

# Group-wise Classification Approach to Improve Malware Detection Accuracy

THESIS

submitted in partial fulfilment  
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

**Ashu Sharma**

under the supervision of

**Dr. Sanjay K. Sahay**



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE**

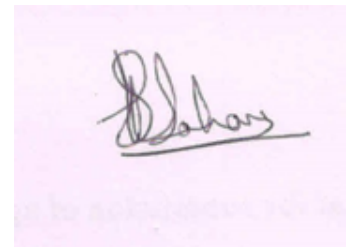
**PILANI (Rajsthan) INDIA**

**2018**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE  
PILANI (RAJASTHAN)**

**CERTIFICATE**

This is to certify that the thesis entitled **Group-wise Classification Approach to Improve Malware Detection Accuracy** submitted by **Ashu Sharma** ID No. **2012PHXF0011G** for award of Ph.D. of the Institute, embodies original work done by her under my supervision.

A handwritten signature in black ink on a light pink background. The signature is stylized and appears to read 'S. Sahay'.

Signature of the Supervisor

Name : Dr. SANJAY KUMAR SAHAY

Designation : Assistant Professor

Date: 6/07/2018

## ACKNOWLEDGEMENT

I would like to acknowledge the love and mercy of the **Divine Providence** in the entire work who helped to overcome all the obstacles by guiding and providing me the needed help through various people, and whose faithfulness is incomprehensible.

I would like to thank my supervisor **Dr. Sanjay K. Sahay** who encouraged and directed me towards the research. He supported me throughout my thesis with his patience and knowledge whilst allowing me to work in my own way. I could not have imagined having such a better and friendly supervisor for my Ph.D. work.

I am thankful to BITS, Pilani, K.K. Birla Goa Campus for the Ph.D. scholarship No. Ph603226/July 2012/01. I also take this opportunity to thank Prof. G. Raghurama, Director, Birla Institute of Technology and Science Pilani, K K Birla Goa Campus, Prof. Souvik Bhattacharyya, Vice Chancellor, BITS, Pilani, Dr. Kumar Mangalam Birla, Chancellor, BITS, Pilani, for giving me the opportunity to pursue research for the Ph.D. degree. My heartfelt gratitude to former directors Prof. Sasikumar Punnekkat, and late Prof. Sanjeev K. Aggarwal, BITS, Pilani K K Birla Goa Campus for their motivation and encouragement.

I wish to thank the Department Research Committee members and the Convenor **Dr. A. Baskar**. I also express my sincere thanks to **Prof. S. K. Verma**, Dean, Academic Research (Ph.D. Programme), BITS, Pilani, **Dr. Prasanta Kumar Das**, Associate Dean, Academic Research, BITS, Pilani K K Birla Goa Campus for providing the necessary facilities.

My sincere thanks also goes to my Doctoral Advisory Committee members **Dr. Bharat Deshpande**, Head of the Department, Computer Science and Information Systems and **Neena Goveas**, Associate Professor, Computer Science and Information Systems, BITS Pilani K K Birla Goa Campus for preliminary assessment of the thesis and helpful suggestions during the seminars. I also thank the faculty members, research scholars and the staffs of our department for their kind cooperation.

I heartily thank Ex-Director Prof. Ajit Kembhavi and Dean Visitor Prof. Kandaswamy Subramanian, Inter-University Center for Astronomy and Astrophysics (IUCAA), Pune for providing hospitality and computation facility at the initial stage of my research. Also, I would like to thank **Malicia** project for the Windows malware dataset and Technische Universitat Braunschweig for providing the **Drebin** (Android malware) dataset for the research work.

I heartily thank my friends for their prayers and constant encouragement to finish this work successfully. I am also thankful to my fellow colleagues whose challenges and productive criticism, especially to the progress of my research, have provided improvement in the presentation of the work.

Finally, this thesis is heartily dedicated to my father Shiv Autar, mother Malti Sharma, brother Rishi Sharma and sister Madhu Sharma who all the time stood with me.

**ASHU SHARMA**

## ABSTRACT

In today's information era, most of the computational devices are connected to the Internet, as a consequence, these devices are very much vulnerable to the cyber threat from the advanced malware. It can penetrate networks, steal confidential information from desktops and smart devices, bring down servers and can cripple infrastructures etc. To combat the threat/attacks from the malware, anti-malware have been developed. The existing anti-malware are mostly based on the assumption that the malware structure does not change appreciably. But the recent advancement in second generation malware which can create millions of its variants have posed a challenge to anti-malware developers, and it is an indisputable fact that the traditional approach to combat the threats/attack from the second generation malware with a technology-centric is ineffective to detect today's highly sophisticated customized malware. Therefore, this thesis primarily focuses to improve the detection accuracy of unknown malware by group-wise classifying the Windows Desktops executables and Android apps using machine learning techniques. Accordingly, we, discusses the types of malware and its detection techniques, and the prior works/efforts done by the researchers to detect Windows Desktops and Android based devices malware.

The metamorphic malware variants lead to a huge signature database for the detection by traditional signature based techniques. Therefore, we investigate the variation in the size of malware generated by metamorphic malware generator kits viz. G2, PS-MPC and NGVCK and observed that the variation in the size of malware generated from the same kit is within 5 Kbytes. Hence, accordingly we partitioned the data set in the range of 5 KB size to detect the unknown malware by Naive Bayes classifier, and the obtained results are compared with the regular method. We find that by group-wise partitioning the data set in the range of 5 KB size, the detection accuracy is  $\sim 8.7\%$  more accurate than the regular method (without grouping the executables).

To find the best classifier for the detection of unknown malware, with Malicia data set we studied the performance of the popular thirteen classifiers using N-fold

cross validation available in machine learning tool WEKA. Among these thirteen classifiers, we studied in-depth top five classifiers (Random forest, Logistic model tree, Naive Bayesian tree, J48 and Functional Tree) and obtained more than 96.28% accuracy for the detection of unknown malware, which is better than the detection accuracy (95.9%) reported by Santos et al. (2013). In this top five classifiers, our approach obtained 97.95% detection accuracy by the Random forest. As we observed that by grouping the data, detection accuracy increases, therefore further we grouped the executables on the basis of malware sizes by using optimal k-Means clustering algorithm, and classified the data by the top five classifiers.

Android based smart devices are growing exponentially and are connected through the Internet accessing billion of online websites. The popularity of these devices encourages malware developer to penetrate the market with malicious apps to annoy and disrupt the victim. Although, for the detection of malicious apps different approaches are discussed. However, proposed approaches are not sufficient to detect the advanced malware to limit/prevent the damages. In this, very few approaches are based on the opcode occurrence to classify the malicious apps. Therefore, with the benchmark *Drebin* dataset, first we investigate the five classifiers using opcodes occurrence as the prominent features for the detection of malicious apps and found that FT detection accuracy (79.27%) is best among the investigated classifiers and the overall accuracy is majorly affected by the false positives. From the previous investigation we concluded that group-wise analysis results in high malware detection accuracy. Therefore, further we experimentally demonstrated how to improve the detection accuracy by group-wise analyzing the Android apps. The analysis shows that by group-wise classifying the data based on permissions, improves the overall average accuracy, and can be achieved up to 97.15% . The obtained results outperform the accuracy obtained without grouping data, Arp, et al. (94%, 2014), Annamalai et al. (84.29%, 2016), Bahman Rashidi et al. (82%, 2017)) and Ali Feizollah, et al. (95.5%, 2017). The analysis also shows that among the groups, MICROPHONE group detection accuracy is least while CALENDAR group apps are detected with the highest accuracy.

# CONTENTS

ACKNOWLEDGEMENT . . . . .	iii
ABSTRACT . . . . .	v
LIST OF TABLES . . . . .	x
LIST OF FIGURES . . . . .	x
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Motivation . . . . .	5
1.3 Research Gap . . . . .	7
1.4 Objectives and Organisation of the Thesis . . . . .	9
1.5 Contribution . . . . .	11
<b>2 Types of Malware and Detection Techniques</b>	<b>12</b>
2.1 Introduction . . . . .	12
2.2 Types of Malware . . . . .	12
2.3 Detection Techniques . . . . .	17
2.3.1 Signature Based Detection . . . . .	18

2.3.2	Heuristics Based Detection . . . . .	18
2.3.3	Malware Normalization . . . . .	19
2.3.4	Machine Learning Techniques . . . . .	20
2.4	Summary . . . . .	20
<b>3</b>	<b>Literature Survey</b>	<b>21</b>
3.1	Introduction . . . . .	21
3.2	Survey on the Detection of Windows Desktops Malware . . . . .	21
3.3	Survey on the Detection of Android Malicious Apps . . . . .	27
3.4	Summary . . . . .	32
<b>4</b>	<b>Group-wise Classification for the Detection of Windows Desktops Malware</b>	<b>33</b>
4.1	Introduction . . . . .	33
4.2	Data Preprocessing . . . . .	34
4.3	Feature Selection . . . . .	35
4.4	Detection of Unknown Malware . . . . .	46
4.4.1	Naive Bayes classifier . . . . .	47
4.4.2	Regular Method . . . . .	47
4.4.3	Group-wise Partitioning the Datasets . . . . .	49
4.5	Summary . . . . .	51
<b>5</b>	<b>Classifiers Selection and K-mean Clustering to Improve the Detection</b>	



<b>Accuracy</b>	<b>52</b>
5.1 Introduction . . . . .	52
5.2 Classifiers selection . . . . .	53
5.3 Result Analysis . . . . .	55
5.4 Improving the Detection Accuracy by Group-wise Classification using Optimal K-mean Clustering Algorithm . . . . .	58
5.5 Summary . . . . .	64
<b>6 Group-wise Classification for the Detection of Android Malicious Apps</b>	<b>65</b>
6.1 Introduction . . . . .	65
6.2 Data Preprocessing and Feature Selection . . . . .	66
6.3 Classification Without Grouping the Apps . . . . .	69
6.4 Grouping of Android Apps . . . . .	73
6.5 Group-wise Classification of Android Malicious Apps . . . . .	75
6.6 Summary . . . . .	93
<b>7 Conclusions and Future Directions</b>	<b>94</b>
<b>A Obfuscation Techniques</b>	<b>98</b>
A.1 Register Renaming . . . . .	98
A.2 Subroutine Permutation . . . . .	99
A.3 Instruction Level Permutation . . . . .	99
A.4 Insertion of Jump Instructions . . . . .	100

A.5 Subroutine Inlining and Outlining . . . . .	101
A.6 Dead Code Insertion . . . . .	101
A.7 Equivalent Code Substitution . . . . .	102
<b>B Windows Desktops Opcode List</b>	<b>103</b>
<b>C Android Opcode List</b>	<b>115</b>
<b>D Brief Description of the Major Classifiers</b>	<b>120</b>
D.1 J48 . . . . .	120
D.2 Random Forest . . . . .	121
D.3 Naive Bayes Trees . . . . .	121
D.4 Logistic Model Trees . . . . .	122
D.5 Functional Tree . . . . .	122
LIST OF PUBLICATIONS . . . . .	135
BRIEF BIOGRAPHY OF THE CANDIDATE . . . . .	136
BRIEF BIOGRAPHY OF THE SUPERVISOR . . . . .	137

## LIST OF TABLES

5.1	Performance of the top five classifiers. . . . .	55
5.2	Number of malware and benign executables for training and testing the classifier. . . . .	58
6.1	Dangerous permissions groups of the Android apps . . . . .	74
6.2	Number of benign and Android malicious apps used for training and testing the classifiers. . . . .	85
6.3	Average accuracy obtained by the classifiers. . . . .	90
6.4	Group-wise maximum accuracy, TP and TN of the classifiers. . . . .	90

## LIST OF FIGURES

1.1	<i>Market share of desktops Operating System, 2017.</i>	4
1.2	<i>Market share of Android based mobile/smart devices, 2017.</i>	5
1.3	<i>Quarter-wise increase in total number of malware from 2015.</i>	6
2.1	<i>First generation malware.</i>	13
2.2	<i>Encrypted malware.</i>	14
2.3	<i>Oligomorphic malware.</i>	15
2.4	<i>Polymorphic malware.</i>	16
2.5	<i>Metamorphic malware.</i>	17
2.6	<i>Traditional detection system.</i>	17
2.7	<i>Malware normalization.</i>	19
4.1	<i>Number of malware with respect to the file size.</i>	34
4.2	<i>Fluctuations in the size of malware generated by NGVCK kit.</i>	35
4.3	<i>Fluctuations in the size of malware generated by G2 kit.</i>	35
4.4	<i>Fluctuations in the size of malware generated by PS-MPC kit.</i>	36

4.5	<i>Normalized opcode occurrence of all the collected malware and benign program keeping the lower threshold 0.02. . . . .</i>	37
4.6	<i>Normalized opcode occurrence of the malware and benign program of size 10-15 KB keeping the lower threshold 0.02. . . . .</i>	38
4.7	<i>Normalized opcode occurrence of the malware and benign program of size 140-145 KB keeping the lower threshold 0.02. . . . .</i>	39
4.8	<i>Normalized opcode occurrence of the malware and benign program of size 240-245 KB keeping the lower threshold 0.02. . . . .</i>	40
4.9	<i>Normalized opcode occurrence of the malware and benign program of size 415-420 KB keeping the lower threshold 0.02. . . . .</i>	41
4.10	<i>Opcodes which found more in malware without grouping the dataset. . . . .</i>	42
4.11	<i>Opcodes which found more in benign programs without grouping the dataset. . . . .</i>	43
4.12	<i>Difference in the occurrence of respective opcodes between benign and malware program of size 0-5 KB and 15-20 KB keeping threshold 0.02. . . . .</i>	44
4.13	<i>Difference in the occurrence of respective opcodes between benign and malware program of size 30-35 KB and 55-60 KB keeping threshold 0.02. . . . .</i>	45
4.14	<i>Flow chart for the detection of unknown malware without partitioning the datasets. . . . .</i>	48
4.15	<i>Flow chart for the detection of unknown malware by group-wise partitioning the executables. . . . .</i>	49
4.16	<i>Detection accuracy obtained by both the methods. . . . .</i>	50
5.1	<i>Flow chart for the detection of unknown malware. . . . .</i>	53
5.2	<i>Accuracy of the thirteen classifiers with N-fold cross validation. . . . .</i>	54
5.3	<i>FP and FN of the top five classifiers. . . . .</i>	56

5.4	<i>TP and TN of the top five classifiers.</i>	56
5.5	<i>Group-wise classification accuracy of the top five classifiers.</i>	57
5.6	<i>Comparison of the accuracy obtained by our approach and others.</i>	57
5.7	<i>Flow chart for the group-wise classification using optimal K-mean clustering algorithm.</i>	59
5.8	<i>Best accuracy of the selected five classifiers.</i>	59
5.9	<i>Detection accuracy obtained by the classifiers from group-1 data.</i>	60
5.10	<i>Detection accuracy obtained by the classifiers from group-2 data.</i>	60
5.11	<i>Detection accuracy obtained by the classifiers from group-3 data.</i>	61
5.12	<i>Detection accuracy obtained by the classifiers from group-4 data.</i>	61
5.13	<i>Detection accuracy obtained by the classifiers from group-5 data.</i>	62
5.14	<i>Detection accuracy obtained by the classifiers from group-6 data.</i>	62
5.15	<i>Detection accuracy obtained by the classifiers from group-7 data.</i>	63
5.16	<i>Detection accuracy obtained by the classifiers from group-8 data.</i>	63
5.17	<i>Detection accuracy obtained by the classifiers from group-9 data.</i>	64
6.1	<i>Top 50 opcodes occurrence without forming the groups.</i>	67
6.2	<i>Flow chart for the detection of Android malicious apps without grouping the data.</i>	69
6.3	<i>Detection accuracy of the selected five classifiers with number of prominent features.</i>	70
6.4	<i>Best accuracy of the selected five classifiers.</i>	70

6.5	<i>True positive rate of the selected five classifiers with number of prominent features.</i>	71
6.6	<i>True negative rate of the selected five classifiers with number of prominent features.</i>	71
6.7	<i>False negative rate of the selected five classifiers with number of prominent features.</i>	72
6.8	<i>False positive rate of the selected five classifiers with number of prominent features.</i>	72
6.9	<i>Top 50 opcodes occurrence difference between benign and malicious apps in the CALENDAR group.</i>	76
6.10	<i>Top 50 opcodes occurrence difference between benign and malicious apps in the CAMERA group.</i>	77
6.11	<i>Top 50 opcodes occurrence difference between benign and malicious apps in the COTACTS group.</i>	78
6.12	<i>Top 50 opcodes occurrence difference between benign and malicious apps in the LOCATION group.</i>	79
6.13	<i>Top 50 opcodes occurrence difference between benign and malicious apps in the MICROPHONE group.</i>	80
6.14	<i>Top 50 opcodes occurrence difference between benign and malicious apps in the OTHER group.</i>	81
6.15	<i>Top 50 opcodes occurrence difference between benign and malicious apps in the PHONE group.</i>	82
6.16	<i>Top 50 opcodes occurrence difference between benign and malicious apps in the SMS group.</i>	83
6.17	<i>Top 50 opcodes occurrence difference between benign and malicious apps in the STORAGE group.</i>	84

6.18	<i>Detection accuracy of the classifiers for the CALENDAR group.</i>	85
6.19	<i>Detection accuracy of the classifiers for the CAMERA group.</i>	86
6.20	<i>Detection accuracy of the classifiers for the CONTACTS group.</i>	86
6.21	<i>Detection accuracy of the classifiers for the LOCATION group.</i>	87
6.22	<i>Detection accuracy of the classifiers for the MICROPHONE group.</i>	87
6.23	<i>Detection accuracy of the classifiers for the OTHERS group.</i>	88
6.24	<i>Detection accuracy of the classifiers for the PHONE group.</i>	88
6.25	<i>Detection accuracy of the classifiers for the SMS group.</i>	89
6.26	<i>Detection accuracy of the classifiers for the STORAGE group.</i>	89
6.27	<i>Group-wise best detection accuracy obtained by the classifiers.</i>	91
6.28	<i>Comparisons of accuracy achieved by us with four other authors.</i>	92
6.29	<i>Group-wise best TP and TN of the classifiers.</i>	92
A.1	Register renaming example	98
A.2	Subroutines permutation	99
A.3	A snippet C code	100
A.4	Variation in the C code after using instruction level permutation.	100
A.5	Insertion of jump instructions to create new variants.	100
A.6	Inlining and outlining of subroutines.	101
A.7	Snippet of Evol virus using dead code insertion.	102



# CHAPTER 1

## INTRODUCTION

### 1.1 Background

The development of computer security has a military origin, and since 1950 it is a major concern. Initially, because of national defense and intelligence, US government was “a major force behind security research and technology” [60]. Till 70’s computers users were small, hence protection of data was easier. But, in 80’s Personal Computers (PC) came into being which was small enough to fit on a desk, and in 90’s the Internet has made a revolutionary impact on the PC user, basically due to its near-instant communication. In 1992, IBM came up with a prototype mobile computing device (phone) that incorporated Personal Digital Assistant (PDA) features (demonstrated it in the Computers Dealer’s Exhibition). Later on, Simon Personal Communicator made the first device that could be really referred as smart device/-phone which receives calls, sends faxes, emails and more. The smart devices/phone technology continued to advance throughout early 2000, and in 2007 Android based smart device was unveiled by Google. Since then the popularity/demand of Android based devices is continuously growing. Today in the smart devices, Android is the most dominant OS. An estimate shows that more than 15 billion smart devices are connected globe-wise and are expected to be reaching 200 billion by end of year 2020 [29].

The ubiquity of the Internet has engendered the prevalence of information sharing among networked users and organizations, and in today’s information era, most

of the computing devices are connected to the Internet, which has rendered possible countless invasions of privacy/security worldwide from the *malware* (the term *malware* is derived from the word **malicious** and **software**, and are often used interchangeably with the virus, even though the two are not the same. Actually, malware is a condensed, conjoined term used to refer to a program (viruses, worms, trojan horses, spyware, adware, rootkits, botnets, etc.) that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system or otherwise annoying or disrupting the victims [92]). An important feature of these malware is its ability to self-replicate. It is not known who created the first self-replicating program in the world, but it is clear that the first malware/virus (Creeper) was created by the Bible Broadcasting Network engineer Robert (Bob) H. Thomas, probably around 1970 and the first smartphones malware was created in the year 2004, which was a worm known as Cabir to infect Symbian-based devices [59].

From the last four decades, malware is continuously evolving with high complexity to evade the available detection techniques and new variants of malware are getting evolve every day, as a consequence, malware defense is becoming a difficult task to protect the computational devices from them. The threat/attack from the advanced/ second generation malware (Chapter 2) are not only limited to individual level, but there are state-sponsored highly skilled hackers developing customized malware to disrupt industries and for military espionage [105]. Attack by such malware can alter the operation of industrial systems, disrupt power plants, e.g. the StuxNet and Duqu malware [18], and many countries continue to incur the costly data breaches, e.g. the two countries who had the highest cost of data breaches in the 2013 was U.S. (\$ 5.4 million) and Germany (\$ 4.8 million) [94]. In this, intrusion into Google systems demonstrates how well-organized attacks are designed to maintain long-term access to an organizations networks [32].

In August 2014, eleven zero-day vulnerabilities were reported, in which six were from the industrial control systems [95]. According to McAfee technical report, in 2014, there were more than 200 million known malware samples [27]. The Symantec 2014 Internet Security Threat report states that 2013 was the mega breach year (62%

more breaches than 2012) [94]. The F-secure document reported an increase in malware attacks against mobile devices based on Android and Apple iOS [35]. This increase in the threat from malware is due to the wide-spread use of World Wide Web. An estimate shows that the web-based attacks were increased by 36% with over 4,500 new attacks each day, annoying/disrupting the victim in terms of confidentiality, integrity, availability of the victim's data [105]. In the first quarter of 2017, the highest number (295 million) of malware samples were detected on the systems of Quick Heal users. However, compared with the first quarter of 2016, there was a drop of 13.61% in the detection count [13]. In this year Quick Heal detected a targeted attack on an Indian government organization. The name of the group behind this attack was "Quarian" and has been active since 2011. This group targeted an Indian government organization using spear phishing e-mails loaded with malicious Microsoft Office documents designed to exploit an old "CVE-2010-3333" vulnerability [12]. On reading the mail, it drops an executable file on the system, collects system information, and provides remote shell access to the attacker.

In the last couple of years, cyber-criminals had focused mainly on the bank customers to steal millions of dollars in a single attack. In 2016, Symantec uncovered the most effective bank robbers (Banswift group) who stolen the US \$81 millions of dollars from Bangladesh's central bank by exploiting weaknesses in the bank's security [99]. Later on, some more cases were reported in South Asia banks, which were attacked with same malware code that was used in Bangladesh. In this, Ransomware continues to plague businesses and consumers, with indiscriminate campaigns pushing out mass. The average ransom demand in 2016 rises to \$1,077, a 366% increased from the previous year. A growing reliance on cloud services also creates vulnerabilities for organisations which cause the ability to hackers to hijack tens of thousands of MongoDB databases and asked for ransom [98]. The number of new ransomware families uncovered during 2016 was 101 i.e. more than three times compared to the year 2015, and Symantec logged a 36 percent increase in ransomware infections [98], while Quick Heal Labs detected ten new ransomware families in same quarter [13]. The uptrend shows that attackers are jumping on the ransomware bandwagon and creating new ransomware families or modifying existing ones.

Since the first virus created in 1970 [106], there is a strong contest between the malware and anti-malware developer. This rate-race led to the improvement in malware and the detection techniques. To defend the malware attacks, anti-malware groups are developing new techniques, on the other hand, malware developers are adopting new tactics/methods to avoid the sight of scanners. Initially, the tools and techniques of malware analysis were in the domain of anti-malware vendors. But the use of malware for espionage, sophisticated cyber attacks, and other crimes motivated the academicians and digital investigators to develop an advanced method to combat the threats/attacks from it. However, due to continuously increase of huge data, anti-malware industries are facing major challenges to check for the potential malicious content. The reason behind these high volumes of different files is that the malware creators uses obfuscation techniques to generate an entirely new variant of the malware after each infection [92].

It's an indisputable fact that the prolong traditional signature based approach for combating the threats/attack with a technology-centric are ineffective to detect the second generation customized malware, and the Internet connected devices are becoming an attractive target for the online criminals. The attackers are investing more and more for the sophisticated attacks viz. ransomware or to steal the valuable personal data from the user device. Recent attacks shows that the security features in computing devices are not as par to completely stop the adversary [99]. Therefore, if in time adequate measures not taken, then the consequence of the scale ( $\sim 317$  million new malware are reported in the year 2014 [95]) at which malware are created will be

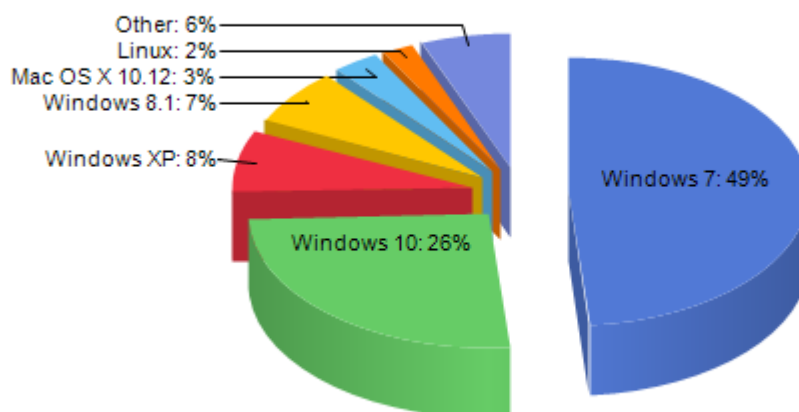


Figure 1.1: *Market share of desktops Operating System, 2017.*

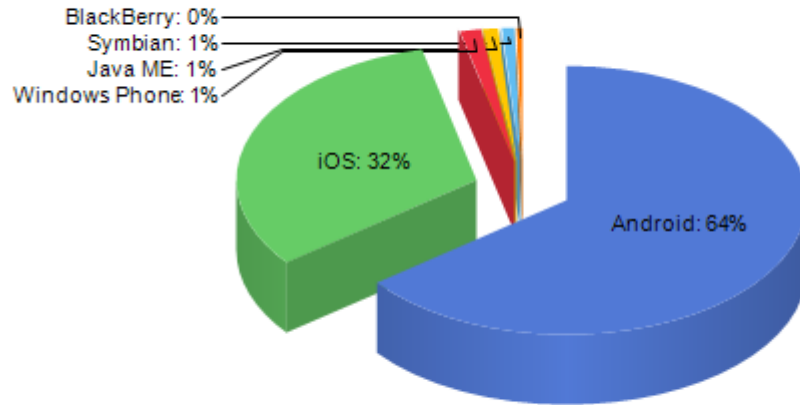


Figure 1.2: Market share of Android based mobile/smart devices, 2017.

devastating. Hence, in this thesis, we study and demonstrate how to effectively classify the unknown advanced/second generation malware of Windows Desktops and Android based devices, which have  $\sim 90\%$  (Figure 1.1<sup>1</sup>) and  $\sim 64\%$  (Figure 1.2<sup>1</sup>) market share respectively.

## 1.2 Motivation

Despite the advancement in the development of anti-malware the number of malware (Figure 1.3) and cyber-attacks are on an uptrend. An estimate by Symantec shows that the rate of creation of new instances of malware was 41%, with a total of over 400 million existing new malware instances [93]. According to FireEye survey [1], 47% of the organization experienced malware security incident/network breaches. Internet security threat report of 2017 says that the web-based attack is increased 36% with over 4,500 new attacks each day, annoying/disrupting the victim in terms of confidentiality, integrity, and availability of the victim's data [93]. As per McAfee technical report of 2014, there was more than 200 million known malware samples [27] and the year 2013 was the mega breach year [94]. In the last quarter of 2015 a 26% increase in new ransomware samples has been reported [8] and in the first quarter of 2016 the Quick Heal Threat Research Lab received more than 340 million malware samples running into hundreds of thousands of devices [4]. Symantec reported 54 zero-day vulnerabilities, and it is doubling each year [96]. McAfee reported a new malware which is capable to

<sup>1</sup>Generated online from <https://www.netmarketshare.com/operating-system-market-share.aspx>

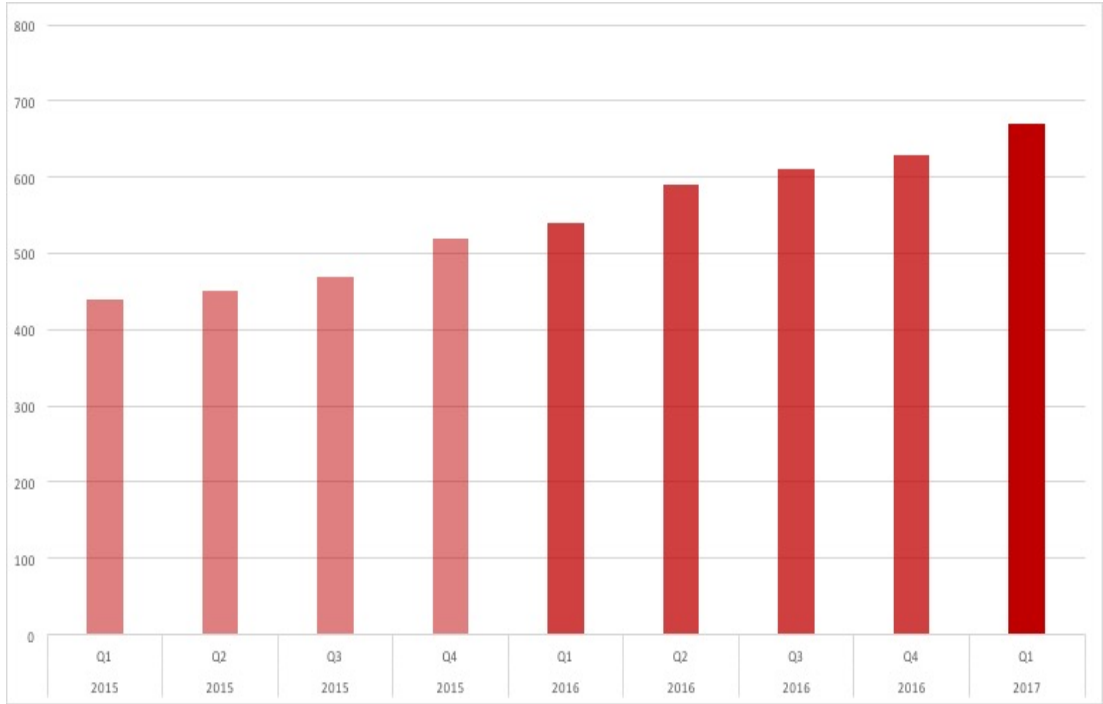


Figure 1.3: *Quarter-wise increase in total number of malware from 2015.*

infect the hard drives and solid state storage device firmware and the infection cannot be removed even by formatting the devices or reinstalling the operating systems [28].

In the smart devices, Android is the most popular operating systems [40], and are connected through the Internet accessing billions of online websites (an estimate shows that 5 out of 6 mobile phones are working on Android OS [97]). The popularity of Android OS is basically due to its open source, exponential increase in the Android supported apps, third-party distribution, free rich SDK and the very much suited Java language. In this growing Android apps market, it is very hard to know which apps are malicious. As per Statista [104]  $\sim 2 \times 10^6$  Android apps are available at Google play store and many third-party Android apps are also available for the users. Because these devices are very much convenient to use for the day to day activities, it holds sensitive information such as personal data, browsing history, financial details etc. In this, the third-party Android apps which are available for the users may be malicious [8]. Hence potential of the malicious apps or malware entering these systems is now at never seen before levels. Thus, attacks on the Android devices is increasing at an unprecedented rate, mainly due to the ease of generating different malware variants [113]. In 2013, there was 200% increase in malicious apps, and 3.7 million of variants added in McAfee's

database [28]. In 2015, Kaspersky reported that the growth rate of new malware variant is 300% with 0.88 million new variants [5]. The number of malicious installations found in 2015 was around three million and around seven thousand mobile banking trojans were also found in the same year [2]. In the 3rd quarter of 2015 Quick Heal Threat Research reported that they had received samples of files at the rate of  $\sim 4.2 \times 10^5$  samples per day for the Android and Windows platforms [6]. Trend Micro estimated that the number of malicious mobile apps will reach 20 million by the end of 2017 [3].

To counter/defend the malware, there are many anti-malware defense systems based on signature matching, code emulation, heuristic code analysis and machine learning (Chapter 3). Some malware are easy to detect and can be removed from the system by commonly used signature based anti-malware software. But the signature based technique can't detect advanced/second generation or unknown variant of malware as it uses advanced obfuscation techniques, which exploit the limitations of state-of-the-art anti-malware products to bypass security protections and eventually evade detection. However time to time, a number of static and dynamic methods has been proposed (Chapter 3). But, it appears that so far proposed approaches are not sufficient to detect the advanced malware to limit/prevent the damages [92] in the fast-growing Internet and computational devices usage into our daily life. Hence, computational device security is viewed as one of the most important areas to be addressed. Therefore understanding the market share of Windows (Figure 1.1) and Android based smart devices (Figure 1.2), which are an attractive target for computer hackers and criminals who develop malware, this thesis proposes methodologies for the effective classification to detect new or previously unknown advanced malware for Windows Desktops and Android based devices.

### 1.3 Research Gap

In classification, feature selection plays a vital role, not only to represent a target concept but also to speed up the learning process. In this, often datasets are represented by many features, however few of them may be sufficient for both to speed-up the learning process, and to improve the concept quality. Many feature

selection techniques viz. information gain, correlation-based feature selection, relief, and hybrid methods [51] has been studied to reduce the number of features in the datasets. However, it appears that studied feature selection techniques are not sufficient to classify the malware with high accuracy. Hence there is a need to develop a feature selection technique to improve the classification accuracy of the malware with low false alarm.

