Analysis of Malicious Applications on Smart Phones Running the Android Operating System

By

Waleed Bin Shahid

Submitted to the Faculty of Information Security Department,

Military College of Signals, National University of Sciences and Technology,

Islamabad in partial fulfillment for the requirements of an M.S. Degree in

Information Security

FEBRUARY 2013

ABSTRACT

Android has now become the most widely used operating system for smartphones in the world. This rapid increase in the use of Android both for smartphones and tablets along with its open source nature has motivated malware authors to write highly sophisticated malware for Android operating system. A lot of work has been carried out by security researchers to counter the effect caused by growing amount of malware.

In this research, a new algorithm for static analysis of Android applications has been proposed that checks an application for maliciousness on the basis of application features and not signatures, as the latter is inefficient in detecting zero day malware. Various features of an Android application, e.g. sending SMS, accessing internet, uploading files, accessing Wi-Fi, have been found out which when occur in different combinations along with other features aid in constituting malicious rules. Hence a rule is combination of multiple features which if exist in an application, tend to show malicious behavior. A set of 958 malicious and 816 benign applications were analyzed against all rules and only those rules were selected for the algorithm whose probability of occurrence was significantly higher in malicious applications than in benign. The Least significant difference between the probabilities of occurrence in malicious and benign applications was computed and Gaussian distribution with 5% level of significance was used to accept rules whose arithmetic difference was greater than the least significance distance. The algorithm has also been supported with a proof of concept application written in C language supported with a Graphical User interface written in Microsoft Foundation Class Library.

The proposed algorithm has been tested on another 247 malicious and 768 benign applications and yields the accuracy of 98.32% and specificity of 99.6% with low false positive ratio. The algorithm shows better results than other related algorithms for countering Android malware. The computational complexity of the algorithm is exceptionally low, thus making it suitable for analyzing applications.

ACKNOWLEDGEMENTS

First of all, I pay my deepest gratitude to Almighty Allah whose blessings and exaltations flourished my thoughts and enabled me to be amongst those who make contributions, special praise to Prophet Muhammad (Peace be upon Him) who is and always be the shiniest torch of knowledge and guidance for mankind.

I owe a debt of gratitude to my ever helpful and dedicated supervisor Dr. Fauzan Mirza, Assistant Professor, NUST School of Electrical Engineering and Computer Science, Islamabad, who put a lot of effort through valuable guidance, sympathetic attitude, keen interest and constructive criticism for the accomplishment of the present work.

I am also highly grateful to Ms. Ayesha Naureen, Mr. Wasim Iqbal and Ms. Rabia Khan, who being the able members of my Guidance and Examination Committee (GEC) guided me throughout my research and provided me support for accomplishing this task. I am also very highly thankful to Mr. Moin-ud-Din, Assistant Professor, NUST SEECS, for his valuable guidance for understanding the statistical aspects associated with my research. I would also like to thank Lt Col Dr. Asif Masood, Head of Information Security Department, Military College of Signals, for providing an encouraging environment for academics and research in the college.

I pay my deepest love and thanks to my mother, brother, sister, their families, and my best friend, whose heartiest cooperation, kindness, prayers and noble guidance were a great source of strength to me. It is because of them that I am able to achieve this milestone in my life.

At the end, I owe my gratitude to my friends, colleagues, and classmates, lab attendants and library staff who afforded me patiently and yet remained highly supportive throughout the research period.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

C h a p t e r 1

INTRODUCTION

1.1 Chapter Overview

This chapter explains the concept of malware analysis of Android applications, its basics, types and components. It also briefly discusses different tools used in analyzing Android malware. The chapter will also discuss the Architecture of Android operating system and its security model. Then the objectives of this research, the problem statement and author's contribution towards the topic have been presented. Finally, organization of this thesis is presented.

1.2 Background

Recent years have seen unprecedented increase in the development and spread of Android malware. The newer types of Android malware are not just proof of concept or early code but they are totally purposeful and matured. Since Android has now become the most widely used smart phone operating system [1], malware is threatening the ever growing Android operating system. The vulnerabilities in the operating system and applications are being exploited by the hackers in order to penetrate into the systems, steal user data and gain financial benefits by compromising the confidentiality, integrity and availability of Android applications and user data [2]. Malwares are found in applications which are cracked or which are not officially available on Google Play. Like cracked windows applications it is becoming increasingly difficult to stop the spread of cracked Android applications because they are available for free even if the original version is costly. Researchers around the world are analyzing android malware by studying its code, features, and functionalities. The objectives which govern the analysis of malicious android applications are, but not limited to:

1. To study and understand the loopholes in applications and operating system that might have provided the malware with a safe heaven.
2. To analyze and find out the features present in the malware and present remedial measures
3. To investigate the severity of malware attack by understanding the malicious code section and functionality.

1.3 Android Operating System

The Android platform was created by the Android Inc. which was later bought by the technology giant Google and later released as the Android Open Source Project. A consortium of 78 different companies formed the Open Handset Alliance whose key responsibility is to develop and distribute the Android Operating System [3]. The development of Android Operating System takes place rapidly, as a newer version replaces the older one after every few months.

1.3.1 Structural Overview of Android Operating System

The Android operating system's architectural stack is shown in figure 1.1 that can be subdivided into five different layers [4]. Applications are at the top, then the Application framework, then Android Runtime, Libraries followed by the lower level tools and Linux Kernel.

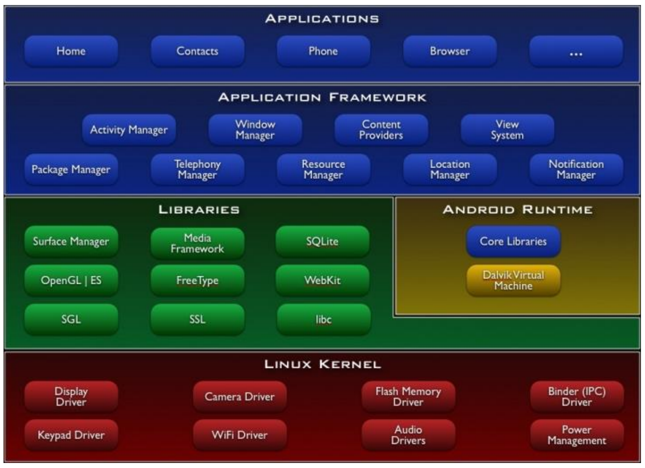


Figure 1: Android Operating System Architecture

The kernel in use is Linux 2.6 series kernel that has been modified and updated to meet some specific requirements like power management, runtime environment and memory management. Above the kernel some Linux typical daemons run for different functions e.g. Bluetooth support and Wi-Fi encryption.

Android operating system is supposed to run on devices with stringent battery conditions with little main memory and CPUs which are low powered, the libraries for CPU intensive tasks are compiled to device optimized native code. In the libraries layer, the surface manager takes care of the screen access for the window manager. The media framework that includes audio and video codecs also resides in this layer.

The Android Runtime consists of the Dalvik virtual machine [5] and Java core libraries. The Dalvik virtual machine provides a virtual environment for the Dalvik byte code to execute that has been transformed from the java byte code.

Frameworks in the application framework layer are written in Java and provide abstractions of the underlying native libraries and Dalvik capabilities to applications. Android applications run in a separate environment and contain multiple components like Activities, Services, Intents, broadcast receivers and content providers.

1.3.2 Components of Android Applications

Components of Android applications contain Activities, Services, Broadcast Receivers and Content providers. Applications can start other applications or specific components of other applications by sending Intent. These intents contain among other things the name of desired executed action. The Intent Manager resolves incoming intents and starts the proper application or component. The reception of Intent can be filtered by an application. Services and broadcast receivers allow applications to perform jobs in the background and provide additional functionality to other components. Broadcast receivers can be triggered by events and only run a short period of time whereas a service may run a long time.

1.3.3 Security Analysis of Android

The core Android development team knows that a robust security model was required to enable guaranteed application security for all applications. As a result, through its entire development lifecycle, Android has been subjected to a professional security program. The Android team has had the opportunity to observe how other mobile, desktop, and server platforms prevented and reacted to security issues and built a security program to address weak points observed in other offerings. But still malware authors find different ways to inject malicious code in benign applications e.g. by cracking them or altogether crafting new applications to meet the purpose for example the cracked version of a famous and widely used Android application, Instagram was found to be malicious [6]. Apart from Google Play, Google's official android application repository, many other forums and websites host applications which have not been analyzed by Google's security team. These types of applications when installed on an Android phone perform malicious actions of a variety of nature.

1.3.4 Recent Vulnerabilities in Android Applications

Almost twenty five percent of Android applications contain code that can access application permissions and cause security vulnerabilities, according to a recent study by mobile security firm TrustGo. Of the 2.3 million Android applications analyzed by TrustGo in the fourth quarter of 2012, 511,000 were identified as high risk, defined as being able to make unauthorized payments, steal data or modify user settings. Not all of the apps are universally available. For example, just 10 percent of apps in the US and Western Europe had a high risk for causing security issues. While China was reported to have the most high risk apps available for download. According to TrustGo's report, 77 percent of all apps available in China pose a high risk for security breaches [7]. The five riskiest app stores to download Android apps from were also reported to be based in China.

1.4 Android Malware Analysis

Android malware is evolving in a rapid manner. According to McAfee Security Company, its database contains more than 90 million samples as Android malware has increased multifold over the years [8]. It is becoming quite challenging to detect zero day malware as they are using new methodologies, signatures and encapsulation techniques. There are many antivirus solutions available for Android which helps in detecting and combating Android malware but a large number of highly sophisticated malware go undetected. Malware researchers study different variables related to the rise, spread and development of Android malware along with the activities it performs.

1.4.1 Android Malware Static Analysis

This research focuses on the static analysis of malicious Android applications. This type of analysis is used to study and analyze the behavior of a malicious sample by examining the code of the application. The other method is the Dynamic Analysis of Android applications which check the behavior of these applications after executing them, preferably in a sandbox environment. It looks for certain features, rules, functions, traces, artifacts, API calls, routines etc. that help in acquiring a complete picture of the overall functionality of the malicious file. Since it is extremely hard to acquire the source code of Android malware samples so the binary code is analyzed to serve the purpose. Many software and solutions are available in the form of debuggers and de-compilers that convert the malware code to its binary or assembly level [9].

1.5 Android Malware Analysis Tools

Various open source tools were used during the course of this research that helped in analyzing the Android applications. These tools helped in decompiling, disassembling, extracting, analyzing and understanding the malicious Android samples.

1.5.1 Android Software Development Kit

Android SDK is an emulator that is widely used to create local Android virtual devices. It contains many packages that are related to mobile phones, android platforms and software development. It needs to be installed along with other packages [10]. It also contains of a panel which can be used to create virtual devices and install latest packages.

1.5.2 Apktool

Apktool [11] is actually a disassembler that is used to analyze Android malware. In this research, Apktool has been used to decode the AndroidManifest.xml file to its original condition. It can also be used to recompile the code after modifications. Apktool makes use of Baksmali for disassembling and Smali for assembling.

1.5.3 7-Zip

7-Zip has been used to decompress the Android APK files in order to extract the dex and manifest file along with resources, libraries and assets of the Android application. This is a very good un-packer with high compression ratio and strong encryption as compared to WinZip.

1.5.4 UltraEdit Text Editor

There are various text editors available like Notepad++ etc. that are used to edit and view the Dex and manifest files. UltraEdit is very easy to use and has a high execution time along with being user friendly. It has a lot of options that makes the analysis part very easy.

1.5.5 Dex2Jar

It has been discussed that the java code is converted to Dalvik executable byte code due to limited power and memory issues. During analysis there is a need to convert that Dex byte code back to java format. Dex2Jar [12] is an easy to use command line tool which serves the purpose. It is platform independent and converts the Dex byte code to readable java code.

1.5.6 JD-GUI

When the Dex byte code has been converted to the readable java code, an easy to use tool JD-GUI [13] is used to view the java code. This gives information about all classes, functions, methods, API calls and routines in the java code.

1.6 Statement of Problem

The algorithm proposed in this research deals with the analysis of malicious applications on smart phones running the Android operating system. There are various commercial applications which serve the purpose but their methodology and source code is unknown. Other research tools and applications written by various Android security researchers have provided a better understanding of how newer types of malware work and also help in analyzing their behavior. But most of them either yield unsatisfactory results or do not cater for zero day malware samples. These algorithms are also not fully tested on large datasets which also raise doubts on their ability. The proposed research analyses Android applications and a novel and efficient anti-malware algorithm has been proposed. However the challenge of detecting newer types of malware has been growing.

1.7 Objectives

The following are the objectives of this research work:

1. To propose an optimal and sophisticated anti-malware algorithm that detects malicious android applications on the basis of a wide range of features present in an application.
2. To present an application for proof of concept that takes an Android application as input and outputs the decision whether the application was malicious or benign.
3. To develop a set of rules which can be tested on a large dataset so that statistical approaches are applied to select and reject a rule based on its presence in both malicious and benign samples.
4. To develop a proof of concept detection tool for android malware

1.8 Author's Contribution

1. The author has proposed an anti-malware algorithm which is supported by a proof of concept malware detection tool written using Python and C language. The tool takes an Android application as input and detects whether it is malicious or not. The tool has also been tested on very recent Android malware samples and the results have been promising.

2. The proposed algorithm looks for certain features in the Android application's code. These features help in knowing the occurrence of malicious rules in the application. Each rule occurrence carries some weight and the overall malicious weight is computed by adding the weights of all rules that are found out in the application.
3. The proposed algorithm makes use of data mining and statistical approaches during the development of the anti-malware algorithm. This is a significant feature of the proposed algorithm as using statistical and data mining approaches incurs efficiency. The algorithm was made after analyzing 958 malicious and 816 benign applications. The large dataset that has been used to devise the algorithm is another strength of the
4. The proposed algorithm was tested on 246 malicious and 758 benign applications and an accuracy of 98.32% was observed which is considerably high as compared to the accuracy of other algorithms discussed in Chapter 2.

1.9 Thesis Organization

The remaining portion of this thesis is organized as follows. Chapter 2 presents a brief literature review of various anti-malware algorithms for Android operating system and their pros and cons. Chapter 3 presents details about datasets used to develop and evaluate the performance of the proposed anti-malware algorithm. Chapter 4 explains the proposed anti-malware algorithm in detail. Chapter 5 presents the results and analysis of proposed algorithm on various malicious and benign Android applications. Chapter 6 concludes the carried research, explains its limitations and gives brief insight into future directions.

LITERATURE REVIEW

2.1 Chapter Overview

This chapter presents a brief overview of existing algorithms which perform static analysis of android applications, discusses their performance and efficiency and highlights pros and cons of these algorithms.

2.2 Stowaway

Stowaway is a tool that demystifies Android applications to see whether a particular compiled android application is over privileged or not. The way Stowaway works [14] is that it finds the API calls an application uses and then maps those API calls to an application's permissions that have been called in the AndroidManifest.xml file.

