MALWARE DETECTION USING MACHINE LEARNING FOR ANDROID FILES

PROJECT REPORT

Submitted by

BAASKAR R	**18GH04**
GURUPRASANTH SJ	**18GH07**
KISOREGUPTHAA R	**17DG15**
SATHISH KUMAR AP	**18GH15**

Under the guidance of
DR.S.SHARMILA

In partial fulfillment of the requirement for the award of

DIPLOMA IN ELECTRONICS AND COMMUNICATION

STATE BOARD OF TECHNICAL EDUCATION

GOVERNMENT OF TAMILNADU

JUNE 2020

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

PSG POLYTECHNIC COLLEGE

(Autonomous and an ISO 9001 certified Institution)

COIMBATORE – 641 004

PSG POLYTECHNIC COLLEGE

(Autonomous and an ISO 9001 certified Institution)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

COIMBATORE – 641 004

CERTIFICATE

Name **SATHISH KUMAR A P** **Register Number** **18GH15**

This is to certify that the Project report entitled

MALWARE DETECTION USING MACHINE LEARNING FOR ANDROID FILES

has been submitted by

SATHISH KUMAR A P

In partial fulfillment for the award of

DIPLOMA IN ELECTRONICS AND COMMUNICATION

of the State Board of Technical Education,

Government of Tamil Nadu.

during the academic year **2019-2020**

Faculty guide Head of the Department

Certified that the candidate was examined by us in the Project work viva-voce examination held on ………………….

Internal Examiner External Examiner

ACKNOWLEDGEMENT

We are ineffably indebted to our Principal DR.B.GIRIRAJ for giving us this opportunity and encouraging us to accomplish this project.

We thank Ms.Dr.S.SHARMILA, for her valuable guidance and constant supervision. Without her able guidance, this project would not have been possible and we shall eternally be grateful to her for her assistance.

We acknowledge with deep sense of reverence, our special gratitude towards our Head in-charge Ms.K.LAKSHMIPRABHA, Department of ELECTRONICS AND COMMUNICATION for his guidance, inspiration and suggestions in our quest for knowledge.

We would like to express our special gratitude and thanks to ELECTRONICS PRODUCT DEVELOPMENT CENTRE and technicians for giving us such attention and time.

ABSTRACT

Malware can be defined as any type of malicious code that has the potential to harm a computer or a network. In recent years, malware analysis has acquired great importance positioning itself as a strategic and promising research field. As Android platform is the most used operating system by common people the malware developers turned their head towards this platform. Hence the threat for the privacy of common people has increased. Signature based malware detection, which is the most common technique adopted by commercial anti-malwares for mobiles is often ineffective. Moreover, it is costly, as the process for obtaining and classifying a malware signature is more laborious and time-consuming. It also cannot find any new or polymorphic malwares. Hence heuristic based approach is introduced. This approach uses machine learning algorithms to find even new and polymorphic malwares as it finds patterns between the malwares rather than its signature. In this work we have developed an algorithm for malware detection for APK files based on machine learning algorithms. APK files are archive files used especially in android platform. Algorithm is written in Python programming language version three. Three classifiers are used with different priorities, they are Random forest algorithm, Decision tree and Logistic regression algorithm. By prioritising the classifiers, the classifier is made more sensitive to the give positive output. Thus, reducing the false positive rate without much compromise in accuracy of the detector. The dataset of 248 samples of which 131 are benign files and 117 are malwares were collected. Totally nineteen features vectors are used to distinguish the malware and benign files. All these nineteen features are given equal weights. During testing the algorithm showed the maximum accuracy of 97% with test train split of 1:4. Finally the algorithm is implemented in a hardware called PYNQ Z2 to increase the performance and reliability of the algorithm. Hardware implementation makes difficult to reverse engineer the code and reduces the burden of main processors of the mobile phones.

TABLE OF CONTENT

S.NO	CHAPTER	PG.NO
	FRONT PAGE	**I**
	CERTIFICATE PAGE	**II**
	ACKNOWLEDGEMENT	**III**
	ABSTRACT	**IV**
	TABLE OF CONTENT	**V**
	LIST OF TABLES	**IX**
	LIST OF FIGURES	**X**
1	**INTRODUCTION……………………………………………………..**	**01**
	1.1 Introduction to tri-classifier malware detector	01
	1.1.1 Problem definition	01
	1.1.2 Objectives	01
	1.2 Existing systems	02
	1.3 Proposed system	02
	1.4 Concluding remarks	03
2	**LITERATURE SURVEY…………………………………………......**	**04**
	2.1 Introduction	04
	2.2 Related studies	04
	2.3 Comparison between different studies	05
	2.4 Concluding remarks	06
3	**CONCEPTS REGARDING MALWARE DETECTION……….. …**	**07**
	3.1 Introduction	07
	3.2 Malwares	07
	3.2.1 Virus	08

3.2.2 Worm 08

3.2.3 Trojon horse 08

3.2.4 Adware 08

3.2.5 Spyware 08

3.2.6 Rootkit 08

3.2.7 Backdoor 09

3.2.8 Keylogger 09

3.2.9 Ransomware 09

3.3 Malwares in android platform 09

3.4 APK file 10

3.5 Detection Methodologies 11

3.5.1 Signature-Based detection 12

3.5.2 Using machine learning 12

3.6 Need for hardware implementation 14

3.7 Conclusion 14

4 SOFTWARE IMPLEMENTATION……………………………….. 15

4.1 Introduction 15

4.2 Training dataset 15

4.3 Feature extraction 16

4.3.1 Andropytool 17

4.3.2 Implementation 17

4.4 Feature selection 19

4.4.1 Python 20

4.4.2 Implementation 20

4.5 Classicification 21

4.5.1 Classification strategy 21

4.5.2 Implementation 22

4.6 Interaction with the program 22

4.7 Concluding remarks 23

5 FEATURE SELECTION…..…………………………………….. 24

5.1 Introduction 24

5.2 Permissions 24

5.3 Opcodes 26

5.4 API calls 27

5.5 API packages 28

5.6 Intents 28

5.7 Concluding remarks 29

6 HARDWARE IMPLEMENTATION……………………………… 30

6.1 Introduction 30

6.2 PYNQ Z2 30

6.2.1Feature of PYNQ Z2 31

6.3 Board setup 33

6.4 Interfacing with computer 34

6.4.1 Jupyter notebook 34

6.5 Output from Pynq Z2 34

6.6 Concluding remarks 35

7 RESULT ANALYSIS……………………………………………..... 36

7.1 Introduction 36

7.2 Software analysis 36

7.2.1 Accuracy 36

7.2.2 False positive rate 36

7.2.3 False negative rate 37

7.2.4 True positive rate 37

7.2.5 True negative rate 37

7.2.6 Confusion matrix 37

7.3 Software evaluation results 37

7.4 Interference from software evaluation results 41

7.5 Concluding remarks 41

8 APPENDIX…………………………………………………… 42

9 CONCLUSION………………………………………………….. 61

BIBLIOGRAPHY……………………………………………….. 62

LIST OF TABLES

TABLE NO	NAME OF THE TABLE	PG.NO
2.1	Comparisons of different studies	06
5.1	Critical permissions in android platform	25
5.2	Opcodes in category wise	26
5.3	API calls used as feature	28
5.4	The intents used as feature	29
7.1	Classifiers mean performance	38
7.2	Confusion matrix of the classifiers	38

LIST OF FIGURES

FIG.NO	NAME OF THE FIGURE	PG.NO
4.1	Process flow of TCMD	15
4.2	Glimpse of the training dataset	16
4.3	Execution of Andropytool image	18
4.4	Glimpse of Andropytool output	19
4.5	The data frame of selected features of an APK file	21
4.6	TCMD output as goodware	22
4.7	TCMD output as malware	23
4.8	Detector results in spreadsheet	23
6.1	Overview of PYNQ-Z2	30
6.2	Block diagram of PYNQ-Z2	31
6.3	Hardware output of the detector	34
7.1	FNR of Classifiers	40
7.2	Accuracy of the classifiers	40

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION TO TRI-CLASSIFIER MALWARE DETECTOR

Malware detector is a system which is employed to detect the malicious code in a computer. It is mostly implemented in software which is served by corporates like Norton, Avast, Kaspersky and several other companies for variety of computer platforms like Windows, Mac and Android. In this work a malware is developed for Android platform by the combination of three machine learning algorithms each with different priority. The algorithm is implemented in hardware which has its own advantages which will be discussed further.

1.1.1 PROBLEM DEFINITION

Today the Android has become most widely used platform. Mostly the Android platforms are employed into smartphones. With the advent of technology, the android has reached many common people. They use smartphones for social interaction and for other utilities like banking, finance etc. Unlike Mac os the Android is very liberal and allows the users to download apps from unknown developers. Android apps available in Google play store is not as thoroughly verified for maliciousness as of Apple's. This makes many malware developers to develop malware for Android platform. These malwares can steal the user's personal information, can encrypt important files, may spread from one phone to another in chain fashion making these phones as their bot for any denial of service attacks. Hence, they can lead to people's psychological and financial stress.

Unlike personal computers smart phones are not provided with sophisticated resources. Hence employing any complex algorithms can be burden to the phone's main processor. This makes the algorithms to be lesser effective especially for full phone scans.

1.1.2 OBJECTIVES

The malwares for Android platform are evolving and new malwares are being developed every day. It is important to detect new and unknown malwares. As Android is mostly used by the common people it has more of entertainment and personal use than professional use when compared to other operating systems like Microsoft and other Linux based operating systems. Therefore, here privacy should have an upper hand when compared to productivity. Privacy issues of an app must not be compromised owing to productivity of an app. As mobile phones are resource constrained the detection algorithm should not burden the main processor and decrease the mobile phone’s performance.

Keeping these things in mind the objective of this detector is made to find even new or unknown malware, to make the detector more sensitive to give positive result than negative result for a given APK file, not to burden the phone’s main processor for sake of execution of this algorithm. To achieve these objectives various techniques are used by this detector.

1.2 EXISTING SYSTEMS

The existing systems uses variety of techniques for malware detection. The signature-based techniques are based on collecting signatures of an unknown malwares and compares with file to be tested. The signature can uniquely identify a file. The commonly used signatures are series of bytes of the file or cryptographic hash. The disadvantage of this method it cannot find new or unknown malwares. Polymorphic and metamorphic malwares may evade this system. This system could be improved by use generic signatures.

Another technique is data mining. It employs machine learning algorithms. Its features could be extracted either by static or dynamic way. For dynamic analysis sandbox environment must be setup. Setting up sandbox environment is tedious in mobile phones as it is resource constrained. Hence dynamic analysis for mobile phones will demand for its performance. Machine learning algorithms may use discrete classifiers or ensemble classifiers. The ensemble classifiers with equal weights are inferior than that of discrete ones.

Most of todays consumer used malware detectors are either software based or web based. Both of these methods are vulnerable for evading. Norton detectors are software based and they use signature based and heuristics-based techniques while Avast based are mostly web based.

1.3 PROPOSED SYSTEM

This detector employs machine learning based detection using nineteen static features. Machine learning approach helps to accomplish the objective of finding new and unknown malwares. Three classifiers are used in a combined form. Those three classifiers are Random forest, Decision tree and Logistic regression. First the Random forest classifier’s output is analyzed. If its output detects the file as malware then the final output of the detector is concluded as that the given test file is malware. Suppose the output of Random forest detects the file as good ware then the Decision tree classifier and Logistic regression classifiers are trained and their outputs are obtained. Now the net output of the detector is based on the majority voting between these three classifiers all with equal weight. This type of classifying techniques helps to fulfill the objective of making the output of classifiers more aligned to positivity than of negativity. This will on other hand will ensure the upper hand of the security than need for the app itself. Finally, the algorithm is implemented in a hardware. This helps to fulfill the objective of getting rid the burden of main processor of the cell phone.

Let us say in Boolean way the ‘true’ state corresponds to the output as malware (positivity) and ‘false’ state corresponds to the output as good ware (negativity) of a classifier, then final output of this detector for a given APK file will be:

$$\mathbf{TCMD = RFO + (DTO \ \& \ LRO)}$$

Where TCMD corresponds to Tri-classifier malware detector output, RFO corresponds to Random forest classifier output, DTO corresponds to decision tree classifier output and LRO corresponds to the Logistic regression classifier output.

1.4 CONCLUDING REMARKS

Thus, this project aims at developing a hardware level malware detector which detects even new or unknown malicious APK files which belongs to android platform. The classifier used consists of three discrete classifiers with varying priorities.

CHAPTER 2
LITERATURE SURVEY

2.1 INTRODUCTION

Since malwares are ever evolving studies related to malware detection are also ever evolving. Use of machine learning algorithms for malware detection has been proposed before almost two decades. But complexity of the algorithm and lack of rich computation power available restricted use of it. As the technology grew the deployment of machine learning algorithms became feasible. Since then many studies has been conducted based on malware detection using machine learning algorithms each with different methods. Some concentrate on finding detector of a particular family of malware while others tried for generic malware detection.

2.2 RELATED STUDIES

Due to vulnerabilities of signature-based detection recent research are based upon machine learning techniques. While dynamic approach may be more accurate but it is too costly and time consuming for end user application [1]. Hence static analysis is better for normal mobile phone users.

The classifier for machine learning approach needs lots of data sample's feature vector for training. Lot of researches focuses on very limited number of features (lesser than 3 for average). Naser Peiravian et al [2] uses Permissions and frequency of API calls as features. They quantify these features by detecting the presence or absence of each and every Permission and API calls available to an APK file in Android. They count to 130 features for Permission and 1326 features for API calls to total of 1156 features. This method is lesser efficient as the weightage given to less significant feature which distinguish the APKs as malware or good ware is given same as of the weightage of significant features. The malware authors can evade this detector by using the insignificant permissions in their code thereby confusing the detector without any compromise in functionality of that malware. Pankaj Kohli [6] used only frequency of critical API calls for detection.

Westyarian et al [3] used frequency of API packages as features. Unlike Naser Peiravian et al they used APIs class only responsible for permissions. Based on this criterion they used frequency of 16 API classes as features. Richard Killam [7] et al used frequency of string literals as their features.

Mahmood Fazlali et al [4] uses all normalized 227 Dalvik opcode's frequency as features to detect metamorphic malwares whereas Gerardo Canfora [5] et al uses frequency of 6 classes of opcodes that represents the alteration of control flow as features based on assumption that the complexity of good ware is more than malwares. More over using opcodes as feature is used to find apps with less optimization, which is commonly noticed in metamorphic and polymorphic malwares.

