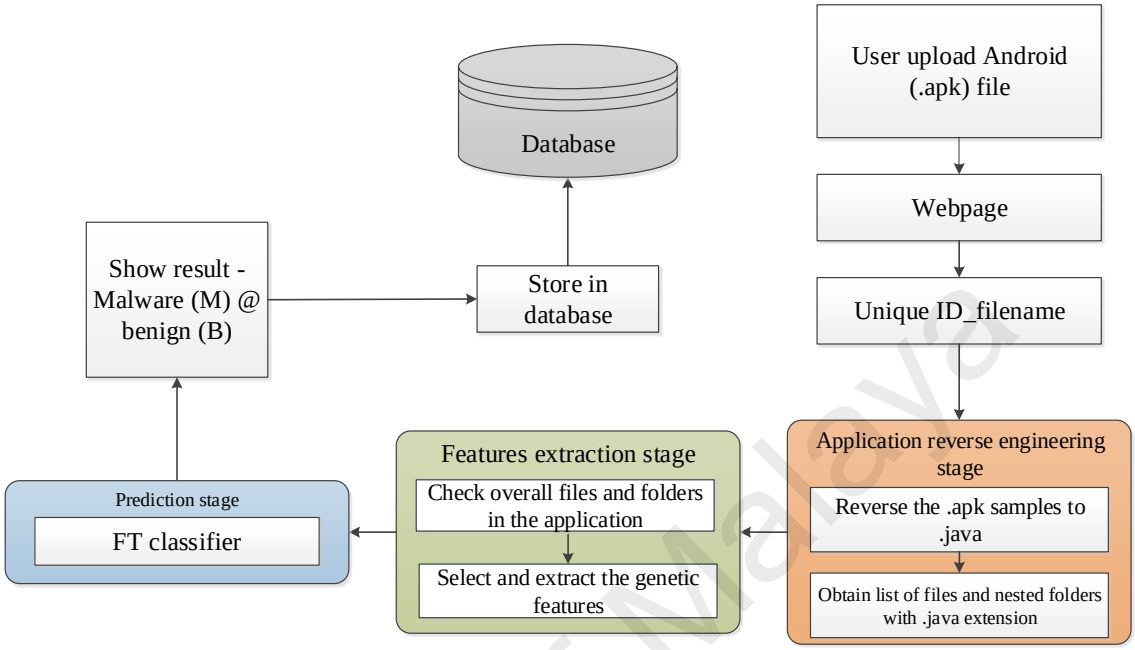


Figure 6.7: Genetic analyzer architecture

Figure 6.8 depicts the upload zone on the first page of the Genetic analyzer. Meanwhile, the subsequent section is the evaluation result of the prediction by using the proposed features.

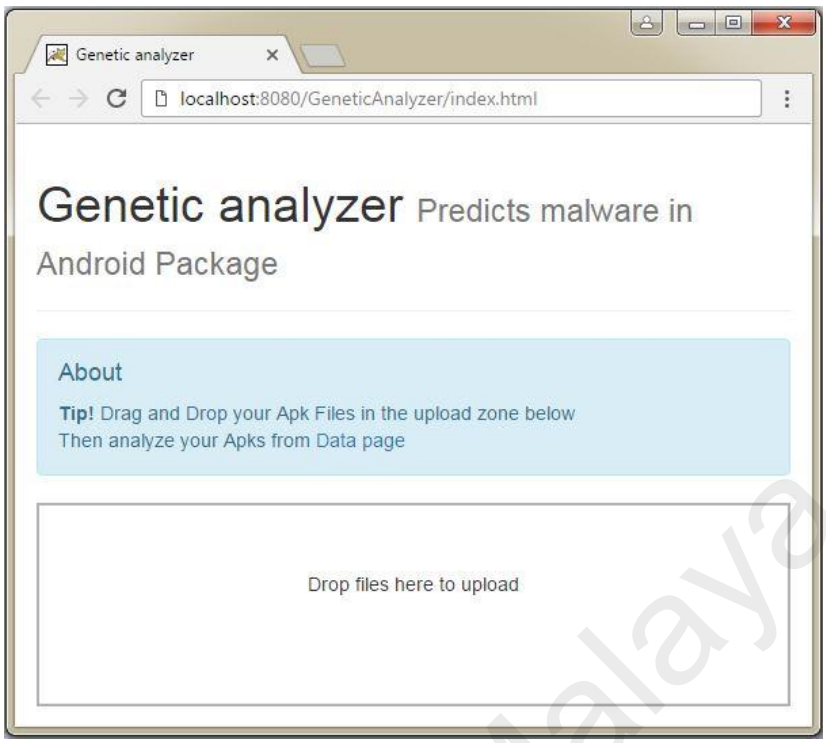


Figure 6.8: Main interface of Genetic analyzer

a) Genetic analyzer result

In order to evaluate the efficiency of the prediction system, it is important to use different dataset during the simulation. Therefore, as this study has used known malware for learning the detection model which is Drebin in the simulation, this practical test utilized another malware dataset called Malgenome. Figure 6.9 displays the ongoing process after this investigation uploaded the Malgenome files. As shown in the figure, the two boxes highlight the bar as black indicated that the processes for those two applications are finished. The bar in the third box is white indicated that the process for that application is still in process. Meanwhile, Figure 6.10 depicts the prediction results (i.e. M indicates as malware and B indicates as benign).

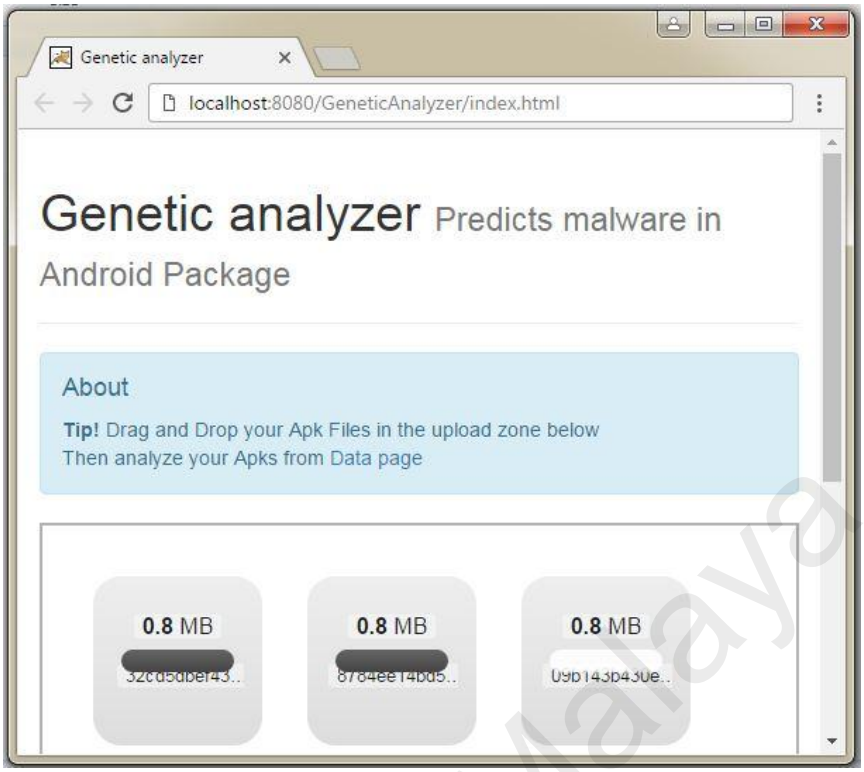


Figure 6.9: Uploading process of Genetic analyzer

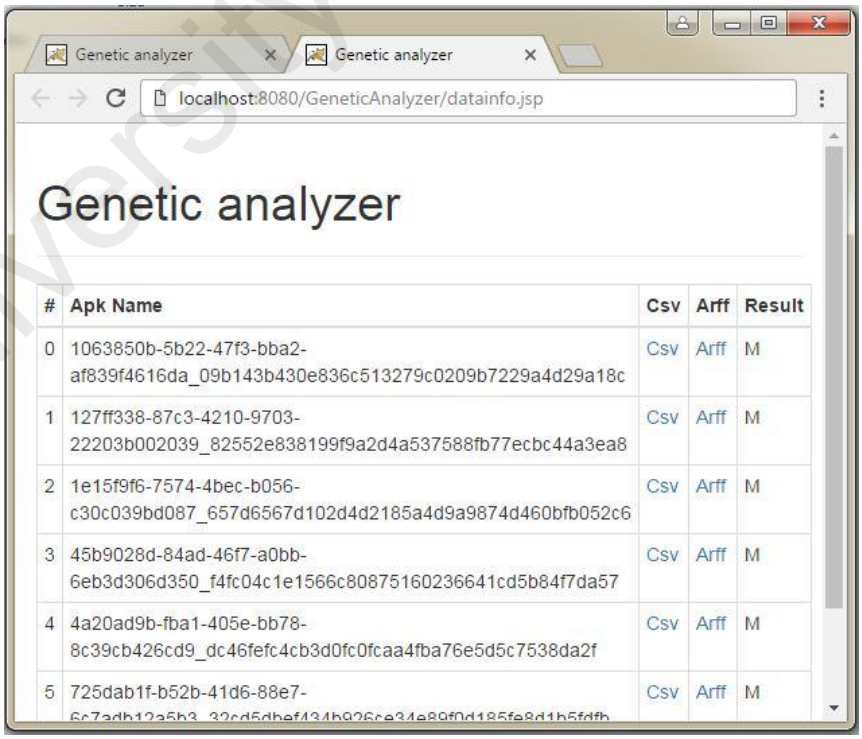


Figure 6.10: Result page of Genetic analyzer

After the system finish processed all the Malgenome samples, this intelligent prediction result shows an outstanding accuracy of 95% in detecting malware. Hence, this accuracy value proves that the proposed features selected from bio-inspired GS combined with FT classifier, are capable to predict unknown malware. In the interest to discover result from another analyzer, the following section provides the Bio analyzer prediction system.

6.3.2 Bio analyzer prediction system

In Section 5.4, MLP is the best classifier that achieves the best prediction in detecting unknown malware in the simulation. Therefore, in order to test the best-proposed features as well as MLP machine learning classifiers in detecting unknown malware, this study developed an intelligent prediction system in website environment called as Bio Analyzer. Figure 6.11 depicts the architecture of the proposed prediction system. While the following figure - Figure 6.12 shows the upload zone on the first page of the Bio analyzer. Meanwhile, the subsequent section is the result of the prediction by using the proposed features.