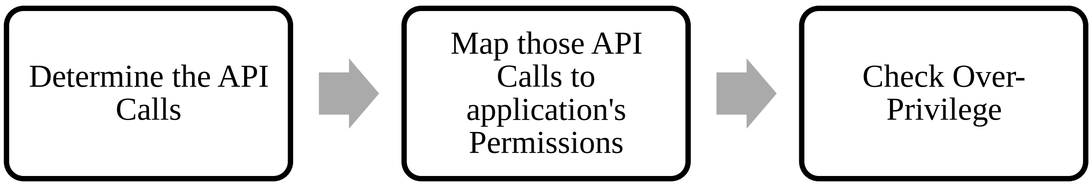
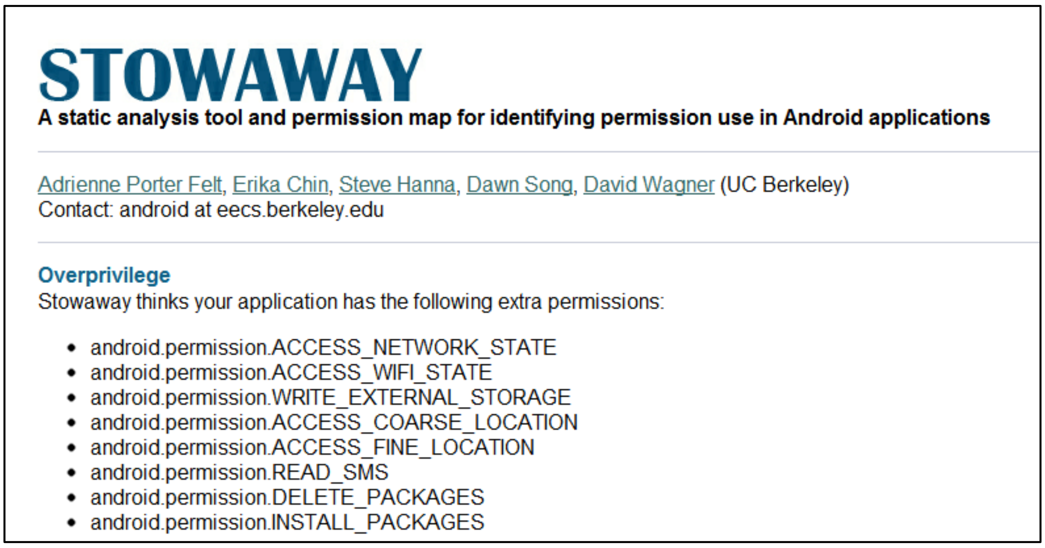


Figure 2: Stowaway's Functionality

2.2.1 Android Permission Map

Since the access control policy of Android is not thoroughly documented but in order to determine whether an application is over privileged or not, there was an immense need to build a permission map for Android. This is the best feature of the research carried out while building stowaway that the researchers have developed a permission map that characterizes the permission required for each API calls that an application makes. The final output of this research was a tool named 'Stowaway' that ascertains the maximum set of permissions any application might ask for. The tool analyses how the application makes use of API calls, Intents, Broadcast receivers and Content Providers to analyze the permissions required by these processes and operations. In this manner Stowaway computes the maximum set of permissions an application can use and after that it computes the actual permissions being used by the application. If the actual permissions being used by the application are greater than the maximum permissions required by the application to smoothly run, then it is over privileged. A set of 940 applications have been used for this analysis. Any android application can be analyzed using stowaway's online portal [15] where applications are uploaded for analysis. A screen shot of an android application being after analysis is shown below.



STOWAWAY

A static analysis tool and permission map for identifying permission use in Android applications

Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, David Wagner (UC Berkeley)
Contact: android at eecs.berkeley.edu

Overprivilege

Stowaway thinks your application has the following extra permissions:

- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_WIFI_STATE
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION
- android.permission.READ_SMS
- android.permission.DELETE_PACKAGES
- android.permission.INSTALL_PACKAGES

Figure 3: Android Malware Gold Dream tested on Stowaway

2.2.2 Critical Analysis of Stowaway

This research work provides a very useful basis for understanding the overall phenomenon of how permissions in Android work. It also utilizes ComDroid [16] which is a tool that determines the threat imposed by dynamic intents. But the most important question that arises here is that can any information be acquired about the maliciousness of an application if it is over privileged, and the following arguments elucidate the matter further.

1. An application can be malicious even if it is not over privileged. This means that the permissions demanded by different operations such as API calls, Intents, Broadcast receivers and Content providers are justified and no extra permission has been demanded by the application. The application can perform its malicious job without necessarily demanding extra permissions.
2. The other claim is that an application can be over privileged but not malicious. This can happen because developers at times do not follow development best practices and add extra permissions in their applications just to be on the safer side because they never want their applications to fail at any point of time in future. These developer errors can be avoided with proper software quality assurance and testing but inexperienced developers just release the application without having it properly tested. So this cannot be ascertained that an application is suspicious or malicious if it is over privileged.

2.3 Findroid

Findroid [17] is a tool that statically analyzes android applications and computes the risk score based on the features it sees in the applications decompiled code. Also if the score is greater than a particular threshold than the android application is labeled as malicious. The algorithm depends on property detectors which were implemented and categorized as Permissions, API call detectors, Command detectors, Presence of executable or zip files in resources and assets, geographic detectors, URL detectors, code size and some set combinations. The algorithm is tested on different android applications and a total risk score is computed which would tell whether an applications is benign or malicious.

2.3.1 Issues with Findroid

The biggest issue with Findroid is that it employs no statistical or data mining approach to compute the score for any property detector. It also follows no technical approach to set a value to the threshold that acts as a benchmark to decide maliciousness of an application. Similarly, only seven property detectors are not sufficient to decide on the maliciousness of an application.

2.4 The Genome Project

Another commendable research work that has been carried out in dissecting android malware is known as the Genome project. This research work [21] successfully systematized and characterized the Android malware available from August 2010 to October 2011. A total of 1260 malware samples have been categorized in 49 families. Four antivirus applications i.e. AVG Antivirus, Lookout Security and Antivirus, Norton Mobile Security Lite and TrendMicro Mobile Security Personal Edition were used to detect maliciousness of these applications. If any application is detected as malicious, these Antiviruses will generate a warning that is recorded by a script written by the researchers in Perl. The scanning results of these four antiviruses are shown below in descending order of performance.

Antivirus	**No. of Malware Families Scanned**	**No. of Applications detected as malicious**
Lookout	39	1003
TrendMicro	42	966
AVG	32	689

Norton	36	254

Table 1: Android Malware Gold Dream tested on Stowaway

2.4.1 Issues with the Genome Project

The research work is appreciable as far as the categorization of android malware is concerned, since it is also available on request for future researchers working in Android Security, but there are some deficiencies in the work.

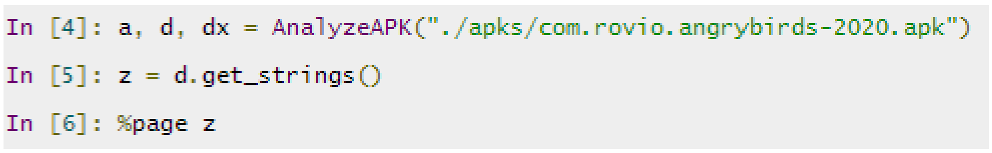
First of all these four antiviruses probably use malware signatures for detection purposes and do not analyze the application's code statically. That is one obvious reason that they fail to detect few latest malwares like BeanBot, CoinPirate, Droid Coupon, DroidKungfuSapp, NickyBot and RogueLemon. The results of the mobile security solutions used are somewhat disappointing as experiments show that the best case detects 79.6 percent of malware, while the worst case detects only 20.2 percent of malware available in the dataset.

2.5 Kirin: Mobile Phone Application Certification

Another useful research work that has been carried out was the development of Kirin [22], an Android application that mitigates malware and allows legitimate applications to be installed on the mobile phone. Kirin works on the principle of rules which when found in an application term it as malicious. A total of nine rules have been made to decide on the maliciousness of an application. The issue is that these nine rules can just detect specific types of malware and might even reject legitimate applications from getting installed. So Kirin can curtail few types of malware but it is not a thorough security application that can be used for wider purposes. Also a set of just nine rules is not sufficient to detect a variety of malwares which are so widely spread in case of Android operating system.

2.6 Androguard

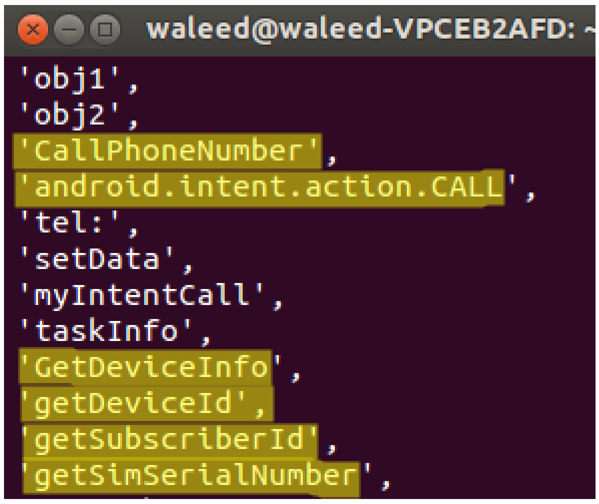
Androguard [23] provides the most useful set of commands and tools for reverse engineering, malware and code analysis of Android applications. It is a tool mainly written in python to play with the Android APK files, DEX (Dalvik executable) files and Android manifest files. It helps in decompiling, disassembling and modifying the Android application's code. It also helps in the static analysis of the code. It is also used for reverse engineering of Android applications and also provides a feature for finding similarities and differences in Android Applications. It would also tell where a particular permission is being used and called in the code and which API call or intent specifically calls the permission. For instance the following set of commands is used to display the strings present in an applications code.



```
In [4]: a, d, dx = AnalyzeAPK("./apks/com.rovio.angrybirds-2020.apk")
In [5]: z = d.get_strings()
In [6]: %page z
```

Figure 4: Acquiring strings in an APK code using Androguard

The following figure shows the strings acquired from the code of Gold Dream malware.



```
waleed@waleed-VPCEB2AFD: ~
'obj1',
'obj2',
'CallPhoneNumber',
'android.intent.action.CALL',
'tel:',
'setData',
'myIntentCall',
'taskInfo',
'GetDeviceInfo',
'getDeviceId',
'getSubscriberId',
'getSimSerialNumber',
```

Figure 5: Strings obtained from Gold Dream using Androguard

2.7 Chapter Summary

In this chapter, various research works in the field of Android security and malware analysis have been discussed. Stowaway is good tool for checking over privileged applications while it does not discuss the maliciousness of an application. Findroid lacks statistical and data mining approaches, Genome research project does not produce promising results while Kirin cannot mitigate a variety of Android malware. Androguard on the other hand is good for research purposes and cannot be used in real time.

C h a p t e r   3

DATASETS

3.1 Chapter Overview

This chapter provides the detail of all malware samples being used for experimentation and validation of the proposed algorithm for mitigation of malware. Many of these samples have been used as datasets for other researches that have been carried out in the same field. These samples belong to different malware families and are obtained from various sources.

3.2 Android Malware Genome Project

In the genome project [24], the existing Android malware has been characterized by a group of researchers after a yearlong effort. A set of 1260 Android applications were collected and categorized in 49 different malware families. The characterization of these malware samples depends on their installation methods, nature of carrying malicious payloads and behavior. The Android Malware Genome Project was supported in part by the US National Science Foundation (NSF) [8] while Google, SOURCEfire, NQ Mobile and Trend Micro supported the project with Hardware donations.

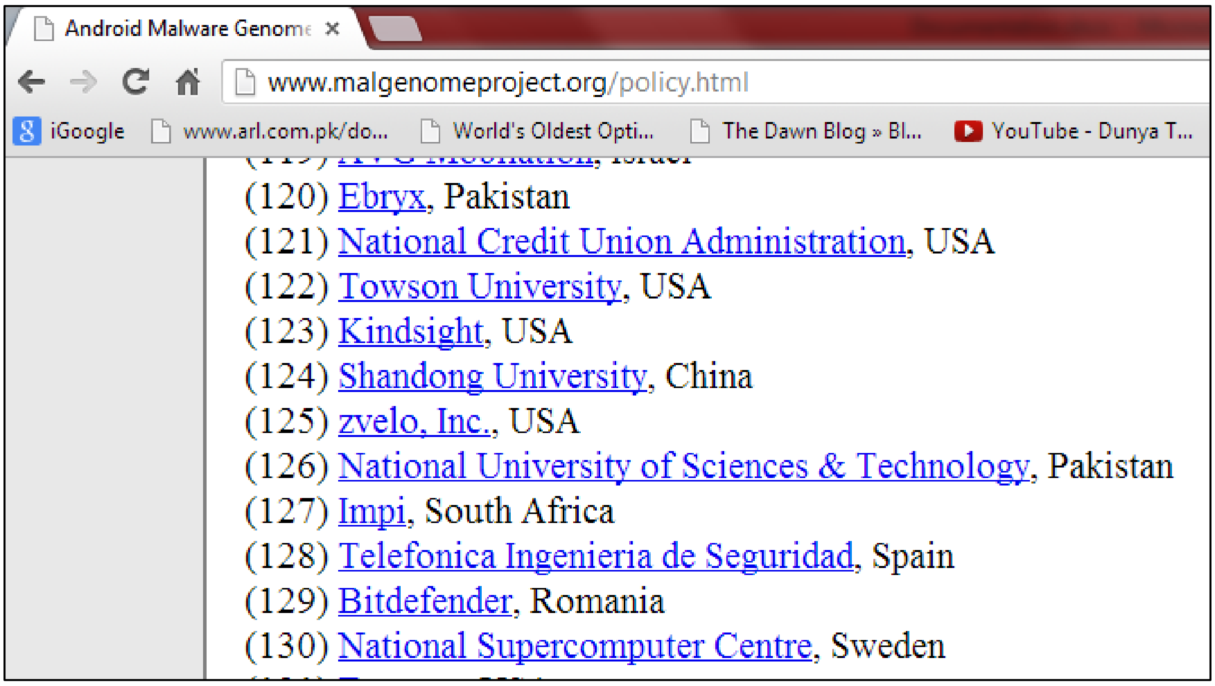
3.2.1 Genome Dataset Release Policy

In order to continue and aid further research in mitigating Android Malware, the researchers working on the Android malware genome project released their dataset to the Android security community. In order to avoid its probable misuse, proper authentication was required to acquire their dataset. Their dataset is available to both researchers working in the industry and academia.

1. For academic research, the student(s) needs to contact through his/her university email also mentioning the faculty involved in the research.
2. For industrial purposes, the researcher(s) need to contact through his/her official email along with the justification clearly stating the reasons why dataset is being requested.