APK analyzers, IDA pro Disassembler, APK parser, Androgruard were the mostly found applications for disassembling the APK files. The dissembled file is used for feature extraction and feature selection. Alejandro Martin et al [8] developed a software tool called Andropytool. This tool integrates multiple tools for dynamic, static and pre-static analysis. It uses Androguard for static and pre-static analysis.

Datasets were commonly collected from open public repositories like VirusTotal.com, VirusSign.com etc…for malwares and benign APKs from play store and other reliable sources. The commonly used classifiers are Random forest, Decision tree (J48), Support vector machine, Naïve bayes and Logistic regression. Some even used bagging classifier.

Alejandro Martin et al used ensemble classifier consisting of Random forest, Decision tree and KNN with same weight. In most of the researches Random forest provided better results than others. Ensemble classifiers and Bagging classifier were less superior to conventional ones in results. Gerardo Canfora [5] recommended the Random forest classifier.

2.3 COMPARISON BETWEEN DIFFERENT STUDIES

The table 2.1 compares different studies based on malware detection using machine learning algorithm including this work. These studies have been mentioned in bibliography serially as per the number on the table for each study. Five parameters have been taken for comparison. They are number of training samples (A), malware to good ware ratio in training samples (B), whether it is static or dynamic analysis (C), number of feature vector (D), features taken (E). All these studies employ many common classifiers. Hence all of them use same classification algorithm except our study which uses Tri-Classification technique.

Table 2.1 Comparisons of different studies

STUDY	A	B	C	D	E
[1]	48919	49:51	Dynamic	42	Battery, Binder, CPU, Memory, Network, Permission
[2]	1860	33:67	Static	1456	Permissions, API Calls
[3]	412	1:1	Static	51	API package
[4]	700	2:5	Static	227	Op-codes
[5]	11120	1:1	Static	8	Op-codes
[7]	5834	11:39	Static	728	Strings
This work	248	47:53	Static	19	Permission, Op-codes, API calls, API packages, Intents

2.4 CONCLUDING REMARKS

Usage of machine learning algorithms for malware detection is still an emerging field. More research of these kind helps to find the features and classifiers which gives maximum accuracy for both specific and generic malware detector and find good pattern between malware and good ware.

CHAPTER 3
CONCEPTS REGARDING MALWARE DETECTION

3.1 INTRODUCTION

The rapid development of the Internet, malware became one of the major cyber threats nowadays. While the diversity of malware is increasing, anti-virus scanners cannot fulfill the needs of protection, resulting in millions of hosts being attacked. Malware detection is indeed now a days. However, currently utilized signature-based methods cannot provide accurate detection of zero-day attacks and polymorphic viruses. That is why the need for machine learning-based detection arises.

In addition to that, there is a decrease in the skill level that is required for malware development, due to the high availability of attacking tools on the Internet nowadays. High availability of anti-detection techniques, as well as ability to buy malware on the black-market result in the opportunity to become an attacker for anyone, not depending on the skill level.

Therefore, malware protection of Android phones which are most used by the people is one of the most important cyber security tasks for single users and businesses, since even a single attack can result in compromised data and sufficient losses. Massive losses and frequent attacks dictate the need for accurate and timely detection methods. The following are the some of the concepts involved in malware detection.

3.2 MALWARES

Any software performing malicious actions, including information stealing, crashing the processor can be referred to as malware. Kaspersky Labs define malware as “a type of computer program designed to infect a legitimate user's computer and inflict harm on it in multiple ways.” The ideology of classifying malwares into types is to understand its nature in criteria based on the similarities and difference between them and identifying malwares easily. To have a better understanding of the methods and logic behind the malware, it is useful to classify it. Malware can be divided into several classes depending on its purpose. A common feature of Malware is that they are specifically designed to damage, disrupt, steal, or in general inflict some other bad or illegitimate actions. The classes are as follows:

3.2.1 VIRUS

This is the simplest form of software. It is simply any piece of software that is loaded and launched without user's permission while reproducing itself or infecting other software.

3.2.2 WORM

This malware type is very similar to the virus. The difference is that worm can spread over the network and replicate to other machines.

3.2.3 TROJON HORSE

This malware class is used to define the malware types that aim to appear as legitimate software. Because of this, the general spreading vector utilized in this class is social engineering, i.e. making people think that they are downloading the legitimate software.

3.2.4 ADWARE

The only purpose of this malware type is displaying advertisements on the computer. Often adware can be seen as a subclass of spyware and it will very unlikely lead to dramatic results.

3.2.5 SPYWARE

As it implies from the name, the malware that performs espionage can be referred to as spyware. Typical actions of spyware include tracking search history to send personalized advertisements, tracking activities to sell them to the third parties subsequently.

3.2.6 ROOTKIT

Its functionality enables the attacker to access the data with higher permissions than is allowed. For example, it can be used to give an unauthorized user administrative access. Rootkits always hide its existence and quite often are unnoticeable on the system, making the detection and therefore removal incredibly hard.

3.2.7 BACKDOOR

The backdoor is a type of malware that provides an additional secret “entrance” to the system for attackers. By itself, it does not cause any harm but provides attackers with broader attack surface. Because of this, backdoors are never used independently. Usually, they are preceding malware attacks of other types.

3.2.8 KEYLOGGER

The idea behind this malware class is to log all the keys pressed by the user, and, therefore, store all data, including passwords, bank card numbers and other sensitive information.

3.2.9 RANSOMWARE

This type of malware aims to encrypt all the data on the machine and ask a victim to transfer some money to get the decryption key. Usually, a machine infected by ransomware is “frozen” as the user cannot open any file, and the desktop picture is used to provide information on attacker’s demands.

3.3 MALWARES IN ANDROID PLATFORM

The first known mobile virus, "Timofonica", originated in Spain and was identified by antivirus labs in Russia and Finland in June 2000. "Timofonica" sent SMS messages to GSM-capable mobile phones that read "Information for you: Telefónica is fooling you." These messages were sent through the Internet

SMS gateway of the MoviStar mobile operator.Android platform starts became the universal front-end in the IoE and IoT, mobile attacks will continue to grow rapidly as new technologies expand the attack surface. Vendors, manufacturer, providers should extend vulnerability shielding and exploit-prevention technologies and Anti-Malware vendors have to enhance their actual solutions, because in this digital generation it is rare to spot a person without mobile phone. Among them most of the population is using an android based mobile phones.

The open source Android platform allows developers to take full advantage of the mobile operating system, but also raises significant issues related to malicious applications. On one hand, the popularity of Android attracts the attention of most developers for producing their applications on this platform. The increased numbers of applications, on the other hand, prepares a suitable prone for some users to develop different kinds of malware

Repackaging is one of the most common techniques for malware developers to install malicious applications on an Android platform. These types of approaches normally start from popular legitimate Apps and misuse them as malware. The developers normally download popular Apps, disassemble them, add their own malicious codes, and then re-assemble and upload the new App to official or alternative markets. The idea behind this work, arise from the awareness that a more effective and holistic anti malware approach for security of an individual which is prior for a human being.

3.4 APK FILE

Application Package (APK) is the package file format used by the Android operating system for distribution and installation of mobile apps, mobile games and middleware.

APK is analogous to other software packages such as APPX in Microsoft Windows or a Debian package in Debian-based operating systems. To make an APK file, a program for Android is first compiled using Android Studio, and then all of its parts are packaged into one container file. An APK file contains all of a program's code (such as .dex files), resources, assets, certificates, and manifest file. As is the case with many file formats, APK files can have any name needed, but it may be required that the file name ends in the file extension ".apk" for being recognized as such.

The Android system allows users to manually install APK files only after they turn on an "Unknown Sources" setting that allows installation from sources other than trusted ones like Google Play. One may do so for many reasons, such as to install apps not found on the store, or to install an older version. Although one can downgrade an app this way by uninstalling the new version first, doing it via Android Debug Bridge is better as it allows for keeping data. An APK file is an archive that usually contains the following files and directories:

- META-INF directory:
  - MANIFEST.MF: the Manifest file
  - The certificate of the application.
  - CERT.SF: The list of resources and a SHA-1 digest of the corresponding lines in the MANIFEST.MF file.
- lib: the directory containing the compiled code that is platform dependent; the directory is split into more directories within it:
  - armeabi: compiled code for all ARM based processors only
  - armeabi-v7a: compiled code for all ARMv7 and above based processors only
  - arm64-v8a: compiled code for all ARMv8 arm64 and above based processors only
  - x86: compiled code for x86 processors only
  - x86_64: compiled code for x86 64 processors only
  - mips: compiled code for MIPS processors only
- res:the directory containing resources not compiled into resources.arsc .
- assets: a directory containing applications assets, which can be retrieved by AssetManager.
- AndroidManifest.xml:An additional Android manifest file, describing the name, version, access rights, referenced library files for the application. This file may be in Android binary XML that can be converted into human-readable plaintext XML with tools such as AXMLPrinter2, apktool, or Androguard.
- classes.dex: The classes compiled in the dex file format understandable by the Dalvik virtual machine and by the Android Runtime.
- resources.arsc: a file containing precompiled resources, such as binary XML for example.

3.5 DETECTION METHODOLOGIES

It is essential to understand the basics of two malware analysis approaches: static and dynamic malware analysis. As it implies from the name, static analysis is performed “statically”, i.e. without execution of the file. In contrast, dynamic analysis is conducted on the file while it is being execute with the help of virtual machine.

3.5.1 SIGNATURE-BASED DETECTION

Now, having the background on malware analysis, we can define the detection methods. The signature-based analysis is a static method that relies on predefined signatures. These can be file fingerprints, e.g. MD5 or SHA1 hashes, static strings, file metadata. The scenario of detection, in this case, would be as follows: when a file arrives at the system, it is statically analyzed by the antivirus software. If any of the signatures is matched, an alert is triggered, stating that this file is suspicious. Very often this kind of analysis is enough since well-known malware samples can often be detected based on hash values. However, attackers started to develop malware in a way that it can change its signature. This malware feature is referred to as polymorphism. Obviously, such malware cannot be detected using purely signature-based detection techniques. Moreover, new malware types cannot be detected using signatures, until the signatures are created.

3.5.2 USING MACHINE LEARNING

To detect unknown malwares machine learning method is used. Machine learning is an application of artificial intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. As stated before, malware detectors that are based on signatures can perform well on previously-known malware, that was already discovered by some antivirus vendors. However, it is unable to detect polymorphic malware, that has an ability to change its signatures, as well as new malware, for which signatures have not been created yet. In turn, the accuracy of heuristics-based detectors is not always sufficient for adequate detection, resulting in a lot of false-positives and false-negatives. Need for the new detection methods is dictated by the high spreading rate of polymorphic viruses. One of the solutions to this problem is reliance on the heuristics-based analysis in combination with machine learning methods that offer a higher efficiency during detection Here instead of merely comparing any parameter of a file with the database, the patterns of parameters are found for each category under the classification using statistical approach. To find patterns of parameters two approaches are commonly used. They are static and dynamic analysis.

- **STATIC ANALYSIS**

Static analysis can be viewed as “reading” the source code of the malware and trying to infer the behavioral properties of the file. Static analysis can include various techniques Static analysis often relies on certain tools. Beyond the simple analysis, they can provide information on protection techniques used by malware. The main advantage of static analysis is the ability to discover all possible behavioral scenarios. Researching the code itself allows the researcher to see all ways of malware execution, that are not limited to the current situation. Moreover, this kind of analysis is safer than dynamic, since the file is not executed and it cannot result in bad consequences for the system. On the other hand, static analysis is much more time-consuming. Because of these reasons it is not usually used in real-world dynamic environments, such as anti-virus systems, but is often used for research purposes, e.g. when developing signatures for zero-day malware.

- **DYNAMIC ANALYSIS**

Another analysis type is dynamic analysis. Unlike static analysis, here the behavior of the file is monitored while it is executing and the properties and intentions of the file are inferred from that information. Usually, the file is run in the virtual environment, for example in the sandbox. During this kind of analysis, it is possible to find all behavioral attributes, such as opened files, memory usage, battery usage etc. On the other hand, the static analysis only shows the behavioral scenario relevant to the current system properties. For example, if our virtual machine has Windows 7 installed, the results might be different from the malware running under Windows 8.1.

In this method, the actual behavior of malware is observed during its execution, looking for the signs of malicious behavior: modifying host files, registry keys, establishing suspicious connections. By itself, each of these actions cannot be a reasonable sign of malware, but their combination can raise the level of suspiciousness of the file. There is some threshold level of suspiciousness defined, and any malware exceeding this level raises an alert.

The accuracy level of heuristics-based detection highly depends on the implementation. The best ones utilize the virtual environment, e.g. the sandbox to run the file and monitor its behavior. Although this method is more time consuming, it is much safer, since the file is checked before actually executing. The main advantage of behavior-based detection method is that in theory, it can identify not only known malware families but also zero-day attacks and polymorphic viruses. Thus, in this work we are implementing the behavior-based method of malware detection.

3.6 NEED FOR HARDWARE IMPLEMENTATION

Mostly resources of a mobile phone based on android platform is less sophisticated when compared with most personal computers or laptop. As mobile phone is resource constrained implementation of complex algorithms can increase its burden and prevent other tasks from using its resources.

Parallel processing in FPGA based hardwares can increase the speed and performance. Reverse engineering of hardwares based on VLSI is more difficult when compared to that of software. Hence the algorithm will be more secure which in turn increases the reliability of the detector as malware developers find difficult to develop malwares which will evade this algorithm. Due to these reasons implementing this algorithm in hardware has an upper hand when compared to mere software implementation.

3.7CONCLUSION

This chapter thus gives general idea of the project and explaining it will detonates the concepts further discussed in this report thereby avoiding misconceptions.

CHAPTER 4
SOFTWARE IMPLEMENTATION

4.1 INTRODUCTION

The flowchart of this detection algorithm is shown in figure 4.1. Each process is implemented using a software. The implementation these each process which uses a software is discussed below.