Due to the availability of intelligent software to create variants of malware [92], it appears that malware creators are ahead of the anti-malware developer. The major challenge to check the potential malicious content is the continuously increasing huge dataset. Microsoft, nearly real-time anti-malware are present on over 160 Million computing devices throughout the globe, which analyzes tens of millions of data files daily [94]. According to a recent Computer Security Institute survey, the average loss from security attacks was about \$ 345,000 per incident [78]. As new variants of malware getting evolve every day, malware defense becoming difficult task to protect the computational devices from them [28]. In August 2014 eleven zero-day vulnerabilities were reported in which six were from the industrial control systems [95].

The advanced malware, in particular, metamorphic malware are getting more complex which pose a big threat and new challenges to the endpoint security. Advanced obfuscation techniques helps a malware to evade detection by the traditional signature-based anti-malware software, and also the number of new malware are increasing significantly. Hence, there is a need to automate the process of malware analysis and detection. To address effectively the challenges posed by advanced/metamorphic malware, there is a need to develop new methods and techniques to analyze its behaviour to make a better detection decision with few false positives. Current techniques for detecting malware are computationally intensive, have poor detection rates, and are not sufficient to detect the advanced malware to limit/prevent the damages [92]. Also, there is no method available to detect zero day attack with 100% accuracy [9]. It has been reported that there exists malware which cannot be detected by any anti-malware [41]. Moreover, it is impossible to develop a generic algorithm to detect all possible malware [110]. Hence regular study is required to put more sophisticated defense to combat the threat/attacks from the advanced unknown malware. TRsym2014



In smart devices, in particular, the most popular Android based devices, malware are increasing at an unprecedented rate, and it is mainly due to the ease of generating malware variants [26]. The recent attacks on smart devices show that there is an urgent need to develop robust anti-malware, in particular for the defense against zero-day attack [92]. The traditional signature matching approaches are efficient from a time perspective but not relevant for the variants nor capable to detect continuously growing zero-day malware attack. Also, to evade the signature-based techniques, malware developer uses several obfuscation techniques (Appendix refappendix:obfuscation) [94]. However, to detect the Android malicious apps, time to time, a number of static and dynamic methods has been proposed viz. *Droid*, *Andro-Dumpsys*, *MADAM*, *Droidkin*, etc. (Chapter 3). But, it appears that so far proposed approaches are not sufficient to detect the advanced Android malware to limit/prevent the damages [92] in the fast-growing Internet and Android based devices usage into our daily life. Hence the smart device security is viewed as one of the most important areas to be addressed. Also, it is an open question, how to detect new malware variants in smart devices which are always hidden in the many different third-party markets [8], and to find out how one can identify repackaged applications from the vast ocean of apps.

## 1.4 Objectives and Organisation of the Thesis

The thesis primarily focuses to improve the detection accuracy of new or previously unknown advanced malware of Windows Desktops and Android based computational devices by group-wise classifying the data using machine learning techniques. Accordingly, Chapter 2, discusses the types of malware and its detection techniques and Chapter 3 discusses the prior works/efforts done by the researchers to detect Windows and Android based devices malware.

In Chapter 4, we investigate the variation in the size of malware generated by metamorphic malware generator kits viz. G2, PS-MPC and NGVCK and accordingly we group-wise analyzes the data set to detect the unknown malware by Naive Bayes classifier, and the obtained results are compared with the regular method (without grouping). As feature selection plays a vital role to represent the target concept,

therefore in this Chapter we also proposed an approach to find the prominent features for the classification of malware.

To find the best classifier for the detection of unknown advanced malware, in Chapter 5, we studied the performance of the popular thirteen classifiers using N-fold cross-validation with Malicia data set and then selected the top five among them for the detection of malware. As discussed in Chapter 4 that group-wise classification improves the detection accuracy, therefore in Chapter 5, we grouped the executables on the basis of malware sizes by using optimal k-Means clustering algorithm and classified the data by the top five identified classifier.

Chapter 6 investigates the performance of the top five classifiers for the effective detection of Android malicious apps, first by classifying the data without grouping data and then by grouping the data based on permission to improve the overall detection accuracy. Finally, Chapter 7 contains conclusions of the thesis and future directions.

## 1.5 Contribution

This thesis brings contributions in the detection of unknown advanced malware to protect the two most popular computational devices, Windows Desktops and Android based devices from the uptrend cyber threat/attacks. In this, we empirically analyzed and show that the detection accuracy of the unknown malware can be improved by Group-wise approach. Following are the published/communicated works that contribute the material in this thesis:

- Feature selection plays a vital role, not only to represent the target concept but also to speed-up the learning and testing process. Hence to find the prominent features for the detection of malware, a simple algorithm has been developed [91].
- Investigated the popular metamorphic malware generator kits viz. G2, PS-MPC and NGVCK and found that the variation in the size of the malware generated by this generator is not more than 5 Kilobyte and also shown that the detection accuracy can be improved by group-wise classification of the data [91], [89].
- In continuation to improve the detection accuracy, we studied the performance of the popular thirteen classifiers using N-fold cross-validation with Malicia data set and then selected the top five among them for the detection of malware by grouping the dataset using Optimal k-Means clustering algorithm [89], [79].
- Understanding the popularity of Android based devices and the cyber threat/attacks on this devices, we investigated for the effective detection of Android malicious apps, first by classifying the data all together i.e. without grouping, and then shown that how the overall average detection accuracy can be improved to detect the Android malicious apps by group-wise classifying the apps based on permissions [88], [90].

# CHAPTER 2

## TYPES OF MALWARE AND DETECTION TECHNIQUES

### 2.1 Introduction

Frederick B. Cohen, the inventor of computer virus defense techniques, has first defined the term “*computer virus* as a program that can infect other programs by modifying them to include a possibly evolved copy of itself” [31]. Since then malware has been continuously evolving with high complexity to evade the detection techniques, and are classified as first and second generation malware. This Chapter provides an overview of different types of malware and its detection techniques.

### 2.2 Types of Malware

The first generation malware are broadly classified on the basis of their infection strategy as Viruses (*attaches itself to a program and propagates copies of itself to other programs*), Worms (*a program that propagates copies of itself to other computers*) and Trojans (*a program that contains unexpected additional functionality*) [103]. Few other notable first-generation malware are rootkits, spyware, crimeware, adware etc. (Figure 2.1) [106]. They all exhibit different sort of malicious behaviour on the target systems, but their structure does not change. But in the second generation,

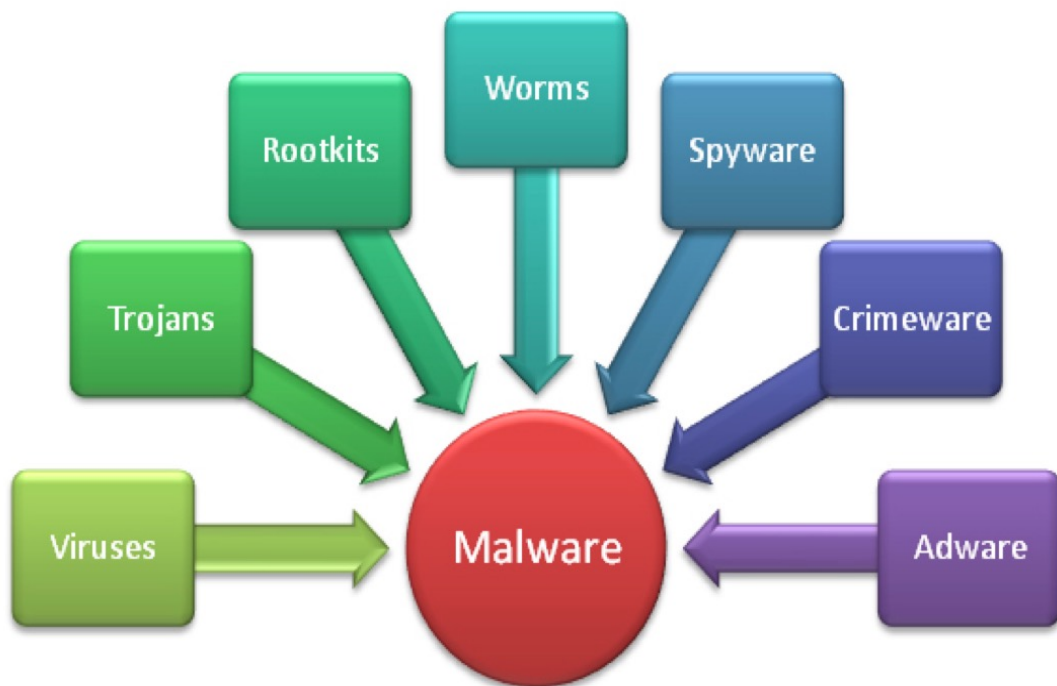


Figure 2.1: *First generation malware.*

after each infection, the structure of malware changes to create a new variant keeping the action same [92]. On the basis of the mechanism by which either the code or the structure varies to conceal the signature of the malware, the second generation malware are further classified as Encrypted, Oligomorphic, Polymorphic and Metamorphic malware.

### **Encrypted Malware**

Encryption was the first concealment techniques used for creating the second generation malware [122]. As shown in Figure 2.2, it consists of two parts, the encrypted body and a decryption code [75]. In this type of malware, first the decryption part is executed to decrypt the body (usually the body is XORed with a key to make it difficult to detect) of the malware and then the code is executed for the action, and after each infection, encrypted malware makes the body unique by using different key to hide its signature. However, the decryption routine remains same, hence it can be detected by analyzing the decryptor. The first encrypted malware was CASCADE [17]. Later on using the CASCADE technique Win95/Mad and Win95/Zombie were created. The

main motivation to use the encryption malware is to evade the signature matching technique and static code analysis [74].

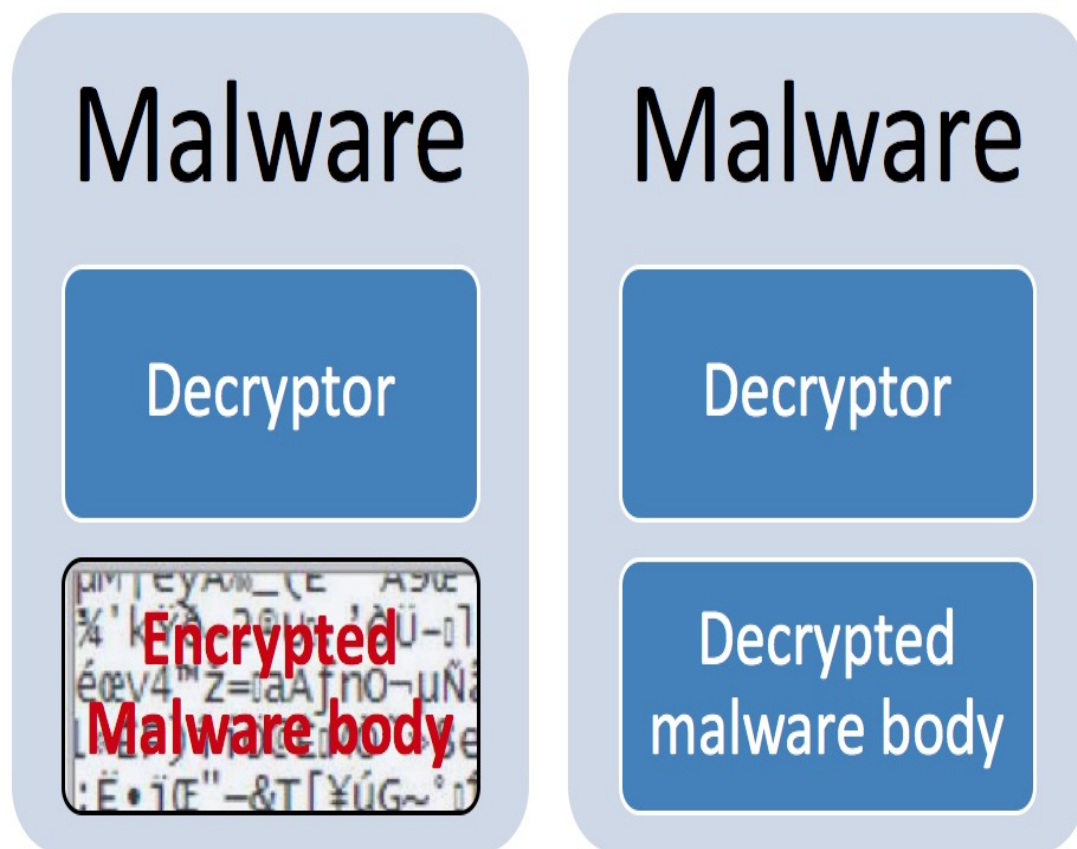


Figure 2.2: *Encrypted malware.*

### Oligomorphic Malware

The shortcomings of the encrypted malware led to the development of Oligomorphic malware and different concealment techniques. In Oligomorphic malware (Figure 2.3) decryptors are mutated from one variant to other. The simple method to create Oligomorphic malware is to provide a set of different decryptors. Initially, this type of malware was capable of changing the decryptor slightly [122], and at the most, this malware can generate few hundred of different decryptors, e.g. Win95/Memorial had the ability to build 96 different decryptor patterns [87]. For the detection of Oligomorphic malware, signature based techniques can be applied by making the signature of all the decryptors. However, in general, signature based techniques are not considered

as a good approach to detect the Oligomorphic malware [75].

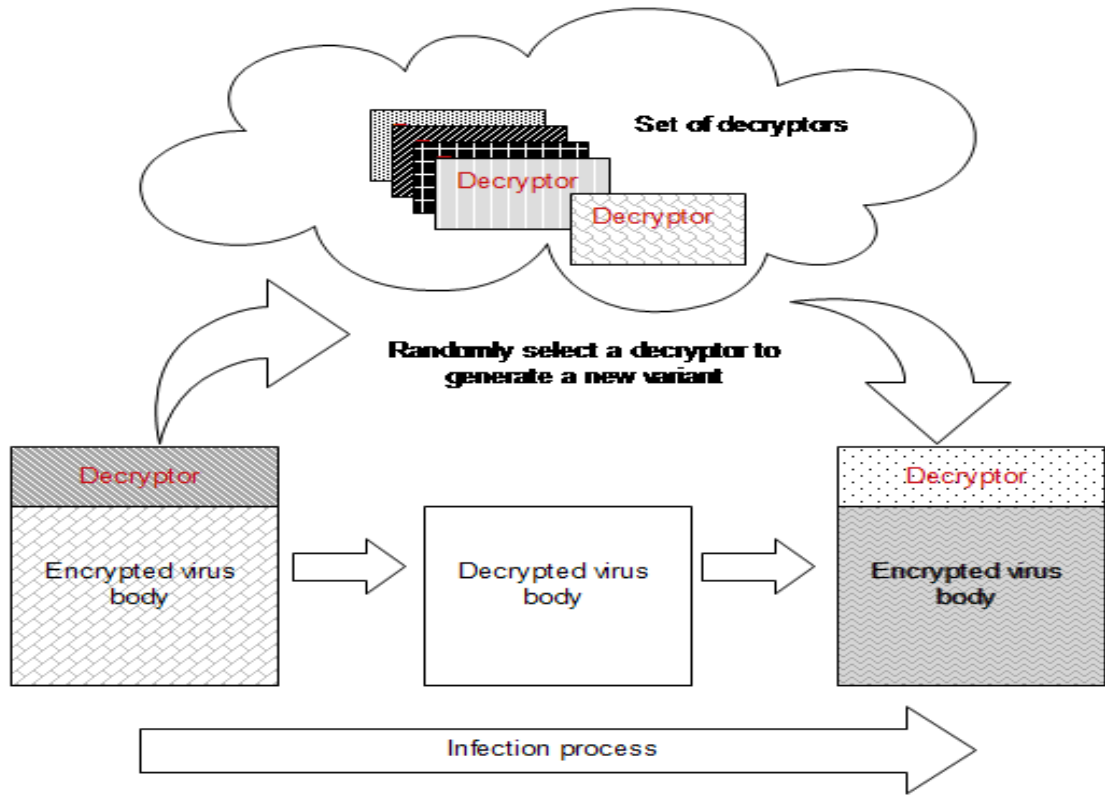


Figure 2.3: *Oligomorphic malware*.

## Polymorphic Malware

To evade the signature based detection technique, polymorphic malware are able to generate millions of decryptors by changing the instructions in next variant of the malware [74]. As shown in Figure 2.4, it also consists of two parts, the first part is the code decryptor to decrypt the second part (body). During the execution of malware, mutation engine creates a new decryptor which is joined with the encrypted malware body to construct a new variant of malware [92]. These malware are created by using different obfuscation techniques viz. Register renaming, Subroutine permutation, Transposition, Changing the control flow, Subroutine inlining and outlining, Equivalent code substitution, Dead code insertion, etc. (Appendix A) [122]. The first known polymorphic malware was 1260, written by Mark Washburn in 1990 [75]. Although, a large number of variants of decryptors can be created, still signature scanning technique can be used to detect the malware by identifying the original program using

the emulation techniques [122].

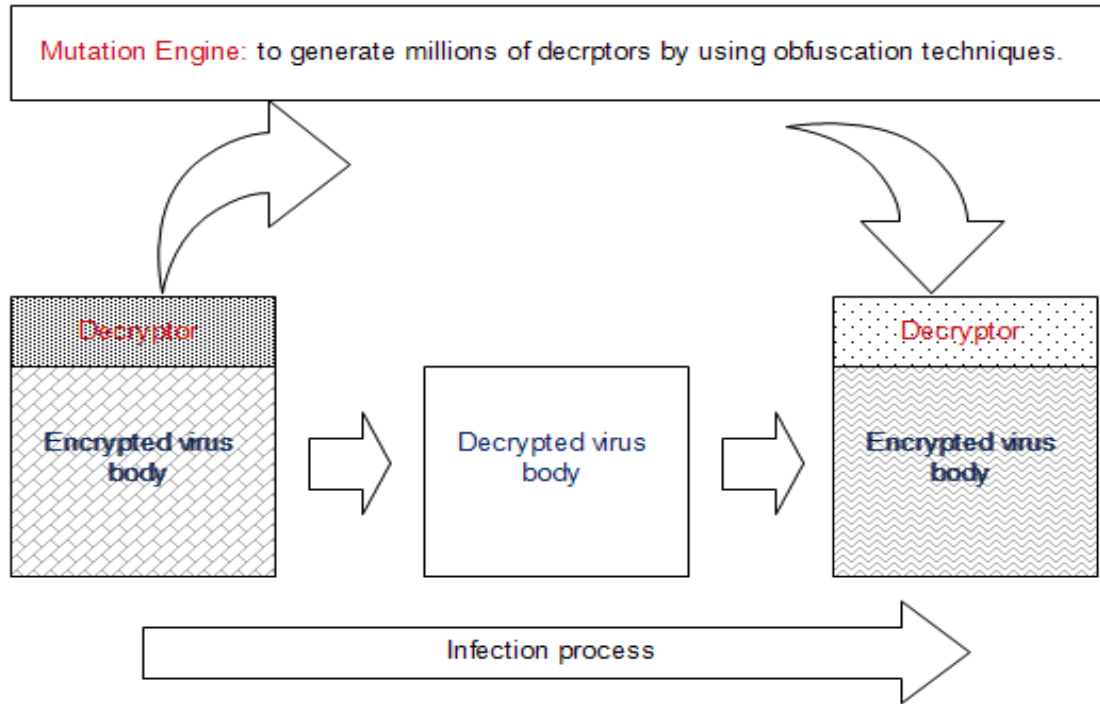


Figure 2.4: *Polymorphic malware.*

## Metamorphic Malware

Metamorphic malware are body-polymorphic (Figure 2.5), i.e. instead of generating new decryptor, a new instance is created without changing its actions [75]. Similar to polymorphic malware, different obfuscation techniques are used to create new instances. It is believed that in future it will harm both computers and smart devices in large scale as it is almost impossible to detect it by the traditional signature based techniques. However, creating a true metamorphic malware without arbitrarily increasing the size is a challenging task. It has been shown that there are few malware which exhibits true metamorphic behaviour [16], e.g. Phalcon/Skism Mass-Produced Code Generator (PS-MPC) [17], Second Generation virus generator (G2) [16], and Virus Creation Lab for Win32 [116]. The first metamorphic virus was created in 1998 called as Win95/Regswap [107]. Later in 2000, Win32/Ghost virus was created with 3628800 different variants [122], and one of the strongest metamorphic malware W32/NGVCK was created in 2001 with the help of Next Generation Virus Creation Kit (NGVCK) [112].



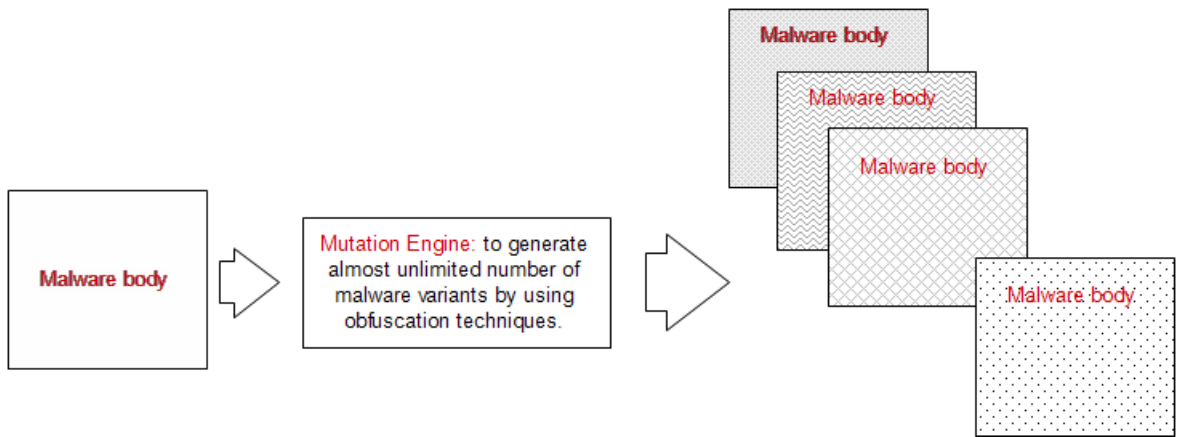


Figure 2.5: *Metamorphic malware.*

## 2.3 Detection Techniques

To combat the threat/attacks from the malware, softwares/anti-malware are developed, which are mostly based on the assumption that the malware structure does not change appreciably (a schematic of a traditional detection system is shown in Figure 2.6). But the variant of second generation malware are very much different to each other. Hence threat/attacks from such malware to the computational devices are increasing day by day. Therefore, there is a need that both academia and anti-malware developers should continually work to prevent damages from the evolution of advanced complex malware. Thus, this section discusses the various techniques used for the detection of malware.

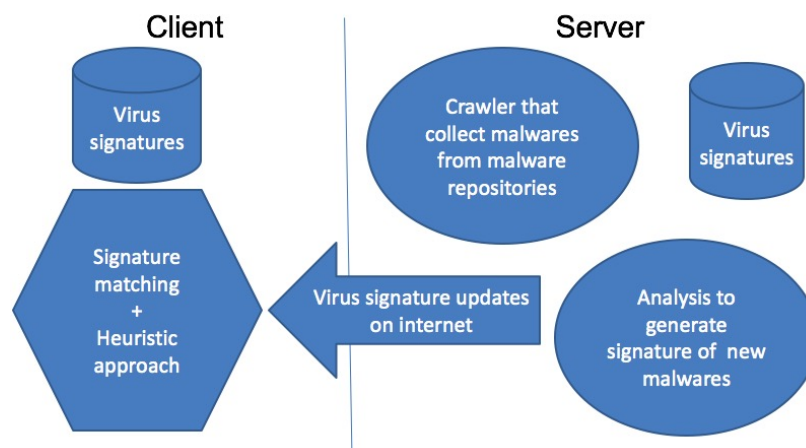


Figure 2.6: *Traditional detection system.*

### 2.3.1 Signature Based Detection

The signature based detection technique is the simplest and an effective way of detecting the known malware [42]. Once the malware is identified, a unique sequence of bytes are extracted from it, which represents the signature of the malware, e.g. Aho-Corasick algorithm scan for the exact matching [111]. These signatures are selected long enough to characterize a specific malware with respect to any other benign program, e.g. *Worm/klez.E* and *Worm/MyP-arty.A* signatures are 33be732d4000bd08104000e89eeafff80bd08 104000be7d2d4000e849eafff6a00e835000000 64756d6d792e65786500653a5c77696e646f77 735c53795374656d33325c644c6c636163686 55c6464642e65786500ff254c404000ff25544040a and aa328cf24554d90b307c407eca9a4cf02 a4d5a90000332c8b26904ffffb840f97f370080040e 1fba0e00b409cd21b8014c001f027c54686 973c363616e042568d54562e2c876b0ffbf0420444f53 respectively [34]. These techniques scans the file to find the defined malware signature, if found then an alert of the presence of malware is sent, but a slight mismatch will escape the detection. Therefore, Veldamna and Wu-Manber proposed the use of the wildcard for detecting slight variance in the malware [74]. By this technique some metamorphic malware could be detected, e.g. W32/Regswap [74]. This method is easy to use, however, the requirement of scanning becomes costly as the database of malware signature is increasing very fast [45]. Also, it is a completely reactive technique, therefore unable to combat threats/attack from the new or previously unseen malware until it causes the damages. Gartner [43] believes that eventually signature-based techniques will be replaced with more robust approaches, because today's signature-based anti-malware have marginal value, as second generation malware easily evades the signature based detection techniques.

### 2.3.2 Heuristics Based Detection

The heuristic based detection technique is one of the promising technique to detect the unknown malware [45]. In this method, there are two approaches for the detection of malware. First, in the static approach suspicious program are disassembled to find a matching of the known malware pattern, if any, and if the analysis result crosses the preset threshold then the program is marked as infected [62]. Secondly, in

dynamic approach, code emulation techniques are used by simulating the processor and operating system to detect suspicious operations (an attempt to open other executable files with the intention of modifying its content, changing the Master Boot Record, concealing themselves from the operating system, etc.) on a virtual machine. However, this detection technique is prone to false alarm [116], which may make the system more vulnerable by taking the real malware as another false alarm. Therefore to reduce the false alarm, researchers augment the results of this detection technique with other method [41].

### 2.3.3 Malware Normalization

The malware generated from advanced toolkits such as *Ultimate Packer for Executables* (UPX) and *Mitsfall* are difficult to detect [120]. For the detection of such malware, normalization techniques can be used to improve the detection rate of an existing anti-malware. In this technique, normalizer accepts the obfuscated version of malware and removes the obfuscation in the program to obtain a normalized executable. After normalization, the signature of malware is extracted and compared with the signature of canonical form (Figure 2.7) [118]. Christodorescu et. al. designed a malware

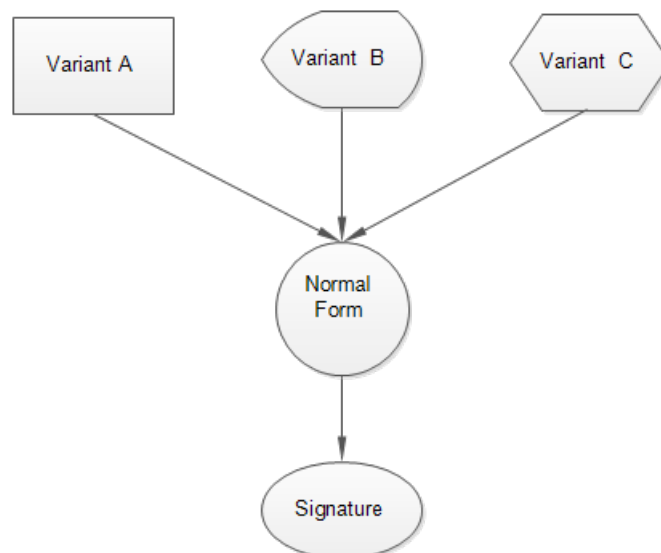


Figure 2.7: *Malware normalization.*

normalizer that handles three common obfuscations viz. code reordering, packing,

and junk insertion [30]. Later on Armor et. al., [14] proposed a generalized malware normalizer which can store obfuscation methods in the form of automata structures and use them for normalizing the metamorphic malware. In 2014, Armoun et. al. proposed a general malware normalizer which can detect the metamorphic variant up to 81% accurately by storing lots of obfuscation methods in the form of automata structures to normalize the metamorphic malware [14].

### **2.3.4 Machine Learning Techniques**

Recently, malware detection with machine learning techniques is gaining popularity. Tom Mitchell defines machine learning as the study of computer algorithms that improves through experiments [65]. The advantage of machine learning techniques is that it will not only detect known malware but also act as knowledge for the detection of new or previously unseen malware. The popular machine learning techniques among the researchers for the detection of second generation malware are Naive Bayes [9], Decision Tree [66], Data Mining [101] , Neural Networks [66], Hidden Markov Models [48], etc. Generally, machine learning techniques are more computationally demanding than the standard anti-malware, hence it may not be suitable for the end user. However, it can be implemented at enterprises gateway level to act as a central anti-malware engine to supplement anti-malware. Although, infrastructure requirement is costly, but it can help in protecting valuable enterprises data from the security threat and can prevent immense financial damages.

## **2.4 Summary**

This Chapter presents an overview of the classification/evolution of malware and it's detection techniques. It discusses in details the type of second generation malware and the popular detection techniques used to detect it viz. Signature matching, Heuristic methods, Normalization, and Machine Learning techniques. As the thesis is on the detection of advanced malware, therefore this Chapter elaborated the second generation malware in more details.

## CHAPTER 3

### LITERATURE SURVEY

#### 3.1 Introduction

The first virus was created in 1970 [106] and since then there is a strong contest between the attackers and defenders. This rat-race led to the development of complex malware and its detection techniques. To defend the malware attacks, anti-malware groups are developing new techniques. On the other hand, malware developers are adopting new tactics/methods to evade the anti-malware. The complexity of the malware is continuously growing for the military espionage, sophisticated cyber attacks and other crimes, which motivated the academicians and digital investigators to develop the advanced method to combat the threats/attacks from it. To combat the threats/attacks from the second generation malware, time to time, a number of static and dynamic methods has been proposed by the researchers [56] [46] [61] [80] [77]. In order to understand various techniques proposed/used for the detection of Windows Desktops and Android based device malware, a survey has been conducted and are discussed in this Chapter.

#### 3.2 Survey on the Detection of Windows Desktops Malware

Static and dynamic analysis are the two main approaches applied for the detection of malware [92]. In static analysis, without executing the malware, the codes are analysed to find a malicious pattern by extracting the features/signature such as N-

grams, Application Program Interface (API) sequences, opcodes, data flow, resources, Dynamic-link Library (DLL) usage, etc. Whereas, in the dynamic analysis the programs are examined in a runtime environment by monitoring the dynamic behaviours, such as network connections, system calls, resources usage, etc. of the programs. However, in both the approaches selected classifiers are trained with a known dataset to differentiate the benign and malware programs.

In 2001, Schultz et al. [85] using the data mining methods claimed that their framework can detect the new unknown executables twice than the detection rate of traditional signature based techniques. For the classification, they used three different static features viz. Portable Executable (PE), strings and byte sequences. From the data set of 3265 malicious and 1001 benign programs, they extracted the features from DLL residing in the PE files, strings are extracted from the executables that are encoded in program files and the bytes sequences are extracted from the executable file. They applied repeated Incremental Pruning to Reduce Error Reduction (RIPPER) [38] to find the patterns in the DLL data, and Naive Bayes was used to find patterns in the string data. Finally, n-grams byte sequences were used as input data for the Multinomial Naive Bayes algorithm and obtained the detection rate of 97.76%

Kolter and Maloof in 2004 [56] uses the techniques of Information Retrieval (text classification) for the detection of unknown malicious executables. They experimented with 1971 benign and 1651 malware programs, selected 255 millions distinct n-grams with variety of inductive methods, including Naive Bayes (NB), Decision Trees (DT) and Support Vector Machines (SVM). Their results from the boosted decision tree outperformed other methods with an area under the Receiver Operating Characteristic (ROC) curve of 0.996. In 2005, Karim et al. constructed a malware phylogeny model using n-perms as a feature to match the possible permuted code [54]. They performed an experiment to compare the relative effectiveness of the vector similarity measures using n-perms and n-grams to compare the permuted variants programs. They found that n-perms provides higher similarity scores for permuted programs and produce comparable phylogeny models. Also, n-perms appear to do a better job in differentiating related and unrelated similarities in sample sets of permuted variants. Hence, a better choice for constructing phylogeny models for the presence of malware

that has evolved through permutations. Their analysis suggests that the phylogeny models generated by this technique may be able to reconcile the name inconsistencies and assist in the investigation of new malicious programs.

In 2006, O. Henchiri et al. [46] proposed a search method to select generic features for the detection of the computer virus of different families. They used 1512 labeled viruses, 1488 benign program and Iterative Dichotomiser-3 (ID3), j48, Naive Bayes, Sequential minimal optimization (SMO) classifier for the evaluation. Their result outperforms the traditional search methods, and the best detection accuracy obtained was 92.56% by the J48 classifier. They claimed that their approach, which uses a family of non-specific features, performs very well, while existing techniques for detecting the previously unseen viruses perform significantly poor. In 2007, Blair [20] discusses the opcodes as a predictor of malware, he disassembled 67 malware and 20 benign executables to study the distributions of malware opcode occurrence. He found that the malware opcode distributions differ statistically significantly from non-malicious programs and the rare opcodes seem to be a stronger predictor. In 2008, Robert Moskovitch et al. [66] based on text categorization concepts proposed a methodology for the detection of unknown malicious code. They performed an extensive evaluation with  $\sim 30,000$  malicious and benign files to investigate the imbalance problem. Their results indicate that around 95% accuracy can be achieved through the use of a training set that contains below 20% malicious file content. Later on, his group [67] proposed a method to detect the unknown malicious program by using n-grams (3, 4, 5 or 6) of the opcodes as features, generated by disassembling the executables. To reduce the number of n-gram features, which ranges from thousands to millions, they first used the document frequency (DF) measure to select the top 1000 features. Then they compared the various classifiers by byte-sequence n-grams (3, 4, 5 or 6) and found that Boosted Decision Tree provides the best accuracy (94.43%). Their analysis shows that the Fisher Score and DF feature selection were better than the Gain Ratio.