3.2.2 Acquisition of Genome Dataset

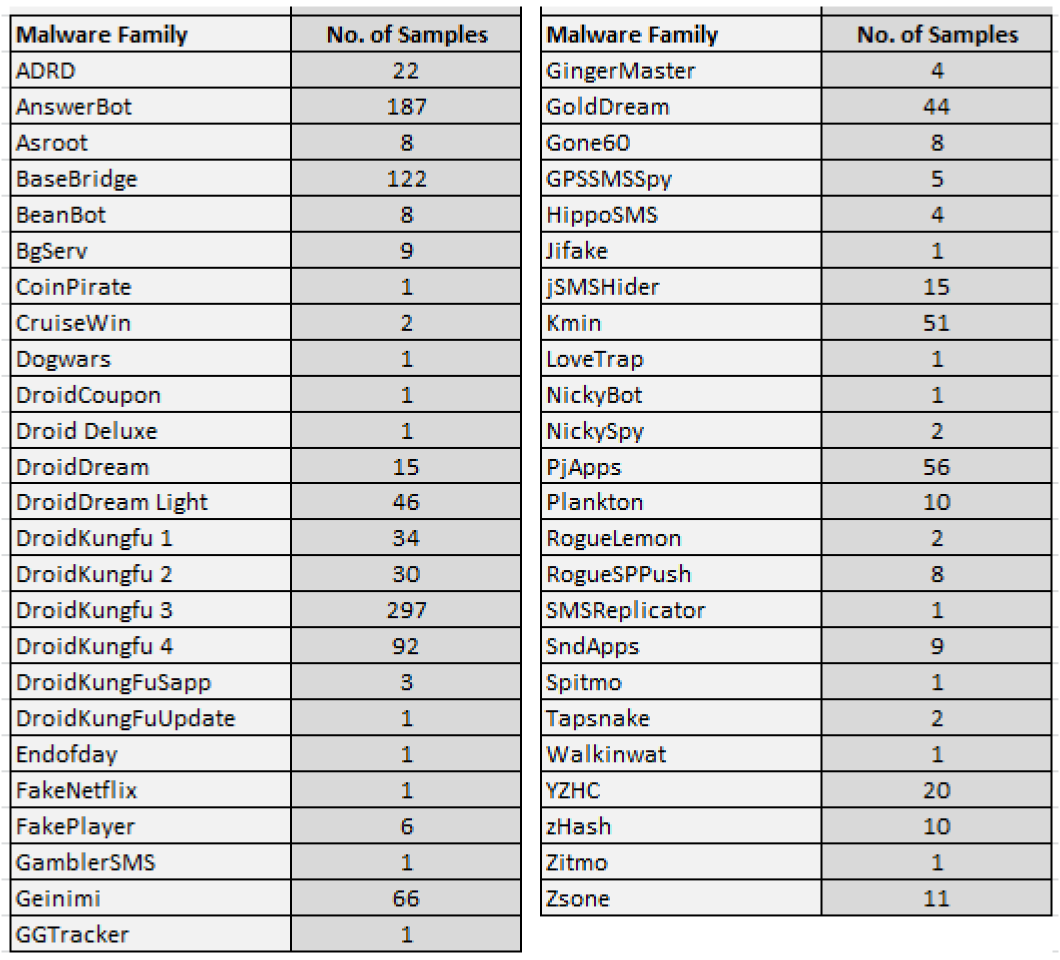
The genome Android malware dataset was acquired by officially requesting the Genome team. The Genome Project has thus mentioned the name of National University of Sciences and Technology (NUST), Pakistan on their portal where they have mentioned universities, research labs and companies for whom the dataset has been released.



(120) Ebryx, Pakistan
(121) National Credit Union Administration, USA
(122) Towson University, USA
(123) Kindsight, USA
(124) Shandong University, China
(125) zvelo, Inc., USA
(126) National University of Sciences & Technology, Pakistan
(127) Impi, South Africa
(128) Telefonica Ingenieria de Seguridad, Spain
(129) Bitdefender, Romania
(130) National Supercomputer Centre, Sweden

Figure 6: NUST has now been mentioned at Android Malware Genome project

The 49 different malware families along with the number of samples contained in each of these families is described in the table below. These malware samples were used during different phases of Algorithm development and testing. The results obtained while testing the proposed algorithm on these samples were compared with the results produced by the Genome project itself in later chapters.



Malware Family	No. of Samples
ADRD	22
AnswerBot	187
Asroot	8
BaseBridge	122
BeanBot	8
BgServ	9
CoinPirate	1
CruiseWin	2
Dogwars	1
DroidCoupon	1
Droid Deluxe	1
DroidDream	15
DroidDream Light	46
DroidKungfu 1	34
DroidKungfu 2	30
DroidKungfu 3	297
DroidKungfu 4	92
DroidKungFuSapp	3
DroidKungFuUpdate	1
Endofday	1
FakeNetflix	1
FakePlayer	6
GamblerSMS	1
Geinimi	66
GGTracker	1
GingerMaster	4
GoldDream	44
Gone60	8
GPSSMSSpy	5
HippoSMS	4
Jifake	1
jSMSHider	15
Kmin	51
LoveTrap	1
NickyBot	1
NickySpy	2
PjApps	56
Plankton	10
RogueLemon	2
RogueSPPush	8
SMSReplicator	1
SndApps	9
Spitmo	1
Tapsnake	2
Walkinwat	1
YZHC	20
zHash	10
Zitmo	1
Zsone	11

Figure 7: Malware families along with the number of samples used as dataset

3.2.3 Dataset: Algorithm Development

From the above dataset obtained, few families were used during the process of Algorithm development. A total of 957 samples from the following malware families were used during the selection of rules and making of the algorithm.

Genome Malware Families (Algorithm Development)	ADRD
	AnswerBot
	BaseBridge

	DroidDream
	DroidDream Light
	DroidKungfu 3
	DroidKungfu 4
	Geinimi
	jSMSHider
	Kmin
	PjApps
	Plankton
	Zsone

Table 2: Genome Malware Families used for Algorithm Development

3.2.4 Dataset: Algorithm Testing

For testing the proposed algorithm, different malware samples were used including the Android genome malware project and Contagiodump [9]. 213 samples from different Android genome malware families which were used for testing the proposed algorithm are mentioned in the table below.

Genome Malware Families (Algorithm Testing)	RogueLemon	RogueSPPush
	SMSReplicator	Spitmo
	Walkinwat	Tapsnake
	LoveTrap	NickyBot
	NickySpy	Jifake
	GGTracker	GingerMaster
	GamblerSMS	DroidKungFuUpdate
	FakeNetflix	CoinPirate

	CruiseWin	Dogwars
	DroidCoupon	Droid Deluxe
	Asroot	BeanBot
	BgServ	DroidKungFuSapp
	GoldDream	

Table 3: Genome Malware Families used for Algorithm Testing

3.3 Contagio Dump

For testing purposes different newer malware samples were acquired from Contagio blogspot [25]. Some of these malware samples hardly a few weeks old. Contagio is a well-known repository for acquiring malicious samples for different platforms.

3.4 Acquisition of Benign Android Applications

The benign applications were acquired from Google play using the Android Market API. Since Google does not allow direct downloading of Android applications on computers, there are various hacks available like the Android Market API and different tools. Few benign applications were also acquired from other research teams working in the field of Android forensics and security in the industry.

3.5 Chapter Summary

This chapter presented details about different datasets used during the development, rectification and testing of the algorithm. These datasets were acquired mainly from the Android genome malware project that consisted of 49 families and 1260 samples. Malware samples were also acquired from Contagio malware dump while benign applications were acquired from Google Play following a procedure mentioned in the chapter.

C h a p t e r   4

PROPOSED ALGORITHM

4.1 Chapter Overview

The chapter presents, in detail, the proposed anti-malware algorithm for Android applications. First of all it discusses the application features and malicious rules that are needed to make the algorithm followed by the refinement and enhancement of the algorithm using data mining and statistical techniques.

4.2 Android Application Features

Every application has some features that provide some knowledge about the behavior of that application. It is important to know that Android application features cannot be regarded as benign, suspicious or malicious; a feature is just a feature that helps in understanding the functionality of an application. Later on it would become clear that how these features are used to decide about the maliciousness of Android applications. Sending SMS, sending MMS, making voice calls, writing to external storage, accessing internet, starting a service, using Bluetooth or accessing WiFi information etc. can be termed as features but most of them cannot be labeled as malicious features.

4.2.1 Features used in the Algorithm

As discussed in Chapter 1, Android application comes in the APK format which when extracted contains the AndroidManifest.xml file, the DEX file, assets and resources. The proposed algorithm looks for these features in the Androidmanifest.xml and classes.dex file. Apparently, it might occur that why not to find these features in Java source code which is in the JAR file that is obtained after converting byte code to Java code. But as the Dalvik byte code is obtained by converting Java code to DEX format while developing the application so while reverse engineering it, it is enough to just look for these features in the DEX file.

So for this purpose three scripts were written in Python which parse the AndroidManifest.xml file, classes.dex file and both combined. The three Python parsers written are

DexFeatures.py	Parses DES file
ManifestFeatures.py	Parses AndroidManifest.xml file
MutualFeatures.py	Parses both DEX and Manifest file

Table 4: Parsers written in Python to find Features in DEX and Manifest file

So these parsers find features in the files specified to them. Since each feature has been assigned a unique number, the features along with their number and name are stored in a comma separate file for further use.

The list of various features which the parsers search in the Android application is given in the table below.

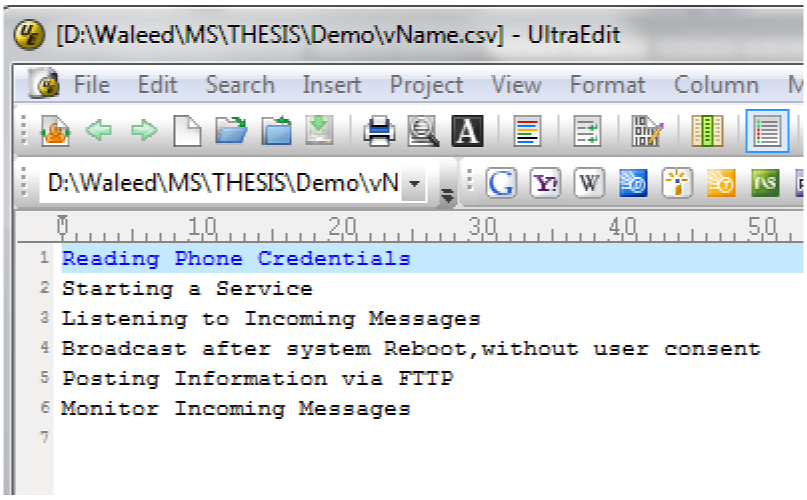
Features
Reading Phone Credentials
Sending SMS
Downloading Files from Internet
Establishing connections over the Internet
Accessing Wi-Fi Information
Reading Contacts
Making a File
Uploading Files on the Internet
Starting a Service

Listening to Incoming Messages
Installing Packages
Copying Assets to the Phone
Enabling the USB Mode
Installing another Application
Using Encryption
Changing Permissions
Using Hashing Algorithms
Running Exploits
Accessing Social Media sites
Establishing Bluetooth Connection
Accessing FINE, COARSE, MOCK Location
Writing to External Storage
Monitoring SMS, MMS
Recording Audio
Web Search
Making a Zip Archive
Making Files on the device
Broadcast after System Reboot
Contacting a Server
Monitor, Modify, Abort Outgoing Calls
Writing Incoming Messages
Getting list of Installed Packages
Writing to System Settings

Deleting Installed Packages
Reading Low level System Logs
Changing Wi-Fi connectivity state
Mounting/Un-mounting File Systems
Turning debugging Mode for other Applications
Recording Audio
Formatting File systems for removable storage

Table 5: Features which are being looked for in Android Applications

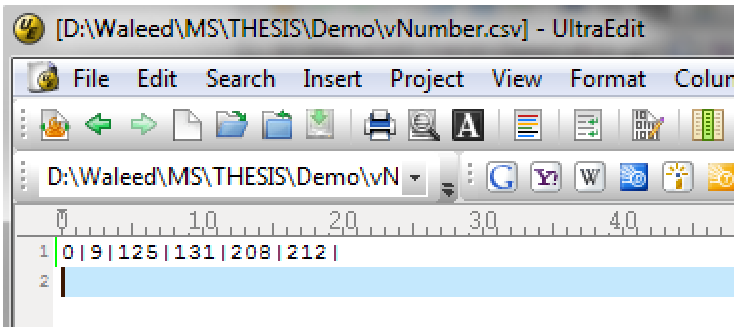
If the parser script written in Python finds any feature, then the name and number of this feature is stored in two separate comma separated files respectively. For instance after scanning the Android malware ZITMO which was acquired from Contagio dump, the following screenshot is taken from the file which stores the names of features found in it.


[D:\Waleed\MS\THESIS\Demo\vName.csv] - UltraEdit
File Edit Search Insert Project View Format Column
D:\Waleed\MS\THESIS\Demo\vN

```
Reading Phone Credentials
Starting a Service
Listening to Incoming Messages
Broadcast after system Reboot,without user consent
Posting Information via FTTP
Monitor Incoming Messages
```

Figure 8: Names of features found in ZITMO malware

Similarly the numbers assigned to these features are also stored in a separate file and the following screenshot displays them.


[D:\Waleed\MS\THESIS\Demo\vNumber.csv] - UltraEdit
File Edit Search Insert Project View Format Colur
D:\Waleed\MS\THESIS\Demo\vN

```
0|9|125|131|208|212|
```

Figure 9: Number of each feature found in ZITMO malware

4.3 Establishing Rules on basis of Features

As discussed earlier, features depict the overall behavior of any application and do not alone help in deciding whether an application is malicious or not. For this purpose, certain Malware Rules [26] have been established with the help of these features that clearly help in deciding whether an application is malicious or not. It should be noted that in most of the cases, different features combine to make one rule and in fewer cases a singular feature can itself act as a rule. The latter case appears rarely because it is quite uncommon for a single feature to be considered malicious.

The rules which have been made for determining the maliciousness of an application are listed below.

	Rule #	**Features in each Rule**
Reading Phone Credentials	**1a**	android.permission.READ_PHONE_STATE android.permission.INTERNET (conn) android.permission.INTERNET (upload) Making File Contacting a Remote Server
	1b	android.permission.READ_PHONE_STATE android.permission.INTERNET (Upload) Zip Archive ***OR File***
	1c	android.permission.READ_PHONE_STATE android.permission.INTERNET (conn) Zip Archive ***OR***

	1d	android.permission.READ_PHONE_STATE android.permission.INTERNET (conn)
	1e	android.permission.READ_PHONE_STATE
Sending SMS	**2a**	android.permission.SEND_SMS android.permission.INTERNET (conn) android.permission.INTERNET (upload) Zip Archive Making File Contacting a Remote Server
	2b	android.permission.SEND_SMS android.permission.INTERNET (Upload) Zip Archive ***OR*** Making File
	2c	android.permission.SEND_SMS android.permission.INTERNET (conn) Zip Archive ***OR*** Making File
	2d	android.permission.SEND_SMS android.permission.INTERNET (conn)
	2e	android.permission.SEND_SMS
Reading Contacts	**3a**	android.permission.READ_CONTACTS android.permission.INTERNET (conn) android.permission.INTERNET (upload) Zip Archive Making File Contacting a Remote Server
	3b	android.permission.READ_CONTACTS android.permission.INTERNET (Upload) Zip Archive ***OR*** Making File
	3c	android.permission.READ_CONTACTS android.permission.INTERNET (conn) Zip Archive ***OR*** Making File
	3d	android.permission.READ_CONTACTS android.permission.INTERNET (conn)

	3e	android.permission.READ_CONTACTS
Access Wi-Fi State	**4**	android.permission.ACCESS_WIFI_STATE android.permission.INTERNET (conn)
Listening to Incoming messages	**5a**	android.permission.RECEIVE_SMS android.permission.INTERNET (conn) android.permission.INTERNET (upload) Zip Archive Making File Contacting a Remote Server
	5b	android.permission.RECEIVE_SMS android.permission.INTERNET (Upload) Zip Archive ***OR*** Making File
	5c	android.permission.RECEIVE_SMS android.permission.INTERNET (conn) Zip Archive ***OR*** Making File
	5d	android.permission.RECEIVE_SMS android.permission.INTERNET (conn)
	5e	android.permission.RECEIVE_SMS
Installing Packages	**6a**	copy Assets android.permission.INSTALL_PACKAGES chmod 775 ***OR*** Root Shell
	6b	copy Assets chmod 775 ***OR*** Root Shell
	6c	copy Assets android.permission.INSTALL_PACKAGES
Encryption	**7**	Hashing SHA1 Key Ciphers PRNG Hashing MD5
Enabling USB Mode	**8**	Enabling USB Mode