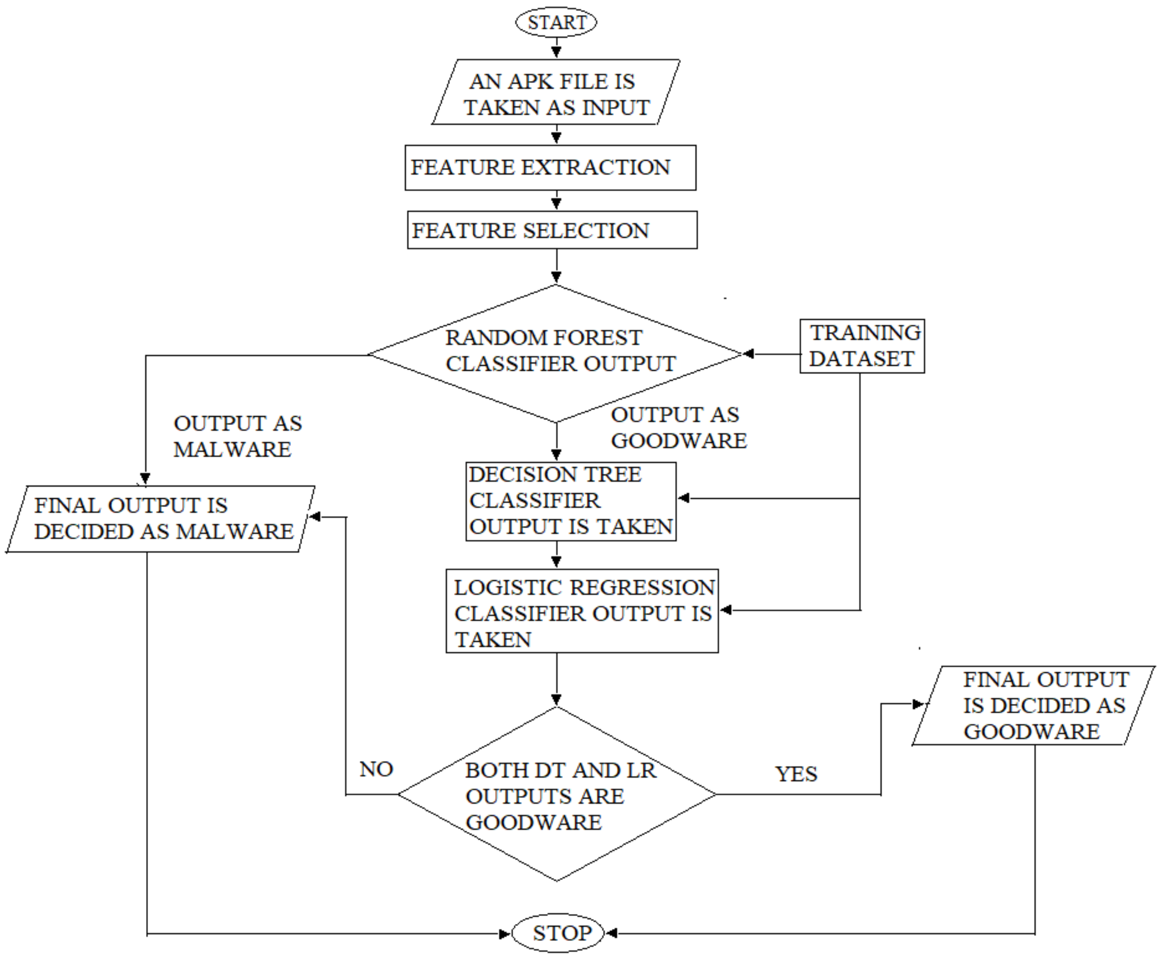
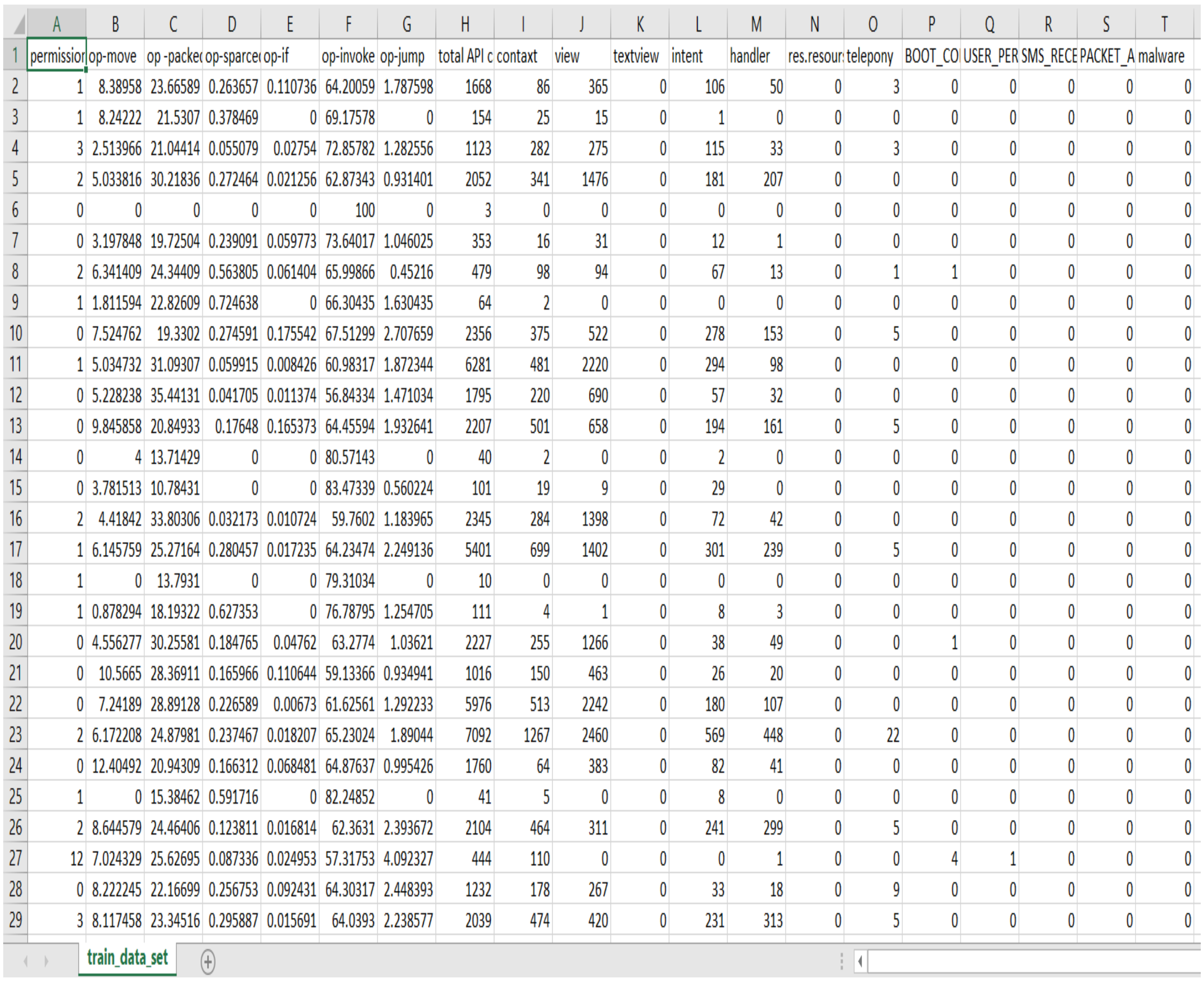

START
AN APK FILE IS TAKEN AS INPUT
FEATURE EXTRACTION
FEATURE SELECTION
RANDOM FOREST CLASSIFIER OUTPUT
TRAINING DATASET
OUTPUT AS MALWARE
OUTPUT AS GOODWARE
FINAL OUTPUT IS DECIDED AS MALWARE
DECISION TREE CLASSIFIER OUTPUT IS TAKEN
LOGISTIC REGRESSION CLASSIFIER OUTPUT IS TAKEN
FINAL OUTPUT IS DECIDED AS GOODWARE
NO
BOTH DT AND LR OUTPUTS ARE GOODWARE
YES
STOP

Fig 4.1 Process flow of TCMD

4.2 TRAINING DATASET

This detection algorithm is based on supervised machine learning. Supervised machine learning algorithms needs training dataset with labelled data unlike unsupervised learning. Here training data set consisting of 248 APK files is used of which 117 APK files are malware and 131 APK files are goodware. These APK files are taken from variety of sources like most of the malwares are taken from the Github repository [11] and goodwares are taken from APK pure website [12]. The malware sample to goodware sample ratio is taken nearly as 1:1, therefore no class imbalance problem. Then these files are undergone the process of feature extraction and feature selection. The resulting output is collected for all the sample files and saved in CSV format. Glimpse of this CSV file is shown in figure 4.2. This file is used to train the classifiers.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	permissior	op-move	op -packe(	op-sparce(	op-if	op-invoke	op-jump	total API c	contaxt	view	textview	intent	handler	res.resour	telepony	BOOT_CO	USER_PER	SMS_RECE	PACKET_A	malware
2	1	8.38958	23.66589	0.263657	0.110736	64.20059	1.787598	1668	86	365	0	106	50	0	3	0	0	0	0	0
3	1	8.24222	21.5307	0.378469	0	69.17578	0	154	25	15	0	1	0	0	0	0	0	0	0	0
4	3	2.513966	21.04414	0.055079	0.02754	72.85782	1.282556	1123	282	275	0	115	33	0	3	0	0	0	0	0
5	2	5.033816	30.21836	0.272464	0.021256	62.87343	0.931401	2052	341	1476	0	181	207	0	0	0	0	0	0	0
6	0	0	0	0	0	100	0	3	0	0	0	0	0	0	0	0	0	0	0	0
7	0	3.197848	19.72504	0.239091	0.059773	73.64017	1.046025	353	16	31	0	12	1	0	0	0	0	0	0	0
8	2	6.341409	24.34409	0.563805	0.061404	65.99866	0.45216	479	98	94	0	67	13	0	1	1	0	0	0	0
9	1	1.811594	22.82609	0.724638	0	66.30435	1.630435	64	2	0	0	0	0	0	0	0	0	0	0	0
10	0	7.524762	19.3302	0.274591	0.175542	67.51299	2.707659	2356	375	522	0	278	153	0	5	0	0	0	0	0
11	1	5.034732	31.09307	0.059915	0.008426	60.98317	1.872344	6281	481	2220	0	294	98	0	0	0	0	0	0	0
12	0	5.228238	35.44131	0.041705	0.011374	56.84334	1.471034	1795	220	690	0	57	32	0	0	0	0	0	0	0
13	0	9.845858	20.84933	0.17648	0.165373	64.45594	1.932641	2207	501	658	0	194	161	0	5	0	0	0	0	0
14	0	4	13.71429	0	0	80.57143	0	40	2	0	0	2	0	0	0	0	0	0	0	0
15	0	3.781513	10.78431	0	0	83.47339	0.560224	101	19	9	0	29	0	0	0	0	0	0	0	0
16	2	4.41842	33.80306	0.032173	0.010724	59.7602	1.183965	2345	284	1398	0	72	42	0	0	0	0	0	0	0
17	1	6.145759	25.27164	0.280457	0.017235	64.23474	2.249136	5401	699	1402	0	301	239	0	5	0	0	0	0	0
18	1	0	13.7931	0	0	79.31034	0	10	0	0	0	0	0	0	0	0	0	0	0	0
19	1	0.878294	18.19322	0.627353	0	76.78795	1.254705	111	4	1	0	8	3	0	0	0	0	0	0	0
20	0	4.556277	30.25581	0.184765	0.04762	63.2774	1.03621	2227	255	1266	0	38	49	0	0	1	0	0	0	0
21	0	10.5665	28.36911	0.165966	0.110644	59.13366	0.934941	1016	150	463	0	26	20	0	0	0	0	0	0	0
22	0	7.24189	28.89128	0.226589	0.00673	61.62561	1.292233	5976	513	2242	0	180	107	0	0	0	0	0	0	0
23	2	6.172208	24.87981	0.237467	0.018207	65.23024	1.89044	7092	1267	2460	0	569	448	0	22	0	0	0	0	0
24	0	12.40492	20.94309	0.166312	0.068481	64.87637	0.995426	1760	64	383	0	82	41	0	0	0	0	0	0	0
25	1	0	15.38462	0.591716	0	82.24852	0	41	5	0	0	8	0	0	0	0	0	0	0	0
26	2	8.644579	24.46406	0.123811	0.016814	62.3631	2.393672	2104	464	311	0	241	299	0	5	0	0	0	0	0
27	12	7.024329	25.62695	0.087336	0.024953	57.31753	4.092327	444	110	0	0	0	1	0	0	4	1	0	0	0
28	0	8.222245	22.16699	0.256753	0.092431	64.30317	2.448393	1232	178	267	0	33	18	0	9	0	0	0	0	0
29	3	8.117458	23.34516	0.295887	0.015691	64.0393	2.238577	2039	474	420	0	231	313	0	5	0	0	0	0	0

Fig 4.2 Glimpse of the training dataset

4.3 FEATURE EXTRACTION

Feature extraction is the transformation of original data to a data set with a reduced number of variables, which contains the most discriminatory information. It builds valuable information from the raw data. Here the raw data is an APK file. An APK file is an archive file which has many sub files of data and meta data. For purposes of analysis with the machine learning algorithms this raw file must be transformed into useful high-level information. There are some feature extraction softwares for APK files like Omnidroid, Dendroid, Androguard and APK_parser. Here for this detector a high-level integrated feature extraction tool called AndroPyTool is used. This tool is used because it user-friendly and no compatibility problem as it is available as docker image.