To detect new or previously unseen polymorphic/metamorphic malware, Yanfang Ye et al. [121] in 2008 analysed the Windows API called by Program executables (PE) files to develop a Intelligent Malware Detection System (IMDS) using Objective-Oriented Association (OOA) mining based classification, which is an inte-

grated system consisting of three major modules: PE parser, OOA rule generator, and rule-based classifier. They conducted a comprehensive experiment on a large collection of PE files (636 malicious and 1207 benign executables) obtained from the anti-virus laboratory of KingSoft Corporation to compare the various malware detection approaches. Their experimental results demonstrate that the accuracy and efficiency of the IMDS system outperform the popular anti-virus software such as Norton AntiVirus, McAfee VirusScan, and previous data mining based detection systems which employed Naive Bayes, Support Vector Machine and Decision Tree techniques. In 2008, Tian et al. [109] discusses *Function Length as a Tool for Malware Classification* and claimed that classifying the Trojans on the lengths of their functions will be fast, simple and scalable. Their result indicates that the function length may play a significant role in classifying the malware, and if combined with other features, may result in a fast, inexpensive and scalable method for the classification. But they also showed that it will be unrealistic to expect function length information on its own to produce perfect accuracy to distinguish the families.

Siddiqui et al. [100] applied data mining techniques for the extraction of variable length instruction sequence to identify the worms from the benign programs. Their analysis was facilitated by the program control flow information contained in the instruction sequences. From these instruction sequences, they formulated a binary classification problem and built tree based classifiers (Decision Tree, Bagging and Random Forest). For experimental analysis, a data set of 2774 (1444 worms and 1330 benign files) are used and detected 95.6% malware in the dataset (not used in building the model). In 2009, S. Momina Tabish et al. [108] proposed a non-signature based technique which analyzes the byte-level file content. They claimed that such technique provides implicit robustness against common obfuscation techniques, and also the framework can classify the given malware family. Their scheme uses the rich features set of 13 different statistical and information-theoretic features, computed on 1 - 4 grams of each file. They have tested their approach with 37,420 malware dataset from VX heaven and 1,800 benign files from different Desktops, and achieved 90% detection accuracy.

In 2009 Syed Bilal Mehdi et al. [64] advocated the need of sophisticated,



efficient, and accurate malware classification techniques that can detect a malware on the first day of its launch, called In-Execution Malware Analysis and Detection (IMAD) that not only identify the zero-day malware without any apriori knowledge but can also detect a malicious process with 90% accuracy while it is executing. Their results indicates that IMAD can achieve better accuracy with significantly lower false alarm rate. They also analyzed the fraction of n-grams required by IMAD to classify an executing malware and shown that approximately  $\sim 50\%$  of malicious processes are classified within first 20% of their execution. Later on, they [63] proposed hyper-grams, a variable-length system call representation scheme, that uses IMAD for zero-day malware detection. Their experimental analysis with 72 benign and malware files shows that in-execution malware detector with hyper-gram representation achieves low processing overheads, but improves the detection accuracy compared to the conventional n-grams.

In 2011, Santos et al. pointed out that supervised machine learning is an effective method to detect the unknown malware, but are not efficient because it requires a significant amount of labeled executables for both malware and benign programs. Therefore, they proposed a single-class learning method for detecting unknown malware based on opcode occurrence, which does not require a large amount of labeled data [82]. They did an empirical study with the dataset comprising 1,000 malicious executables and benign executables each [114]. Their analysis shows that the single-class learning reduces the detection cost of the unknown malware. Additionally, they found that it is more important to obtain labeled malware samples than benign programs, and shown that by labeling 60% of the legitimate programs, one can achieve  $\sim 85\%$  accuracy. In 2012, Chandrasekar Ravi et al. [77] proposed an association mining based classification which yields higher detection accuracy than previous data mining methods, by employing Naive Bayes, Support Vector Machine and Decision Tree techniques. Their detection system uses API call sequence modeled by the third order Markov chain, and is an iterative learning process combined with the run-time monitoring of program execution behavior to make it as a dynamic malware detection system. They compared the accuracy of the proposed malware detection system with the existing data mining methods and claimed that their proposed system outperforms (90% of accuracy) the existing malware detection systems. In 2013, Chatchai Liangboonprakong

et al. [61] proposed a classification of malware families based on N-grams sequential pattern features. They conducted the experiment with four different sizes of n-grams ( $n = 1, 2, 3,$  and  $4$ ) and 3 classification models (C4.5 Decision Tree, Artificial Neural Network, and Support Vector Machine). From the analysis they concluded that due to the complexities of malware, the larger n-gram size yields the higher accuracy, and the proposed feature extraction methods achieves 96.64% accuracy with 4-gram and Support Vector Machine.

In 2013, Santos et al. proposed a method to detect unknown malware, based on the occurrence of opcode sequences, and also described a technique to mine the relevance of each opcode. They experimented with a malware dataset of 13,189 [114] and 13,000 benign executables from different systems and applications viz. text processors, drawing tools, windows games, Internet browsers, etc. They claimed that their method provides a good detection ratio of unknown malware with a low false positive ratio. In addition, they provided an empirical validation of the method, which is capable of detecting unknown malware, and found that SVM provides an accuracy of 92.92% and 95.90% for one opcode and two opcode sequence length respectively [81]. To identify the malicious file Zahra Salehi et al. [80] generated three feature set based on the runtime behaviour of malicious and benign files from the monitored log files, including API names, their input arguments and a combination of this files. Then the features were reduced by removing irrelevant and redundant attributes, which do not have a significant impact on modeling the binaries. They conducted the experiment with 385 benign (collected from Windows system files as well from the wide range of portable benign tools) and 826 malware files (collected from seven categories: backdoors, constructors, exploits, hacktools, Peer-to-Peer (P2P) worms, trojans and viruses). For the classification, they used Rotation Random Forest, Random Forest, J48, FT and Naive Bayes classifiers, and 10-fold cross-validation for training and testing. They obtained highest True positive rate (94.6%) from Random forest by taking only API calls as features, whereas when the only argument was taken as features they found 98.1% true positive rate [80].

### 3.3 Survey on the Detection of Android Malicious Apps

Similar to the Detection of Windows Desktops malware, static and dynamic analysis are the two main approaches applied for the detection of Android malicious apps [53], and in both the approaches classifiers are trained with a known dataset for the classification of apps. In static analysis, without executing the apps, the codes are analysed to find a malicious pattern by extracting the features such as permissions, APIs used, control flow, data flow, broadcast receivers, intents, hardware components etc. Whereas, in the dynamic analysis the apps are examined in the runtime environment by monitoring the dynamic behaviour (resources usage, tracing system calls, API call, network traffic, etc.) of the apps and the system response.

In Android, the application can share their code and resources with other applications, but sharing is tightly controlled by the permissions. However, in general, users do not understand what applications will do with their data/resources, and therefore not able to decide which permissions shall be allowed to the application to run with. Therefore Fuchs, et al., [37]. developed a tool called SCANDROID (suppose to be the first program analysis tool for the Android based devices) which can extract security specifications from manifests that accompany such applications, and checks whether data flows through those applications are consistent with those specifications or not.

In 2012 Sanz. et al. [83] proposed an approach that can automatically characterise the different types of applications to detect the Android malicious apps by categorizing Android applications through machine-learning techniques. For the classification, their feature sets were (i) frequency of the occurrence of printable strings, (ii) different permissions of the application, and (iii) permissions of the application extracted from the Android Market. Their experiment with 7 different categories (820 samples) and five classifiers (DT, KNN, BN, RF and SVM) shows that among the selected five classifiers, Bayes Tree Augmented Naive Bayes (TAN) was the best classifier (0.93 Area Under the ROC Curve), while Random Forest was the second best classifier with an AUC of 0.9. Among the analysed classifier, Decision Tree with J48 perform worst (0.64 AUC).

In 2012, Wu. et al. proposed *DroidMat*, a static feature-based mechanism for the detection of Android malware which analyzes AndroidManifest.xml and the systems calls. For the experiment, they used 238 malicious and 15000 benign programs and claimed that their approach is effective (97.87% accuracy), scalable and efficient [117]. In 2013 Michael Spreitzenbarth et al. proposed a Mobile-Sandbox to automatically analyze the Android apps in two steps. In the first step (static analysis), applications Manifest files are parsed and decompiled, then it is determined that the applications are using suspicious permissions/intents or not. In the next step, sandbox performs the dynamic analysis, where the application was executed in order to log all performed actions including those stemming from the native API calls. They evaluated the system on more than 36,000 applications from Asian third-party mobile markets and found that 24% of all applications actually use native calls in their code. [102]. Min Zheng et al. [123] proposed a signature based analytic system called *DroidAnalytics* for collecting the malware automatically, and to generate signatures for the identification of the malicious code segment. They conducted extensive experiments with 150,368 Android applications, and successfully determine 2,494 Android malware from 102 different families, in which 342 of them being zero-day malware samples from six different families. They claimed that their methods have significant advantages over the traditional MD5 hash based signature, and can be a valuable tool for the classification of Android malicious apps.

In 2014, Quentin et al. proposed a feature based detection mechanism relying on opcode-sequences. They tested the machine learning algorithms such as libsvm and SVM classifier with the reduced data set (11,960 malware and 12,905 benign applications) and obtained 0.89% F-measure. However, their approach is not capable to detect completely different malware [50]. In 2014, Kevin Allix et al. [10] devised several machine learning classifiers that rely on the set of features that are built from the applications control flow graphs (CFGs). They analysed their techniques with a sizeable dataset of over 50,000 Android applications and claimed that their approach outperformed existing machine learning approaches. Also from the analysis, they concluded that the 10-fold validation on the usual dataset sizes is not a reliable performance indicator for the realistic malware detectors.

Smartphone can act as a zombie device, controlled by the hackers via command and control servers. It has been found that mobile malware are targeting Android devices to obtain root level access to execute instructions from the remote server. Hence, such type of malware can be a big threat to homeland security. Therefore Seo, et al. [86] discusses the defining characteristics inherent in mobile, and shown the feasible mobile attack scenario against the Homeland security. They analyzes various mobile malware samples viz. banking, flight tracking and booking, home and office monitoring, from both the official market and the unofficial markets to discover the potential vulnerabilities. Their analysis discovered that two banking apps (mellat.bank and axis.bank app) were disguised to charge SMS for malicious purposes and two banking apps (KFH.bank and S.bank app) was modified to get the permissions without consent. Finally, they discusses a methodology that mitigates mobile malware threats against the Homeland Security.

In 2015, Jehyun Lee et al. proposed an Android malware detection mechanism that uses automated family signature extraction and family signature matching. They claimed that compare to previous behavior analysis techniques for the family detection, their proposed family signature matching have higher detection accuracy and can detect new variants of known malware more efficiently and accurately than the legacy signature matching. The experimental analysis was done with 5846 real world Android malware samples belonging to 48 families, collected in April 2014 and achieved 97% accuracy. In 2015, Smita Naval, et al. [71] addressed the problem of system-call injection attack, which allows the malicious binaries to inject irrelevant and independent system-calls during the program execution, thus modifying the execution sequences defeating the existing system-call-based detection. Therefore, they proposed an evasion-proof solution that is not vulnerable to system-call injection attacks. Their approach characterizes program semantics using asymptotic equipartition property, which allows to extracting the information-rich call sequences that are further quantified to detect the malicious binaries. Their analysis demonstrated that the semantically-relevant paths can be used to infer the malicious behavior and also to detect the numerous new and unseen malware samples. They claimed that the proposed solution is robust against the system-call injection attacks and are effective in identifying the real malware instances.

In 2016, Saracino, et al. [84] proposed a Multi-Level Anomaly Detector for Android Malware (*MADAM*), a host-based malware detection system for Android devices which simultaneously analyzes and correlates features at four levels: kernel, application, user and package to detect and stop malicious behaviours of 125 existing malware families, encompasses most of the known malware. *MADAM* takes into account of behaviours characteristics for almost every real malware which can be found in the wild. They claimed that it can block more than  $\sim 96\%$  of malicious apps, which come from the three large datasets. Their analysis on a testbed of 9,804 genuine apps shows low false alarm rate, negligible performance overhead, and limited battery consumption (4% energy overhead). They claimed that *MADAM* is the first system which aims to detect and stop any kind of malware at run-time, without focusing on a specific security threat by using a behavior-based and multi-level approach. BooJoong et al. [52], proposed an n-opcode based static analysis for the classification and categorising the Android malware. Their approach does not utilize the defined features viz. API calls, permissions intents, and other application properties, rather it automatically discovers the features that eliminate the need of expert or domain knowledge to find the required features. Empirically they showed that by using frequency n-opcodes with low n, good classification accuracy can be achieved. For  $n = 3$  and  $n = 4$ , they achieved maximum F-measure of  $\sim 98\%$  for both malware classification and categorization.

Based on inter-component communication (ICC) related features, Ke Xu, et al. [119] proposed a malware detection method called ICCDetector, which can capture the interaction between the components or cross application boundaries. They evaluated the performance of their approach with 5264 malware and 12026 benign apps and achieved an accuracy of 97.4%. Also, after manually analyzing false positives, they discovered 43 new malware from the benign data set and reduced the number of false positives to seven. Jae-wook Jang, et al. [49] proposed *Andro-Dumpsys*, a feature-rich hybrid anti-malware system, which can detect and classify the malware samples of similar behaviour groups. Their experimental results demonstrate that the *Andro-Dumpsys* is scalable and performs well in detecting the malware and classifying the malware families with low false positives and false negatives. It is also capable of responding zero-day threats as well. Gerardo Canfora, et al. [23] evaluated two techniques for detecting the Android malware; the first one was based on Hidden Markov Model, while the second

one exploits structural entropy. They claimed that their approach is effective for PCs viruses, and are also successful to classify the malicious apps. Experimentally they obtain a precision of 0.96 to discriminate a malware application and 0.978 to identify the malware family [23]. Sanjeev Das, et al. [33] proposed a hardware enhanced architecture, *GuardOL* to detect the malware at runtime. It is a combined approach, uses processor and field-programmable gate array. Their approach first extracts the system calls and constructs the features based on the high-level semantics of malicious behavior. Then the features are used to train machine learning classifier and multilayer perceptron to detect the malware at runtime. The advantage of their design was that the approach can detect 46% of malware within first 30% of their execution, while 97% of the samples at 100% of their execution, with 3% false positives. [33].

Recently, Ali Feizollah, et al. [36] proposed AndroDialysis to evaluate the effectiveness of Android Intents (explicit and implicit) as a distinguishing feature to identify the malicious apps and shown that Intents are semantically rich features to encode the intentions of malware when compared to other well-studied features, such as permissions. They also argue that this type of feature is not the ultimate solution, and it should be used in conjunction with other known features. They conducted experiments using a dataset of 7406 applications that comprise 1846 clean and 5560 infected applications. Their approach obtained a detection rate of 91% using Android Intent, 83% using Android permission and by combining both the features they obtained the detection rate of 95.5%. They claimed that for the malware detection, Android Intent is indeed more effective than Android permission. Bahman Rashidi et al. [76] proposed an Android resources usage risk assessment called *XDroid*. They showed that the use of temporal behavior tracking can significantly improve the malware detection accuracy, and the HMM can generate security alerts when suspicious behaviors are detected. Also, they claim that their model can inform users about the risk level of their apps in real-time, and can dynamically update the parameters of the model by using an on-line algorithm and users preferences. They conducted the experiment on the *Drebin* benchmark malware dataset and demonstrated that the proposed model can accurately ( $\sim 82\%$ ) assess the risk levels of malicious applications and provide adaptive risk assessment based on user input.

### 3.4 Summary

Time to time, number of static and dynamic methods are proposed by the researchers for the detection of Windows Desktops and Android based devices has been discussed in this Chapter. However, to evade the detection techniques the malware writers changes the obfuscations more than the behaviour of the malware. From the literature survey, we understand that all the malware are not built with the same functionality. Therefore, if one can separate the malware family with its functionality then the malware detection accuracy can be improved. Hence, in this thesis, we systematically develop and proposed new techniques to detect the advanced malware with high accuracy for the Windows Desktops and Android based devices.



## CHAPTER 4

# GROUP-WISE CLASSIFICATION FOR THE DETECTION OF WINDOWS DESKTOPS MALWARE

### 4.1 Introduction

The advanced malware, in particular, metamorphic variants with the same malicious behavior (family), can obfuscate themselves to look different from each other. Therefore, the prolong traditional signature matching technique to combat the threats/attacks from such an advanced malware is no more effective. Also, to detect the unknown malware with high accuracy is always a challenging task. Therefore, to detect the advanced malware with high accuracy, in this Chapter we study variation in the size of malware generated by the three popular advanced malware creator kits viz. Next Generation Virus Creation Kit (NGVCK) [112], Phalcon-Skism Mass-Produced Code Generator (PS-MPC) [17] and Second generation virus generator (G2) [16], and then classified the unknown malware by two methods. In the first/regular method (without grouping), similar to other authors [77] [64] [68] approaches, we selected the features by taking all the dataset together, and in the second method, we selected the features after group-wise partitioning the executables in 5 KB size for the classification of unknown malware.

## 4.2 Data Preprocessing

For the study, we downloaded 11088 malware (*2014*) from the Malicia project [69] and collected 4006 benign programs (also verified from [virustotal.com](http://virustotal.com) [24]) from different Windows Desktops. In the malware dataset, we observed that 97.18% malware are below 500 KB (Figure 4.1), hence for the analysis, we took the dataset (both malware and benign executables) which are below 500 KB. Then we converted all the selected executables (10558 malware and 2454 benign) to their assembly codes by *objdump* utility available in the Linux system and found that the executables are basically based on 1147 opcodes. To simplify the analysis we uniquely mapped the opcodes with a fixed integer (Appendix B).

For the classification, we investigated the size of the malware variant generated by the advanced malware generator kits, by generating 100 malware each from the NGVCK, PS- MPC and G2 (fig 4.2 - 4.4). From the figure, we observed that the

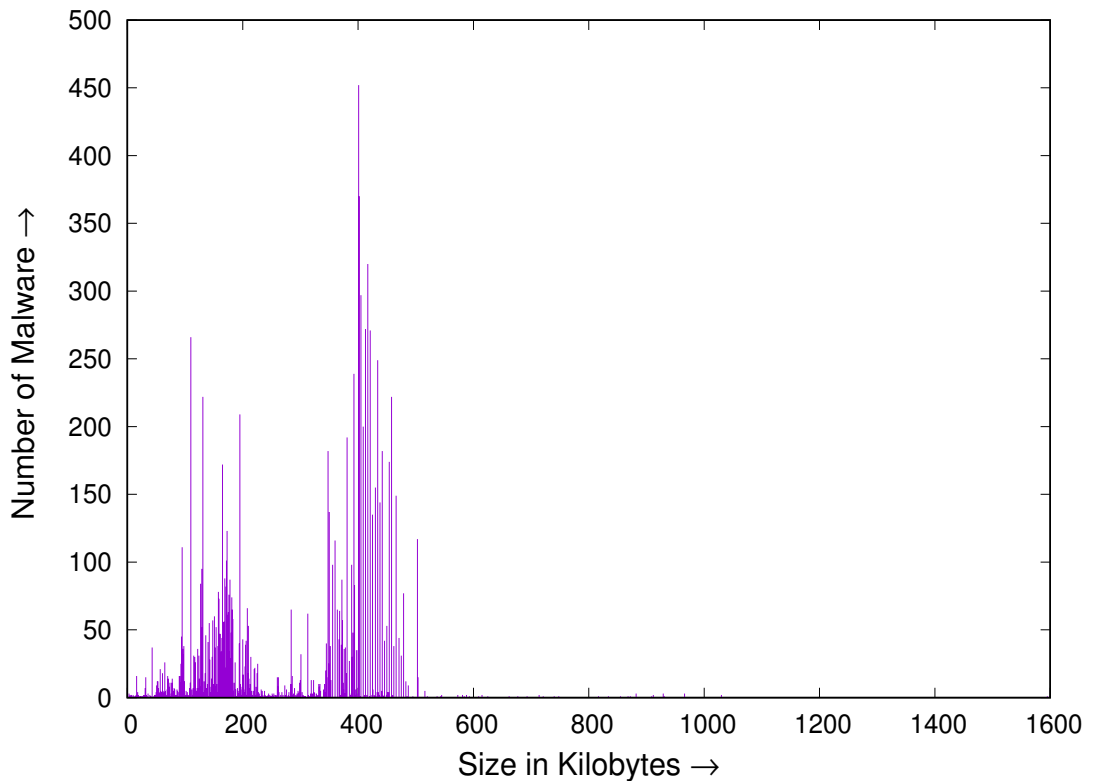


Figure 4.1: Number of malware with respect to the file size.

variation in the size of malware generated from same kits are within 5 KB. Hence, we partitioned the data in 100 groups in 5 KB range. To compare the regular method with the partitioning method we randomly selected 750 malware and 610 benign programs from all the groups, such that at least five executables from each group can be randomly tested by the trained classifiers.

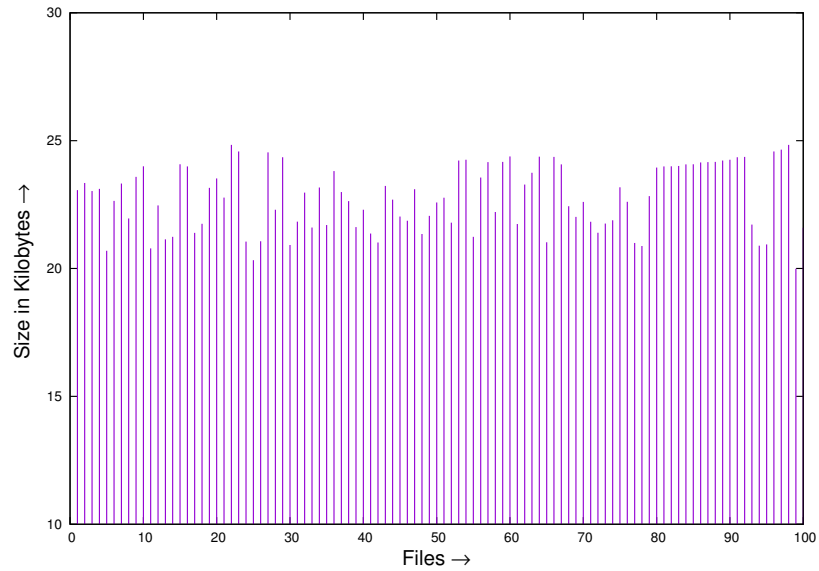


Figure 4.2: *Fluctuations in the size of malware generated by NGVCK kit.*

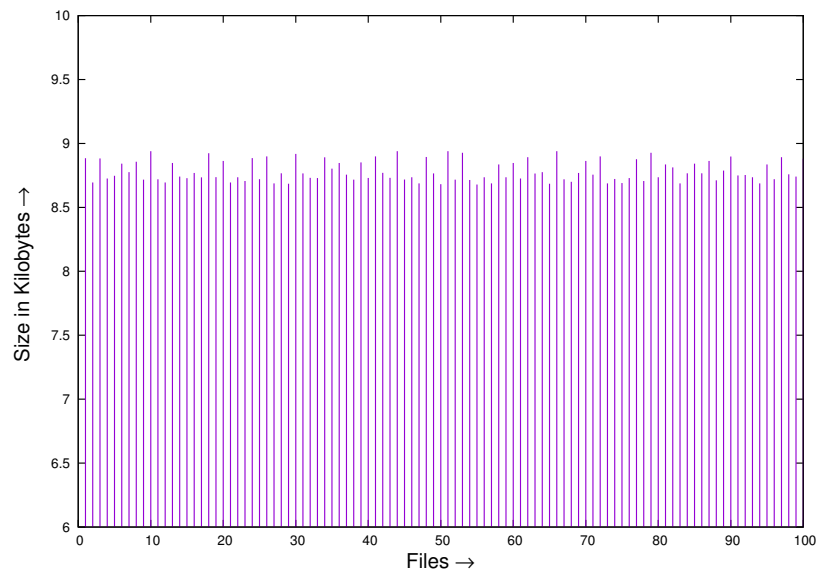


Figure 4.3: *Fluctuations in the size of malware generated by G2 kit.*

### 4.3 Feature Selection

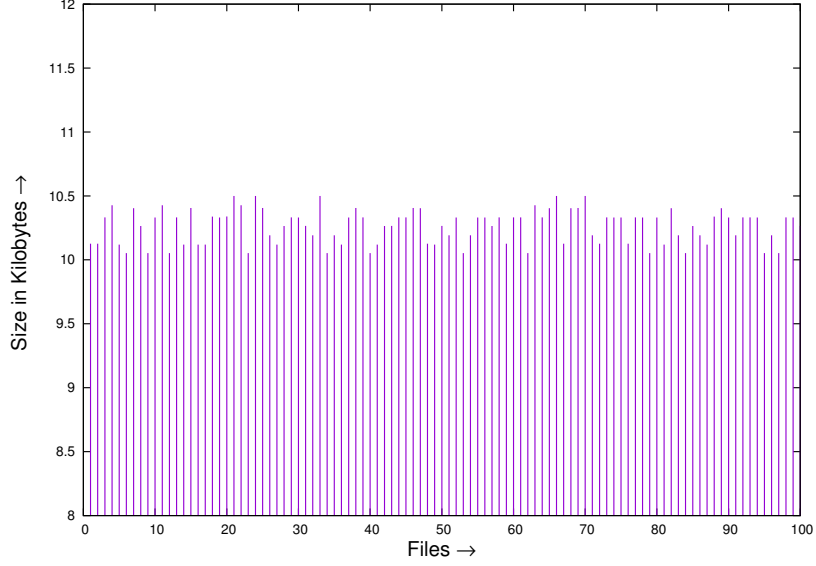


Figure 4.4: *Fluctuations in the size of malware generated by PS-MPC kit.*

For the detection of malware, feature selection plays a vital role, not only to represent the target concept but also to speed-up the learning and testing process. In this, often datasets are represented by many features, however, few of them may be sufficient to improve the concept quality, and also limiting the features will speed-up the classification. The analysis (Sec. 4.2) shows that the executables are based on 1147 opcodes (Appendix B), hence executable can be represented as a vector of 1147 opcodes, and some of these opcodes can be used as features for the effective and efficient detection of malware. Therefore, to find the prominent features which can represent the target concept, opcodes of the collected executables are obtained by using freely available *objdump* utility available in the Linux system. Then we computed the normalized opcode occurrence of the malware and benign programs in all together (i.e. without grouping) and also for each group separately. We observed that the opcode occurrence in the malware and benign programs without grouping the executables differ in large (Figure 4.5) as well in each formed groups, e.g. Figures 4.6, 4.7, 4.8 and 4.9 shows the normalized opcode occurrence for the group 10-15, 140-145, 240-245 and 415-420 KB size respectively keeping the lower threshold 0.02. We also separated the opcodes of the datasets (without grouping) which occur more in malware and benign programs (Figures. 4.10 and 4.11). The above analysis shows that the opcode occurrence in any group differs significantly when compared with the opcode occurrence obtained without forming the groups and also between the groups (e.g. Figure 4.12

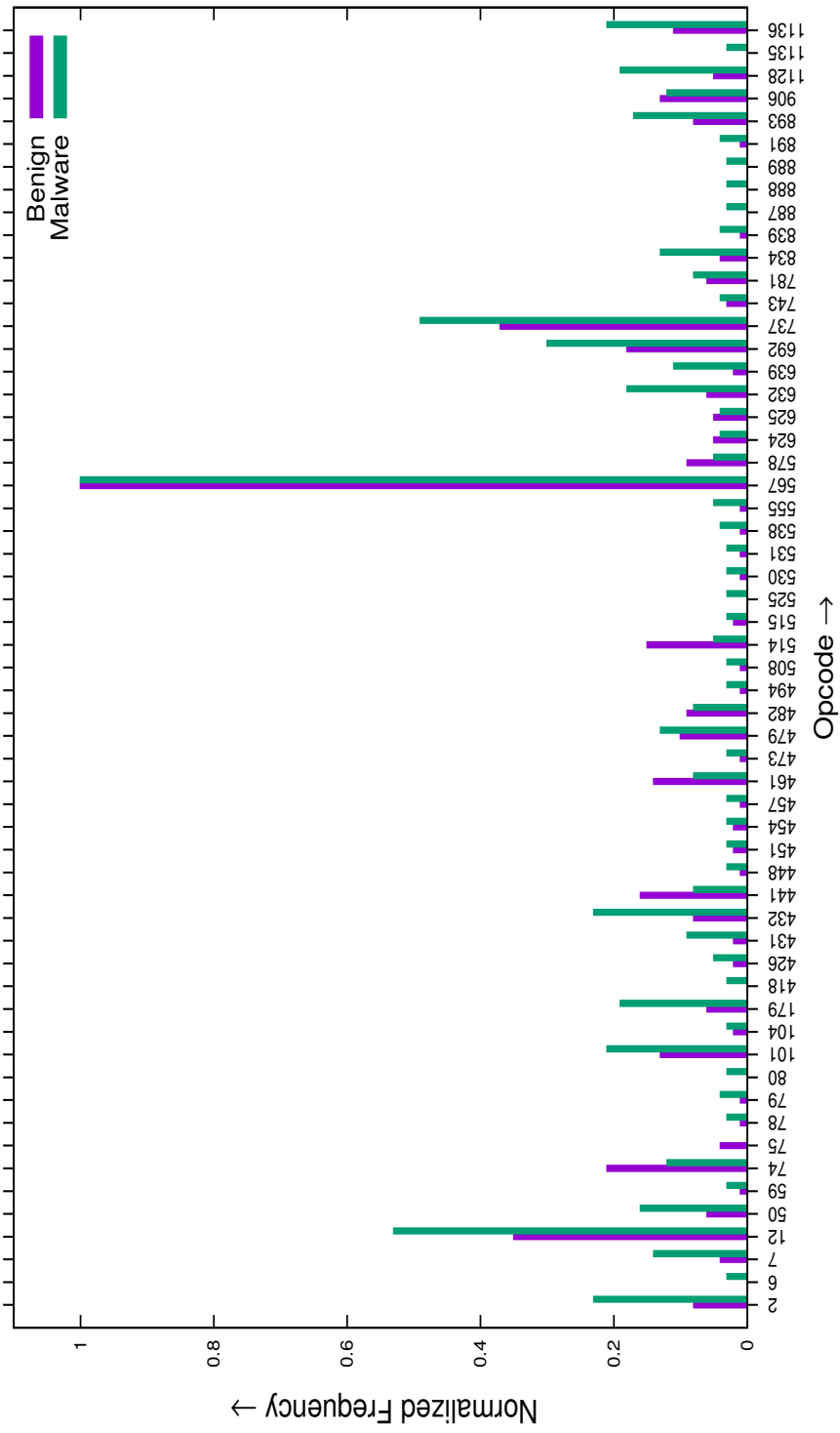


Figure 4.5: Normalized opcode occurrence of all the collected malware and benign program keeping the lower threshold 0.02.

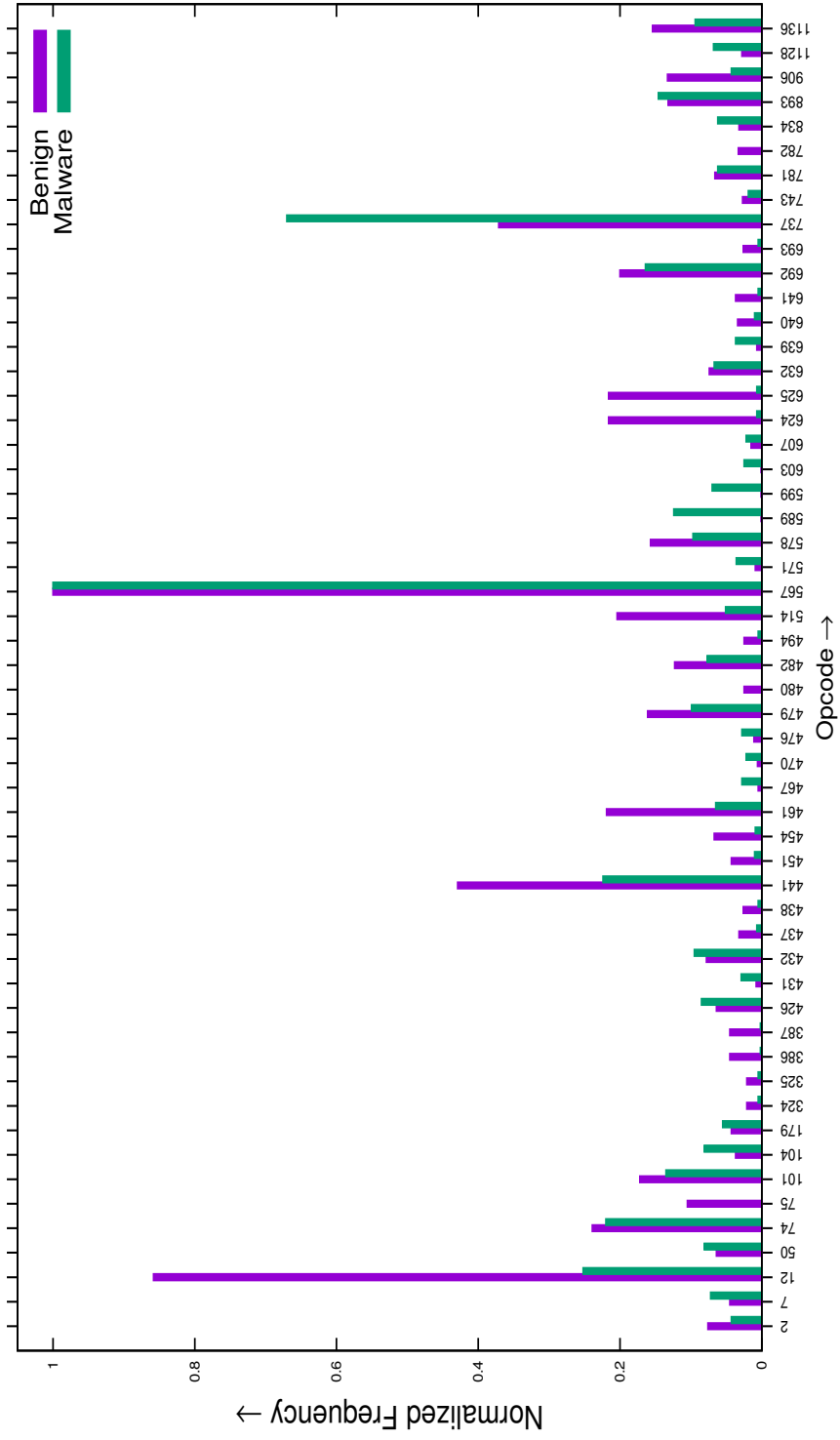


Figure 4.6: Normalized opcode occurrence of the malware and benign program of size 10-15 KB keeping the lower threshold 0.02.