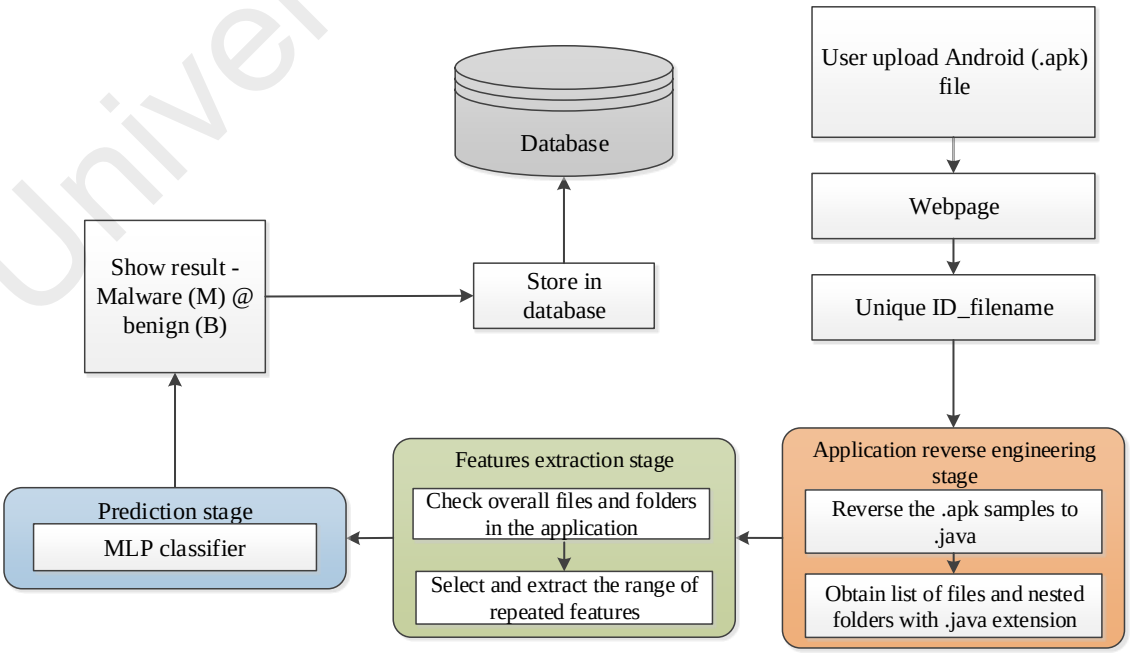


Figure 6.11: Bio analyzer architecture

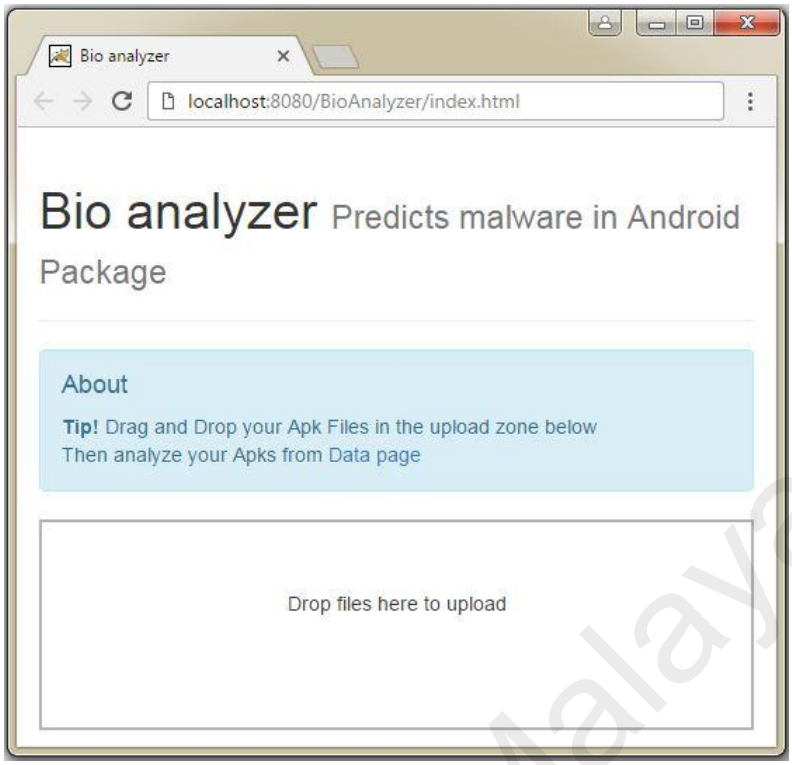


Figure 6.12: Main interface of Bio analyzer

a) Bio analyzer prediction result

In order to evaluate the efficiency of Bio analyzer prediction system, it is important to use different dataset during the simulation. Therefore, as this study has used known malware for learning the detection model which is Drebin in the simulation, this practical test utilized another malware dataset called Malgenome. Figure 6.13 displays the ongoing process after uploaded the Malgenome files. As shown in the figure, the two boxes highlight the bar as black indicated that the processes for those two applications are finished. The bar in the third box is white indicated that the process for that application is still in process. The next figure is Figure 6.14 which displays the second page of the Bio analyzer that provides the prediction results (i.e. M indicates as malware and B indicates as benign).

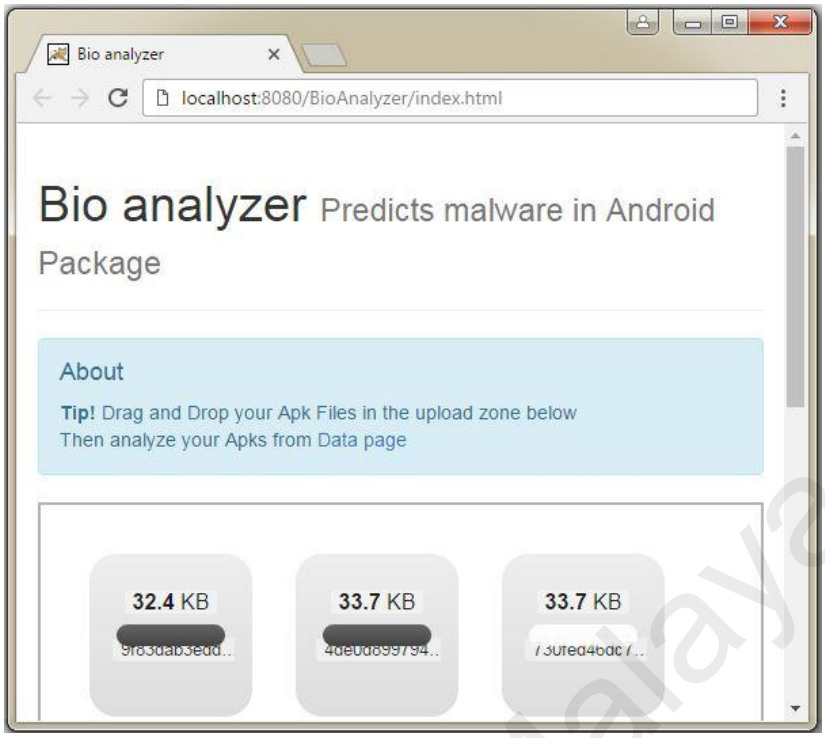
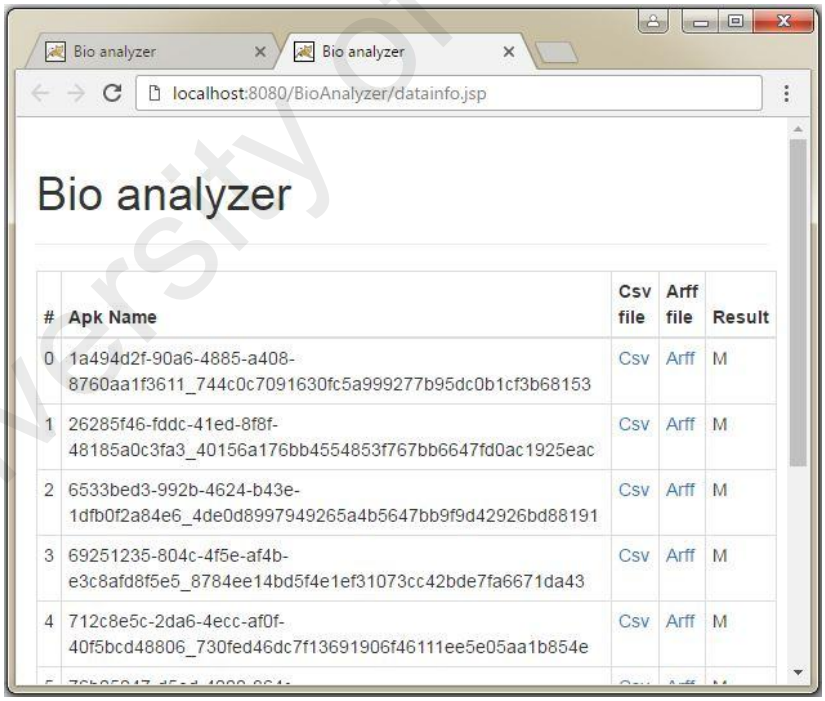


Figure 6.13: Uploading process of Bio analyzer



#	Apk Name	Csv file	Arff file	Result
0	1a494d2f-90a6-4885-a408-8760aa1f3611_744c0c7091630fc5a999277b95dc0b1cf3b68153	Csv	Arff	M
1	26285f46-fddc-41ed-8f8f-48185a0c3fa3_40156a176bb4554853f767bb6647fd0ac1925eac	Csv	Arff	M
2	6533bed3-992b-4624-b43e-1dfb0f2a84e6_4de0d8997949265a4b5647bb9f9d42926bd88191	Csv	Arff	M
3	69251235-804c-4f5e-af4b-e3c8afd8f5e5_8784ee14bd5f4e1ef31073cc42bde7fa6671da43	Csv	Arff	M
4	712c8e5c-2da6-4ecc-af0f-40f5bcd48806_730fed46dc7f13691906f46111ee5e05aa1b854e	Csv	Arff	M

Figure 6.14: Result page of Bio analyzer

After the system finish processed all the Malgenome samples, the intelligent prediction result shows an outstanding accuracy of 97% in detecting malware. Hence, this

accuracy value proves that the proposed features derived from the novel technique, combined with bio-inspired MLP classifier are capable to predict unknown malware.

6.3.3 Root analyzer prediction system

In the Section 5.2.1, MLP is the best classifier that achieves the best prediction in detecting unknown root exploit in the simulation. Therefore, in order to test the best-proposed features as well as MLP machine learning classifiers in detecting unknown root exploit, this study developed an intelligent prediction system in website environment called as Root analyzer. Figure 6.15 depicts the architecture of the system.