4.3.1 ANDROPYTOOL

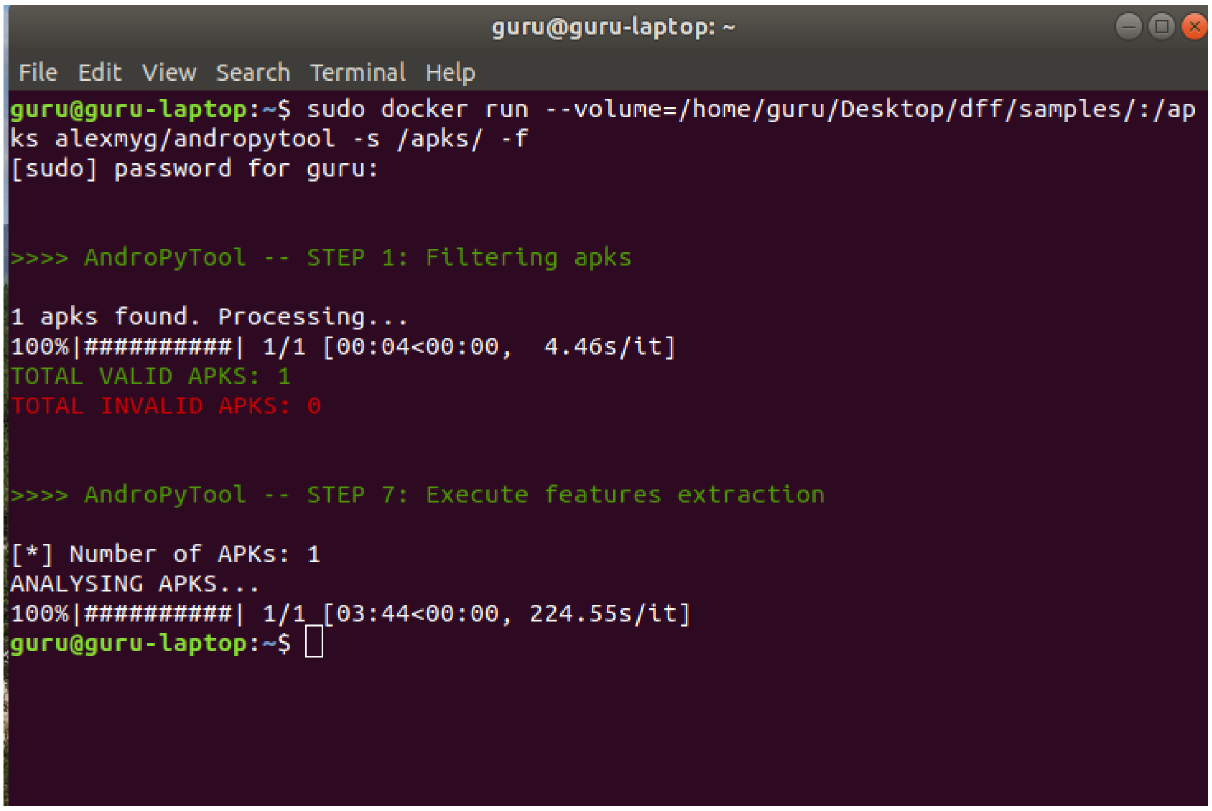
AndroPyTool is a tool for the extraction of both, static and dynamic features from Android applications [8]. It aims to provide Android malware analysis with an integrated environment to extract multi-source features able of modeling the behavior of a sample and that can be used to discern its nature, whether malware or good ware. The pre-static analysis is focused on extracting meta-information, features from the compressed APK file which do not require code inspection. The static analysis step is employed to extract more detailed features, such as declared Application Programming Interface (API) calls or permissions required. AndroPyTool integrates different analysis tools and Android applications processing tools, in order to deliver fine-grained reports drawing their individual behavior and characteristics. This framework, which has been implemented as a Python project, takes the APK file to be analyzed as input.

4.3.2 IMPLEMENTATION

AndroPyTool is also available as a docker image. Docker can be directly installed if the operating system uses Linux based kernel. In other cases, like Windows operating system the Docker can installed by using virtual box based on Linux. Here in this case Windows 10 home operating system is used. After installing Docker and pulling AndroPyTool image various commands can be used to use the various features of this tool. Here for this project the following command is being used:

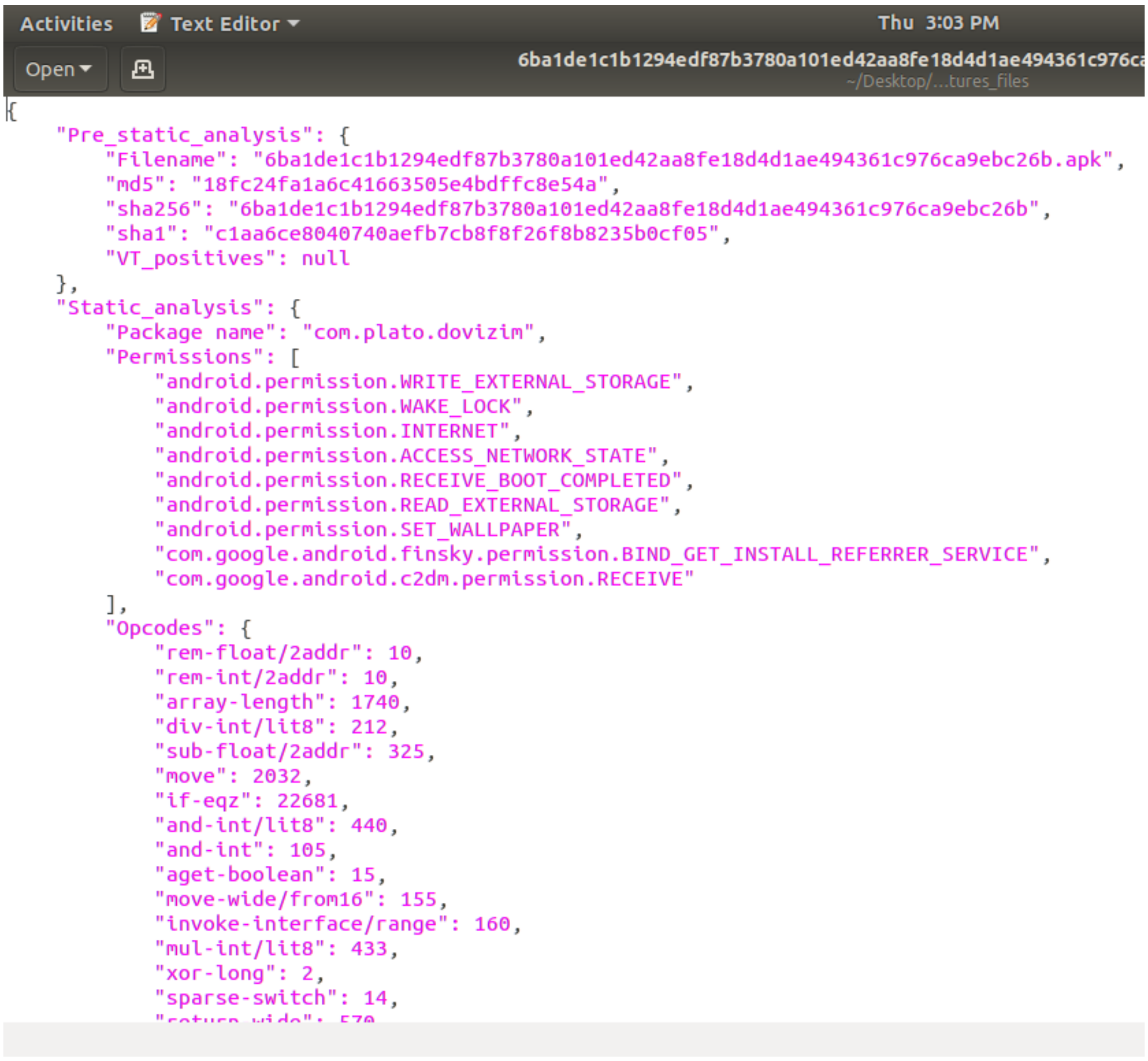
docker run –volume=/path of the APK file/:/apks alexmyg/andropytool -s /apks/ -f

This command will first filter the given file as valid APK file or not. If it is a valid APK file then it extracts static and pre-static features and presents the result in JSON format. Figure 4.3 shows the execution of this command in command prompt and Figure 4.4 shows the glimpse of output of AndroPytool.



```
guru@guru-laptop:~$ sudo docker run --volume=/home/guru/Desktop/dff/samples/:/ap
ks alexmyg/andropytool -s /apks/ -f
[sudo] password for guru:


>>>> AndroPyTool -- STEP 1: Filtering apks

1 apks found. Processing...
100%|##########| 1/1 [00:04<00:00,  4.46s/it]
TOTAL VALID APKS: 1
TOTAL INVALID APKS: 0


>>>> AndroPyTool -- STEP 7: Execute features extraction

[*] Number of APKs: 1
ANALYSING APKS...
100%|##########| 1/1 [03:44<00:00, 224.55s/it]
guru@guru-laptop:~$
```

Fig 4.3 Execution of AndroPyTool image



```
{
    "Pre_static_analysis": {
        "Filename": "6ba1de1c1b1294edf87b3780a101ed42aa8fe18d4d1ae494361c976ca9ebc26b.apk",
        "md5": "18fc24fa1a6c41663505e4bdffc8e54a",
        "sha256": "6ba1de1c1b1294edf87b3780a101ed42aa8fe18d4d1ae494361c976ca9ebc26b",
        "sha1": "c1aa6ce8040740aefb7cb8f8f26f8b8235b0cf05",
        "VT_positives": null
    },
    "Static_analysis": {
        "Package name": "com.plato.dovizim",
        "Permissions": [
            "android.permission.WRITE_EXTERNAL_STORAGE",
            "android.permission.WAKE_LOCK",
            "android.permission.INTERNET",
            "android.permission.ACCESS_NETWORK_STATE",
            "android.permission.RECEIVE_BOOT_COMPLETED",
            "android.permission.READ_EXTERNAL_STORAGE",
            "android.permission.SET_WALLPAPER",
            "com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE",
            "com.google.android.c2dm.permission.RECEIVE"
        ],
        "Opcodes": {
            "rem-float/2addr": 10,
            "rem-int/2addr": 10,
            "array-length": 1740,
            "div-int/lit8": 212,
            "sub-float/2addr": 325,
            "move": 2032,
            "if-eqz": 22681,
            "and-int/lit8": 440,
            "and-int": 105,
            "aget-boolean": 15,
            "move-wide/from16": 155,
            "invoke-interface/range": 160,
            "mul-int/lit8": 433,
            "xor-long": 2,
            "sparse-switch": 14,
```

Fig 4.4 Glimpse of AndroPyTool output

4.4 FEATURE SELECTION

In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for several reasons like simplification of models to make them easier to interpret by researchers/users, shorter training times, to avoid the curse of dimensionality, enhanced generalization by reducing over fitting (formally, reduction of variance) and give more weightage to the features which are most distinguishable between the objects under classification. Here this detector uses nineteen static features and they are discussed in detail in next chapter. These features are taken by performing feature selection on the output of

AndroPyTool. The process is carried out by the algorithm written in Python language version 3.

4.4.1 PYTHON

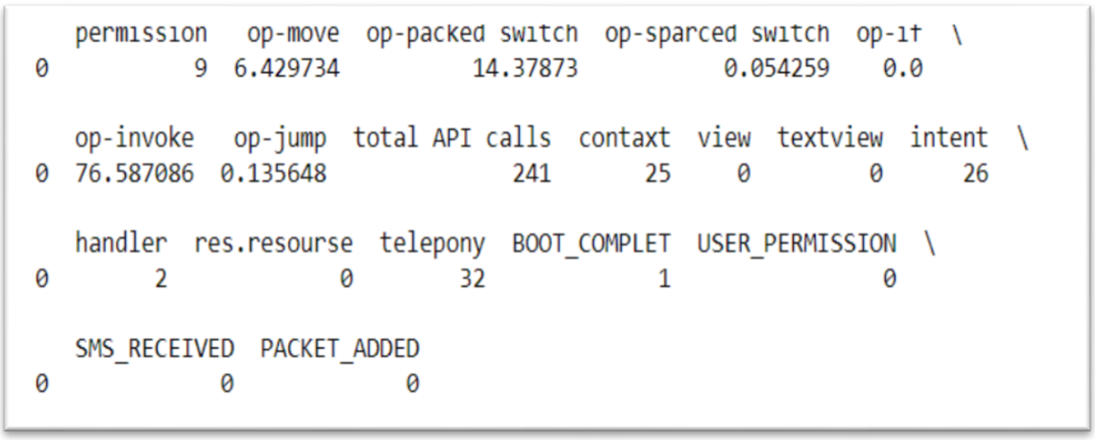
Python is a widely used high-level programming language for general-purpose programming. Apart from being open source programming language, Python is a great object-oriented, interpreted, and interactive programming language. Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high-level dynamic data types, and dynamic typing. There are interfaces to many system call and libraries, as well as to various windowing systems. New built-in modules are easily written in C or C++ (or other languages, depending on the chosen implementation). Python is also usable as an extension language for applications written in other languages that need easy-to-use scripting or automation interfaces. Python is widely considered as the preferred language for machine learning due to the following reasons:

1. It's simple to learn. As compared to C, C++ and java the syntax is simpler and Python also consists of a lot of code libraries for ease of use.
2. Capability of interacting with almost all the third-party languages and platforms.
3. Open source– Python along with R is gaining momentum and popularity in the analytics domain since both of these languages are open source.
4. Scientific Python packages such as numpy, scipy, and matplotlib can be installed in a program running on Python. These packages cater to machine learning and help developers detect patterns in big sets of data.

4.4.2 IMPLEMENTATION

Feature selection process carried out on the output of AndroPyTool which is in JSON format. Python has a way to store the collection of data called Dictionary. In dictionary the data are in key-value pair similar to the way the data are stored in JSON format. Hence the AndroPyTool output can be parsed by converting it into a dictionary object. This is accomplished with the help of Python's inbuilt package called json. This package allows to import an JSON file and convert it into a dictionary object.

After the conversion the required data can be gathered using appropriate keys and using necessary operators like 'in' and exceptions. The gathered data is quantised into necessary features. As classifiers gets input in Pandas data frame the quantised features are converted into a Pandas data frame using Pandas library. Figure 4.5 shows the data frame of selected features of an APK file.



```
   permission   op-move  op-packed switch  op-sparced switch  op-if  \
0           9  6.429734          14.37873           0.054259    0.0

   op-invoke   op-jump  total API calls  contaxt  view  textview  intent  \
0  76.587086  0.135648              241       25     0         0      26

   handler  res.resourse  telepony  BOOT_COMPLET  USER_PERMISSION  \
0        2             0        32             1                0

   SMS_RECEIVED  PACKET_ADDED
0             0             0
```

Fig 4.5 The data frame of selected features of an APK file

4.5 CLASSIFICATION

In machine learning and statistics, classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Classification is considered an instance of supervised learning, i.e., learning where a training set of correctly identified observations is available. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. This detector uses three machine learning classifiers in combination for classification. They are Random forest classifier, Decision Tree classifier and Logistic Regression classifier.