Installing other Applications	**9a**	action.download_apk intsall apps Download files
	9b	action.download_apk intsall apps
Rooting/Shell Scripts	**10a**	action.download_shells chmod Root Shell
	10b	action.download_shells Root Shell
Phone State and Audio	**11**	android.permission.READ_PHONE_STATE android.permission.RECORD_AUDIO android.permission.INTERNET (conn)
Access FINE Location	**12a**	android.permission.ACCESS_FINE_LOCATION RECEIVE BOOT COMPLETED android.permission.INTERNET (conn)
	12b	android.permission.ACCESS_FINE_LOCATION android.permission.INTERNET (conn)
Access CORASE Location	**13a**	android.permission.ACCESS_COARSE_LOCATION RECEIVE BOOT COMPLETED android.permission.INTERNET (conn)
	13b	android.permission.ACCESS_COARSE_LOCATION android.permission.INTERNET (conn)
Receive/Write SMS	**14**	android.permission.RECEIVE_SMS android.permission.WRITE_SMS
Send/Write SMS	**15**	android.permission.SEND_SMS android.permission.WRITE_SMS
CALL_PRIVILEGED	**16**	android.permission.CALL_PRIVILEGED
Calling Phone	**17a**	android.permission.CALL_PHONE Internet android.permission.PROCESS_OUTGOING_CALL
	17b	android.permission.CALL_PHONE Internet

	17c	android.permission.PROCESS_OUTGOING_CALL Internet
	17d	android.permission.CALL_PHONE
	17e	android.permission.PROCESS_OUTGOING_CALL
Accessing Web and Social Media	**18a**	Accessing Twitter Internet Web Search
	18b	Accessing Twitter Internet
Uploading Bluetooth data	**19a**	Bluetooth Upload Files
	19b	Bluetooth
Starting Service	**20**	Service BOOT_COMPLETED INTERNET
Accessing MOCK Location	**21a**	android.permission.ACCESS_MOCK_LOCATION RECEIVE BOOT COMPLETED android.permission.INTERNET (conn)
	21b	android.permission.ACCESS_MOCK_LOCATION android.permission.INTERNET (conn)
Writing to External Storage	**23**	WRITE_EXTERNAL_STORAGE
Web Search	**24**	WEB_SEARCH DOWNLAOD INTERNET
Contacting Server	**25**	SERVER INTERNET
Installed Packages	**26**	LIST OF PACKAGES INTERNET

Reading Logs	**27**	READ_LOGS INTERNET
Mounting/Un-mounting File Systems	**28a**	MOUNTING/UNMOUNTING INTERNET
	28b	MOUNTING/UNMOUNTING
SET_DEBUG_APPS	**29a**	SET_DEBUG_APPS INTERNET
	29b	SET_DEBUG_APPS
Format File Systems	**30a**	FORMAT_FILE_SYSTEMS INTERNET
	30b	FORMAT_FILE_SYSTEMS
Delete Packages	**31**	android.permission.DELETE_PACKAGES
Write Settings	**32**	android.permission.WRITE_SETTINGS
Write Security Settings	**33**	android.permission.WRITE_SECURE_SETTINGS

Table 6: Rules which are being looked for in Android Applications

4.4 Checking Presence of Rules in Dataset

After these rules are made, the dataset used in making the algorithm as mentioned in Chapter 3 is checked for the presence of these rules. So it is important to know that these rules will be checked for in both the malicious and benign datasets.

4.4.1 Presence in Malware Dataset

An automated script written in Python checks for the presence of these rules in the malware dataset which consists of 957 samples. The malware dataset is placed at a different location on the drive and a batch file picks each sample, copies it to the location where the analysis is being performed, extracts and stores the results in a data repository and this process continues till the acquisition of complete results. The following Flowchart explains the process in an elucidated manner.

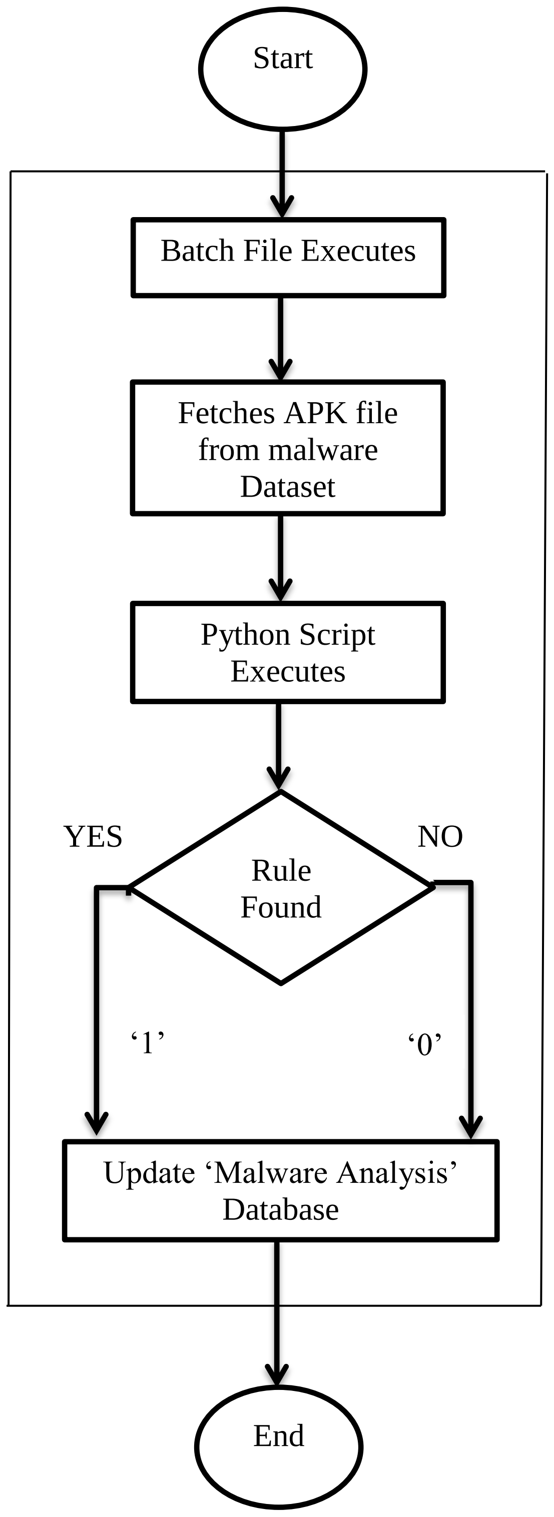

Start
Batch File Executes
Fetches APK file from malware Dataset
Python Script Executes
Rule Found
YES
NO
'1'
'0'
Update 'Malware Analysis' Database
End

Figure 10: Flowchart for finding Rules in Malware Database

After the process completes and results are gathered in the repository, presence of rules for overall malicious dataset is computed. The following steps would explain how information about the existence of rules is computed.

- When a particular rule is found in a malware sample, the algorithm returns 1, else it returns 0.
- Let '$\boldsymbol{N}$' be the total number of malware samples tested for Rule presence.
- Let $\boldsymbol{Oi}$ be the total number of 1s against a particular rule '$\boldsymbol{i}$' which means that Rule '$\boldsymbol{i}$' is found in '$\boldsymbol{Oi}$' samples out of the total '$\boldsymbol{N}$' samples.
- Number of samples where Rule '$\boldsymbol{i}$' is not found would be $\boldsymbol{N - Oi}$
- Now the probability of occurrence of Rule '$\boldsymbol{i}$' can be found out by:

$$\boldsymbol{P(i = 1) = Oi/N} \tag{4.1}$$

- And the probability of absence of Rule '$\boldsymbol{i}$' can be found out by:

$$\boldsymbol{P(i = 0) = (1 - Oi)/N} \tag{4.2}$$

Now, since it is clear how the probabilities of occurrence and absence of a rule are being calculated for the malware dataset, the actual results of all 67 rules are displayed in the table below:

RULES		**MALWARE**	
Name	**No.**	**P(0)**	**P(1)**
Reading Phone State & Internet	**1d**	0.017763845	0.982236155
Reading Phone State, Internet & Zip Archive	**1c**	0.257053292	0.742946708
Starting a Service	**20**	0.399164054	0.600835946
Accessing Wi-Fi State	**4**	0.287356322	0.712643678
Sending SMS and Internet	**2d**	0.562173459	0.437826541
Reading Contact & Internet	**3d**	0.562173459	0.437826541
Sending SMS and Internet & Zip Archive	**2c**	0.587251829	0.412748171
SEND/WRITE SMS	**15**	0.612330199	0.387669801
RECEIVE SMS & Internet	**5d**	0.595611285	0.404388715
RECEIVE/WRITE SMS	**14**	0.635318704	0.364681296
RECEIVE SMS, Internet & Zip Archive	**5c**	0.62800418	0.37199582
Enabling USB Mode	**8**	0.670846395	0.329153605

Calling Phone & Internet	**17b**	0.546499478	0.453500522
Contacting a Server	**25**	0.793103448	0.206896552
Access COARSE Location, Boot Completed & Internet	**13a**	0.764890282	0.235109718
Reading Logs	**27**	0.779519331	0.220480669
Access FINE Location, Boot Completed & Internet	**12a**	0.781609195	0.218390805
Call Phone	**17d**	0.545454545	0.454545455
Access COARSE Location & Internet	**13b**	0.5276907	0.4723093
Hashing	**7c**	0.228840125	0.771159875
Disallowed Broadcasts	**34**	0.92476489	0.07523511
Reading Contact, Internet & Zip Archive	**3c**	0.931034483	0.068965517
Mounting/Un-mounting & Internet	**28a**	0.920585162	0.079414838
Ciphers	**7b**	0.360501567	0.639498433
Process Outgoing Call & Internet	**17c**	0.955067921	0.044932079
Process Outgoing Call	**17e**	0.954022989	0.045977011
Deleting Packages	**31**	0.96969697	0.03030303
Writing to External Storage	**23**	0.310344828	0.689655172
Mounting/Un-mounting	**28b**	0.91954023	0.08045977
Call Phone, Process Outgoing Call & Internet	**17a**	0.967607106	0.032392894
Access FINE Location & Internet	**12b**	0.593521421	0.406478579
Write Settings	**32**	0.918495298	0.081504702
Receive SMS	**5e**	0.987460815	0.012539185
Reading Phone State	**1e**	0.988505747	0.011494253
Reading Phone State, Upload & Zip Archive	**1b**	0.995820272	0.004179728
Sending SMS	**2e**	0.995820272	0.004179728
Reading Contacts	**3e**	0.995820272	0.004179728
RECEIVE SMS, Upload & Zip Archive	**5b**	0.995820272	0.004179728
Reading Phone State, Upload, Zip Archive, Server	**1a**	1	0
Sending SMS, Upload, Zip Archive, Server	**2a**	1	0
Reading Contacts, Upload, Zip Archive, Server	**3a**	1	0
Reading Contacts, Upload, Zip Archive	**3b**	1	0
Receive SMS, Upload, Zip Archive, Server	**5a**	1	0
Copy Assets, Install Packages & Rooting	**6a**	1	0
Copy Assets & Rooting	**6b**	1	0
Copy Assets & Install Packages	**6c**	1	0
Download files & Install APK	**9a**	1	0
Download APKS	**9b**	1	0
Downloading Shell Script & Changing Permissions	**10a**	1	0
Downloading Shell Script & Rooting	**10b**	1	0
Accessing Web and Social Media	**18a**	1	0
Bluetooth Upload	**19a**	1	0
Web Search	**24**	1	0
List of Installed Packages & Internet	**26**	1	0
SET_DEBUG_APPS & Internet	**29a**	1	0
SET_DEBUG_APPS	**29b**	1	0

Format File Systems & Internet	**30a**	1	0
Format File Systems	**30b**	1	0
Reading Phone State, Recording Audio & Internet	**11**	0.973876698	0.026123302
Sending SMS, Upload, Zip Archive	**2b**	0.998955068	0.001044932
Writing Security Settings	**33**	0.991640543	0.008359457
Privileged Call Permission	**16**	0.996865204	0.003134796
Accessing MOCK Location, Boot Completed & Internet	**21b**	0.997910136	0.002089864
Bluetooth Upload	**19d**	0.997910136	0.002089864
Encryption, Hashing, Ciphering	**7a**	0.996865204	0.003134796
Accessing Twitter	**18b**	0.979101358	0.020898642

Table 7: Probability of Occurrence and Rejection of Rules in Malware Dataset

4.4.2 Presence in Benign Dataset

Another automated script written in Python checks for the presence of these rules in the benign dataset which consists of 816 samples in the similar manner as it was done for malicious samples. The following Flowchart explains the process in an elucidated manner.

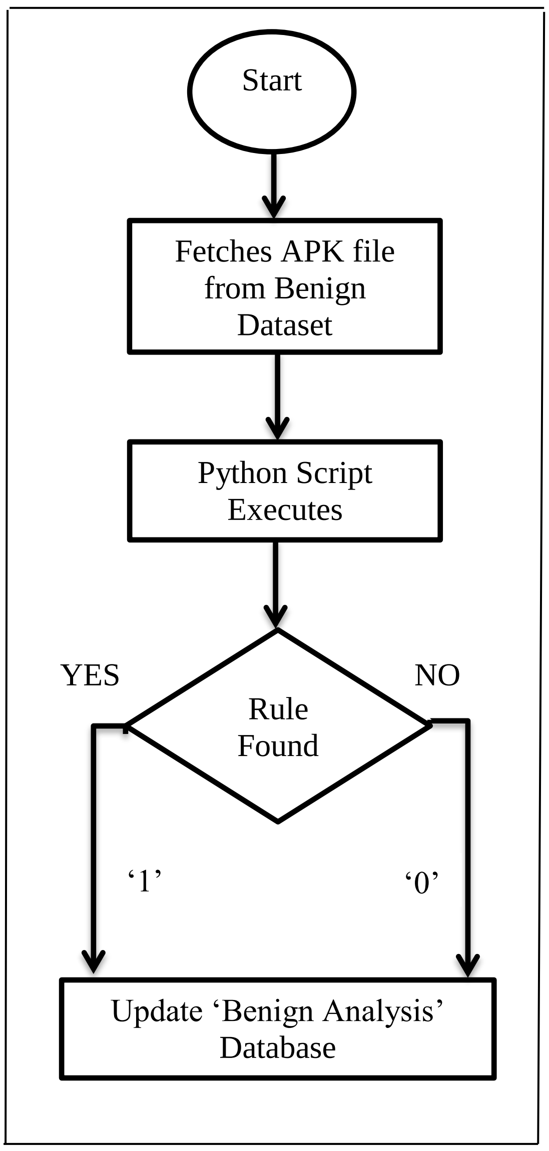

Start
Fetches APK file from Benign Dataset
Python Script Executes
Rule Found
YES
NO
'1'
'0'
Update 'Benign Analysis' Database

Figure 11: Flowchart for finding Rules in Benign Database

After the process completes and results are gathered in the repository, presence of rules for overall benign dataset is computed. The following steps would explain how information about the existence of rules is computed.

- When a particular rule is found in a benign sample, the algorithm returns 1, else it returns 0.
- Let '$\boldsymbol{M}$' be the total number of malware samples tested for Rule presence.
- Let $\boldsymbol{Oi}$ be the total number of 1s against a particular Rule '$\boldsymbol{i}$' which means that Rule '$\boldsymbol{i}$' is found in '$\boldsymbol{Oi}$' samples out of the total '$\boldsymbol{M}$' samples.