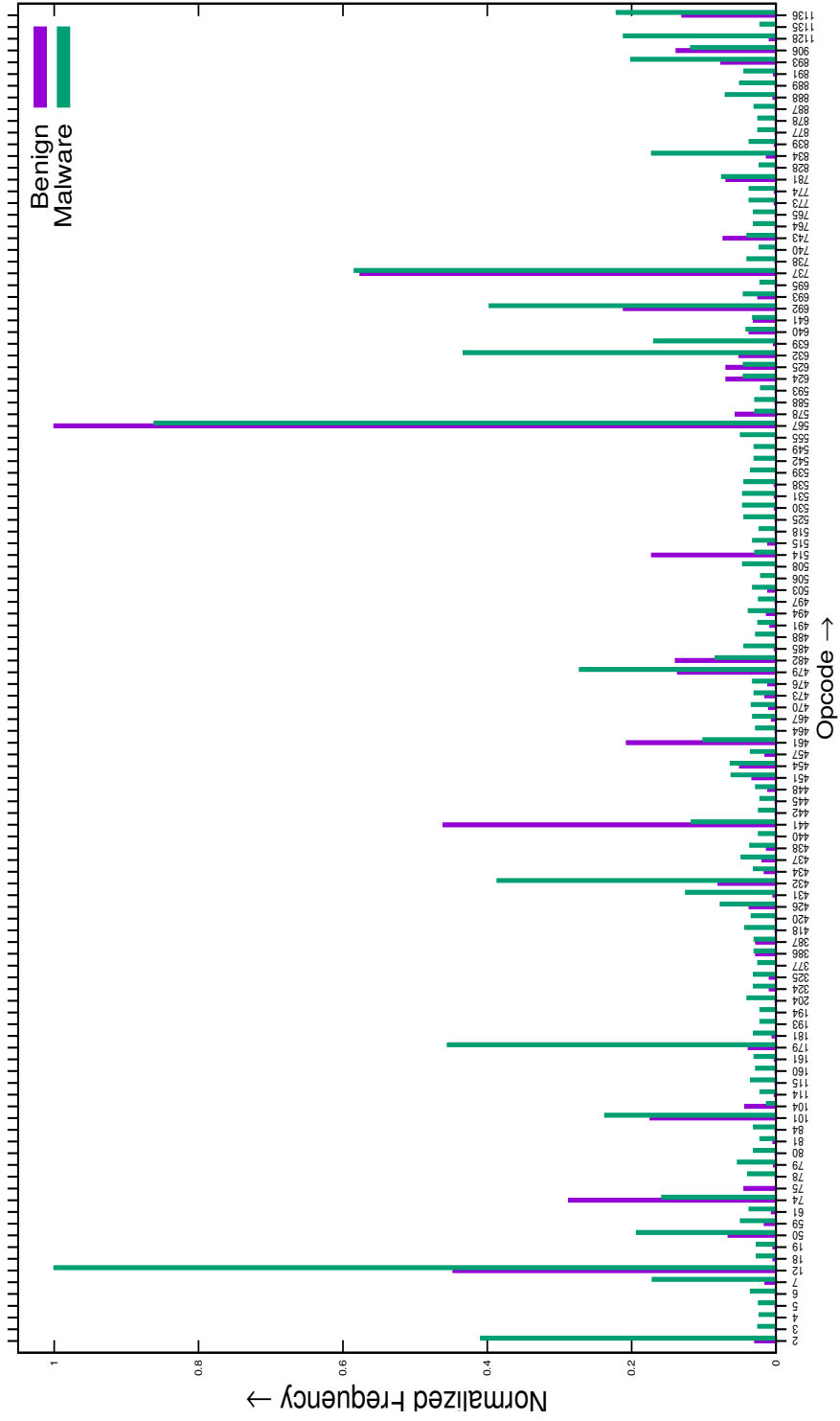


Figure 4.7: Normalized opcode occurrence of the malware and benign program of size 140-145 KB keeping the lower threshold 0.02.

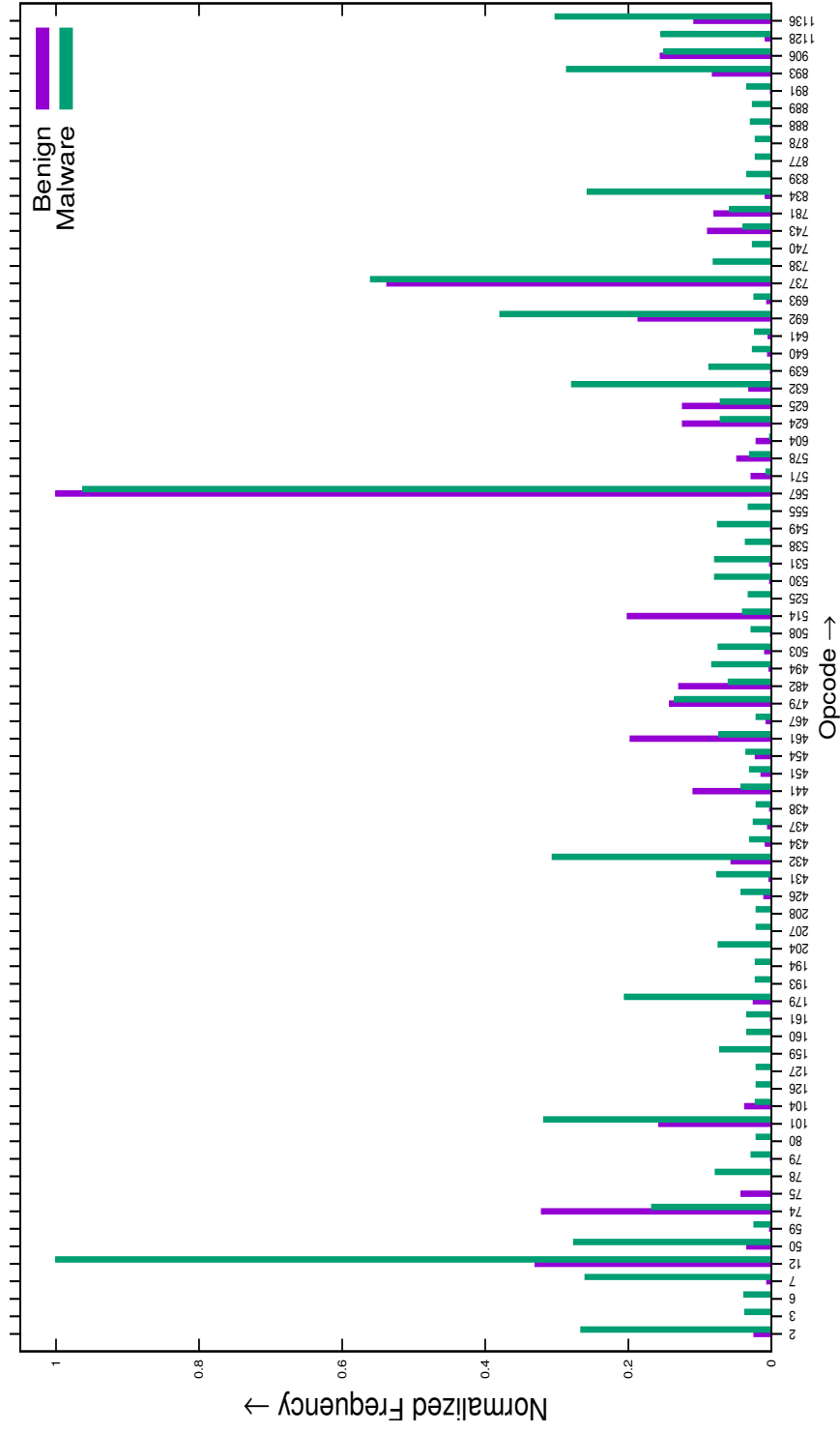


Figure 4.8: Normalized opcode occurrence of the malware and benign program of size 240-245 KB keeping the lower threshold 0.02.



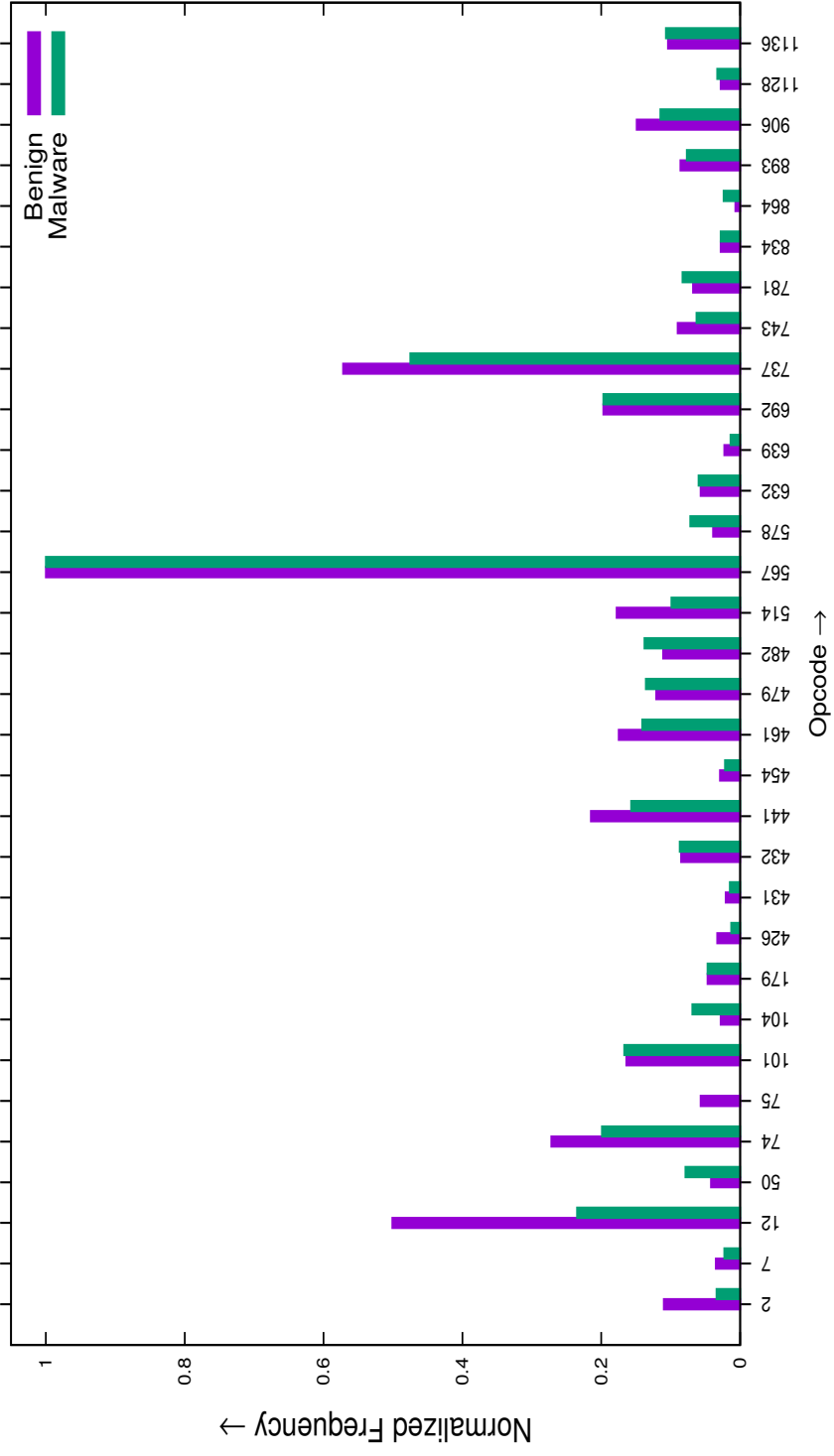


Figure 4.9: Normalized opcode occurrence of the malware and benign program of size 415-420 KB keeping the lower threshold 0.02.

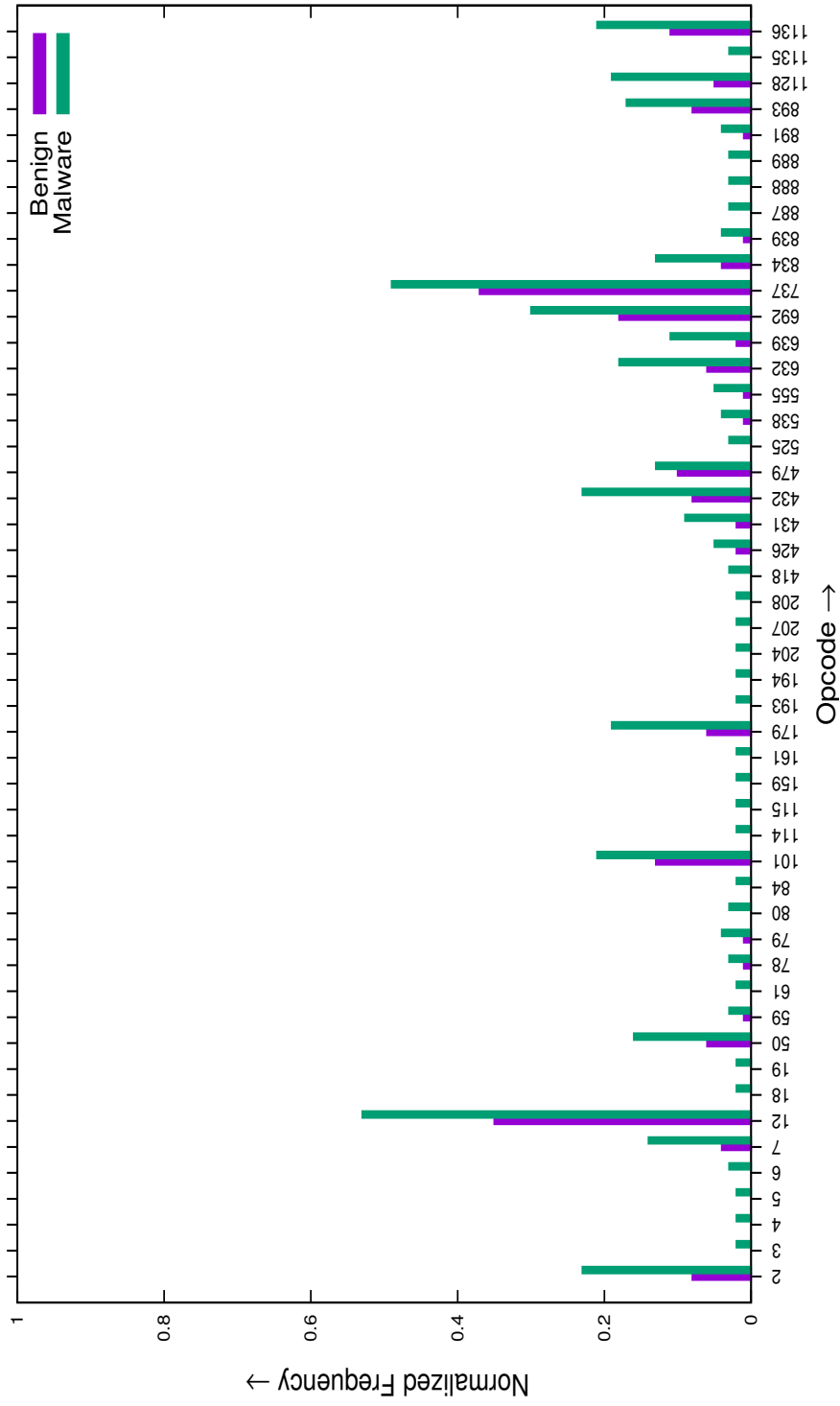


Figure 4.10: Opcodes which found more in malware without grouping the dataset.

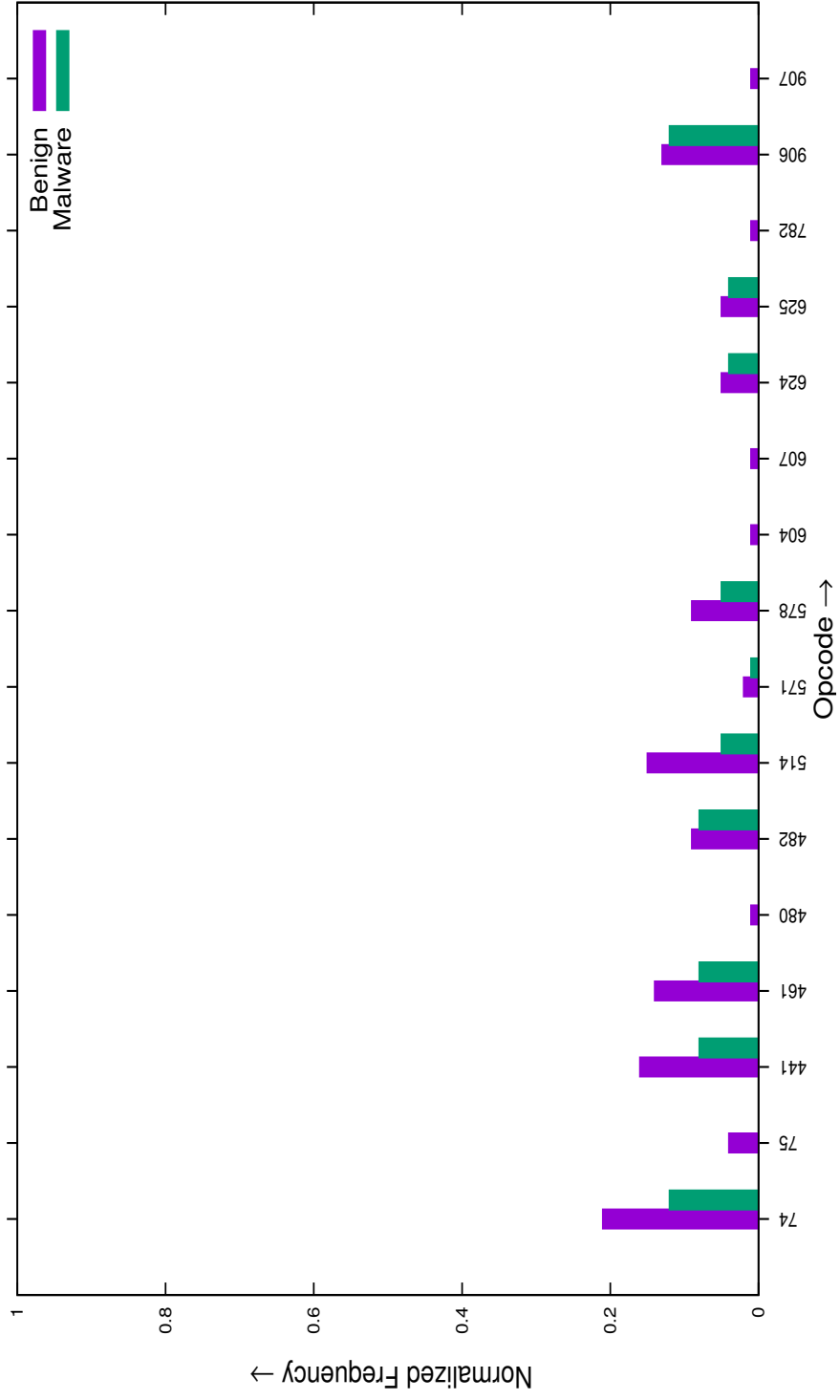


Figure 4.11: Opcodes which found more in benign programs without grouping the dataset.

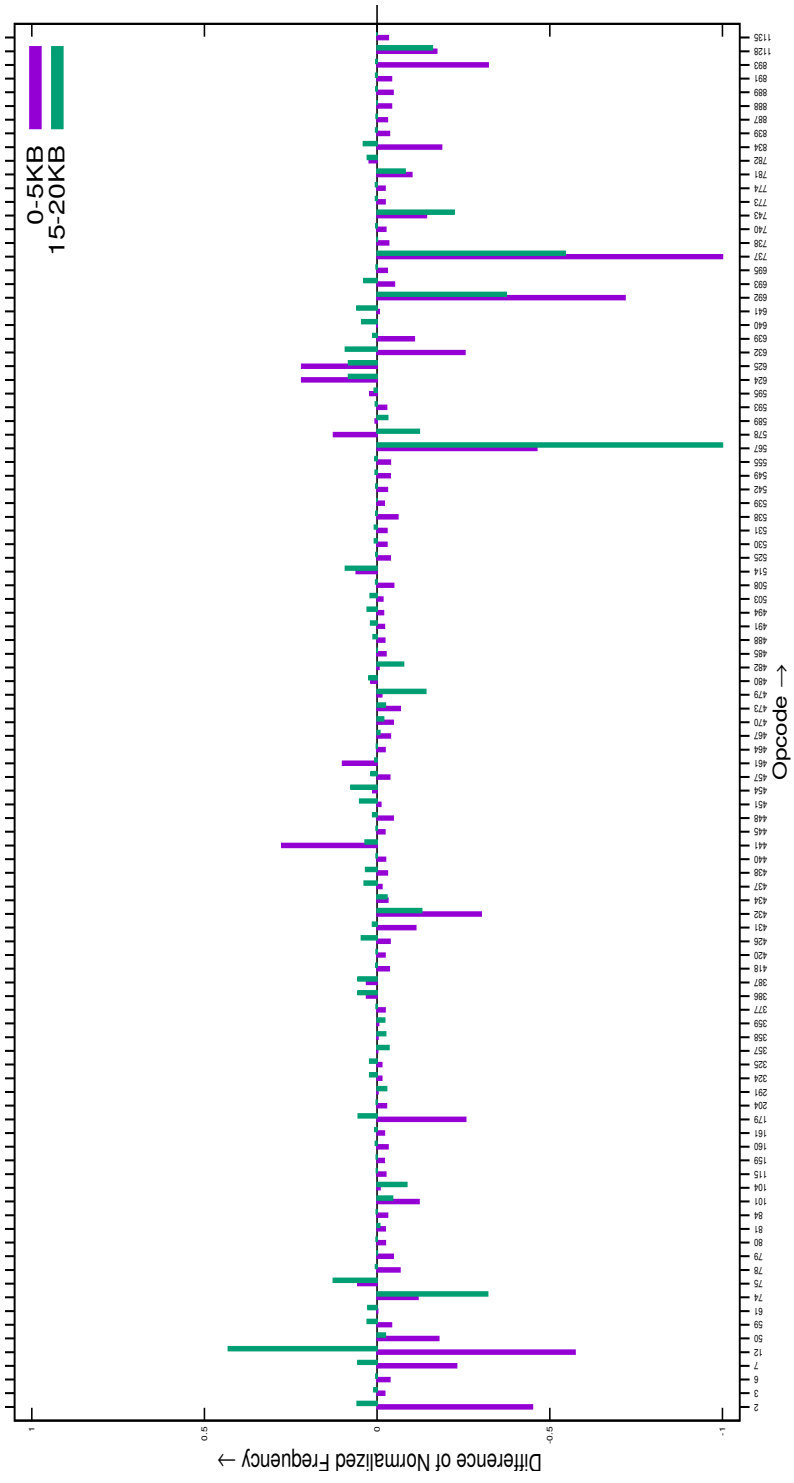


Figure 4.12: Difference in the occurrence of respective opcodes between benign and malware program of size 0-5 KB and 15-20 KB keeping threshold 0.02.

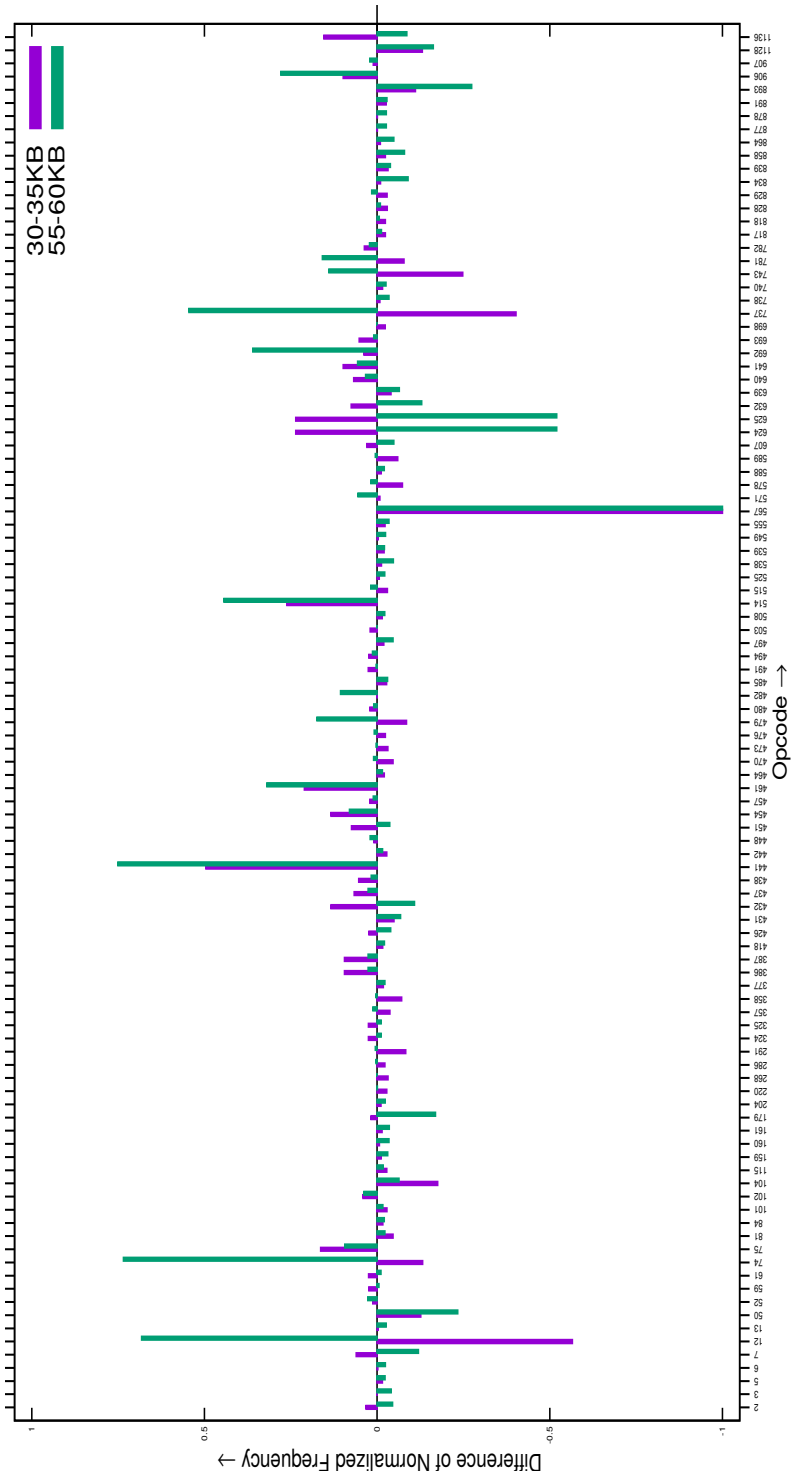


Figure 4.13: Difference in the occurrence of respective opcodes between benign and malware program of size 30-35 KB and 55-60 KB keeping threshold 0.02.

and Figure 4.13 shows the difference in the occurrence of respective opcodes between benign and malware program of sizes 0-5 KB & 15-20 KB and 30-35 KB & 55-60 KB keeping the lower threshold 0.02). Finally, the features are selected after ordering the opcodes by their occurrence difference (Algorithm 4.1) in all together and in each group separately for the classification of malware by the regular method and group-wise partitioning the data respectively.

---

**Algorithm 4.1 : Feature Selection**

---

**INPUT:** Pre-processed data

$N_b$ : Number of benign executables,  $N_m$ : Number of malware executables,  $n$ : Number of features required

**OUTPUT:** List of features

**BEGIN**

**for all** benign data **do**

Add all frequency  $f_i$  of each opcode  $o$  and Normalize them with respect to  $N_b$

$$F_b(o_j) = (\sum f_i(o_j))/N_b$$

**end for**

**for all** malware data **do**

Add all frequency  $f_i$  of each opcode  $o$  and Normalize them with respect to  $N_m$

$$F_m(o_j) = (\sum f_i(o_j))/N_m$$

**end for**

**for all** opcode  $o_j$  **do**

Find the difference of each opcode normalized frequency  $D(o_j)$ .

$$D(o_j) = |F_b(o_j) - F_m(o_j)|$$

**end for**

**return**  $n$  number of opcodes with highest  $D(o)$ .

---

## 4.4 Detection of Unknown Malware

To study the classification of unknown malware by the regular method and our novel group-wise partitioning the data, we selected the malware and benign program of below 500 KB size (10558 malware and 2454 benign executables). For the classification, we used Naive Bayes classifier which can handle an arbitrary number of independent variables and is briefly described below.

#### 4.4.1 Naive Bayes classifier

Given a set of features (opcodes),  $O = o_1, o_2, o_3, \dots, o_n$ , the Naive Bayes classifier gives the posterior probability for class  $C$  (malware/benign) and can be written as

$$P(C|o_1, o_2, \dots, o_n) = \frac{P(C)P(o_1, o_2, \dots, o_n|C)}{P(o_1, o_2, \dots, o_n)} \quad (4.1)$$

where,  $P(C|o_1, o_2, \dots, o_n)$  is a posterior probability of the class membership, i.e. probability of a test executable that belongs to class  $C$ . Since Naive Bayes assumes that the conditional probabilities of independent variables are statistically independent, we can decompose the likelihood to a product of terms as

$$P(o_1, o_2, \dots, o_n|C) = \prod_{i=1}^n P(o_i|C) \quad (4.2)$$

Here,  $P(o_i|C)$  is the probable similarity of occurrence of feature  $o_i$  of class  $C$  and can be computed by the equation

$$P(o_i|C) = \frac{1}{\sqrt{2\pi\sigma_C^2}} e^{-\frac{(o - \mu_C)^2}{2\sigma_C^2}} \quad (4.3)$$

where  $o$  denotes the feature  $o_i$  opcode of the test executable and  $\mu_C, \sigma_C$  are the mean and variance of class  $C$ .

From the above, classification can be done by comparing the posterior probability between both the class models, if the malware class posterior probability of test executable is high then it is classified as malware else classified as benign programs.

#### 4.4.2 Regular Method

Figure 4.14 shows the regular method for the classification of unknown malware. In this method, for the classification, the promising features are obtained by computing the difference in the normalized opcodes frequency between benign and

malware executables (Figure 4.5). Then we trained the classifier by 9808 malware and 1844 benign executables and tested with 750 malware and 610 benign programs.

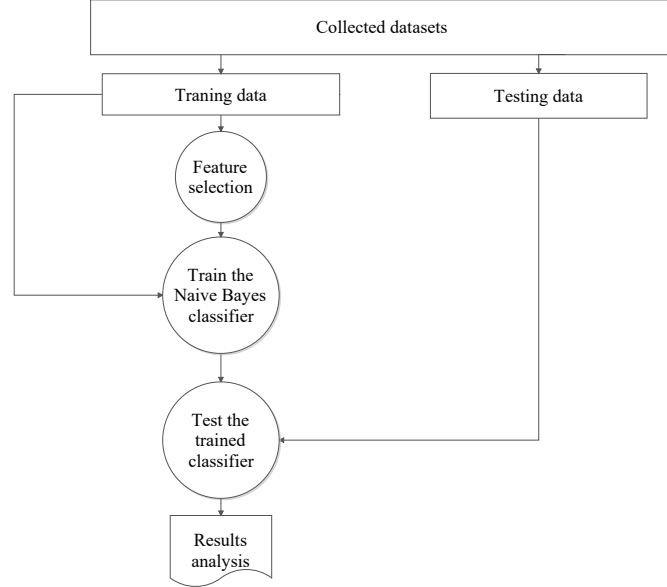


Figure 4.14: Flow chart for the detection of unknown malware without partitioning the datasets.

For the classification, we have used Waikato Environment for Knowledge Analysis (WEKA) (a well suited open source software for machine learning which contains the implementation of various data mining algorithms. It provides algorithms for data pre-processing, classification, regression, clustering, association rules and visualisation that are meant for the applications which can use machine learning techniques to solve various real-world problems [47]). We have chosen the Naive Bayes classifier for the study, and its performance is measured by computing the detection accuracy given as

$$Accuracy(\%) = \frac{TP + TN}{TM + TB} \times 100 \quad (4.4)$$

where,

$TP$   $\rightarrow$  True positive, the number of malware correctly classified.

$TN$   $\rightarrow$  True negative, the number of benign correctly classified.

$TM$   $\rightarrow$  Total number of malware.

$TB$   $\rightarrow$  Total number of benign.