4.5.1 CLASSIFIACTION STRATEGY

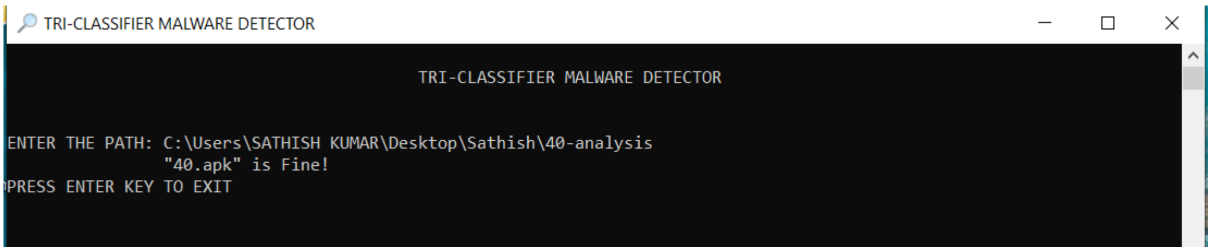
Most of the researches show that the Random forest has better accuracy than other classifiers. Gerardo Canfor et al [5] recommends the Random forest classifier owing to its accuracy. Many ensemble classifiers and bagging classifier fail to compete with the accuracy of Random forest classifier because the drawback of using ensemble classifier with mere majority voting is that the result may be influenced by the inferior classifiers which is given same weightage of superior classifier like Random forest. The algorithm is made more sensitive to give positive result than of negative result. To achieve this these three classifiers are used with varying priority while deciding an APK file as malware. Random forest algorithm is given the highest priority because most of the research shows that this algorithm shows more accuracy in result when compared to other classifiers. Decision tree algorithm and Logistic regression algorithm are given lesser priority because studies show that these are less accurate when compared to Random forest algorithm. At first the result of random forest algorithm is analyzed. If it is positive without any further analysis the APK file is declared as malware. If the result is negative then results of decision tree and logistic regression is taken. The final result is declared by taking majority voting with the output of all these classifiers with each with equal weightage.

4.5.2 IMPLEMENTATION

Sci-kit learn is a Python library which is used to implement machine learning algorithms. Classifiers like Random forest, Decision tree and Logistic regression can be directly implemented using this library. The classifiers need to be trained by the training dataset in Pandas data frame. The feature extracted and feature selected training dataset is already been stored externally to the program in CSV format. This file is imported into the program and converted into Pandas data frame using Pandas library. Then the classifiers are trained. The selected features of an APK file which must be tested is also converted into Pandas data frame and it is given as input to the classifiers.

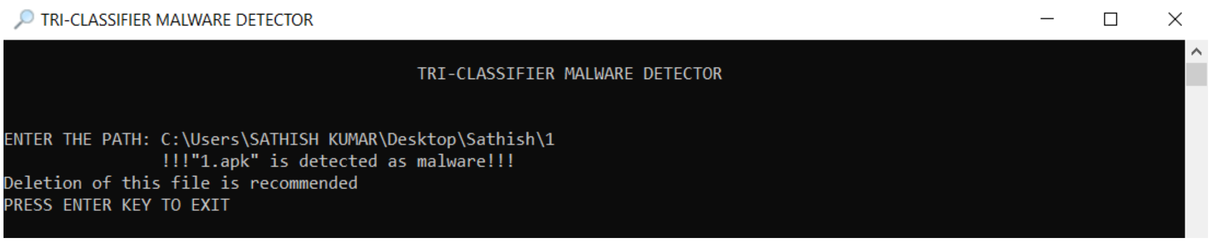
4.6 INTERACTION WITH THE PROGRAM

When the program is executed, it will ask for the path of the file which contains the extracted features of the APK file to be analyzed. At entering the path after very few seconds the program will output its detection result as either malware or goodware as shown in figure 4.6 and figure 4.7.



```
                              TRI-CLASSIFIER MALWARE DETECTOR


ENTER THE PATH: C:\Users\SATHISH KUMAR\Desktop\Sathish\40-analysis
                "40.apk" is Fine!
PRESS ENTER KEY TO EXIT
```

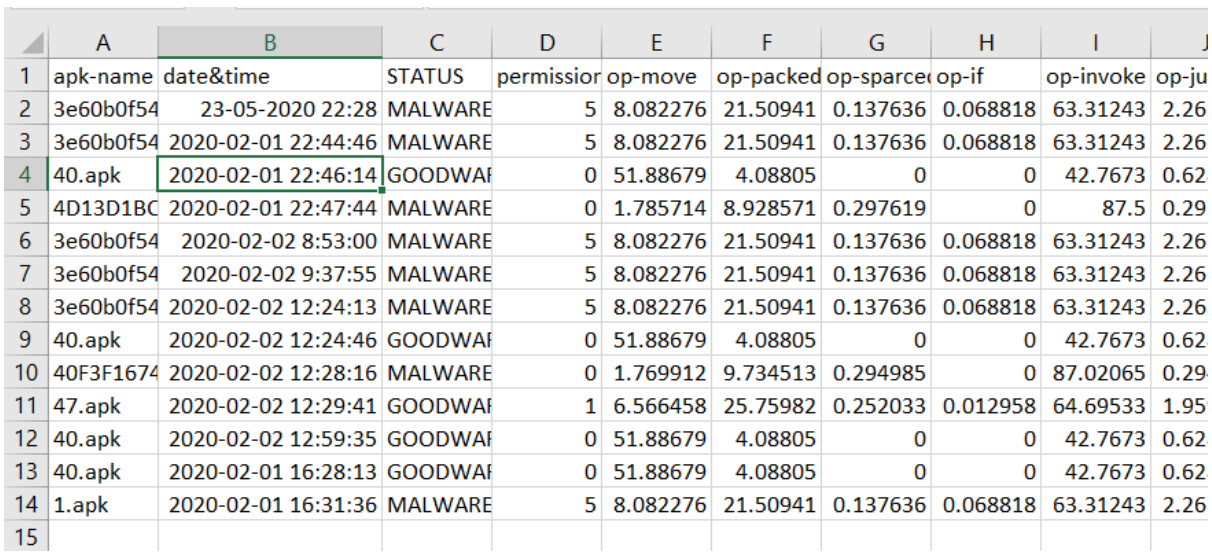
Fig 4.6 TCMD output as goodware



```
                              TRI-CLASSIFIER MALWARE DETECTOR

ENTER THE PATH: C:\Users\SATHISH KUMAR\Desktop\Sathish\1
                !!!"1.apk" is detected as malware!!!
Deletion of this file is recommended
PRESS ENTER KEY TO EXIT
```

Fig 4.7 TCMD output as malware

Further the detection result and the features selected from that APK file will be stored in a excel workbook in XLSX format along with the time and date at which the detection program is executed. Knowing the time of execution of the detection program and exporting the data to excel workbook is performed using Python libraries Datetime and Openpyxl respectively. Figure 4.8 shows the screenshot of the excel workbook which stores the result of this detector.



	A	B	C	D	E	F	G	H	I	J
1	apk-name	date&time	STATUS	permissior	op-move	op-packed	op-sparce(	op-if	op-invoke	op-ju
2	3e60b0f54	23-05-2020 22:28	MALWARE	5	8.082276	21.50941	0.137636	0.068818	63.31243	2.26
3	3e60b0f54	2020-02-01 22:44:46	MALWARE	5	8.082276	21.50941	0.137636	0.068818	63.31243	2.26
4	40.apk	2020-02-01 22:46:14	GOODWAI	0	51.88679	4.08805	0	0	42.7673	0.62
5	4D13D1BC	2020-02-01 22:47:44	MALWARE	0	1.785714	8.928571	0.297619	0	87.5	0.29
6	3e60b0f54	2020-02-02 8:53:00	MALWARE	5	8.082276	21.50941	0.137636	0.068818	63.31243	2.26
7	3e60b0f54	2020-02-02 9:37:55	MALWARE	5	8.082276	21.50941	0.137636	0.068818	63.31243	2.26
8	3e60b0f54	2020-02-02 12:24:13	MALWARE	5	8.082276	21.50941	0.137636	0.068818	63.31243	2.26
9	40.apk	2020-02-02 12:24:46	GOODWAI	0	51.88679	4.08805	0	0	42.7673	0.62
10	40F3F1674	2020-02-02 12:28:16	MALWARE	0	1.769912	9.734513	0.294985	0	87.02065	0.29
11	47.apk	2020-02-02 12:29:41	GOODWAI	1	6.566458	25.75982	0.252033	0.012958	64.69533	1.95
12	40.apk	2020-02-02 12:59:35	GOODWAI	0	51.88679	4.08805	0	0	42.7673	0.62
13	40.apk	2020-02-01 16:28:13	GOODWAI	0	51.88679	4.08805	0	0	42.7673	0.62
14	1.apk	2020-02-01 16:31:36	MALWARE	5	8.082276	21.50941	0.137636	0.068818	63.31243	2.26
15										

Fig 4.8 Detector results in spreadsheet

4.7 CONCLUSION REMARKS

Thus, this detection algorithm is accomplished by using softwares such as AndroPyTool and Python programming language. To successfully use these softwares additional softawares and programs were used as compliment like Docker, Python IDE and Python libraries like json, Pandas, sci-kit learn, openpyxl and datetime. These softwares are used because they are user friendly, open sourced and also efficient enough to accomplish the task.

CHAPTER 5
FEATURE SELECTION

5.1 INTRODUCTION

In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for several reasons like simplification of models to make them easier to interpret by researchers/users, shorter training times, to avoid the curse of dimensionality, enhanced generalization by reducing over fitting (formally, reduction of variance) and give more weightage to the features which are most distinguishable between the objects under classification. This malware detector uses multiple features to detect whether the user given APK file is malware or not. This uses 19 static features. The following are the features used.

5.2 PERMISSIONS

Permissions that an APK file seeks from the Android can be extracted from that APK's manifest.hml file. Android don't allow the file to access the resources if corresponding permission is not declared at manifest.hml. Using permissions as feature is conventional yet finds most of the malware. Usually this feature is quantified by taking in account all the available permissions as features [2] or using set of critical permissions like read_phone_state which allows read phones information like phone number, ongoing calls and network information, send_sms, receive_sms and some more as used by alejadro martin et al [8]. Here in this detector the feature relating to permissioin is quantified by counting the number of critical permissions of an APK file. Thus, using permission data of an APK file only one feature is selected. This is taken as feature based upon the fact that number of critical permissions used by malware is greater than goodwares making the value of this feature distinguishable between them. Table 5.1 shows the permissions whose net occurrence is taken as a feature.

TABLE 5.1 Critical permissions in android platform

SI	CRICTICAL PERMISSIONS IN AN APK
1	READ_CALENDAR
2	WRITE_CALENDAR
3	CAMERA
4	READ_CONTACTS
5	WRITE_CONTACTS
6	GET_ACCOUNTS
7	ACCESS_FINE_LOCATION
8	ACCESS_COARSE_LOCATION
9	RECORD_AUDIO
10	READ_PHONE_STATE
11	READ_PHONE_NUMBERS
12	CALL_PHONE
13	ANSWER_PHONE_CALLS
14	READ_CALL_LOG
15	WRITE_CALL_LOG
16	ADD_VOICEMAIL
17	USE_SIP
18	PROCESS_OUTGOING_CALLS
19	BODY_SENSORS
20	SEND_SMS
21	RECEIVE_SMS
22	READ_SMS
23	RECEIVE_WAP_PUSH
24	RECEIVE_MMS
25	READ_EXTERNAL_STORAGE
26	WRITE_EXTERNAL_STORAGE

5.3 OPCODES

An opcode is referred as operation code which instructs the system to execute the commanded job. The code optimization of goodwares tend to be better than malwares [5]. Also, the code complexity of goodwares are tend to be greater than the malwares [5]. Moreover, using opcodes of a file even its morphs could be found [4]. Based on these facts opcodes are used as feature for malware detection. Here the method used for quantization of feature is combination of work done by Mahmood Fazlali et al [4] and Gerardo Canfora et al [5].

TABLE 5.2 Opcodes in category wise

MOVE	**IF**	**INVOKE**	**JUMP**	**PACKED SWITCH**	**SPARSE SWITCH**
move-wide	If-eq	invoke-virtual	Throw/range	packed-switch	Spares-switch
move-object/from16	If-ne	invoke-super	goto target	-	-
move-object/16	If-ltz	invoke-direct	goto/16target	-	-
move-exception	If-lt	invoke-static	goto/32target	-	-
move/from	If-gt	invoke-interface	-	-	-
move/16	If-ge	Invoke-e-virtual/range	-	-	-
move-wide/16	If-nez	invoke-direct-empty	-	-	-
move-object	If-lez	invoke-super-quick	-	-	-
move-object/from16	If-le	invoke-super-quick/range	-	-	-

move-result-wide	If-gtz	invoke-quick/range	-	-	-
move-result-object	If-eqz	invoke-static/range	-	-	-
Move	If-gez	invoke-super/range	-	-	-
move-wide/from 16	Check-cast	invoke-interface-range	-	-	-
-	Instance-of	invoke-virtual-quick	-	-	-
-	New-instance	invoke-virtual-quick/range	-	-	-

The table 5.2 shows the six categories of opcodes used for this detector. Six features are selected using opcode data of an APK file. Each of the six feature is quantified as follows:

$$\textbf{Quantized value of each feature} = \frac{\textbf{Total sum of occurrences of opcodes present in their category}}{\textbf{Total sum of occurrences of opcodes present in all six categories}}$$

5.4 API CALLS

An Application Program Interface (API) is a set of procedures and tools for building software applications. These processes make it easier for developers to use certain technologies in building applications. Using the API calls an APK file interact with underlying operating system and other apps. Unlike some researches which use all the available API calls as features here only the API calls which are distinguishably present in malware and goodware are used. According to Westyarian et al [3] total number of API calls called by an application is also a deciding factor. Hence the total number of API calls called by an APK file and six other distinguishable API calls is taken as features. They are listed in table 5.3.

Table 5.3 API calls used as feature

SI	API CALLS USED AS FEATURE
1	android.content.Context
2	android.view.View
3	android.widget.Textview
4	android.content.Intent
5	android.os.Handler
6	android.content.res.Resourses

5.5 API PACKAGES

API calls form a useful mechanism to identify relevant deference between samples with good or non-legal intentions. These calls are grouped by the API packages in which they are defined, in order to depict general characteristic patterns. In general, both types of samples exhibit a similar use of the most common API packages. For instance, theandroid.app and java. lang packages, which are present in all samples, include the most basic functionality of the Android environment and Java language features respectively. An important difference can be found in the android. Telephony groups of API calls [8]. While 83% of the malicious samples invoke calls contained in this package, this number is reduced to 63% in the benign ware set. Hence android.Telephony is chosen as a feature.