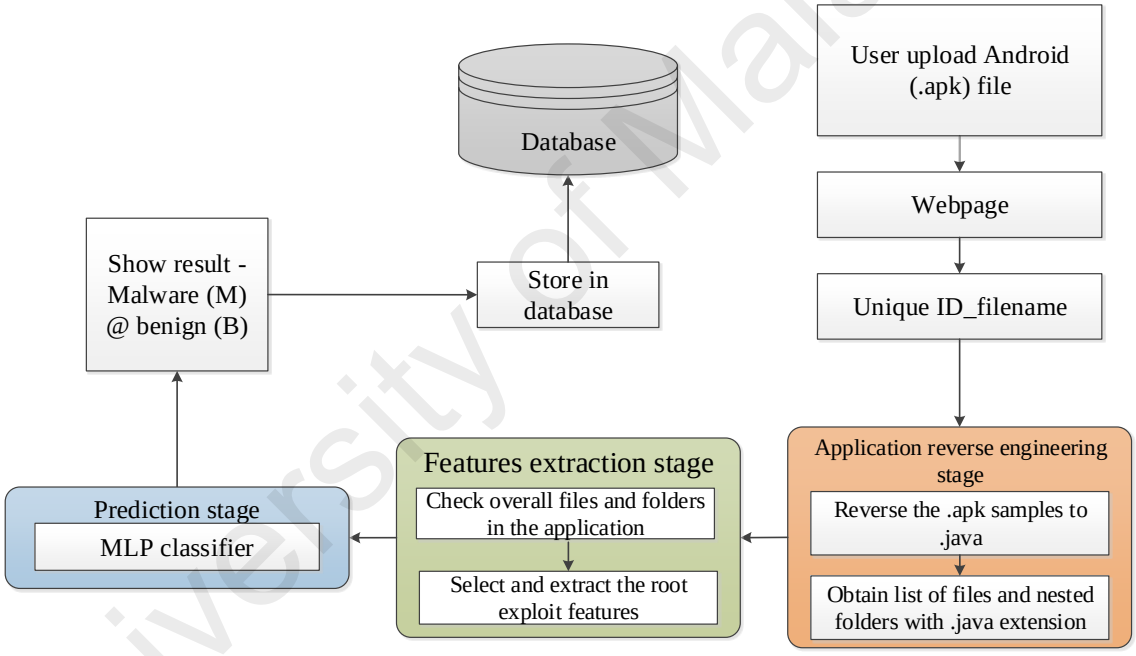


Figure 6.15: Root analyzer architecture

The following figure - Figure 6.16 shows the upload zone on the first page of the Root analyzer. Meanwhile, the subsequent section is the evaluation process of the prediction result.

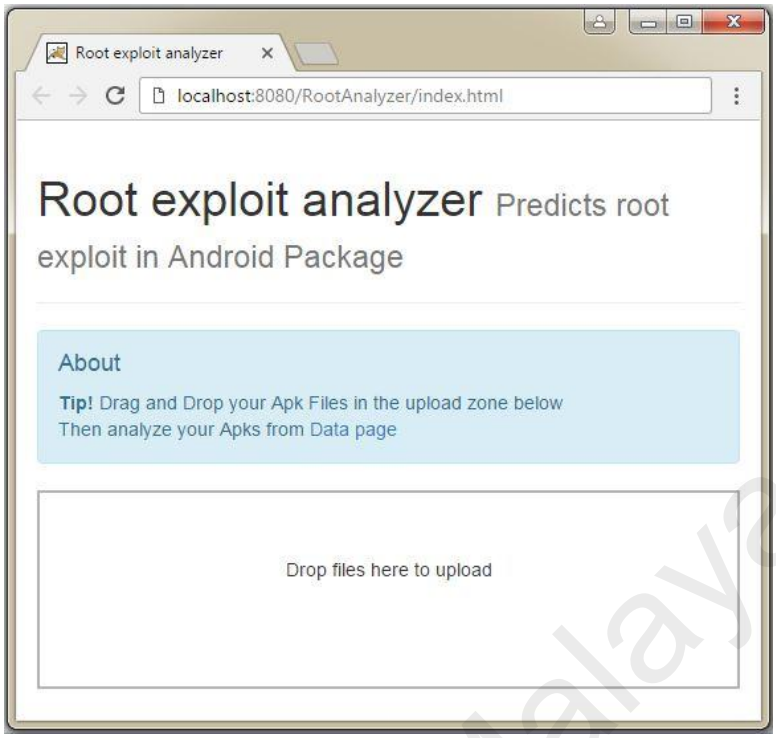


Figure 6.16: Main interface of Root Analyzer

a) Root analyzer result

In order to evaluate the efficiency of the prediction system, it is important to use different dataset during the simulation. Therefore, as this study has used known root exploit for learning the detection model which is in Malgenome for the previous simulation, this practical test utilized different root exploit samples that included in Drebin. The samples are Droidrooter and Rooter families with each family represent three applications. Furthermore, these samples are excluded in this experiment's learning machine learning detection model. Table 6.1 lists the root exploit in detail. Figure 6.17 shows the ongoing process after user uploading the .apk file. As shown in the figure, the two boxes highlight the bar as black indicated that the processes for those two applications are finished. The bar in the third box is white indicated that the process for that application is still in process. Meanwhile, Figure 6.18 depicts the prediction results (i.e. M indicates as malware and B indicates as benign).

Table 6.1: Root exploit information for prediction system testing

Family	Sha256	Size (kilobyte)
Rooter	1f5a97fb0cbaa2e10e1f080571ae081d9d85fc95519ef59a85b83ca366b10df2	13
DroidRooter	226dc739a76faf5127a245b9cc759d4db3086710d4e71594c5578ae642774f5c	950
DroidRooter	94112b350d0feceff0a788fb042706cb623a55b559ab4697cb10ca6200ea7714	862
Rooter	94ea44688feb558e2786e52fbfa46d90984e40c0980e28035fd2311d5f17f8e3	13.7
Rooter	add10b0368753ec38de0dca15550d824ac141f0c86f2f123f30551bd82e82415	13
DroidRooter	edf568790907e970da583855e9b923b2f897fbeb4faf41b87436b23e262b821a	953

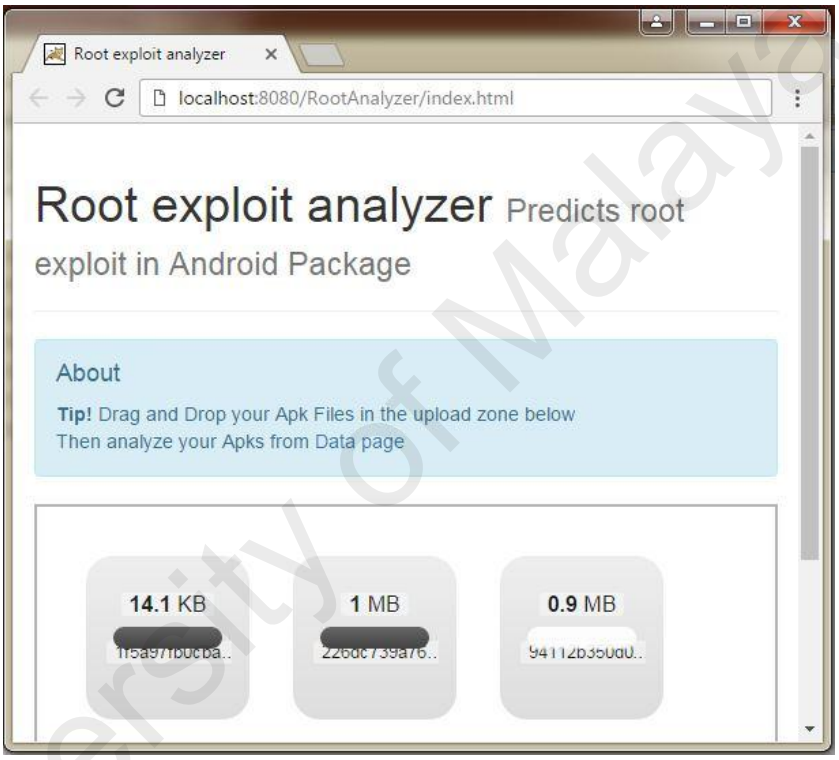


Figure 6.17: Uploading process of Root analyzer

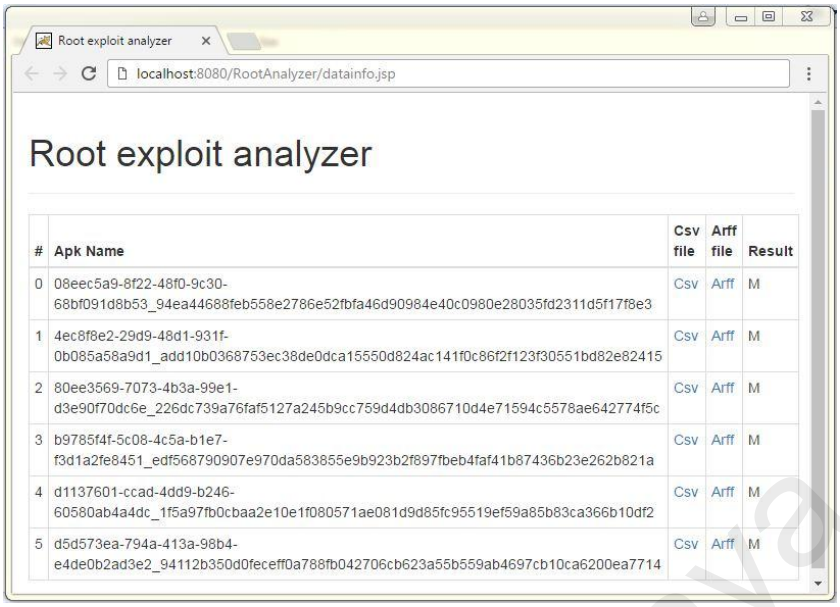


Figure 6.18: Result page of Root analyzer

After the system finish processed the entire root exploit samples, the system successfully predicts all the six samples as root exploit as shown in Figure 6.18. Hence, this prediction proves that the proposed features with novel ADB type are capable to predict unknown root exploit. In the interest to discuss another analyzer, the next section provides Genetic analyzer prediction system.

6.4 Performance of the analyzers

One of the static analysis advantages is rapid processing. Therefore, this section provides ten benign samples to test the performance of each analyzer (i.e. Genetic, Bio and Root) in processing the detection. Table 6.2 tabulates the result of the performance and detail of each sample. Figure 6.19 depicts this result in graph manner for easier observation.