- Number of samples where Rule '$\boldsymbol{i}$' is not found would be $\boldsymbol{M - Oi}$
- Now the probability of occurrence of Rule '$\boldsymbol{i}$' can be found out by:

$$\boldsymbol{Q(i = 1) = Oi/M} \tag{4.3}$$

- And the probability of absence of Rule '$\boldsymbol{i}$' can be found out by:

$$\boldsymbol{Q(i = 0) = (1 - Oi)/M} \tag{4.4}$$

Now, since it is clear how the probabilities of occurrence and absence of a rule are being calculated for the malware dataset, the actual results of all 67 rules are displayed in the table below:

RULES		**BENIGN**	
Name	**No.**	**Q(0)**	**Q(1)**
Reading Phone State & Internet	**1d**	0.881127451	0.118872549
Reading Phone State, Internet & Zip Archive	**1c**	0.933823529	0.066176471
Starting a Service	**20**	0.962009804	0.037990196
Accessing Wi-Fi State	**4**	0.830882353	0.169117647
Sending SMS and Internet	**2d**	0.995098039	0.004901961
Reading Contact & Internet	**3d**	0.995098039	0.004901961
Sending SMS and Internet & Zip Archive	**2c**	0.996323529	0.003676471
SEND/WRITE SMS	**15**	0.991421569	0.008578431
RECEIVE SMS & Internet	**5d**	0.968137255	0.031862745
RECEIVE/WRITE SMS	**14**	0.993872549	0.006127451
RECEIVE SMS, Internet & Zip Archive	**5c**	0.975490196	0.024509804
Enabling USB Mode	**8**	1	0
Calling Phone & Internet	**17b**	0.781862745	0.218137255
Contacting a Server	**25**	0.99877451	0.00122549
Access COARSE Location, Boot Completed & Internet	**13a**	0.955882353	0.044117647
Reading Logs	**27**	0.964460784	0.035539216
Access FINE Location, Boot Completed & Internet	**12a**	0.955882353	0.044117647
Call Phone	**17d**	0.724264706	0.275735294
Access COARSE Location & Internet	**13b**	0.667892157	0.332107843
Hashing	**7c**	0.329656863	0.670343137
Disallowed Broadcasts	**34**	0.99877451	0.00122549
Reading Contact, Internet & Zip Archive	**3c**	0.993872549	0.006127451
Mounting/Un-mounting & Internet	**28a**	0.968137255	0.031862745
Ciphers	**7b**	0.432598039	0.567401961
Process Outgoing Call & Internet	**17c**	0.993872549	0.006127451
Process Outgoing Call	**17e**	0.990196078	0.009803922

Deleting Packages	**31**	0.99877451	0.00122549
Writing to External Storage	**23**	0.37254902	0.62745098
Mounting/Un-mounting	**28b**	0.958333333	0.041666667
Call Phone, Process Outgoing Call & Internet	**17a**	0.995098039	0.004901961
Access FINE Location & Internet	**12b**	0.645833333	0.354166667
Write Settings	**32**	0.947303922	0.052696078
Receive SMS	**5e**	1	0
Reading Phone State	**1e**	1	0
Reading Phone State, Upload & Zip Archive	**1b**	1	0
Sending SMS	**2e**	1	0
Reading Contacts	**3e**	1	0
RECEIVE SMS, Upload & Zip Archive	**5b**	1	0
Reading Phone State, Upload, Zip Archive, Server	**1a**	1	0
Sending SMS, Upload, Zip Archive, Server	**2a**	1	0
Reading Contacts, Upload, Zip Archive, Server	**3a**	1	0
Reading Contacts, Upload, Zip Archive	**3b**	1	0
Receive SMS, Upload, Zip Archive, Server	**5a**	1	0
Copy Assets, Install Packages & Rooting	**6a**	1	0
Copy Assets & Rooting	**6b**	1	0
Copy Assets & Install Packages	**6c**	1	0
Download files & Install APK	**9a**	1	0
Download APKS	**9b**	1	0
Downloading Shell Script & Changing Permissions	**10a**	1	0
Downloading Shell Script & Rooting	**10b**	1	0
Accessing Web and Social Media	**18a**	1	0
Bluetooth Upload	**19a**	1	0
Web Search	**24**	1	0
List of Installed Packages & Internet	**26**	1	0
SET_DEBUG_APPS & Internet	**29a**	1	0
SET_DEBUG_APPS	**29b**	1	0
Format File Systems & Internet	**30a**	1	0
Format File Systems	**30b**	1	0
Reading Phone State, Recording Audio & Internet	**11**	0.986519608	0.013480392
Sending SMS, Upload, Zip Archive	**2b**	1	0
Writing Security Settings	**33**	0.996323529	0.003676471
Privileged Call Permission	**16**	0.993872549	0.006127451
Accessing MOCK Location, Boot Completed & Internet	**21b**	0.992647059	0.007352941
Bluetooth Upload	**19d**	0.984068627	0.015931373
Encryption, Hashing, Ciphering	**7a**	0.944852941	0.055147059
Accessing Twitter	**18b**	0.829656863	0.170343137

Table 8: Probability of Occurrence and Rejection of Rules in Benign Dataset

4.4.3 Computing Differences between Malware and Benign Datasets

Since the information about the presence of all rules has been calculated for both Malicious and Benign datasets, the next step is to examine which Rules are more prevalent in malicious samples as compared to benign. The most apparent method to check the difference between the prevalence of rules in both sets is to compute the arithmetic difference between probabilities of existence or a rule in both datasets. This has been computed and shown below in the following table:

RULES		DIFFERENCE
Name	**No.**	**$\Omega = P(1)-Q(1)$**
Reading Phone State & Internet	**1d**	0.863363606
Reading Phone State, Internet & Zip Archive	**1c**	0.676770238
Starting a Service	**20**	0.56284575
Accessing Wi-Fi State	**4**	0.543526031
Sending SMS and Internet	**2d**	0.43292458
Reading Contact & Internet	**3d**	0.43292458
Sending SMS and Internet & Zip Archive	**2c**	0.409071701
SEND/WRITE SMS	**15**	0.37909137
RECEIVE SMS & Internet	**5d**	0.37252597
RECEIVE/WRITE SMS	**14**	0.358553845
RECEIVE SMS, Internet & Zip Archive	**5c**	0.347486016
Enabling USB Mode	**8**	0.329153605
Calling Phone & Internet	**17b**	0.235363268
Contacting a Server	**25**	0.205671062
Access COARSE Location, Boot Completed & Internet	**13a**	0.190992071
Reading Logs	**27**	0.184941453
Access FINE Location, Boot Completed & Internet	**12a**	0.174273158
Call Phone	**17d**	0.17881016
Access COARSE Location & Internet	**13b**	0.140201457
Hashing	**7c**	0.100816737
Disallowed Broadcasts	**34**	0.07400962
Reading Contact, Internet & Zip Archive	**3c**	0.062838066
Mounting/Un-mounting & Internet	**28a**	0.047552093
Ciphers	**7b**	0.072096472
Process Outgoing Call & Internet	**17c**	0.038804628
Process Outgoing Call	**17e**	0.03617309
Deleting Packages	**31**	0.02907754
Writing to External Storage	**23**	0.062204192

Mounting/Un-mounting	**28b**	0.038793103
Call Phone, Process Outgoing Call & Internet	**17a**	0.027490934
Access FINE Location & Internet	**12b**	0.052311912
Write Settings	**32**	0.028808624
Receive SMS	**5e**	0.012539185
Reading Phone State	**1e**	0.011494253
Reading Phone State, Upload & Zip Archive	**1b**	0.004179728
Sending SMS	**2e**	0.004179728
Reading Contacts	**3e**	0.004179728
RECEIVE SMS, Upload & Zip Archive	**5b**	0.004179728
Reading Phone State, Upload, Zip Archive, Server	**1a**	0
Sending SMS, Upload, Zip Archive, Server	**2a**	0
Reading Contacts, Upload, Zip Archive, Server	**3a**	0
Reading Contacts, Upload, Zip Archive	**3b**	0
Receive SMS, Upload, Zip Archive, Server	**5a**	0
Copy Assets, Install Packages & Rooting	**6a**	0
Copy Assets & Rooting	**6b**	0
Copy Assets & Install Packages	**6c**	0
Download files & Install APK	**9a**	0
Download APKS	**9b**	0
Downloading Shell Script & Changing Permissions	**10a**	0
Downloading Shell Script & Rooting	**10b**	0
Accessing Web and Social Media	**18a**	0
Bluetooth Upload	**19a**	0
Web Search	**24**	0
List of Installed Packages & Internet	**26**	0
SET_DEBUG_APPS & Internet	**29a**	0
SET_DEBUG_APPS	**29b**	0
Format File Systems & Internet	**30a**	0
Format File Systems	**30b**	0
Reading Phone State, Recording Audio & Internet	**11**	0.01264291
Sending SMS, Upload, Zip Archive	**2b**	0.001044932
Writing Security Settings	**33**	0.004682986
Privileged Call Permission	**16**	-0.002992655
Accessing MOCK Location, Boot Completed & Internet	**21b**	-0.005263077
Bluetooth Upload	**19d**	-0.013841508
Encryption, Hashing, Ciphering	**7a**	-0.052012263
Accessing Twitter	**18b**	-0.149444496

Table 9: Difference between presence of all Rules in malicious and benign datasets

4.5 Selection of Rules

Since the difference between the existence of rules in malicious and benign samples has been acquired and the rules with positive values of difference can be selected for the anti-malware but it has been ascertained that just the arithmetic difference is not a good measure for this purpose because it is not known that how greater the probability of existence of a particular rule in malicious samples should be from the probability of its existence in benign samples. The following method has been used to make the process of rule selection statistically correct.

4.5.1 Bernoulli's Trial

Bernoulli's Trial [24] is a statistical experiment whose result can be either two possible outcomes, either '1' or '0'. If Bernoulli's process [27] is applied to this particular research problem, it can be said that the Probability of Occurrence and Probability of Absence of a rule in both malicious and benign datasets are the only two possible outcomes e.g. 'existence of a rule' and 'absence of a rule' such that:

- '1' corresponds to 'Presence of Rule'
- '0' corresponds to 'Absence of Rules'

Applying Bernoulli's Trial to the scenario

$\boldsymbol{X}$	$\boldsymbol{P(X)}$	$\boldsymbol{X.P(X)}$	$\boldsymbol{X^2.P(X)}$
0	1-p	0	0
1	p	p	p
	1	$E(X) = p$	$E(X^2) = p$

Table 10: Applying Bernoulli's Trial to Rules

Applying Bernoulli's Trial to Rule '1c' for both malicious and benign datasets, the following information is acquired.

$\boldsymbol{X}$	$\boldsymbol{P(X)}$	$\boldsymbol{X.P(X)}$	$\boldsymbol{X^2.P(X)}$
0	0.017763845	0	0

1	0.982236155	0.982236155	p
	1	0.982236155	0.982236155

Table 11: Rule '1c' in Malware Dataset

$\boldsymbol{X}$	$\boldsymbol{P}(\boldsymbol{X})$	$\boldsymbol{X}.\boldsymbol{P}(\boldsymbol{X})$	$\boldsymbol{X}^{\boldsymbol{2}}.\boldsymbol{P}(\boldsymbol{X})$
0	0.881127451	0	0
1	0.118872549	0.118872549	p
	1	0.118872549	0.118872549

Table 12: Rule '1c' in Benign Dataset

4.5.2 Computing Variance and Standard Error

As variance [28] is a statistical concept for measuring how far the numbers are spread out and more specifically how far the set of numbers lie from the mean value of that set of numbers. So keeping in view Table 4.6 and Table 4.7 the variance is computed as.

$$
\begin{aligned}
Var(X) &= E(X^2) - [E(X)]^2 \\
&= p - p^2 \\
&= p(1-p) \\
&= p.q
\end{aligned}
\tag{4.5}
$$

For n number of samples, Variance becomes $(p.q)/n$.

After computing variance, the next step is to compute Standard Error which is the estimate of the standard deviation [27] derived from the presented dataset. Standard Error is related to variance in the following manner

$$SE = \sqrt{(p.q)/n} \tag{4.6}$$

Since the requirement is to know the difference between the probabilities of occurrence of a rule in malicious and benign datasets respectively i.e. $\boldsymbol{P}(\boldsymbol{i} = \mathbf{1})$ and $\boldsymbol{Q}(\boldsymbol{i} = \mathbf{1})$ so the Standard Error of [$\boldsymbol{P}(\mathbf{1})$ -$\boldsymbol{Q}(\mathbf{1})$] should be calculated

$$SE\ [P(1)\ - Q(1)] = \sqrt{\frac{p(1).q(1)}{n1} + \frac{p(2).q(2)}{n2}} \tag{4.7}$$

The computed values of Variance and Standard deviation [28] are shown in the table below:

RULES	DIFFERENCE	VARIANCE	STANDARD ERROR
	Ω = P(1)-Q(1)	n1 = 957, n2 = 816	S.E (P1-P2)
1d	0.863363606	0.000146592	0.012107535
1c	0.676770238	0.00027529	0.016591856
20	0.56284575	0.000295396	0.017187094
4	0.543526031	0.000386186	0.019651616
2d	0.43292458	0.000263172	0.016222566
3d	0.43292458	0.000263172	0.016222566
2c	0.409071701	0.000257767	0.016055123
15	0.37909137	0.000258471	0.016077021
5d	0.37252597	0.000289484	0.017014232
14	0.358553845	0.000249562	0.015797539
5c	0.347486016	0.000273412	0.016535177
8	0.329153605	0.000230733	0.015189899
17b	0.235363268	0.000467985	0.021632965
25	0.205671062	0.000172963	0.01315155
13a	0.190992071	0.000239594	0.015478821
27	0.184941453	0.000221596	0.014886118
12a	0.174273158	0.000230046	0.015167284
17d	0.17881016	0.000503811	0.022445735
13b	0.140201457	0.00053226	0.023070772
7c	0.100816737	0.000455214	0.021335753
34	0.07400962	7.42009E-05	0.008613995
3c	0.062838066	7.45575E-05	0.008634666
28a	0.047552093	0.000114196	0.010686269
7b	0.072096472	0.000541704	0.023274534
17c	0.038804628	5.23045E-05	0.007232184
17e	0.03617309	5.77308E-05	0.007598079
31	0.02907754	3.22051E-05	0.00567495
23	0.062204192	0.000510114	0.022585698
28b	0.038793103	0.000126245	0.011235873

17a	0.027490934	3.87298E-05	0.006223326
12b	0.052311912	0.000532403	0.023073868
32	0.028808624	0.000139401	0.011806815
5e	0.012539185	1.29383E-05	0.003596985
1e	0.011494253	1.18727E-05	0.003445673
1b	0.004179728	4.34928E-06	0.002085492
2e	0.004179728	4.34928E-06	0.002085492
3e	0.004179728	4.34928E-06	0.002085492
5b	0.004179728	4.34928E-06	0.002085492
1a	0	0	0
2a	0	0	0
3a	0	0	0
3b	0	0	0
5a	0	0	0
6a	0	0	0
6b	0	0	0
6c	0	0	0
9a	0	0	0
9b	0	0	0
10a	0	0	0
10b	0	0	0
18a	0	0	0
19a	0	0	0
24	0	0	0
26	0	0	0
29a	0	0	0
29b	0	0	0
30a	0	0	0
30b	0	0	0
11	0.01264291	4.28814E-05	0.006548387
2b	0.001044932	1.09074E-06	0.001044386
33	0.004682986	1.3151E-05	0.003626425
16	-0.002992655	1.07285E-05	0.003275439
21b	-0.005263077	1.11239E-05	0.003335252
19d	-0.013841508	2.13919E-05	0.004625138
7a	-0.052012263	6.71206E-05	0.008192716
18b	-0.149444496	0.000194575	0.013949027

Table 13: Values of Variance and Standard Error for all Rules

4.5.3 Computing least Significant Distance using Standard Normal Distribution

Now since the standard error has been computed, the next step is to compute the Least Significant Distance. For calculating the LSD, Gaussian distribution [29], [30] is used. This will be used because in the proposed algorithm the total number of samples is greater than 30. For experiments where the total number of samples is less than 30, Tailor's distribution is used.