5.6 INTENTS

Intents are a key communication element in the Android environment. Basically, they allow to ask permission to the system to run another application component. Thus, Intents describe the actions that an application can take. For instance, an Intent could demand actions such as making a phone call or taking a picture. There are noticeable patterns which can be inferred from this ranking. Only distinguishable intents are used as feature. Table 5.4 shows the features taken based on intents.

Table 5.4 The intents used as feature

SI	INTENTS USED AS FEATURE
1	android.intent.action.BOOT_COMPLETED
2	android.intent.action.USER_PRESENT
3	android.provider.Telephony.SMS_RECEIVED
4	android.intent.action.PACKAGE_ADDED

5.7 CONCLUDING REMARKS

Thus, totally nineteen static features are used to distinguish malware and benign file. The features which are less distinguishable and found in any same amounts between malware and goodwares are avoided. Only the features which are most distinguishable are selected. This can reduce the complexity of the algorithm which will in turn will reduce time and energy for execution. This will also shift more weightage to the features which are more remarkable from the redundant features.

CHAPTER 6
HARDWARE IMPLEMENTATION

6.1 INTRODUCTION

After testing this detector in software, it is implemented in Zynq SoC based board called PYNQ Z2. Implementing in hardware has its own advantages. As smartphones are resource constrained it reduces the burden of phone's main processor. Allocation of separate hardware can increase the speed of execution of a program as it is free from multitasking or any external interrupts. The speed of execution matters when this detector is used for full phone scans which is time consuming. Implementing in HLS based boards such as PYNQ Z2 has the flexibility of accelerating the algorithm in future by using overlays in program.

6.2 PYNQ Z2

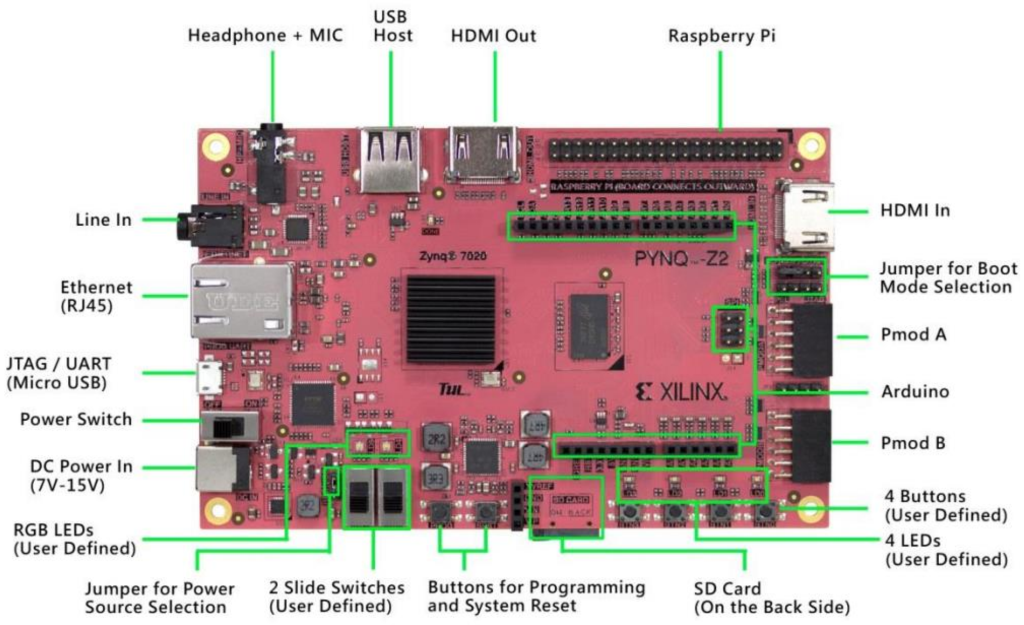


Fig 6.1 Overview of PYNQ-Z2

The PYNQ-Z2 board is based on Xilinx ZYNQ SoC and is designed to support the PYNQ (Python productivity for ZYNQ) framework and embedded systems development. This board is an open source network which enables the embedded programmers to explore the possibilities of Xilinx ZYNQ SoC's without the need to design the programming logic circuits (PLC). The PLC's are imported as hardware libraries and are programmed through API's in the same way that of the software libraries which are imported and programmed. The python language can be used to program the SoC's and the code can be directly tested on the PYNQ-Z2 board. Using the libraries and the python language the designers can exploit the benefits of microprocessors in ZYNQ for building exciting and more capable embedded systems.

PYNQ users can create high performance embedded applications with parallel hardware execution, high frame-rate video processing, hardware accelerated algorithms, real-time signal processing, high bandwidth IO, and low latency control. It is a combination of programmable logic gates and microprocessors which enhances the designer to emphasis new innovative projects in an easier manner as it supports the python language and provides a modern heterogeneity in designing platform as allowing implementing the concepts of machine learning and many more. Figure 6.1 shows the overview of PYNQ Z2.

6.2.1 FEATURES OF PYNQ Z2

PYNQ Z2 has ARM-Cortex A9 processor which is referred to as the Processing System (PS) is integrated with FPGA fabric referred to as Programmable Logic (PL). The PS subsystem includes a number of dedicated peripherals and can be extended with additional hardware IP in a PL Overlay. Figure 6.2 shows the block diagram of PYNQ Z2.

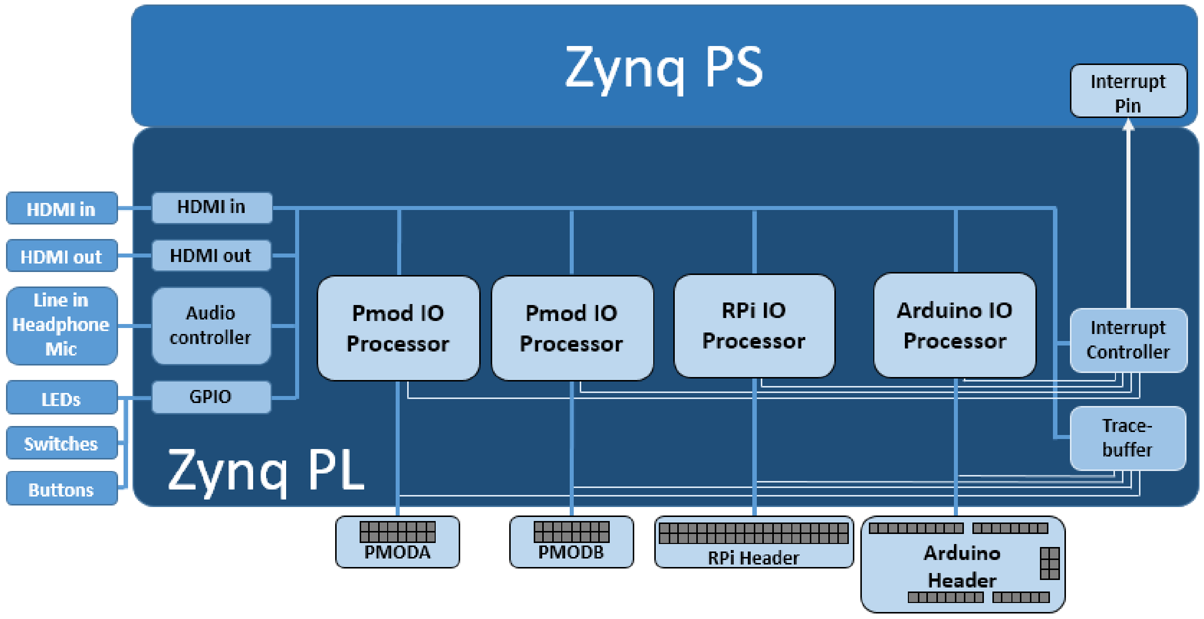


Fig 6.2 Block diagram of PYNQ-Z2

- **PROCESSOR TYPE**

650 MHz clock frequency dual-core Cortex-A9 processor acts as the brain of this hardware. DDR3 memory controller with 8 DMA channels and 4 high performance AX13 Slave ports.

- **HIGH-BANDWIDTH PERIPHERAL CONTROLLERS**

1G Ethernet, USB 2.0, SDIO.

- **LOW-BANDWIDTH PERIPHERAL CONTROLLERS**

SPI, UART, CAN, I2C.

- **POWER**

Powered from USB or 7V-15V external power source.

- **MEMORY AND STORAGE**

512MB DDR3 / 128MB FLASH memory with 16-bit bus at 1050Mbps and16MB Quad-SPI Flash with factory programmed 48-bit globally unique EUI-48/64 compatible identifier. It also consists of a microSD slot.

- **USB AND ETHERNET**

  Gigabit Ethernet PHY, Micro USB-JTAG Programming circuitry, Micro USB-UART bridge, USB 2.0 OTG PHY (supports host only).

- **PROGRAMMABLE LOGIC**

  13,300 logic slices, each with four 6-input LUTs and 8 flip-flops,630KB block RAM, 4 clock management tiles, each with a phase locked loop (PLL) and mixed-mode clock manager (MMCM), 220 DSP slices, On-chip Xilinx analog-to-digital converter (XADC).

- **AUDIO AND VIDEO**

  HDMI sink port (input), HDMI source port (output), I2S interface with 24bit DAC with 3.5mm TRRS jack, Line-in with 3.5mm jack.

- **SWITCHES, PUSH-BUTTONS AND LEDS**

  4 push-buttons,2 slide switches,4 LEDs,2 RGB LEDs.

- **EXPANSION CONNECTORS**

  Two standard Pmod ports, 16 Total FPGA I/O (8 shared pins with Raspberry Pi connector).

- **ARDUINO SHIELD CONNECTOR**

  24 Total FPGA I/O, 6 Single-ended 0-3.3V Analog inputs to XADC.

- **RASPBERRY PI CONNECTOR**

  28 Total FPGA I/O of which 8 are shared pins with Pmod A port.

- **PROGRAMMABLE FROM**

  JTAG, Quad-SPI flash and microSD card.

6.3 BOARD SETUP

Initially the board can set to boot up in several ways like using Pynq image from micro SD card or using any other external memory which has Pynq image and interfacing it using JTAG. To set the board to boot from the micro-SD card, set the “Boot” jumper to the SD position and to power the board from micro USB cable, set the power jumper to the USB position. Insert the micro SD card loaded with

PYNQ-Z2 image into the Micro SD card slot below the board. Connect the PROG-UART micro USB port on the board to power source between 7V to 12V.

Slide the power switch to the ON position to turn on the board. The LED indicating the presence of power source will glow in red to indicate that the board has powered. Two blue LED’s and four general purpose green LED’s LED0 to LED3 will flash simultaneously after a minute indicating that the board has booted up. The blue LED’s will then turn off and on while the general-purpose LED’s will remain on. The board is now ready for use.

6.4 INTERFACING WITH COMPUTER

Once the board is setup, it can be accessed from PC using Ethernet. The PC and board are connected using ethernet cable. Network interface is permitted in PC. PC is allocated a static IP address. The board can be accessed from PC as a network drive. It will ask for authentication where the user name and password are ‘xilinx’. The board can also be accessed by browsing http://192.168.2.99 in Jupyter notebook. 192.168.2.99 is the initial IP address of the board which can be changed if needed.

6.4.1 JUPYTER NOTEBOOK

The Jupyter Notebook is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala. Has leverage big data tools, such as Apache Spark, from Python, R and Scala. It can be used to explore that same data with Pandas, Scikit-learn, ggplot2, TensorFlow.

6.5 OUTPUT FROM PYNQ Z2

The program can be uploaded into the board using Jupyter Notebook. The PYNQ Z2 upon execution of program processes it in its PL and PS and returns the output. Based on the output the peripherals can be controlled using base overlays. In this detector the pattern of general-purpose LEDs in the board are changed based on the final output as malware or goodware. This confirms the execution of the program in PYNQ Z2. Figure 6.3 shows output of the detector in PYNQ Z2.

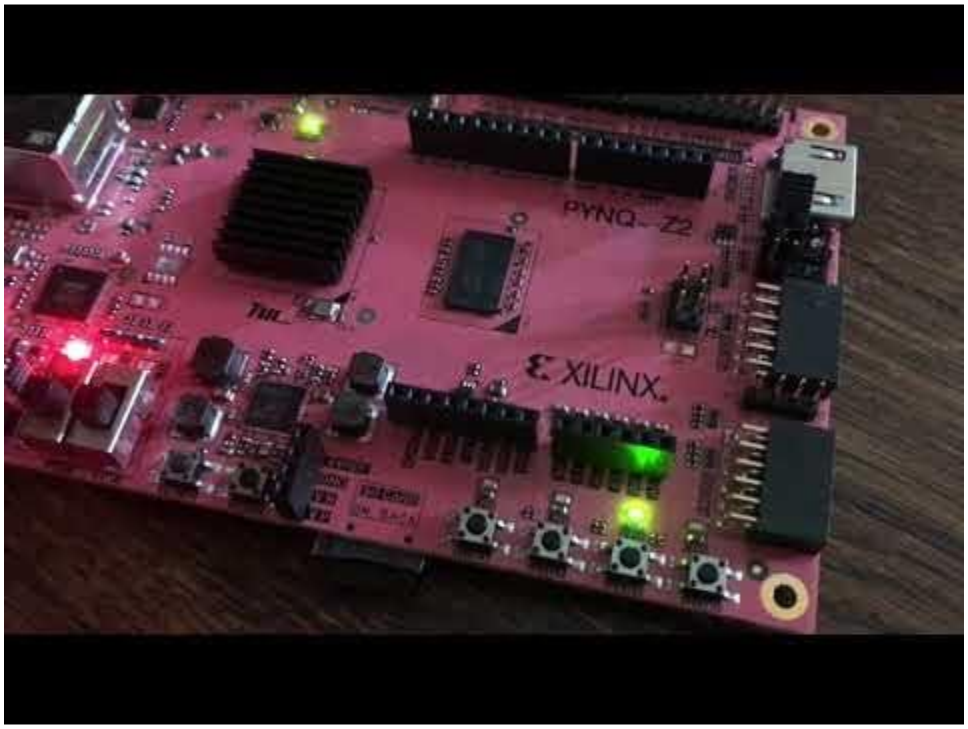


Fig 6.3 Hardware output of the detector

6.6 CONCLUDING REMARKS

The detector is currently implemented in hardware using a Python script without using any overlays. Implementing using overlays can reduce the time of execution by doing parallel processing. All 19 features could be selected from AndroPyTool's output simultaneously in few clock cycles. In future the program could modified to use overlay which parses the JSON file and selects the features simultaneously. The machine learning implementation in FPAG's is at very beginner's stage due to the complexity in the training processes. If this gets sorted it make the detector to bring output in very few clock cycles. All this will be possible in future with the development of efficient techniques to implement the Machine learning algorithms and file parsing algorithms in FPGA's available in the market.