The performance of the Naive Bayes classifier has been investigated with the testing data (750 malware 610 benign programs) which are not used for the training with 20 - 200 best features incrementing 5 features at each step, and the results obtained are shown in the Figure 4.16. We observed that the accuracy of the classifier is almost flat if the number of features is more than 90, and the best accuracy obtained by this method is 78.33%.

### 4.4.3 Group-wise Partitioning the Datasets

In this method as shown in Figure 4.15, we group-wise partitioned the collected dataset in the 5 KB size range. The partition size is based on the study (Sec. 4.2) that the size of malware generated by NGVCK, PS-MPC and G2 kits does not vary by more than 5 KB.

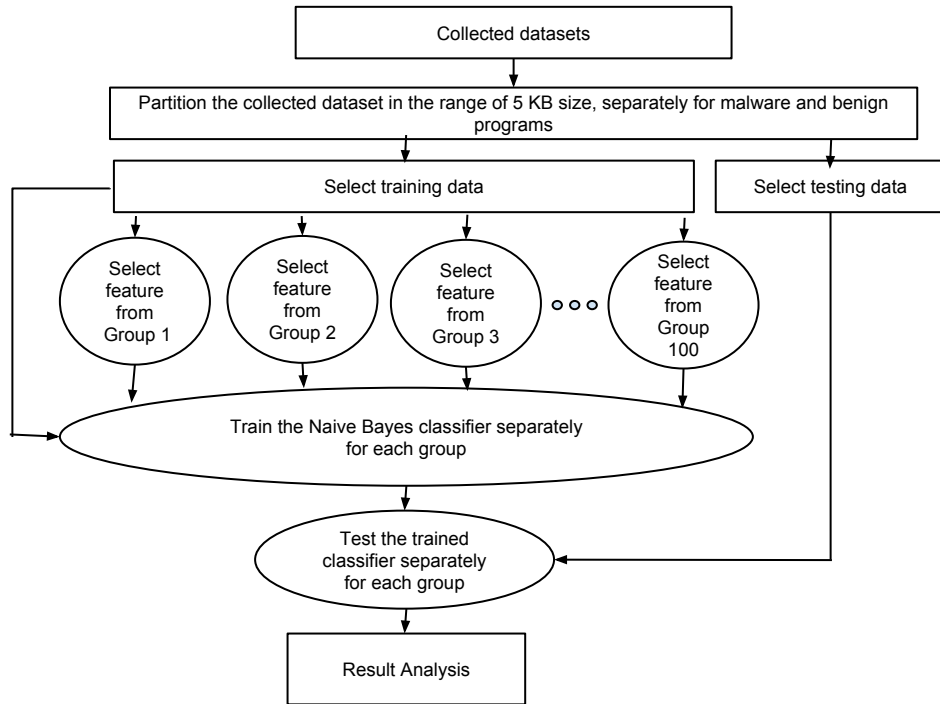


Figure 4.15: Flow chart for the detection of unknown malware by group-wise partitioning the executables.

For the comparative analysis, we took the same training and testing data which were used in the regular method. However, to improve the detection accuracy we

obtained the features from each group, then serially trained the Naive Bayes classifier and classified the test data which are not used for the training with 20 - 200 best features incrementing 5 features at each step, and the results obtained are shown in Figure 4.16. We found that the accuracy obtained by this method outperformed the regular method and the best accuracy obtained is 87.02% i.e. the detection accuracy is  $\sim 8.7\%$  more compared to the regular method.

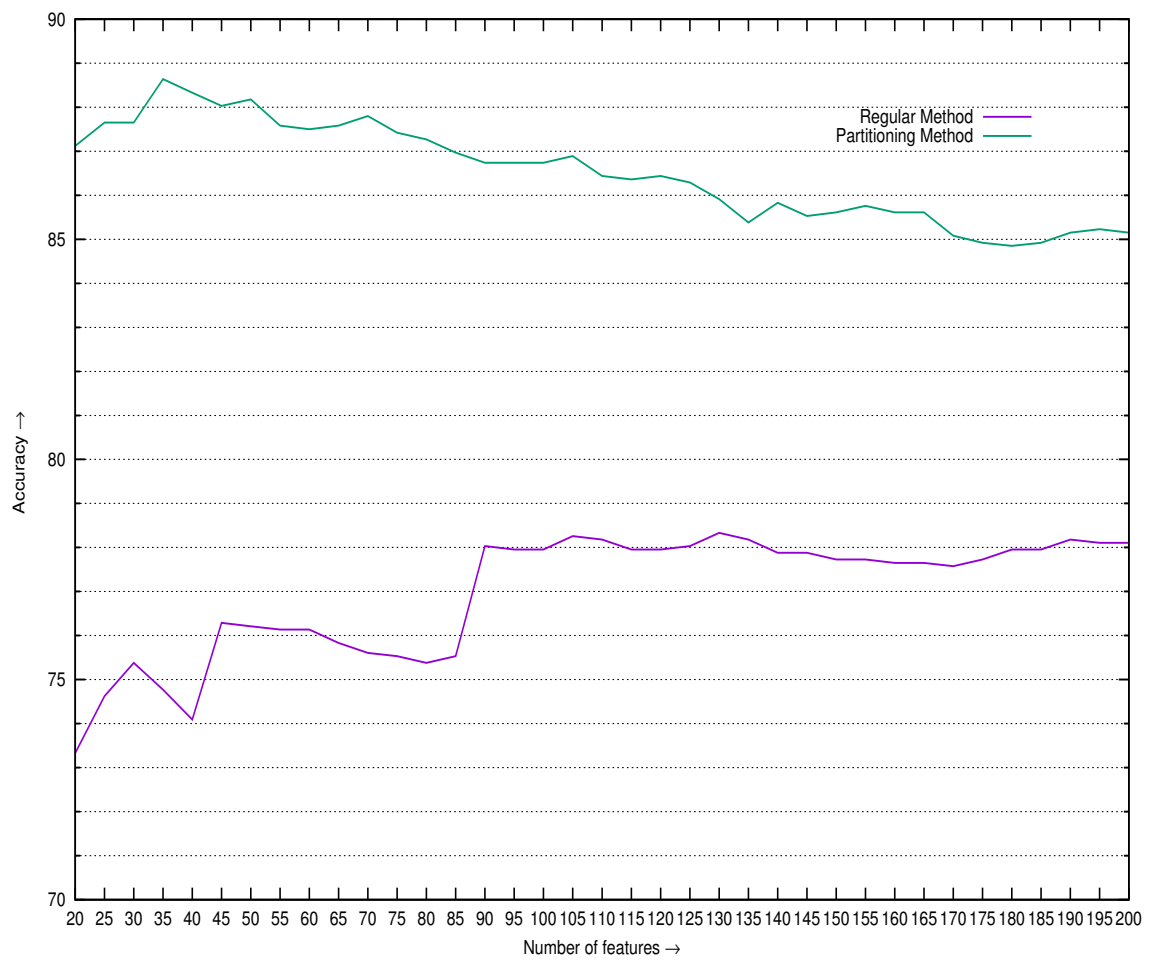


Figure 4.16: Detection accuracy obtained by both the methods.

## 4.5 Summary

For the detection of malware, feature selection plays a vital role, not only to represent the target concept but also to speed-up the learning and testing process. Therefore, we proposed an approach to find prominent features for the classification of malware. To improve the malware detection accuracy, we investigated the variation in the size of malware generated by G2, PS-MPC and NGVCK and found that the variation in the size of malware generated from the same kit is within 5 KB range. Therefore we partitioned the collected dataset in 100 groups, each in 5 KB range of size and then selected the feature from each group to train the classifier for the detection of unknown malware. We found that, if features are selected by group-wise partitioning the dataset in the range of 5 KB then the malware can be detected  $\sim 8.7\%$  more accurately than the regular method.

## CHAPTER 5

# CLASSIFIERS SELECTION AND K-MEAN CLUSTERING TO IMPROVE THE DETECTION ACCURACY

### 5.1 Introduction

In the previous Chapter, we demonstrated that group-wise classification significantly improves malware detection accuracy. Nevertheless, the detection accuracy also depends on the performance of the classifier. Therefore, in this Chapter, we study the performance of the popular thirteen classifiers viz. RF, J48, REPTREE, LMT, Decision stump, ADT, NBT, FT, LAD, Random Tree, Simple CART, BFT and J48 Graft using N-fold cross-validation (a popular statical procedure to estimate and compare the effectiveness of machine learning algorithms, in which dataset is randomly divided into N chunks of almost equal sizes and then classification model is trained and tested N times. Each time it is trained on (N - 1) parts and tested on the remaining single part. Finally, effectiveness/accuracy of the model is obtained by averaging the N individual accuracy [22]). Among these thirteen classifiers further we studied in-depth the top five classifiers (RF [21], J48 [19], LMT [58], FT [57] and NBT [55]) to improve the detection accuracy by grouping the executables using Optimal k-Means clustering algorithm and then classified the data with the promising features obtained from each of the cluster/groups. For the analysis same *Malicia* dataset (Sec. 4.2) and the feature

selection technique (Sec. 4.3) has been used.

## 5.2 Classifiers selection

In order to select the classifiers, we investigate the performance of classifier, (a schematic of our novel approach is shown in the Figure 5.1). It involves finding the promising features (Algorithm 4.1), training of classifiers and finally finding the detection accuracy of the classifiers. For the purpose, first we have chosen the popular thirteen classifiers and selected the effective features from the formed 100 groups (Sec. 4.3) by taking the union of the top ten features from each group. Then with these features the chosen popular thirteen tree based classifiers viz. RF, J48, REPTREE, LMT, Decision stump, ADT, NBT, FT, LAD, Random Tree, Simple CART, BFT and J48 Graft available in *WEKA* has been tested by N-fold cross-validation process.

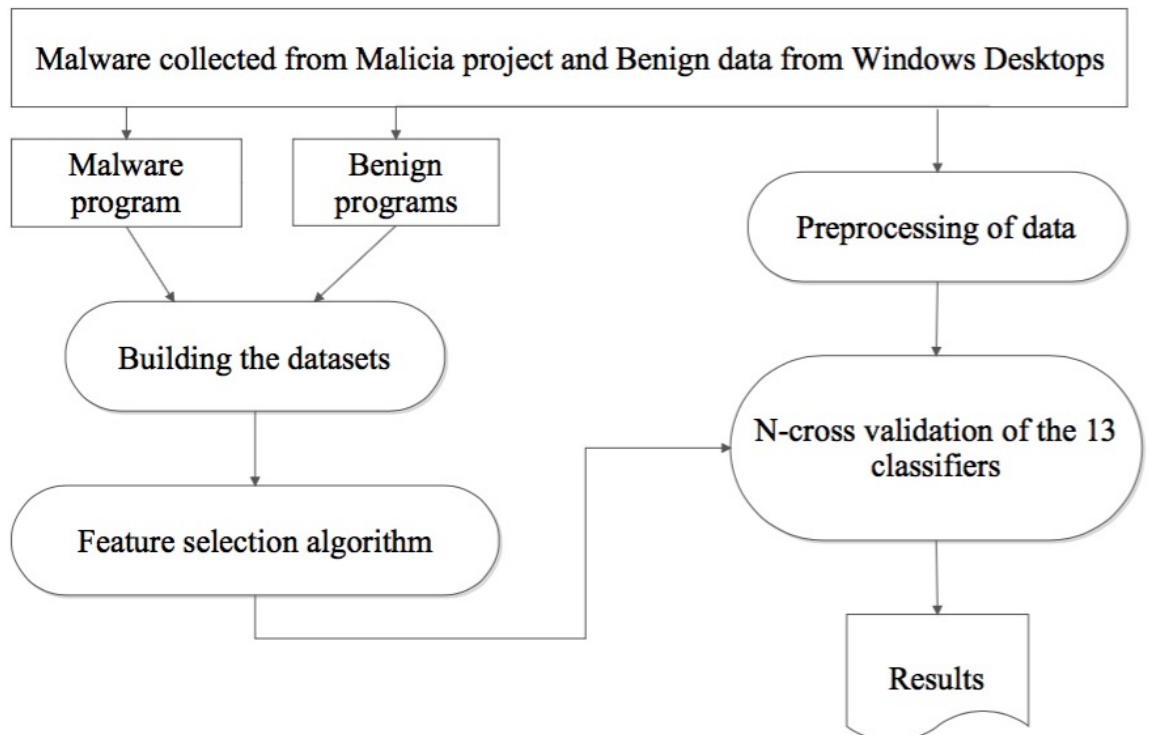


Figure 5.1: Flow chart for the detection of unknown malware.

Figure 5.2 shows the accuracy obtained by all the thirteen investigated classifiers for  $n = 2, 4, 6, \dots, 16$  folds. We observed that RF is the best classifier and its accuracy is almost flat after  $n = 2$ . Rest twelve classifiers accuracy fluctuates. However, after ten-fold cross-validation, the fluctuations in all the classifiers are least and observe maximum detection accuracy at ten-fold cross-validation by RF.

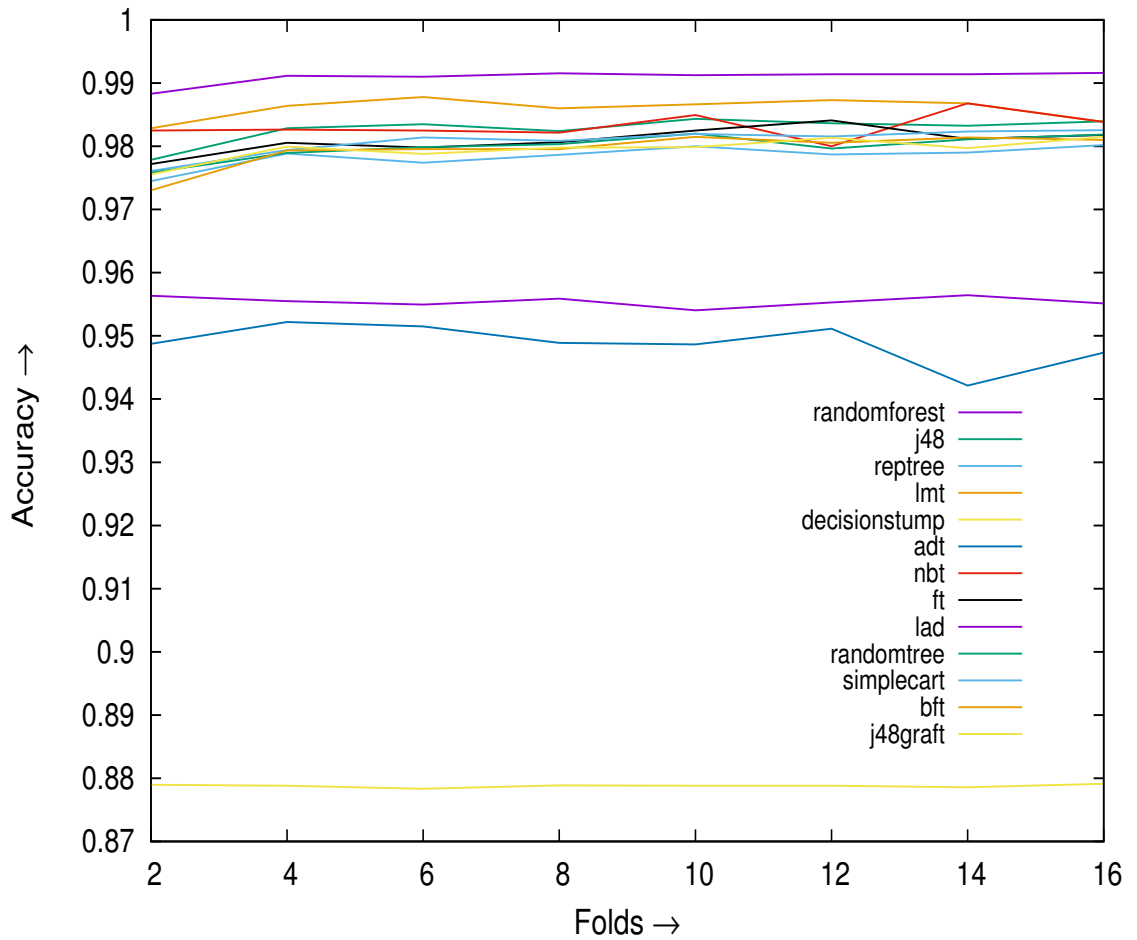


Figure 5.2: Accuracy of the thirteen classifiers with  $N$ -fold cross validation.

Now, among the thirteen investigated classifier, we selected the top five viz. RF, LMT, NBT, J48, and FT to understand its performance in view of that if our approach is implemented for detection of unknown malware, then what will be the accuracy of our approach. Therefore to study the overall performance of these five classifiers, we randomly selected 750 malware and 610 benign programs from all the

groups, such that at least five executables from each group can be randomly tested by the trained classifiers (training is done by 9808 malware and 1844 benign executables) for the detection of unknown malware. Table 5.1 shows the result obtained in terms of True Positive Ratio (TPR), True Negative Ratio (TNR), False Positive Ratio (FPR), False Negative Ratio (FNR), and the detection accuracy.

Classifiers	TPR	FNR	FPR	TNR	Detection Accuracy
RF	98.53	1.47	2.81	97.19	97.95
LMT	97.87	2.13	4.04	95.96	97.04
NBT	97.07	2.93	3.33	96.67	96.89
J48	97.20	2.80	4.04	95.96	96.66
FT	97.20	2.80	4.91	95.09	96.28

Table 5.1: Performance of the top five classifiers.

### 5.3 Result Analysis

From the investigation, it is clear that RF is the best classifier for identification of unknown malware. Nevertheless, the other classifiers are also reasonably good (accuracy  $> 96.2\%$ ) for the detection of unknown malware. We observed that the NBT, J48 and FT classifiers have almost same True positive ratio. In this, the overall accuracy of Functional Tree classifier is lowest, which is basically due to high False positive ratio. Figure 5.3 shows the variation of False positive and False negative of the studied classifiers. We found that False positives ratio of RF is almost double than the False positive ratio of LMT however, for overall accuracy both (FP and FN) has to be low. From Figure 5.4 we find that the True positives of all the classifiers are more than the TN, i.e. malware are more correctly classified then the benign programs. The best accuracy obtained by the selected five classifiers are shown in Figure 5.5, and the comparison of our results with Santos et al., Siddiqui et al., Asaf Shabtai et. al. for RF and Mehdi et al., Santos et al., Olivier HENCHIRI et al. for J48 are shown in Figure 5.6. Among these authors, our approach uncovers the malware with the best accuracy (97.95%).

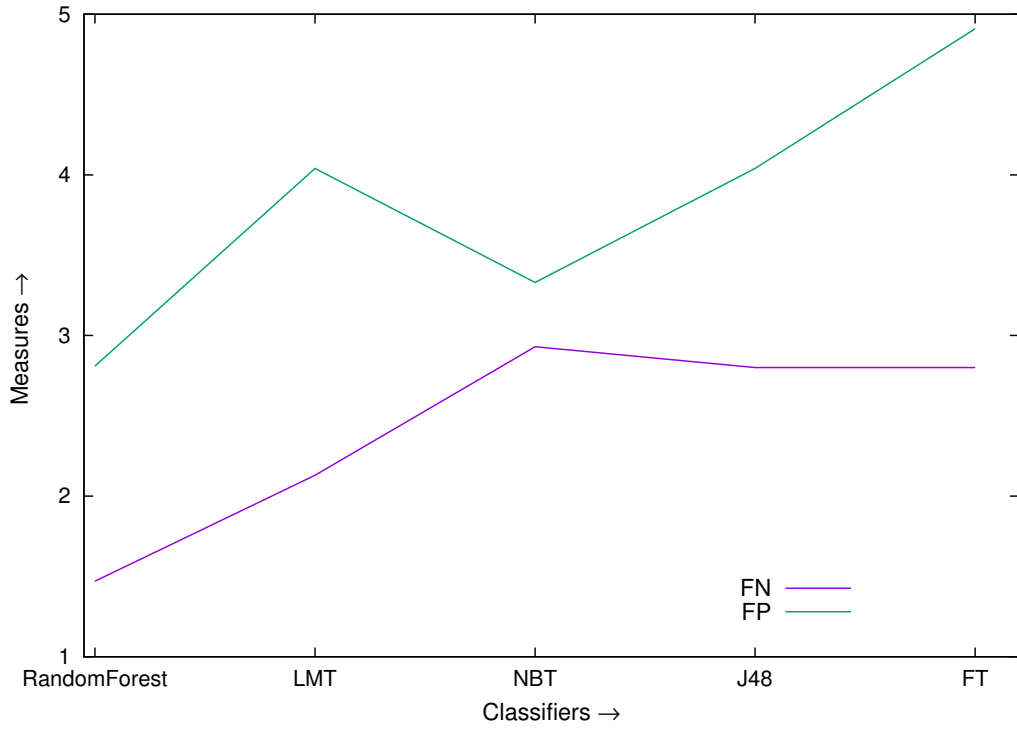


Figure 5.3: *FP and FN of the top five classifiers.*

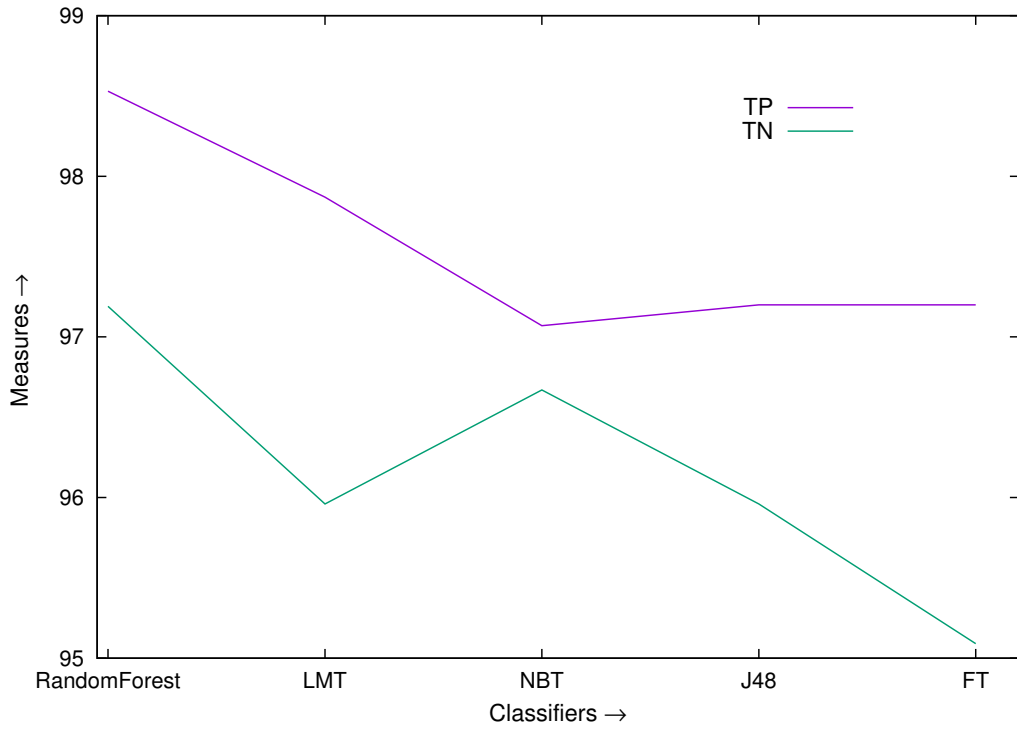


Figure 5.4: *TP and TN of the top five classifiers.*



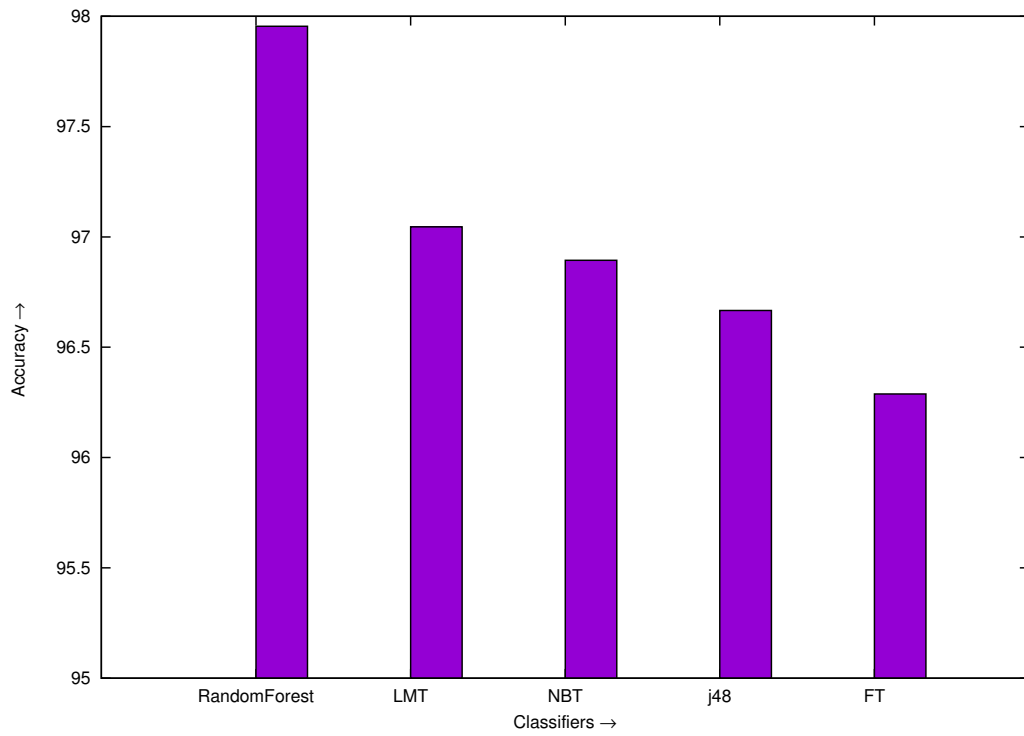


Figure 5.5: Group-wise classification accuracy of the top five classifiers.

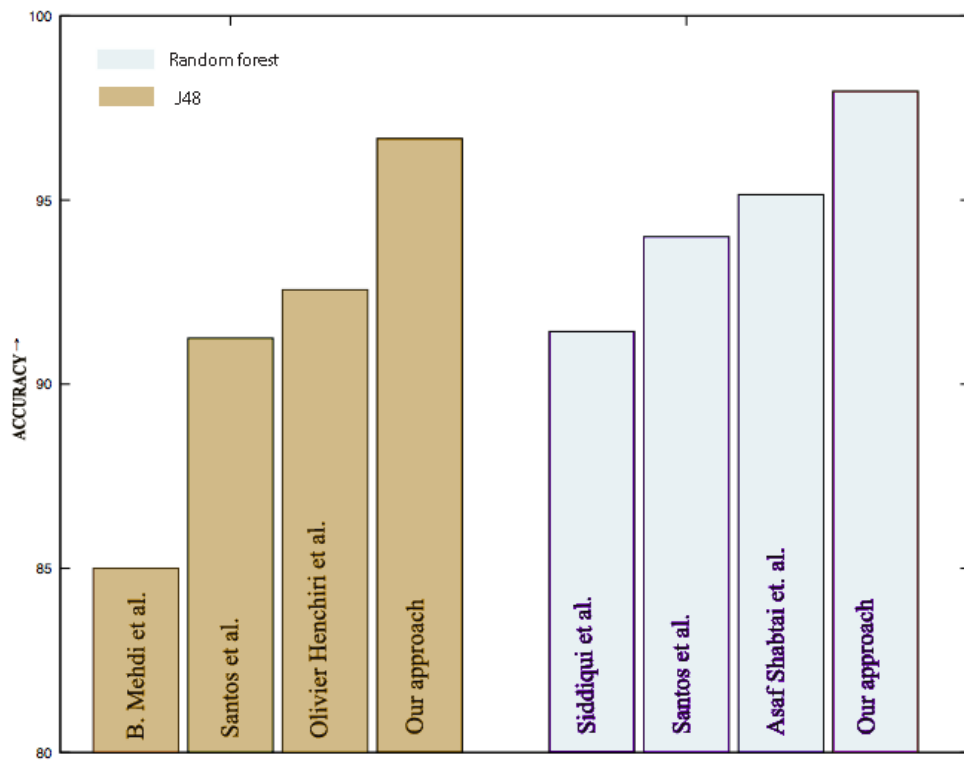


Figure 5.6: Comparison of the accuracy obtained by our approach and others.

## 5.4 Improving the Detection Accuracy by Group-wise Classification using Optimal K-mean Clustering Algorithm

To improve the detection accuracy, further we group-wise partitioned the malware dataset on the basis of their sizes into nine groups by the Optimal k-Means clustering algorithm, and the benign programs are accordingly grouped with the cluster sizes. The number of clusters (value of  $K$ ) is obtained by the Bayesian information criterion [25]. Then each group created has been divided into two sets, one set is used for training of the classifiers, and the other set is used for finding the detection accuracy. The details of the dataset i.e. the number of malware and benign executables for

Cluster	No. of malware for training	No. of benign for training	No. of malware for testing	No. of benign for testing
1	322	43	55	18
2	1234	20	221	9
3	1489	20	265	9
4	714	71	128	13
5	335	2227	61	402
6	886	36	158	11
7	2716	40	481	11
8	1148	33	204	11
9	18	156	4	21
total	8862	2646	1577	505

Table 5.2: Number of malware and benign executables for training and testing the classifier.

training and testing the classifiers are given in Table 5.2, and the Figure 5.7 represents the procedure to group-wise partitioning the dataset, finding the promising features from each formed group and the classification of unknown malware by RF, J48, LMT, FT and NBT classifiers.

In the experimental analysis, we ensure that at least 15% of the executables in the cluster which is not used for training purpose are taken for the testing of the classifiers. For training and testing of the five classifiers, we selected the promising features from each group as described in Algorithm 4.1. To measure the effectiveness of the classifiers with the number of features we took 20, 40, 60, 80 and 100 number of best features for the classification, and obtained the detection accuracy for each groups

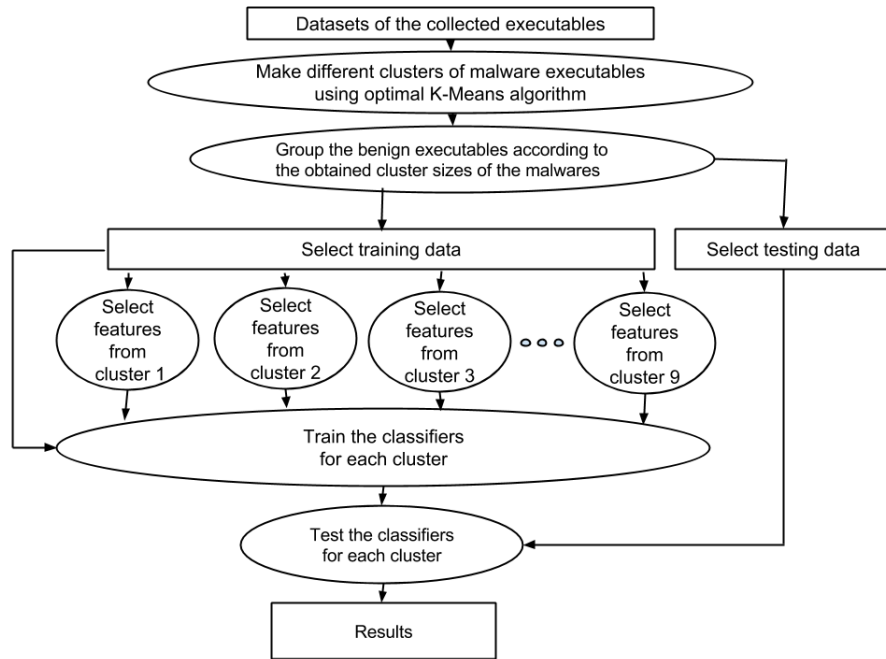


Figure 5.7: Flow chart for the group-wise classification using optimal K-mean clustering algorithm.

using the respective test data. We found that all the five classifiers detection accuracy is more than 98%, in which NBT give the highest accuracy of 99.11% (Figure 5.8). The Figure 5.9 - 5.17 shows the performance of classifiers for each group with respect to different numbers of features.

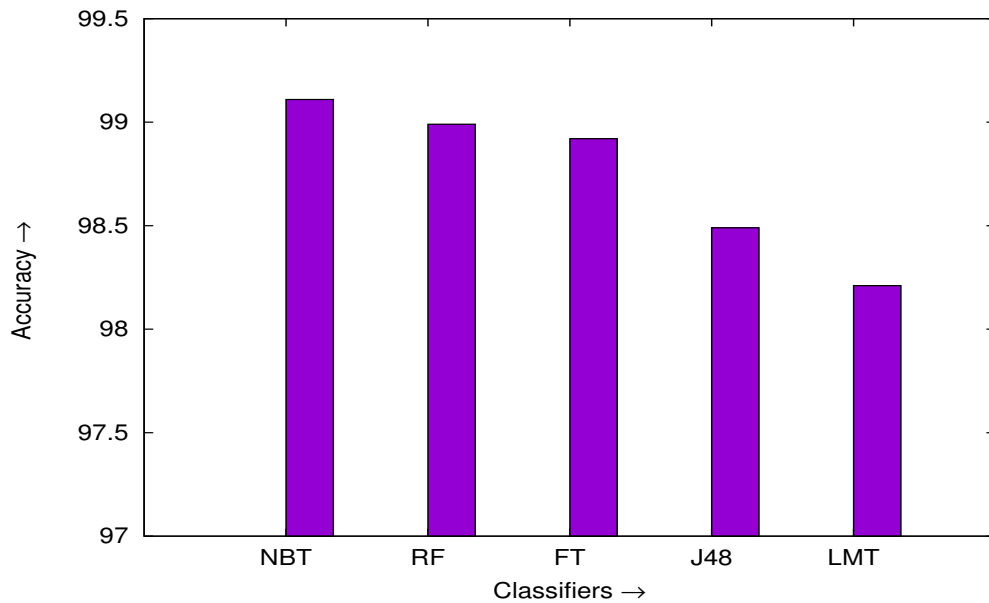


Figure 5.8: Best accuracy of the selected five classifiers.