Table 6.2: Ten benign samples for performance testing

Time (second)			Size (MB)	Sha256
Genetic	Bio	Root		
18	21	43	0.627	018613c2e4174b5251e0d41963a34067ab9ad38d844718542847e4d854c8713d.apk
49	52	54	1.53	01c6d025efb072d7ba693d448adb159e8cff312555462bbe9aa1fe377a9caca2.apk

50	56	70	5.34	01c708d27ff56ceb8c3d63ed782ffd7fcd9c5b38717b53690191a049f43f4b7b.apk
81	90	104	10.4	017373928016819937f701baeb12699592e44b345eba84e0aacb26958872f1c3.apk
102	111	119	12.6	014f3e37b9a33305ae6e7c110b7ad41962fb069abf148197ec4393f8c57e7909.apk
125	155	160	15	013d29b38cd765fb4a0f1dbd5b1ccae0e16810ff0377da915e958024d8e93e88.apk
133	161	175	16.3	01bf9c2de95ab5491b7d9b823bf3fc5cb8278eeb729d00831118247d12c7c189.apk
135	167	179	18.7	01dd3520955373acb27b755b15f4944a8415cc98003bfb794f4bf566fa2e897d.apk
140	170	202	23.5	018c46baba3e010cb87bb42d426fffce9fede0b871dd84974683ac4178332c31.apk
153	177	210	26.2	01b3e63b2d4592d3b958380a4aa7a2a911e4d03ea030760f02b5093048878477.apk
179	193	218	28.5	018d984b22bae8e9352a2097c0fee85178de7279f56c4b600ac1127da7b5da17.apk

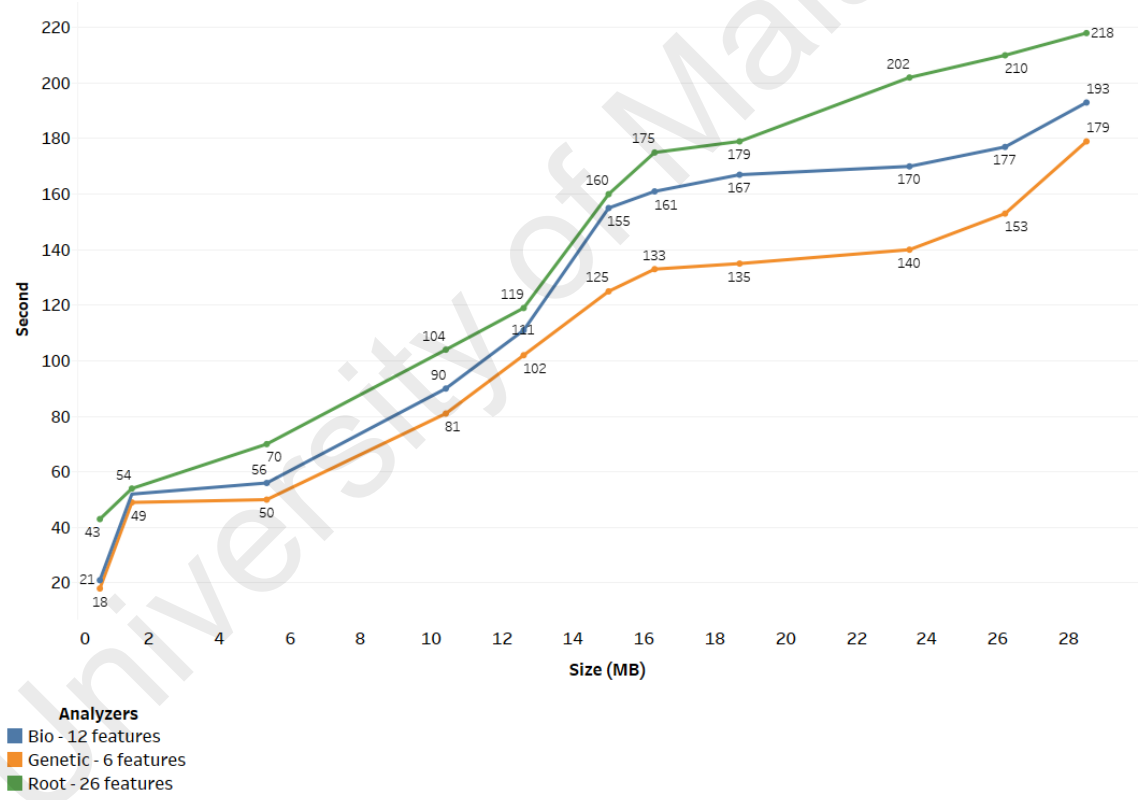


Figure 6.19: Result of the performance

In comparison to other analyzers, genetic analyzer utilized only 6 features and shows the significant gap. As root analyzer utilized more features than other analyzers, it processed much longer and consumed much more time in detection. This proved that machine learning processed much faster with less features. Moreover, this experiment

proved that static analysis processed in fast manner which only need 281 second for 28.5 MB sample. In order to sum up all the advantages and limitations of this thesis framework, next section provides this information in detail.

6.5 Advantages and limitations

In providing a flexible platform to security analysts for configuring, analyzing and making a wise decision using the prediction results, the web modules give the following advantages:

a) **Less confuse results:** The web modules provide only two classes either the uploaded file is malware or benign. If the results indicate more than two classes, the user may confuse the uploaded file is safe or dangerous to install on their mobile device.

b) **Provide csv and arff files:** The web modules also provide the two important files (i.e. csv and arff) that available to be downloaded after the prediction process is done. These files are useful for security analysts to test in other detection systems, other simulation applications or discovering the features of the uploaded application.

c) **User friendly interface:** The web modules provide an easy interface to ease a novice user to use the systems. By adopting this friendly interface and few buttons, many users are able to predict their Android application package file either malware or benign.

d) **Able to utilize it in mobile browser:** The modules are based on web and therefore it is available for internet browser (i.e. Opera, Chrome, Firefox) on mobile devices. Therefore, the users are able to use the web modules in normal desktop computer as well as on the mobile devices.

e) **Rapid processing:** As the web modules adopted static analysis, it only involves reverse engineer the application and searches the features in the code without executing the application in a certain range of time to monitor the behavior. These

light-weight static analysis processes only need a short time and low specification of hardware.

In addressing the advantages of the web modules, they also inherit some limitations as follows:

a) **Applications Dependent:** As the web is served by utilizing a web server, it depends on the productivity of the web server itself, in the event of the server is disconnected, the prediction procedure is impossible. Furthermore, as it also based on the World Wide Web, it additionally depends upon the system consistency to import and trade information.
b) **Acquire different vulnerabilities:** As the web utilized the web applications, it is open to the web application vulnerabilities (i.e. cross-website scripting, SQL injection, HTTP Parameter Pollution (HPP), and session hijacking). In addition, the web modules also powerless against alternate vulnerabilities, such as equipment (e.g. web server) and software (e.g. internet browser).

Therefore, for future works, it is imperative to address these limitations by using other security precautions and countermeasures.

6.6 Summary

This chapter presented the implementation stage of the proposed framework by providing some examples and snapshots from the web modules consist of three analyzers namely Root, Genetic and Bio analyzers. The details of its modules, system architecture, state diagrams and web modules have been presented to show how it interrelates.

The reasons of presenting and demonstrating the details of the modules are to provide a better understanding of how the proposed framework interacts, and how the internal

modules are affected by the external interactions. The following chapter is the conclusion part that briefly discussed the limitations among others.

CHAPTER 7: CONCLUSION

This chapter outlines the study by revisiting the research aim and objectives as well as findings. The most important finding, in addition to its limitations, are reemphasized. The capability of alternative studies in the same domain was also exploited and this was then used to develop the proposed framework which could be used to improve future research works in the same domain.

7.1 Research objectives

This study aims to develop an intelligent anomaly-based detection system using static analysis and the machine learning approach. Section 1.4 had described the four research objectives of this study. It had also maintained how the study would accomplish its research aim by fulfilling the following research objectives. This section is to answer the following research questions: a) RQ 1: What are the best features to detect malware in static analysis using machine learning? b) RQ2: What method should be applied to search for the best features in minimal amount? c) RQ3: What are the best features to detect malware in static analysis using machine learning? d) RQ4: What are the specific features to detect particularly on root exploit?

Objective 1: To review the domain of Android static analysis and its key issues.

The first objective was to critically investigate the current state-of-the-art malware detection. The research objective was accomplished by conducting a thorough review of the most crucial works published in online scholarly journals extracted from digital libraries which were accessed through the University of Malaya's access portal; they include the Institute of Electrical and Electronics Engineers (IEEE), the Association for Computing Machinery (ACM), Elsevier and the Web of Science portals. Recent literature extracted from journals and conference papers were also scanned, focus and investigation were given to the analytical issues. In addition, recent studies which

focused on the detection taxonomy, the machine learning approach and the issues related to features were also reviewed. The intention was to fulfill the review's thoroughness. Consequently, this study was able to reveal that using an approach that combines static analysis with anomaly based detection would offer a higher potential in uncovering unknown malware. The outcome offers results that are easily gained from a process that is rapid and resources that are low (i.e. CPU, memory, network and storage).

Objective 2: To establish the need for an intelligent intrusion detection system by using static analysis and machine learning to identify the best features in minimal amount.

The second objective of this study was to establish the information of static analysis in anomaly based detection by using the intelligent prediction of the machine learning approach. The outcomes gathered from this study would highlight the advantages and disadvantages of the signature versus anomaly based detection. From this study, the difference between static analysis and dynamic analysis was also noted. It is hereby reiterated that this study followed the path of examining the minimal features in malware detection using static analysis and its impact on the detection results. This study also explained the uniqueness of the study in focusing on the selection of features (i.e. GS and range of repeated features in similar application). It is complemented by a section that explains how the root exploit features manifest themselves and the need to conduct a search for features in overall files rather than one particular file. This study also stressed that it needs to refer to the official list as a main source of credibility.

Objective 3: To design and develop a novel framework by applying the proposed features in the intelligent intrusion detection system.

The third objective of this study was to design a prototype of malware detection based on a novel framework according to the exclusive features discovered in the research. This study then devised a web based system to predict whether the Android application package (.apk) file is malware or benign. The .apk file is uploaded from the user; the system reverses the engineering, identifies the features, extracts the features from the application, and then identifies the classes of application with the intelligent prediction. From the review of literature, this study had selected the best machine classifier by comparing it with the Weka results. To gain clearer results from the examination of certain features and types of malware, the system used in this study would be further enhanced by three analyzers: Genetic, Bio and Root analyzers. These will provide the Comma Separated Values (CSV) and Attribute-Relation File Format (ARFF) files for security practitioners to conduct further investigations.

Objective 4: To evaluate the proposed features to detect unknown malware as well as the features specifically in root exploit in terms of accuracy and performance.