CHAPTER 7
RESULT ANALYSIS

7.1 INTRODUCTION

After a system is built its performance is evaluated using various parameters. Evaluation of a system is important to understand the system better and make comparison with the other existing systems. A system can be decided to use for a specific application by analysing its test results. Further it gives room for improvement if the test results are not satisfactory. This malware detector is evaluated by testing for various parameters and their results are presented.

7.2 ALGORITHM ANALYSIS

Algorithm analysis employs the evaluation of detection results of the detector. For sake of analysis the known dataset is divided into test data and train data in ratio of 3:7 respectively. This ratio is an industry standard. This split is implemented by using 'train_test_split' function of Sci-kit learn library. This function will randomly split the dataset into test data and training data of 3:7 ratio respectively. The train dataset will then train the classifiers and the test data are used as input data to the classifiers. The classifiers predictions result is compared with the known result of these test data. To evaluate the parameters with high precision the output of the classifiers is taken for 100 times and the mean value is taken. Each time the dataset is split in same ratio but the samples are randomly allocated as test or training data by the 'train_test_split' function. Therefore, the test and train dataset will be different during each time of execution and this will ensure the diversity of data. Following are some of the parameters using which the classifiers of the detector can be evaluated.

7.2.1 ACCURACY

Accuracy of a detector is the ratio of total number of samples correctly detected by the detector to the total number of samples analysed by the detector. It can also be expressed in terms of percentage.

7.2.2 FALSE POSITIVE RATE

The false positive rate of a detector is the ratio of the total number of wrongly detected samples as malware to the total number of detected samples as malware by the detector.

7.2.3 FALSE NEGATIVE RATE

The false negative rate of a detector is the ratio of the total number of wrongly detected samples as goodware to the total number of detected samples as goodware by the detector.

7.2.4 TRUE POSITIVE RATE

The true positive rate of a detector is the ratio of the total number of correctly detected samples as malware to the total number of detected samples as malware by the detector.

7.2.5 TRUE NEGATIVE RATE

The true negative rate of a detector is the ratio of the total number of correctly detected samples as goodware to the total number of detected samples as goodware by the detector.

7.2.6 CONFUSION MATRIX

The Confusion matrix is a summary of prediction results on classification problem. The number of correct and incorrect predictions are summarised with count values and broken down by each class. The Confusion matrix shows the ways in which your classification model is confused when it makes prediction. It gives insight on types of error made by the classifier.

7.3 EVALATION RESULTS

The table 7.1 shows the results of the parameters using which the classifiers are evaluated for seven classifiers. Totally seven classifiers along with the Tri-Classifier which is employed in this detector are evaluated. Each result is the mean of outputs gathered through executing each classifier 100 times each classifier with same dataset and same test-train split cycle so that during each time of execution all classifiers are trained with same dataset and take same test data as input. Therefore, the classifiers can be compared without worrying about the effectiveness of train data and test data.

Table 7.1 Classifiers mean performance

CLASSIFIER	ACCURACY	FNR	FPR	TPR	TNR
Random forest	0.82893333	0.191564	0.138695	0.861305	0.808436
Decision Tree	0.74506667	0.235009	0.270196	0.729804	0.764991
Logistic Regression	0.7804	0.232654	0.197765	0.802235	0.767346
Naïve Bayes	0.6894	0.2393	0.329339	0.670661	0.7607
SVM	0.7794	0.238877	0.189317	0.810683	0.761123

K-means	0.7916	0.227103	0.177628	0.822372	0.772897
Tri-Classifier	0.81973333	0.189482	0.164078	0.835922	0.810518

The table 7.2 shows the confusion matrix of seven classifiers along with the Tri-classifier which is used for this detector. As the dataset consists of 248 APK file samples and the test and train data are split in 3:7 ratio respectively, on each execution the 75 APK files which are randomly selected from the dataset, are given as input to classifiers as test data. Remaining 173 APK file samples are used as training data to the classifier. For the purpose of obtaining the confusion matrix these classifiers are executed with same test and training data at 3:7 ratio for one time. This implemented using pandas.crosstab() object from Pandas library after converting the output of the classifiers into Pandas serial datatype.

Table 7.2 Confusion matrix of the classifiers

I	TEST SAMPLES=75	TRUE GOODWARE	TRUE MALWARE
NAÏVE BAYES	DETECTED AS GOODWARE	38	5
	DETECTED AS MALWARE	18	14
II	TEST SAMPLES=75	TRUE GOODWARE	TRUE MALWARE
SUPPORT VECTOR MACHINE	DETECTED AS GOODWARE	34	9
	DETECTED AS MALWARE	10	22
III	TEST SAMPLES=75	TRUE GOODWARE	TRUE MALWARE
K-MEANS	DETECTED AS GOODWARE	34	9
	DETECTED AS MALWARE	11	21
IV	TEST SAMPLES=75	TRUE GOODWARE	TRUE MALWARE
DECISION TREE	DETECTED AS GOODWARE	28	5
	DETECTED AS MALWARE	17	25

V	TEST SAMPLES=75	TRUE GOODWARE	TRUE MALWARE
LOGISTIC REGRESSION	DETECTED AS GOODWARE	37	6
	DETECTED AS MALWARE	11	21
VI	TEST SAMPLES=75	TRUE GOODWARE	TRUE MALWARE
RANDOM FOREST	DETECTED AS GOODWARE	39	4
	DETECTED AS MALWARE	8	24
VII	TEST SAMPLES=75	TRUE GOODWARE	TRUE MALWARE
TRI-CLASSIFIER	DETECTED AS GOODWARE	38	7
	DETECTED AS MALWARE	6	24

The figures 7.1 and 7.2 shows graphically the false negative rate and accuracy of the seven classifiers along with the Tri-Classifier which is used in this detector. These values plotted from Table 7.1.

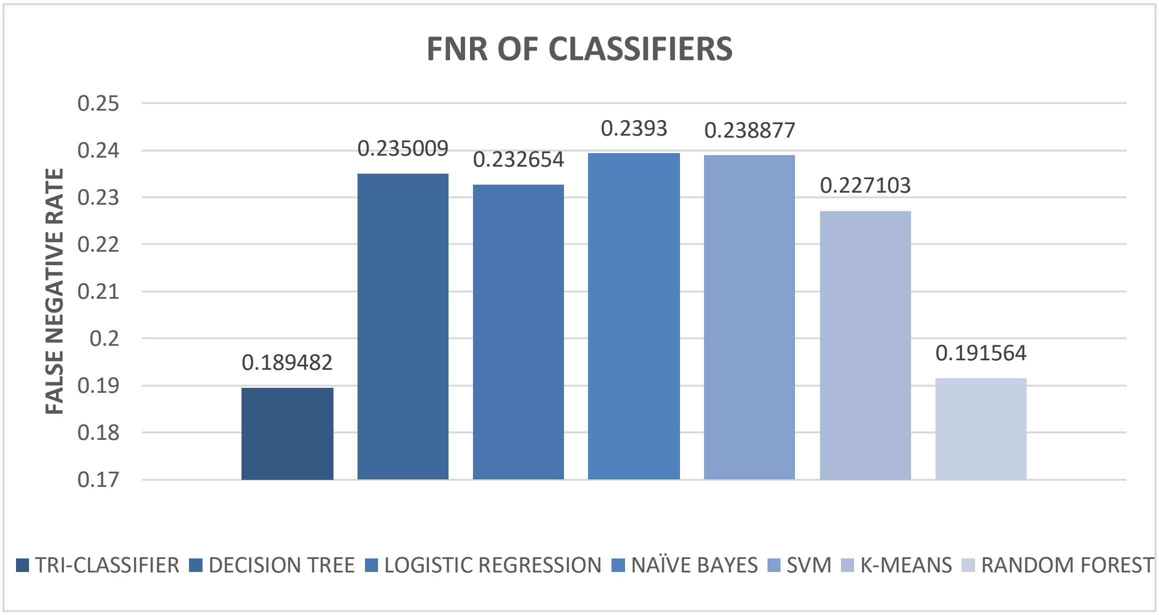


Fig 7.1 FNR of Classifiers

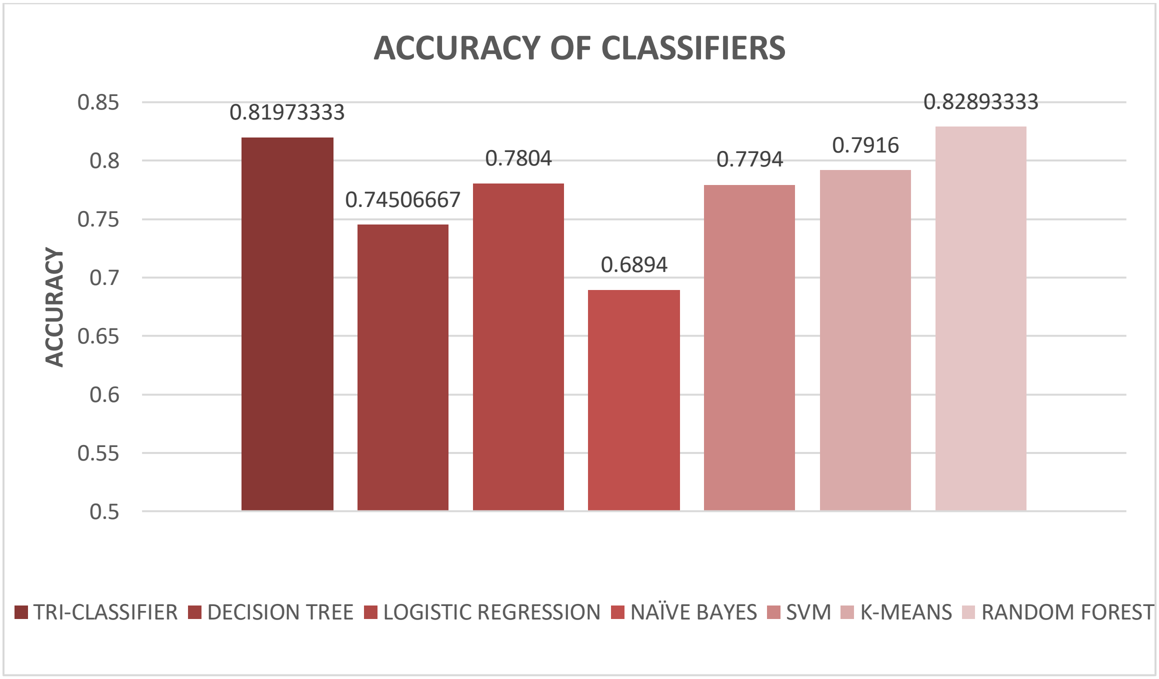


Fig 7.2 Accuracy of the classifiers

7.4 INFERENCE FROM EVALUATION RESULTS

From the software evaluation results it is evident that the Tri-classifier has better accuracy than Decision tree classifier, Naïve Bayes classifier, K-means classifier, Logistic regression and Support vector machine by at least by 2%. It falls short to the accuracy of Random forest classifier just by 0.9% whereas other six classifiers fall short to the accuracy of Random forest by at least 3%. In other hand the false negative rate of Tri-Classifier is the least among these seven classifies listed here. Hence Tri-Classifier can be used for any application which needs accuracy close to Random forest classifier at the same time should have false negative rate lower than the Random forest classifier. Here in this case the application aims to provide security from malware in Android platform which is mostly used for non-professional but personal purposes. Therefore, false negative rate is held at higher priority than accuracy as the chance of detecting any goodware as malware is lower. Tri-Classifier serves this purpose well.

7.5 CONCLUDING REMARKS

The accuracy of the classifier could be increased using better training data sample in more number. As the number of the accurate training data sample increases the accuracy of the classifier also increases. As mentioned in literature survey many studies used thousands of training data sample. With the growth of technology, the faster internet and cloud storage becomes more feasible. Due to this a large training data of terabytes range can be held at cloud storage and used for training of classifiers.