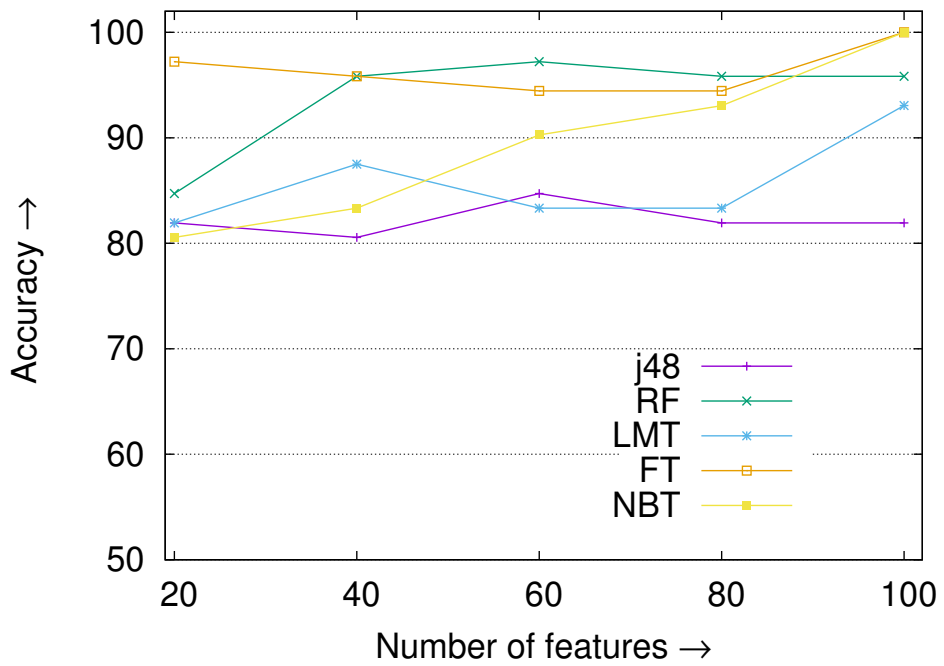


Figure 5.9: Detection accuracy obtained by the classifiers from group-1 data.

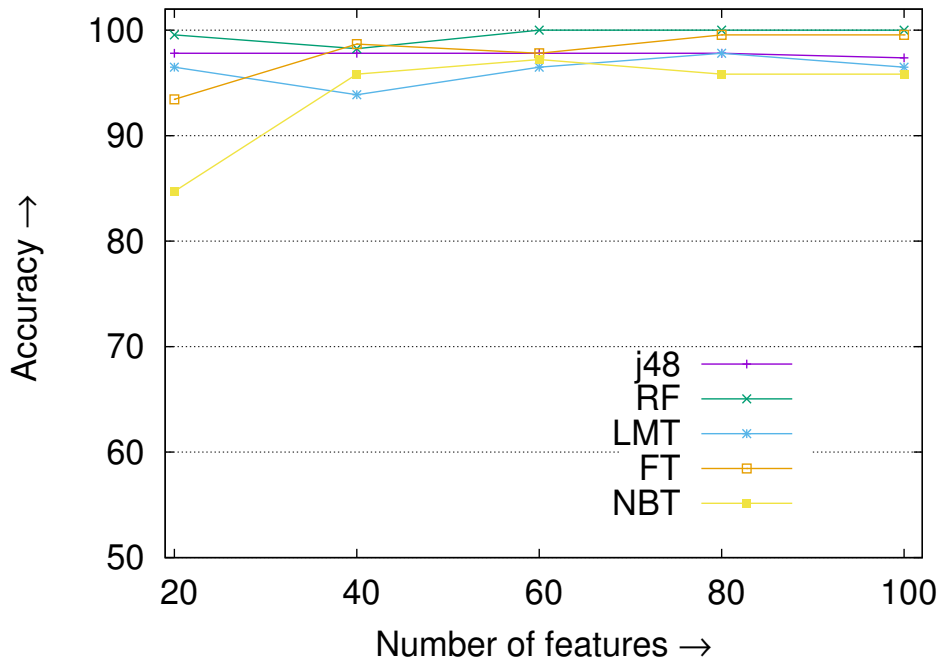


Figure 5.10: Detection accuracy obtained by the classifiers from group-2 data.

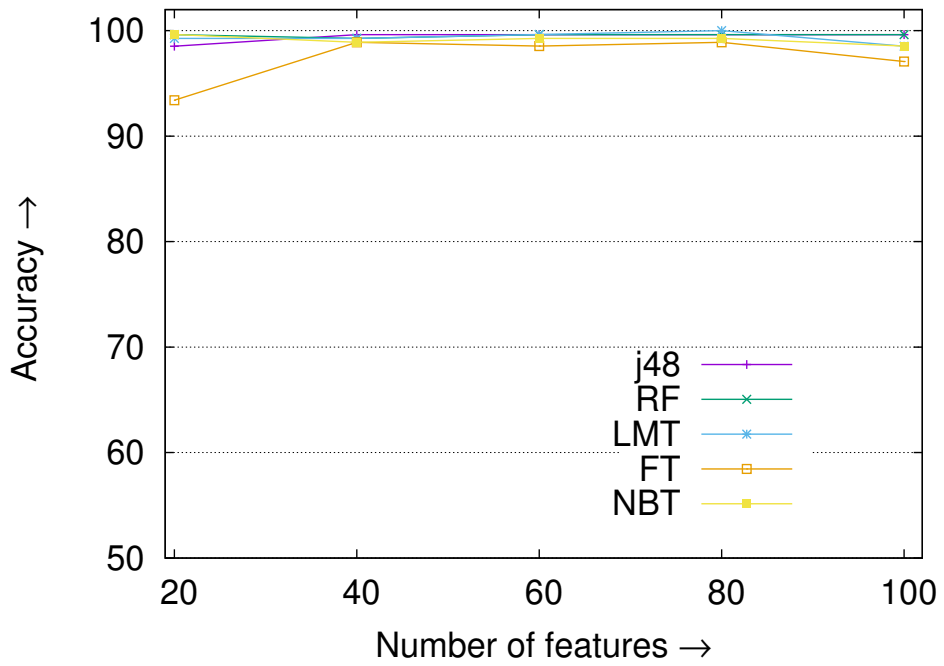


Figure 5.11: Detection accuracy obtained by the classifiers from group-3 data.

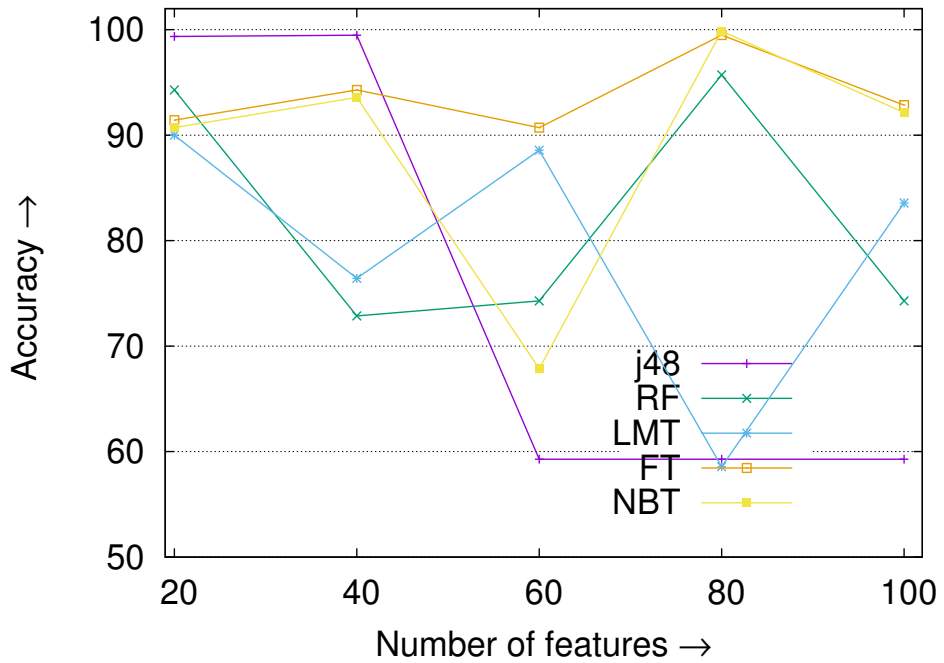


Figure 5.12: Detection accuracy obtained by the classifiers from group-4 data.

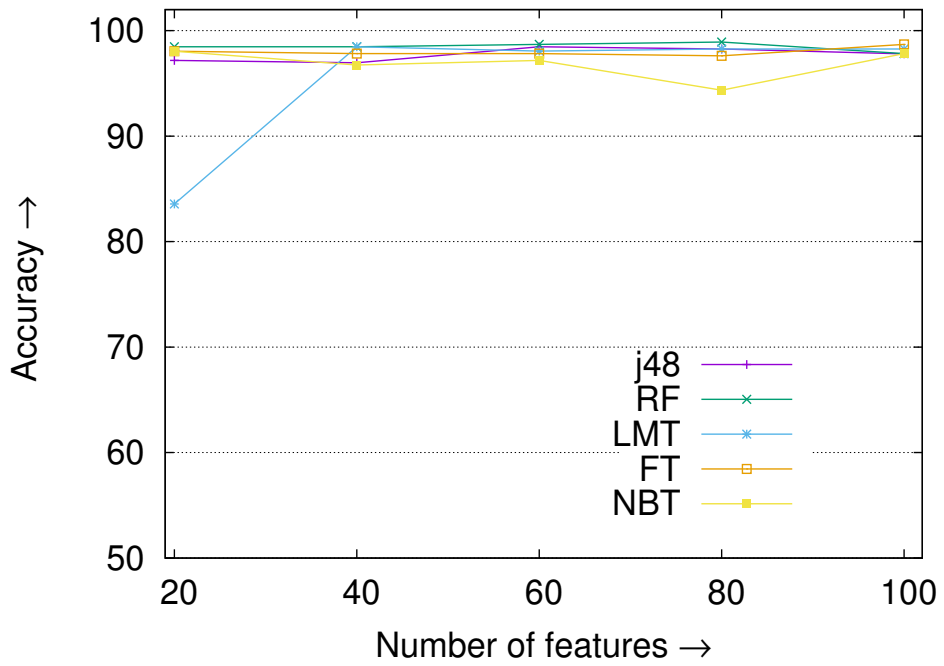


Figure 5.13: Detection accuracy obtained by the classifiers from group-5 data.

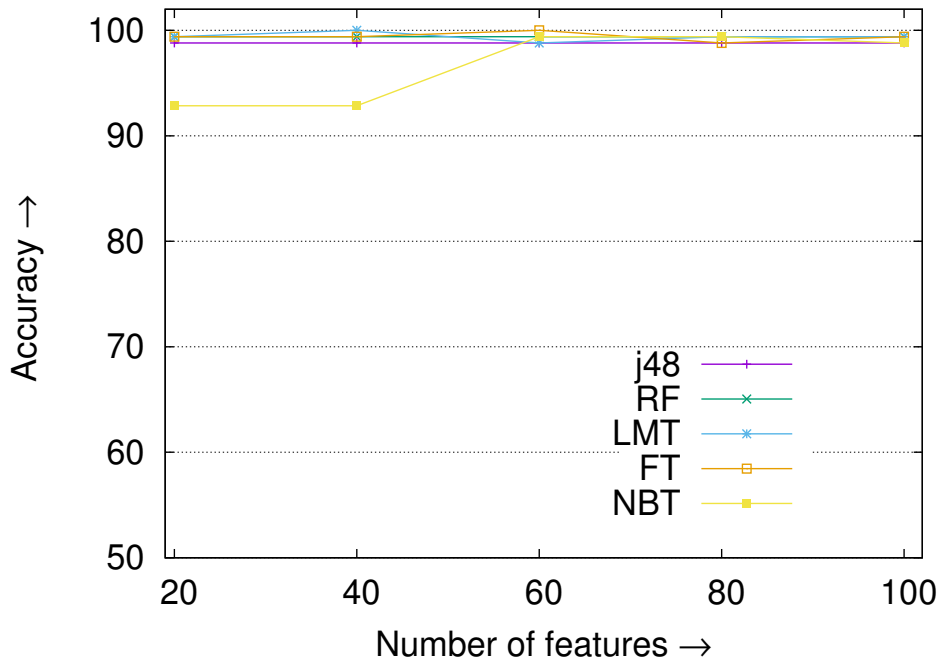


Figure 5.14: Detection accuracy obtained by the classifiers from group-6 data.

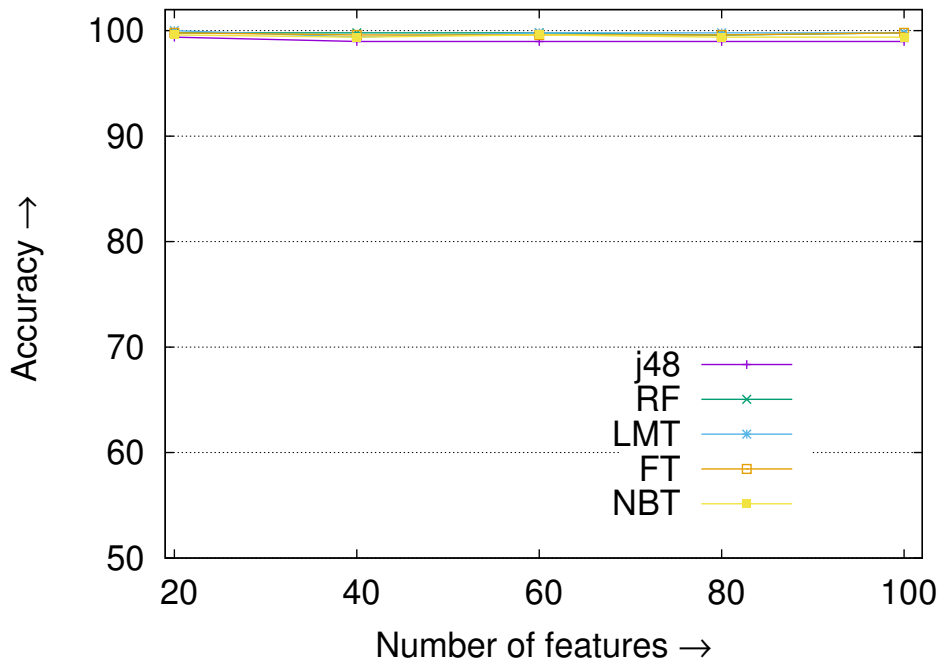


Figure 5.15: Detection accuracy obtained by the classifiers from group-7 data.

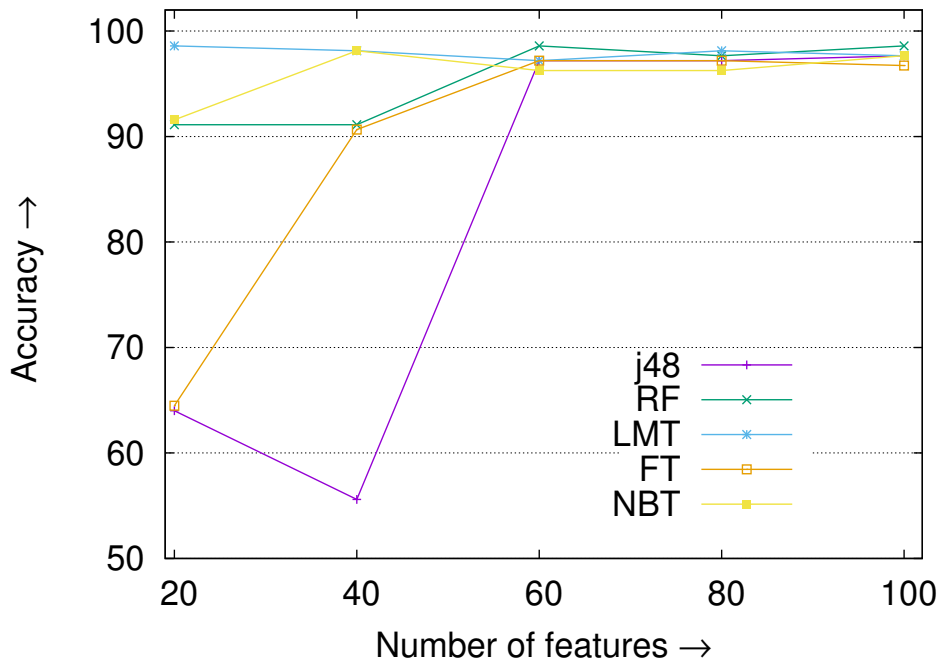


Figure 5.16: Detection accuracy obtained by the classifiers from group-8 data.

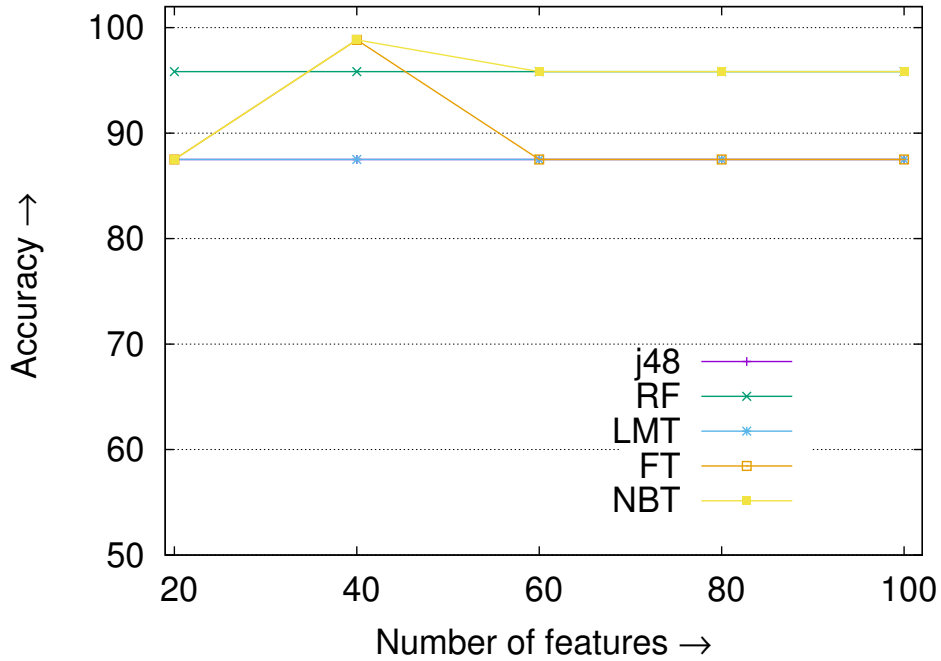


Figure 5.17: Detection accuracy obtained by the classifiers from group-9 data.

## 5.5 Summary

Extensive experimental analysis has been done to study the performance of the classifiers viz. RF, LMT, NBT, J48, and FT in terms of TPR, TNR, FPR, FNR and detection accuracy by analyzing benchmark *Malicia* project dataset and collected benign programs from different Windows desktops. By our approach, all five classifiers are able to uncover unknown malware with greater than 96.28% accuracy, which is better than the maximum detection accuracy (95.9%) reported by Santos et. al. (2013). Among these classifiers, we found that RF is the best (97.95%) classifier to detect the unknown malware. Further to improve the detection accuracy we group-wise partitioned the executables in nine groups using Optimal k-Means clustering algorithm and accordingly obtained the features from each group to test the classifiers (RF, J48, LMT, FT, and NBT) for the classification of malware. We found that all classifiers detection accuracy is more than 98%, in which NBT give the highest accuracy of 99.11%.



## CHAPTER 6

# GROUP-WISE CLASSIFICATION FOR THE DETECTION OF ANDROID MALICIOUS APPS

### 6.1 Introduction

The attractive features and mobility of smart devices have drastically changed the today's environment. Many functionalities of these devices are similar to the traditional information technology system, and can also access the enterprise's applications and data, enabling employees to do their work remotely. Also, due to the ease of use, these devices hold sensitive information such as personal data, browsing history, shopping history, financial details, etc. [6] i.e. users are ever more frequent to use the Internet, as a consequence, these devices are a bullseye for the cyber attacks. The security risks of these devices are not only limited to Bring Your Own Smart Device (BYOSD) scenarios but also for the devices which are adopted on an ad hoc basis. The recent attack shows that the security features in these devices are not as par to completely stop the adversary [99]. Hence smart devices are becoming an attractive target for online criminals, and they are investing more and more for the sophisticated attacks viz. ransomware or to steal the valuable personal data from the user devices.

In the fast-growing smart devices, Android is the most popular OS, and the popularity of these devices which are connected through the Internet accessing billion of online websites encourages malware developer to penetrate the market with malicious apps to annoy and disrupt the victim. Therefore, any security gap in these devices means that the information stored or accessing smart devices are at high risk of being breached. To combat the threat/attack from the malware generally, the traditional approaches based on the signature matching are used. These signature matching techniques are efficient from a time perspective but are not effective for the variant and nor capable to detect continuously growing zero-day malware attack. Also, to evade the signature-based techniques, malware developer uses several obfuscation techniques. However, to detect the Android malicious apps, time to time, a number of static and dynamic methods has been proposed [15], [70],[76], [36] and is viewed as one of the most important areas to be addressed. However, proposed approaches are not sufficient to detect the advanced malware to limit/prevent the damages, and very few approaches are based on opcode occurrence to classify the malicious apps. Therefore, using *Drebin* benchmark malware dataset, in this chapter first we investigate the top five classifiers viz. FT, RF, J48, LMT and NBT (Chapter 5) using opcodes occurrence as the prominent features for the detection of malicious apps, and then to improve the detection accuracy we group-wise classified the Android apps after grouping the dataset based on permissions.

## 6.2 Data Preprocessing and Feature Selection

For the experimental analysis, we downloaded 5531 *Drebin* [15] benchmark malware dataset and 4235 benign apps (cross verified from virustotal.com [7]) from google play store. These apps can be represented as a vector of 256 opcodes [73], and some of these opcodes can be used as features for the effective and efficient detection of Android malicious apps. As discussed in Chapter 4 feature selection plays a vital role in the data classification, and often datasets are represented by many features, however, few of them may be sufficient to improve the concept quality. Therefore, to find the prominent features which can represent the target concept, opcodes of the collected Android apps are extracted as follows

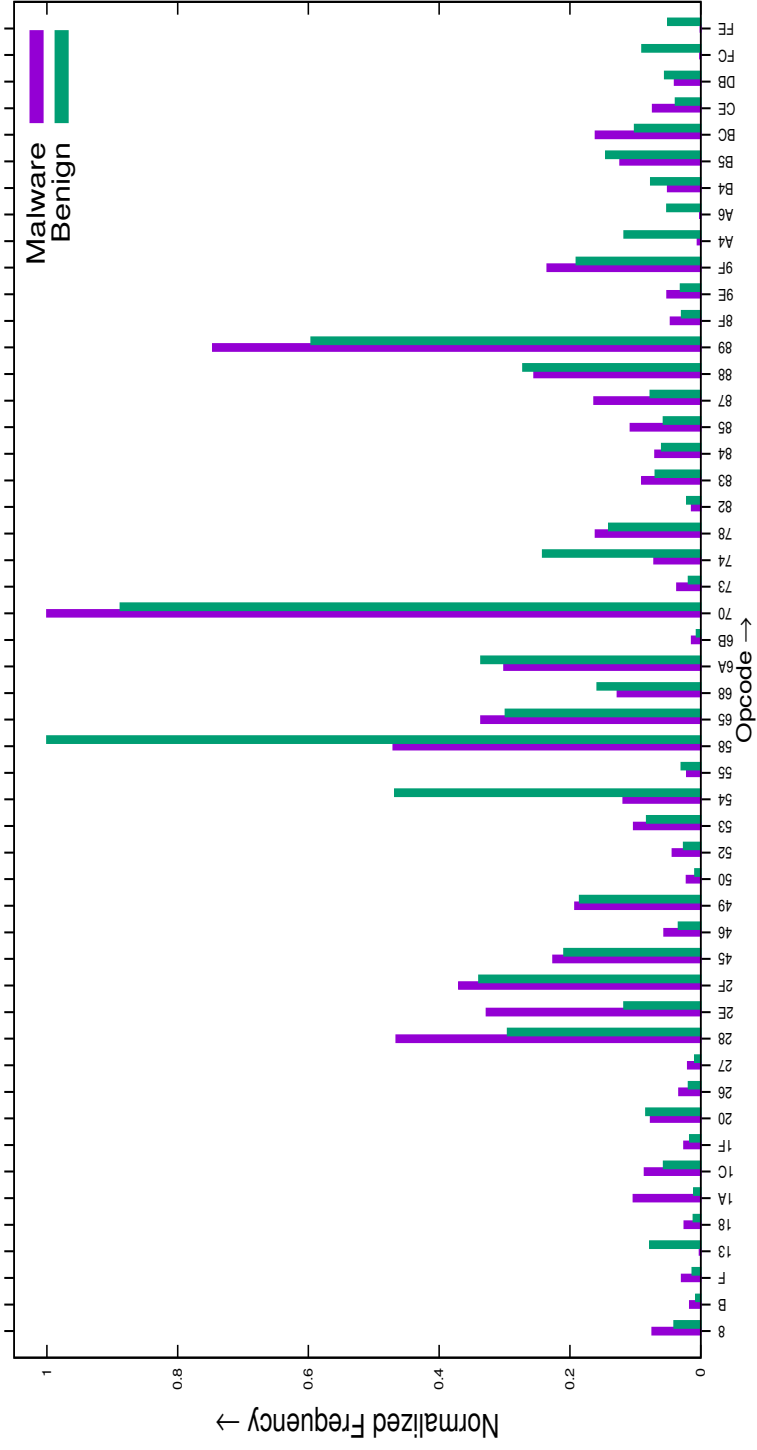


Figure 6.1: Top 50 opcodes occurrence without forming the groups.

1. The *.apk* files (Android apps) has been decompiled by using freely available *apk-tool* [115].
2. From the decompiled data, we kept only the *.smali* files and discarded other data, and then
3. Opcodes are extracted from the *.smali* files.

We analysed the opcode occurrence of all the Android apps and found that the occurrence of many opcodes in malware and benign apps differ in large (e.g. the normalized top 50 opcodes whose occurrence significantly differ in malicious and benign apps is shown in Figure 6.1. The mapping of the opcodes with hexadecimal representation has been kept same as given by the Android developers (Appendix F [73])). The prominent opcodes (features), which suppose to distinguish the malicious and benign Android apps are obtained as described in the Algorithm 6.1. For the

---

**Algorithm 6.1 :** Feature Selection

---

**INPUT:** Pre-processed data

$\mathbf{N}_B$ : Number of benign Android apps,  $\mathbf{N}_M$ : Number of Android malicious apps,

$\mathbf{n}$ : Total number of prominent features required.

**OUTPUT:** List of prominent features

**BEGIN**

**for all** benign apps **do**

Compute sum of the frequencies  $\mathbf{f}_i$  of each opcode  $\mathbf{Op}$  and normalize it.

$$F_B(Op_j) = (\sum f_i(Op_j))/N_B$$

**end for**

**for all** malware data **do**

Compute sum of the frequencies  $\mathbf{f}_i$  of each opcode  $\mathbf{Op}$  and normalize it.

$$F_M(Op_j) = (\sum f_i(Op_j))/N_M$$

**end for**

**for all** opcode  $\mathbf{Op}_j$  **do**

Find the difference of the normalized frequencies for each opcode  $\mathbf{D}(\mathbf{Op}_j)$ .

$$D(Op_j) = |F_B(Op_j) - F_M(Op_j)|$$

**end for**

**return**  $\mathbf{n}$  number of prominent opcodes as features with high  $\mathbf{D}(\mathbf{Op})$ .

---

classification, we used the same Waikato Environment for Knowledge Analysis (WEKA [47]), and on the basis of investigation results of previous Chapters, we selected the best classifier (RF , LMT , NBT, J48, and FT) for analysis, first without grouping the apps, and then to improve the detection accuracy we classified the apps after grouping the collected datasets based on permissions.

### 6.3 Classification Without Grouping the Apps

A novel approach to classify the Android malicious apps is shown in Figure 6.2, which involves finding the promising features (Algorithm 6.1), and the detection processes. The five selected classifiers are analysed without grouping the apps by

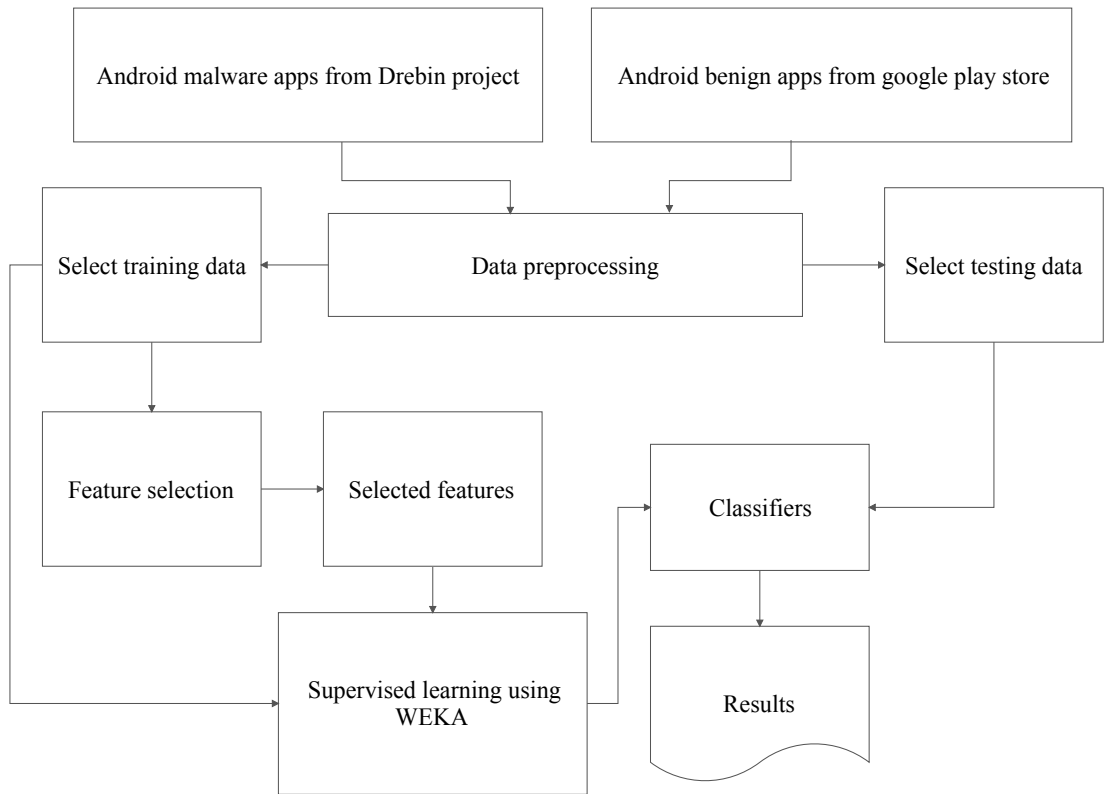


Figure 6.2: Flow chart for the detection of Android malicious apps without grouping the data.

applying supervised machine learning technique. For the analysis, we first obtained the top 200 promising features. Then the performance of the classifier (Equation 4.4) has been studied by taking 20% of available data (not used for training) with 20 - 200 best features, incrementing 20 features at each step and the results obtained are shown

in Figure 6.3. From the analysis, we find that the best accuracy is obtained by FT, Random forest, J48, LMT, and NBT is approximately 79.27, 74.95, 71.73, 70.51 and 68.87 (Figure 6.4) respectively. Among these classifiers with the variation in features the

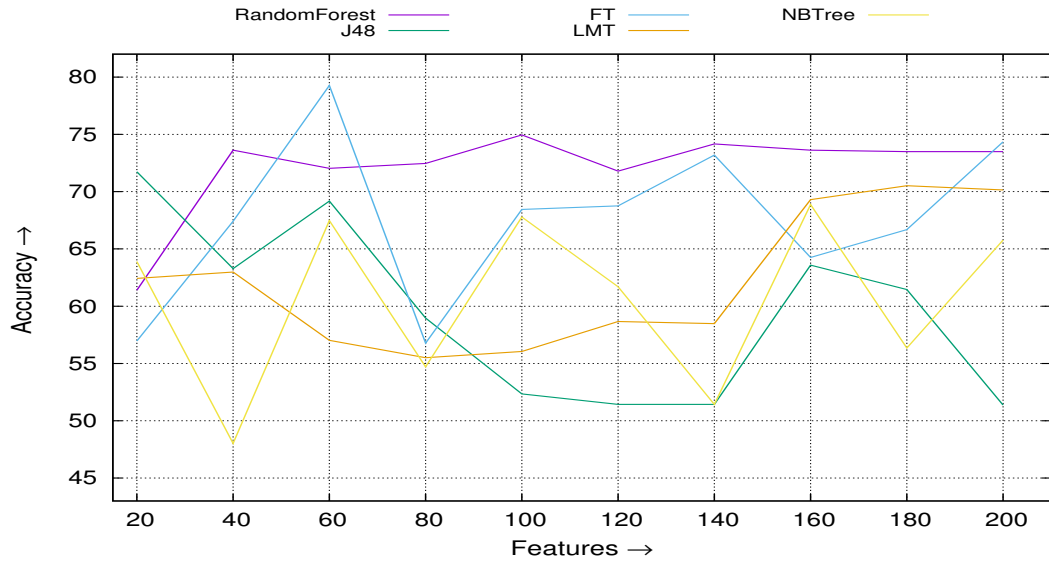


Figure 6.3: Detection accuracy of the selected five classifiers with number of prominent features.