The fourth objective of this research was to evaluate the exclusive features noted in unknown malware detection. Thus, the evaluation of these features was examined in two platforms: a) Weka and b) Prototype. In the Weka simulation, the experiments tested the features in six evaluation measures: accuracy, True Positive Rate (TPR), recall, precision, f-measure and False Positive Rate (FPR). In the prototype platform, the experiment evaluated the features in terms of the accuracy and performance of each analyzer (i.e. Genetic, Bio, and Root) in a practical environment. Unlike the simulation platform, the prototype platform includes the practical steps not included in the Weka simulation. They encompass processes such as reverse engineering and identifying and

extracting the features from the application. From the identification of the prototype of the system's results, it appears that the results were able to achieve the accuracy range, starting from 95% and higher. In the performance results, the systems noted the best results by predicting the value of 0.627 MB application size in 18 seconds only.

7.2 Achievement of the study

This study had begun by investigating the different types of malware detection system in mobile devices. It explored the issues linked to the static analysis features and methods. It also attempted to extract the best features noted in the unknown malware detection including root exploit. This was accomplished by using a methodical approach. In the context of this study, a novel framework that would be used to address the static features and to facilitate the intelligent anomaly-based detection process in static analysis was proposed. Several features noted in the extraction and the machine learning classifiers were also explored. Their capabilities were evaluated so as to satisfy the aim of the study.

Within the proposed framework which consists of the experiments in addition to the prototypes of the intelligent prediction system, this study can thus be considered as successful. Several points of interest are also noted below:

a) **The development of the intelligent anomaly-based model for Malware Detection:** The studies thus far mentioned in this thesis have established three models which can be used to detect unknown malware including root exploit. Using the proposed approach to search for the best features, it seems that the proposed model was able to classify the classes of features in one application either as malware or benign. Chapter 4 describes this issue in detail. A survey study was conducted to explore the strategies in selecting the best static features for anomaly-based detection. To demonstrate the plausibility and reasonableness of the models,

several experiments were conducted and their outcomes derived positive results. Chapter 5 provides the results in detail.

b) **Issues in static analysis and feature studies:** In Chapter 3, the study had established a critical analysis which composed of different perspectives when addressing the significant problems of static analysis during the feature selection process. In relation to this, its challenges were also discussed. Due to the desire to create an anomaly-based static detection framework, various issues were explored for the purpose of selecting the best static features. In presenting the advantages and disadvantages of these issues, this study was able to identify the use of multiple strategies as noted by previous studies. The disadvantages noted in previous approaches were also used to improve the feature selection process so that it becomes more efficient for detecting the malware.

c) Exclusive features to detect unknown root exploit malware: This study had proposed a novel ADB type of features in the framework which is useful for detecting unknown root exploit. This proposed ADB framework was combined with other categories of features which consist of system command, directory path, and code-based.

d) **Adopting a bio-inspired Genetic Algorithm (GA) to select static features genetically:** The study proposed Genetic Search (GS) based on GA to search the best generation of features that malware frequently used in genetic way. This is to assists the anomaly-based mechanism in detecting unknown malware.

e) **A novel approach which addresses the best features by inspecting the range of the repeated features in a similar application:** This study had also proposed to develop a novel approach that is useful for addressing the best features by critically investigating the frequency of the same features that were continuously repeated in the same application.

f) **Investigating multiple categories of features:** As various categories of features exist and are ready to be explored at the same time, this study had investigated features noted in multiple categories including permission, system command, directory path, code-based and telephony. Such categories assist the anomaly-based detection to identify the unknown malware more accurately.

g) **Stages of comprehensive evaluation for the proposed framework:** In addressing the static analysis and the features selection process in the anomaly-based detection, the proposed framework had also outlined several models and strategies which require further evaluation. The purpose of this evaluation is to examine the proposed framework as well as to decide if it is adequate in facilitating the anomaly-based detection, using static analysis, to detect the unknown malware. This evaluation was performed in different phases. The proposed framework which encompasses models and strategies selected was evaluated for its effectiveness and performance. Chapter 5 provides the results in detail. The progressive results which demonstrate the suitability and feasibility of the proposed framework in enhancing the search for the best features in the anomaly-based detection were presented according to phases. More importantly, through the outstanding evaluation metric scores, the main criteria of the framework, as a support for malware detection, is fulfilled.

h) **Implementation of the proposed framework:** To further extend on the study, expand on the feasibility of the proposed framework and to demonstrate its practical anomaly-based detection, a proof-of-concept study was designed and realized in Chapter 6. As an extension to the evaluation study, the implementation stage has developed a web-based system that focuses on the intelligent prediction models of the proposed framework. To illustrate the implementation stage, details of the proposed framework, using several modeling languages were thus presented. They include case diagrams, state diagrams as well as snapshots of the prototype pages.

7.3 Limitation of the study

The discussions noted in previous chapters have validated that this study has adequately achieved its aims and objectives - the establishment of a novel framework that is useful for detecting unknown malware in anomaly-based detection environment. However, a number of limitations and challenges were encountered during the study and they are listed here for future references.

a) **Real-time malware activities:** As this study applied the static analysis method, it lacks the dynamic analysis real-time inspection. In particular, static analysis is unable to detect a benign application that has updated its form and evolved from benign to malware application. To relieve this drawback, the current study needs to capture and obtain the Android application package (.apk) of the current application for malware detection in two situations: 1) after the application updates its content; 2) at least once a month. Thus, one of the future works proposed in this study is to highlight this issue of normal application that evolves to malware form.

b) **Obfuscation:** Another limitation of using static analysis, as is noted in this study, is obfuscation (Tam *et al.*, 2017). It is an approach which complicates the decompiling process hence, it confuses the results. Since this study adopts static analysis, it is imperative that this issue be adequately addressed. At the time of writing this thesis, the tool to gain the exact native code was still unavailable. Thus, it is crucial for others to locate a reverse engineering tool that produces results which counter the obfuscated form. Once the application is obfuscated, it is hard to re factor the obfuscated code back to its original and perfect form. Henceforth, the experiment used in this study used Jadx as the reverse engineering tool which converts the .apk file to .java. It also provides the option of de-obfuscation, which is the best way of dealing with the obfuscated code in minimal error.

c) **Take account the processing time:** A comparison of the static analysis and dynamic @ real-time analysis detection had indicated that it carries a rapid processing speed. However, this study had excluded the activity of recording the time taken when the prediction had taken place in the mobile device itself. Hence, this activity of time taking needs to be included in future studies. This could provide the time processing speed of different specifications in mobile devices.

7.4 Suggestions and Scope for Future Work

A number of suggestions for future work outside the scope of this study have been identified as follows:

a) **Detecting real-time malware with static analysis:** As static analysis only inspects the application code, it is unable to detect applications that update their forms and applications which have evolved from benign to malware. Thus, it would be beneficial for future studies to investigate this kind of malware in two situations - before and after the update process. This is to test that static analysis is able to detect malware after the update process.

b) **Recording the time taken in mobile device:** Another recommendation for future studies is to conduct an experiment by running the prediction in the mobile device itself. This is to observe the time taken in the mobile device as noted in the browser during the time of the prediction process. This enable the study to identify the minimum specifications needed to run static analysis in a mobile device.

7.5 Summary – The future for mobile malware detection system

Security analysts had detected that the existence of Android malware is on the rise. They have also identified that relying on manual processes to predict and detect malware is time-consuming, difficult, sophisticated, complicated and error-prone. This study has presented a novel framework that predicts unknown malware by using

anomaly-based detection with static analysis features. The framework provide security analysts with the assistance of uncovering malware noted in previous studies. In fact, the entire study has demonstrated the advantages of using the proposed framework in facilitating malware detection. The study not only focused on single category of features (i.e. permission) but also other categories (i.e. directory path, system command, telephony, code-based) as well. The important concept lying behind the proposed framework is the methodical steps which include the search for all features in all files in each application (not depending on one single file only) and catering to the exclusive list of features in minimal amount for intelligent prediction in malware detection by using static features in static analysis.

With the bio-inspired methods and the repeated features investigation such as GS and range algorithm which selects the best features in detecting malware, the adoption of the proposed framework has given new perspectives into future research. In fact, the future of static analysis with the anomaly-based detection technique is a step forward from the malware detection proposed by this study. All the outstanding results contribute significantly to the discovery of a new malware and this depends on the best features that were proposed in the study. Moreover, the machine learning classifier models noted in the prototype can assist users in discovering malware by merely uploading their desired Android application package file (.apk). Furthermore, this study also provides the best features that can be used to detect unknown root exploit. Hence, the system is able to predict malware including root exploit that have not been found before.

In reality, the future of detecting malware through the intrusion detection system still depends on human intervention and this is normal with any security analysis. Nonetheless, with the help of modern technology and the anomaly-based detection technique, such interventions are gradually declining. This study has contributed to that

domain in some significant degree. Although human intervention is still needed to check on each application with in-depth, the best features that combine with the anomaly-based detection mechanism are important in double-confirming the class of the application as benign or malware. With more studies working to investigate the best features for the anomaly-based detection technique, it is hoped that one day the updated features could improve the static analysis investigation.

REFERENCES

1mobile. (2017). 1mobile. Retrieved January 12, 2017, from http://market.1mobile.com/

Aafer, Y., Du, W., & Yin, H. (2013). DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. *Security and Privacy in Communication Networks* (pp. 86–103).

Adrian. (2012). For Android apps, "zero permissions" does not actually mean zero permissions. Retrieved February 26, 2017, from http://www.androidauthority.com/android-apps-zero-permissions-75001/

Afifi, F., Anuar, N. B., Shamshirband, S., & Choo, K.-K. R. (2016). DyHAP: Dynamic Hybrid ANFIS-PSO Approach for Predicting Mobile Malware. *Plos One*, *11*(9), 1–21. Retrieved from http://dx.plos.org/10.1371/journal.pone.0162627

Allix, K., Bissyandé, T. F., Klein, J., & Traon, Y. Le. (2016). AndroZoo: Collecting Millions of Android Apps for the Research Community. *MSR '16 Proceedings of the 13th International Conference on Mining Software Repositories, Austin, Texas* (pp. 468–471).

Alzahrani, A. J., Stakhanova, N., Gonzalez, H., & A.Ghorbani, A. (2014). Characterizing Evaluation Practices of Intrusion Detection Methods for Smartphones. *River Journal*, 1–36.