Gaussian distribution or Normal distribution is a continuous probability distribution in probability theory. The entire family of normal probability distributions is defined by its mean and its standard deviation

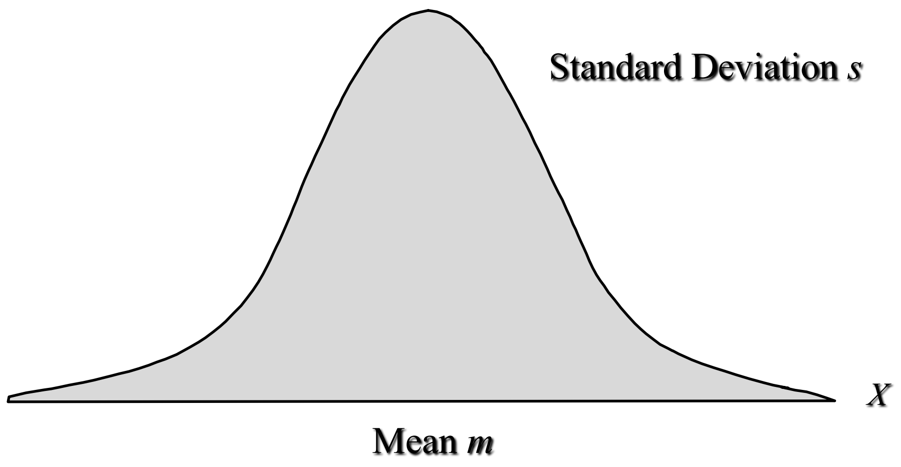


Figure 12: Normal Distribution

The highest point on the Normal Curve is at the mean, which is also the Median and Mode.

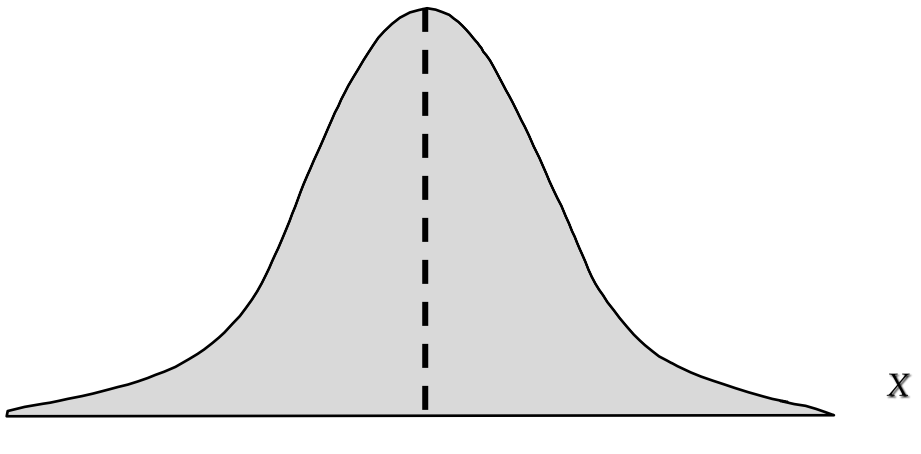


Figure 13: Mean of Normal Distribution

Probabilities for the Random Normal Variable are given by Areas under the curve. The total area under the curve is '1' ('0.5' to the left of mean and '0.5' to the right) as shown in the figure below:

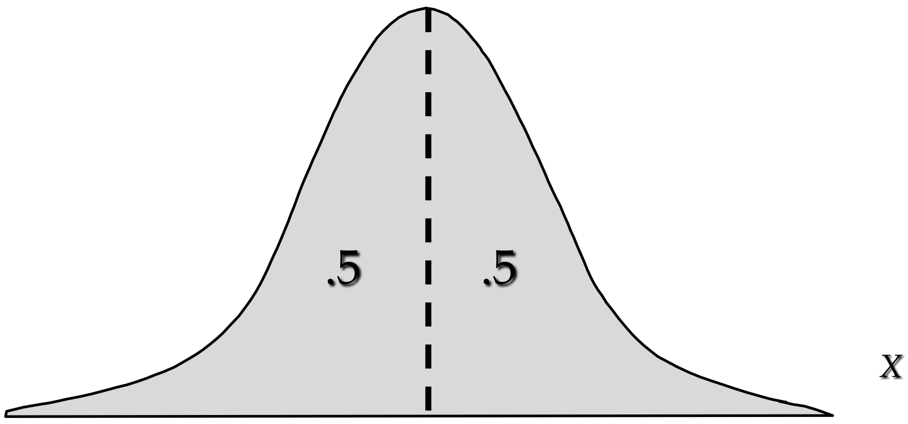


Figure 14: Area under the Gaussian Curve

There are certain characteristics [31] of the standard normal distribution

- 68.26% of values of a Normal Random Variable are within '+/-1 Standard Deviation' of its mean.
- 95.44% of values of a Normal Random Variable are within '+/-2 Standard Deviation' of its mean.
- 99.72% of values of a Normal Random Variable are within '+/-3 Standard Deviation' of its mean.

The characteristics mentioned above are depicted in the figure below:

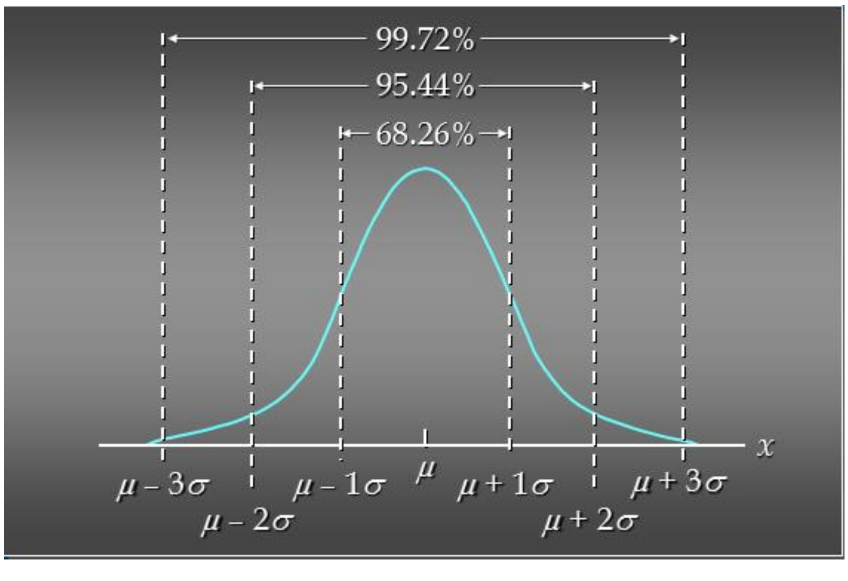


Figure 15: Characteristics of a Normal Distribution

Now, a random variable having a normal distribution with a mean of '0' and a standard deviation of '1' is said to have a Standard Normal Probability Distribution. The letter 'z' is used to designate the Standard Normal random Variable.

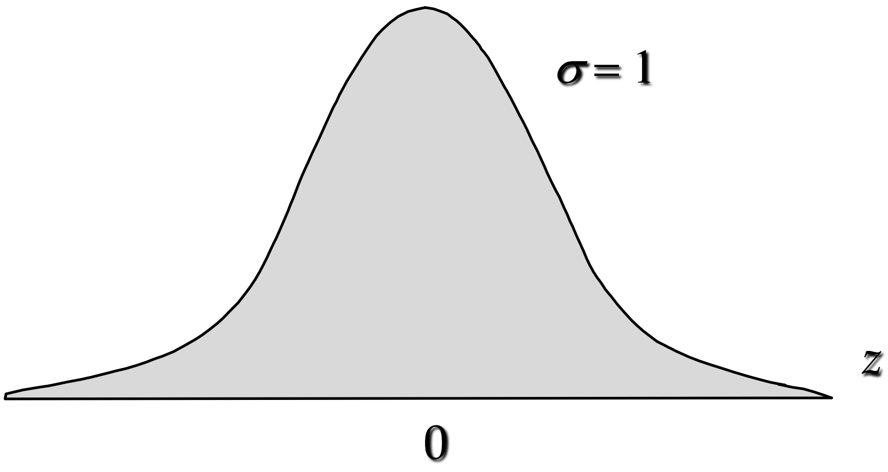


Figure 16: Standard Normal Probability Distribution

Converting to the Standard Normal Distribution

$$z = \frac{\text{x−Mean}}{\text{SD}} \tag{4.8}$$

'z' can be thought of as a measure of the number of Standard Deviations x is from Mean. In this research the Standard Normal Distribution with 5% level of significance has been used as shown in the figure below:

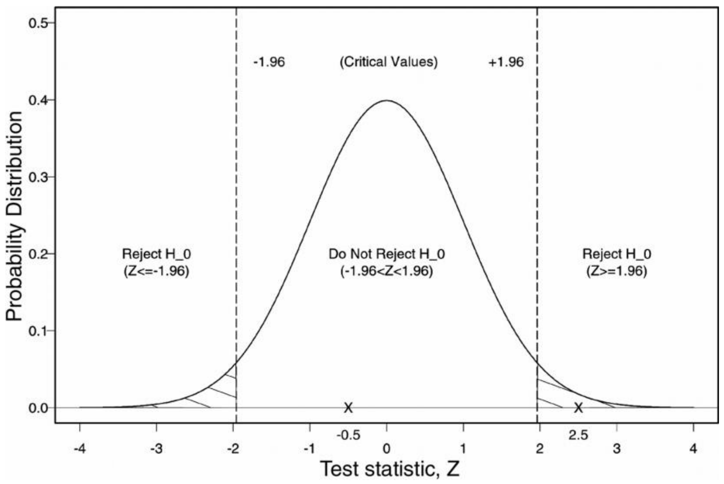


Figure 17: Standard Normal Probability Distribution

The above graph shows the Normal Distribution of the test statistic ‘z’ in a two sided hypothesis test. ‘z’ has a standard normal distribution with a mean of Zero and a variance of 1. The critical values for 5% level of significance are fixed at +1.96 and -1.96. The rejection regions are the areas marked with oblique lines under the two tails of the curve, and they correspond to any test statistic lying either below -1.96 or above +1.96. The value of 1.96 is obtained by looking at the standard graph of Areas under the Standard Normal Curve from 0 to ‘z’ as shown below.

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07
0.0	0.0000	0.0040	0.0080	0.0120	0.0160	0.0199	0.0239	0.0279
0.1	0.0398	0.0438	0.0478	0.0517	0.0557	0.0596	0.0636	0.0675
0.2	0.0793	0.0832	0.0871	0.0910	0.0948	0.0987	0.1026	0.1064
0.3	0.1179	0.1217	0.1255	0.1293	0.1331	0.1368	0.1406	0.1443
0.4	0.1554	0.1591	0.1628	0.1664	0.1700	0.1736	0.1772	0.1808
0.5	0.1915	0.1950	0.1985	0.2019	0.2054	0.2088	0.2123	0.2157
0.6	0.2257	0.2291	0.2324	0.2357	0.2389	0.2422	0.2454	0.2486
0.7	0.2580	0.2611	0.2642	0.2673	0.2704	0.2734	0.2764	0.2794
0.8	0.2881	0.2910	0.2939	0.2967	0.2995	0.3023	0.3051	0.3078
0.9	0.3159	0.3186	0.3212	0.3238	0.3264	0.3289	0.3315	0.3340
1.0	0.3413	0.3438	0.3461	0.3485	0.3508	0.3531	0.3554	0.3577
1.1	0.3643	0.3665	0.3686	0.3708	0.3729	0.3749	0.3770	0.3790
1.2	0.3849	0.3869	0.3888	0.3907	0.3925	0.3944	0.3962	0.3980
1.3	0.4032	0.4049	0.4066	0.4082	0.4099	0.4115	0.4131	0.4147
1.4	0.4192	0.4207	0.4222	0.4236	0.4251	0.4265	0.4279	0.4292
1.5	0.4332	0.4345	0.4357	0.4370	0.4382	0.4394	0.4406	0.4418
1.6	0.4452	0.4463	0.4474	0.4484	0.4495	0.4505	0.4515	0.4525
1.7	0.4554	0.4564	0.4573	0.4582	0.4591	0.4599	0.4608	0.4616
1.8	0.4641	0.4649	0.4656	0.4664	0.4671	0.4678	0.4686	0.4693
1.9	0.4713	0.4719	0.4726	0.4732	0.4738	0.4744	0.4750	0.4756

Table 14: Graph for checking Area under the Standard Normal Curve

Finally the Least Significant Distance is computed by multiplying the Standard Error with 1.96 since Standard Normal Distribution with 5% significance is being used. The computed values of Least Significant Distance are listed in the table below:

RULES	DIFFERENCE	VARIANCE	STANDARD ERROR	LSD
	Ω = P(1)-Q(1)	n1 = 957, n2 = 816	S.E (P1-P2)	LSD (p1-p2)
1d	0.863363606	0.000146592	0.012107535	0.023730769
1c	0.676770238	0.00027529	0.016591856	0.032520038
20	0.56284575	0.000295396	0.017187094	0.033686703
4	0.543526031	0.000386186	0.019651616	0.038517168
2d	0.43292458	0.000263172	0.016222566	0.03179623
3d	0.43292458	0.000263172	0.016222566	0.03179623
2c	0.409071701	0.000257767	0.016055123	0.031468042
15	0.37909137	0.000258471	0.016077021	0.03151096
5d	0.37252597	0.000289484	0.017014232	0.033347894
14	0.358553845	0.000249562	0.015797539	0.030963176
5c	0.347486016	0.000273412	0.016535177	0.032408946
8	0.329153605	0.000230733	0.015189899	0.029772202
17b	0.235363268	0.000467985	0.021632965	0.042400612
25	0.205671062	0.000172963	0.01315155	0.025777039
13a	0.190992071	0.000239594	0.015478821	0.03033849
27	0.184941453	0.000221596	0.014886118	0.02917679
12a	0.174273158	0.000230046	0.015167284	0.029727876
17d	0.17881016	0.000503811	0.022445735	0.04399364
13b	0.140201457	0.00053226	0.023070772	0.045218712
7c	0.100816737	0.000455214	0.021335753	0.041818076
34	0.07400962	7.42009E-05	0.008613995	0.016883431
3c	0.062838066	7.45575E-05	0.008634666	0.016923945
28a	0.047552093	0.000114196	0.010686269	0.020945087
7b	0.072096472	0.000541704	0.023274534	0.045618088
17c	0.038804628	5.23045E-05	0.007232184	0.014175081
17e	0.03617309	5.77308E-05	0.007598079	0.014892235
31	0.02907754	3.22051E-05	0.00567495	0.011122903
23	0.062204192	0.000510114	0.022585698	0.044267968
28b	0.038793103	0.000126245	0.011235873	0.022022311
17a	0.027490934	3.87298E-05	0.006223326	0.012197719
12b	0.052311912	0.000532403	0.023073868	0.045224781
32	0.028808624	0.000139401	0.011806815	0.023141356
5e	0.012539185	1.29383E-05	0.003596985	0.007050091
1e	0.011494253	1.18727E-05	0.003445673	0.006753518
1b	0.004179728	4.34928E-06	0.002085492	0.004087564
2e	0.004179728	4.34928E-06	0.002085492	0.004087564

3e	0.004179728	4.34928E-06	0.002085492	0.004087564
5b	0.004179728	4.34928E-06	0.002085492	0.004087564
1a	0	0	0	0
2a	0	0	0	0
3a	0	0	0	0
3b	0	0	0	0
5a	0	0	0	0
6a	0	0	0	0
6b	0	0	0	0
6c	0	0	0	0
9a	0	0	0	0
9b	0	0	0	0
10a	0	0	0	0
10b	0	0	0	0
18a	0	0	0	0
19a	0	0	0	0
24	0	0	0	0
26	0	0	0	0
29a	0	0	0	0
29b	0	0	0	0
30a	0	0	0	0
30b	0	0	0	0
11	0.01264291	4.28814E-05	0.006548387	0.012834839
2b	0.001044932	1.09074E-06	0.001044386	0.002046997
33	0.004682986	1.3151E-05	0.003626425	0.007107793
16	-0.002992655	1.07285E-05	0.003275439	0.00641986
21b	-0.005263077	1.11239E-05	0.003335252	0.006537093
19d	-0.013841508	2.13919E-05	0.004625138	0.009065271
7a	-0.052012263	6.71206E-05	0.008192716	0.016057724
18b	-0.149444496	0.000194575	0.013949027	0.027340092

Table 15: Values of L.S.D computed using Standard Normal Distribution

4.5.4 Selection of Rules

After the computation of Least Significant Difference between the probability of occurrence of a rule between malicious and benign applications a threshold should be defined in order to select some specific rules. The is set such that if the arithmetic difference of the probability of occurrence of a rule in malicious and benign is less than the Least Significant Difference of the probability of occurrence of a rule in malicious and benign dataset, the rule is rejected, otherwise accepted. The process is shown in the form of pseudo code as shown below.

```
If (Arithmetic Difference of P(i) and Q(i) > L.S.D of P(i) and Q(i))
        Accept Rule
Else
        Reject Rule
```

The computed values along with the decision of acceptance and rejection are illustrated in the table below.