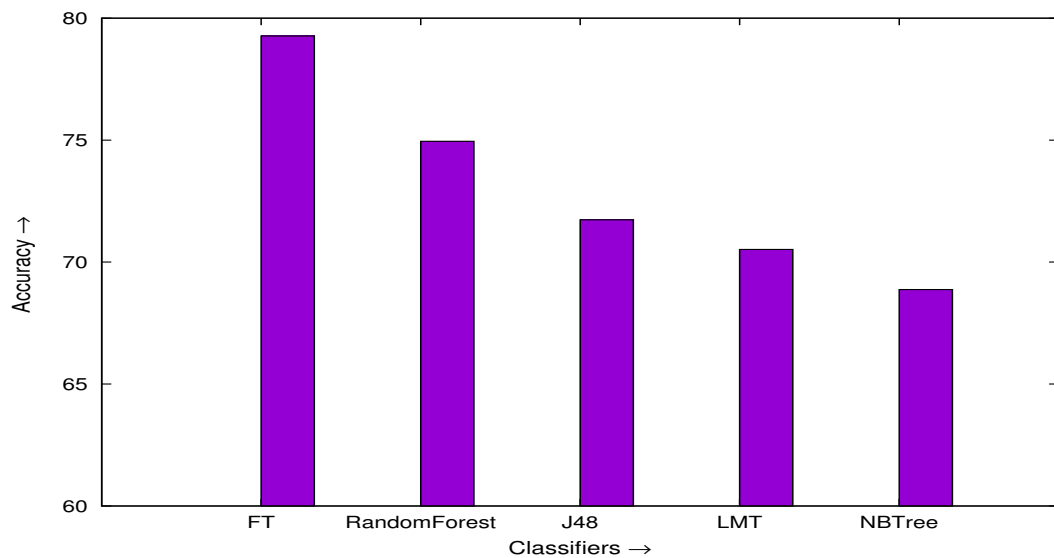


Figure 6.4: Best accuracy of the selected five classifiers.

least fluctuation is observed in Random forest (Figure 6.3). Figure 6.5 shows the TPR (malware detection rate) of all five classifiers with the number of prominent features. We found that compared to other classifiers, the RF gives maximum TPR with least

fluctuation.

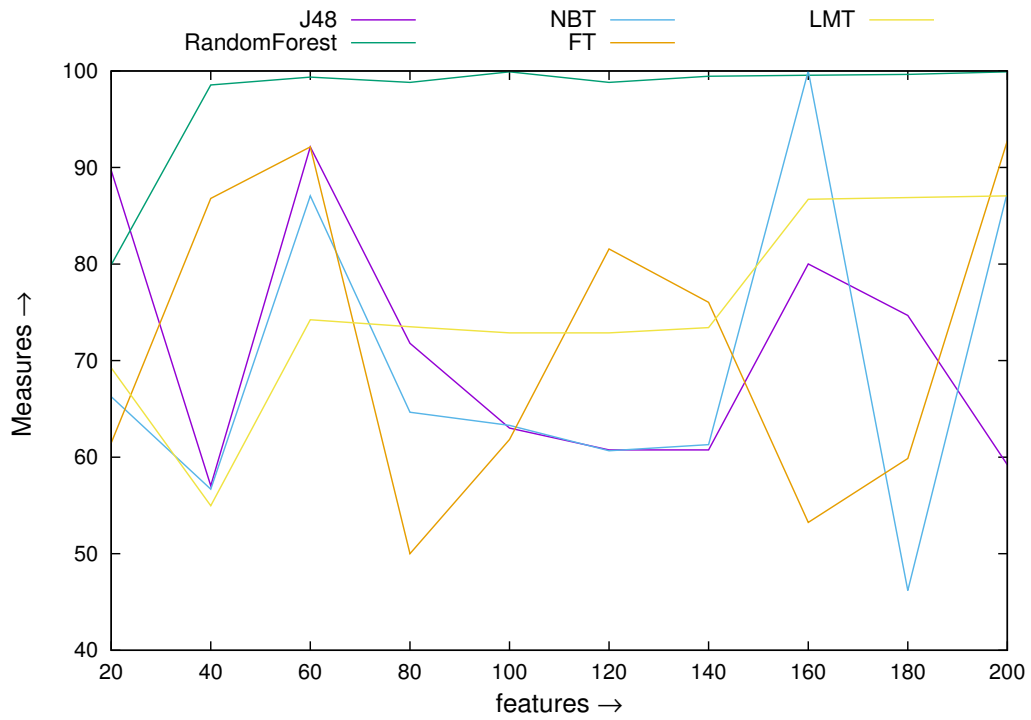


Figure 6.5: True positive rate of the selected five classifiers with number of prominent features.

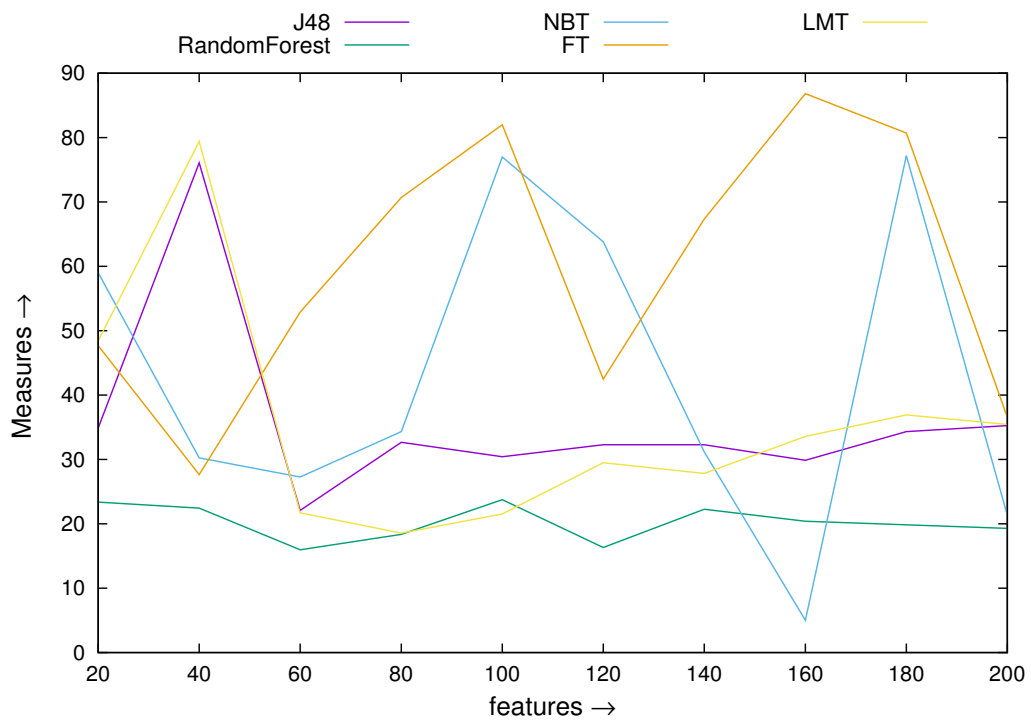


Figure 6.6: True negative rate of the selected five classifiers with number of prominent features.

Figure 6.6 shows the TNR (benign detection rate) for all five classifiers with number of prominent features. Here with some exception, we observed that FT with the number of prominent features detected the benign better than the other classifiers.

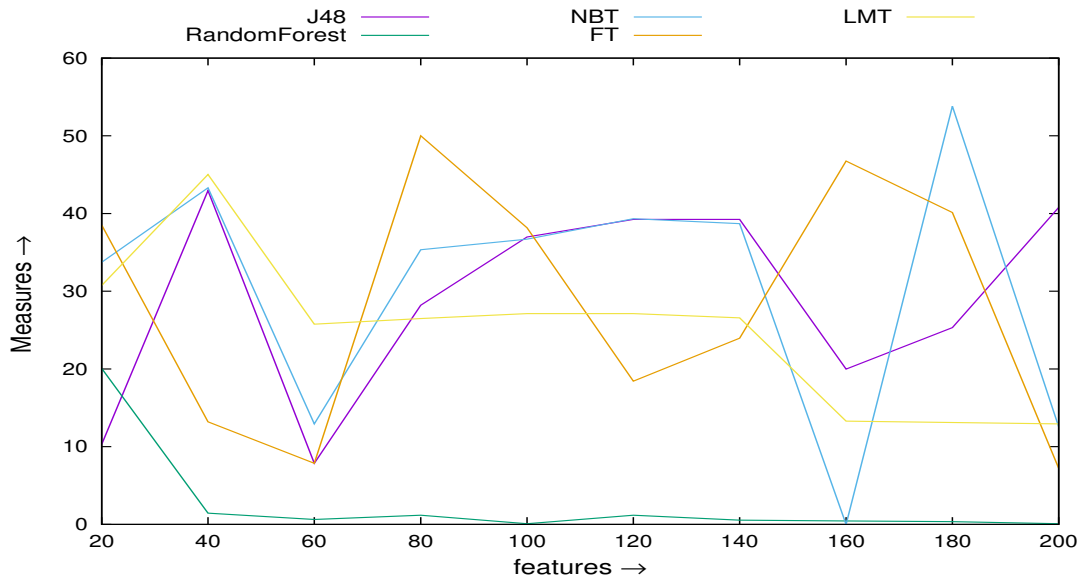


Figure 6.7: False negative rate of the selected five classifiers with number of prominent features.

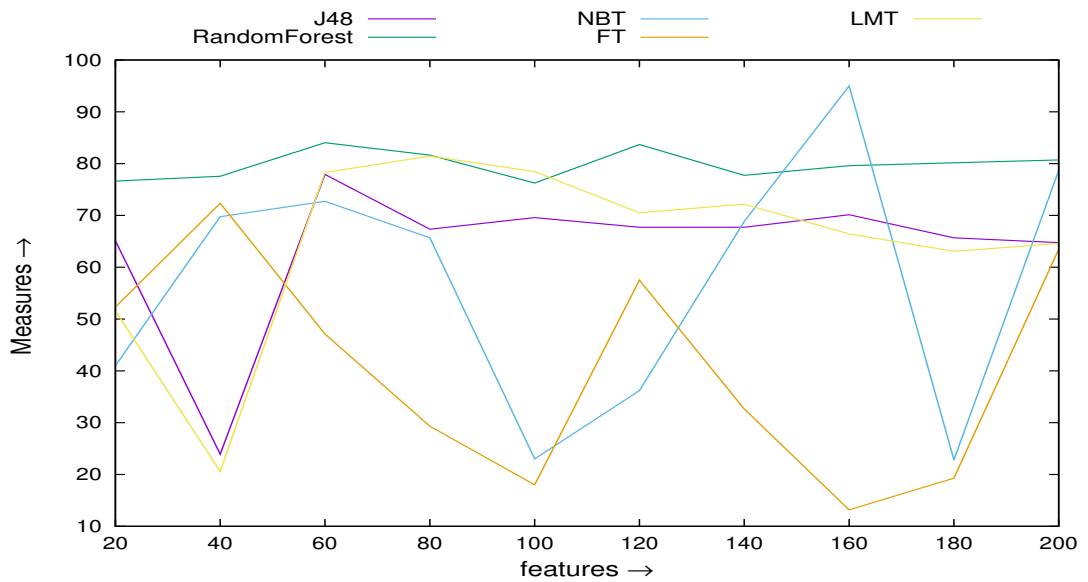


Figure 6.8: False positive rate of the selected five classifiers with number of prominent features.

Figure 6.7 shows the false negative rate of all the selected classifier, we find that, compared to other classifiers the RF is good, and also fluctuates least with the number of features. Figure 6.8 shows the false positive rate of the analysed classifiers, and here



we observed that all the five classifier does not gives a good result, hence very much affects the final accuracy. However, although the false negative rate of RF is not as par, but the fluctuation with the number of features is least compared to other classifiers.

## 6.4 Grouping of Android Apps

Android is a privilege-separated operating system in which each application runs as a separate process with unique user/group ID, and operates in an application sandbox so that apps execution can be kept in isolation from other apps and the system. Hence, to access the user data or resources from the system, apps need additional capabilities that are not provided by the basic sandbox. To access data or resources outside the sandbox, the apps have to explicitly request the needed permission. Depending on how sensitive the area/data is, the requested permission may be granted automatically by the system or ask the user to approve or reject the request. In Android, these permissions can be found in Manifest.permission file, e.g. an app that needs to monitor incoming SMS messages would specify:

```
< manifestxmlns : Android = "http://schemas.Android.com/apk/res/Android"
package = "com.Android.app.myapplication" >
< uses - permissionAndroid : name = "android.permission.RECEIVE_SMS" / >
...
< /manifest >
```

In total there are 235 permissions out of which 163 are hardware accessible and remaining are for user information access [72]. In terms of security, all these permissions can be put into two categories i.e. normal and dangerous permissions [11]. Therefore it will be important to study the classification of Android malicious apps after grouping them into dangerous (Table 6.1) and normal/other permissions.

In Android, if an application need to access resources or data outside its sandbox, and if there's very little/no risk to the user's privacy or in the operation of other apps, then such permissions are called normal permission. If the requested permission belongs to normal permission group, then the permission is automatically

Permission Group	Permissions
CALENDAR	READ_CALENDAR WRITE_CALENDAR
CAMERA	CAMERA
CONTACTS	READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS
LOCATION	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	READ_PHONE_STATE CALL_PHONE READ_CALL_LOG WRITE_CALL_LOG ADD_VOICEMAIL USE_SIP PROCESS_OUTGOING_CALLS
SENSORS	BODY_SENSORS
SMS	SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_WAP_PUSH RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

Table 6.1: Dangerous permissions groups of the Android apps

granted to the app, e.g. permission to access the time zone is a normal permission, and if an app request permission to access time zone, then it is granted automatically. Whereas, if an application needs to access resources or data outside its sandbox which could potentially affect the user’s privacy/data, or in the operation of other apps, then such permission are called dangerous permission. If the requested permissions belongs to a dangerous permission group, then the user has to approve/reject the permission, e.g. the permission to access the user’s contacts is a dangerous permission, and if an application request permission to access users contacts, then the user has to approve/reject the permission.

To improve the detection accuracy of Android malicious apps we grouped the *Drebin* [15] 5531 benchmark malware dataset and 4235 benign apps available at Google play store. Our analysis shows that the *Drebin* dataset does not contain any apps which need BODY.SENSOR permission, therefore we ignored the SENSOR group

in our experimental analysis, and made total nine groups (eight groups of dangerous permissions and one group of normal/other permissions) for the classification of Android apps.

## 6.5 Group-wise Classification of Android Malicious Apps

Similar to section 6.2 we studied the occurrence of opcodes in both benign and malicious Android apps, separately in each formed group. Then using the Algorithm 6.1 we obtain differences in the opcodes occurrence between benign and malicious apps and the group-wise top 50 opcodes whose occurrence significantly differ are shown in Figures 6.9 - 6.17 for the CALENDAR, CAMERA, CONTACTS, LOCATION, MICROPHONE, OTHERS, PHONE, SMS, and STORAGE group respectively, whereas, difference in the opcodes occurrence without forming the group is shown in Figures 6.1. From the analysis (Figures 6.9 - 6.17) we find that the opcode occurrence between any group differs significantly when compared with the opcode occurrence obtained without forming the groups, and also between any two groups. Hence, the final features are selected after ordering the opcodes by their occurrence difference in each group and used it for the classification of Android apps.

For the classification, the detail distribution (No. of training and testing malicious and benign apps, Total No. of apps in the group used for the classification) of the collected dataset is given in Table 6.2. For the group-wise classification, we used same WEKA tool, and on the basis of previous investigation (Chapter 5), we selected the same classifier (Random Forest, Logistic model trees, Naive-Bayes Tree, J48 and Functional Tree) for the analysis, but prominent features to train and test the classifiers, the data are taken from the considered group only (Table 6.2). To measure the goodness of trained models, we evaluate the detection accuracy as discussed in Section 4.4.2.

The performance of the classifier has been investigated for each group by taking randomly 20% of the collected data (other than the training) with 20 - 200 best features incrementing 20 features at each step, and the results obtained are shown

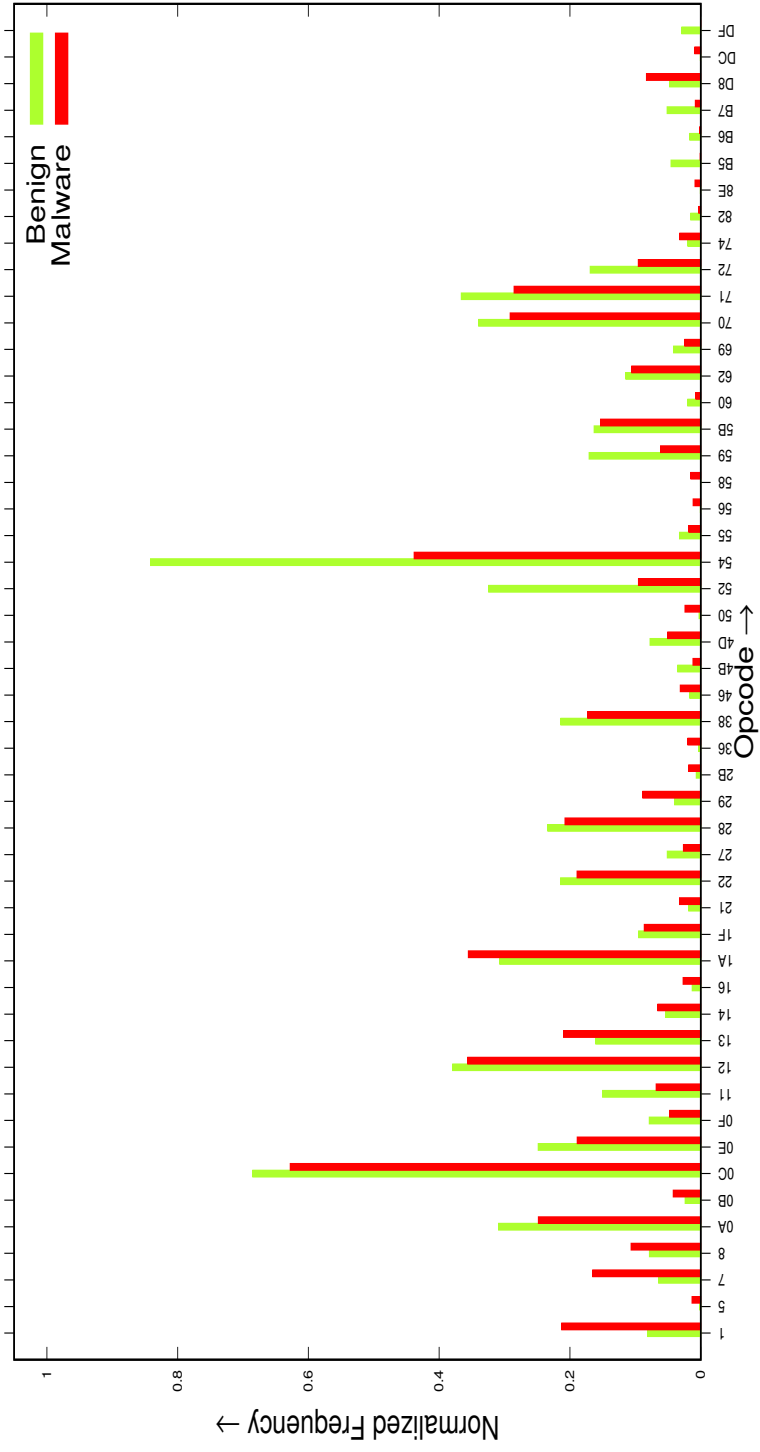


Figure 6.9: Top 50 opcodes occurrence difference between benign and malicious apps in the CALENDAR group.

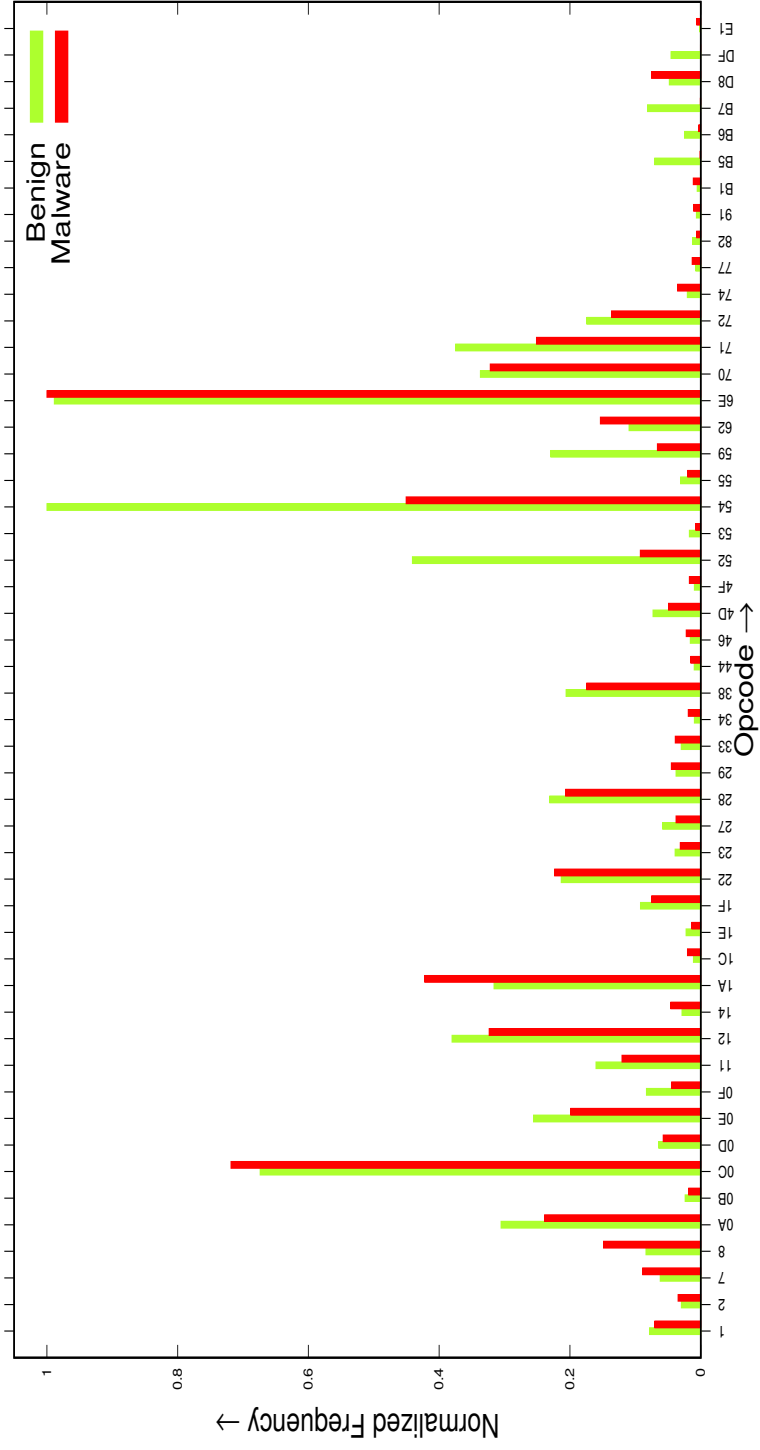


Figure 6.10: Top 50 opcodes occurrence difference between benign and malicious apps in the CAMERA group.

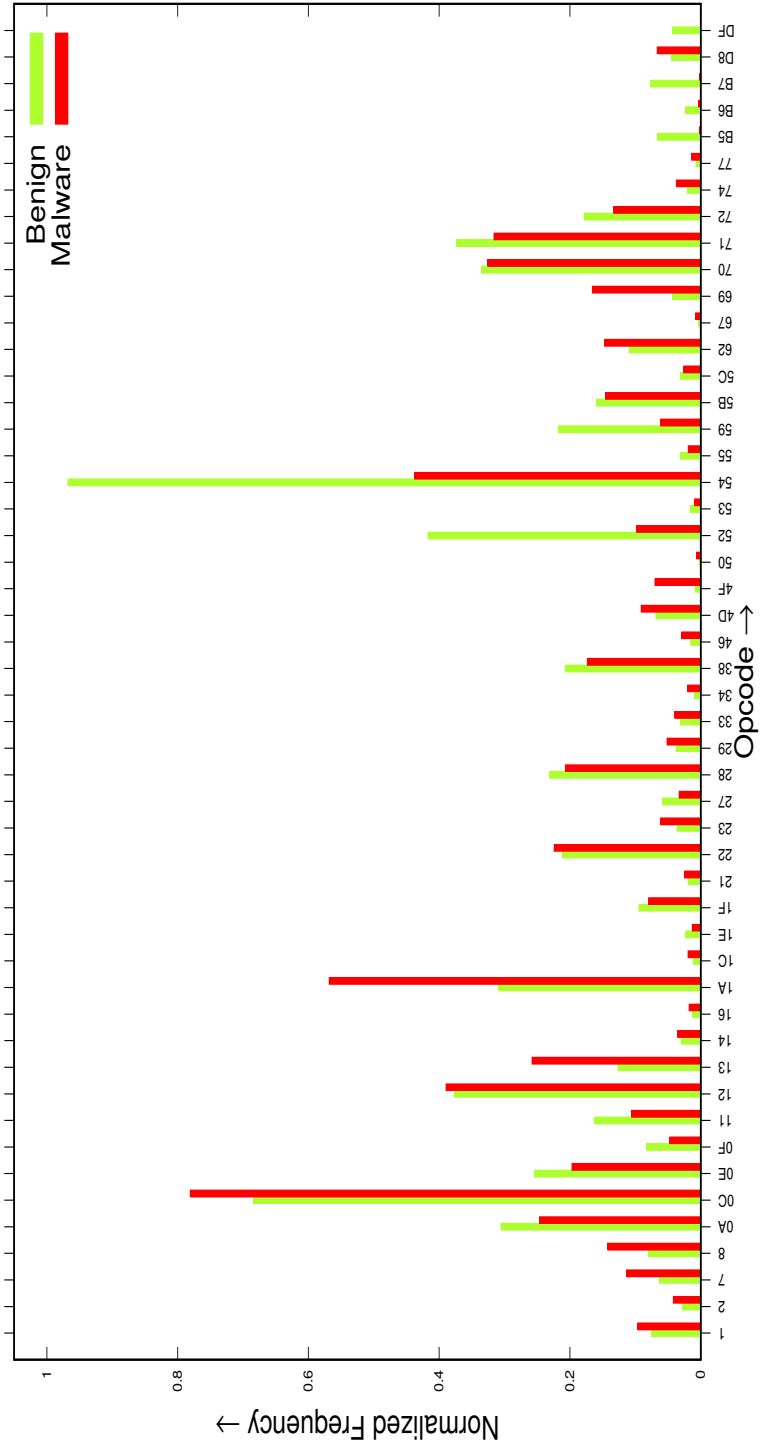


Figure 6.11: Top 50 opcodes occurrence difference between benign and malicious apps in the COTACTS group.

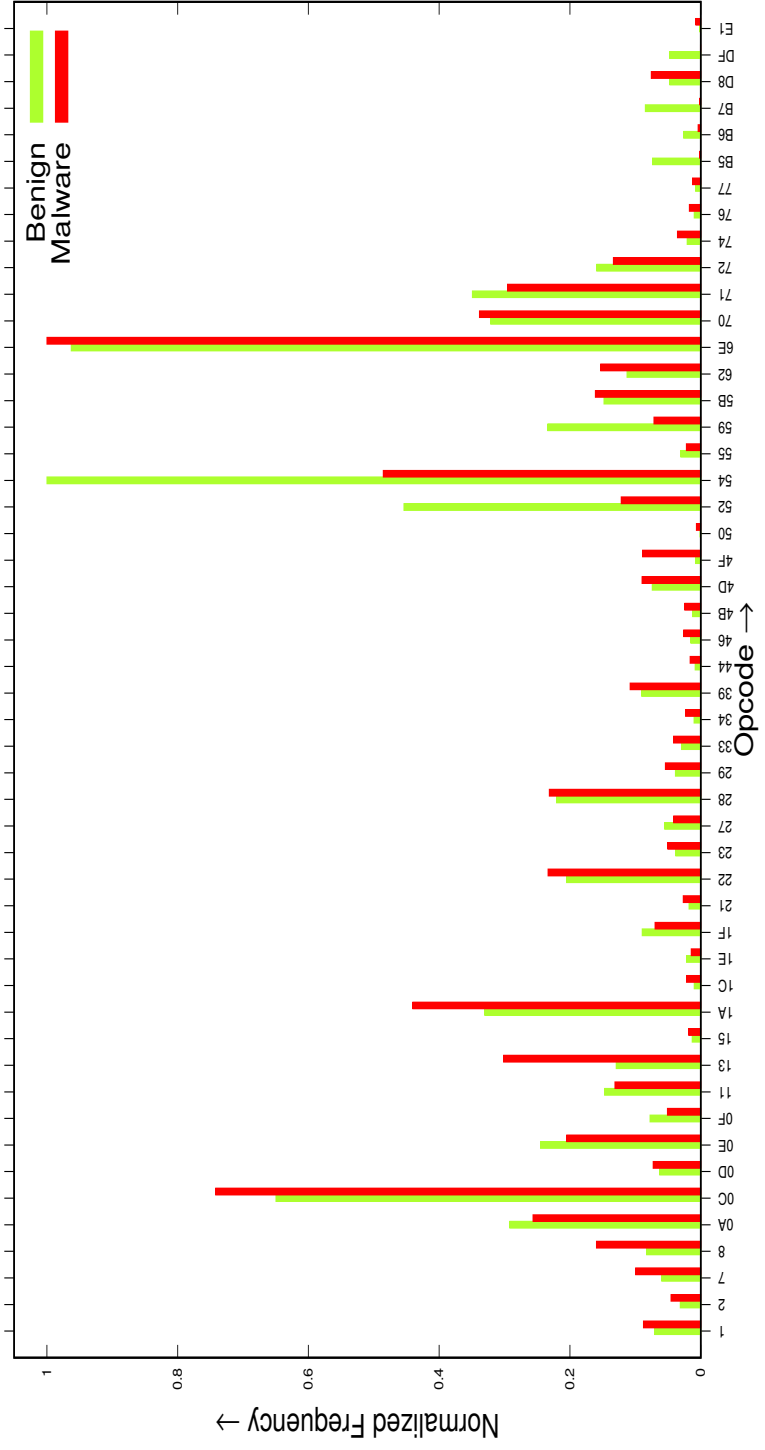


Figure 6.12: Top 50 opcodes occurrence difference between benign and malicious apps in the LOCATION group.

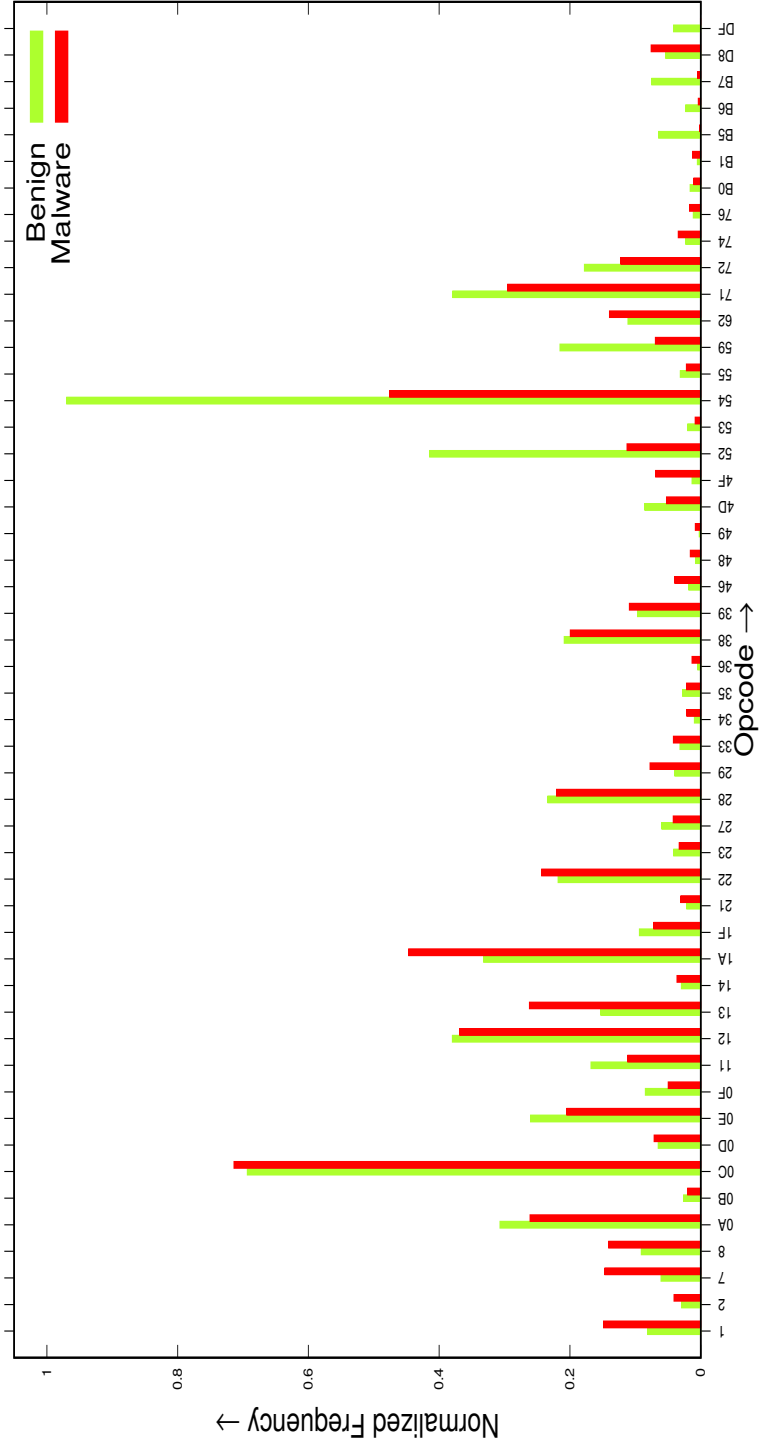


Figure 6.13: Top 50 opcodes occurrence difference between benign and malicious apps in the MICROPHONE group.