Anderson, B. (2016). Understanding the Android File Hierarchy. Retrieved July 8, 2016, from http://www.all-things-android.com/content/understanding-android-file-hierarchy

Android. (2015). App Manifest. Retrieved April 28, 2015, from http://developer.android.com/guide/topics/manifest/manifest-intro.html

Android. (2016). Telephony Manager. Retrieved July 1, 2016, from https://developer.android.com/reference/android/telephony/TelephonyManager.html

Android Developer. (2017). Android Debug Bridge (ADB). Retrieved January 1, 2017, from http://developer.android.com/tools/help/adb.html

Android Developers. (2015). Android Security Overview. *Android*. Retrieved September 1, 2015, from https://source.android.com/devices/tech/security/

Angeeks. (2017). Angeeks. Retrieved January 12, 2017, from http://bbs.angeeks.com/portal.php

Anuar, N. B., Papadaki, M., Furnell, S., & Clarke, N. (2013). Incident prioritisation using analytic hierarchy process (AHP): Risk Index Model (RIM). *Security and Communication Networks*, *6*(9), 1087–1116.

Anuar, N. B., Sallehudin, H., Gani, A., & Zakari, O. (2008). Identifying false alarm for network intrusion detection system using hybrid data mining and decision tree.

Malaysian Journal of Computer Science, *21*(2), 101–115.

Anwar, S., Mohamad Zain, J., Zolkipli, M. F., Inayat, Z., Khan, S., Anthony, B., & Chang, V. (2017). From Intrusion Detection to an Intrusion Response System: Fundamentals, Requirements, and Future Directions. *Algorithms*, *10*(2), 39. Retrieved from http://www.mdpi.com/1999-4893/10/2/39

Anzhi. (2017). Anzhi. Retrieved January 12, 2017, from http://www.anzhi.com/

Appavu, S., Rajaram, R., Nagammai, M., Priyanga, N., & Priyanka, S. (2011). Bayes Theorem and Information Gain Based Feature Selection for Maximizing the Performance of Classifiers. *International Conference on Computer Science and Information Technology (CCSIT), Bangalore, India* (pp. 501–511).

AppChina. (2017). AppChina. Retrieved January 12, 2017, from http://www.appchina.com/

Apvrille, A., & Strazzere, T. (2012). Reducing the window of opportunity for Android malware Gotta catch 'em all. *Journal in Computer Virology*, *8*(1), 61–71. Retrieved February 12, 2014, from http://link.springer.com/10.1007/s11416-012-0162-3

Arghire, I. (2016). Android Root Exploits Abuse Dirty COW Vulnerability. Retrieved November 4, 2016, from http://www.securityweek.com/android-root-exploits-abuse-dirty-cow-vulnerability

Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., & Rieck, K. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. *21th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA* (pp. 1–15).

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Traon, Y. Le, *et al.* (2014). FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, Edinburgh, United Kingdom* (pp. 259–269).

Aung, Z., & Zaw, W. (2013). Permission-Based Android Malware Detection. *International Journal of Scientific & Technology Research*, *2*(3), 228–234.

Azhagusundari, B., & Thanamani, A. S. (2013). Feature Selection based on Information Gain. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, *2*(2), 18–21.

Bartel, A., Klein, J., Le Traon, Y., & Monperrus, M. (2012). Automatically securing permission-based software by reducing the attack surface: an application to Android. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), Essen, Germany* (pp. 274–277).

BBC, N. (2016). Malware hits millions of Android phones. Retrieved September 4, 2016, from http://www.bbc.com/news/technology-36744925

Bickford, J., O'Hare, R., Baliga, A., Ganapathy, V., & Iftode Liviu. (2010). Rootkits on smart phones: attacks, implications and opportunities. *HotMobile '10 Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications, Annapolis, Maryland* (pp. 49–54).

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc. Retrieved from http://www.nltk.org/book_1ed/

Blum, A. L., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, *97*(97), 245–271.

Caruana, R., Karampatziakis, N., & Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. *Proceedings of the 25th international conference on Machine learning, Helsinki, Finland* (pp. 96–103).

Chambers, J. (2017). The Comprehensive R Archive Network. Retrieved January 1, 2017, from https://cran.r-project.org/

Chan, P. P. K., & Song, W. K. (2015). Static detection of Android malware by using permissions and API calls. *International Conference on Machine Learning and Cybernetics, Lanzhou, China* (Vol. 1, pp. 82–87).

Chang, T.-K., & Hwang, G.-H. (2007). The design and implementation of an application program interface for securing XML documents. *Journal of Systems and Software*, *80*(8), 1362–1374.

Chess, B., & McGraw, G. (2004). Static analysis for security. *IEEE Security & Privacy Magazine*, *2*(6), 76–79. Retrieved from www.ieeexplore.ieee.org/iel5/8013/29915/01366126.pdf

Cimpanu, C. (2016). A new Android malware family called DressCode can be used as a proxy to relay attacks inside corporate networks and steal information from servers previously considered secure. Retrieved September 1, 2016, from http://news.softpedia.com/news/dresscode-android-malware-discovered-on-official-google-play-store-507829.shtml

Crussell, J., Gibler, C., & Chen, H. (2012). Attack of the clones: Detecting cloned applications on Android markets. *Computer Security – ESORICS 2012. Lecture Notes in Computer Science* (Vol. 7459, pp. 37–54).

Deshotels, L., Notani, V., & Lakhotia, A. (2014). DroidLegacy: Automated Familial Classification of Android Malware. *Proceedings of ACM SIGPLAN on Program Protection and Reverse Engineering Workshop, San Diego, CA, USA* (pp. 1–12).

Desnos, A. (2015). Androguard. Retrieved June 29, 2015, from https://github.com/androguard/androguard

Díaz-Uriarte, R., & Alvarez de Andrés, S. (2006). Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, *7*(3), 1–13.

Duch, W., Biesiada, J., Winiarski, T., Grudziński, K., & Grąbczewski, K. (2003). Feature Selection Based on Information Theory Filters. *Neural Networks and Soft*

Computing (pp. 173–178). Physica, Heidelberg.

eBay. (2016). Online Shopping. Retrieved April 4, 2016, from www.ebay.com

F-Droid. (2017). F-Droid. Retrieved January 12, 2017, from http://f-droid.org

Faruki, P., Ganmoor, V., Laxmi, V., Gaur, M. S., & Bharmal, A. (2013). AndroSimilar: Robust Statistical Feature Signature for Android Malware Detection. *Proceedings of the 6th International Conference on Security of Information and Networks, Aksaray, Turkey* (pp. 152–159).

Feizollah, A., Anuar, N. B., Salleh, R., Amalina, F., Ma'arof, R. R., & Shamshirband, S. (2013). A Study Of Machine Learning Classifiers For Anomaly-Based Mobile Botnet Detection. *Malaysian Journal of Computer Science*, *26*(4), 251–265.

Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., & Furnell, S. (2017). AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection. *Computers & Security*, *65*, 121–134. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/S0167404816301602

Feizollah, A., Anuar, N. B., Salleh, R., & Wahab, A. W. A. (2015). A review on feature selection in mobile malware detection. *Digital Investigation*, *13*, 22–37. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/S1742287615000195

Feizollah, A., Shamshirband, S., Anuar, N. B., Salleh, R., & Kiah, M. L. M. (2013). Anomaly Detection Using Cooperative Fuzzy Logic Controller. *16th FIRA RoboWorld Congress (FIRA), Kuala Lumpur, Malaysia* (pp. 220–231).

Felt, A. P., Finifter, M., Chin, E., Hanna, S., & Wagner, D. (2011). A survey of mobile malware in the wild. *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM), Illinois, USA* (pp. 3–14).

Feng, Y., Anand, S., Dillig, I., & Aiken, A. (2014). Apposcopy: Semantics-Based Detection of Android Malware Through Static Analysis. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, China* (pp. 576–587).

Firdaus, A., & Anuar, N. B. (2015). Root-exploit Malware Detection using Static Analysis and Machine Learning. *Proceedings of the Fourth International Conference on Computer Science & Computational Mathematics (ICCSCM 2015), Langkawi, Malaysia* (pp. 177–183).

Foreman, G. (2003). An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research*, *3*, 1289–1305.

Forni, A. A., & Van der Meulen, R. (2016). Gartner Says Five of Top 10 Worldwide Mobile Phone Vendors Increased Sales in Second Quarter of 2016. Retrieved January 1, 2017, from http://www.gartner.com/newsroom/id/3415117

Frank, E., Hall, M. A., & Witten, I. H. (2016). *The WEKA Workbench*. *Morgan Kaufmann, Fourth Edition*. Retrieved from http://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf

Freewarelovers. (2017). Freewarelovers. Retrieved January 12, 2017, from http://www.freewarelovers.com

Freund, Y., & Schapire, R. E. (1999). Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, *37*(3), 277–296.

Fröhlich, H., Chapelle, O., & Schölkopf, B. (2003). Feature Selection for Support Vector Machines by Means of Genetic Algorithms. *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03), Sacramento, California, USA* (pp. 142–148).

Gascon, H., Yamaguchi, F., Arp, D., & Rieck, K. (2013). Structural Detection of Android Malware using Embedded Call Graphs. *Proceedings of the 2013 ACM workshop on Artificial Intelligence and Security, Berlin, Germany* (pp. 45–54).

Ghallali, M., & El Ouahidi, B. (2012). Security of Mobile Phones: Prevention Methods for The Spread of Malware. *6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), Sousse, Tunisia* (pp. 648–651).

Goldberg, D. E., & Holland, J. H. (1988). Genetic Algorithms and Machine Learning. *Machine Learning*, *3*(2–3), 95–99.

Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., *et al*. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science (New York, N.Y.)*, *286*(5439), 531–537.

Google. (2014). Google Play Store. Retrieved January 1, 2014, from https://play.google.com/store?hl=en

Gordon, M. I., Kim, D., Perkins, J., Gilham, L., Nguyen, N., & Rinard, M. (2015). Information-Flow Analysis of Android Applications in DroidSafe. *Network and Distributed System Security Symposium (NDSS), San Diego, CA* (pp. 8–11).

Grace, M. C., Zhou, W., Jiang, X., & Sadeghi, A.-R. (2012). Unsafe Exposure Analysis of Mobile In-App Advertisements. *Proceeding 5th ACM conference on Security and Privacy in Wireless and Mobile Networks, Tucson, Arizona, USA* (Vol. 67, pp. 101–112).

Grace, M., Zhou, Y., Wang, Z., & Jiang, X. (2012). Systematic Detection of Capability Leaks in Stock Android Smartphones. *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS), San Diego, CA* (pp. 1–15).

Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2011). RiskRanker: Scalable and Accurate Zero-day Android Malware Detection. *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, Low Wood Bay, Lake District, UK* (pp. 281–293).

Guyon, I., & Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, *3*(3), 1157–1182.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations*, *11*(1), 10–18.