RULES	STANDARD ERROR	LSD	DECISION	ACCEPTANCE DECISION
	S.E (P1-P2)	LSD (p1-p2)	whether Ω > LSD	
1d	0.012107535	0.023730769	0.839632837	Accept
1c	0.016591856	0.032520038	0.6442502	Accept
20	0.017187094	0.033686703	0.529159046	Accept
4	0.019651616	0.038517168	0.505008863	Accept
2d	0.016222566	0.03179623	0.401128351	Accept
3d	0.016222566	0.03179623	0.401128351	Accept
2c	0.016055123	0.031468042	0.377603659	Accept
15	0.016077021	0.03151096	0.34758041	Accept
5d	0.017014232	0.033347894	0.339178076	Accept
14	0.015797539	0.030963176	0.327590669	Accept
5c	0.016535177	0.032408946	0.31507707	Accept
8	0.015189899	0.029772202	0.299381403	Accept
17b	0.021632965	0.042400612	0.192962656	Accept
25	0.01315155	0.025777039	0.179894023	Accept
13a	0.015478821	0.03033849	0.160653581	Accept
27	0.014886118	0.02917679	0.155764663	Accept
12a	0.015167284	0.029727876	0.144545282	Accept
17d	0.022445735	0.04399364	0.13481652	Accept
13b	0.023070772	0.045218712	0.094982745	Accept
7c	0.021335753	0.041818076	0.058998662	Accept
34	0.008613995	0.016883431	0.057126189	Accept
3c	0.008634666	0.016923945	0.045914122	Accept
28a	0.010686269	0.020945087	0.026607006	Accept
7b	0.023274534	0.045618088	0.026478384	Accept
17c	0.007232184	0.014175081	0.024629548	Accept
17e	0.007598079	0.014892235	0.021280855	Accept
31	0.00567495	0.011122903	0.017954638	Accept

23	0.022585698	0.044267968	0.017936224	Accept
28b	0.011235873	0.022022311	0.016770792	Accept
17a	0.006223326	0.012197719	0.015293215	Accept
12b	0.023073868	0.045224781	0.007087131	Accept
32	0.011806815	0.023141356	0.005667267	Accept
5e	0.003596985	0.007050091	0.005489094	Accept
1e	0.003445673	0.006753518	0.004740735	Accept
1b	0.002085492	0.004087564	9.21639E-05	Accept
2e	0.002085492	0.004087564	9.21639E-05	Accept
3e	0.002085492	0.004087564	9.21639E-05	Accept
5b	0.002085492	0.004087564	9.21639E-05	Accept
1a	0	0	0	Reject
2a	0	0	0	Reject
3a	0	0	0	Reject
3b	0	0	0	Reject
5a	0	0	0	Reject
6a	0	0	0	Reject
6b	0	0	0	Reject
6c	0	0	0	Reject
9a	0	0	0	Reject
9b	0	0	0	Reject
10a	0	0	0	Reject
10b	0	0	0	Reject
18a	0	0	0	Reject
19a	0	0	0	Reject
24	0	0	0	Reject
26	0	0	0	Reject
29a	0	0	0	Reject
29b	0	0	0	Reject
30a	0	0	0	Reject
30b	0	0	0	Reject
11	0.006548387	0.012834839	-0.000191929	Reject
2b	0.001044386	0.002046997	-0.001002064	Reject
33	0.003626425	0.007107793	-0.002424807	Reject
16	0.003275439	0.00641986	-0.009412515	Reject
21b	0.003335252	0.006537093	-0.01180017	Reject
19d	0.004625138	0.009065271	-0.022906779	Reject
7a	0.008192716	0.016057724	-0.068069987	Reject
18b	0.013949027	0.027340092	-0.176784588	Reject

Table 16: Accepted and Rejected Rules

4.5.5 Score Assignment to Selected Rules

After the selection of rules which are to be used in malware detector, the next step is to assign scores to each of the selected rules [32]. The approach followed for assigning weightage depends on the arithmetic difference of P(1) and Q(1).

It is assumed that the highest score that can be assigned to any rule be '100' and the lowest score for any rule can very clearly be '0'. Let Rule X be a hypothetic rule with a score 100 because for Rule X

- P(1) = 1 and Q(1) = 0

The possibility of having a Rule X satisfying the above criteria is very difficult because for this to satisfy the rule should be present in all malicious samples and in no benign samples.

Now, the rule will have '0' or no score when it would satisfy the below conditions

- P(1) = 0 and Q(1) = 0
- P(1) = Q(1)

Since the above technique solely depends upon the difference in the probability of occurrence of a rule in malicious and benign datasets, so all rules have been assigned weights that equal the difference in P(1) and Q(1). All the selected rules that have been assigned weights using the above mentioned technique are shown in the table below.

RULES	DIFFERENCE	LSD	DECISION	ACCEPTANCE DECISION	WEIGHTS
	Ω = P(1)-Q(1)	LSD (p1-p2)	whether Ω > LSD		
1d	0.863363606	0.023730769	0.839632837	Accept	86
1c	0.676770238	0.032520038	0.6442502	Accept	68
20	0.56284575	0.033686703	0.529159046	Accept	56
4	0.543526031	0.038517168	0.505008863	Accept	54
2d	0.43292458	0.03179623	0.401128351	Accept	43
3d	0.43292458	0.03179623	0.401128351	Accept	43
2c	0.409071701	0.031468042	0.377603659	Accept	41
15	0.37909137	0.03151096	0.34758041	Accept	38

5d	0.37252597	0.033347894	0.339178076	Accept	37
14	0.358553845	0.030963176	0.327590669	Accept	36
5c	0.347486016	0.032408946	0.31507707	Accept	35
8	0.329153605	0.029772202	0.299381403	Accept	33
17b	0.235363268	0.042400612	0.192962656	Accept	24
25	0.205671062	0.025777039	0.179894023	Accept	21
13a	0.190992071	0.03033849	0.160653581	Accept	19
27	0.184941453	0.02917679	0.155764663	Accept	18
12a	0.174273158	0.029727876	0.144545282	Accept	17
17d	0.17881016	0.04399364	0.13481652	Accept	18
13b	0.140201457	0.045218712	0.094982745	Accept	14
7c	0.100816737	0.041818076	0.058998662	Accept	10
34	0.07400962	0.016883431	0.057126189	Accept	7
3c	0.062838066	0.016923945	0.045914122	Accept	6
28a	0.047552093	0.020945087	0.026607006	Accept	5
7b	0.072096472	0.045618088	0.026478384	Accept	7
17c	0.038804628	0.014175081	0.024629548	Accept	4
17e	0.03617309	0.014892235	0.021280855	Accept	4
31	0.02907754	0.011122903	0.017954638	Accept	3
23	0.062204192	0.044267968	0.017936224	Accept	6
28b	0.038793103	0.022022311	0.016770792	Accept	4
17a	0.027490934	0.012197719	0.015293215	Accept	3
12b	0.052311912	0.045224781	0.007087131	Accept	5
32	0.028808624	0.023141356	0.005667267	Accept	3
5e	0.012539185	0.007050091	0.005489094	Accept	1
1e	0.011494253	0.006753518	0.004740735	Accept	1
1b	0.004179728	0.004087564	9.21639E-05	Accept	1
2e	0.004179728	0.004087564	9.21639E-05	Accept	1
3e	0.004179728	0.004087564	9.21639E-05	Accept	1
5b	0.004179728	0.004087564	9.21639E-05	Accept	1

Table 17: Accepted Rules with respective Scores

4.6 Computing Overall Malicious Score

After assigning individual scores to all rules, the same dataset is used for computing over all malicious score of all these samples [33], both malicious and benign. For this purpose a script has been written in python that checks the presence of all rules and computes overall score by adding the respective scores of all those rules which are found in the sample.

Let Ri be a rule that has been found in the sample and Si be its respective score as mentioned in Table 4.13. The script runs for all the malicious dataset and fills in the database such that the total malicious score 'Ś' of a sample is

$$Ś = \sum Si \tag{4.9}$$

4.6.1 Malicious Score for Malware Dataset

The same dataset of 957 malware samples was again scanned in order to compute the total malicious score of each sample. After executing the python script and checking the results by analysis, it was calculated the average malicious score for malware dataset was '398.4252874'.

$$\mu = (\sum Si)/N \tag{4.10}$$

$$\mu = 381293/957$$

$$\mu = 398.4252874$$

Where 'N' is the total number of malicious applications scanned.

4.6.2 Malicious Score for Benign Dataset

The same dataset of 816 benign samples was again scanned in order to compute the total malicious score of each sample. After executing the python script and checking the results by analysis, it was calculated the average malicious score for malware dataset was '63.1764705'.

$$\mu = (\sum Si)/M \tag{4.11}$$

$$\mu = 51552/816$$

$$\mu = 63.1764705'$$

Where 'M' is the total number of benign applications scanned.

So it has been calculated that the proposed algorithm on the basis of its Rules produces very acceptable results. The mean of overall score for malicious dataset is far greater than the mean of benign dataset's malware score.

4.6.3 Setting the Malicious Score Threshold

Since the malicious scores for all samples in the benign and malicious datasets has been computed and the average malicious score for malware dataset has been found out to be more than 6 times the average malicious score for benign dataset. Now, there should be a threshold such that if malicious score of an application is greater than that threshold score, it is considered to be malicious, else benign.

For this purpose the ‘Percentile’ approach is followed. Firstly, the value in malicious dataset is found that has a percentile value of 5 such that just 5% values are lesser then this value.

Let ‘N’ be the total samples in the malware dataset. Such that

$$N = 957$$

$$P_{05} = \frac{5}{100}.(957 + 1) \qquad (4.12)$$

$$= 48$$

It means that the 48th value has the percentile value of 5. So the Value is 156 that mean that just 5% values of score in the malware dataset are less than 156.

Now the same approach is followed for benign dataset but with 95 percentile value. A value which is greater than 95% of all scores in the benign dataset is found out. The following graph shows that scores of malware dataset along with the percentile value of 5. 95% values are greater than the value 156.

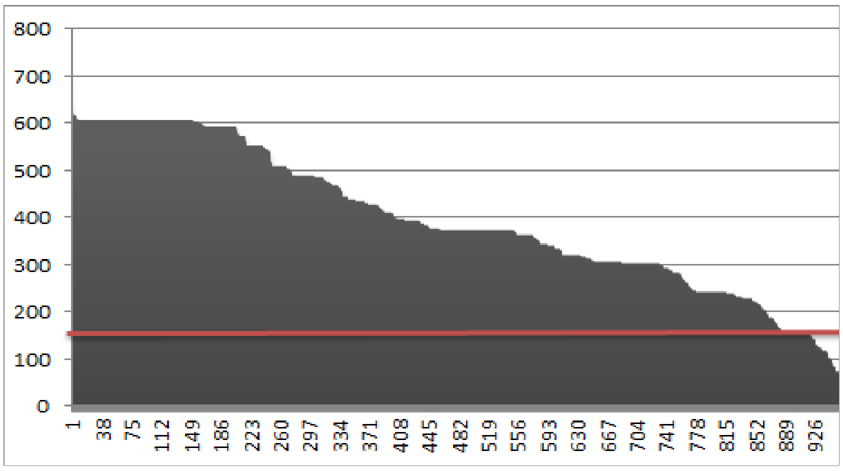


Figure 18: Malware Dataset with a percentile value of 5

Let ‘M’ be the total samples in the malware dataset. Such that

$$M = 816$$

$$Q_{95} = \frac{95}{100}.(816+1) \qquad (4.13)$$

$$= 776$$

This means that in the benign dataset the 776th value has a percentile value of 95. After looking at the benign dataset the value is found out to be 198.

The following graph shows that scores of benign dataset along with the percentile value of 95. Only 5% values are greater than the value 198.

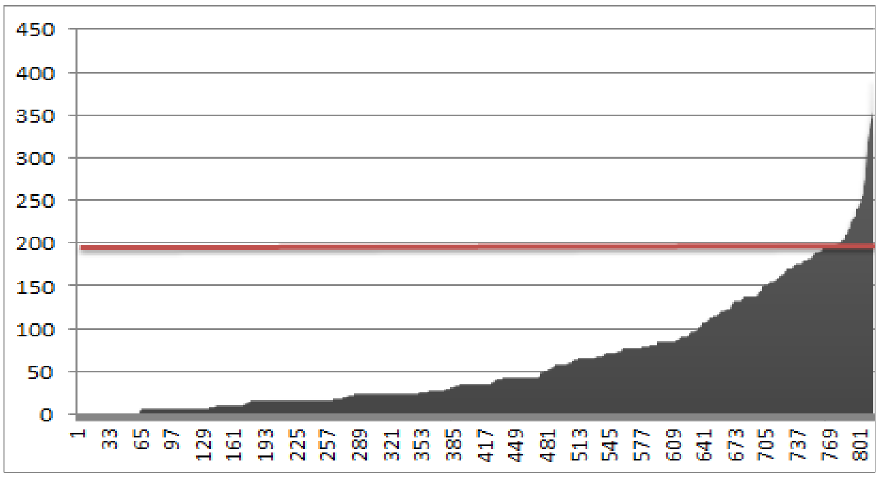


Figure 19: Benign Dataset with a percentile value of 95

Now a range is acquired such that the threshold value has to be between P_{05} and Q_{95}. The threshold value is set to P_{05} such that the false positive value of 5% is tolerated keeping an eye the current dataset. So the threshold value is 156.

4.7 Chapter Summary

This chapter discussed in detail the proposed Algorithm for detecting malicious Android applications. Firstly, a set of rules is proposed which after utilizing Data mining approaches gets filtered and selected rules are used in the anti-malware algorithm. Each rule is assigned a weightage and both the malicious and benign dataset are scanned for the presence of these rules. The mean of overall score for malicious and benign datasets is computed and appreciable results are acquired. Finally the threshold score for deciding whether an application is benign or malicious is computed.