APPENDIX

The following is python code for feature selection from AndroPytool's output and classification using Tri-classifier written by own.

```
import json

#getting the path of the AndroPyTool output file

print("\n\t\t\t\t\t  TRI-CLASSIFIER MALWARE DETECTOR\n\n")

path =input("ENTER THE PATH: ")+'.json'

#JSON file to dictionary format

raw_data=json.load(open(path))

keywords_x = raw_data.keys()

keywords=[]

global output

output = []

for key in keywords_x:

    keywords.append(key)

given = keywords[0]

if given == 'Pre_static_analysis':

    pre_static_data = raw_data['Pre_static_analysis']

    static_data = raw_data['Static_analysis']

else:

    root_data = raw_data[given]

    pre_static_data = root_data['Pre_static_analysis']

    static_data = root_data['Static_analysis']

apk_name=pre_static_data['Filename']
```

```
#extracting the features regarding the permissions

def permission_f(static_data_x):

   permission_output= static_data_x['Permissions']

   f1 = 'android.permission.CAMERA' in permission_output

   f2 = 'android.permission.READ_CALENDAR' in permission_output

   f3 = 'android.permission.WRITE_CALENDAR' in permission_output

   f4 = 'android.permission.READ_CONTACTS' in permission_output

   f5 = 'android.permission.WRITE_CONTACTS' in permission_output

   f6 = 'android.permission.GET_ACCOUNTS' in permission_output

   f7 = 'android.permission.ACCESS_FINE_LOCATION' in permission_output

   f8 = 'android.permission.ACCESS_COARSE_LOCATION' in permission_output

   f9 = 'android.permission.RECORD_AUDIO' in permission_output

   f10 = 'android.permission.READ_PHONE_STATE' in permission_output

   f11 = 'android.permission.READ_PHONE_NUMBERS' in permission_output

   f12 = 'android.permission.CALL_PHONE' in permission_output

   f13 = 'android.permission.ANSWER-PHONE_CALLS' in permission_output

   f14 = 'android.permission.READ_CALL_LOG' in permission_output

   f15 = 'android.permission.WRITE_CALL_LOG' in permission_output

   f16 = 'android.permission.ADD_VOICEMAIL' in permission_output

   f17 = 'android.permission.USE_SIP' in permission_output

   f18 = 'android.permission.PROCESS_OUTGOING CALLS' in permission_output

   f19 = 'android.permission.BODY_SENSORS' in permission_output

   f20 = 'android.permission.SEND_SMS' in permission_output

   f21 = 'android.permission.RECEIVE_SMS' in permission_output
```

```
  f22 = 'android.permission.READ_SMS' in permission_output
  f23 = 'android.permission.RECEIVE_WAP_PUSH' in permission_output
  f24 = 'android.permission.RECEIVE_MMS' in permission_output
  f25 = 'android.permission.READ_EXTERNAL_STORAGE' in permission_output
  f26 = 'android.permission.WRITE_EXTERNAL_STORAGE' in permission_output
  yy =[f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14, f15, f16, f17, f18, f19, f20, f21,
f22, f23, f24, f25, f26]
  permissions =0
  for i in yy:
    if i == True:
      permissions = permissions+1
    else:
      permissions = permissions+0
  output.append(permissions)
#Extracting the features regarding the Op-codes
def opcodes_f(static_data_x):
   op = static_data_x['Opcodes']
   try:
     m11 =op['move']
   except KeyError:
     m11 =0
   try:
     m12 =op['move/from']
   except KeyError:
     m12 =0
```

```
try:

  m13 =op['move/16']

except KeyError:

  m13 =0



try:

  m14 =op['move-wide']

except KeyError:

  m14 =0

try:

  m15 =op['move-wide/from16']

except KeyError:

  m15 =0

try:

  m16 =op['move-wide/16']

except KeyError:

  m16 =0

try:

  m17 =op['move-object']

except KeyError:

  m17 = 0

try:

  m18 =op['move-object/from16']

except KeyError:
  m18 = 0

try:

  m19 =op['move-object/16']

except KeyError:

  m19 = 0

try:

  m20 =op['move-result-wide']

except KeyError:

  m20 = 0

try:

  m21 =op['move-result object']

except KeyError:

  m21 = 0

try:

  m22 =op['move-exception']

except KeyError:

  m22 =0

try:

  p11 =op['packed-switch']

except KeyError:

  p11 = 0

try:

  s11 =op['sparse-switch']

except KeyError:
  s11 = 0
try:
  i11 =op['if-eq']
except KeyError:
  i11 = 0
try:
  i12 =op['if-ne']
except KeyError:
  i12 = 0
try:
  i13 =op['if-lt']
except KeyError:
  i13 =0
try:
  i14 =op['if-ge']
except KeyError:
  i14 = 0
try:
  i15 =op['if-gt']
except KeyError:
  i15 = 0
try:
  i16 =op['if-le']
except KeyError:
  i16 = 0
try:
  i17 =op['if-eqz']
except KeyError:
  i17 = 0


try:
  i18 =op['if-nez']
except KeyError:
  i18 = 0
try:
  i19 =op['if-ltz']
except KeyError:
  i19 = 0
try:
  i20 =op['if-gez']
except KeyError:
  i20 = 0
try:
  i21 =op['if-gtz']
except KeyError:
  i21 =0
try:
  i22 =op['if-lez']
```

```
except KeyError:
  i22 =0
try:
  i23 =op['check-cast']
except KeyError:
  i23 = 0
try:
  i24 =op['instance-of']
except KeyError:
  i24 =0
try:
  i25 =op['new-instance']
except KeyError:
  i25 =0
try:
  iv11 =op['invoke-virtual']
except KeyError:
  iv11 =0
try:
  iv12 =op['invoke-super']
except KeyError:
  iv12 = 0
try:
  iv13 =op['invoke-direct']
```

```
except KeyError:
  iv13 =0
try:
  iv14 =op['invoke-static']
except KeyError:
  iv14 =0
try:
  iv15 =op['invoke-interface']
except KeyError:
  iv15 =0
try:
  iv16 =op['invoke-virtual/range']
except KeyError:
  iv16 =0
try:
  iv17 =op['invoke-super/range']
except KeyError:
  iv17 =0
try:
  iv18 =op['invoke-direct/range']
except KeyError:
  iv18 =0
try:
  iv19 =op['invoke-static/range']
```

```
except KeyError:
  iv19 =0
try:
  iv20 =op['invoke-interface-range']
except KeyError:
  iv20 =0
try:
  iv21 =op['invoke-virtual-quick/range']
except KeyError:
  iv21 =0
try:
  iv22 =op['invoke-super-quick']
except KeyError:
  iv22 =0
try:
  iv23 =op['invoke-direct-empty']
except KeyError:
  iv23 =0
try:
  iv24 =op['invoke-vritual-quick']
except KeyError:
  iv24 =0
try:
  iv25 =op['invoke-super-quick/range']
```

```
    except KeyError:
      iv25 =0
    try:
      j11 =op['throw']
    except KeyError:
      j11 = 0
    try:
      j12 =op['goto/target']
    except KeyError:
      j12 =0
    try:
      j13 =op['goto/16target']
    except KeyError:
      j13 =0
    try:
      j14 =op['goto/32target']
    except KeyError:
      j14 = 0
    total_op_c =[m11, m12 ,m13 ,m14 ,m15 ,m16 ,m17 ,m18 ,m19 ,m20 ,m21 ,m22 ,p11 ,s11
,i11 ,i12 ,i13 ,i14 ,i15 ,i16 ,i17 ,i18 ,i19 ,i20 ,i21 ,i22 ,i23 ,i24 ,iv11, iv12, iv13 ,iv14, iv15,
iv16, iv17, iv18, iv19,iv20, iv21, iv22, iv23, iv24, iv25, j11, j12, j13, j14]
    TOTAL_OP_C = 0
    for x in total_op_c:
       TOTAL_OP_C = TOTAL_OP_C + x
    m = 0
```

```
i = 0
p = p11
s = s11
iv= 0
j = 0
tm = [m11, m12 ,m13 ,m14 ,m15 ,m16 ,m17 ,m18 ,m19 ,m20 ,m21]
ti = [i11 ,i11 ,i12 ,i13 ,i14 ,i15 ,i16 ,i17 ,i18 ,i19 ,i20 ,i21 ,i22 ,i23 ,i24]
tiv= [iv11, iv12, iv13 ,iv14, iv15, iv16, iv17, iv18, iv19,iv20, iv21, iv22, iv23, iv24, iv25]
tj = [j11, j12, j13, j14]
for x in tm :
   m = m + x
for x in ti :
   i = i + x
for x in tiv :
   iv = iv + x
for x in tj :
   j = j + x
total = 100/TOTAL_OP_C
om = total*m
oi = total*i
op = total*p
os = total*s
oiv= total*iv
oj = total*j
```

```
    output.append(om)
    output.append(oi)
    output.append(op)
    output.append(os)
    output.append(oiv)
    output.append(oj)
#Extracting the features regarding API calls
def API_CALLS(static_data_x):
    yp= static_data_x['API calls']
    api_total_count= 0
    for x in yp:
      api_total_count = api_total_count +1
    try:
      a11 = yp['android.content.Context']
    except KeyError:
      a11 = 0
    try:
      a12 = yp['android.view.View']
    except KeyError:
      a12 = 0
    try:
      a13 = yp['android.widget.Textview']
    except KeyError:
      a13 = 0
    try:
      a14 = yp['android.content.Intent']
    except KeyError:
      a14 = 0
    try:
      a15 = yp['android.os.Handler']
    except KeyError:
      a15 = 0
    try:
      a16 = yp['android.content.res.Resourses']
    except KeyError:
      a16 = 0
    output.append(api_total_count)
    output.append(a11)
    output.append(a12)
    output.append(a13)
    output.append(a14)
    output.append(a15)
    output.append(a16)
#Extracting the features regarding API packages and Intents
def api_intent(static_data_x):
   yp= static_data_x['API packages']
   zp = static_data_x['Intents']
```

```
try:
  api_packages =yp['android.telephony']
except KeyError:
  api_packages = 0


try:
  ap11 = zp['android.intent.action.BOOT_COMPLETED']
except:
  ap11 = 0
try:
  ap12 = zp['android.intent.action.USER_PRESENT']
except:
  ap12 = 0
try:
  ap13 = zp['android.provider.Telephony.SMS_RECEIVED']
except:
  ap13 = 0
try:
  ap14 = zp['android.intent.action.PACKAGE_ADDED']
except:
  ap14 = 0
output.append(api_packages)
output.append(ap11)
output.append(ap12)
```

```
  output.append(ap13)
  output.append(ap14)
#Collecting all 19 features in list
permission_f(static_data)
opcodes_f(static_data)
API_CALLS(static_data)
api_intent(static_data)
data = output
#Converting the list of features into pandas dataframe
import pandas as pd
header = {'permission':data[0], 'op-move':data[1], 'op-packed switch':data[2], 'op-sparced
switch':data[3], 'op-if':data[4], 'op-invoke':data[5], 'op-jump':data[6], 'total API calls':data[7],
'contaxt':data[8], 'view':data[9], 'textview':data[10], 'intent':data[11], 'handler':data[12],
'res.resourse':data[13], 'telepony':data[14], 'BOOT_COMPLET':data[15],
'USER_PERMISSION':data[16], 'SMS_RECEIVED':data[17],
'PACKET_ADDED':data[18]}
kk=pd.DataFrame(header,index=[0])
#Implementing the Classification algorithms
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
import datetime
from openpyxl import load_workbook
df = pd.read_csv('train_data_set.csv')
x = df.drop(columns=['malware'])
```

```
y = df['malware']
v = kk
module =RandomForestClassifier()
module.fit(x, y)
pd= module.predict(v)
if pd == 1:
        print('\t\t!!!"'+apk_name+'" is detected as malware!!!\nDeletion of this file is
recommended')
        ty='MALWARE'
         t=1
elif pd ==0:
        module =DecisionTreeClassifier()
        module.fit(x, y)
        qw= module.predict(v)
        x= preprocessing.scale(x)
        module =LogisticRegression()
        module.fit(x, y)
        er= module.predict(v)
        if qw*er==1:
                print('\t\t!!!"'+apk_name+'" is detected as malware!!!\nDeletion of this file is
recommended')
                ty='MALWARE'
                 t=1
        else:
                print('\t\t"'+apk_name+'" is Fine!')
        ty='GOODWARE'
         t=0
#Storing the output in Excel workbook
time=datetime.datetime.now()
b=[apk_name,time,ty]
data=b+data
wb= load_workbook("RTK_output.xlsx")
ws= wb.worksheets[0]
ws.append(data)
wb.save("RTK_output.xlsx")
#Changing the glowing LED pattern in PYNQ Z2 board based on the output of the classifier.
from pynq.overlays.base import BaseOverlay
from pynq.lib import  LED, Button, Switch
base = BaseOverlay("base.bit")
led0 = base.leds[0]
led1 = base.leds[1]
led2 = base.leds[2]
led3 = base.leds[3]
if t == 1:
  led0.off()
  led1.on()
  led2.on()
  led3.off()
```

```
elif t ==0:
  led0.on()
  led1.off()
  led2.off()
  led3.on()

path = input('PRESS ENTER KEY TO EXIT')
```

CONCLUSION

Malwares are ever evolving. For any newly developed detector malware developers tend to find new methods to evade the detection. Hence the technique used for detection must be ever evolving. So the techniques mentioned here must also be evolved. With the advancement of technology, capabilities of smart phones are increasing equivalent to personal computer. Today most of the smart phones are available with 4GB or more RAM and expandable storage more than 128GB. Dynamic analysis which requires extensive resources and capabilities of the processor as it runs an application in a sandbox environment to protect other applications from it. Hence dynamic analysis is not so effective in most of the today's commonly used smart phones. This situation may be changed in following years with advent of the technology. Hence our future work may be based on features based on dynamic analysis. Instead of using third party application a reverse engineering tool purely based on python without any external influence could be developed. This increase the portability and effectiveness of the classifier reducing the number of instructions. Methodologies for feature selection could be enhanced. With increase of number of features after a threshold the accuracy of classifier tends to be decreasing. This is because with so much features the weigthage of features which are more distinguishable decreases when compared to lesser distinguishable features. A new algorithm could be developed to increase the number of features as well as increasing the accuracy of the classifier. Malware writers develop methodologies to evade detector by understanding that detector. Hence more security is need for any malware detection technique from malware developers. Security can be enhanced by using dedicated hardware for malware detector instead of sharing the main processor using software. This is because the it hard to analyze the functioning of the detector by looking at a silicon chip than looking at code used by the processor. Currently we have used python programming language to program PYNQ Z2. This program can be developed as an overlay and used with other useful ovelays which would result in parallel processing of these overlays. Moreover, using hardware definition languages like Verilog or VHDL rather than programming language can lead to parallel processing within an overlay. These codes could also used to manufacture an application specific integrated circuit (ASIC) specific to malware detection. These improvements can increase the reliability and performance of malware detection for mobile phones.

REFERENCES

1. Mohammed Alam et al s, “Random Forest Classification for Detecting Android Malware”, IEEE International conference on Green Computing and Communications and IEEE Internet of things,October,2013.

2. Naser Peiravian et al, “Machine Learning for Android Malware Detection Using Permission and API Calls”, IEEE 25th International conference on tools with Artificial Intelligence,September,2013.

3. Westyarian et al, “Malware Detection on Android Smart phones using API Class and Machine Learning”, The 5th International Conference on Electrical Engineering and Informatics, Bali ,Indonesia,October,2015.

4. Mahmood Fazlali et al, “An efficient algorithm for Detecting Metamorphic Malware by using Opcode Frequency Rate and Decision Tree”, International journal of information security and privacy, July 2016.

5. Gerardo Canfora et al, “Mobile Malware Detection Using Op-code Frequency Histogram”, Conference paper on mobile security, May 2015.

6. Pankaj Kohli et al, “Signature Generation and Detection of Malware Families”, Center for Security, Theory and Algorithmic Research (C-STAR),May 2016.

7. Richard Killam et al, “Android Malware Classification through analysis of string literals”, University of New Brunswick

8. Alejadro Martin et al, “Android malware detection through hybrid features fusion and ensemble classifiers: The andropytool framework and omnidroid dataset”, article in Information Fusion, December 2018.

9.List of normal and critical android permissions, Available: https://stackoverflow.com/questions/36936914/list-of-android-permissions-normal-permissions-and-dangerous-permissions-in-api

10. List of Dalvik opcodes used in android Available: http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

11.Malware malware samples repository Available: https://github.com/ashishb/android-malware

12.Benign android file samples Available: https://m.apkpure.com/