Han, J., Kamber, M., & Pei, J. (2001). *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems.

HiApk. (2017). HiApk. Retrieved January 12, 2017, from http://www.hiapk.com

Hou, O. (2016). A Look at Google Bouncer. Retrieved September 9, 2016, from http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/

Huang, C.-Y., Tsai, Y.-T., & Hsu, C.-H. (2012). Performance Evaluation on Permission-Based Detection for Android Malware. *Proceedings of the International Computer Symposium ICS 2012 Held at Hualien, Taiwan* (Vol. 21, pp. 111–120).

Huang, J., Zhang, X., Tan, L., Wang, P., & Liang, B. (2014). AsDroid: Detecting Stealthy Behaviors in Android Applications by User Interface and Program Behavior Contradiction. *Proceeding ICSE 2014 Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India* (pp. 1036–1046).

Imam, I. F., Michalski, R. S., & Kerschberg, L. (1993). Discovering Attribute Dependence in Databases by Integrating Symbolic Learning and Statistical Analysis Techniques. *Proceedings of the 1st International Workshop on Knowledge Discovery in Databases, Washington, DC* (pp. 1–13).

Inayat, Z., Gani, A., Anuar, N. B., Khan, M. K., & Anwar, S. (2016). Intrusion response systems: Foundations, design, and challenges. *Journal of Network and Computer Applications*, *62*, 53–74. Elsevier. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/S1084804515002994

Jensen, R., & Shen, Q. (2008). *Computational Intelligence and Feature Selection: Rough and Fuzzy Approaches*. Wiley-IEEE Press.

Junaid, M., Liu, D., & Kung, D. (2016). Dexteroid: Detecting Malicious Behaviors in Android Apps Using Reverse-Engineered Life Cycle Models. *Computers and Security*, *59*, 92–117.

Kang, H., Jang, J., Mohaisen, A., & Kim, H. K. (2015). Detecting and Classifying Android Malware using Static Analysis along with Creator Information. *International Journal of Distributed Sensor Networks - Special issue on Advanced Big Data Management and Analytics for Ubiquitous Sensors*, *11*(6), 1–17.

Karim, A., Salleh, R., & Khan, M. K. (2016). SMARTbot: A Behavioral Analysis Framework Augmented with Machine Learning to Identify Mobile Botnet Applications. *Plos One*, *11*(3), 1–35. Retrieved from http://dx.plos.org/10.1371/journal.pone.0150077

Karim, A., Salleh, R., Khan, M. K., Siddiqa, A., & Choo, K.-K. R. (2016). On the

Analysis and Detection of Mobile Botnet. *Journal of Universal Computer Science*, *22*(4), 567–588.

Kasperksy. (2016). IT threat evolution Q3 2016 Statistics. Retrieved November 15, 2016, from https://securelist.com/analysis/quarterly-malware-reports/76513/it-threat-evolution-q3-2016-statistics/

Kent, J. T. (1982). Information gain and a general measure of correlation. *Biometrika*, 163–173.

Koller, D., & Sahami, M. (1996). Toward optimal feature selection. *Proceedings of the Thirteenth International Conference on Machine Learning (ICML), San Francisco, CA* (pp. 284–292).

Komili, O. (2016). Sophos detects 100% of Android malware in independent test – for the sixth time in a row. Retrieved January 1, 2016, from https://blogs.sophos.com/2015/08/14/sophos-detects-100-of-android-malware-in-independent-test-for-the-sixth-time-in-a-row/

Kotsiantis, S. B. (2013). Decision trees: A recent overview. *Artificial Intelligence Review*, *39*(4), 261–283.

Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2006). Machine learning: A review of classification and combining techniques. *Artificial Intelligence Review*, *26*(2006), 159–190.

Lai, H., Tang, Y., Luo, H., & Pan, Y. (2011). Greedy feature selection for ranking. *International Conference on Computer Supported Cooperative Work in Design (CSCWD), Lausanne, Switzerland* (pp. 42–46).

Lee, J., Lee, S., & Heejo, L. (2015). Screening Smartphone Applications Using Malware Family Signatures. *Computers & Security*, *52*, 234–249. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-39218-4_2

Lee, S.-H., & Jin, S.-H. (2013). Warning System for Detecting Malicious Applications on Android System. *International Journal of Computer and Communication Engineering*, *2*(3), 324–327.

Li, Q., & Clark, G. (2013). Mobile security: a look ahead. *Security & Privacy*, (February), 78–81.

Liang, S., Keep, A. W., Might, M., Lyde, S., Gilray, T., Aldous, P., & Horn, D. Van. (2013). Sound and Precise Malware Analysis for Android via Pushdown Reachability and Entry-Point Saturation. *ACM workshop on Security and privacy in smartphones and mobile devices, Berlin, Germany* (pp. 21–32).

Lippmann, R. P. (1987). An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, *4*(2), 4–22.

Lu, L., Li, Z., Wu, Z., Lee, W., & Jiang, G. (2012). CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities. *CCS Proceedings of the ACM Conference on Computer and Communications Security, Raleigh, North Carolina,*

USA (pp. 229–240).

Lu, Y., Zulie, P., Jingju, L., & Yi, S. (2013). Android malware detection technology based on improved Bayesian Classification. *Instrumentation, Measurement, Computer, Communication and Control (IMCCC), Shenyang, China* (pp. 1338–1341).

Luoshi, Z., Yan, N., Xiao, W., Wang, Z., & Xue, Y. (2013). A3: Automatic Analysis of Android Malware. *International Workshop on Cloud Computing and Information Security (CCIS), Shanghai, China* (pp. 89–93).

Ma, Y., & Sharbaf, M. S. (2013). Investigation of Static and Dynamic Android Anti-virus Strategies. *10th International Conference on Information Technology: New Generations (ITNG), Las Vegas, Nevada* (pp. 398–403).

McAfee. (2016). McAfee Labs Threats Report. Retrieved December 1, 2016, from http://www.mcafee.com/us/resources/reports/rp-quarterly-threats-sep-2016.pdf

Middlemiss, M. J., & Dick, G. (2003). Weighted feature extraction using a genetic algorithm for intrusion detection. *The 2003 Congress on Evolutionary Computation (CEC), Canberra, Australia* (Vol. 3, pp. 1–7).

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, Taipei, Taiwan* (pp. 156–161).

Narudin, F. A., Feizollah, A., Anuar, N. B., & Gani, A. (2014). Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, *20*(1), 343–357. Retrieved November 18, 2014, from http://link.springer.com/10.1007/s00500-014-1511-6

Nissim, N., Moskovitch, R., Rokach, L., & Elovici, Y. (2014). Novel active learning methods for enhanced PC malware detection in windows OS. *Expert Systems with Applications*, *41*(13), 5843–5857. Retrieved from http://dx.doi.org/10.1016/j.eswa.2014.02.053

Patel, A., Taghavi, M., Bakhtiyari, K., & Ju, J. C. (2013). An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of Network and Computer Applications*, *36*, 25–41.

Peiravian, N., & Zhu, X. (2013). Machine Learning for Android Malware Detection Using Permission and API Calls. *International Conference on Tools with Artificial Intelligence (ICTAI), Herndon, VA, USA* (pp. 300–305).

Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., *et al.* (2012). Using Probabilistic Generative Models for Ranking Risks of Android Apps. *ACM Conference on Computer and Communications Security, (CCS), Raleigh, North Carolina, USA* (pp. 241–252). Retrieved from http://doi.acm.org/10.1145/2382196.2382224

Powell, W. B. (2011). *Approximate Dynamic Programming: Solving the Curses of*

Dimensionality. Wiley Series in Probability and Statistics.

Priyadarsini, R. P., Valarmanthi, M. L., & Sivakumari, S. (2011). Gain Ratio Based Feature Selection Method for Privacy Preservation. *ICTACT Journal on Soft Computing*, *1*(4), 201–205.

ProAndroid. (2017). ProAndroid. Retrieved January 12, 2017, from http://proandroid.net

Punch, W. F., Goodman, E. D., Pei, M., Chia-Shun, L., Hovland, P., & Enbody, R. (1993). Further Research on Feature Selection and Classification Using Genetic Algorithms. *Proceedings of the 5th International Conference on Genetic Algorithms, San Francisco, CA, USA* (pp. 557–564).

Rasthofer, S., Arzt, S., & Bodden, E. (2014). A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks. *Symposium on Network and Distributed System Security (NDSS), San Diego, CA, USA* (pp. 1–15).

Razak, M. F. A., Anuar, N. B., Salleh, R., & Firdaus, A. (2016). The rise of malware'': Bibliometric analysis of malware study. *Journal of Network and Computer Applications*, *75*, 58–76. Retrieved from http://dx.doi.org/10.1016/j.jnca.2016.08.022

Roobaert, D., Karakoulas, G., & Chawla, N. V. (2006). Information Gain, Correlation and Support Vector Machines. *Feature Extraction: Foundations and Applications* (Vol. 207, pp. 463–470).

Russon, M.-A. (2016). Android malware discovered on Google Play has infected millions of users with spyware. Retrieved June 13, 2016, from http://www.ibtimes.co.uk/android-malware-discovered-google-play-store-1553341

Sabry, A., Orman, Z., Mohsin, A., & Brifcani, A. (2015). A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems. *Expert Systems With Applications*, *42*(5), 2670–2679. Retrieved from http://dx.doi.org/10.1016/j.eswa.2014.11.009

Sahs, J., & Khan, L. (2012). A Machine Learning Approach to Android Malware Detection. *European Intelligence and Security Informatics Conference, (EISIC), University of Southern Denmark Odense, Denmark* (pp. 141–147).

Samra, A. A. A., Kangbin, Y., & Ghanem, O. A. (2013). Analysis of Clustering Technique in Android Malware Detection. *Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), Taichung, Taiwan* (pp. 729–733).

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G., & Alvarez, G. (2013). PUMA: Permission Usage to detect Malware in Android. *Advances in Intelligent Systems and Computing* (pp. 289–298).

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Nieves, J., Bringas, P. G., & Álvarez Marañón, G. (2013). Mama: Manifest Analysis for Malware Detection in Android. *Cybernetics and Systems*, *44*(6–7), 469–488.

Sarip, A. G., Hafez, M. B., & Daud, M. N. (2016). Application Of Fuzzy Regression Model For Real Estate Price Prediction. *Malaysian Journal of Computer Science*, *29*(1), 15–27.

Sarma, B., Li, N., Gates, C., Potharaju, R., Nita-rotaru, C., & Molloy, I. (2012). Android Permissions: A Perspective Combining Risks and Benefits. *SACMAT '12 Proceedings of the 17th ACM symposium on Access Control Models and Technologies, New Jersey, USA* (pp. 13–22).