C h a p t e r 5

TESTING

5.1 Chapter Overview

This chapter contains information about the performance of proposed algorithm after testing it on the test dataset containing both malicious and benign applications. The test results are computed and performance graphs are computed. The accuracy along with True Negative and False Positive ratios are also calculated.

5.2 Test Dataset

The dataset for testing comprised of 246 malicious and 768 benign applications. Malicious applications were acquired from Gnome android malware project and contagion while the benign applications were acquired from Google Play. The dataset was tested with the algorithm and the malicious score for each sample (both malicious and benign) was computed. The value which acts as a threshold for deciding whether a sample is malicious is set to be 156.

5.3 Testing on Malware Dataset

A set of 246 malicious applications were scanned with the proposed anti-malware algorithm. A python script checks each application for rule presence and respective scores for each rule is added to compute the overall malicious score. The graph below shows the values of malicious score of each application.

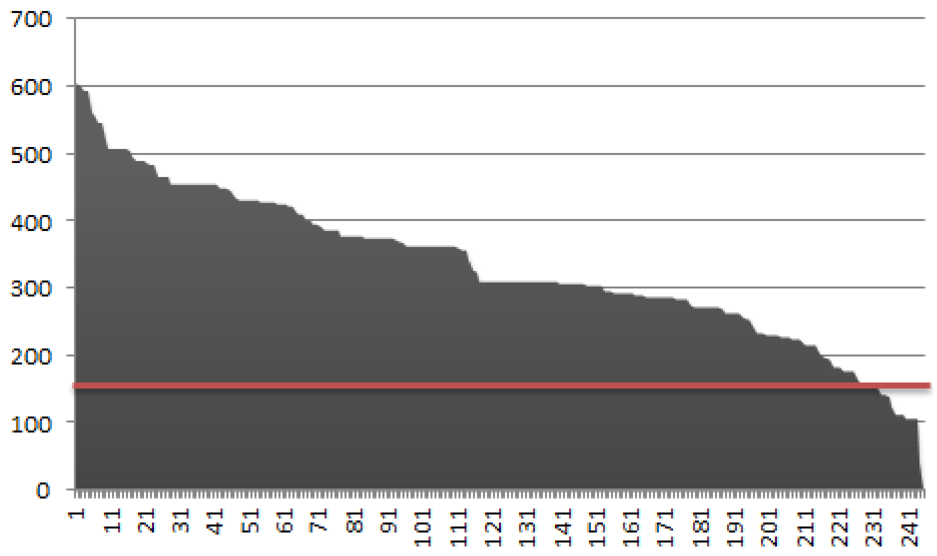


Figure 20: Scores of Malicious Test Dataset

The above graph shoes that there are some values which are below the threshold which means that there will be some value of True Negatives as well. Let ‘N’ be the total number of samples and N_0 be the number of samples with a malicious score less than 156.

$$\begin{aligned} TP &= N - N_0 \qquad (5.1) \\ &= 246 - 14 \\ &= 232 \end{aligned}$$

5.4 Testing on Benign Dataset

A set of 768 benign applications were scanned with the proposed anti-malware algorithm. A python script checks each application for rule presence and respective scores for each rule is added to compute the overall malicious score. The graph below shows the values of malicious score of each application.

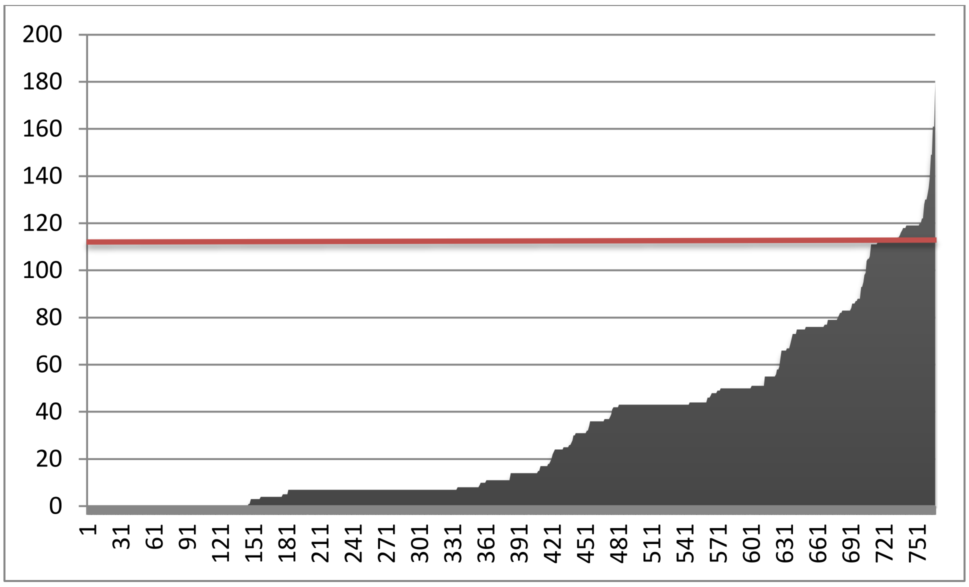


Figure 21: Scores of Benign Test Dataset

The above graph shoes that there are some values which are above the threshold which means that there will be some value of False Positives as well. Let 'M' be the total number of samples and M_0 be the number of samples with a malicious score greater than 156.

$$TN = N - N_0 \qquad (5.2)$$
$$= 768 – 3$$
$$= 765$$

5.5 Algorithm performance and Efficiency

The efficiency and performance of the proposed algorithm has also been computed by calculating certain metrics like the TP ratio, FP ratio, Accuracy, Specificity and Sensitivity.

5.5.1 True Positive Rate

True positive ratio [34] or sensitivity of the algorithm has been computed using the following formula

$$TPR = \frac{TP}{P} = TP/(TP + FN) \qquad (5.3)$$

Putting in the values,

$$TPR = 232/(232 + 14)$$

$$TPR = 0.943$$

$$TPR = Sensitivity = 94.3\%$$

5.5.2 False Positive Rate

False positive ratio [35] of the algorithm has been computed using the following formula

$$FPR = \frac{FP}{N} = FP/(FP + TN) \tag{5.4}$$

$$FPR = 3/(3 + 765)$$

$$= 0.00395$$

$$= 0.395\%$$

5.5.3 Accuracy

The accuracy of the proposed Algorithm has been computed using the following formula

$$Accuracy = (TP + TN)/(P + N) \tag{5.5}$$

$$Accuracy = (232 + 765)/(246 + 768)$$

$$Accuracy = 0.9832$$

$$Accuracy = 98.32\%$$

5.5.4 Specificity

The specificity of the proposed Algorithm has been computed using the following formula

$$Specificity = TN/N \tag{5.6}$$

$$Specificity = TN/(FP + TN)$$

$$Specificity = 765/(3 + 765)$$

$$Specificity = 99.6\%$$

Chapter 6

CONCLUSION AND FUTURE DIRECTIONS

6.1 Chapter Overview

This chapter provides overview of the carried research, objectives achieved, limitations of proposed solution and future directions.

6.2 Research Overview

In this thesis, a novel anti-malware algorithm for Android applications has been proposed. The proposed algorithm has been made after analyzing more than 1600 Android applications, both benign and malicious and it is producing highly accurate and acceptable results. The thesis has explained the proposed anti-malware algorithm for Android applications in a chronological order. A brief introduction to Android malware, its background and applications, unresolved problems and research objectives have been presented in Chapter 1. Chapter 2 presented various state-of the art algorithms for detecting malicious Android applications and their categories. Chapter 3 presented the details of dataset used for experimentation and validation of proposed algorithm. Chapter 4 presented the proposed anti-malware algorithm. Chapter 5 evaluated the performance of proposed algorithm on almost 1000 samples. The proposed algorithm efficiently detected malicious applications with an accuracy of 98.32% at a very low computational cost, thus making it suitable for real-time applications.

6.3 Objectives Achieved

This research produced a novel anti-malware algorithm for detecting malicious Android applications. The Algorithm looks for certain features in the application and observe the presence of malicious rules in the application being tested. Since each rule’s presence carries a score, the overall malicious score of an application is computed by adding all weights of all the rules which are found. The selection of rules is done by applying statistical approaches and using the standard normal distribution. Finally, the experimental results show efficient results as very high accuracy has been achieved with an extremely low computational complexity, this making the algorithm suitable for real-time applications.

6.4 Limitations

The proposed algorithm works well for any kind of android malware samples so far. It has been designed to detect zero day exploits and malwares but still there is a room for improvement. The algorithm does not specifically detect the re-packaging [36] of applications which is a very prevalent feature of malicious applications. There is also a need for enhancing the detection of malicious features in the application. Furthermore, the proposed application provides no statistical justification for the selection of features which are looked for in the application. The proposed algorithm takes the extracted files of an Android application as input and not the actual Android packaged file as in other algorithms for the same purpose [37].

6.5 Future Directions

To further increase and enhance the True positive rate there is a need to improve further, the selection and augmentation of malicious features. This would help in efficiently detecting the zero day exploits which are being used in newer versions of Android malware. Another appreciable work would be to make an Android application for this purpose which would check the applications being installed on the smartphone. Another direction would be to regular change and update the application for newer versions of Android operating system. A web portal can also be made that would ask researchers to upload the Android application that needs to be tested.

REFERENCES

[1] Francesco Di Cerbo, Andrea Girardello, Florian Michahelles and Svetlana Voronkova, “Detection of Malicious Applications on Android OS,” in IWCF 2012, LNCS, pp138-149, 2011.

[2] Stefan Brahler, “Analysis of the Android Architecture”, Karlsruhe institute for technology,” , pp. 4-1, October 2010.

[3] Vadodil Joel Varghese, Stuart Walker, "Dissecting Andro Malware," in *SANS Institute InfoSec Reading Room*, University of Essex, UK, 2011.

[4] L.-K. Yan and H. Yin, “Droidscope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis,” in Proceedings of the 21st USENIX Security Symposium, 2012.

[5] Johnson, R.,Wang, Z., Gagnon, C., Stavrou, A.: Analysis android applications permissions. In: Proceedings of the 6th International Conference on Software Security and Reliability. (2012)

[6] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, “Privilege Escalation Attacks on Android,” in *Proceedings of the 13th Information Security Conference* (ISC ’10), Oct. 2010.

[7] Fake Instagram Android Application spreads malware http://nakedsecurity.sophos.com/2012/04/18/fake-instagram-app-android-malwar/

[8] Malware on the Rise, TrustGo and Lookout http://securitywatch.pcmag.com/none/308184-trustgo-and-lookout-top-android-mobile-security-test

[9] Rise in Android Malware, Report by McAfeehttp://www.redmondpie.com/mcafee-mobile-malware-increased-by-700-over-2011-mostly-targeting-android/

[10] Android SDK, http://developer.android.com/sdk/index.html

[11] Android-apktool, http://code.google.com/p/android-apktool/

[12] Dex2jar, Tools to work with android .dex and java .class files http://code.google.com/p/dex2jar/

[13] JD-GUI, http://java.decompiler.free.fr/?q=jdgui

[14] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified" Technical Report UCB/EECS-2011-48, University of California, Berkeley, May 2011.

[15] Stowaway: Web Portal, http://www.android-permissions.org/

[16] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android" in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011)*, June 2011.

[17] M. Conti, V. T. N. Nguyen, and B. Crispo, "CRePE: Context-related policy enforcement for Android" in *Proceedings of the Thirteen Information Security Conference (ISC '10)*, Boca Raton, FL, Oct. 2010.

[18] Zhou,Y., and Jiang, X, "Dissecting android malware: Characterization and evolution" in *Proceedings of the 33rd IEEE Symposium on Security and Privacy (Oakland 2012)*, San Francisco, CA, USA, May 2012.

[19] Enck, W., Ongtang, M., and Mcdaniel, P., "On Lightweight Mobile Phone Application Certification" in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, November 2009.

[20] Androguard, http://code.google.com/p/androguard/

[21] Zhou,Y., and Jiang, X, "Dissecting android malware: Characterization and evolution" in *Proceedings of the 33rd IEEE Symposium on Security and Privacy (Oakland 2012)*, San Francisco, CA, USA, May 2012.

[22] http://contagiodump.blogspot.com/search?q=Android

[23] Blount, J.J, Tauritz, D.R, Mulder, S.A, "Adaptive Rule-Based Malware Detection Employing Learning Classifier Systems: A Proof of Concept," in *35th Annual Computer Software and Applications Conference Workshops*, pp. 110-115, 18-22 July 2011.

[24] Damiano Varagnolo, Gianluigi Pillonetto, and Luca Schenato, "Distributed statistical estimation of the number of nodes in Sensor Networks" in *IEEE Conference on Decision and Control*, pages 1498-1503, Atlanta, USA, December 2010.

[25] Siegel, A., "Toward a Usable Theory of Chernoff Bounds for Heterogeneuos and Partially Dependent Random Variables," in *manuscript*, New York University, 1992.

[26] Morganstein, D. R., and Brick, J. M., "WesVarPC: Software for computing variance estimates from complex designs" in *Proceedings of the Bureau of the Census 1996 Annual Research Conference*, pp. 861-866. Washington, DC: Bureau of the Census.

[27] Ahn, S. and Fessler, A., "Standard Errors of Mean, Variance, and Standard Deviation Estimators" in *Technical Report*, Ann Arbor, MI, USA, EECS Department, University of Michigan, July 2003.

[28] Giles, D. E. A., "Calculating a standard error for the Gini coefficient: Some

further results" in *Oxford Bulletin of Economics and Statistics*, 66(3), pp. 425-433.

[29] Aludaat, K.M. and Alodat, M.T., "A note on approximating the normal distribution function" in *Applied Mathematical Sciences*, Vol 2, no 9, pp 425-429.

[30] Bowling, S. R., Khasawneh, M. T. , Kaewkuekool, S.,Cho, B. R., "A logistic approximation to the cumulative normal distribution" in *Journal of Industrial Engineering and Management*, vol. 2, no. 1, pp. 114-127, 2009.

[31] Balakrishnan, N. and Malik, H.J., "Means, Variances and covariances of logistic order statistics for sample size up to fifty", in *J. Statist. Plann. Inf*, Vol 13, pp. 117-129.

[32] H. Khan, F. Mirza, and S. A. Khayam, "Determining Malicious Executable Distinguishing Attributes and Low-Complexity Detection," in *Springer Journal in Computer Virology (JCV)*, vol. 7, no. 2, pp. 95-105, January 2010.

[33] Anderson, B., Quist, D., Neil, J., Storlie, C. and Lane, T., "Graph-Based Malware Detection using Dynamic Analysis" in *Journal in Computer Virology 7*, pp 247-258.

[34] Wen Zhu, Nancy Zeng, Ning Wang, "Sensitivity, Specificity, Accuracy, Associated Confidence Interval and ROC Analysis with Practical SAS Implementations" in *Proceedings of the SAS Conference*, pp 9, 2010.Baltimore, Maryland.

[35] A. Foss and O. R. Zaıane, "A hybrid classification and clustering approach for medical diagnostics and other high dimensional data" in *Technical Report TR08-15*, University of Alberta, 2008.

[36] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, “DroidMOSS: Detecting Repackaged Smartphone Applications in Third-Party AndroidMarketplaces” in *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy, CODASPY’ 12,* 2012.

[37] Baker, B.S., Baker, B.S., “Parameterized duplication in strings: Algorithms and an application to software maintenance” in , pp. 1343-1362, 1997.