Schmidt, A.-D., Bye, R., Schmidt, H.-G., Clausen, J., Kiraz, O., Yuksel, K. A., Camtepe, S. A., *et al.* (2009). Static Analysis of Executables for Collaborative Malware Detection on Android. *IEEE International Conference on Communications (ICC), Dresden, Germany* (pp. 1–5).

Schmidt, A., Schmidt, H., Batyuk, L., Clausen, J. H., Camtepe, S. A., Albayrak, S., & Yildizli, C. (2009). Smartphone Malware Evolution Revisited: Android Next Target? *IEEE Conference Publications, Montreal, Quebec, Canada* (pp. 1–7).

Schneider, J. (2016). Cross Validation. Retrieved August 1, 2016, from http://www.cs.cmu.edu/~schneide/tut5/node42.html

Seo, S.-H., Gupta, A., Mohamed Sallam, A., Bertino, E., & Yim, K. (2014). Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications*, *38*, 43–53. Retrieved May 23, 2014, from http://linkinghub.elsevier.com/retrieve/pii/S1084804513001227

Shabtai, A., Fledel, Y., & Elovici, Y. (2010). Automated Static Code Analysis for Classifying Android Applications Using Machine Learning. *Ninth International Conference on Computational Intelligence and Security, Nanning, Guangxi Zhuang Autonomous Region China* (pp. 329–333).

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(July 1928), 379–423. Retrieved from http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

Sharif, M., Yegneswaran, V., Saidi, H., Porras, P., & Lee, W. (2008). Eureka: A framework for enabling static malware analysis. *Lecture Notes in Computer Science* (Vol. 5283, pp. 481–500).

Sheen, S., Anitha, R., & Natarajan, V. (2015). Android based malware detection using a multifeature collaborative decision fusion approach. *Neurocomputing*, *151*, 905–912. Retrieved from http://dx.doi.org/10.1016/j.neucom.2014.10.004

Skylot. (2015). Jadx. Retrieved February 1, 2014, from https://github.com/skylot/jadx

Slideme. (2017). Slideme. Retrieved January 12, 2017, from http://slideme.org/

Spolaôr, N., Cherman, E. A., Monard, M. C., & Lee, H. D. (2013). A comparison of multi-label feature selection methods using the problem transformation approach. *Electronic Notes in Theoretical Computer Science*, *292*, 135–151.

Stein, G., Chen, B., Wu, A. S., & Hua, K. A. (2005). Decision Tree Classifier For

Network Intrusion Detection With GA-based Feature Selection. *ACM-SE 43 Proceedings of the 43rd annual Southeast regional conference, Kennesaw, Georgia* (Vol. 2, pp. 136–141).

Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P., & Blasco, J. (2014). Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families. *Expert Systems with Applications*, *41*(4), 1104–1117. Retrieved from http://dx.doi.org/10.1016/j.eswa.2013.07.106

Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P., & Ribagorda, A. (2014). Evolution, detection and analysis of malware for smart devices. *IEEE Communications Surveys and Tutorials*, *16*(2), 961–987.

Symantec. (2015). Symantec Intelligence Report. Retrieved December 1, 2015, from http://www.symantec.com/content/en/us/enterprise/other_resources/b-intelligence-report-01-2015-en-us.pdf

Talha, K. A., Alper, D. I., & Aydin, C. (2015). APK Auditor: Permission-based Android malware detection system. *Digital Investigation*, *13*, 1–14. Retrieved from http://www.sciencedirect.com/science/article/pii/S174228761500002X

Tam, K., Feizollah, A. L. I., Anuar, N. O. R. B., Salleh, R., & Cavallaro, L. (2017). The Evolution of Android Malware and Android Analysis Techniques. *ACM Computing Surveys (CSUR)*, *49*(4), 1–41.

Thomas, P. (2015). Google's Android Operating System Dominates the Smartphone Market. Retrieved June 11, 2016, from http://finance.yahoo.com/news/google-android-operating-system-dominates-170640913.html

Tropp, J. A. (2004). Greed is Good: Algorithmic Results for Sparse Approximation. *IEEE Transactions on Information Theory*, *50*(10), 2231–2242.

Ueltschi, D. (2006). Shannon entropy. *Chapter 6*. Retrieved from http://www.ueltschi.org/teaching.html

Union, I. T. (2016). ICT facts and figures. Retrieved from http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2016.pdf

VirusTotal. (2016). VirusTotal. Retrieved August 24, 2016, from https://www.virustotal.com/

Waikato, U. (2017). Weka 3: Data Mining Software in Java. Retrieved January 1, 2017, from http://www.cs.waikato.ac.nz/ml/weka/

Walczak, B., & Massart, D. L. (2000). Local modelling with radial basis function networks. *Chemometrics and Intelligent Laboratory Systems*, *50*(2), 179–198. Retrieved from http://dx.doi.org/10.1016/S0169-7439(99)00056-8

Walenstein, A., Deshotels, L., & Lakhotia, A. (2012). Program Structure-Based Feature Selection for Android Malware Analysis. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering* (Vol. 107, pp. 51–52).

Wei, T., Lee, H., Tyan, H.-R., Liao, H. M., Jeng, A. B., & Wang, J. (2015). DroidExec: Root Exploit Malware Recognition Against Wide Variability via Folding Redundant. *17th International Conference Advanced Communication Technology (ICACT), PyeongChang, Korea* (pp. 161–169).

Williams, G. (2010). ARFF Data. Retrieved September 10, 2015, from http://datamining.togaware.com/survivor/ARFF_Data0.html

Wiśniewski, R. (2015). Apktool. Retrieved June 29, 2015, from https://ibotpeaches.github.io/Apktool/

Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., & Wu, K.-P. (2012). DroidMat: Android Malware Detection through Manifest and API Calls Tracing. *Seventh Asia Joint Conference on Information Security, Tokyo, Japan* (pp. 62–69).

Yang, Z., & Yang, M. (2012). Leak miner: Detect information leakage on android with static taint analysis. *Third World Congress on Software Engineering (WCSE), Wuhan, China* (pp. 101–104).

Yerima, S. Y., Sezer, S., & McWilliams, G. (2014). Analysis of Bayesian classification-based approaches for Android malware detection. *IET Information Security*, *8*(1), 25–36. Retrieved January 23, 2014, from http://digital-library.theiet.org/content/journals/10.1049/iet-ifs.2013.0095

Yerima, S. Y., Sezer, S., McWilliams, G., & Muttik, I. (2013). A New Android Malware Detection Approach Using Bayesian Classification. *IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain* (pp. 121–128).

Yerima, S. Y., Sezer, S., & Muttik, I. (2014). Android Malware Detection Using Parallel Machine Learning Classifiers. *Eight International Conference on Next Generation Mobile Apps, Services and Technologies, (NGMAST), St. Anthony's College of the University of Oxford, UK* (pp. 37–42).

Yerima, S. Y., Sezer, S., & Muttik, I. (2015). High Accuracy Android Malware Detection Using Ensemble Learning. *IET Information Security*, *9*(6), 313–320. Retrieved from http://digital-library.theiet.org/content/journals/10.1049/iet-ifs.2014.0099

Yu, L., & Liu, H. (2004). Efficient Feature Selection via Analysis of Relevance and Redundancy. *Journal of Machine Learning Research*, *5*, 1205–1224. Retrieved from http://portal.acm.org/citation.cfm?id=1044700

Zhang, T. (2009). On the Consistency of Feature Selection using Greedy Least Squares Regression. *Journal of Machine Learning Research*, *10*, 555–568.

Zhang, V. (2016). "GODLESS" Mobile Malware Uses Multiple Exploits to Root Devices. Retrieved November 4, 2016, from http://blog.trendmicro.com/trendlabs-security-intelligence/godless-mobile-malware-uses-multiple-exploits-root-devices/

Zheng, M., Sun, M., & Lui, J. C. S. (2013). Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware. *12th IEEE*

International Conference on Trust, Security and Privacy in Computing and Communications, (TrustCom), Melbourne, VIC, Australia (pp. 163–171).

Zhou, W., Zhou, Y., Grace, M., Jiang, X., & Zou, S. (2013). Fast, scalable detection of "Piggybacked" mobile applications. *CODASPY '13 Proceedings of the second ACM conference on Data and Application Security and Privacy, San Antonio, Texas, USA* (pp. 185–195).

Zhou, W., Zhou, Y., Jiang, X., & Ning, P. (2012). Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces. *CODASPY '12 Proceedings of the second ACM conference on Data and Application Security and Privacy, San Antonio, Texas, USA* (pp. 317–326).

Zhou, X., Demetriou, S., He, D., Naveed, M., Pan, X., Wang, X., Gunter, C. a., *et al.* (2013). Identity, location, disease and more: inferring your secrets from android public resources. *Proceedings of the ACM SIGSAC conference on computer & communications security, Berlin, Germany* (pp. 1017–1028).

Zhou, Y., & Jiang, X. (2012a). Dissecting Android Malware: Characterization and Evolution. *IEEE Symposium on Security and Privacy, San Francisco, CA* (pp. 95–109). Retrieved December 12, 2013, from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6234407

Zhou, Y., & Jiang, X. (2012b). Android Malware Genome Project. Retrieved from http://www.malgenomeproject.org/

Zia, T., Akhter, M. P., & Abbas, Q. (2015). Comparative Study of Feature Selection Aapproaches for Urdu Text Categorization. *Malaysian Journal of Computer Science*, *28*(2), 93–109.

LIST OF PUBLICATIONS AND PAPERS PRESENTED

1. A. Firdaus, N. B. Anuar, M. F. A. Razak, and A. K. Sangaiah, "Bio-inspired computational paradigm for feature investigation and malware detection: interactive analytics," Multimedia Tools and Applications, pp. 1–37, 2017.

2. A. Firdaus, N. B. Anuar, A. Karim, and M. F. A. Razak, "Discovering Optimal Features using Static Analysis and Genetic Search Based Method for Android Malware Detection", Journal of Frontiers of Information Technology & Electronic Engineering, Accepted 15 March 2017.

3. M. F. A. Razak, N. B. Anuar, R. Salleh, and A. Firdaus, "The rise of malware'': Bibliometric analysis of malware study," Journal of Network and Computer Applications, vol. 75, pp. 58–76, 2016.

4. A. Firdaus and N. B. Anuar, "Root-exploit Malware Detection using Static Analysis and Machine Learning," in Proceedings of the Fourth International Conference on Computer Science & Computational Mathematics (ICCSCM 2015), Langkawi, Malaysia, 2015, pp. 177–183.