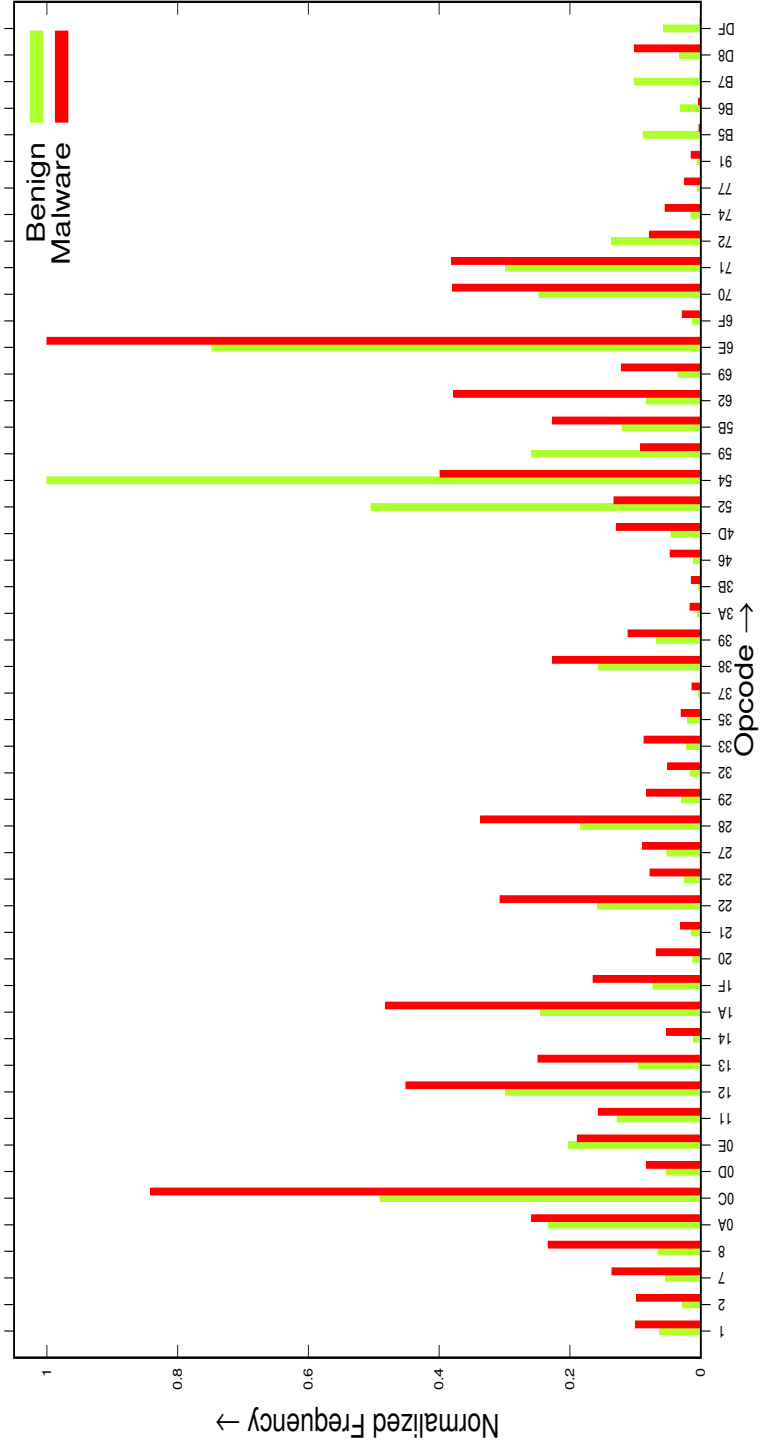


Figure 6.14: Top 50 opcodes occurrence difference between benign and malicious apps in the OTHER group.

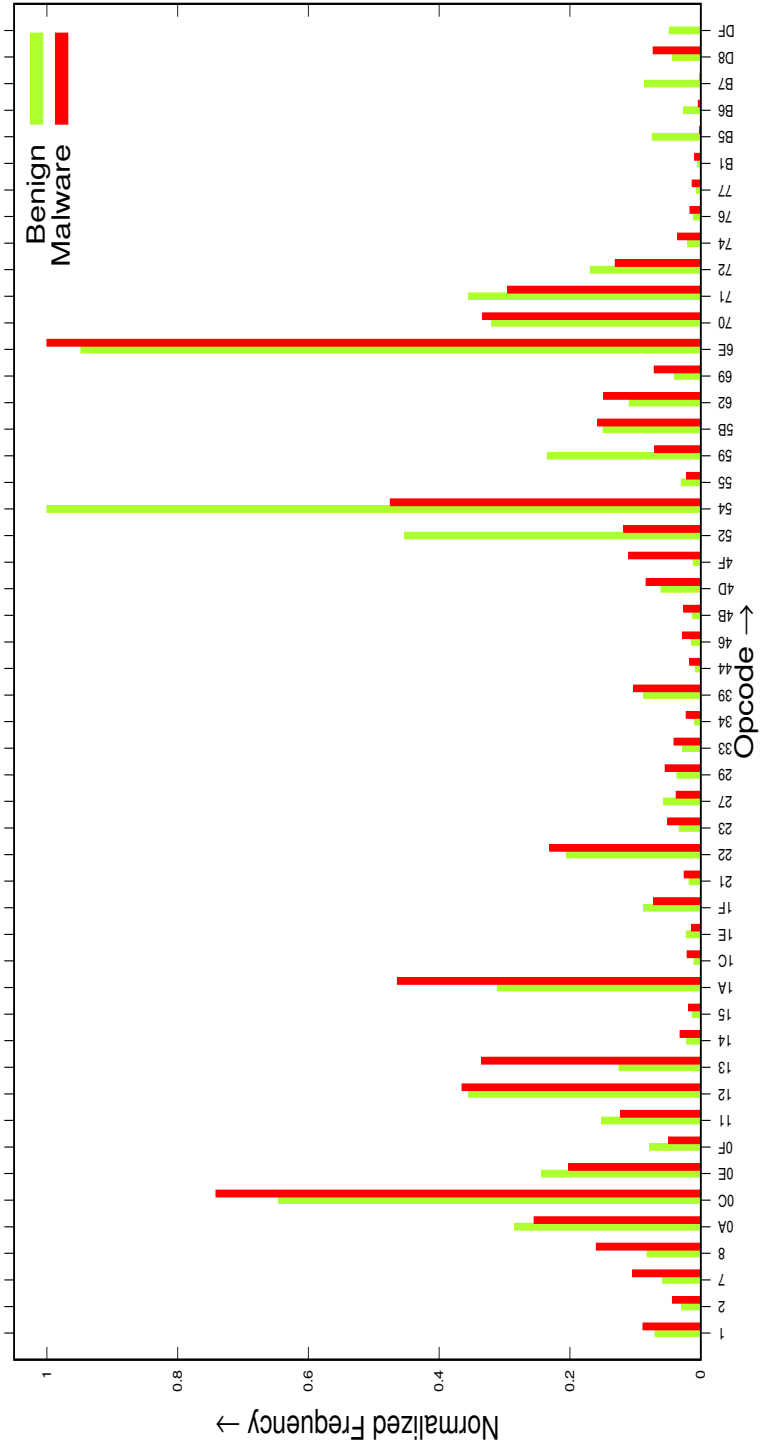


Figure 6.15: Top 50 opcodes occurrence difference between benign and malicious apps in the PHONE group.

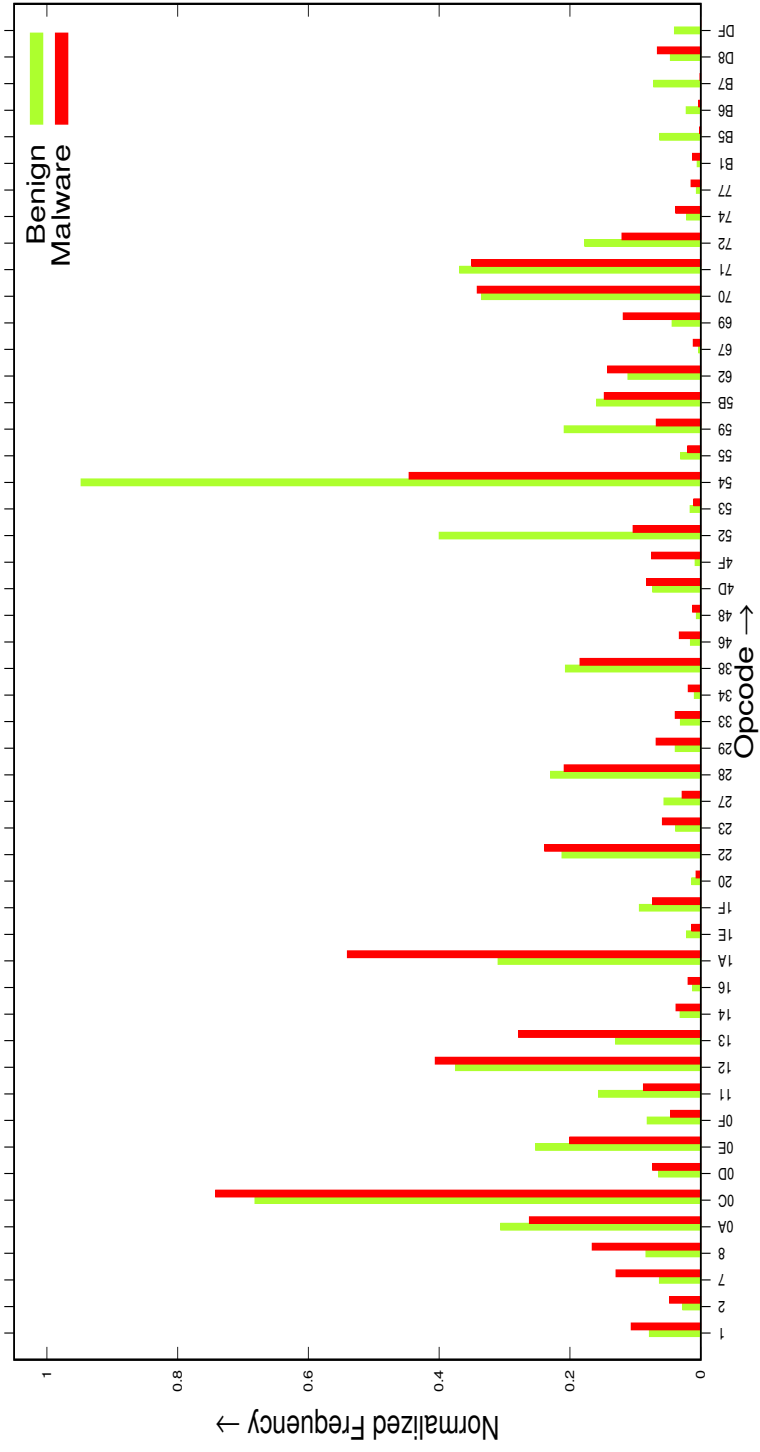


Figure 6.16: Top 50 opcodes occurrence difference between benign and malicious apps in the SMS group.

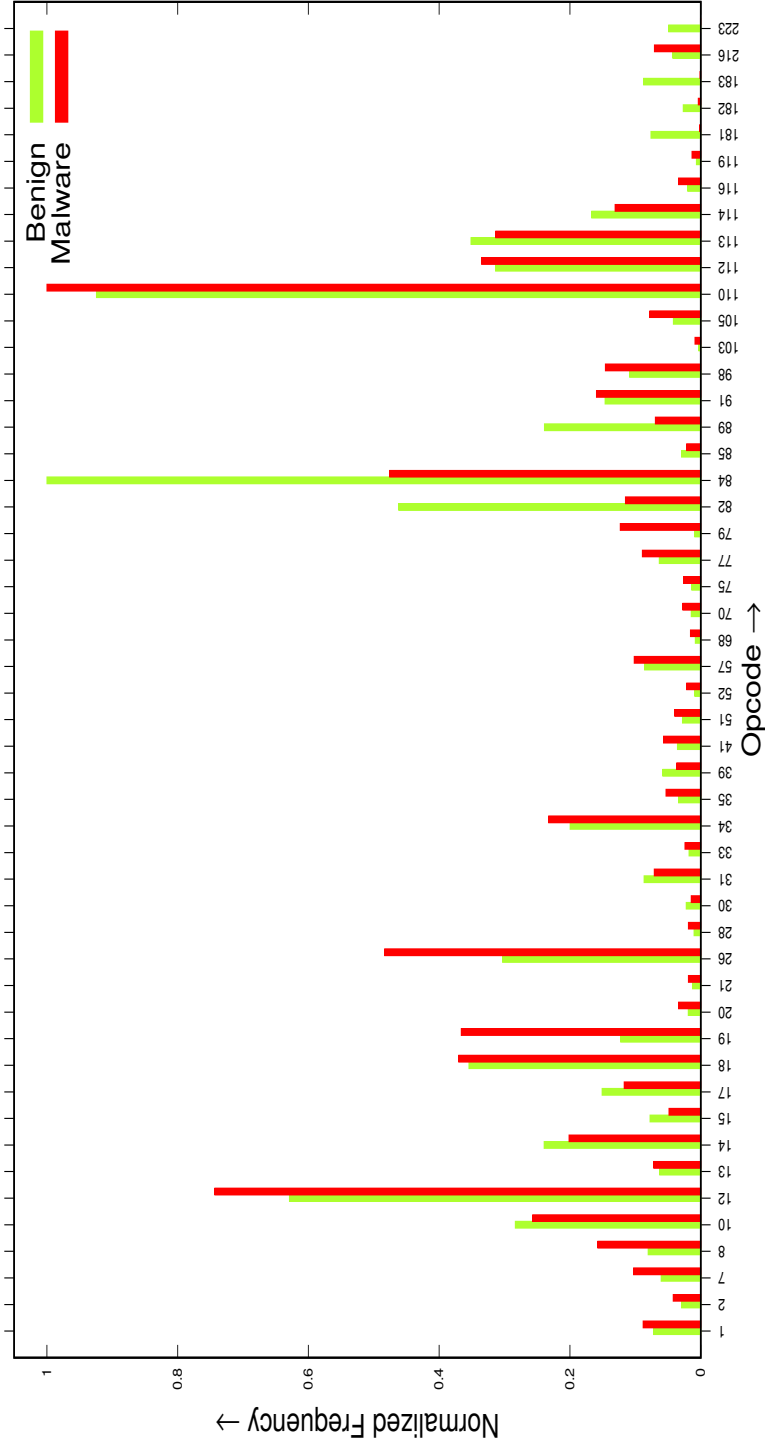


Figure 6.17: Top 50 opcodes occurrence difference between benign and malicious apps in the STORAGE group.

in Figures 6.18 - 6.26 for the CALENDAR, CAMERA, CONTACTS, LOCATION, MICROPHONE, OTHERS, PHONE, SMS and STORAGE group respectively.

Groups	No. of malicious apps for the training	No. of benign apps for the training	No. of malicious apps for the testing	No. of benign apps for the testing	Total No. of apps
CALENDAR	59	57	14	14	144
CAMERA	179	423	44	106	752
CONTACTS	1073	356	268	89	1786
LOCATION	1538	68	383	18	2007
MICROPHONE	95	218	23	55	391
OTHERS	110	891	27	223	1251
PHONE	3981	1453	986	373	6793
SMS	2712	239	677	60	3688
STORAGE	2923	837	730	210	4700

Table 6.2: Number of benign and Android malicious apps used for training and testing the classifiers.

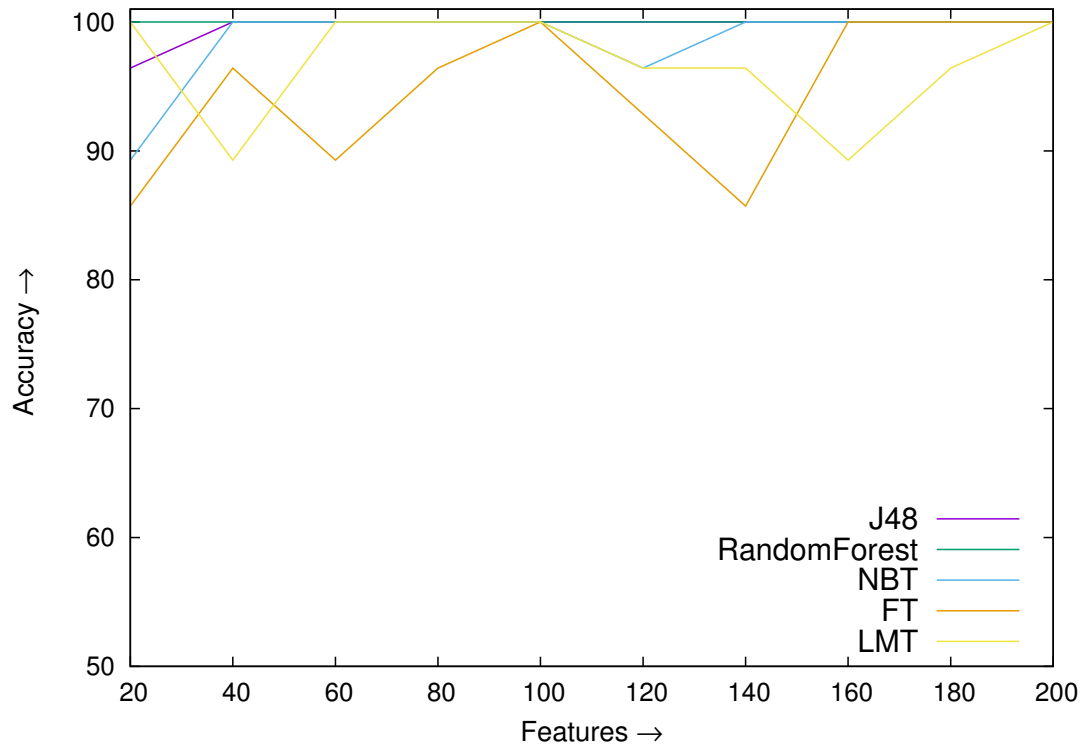


Figure 6.18: Detection accuracy of the classifiers for the CALENDAR group.

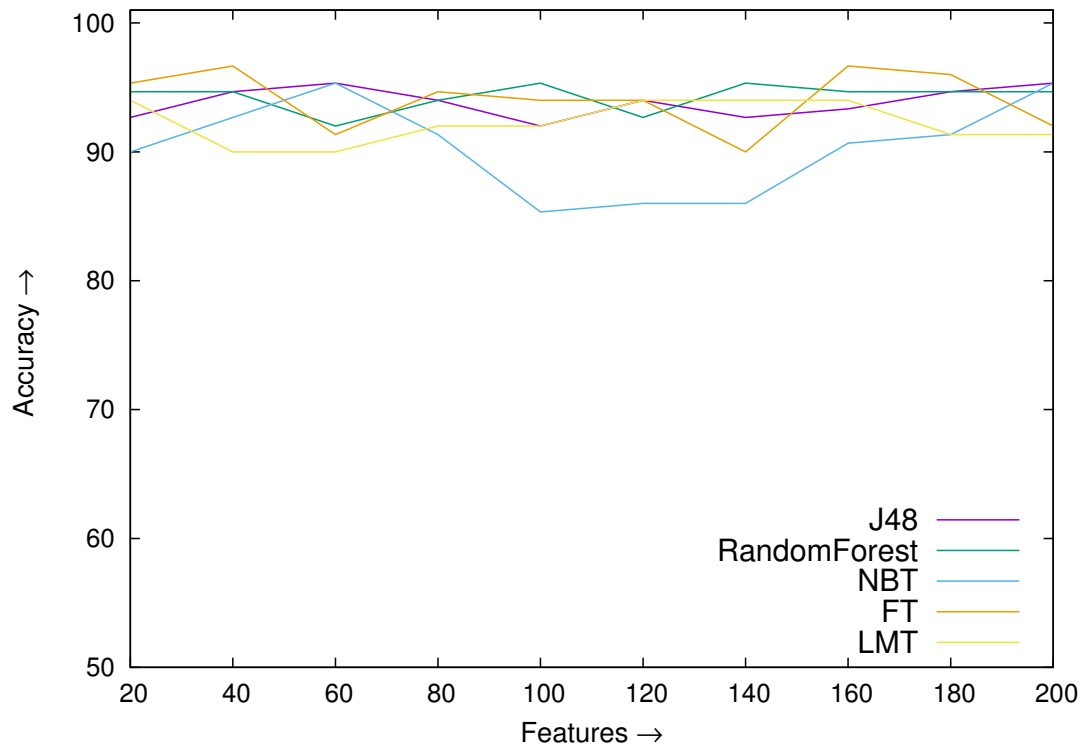


Figure 6.19: Detection accuracy of the classifiers for the CAMERA group.

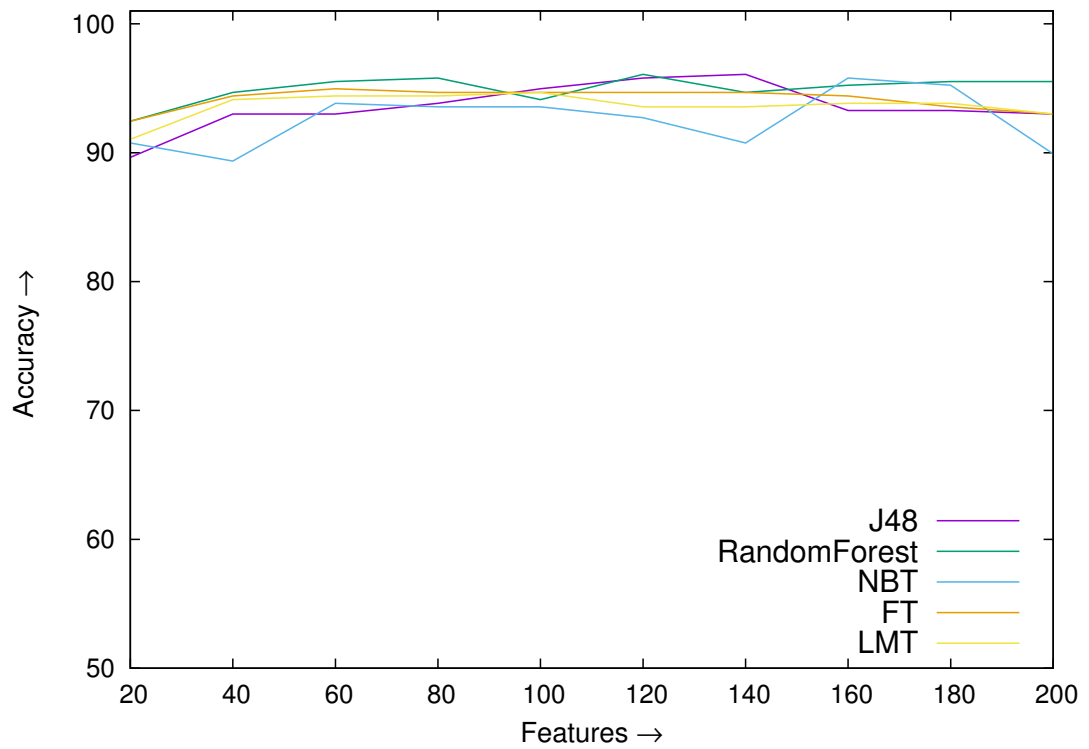


Figure 6.20: Detection accuracy of the classifiers for the CONTACTS group.

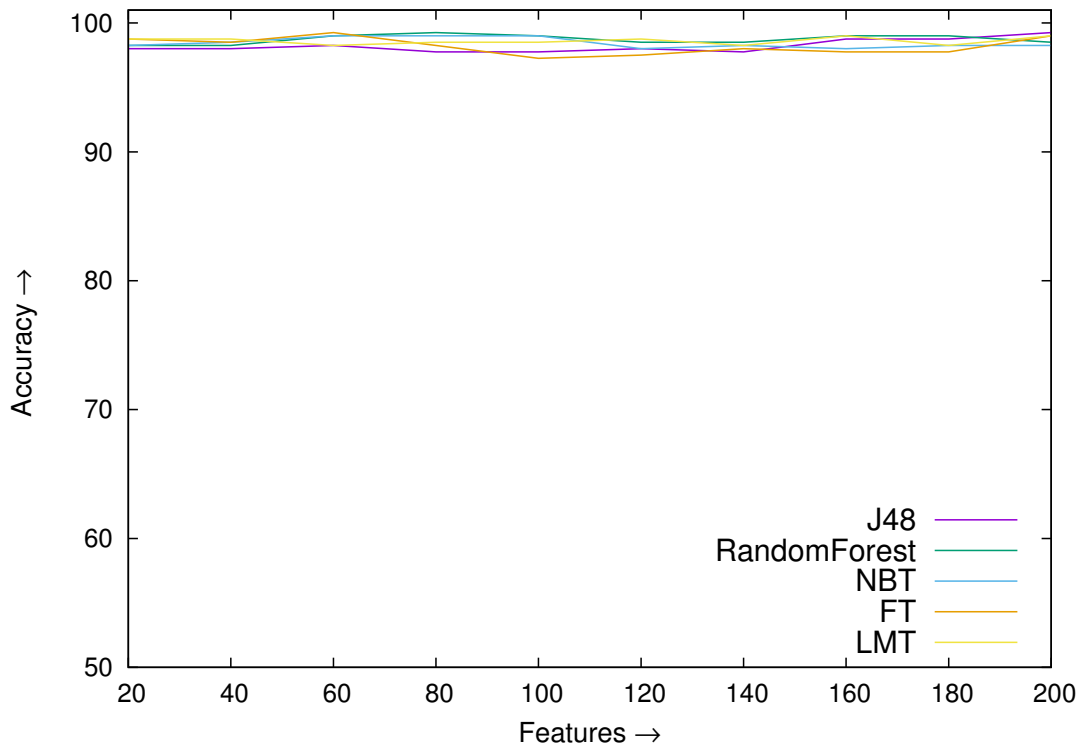


Figure 6.21: Detection accuracy of the classifiers for the LOCATION group.

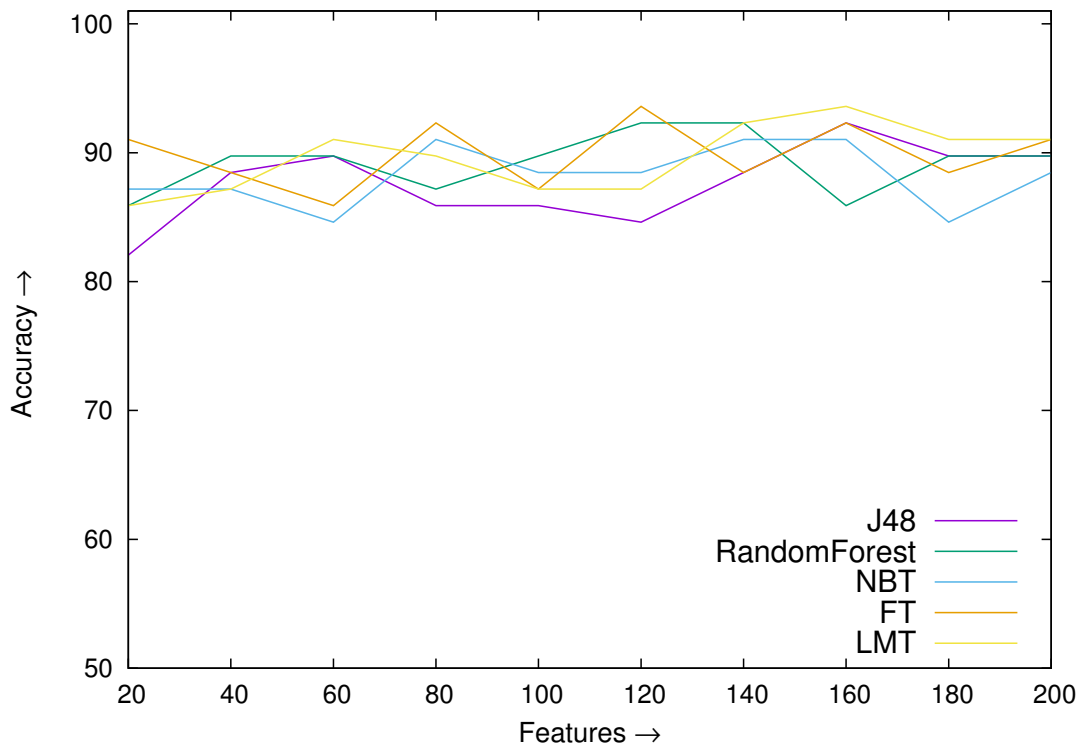


Figure 6.22: Detection accuracy of the classifiers for the MICROPHONE group.

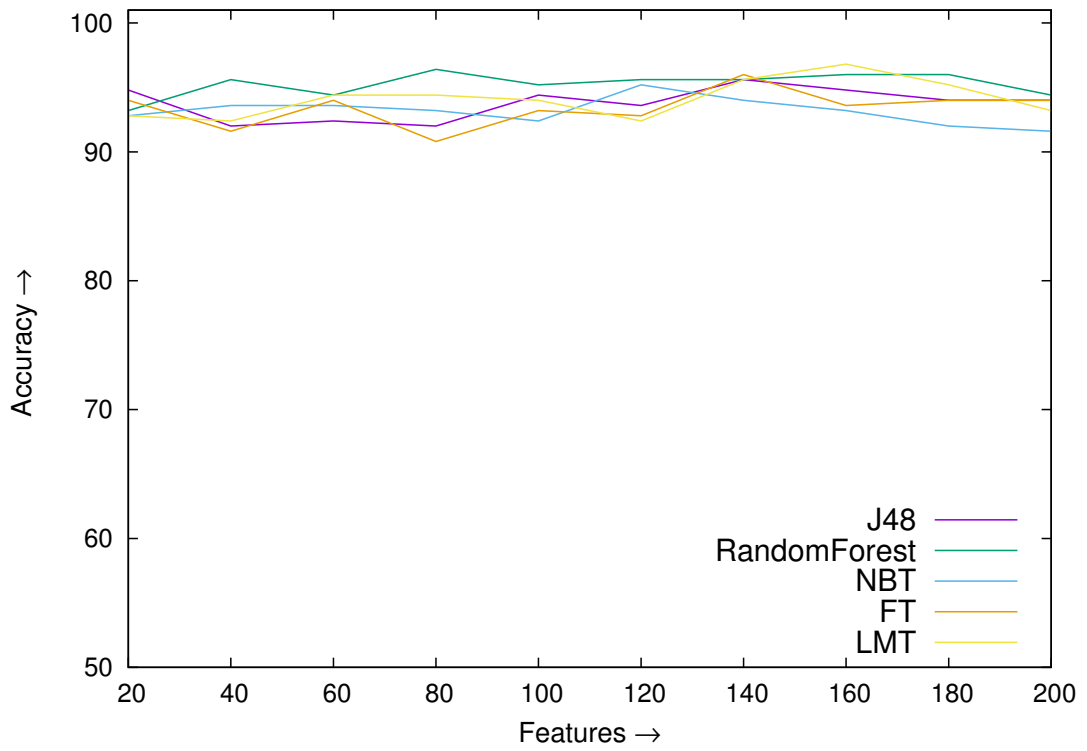


Figure 6.23: Detection accuracy of the classifiers for the OTHERS group.

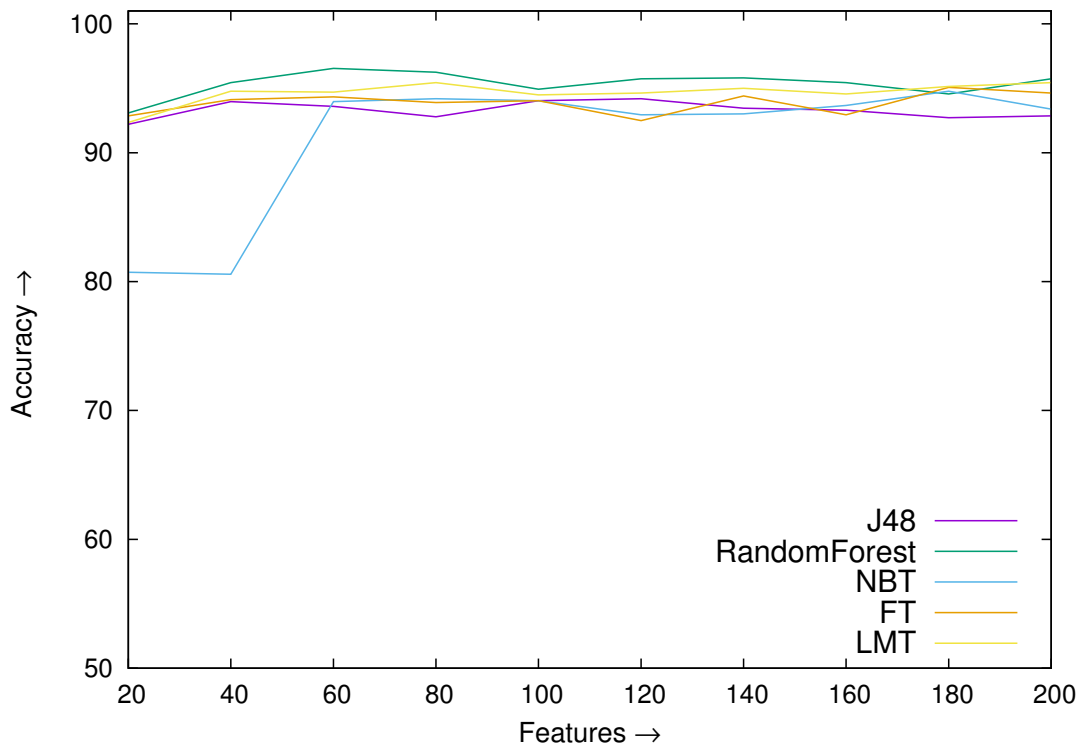


Figure 6.24: Detection accuracy of the classifiers for the PHONE group.



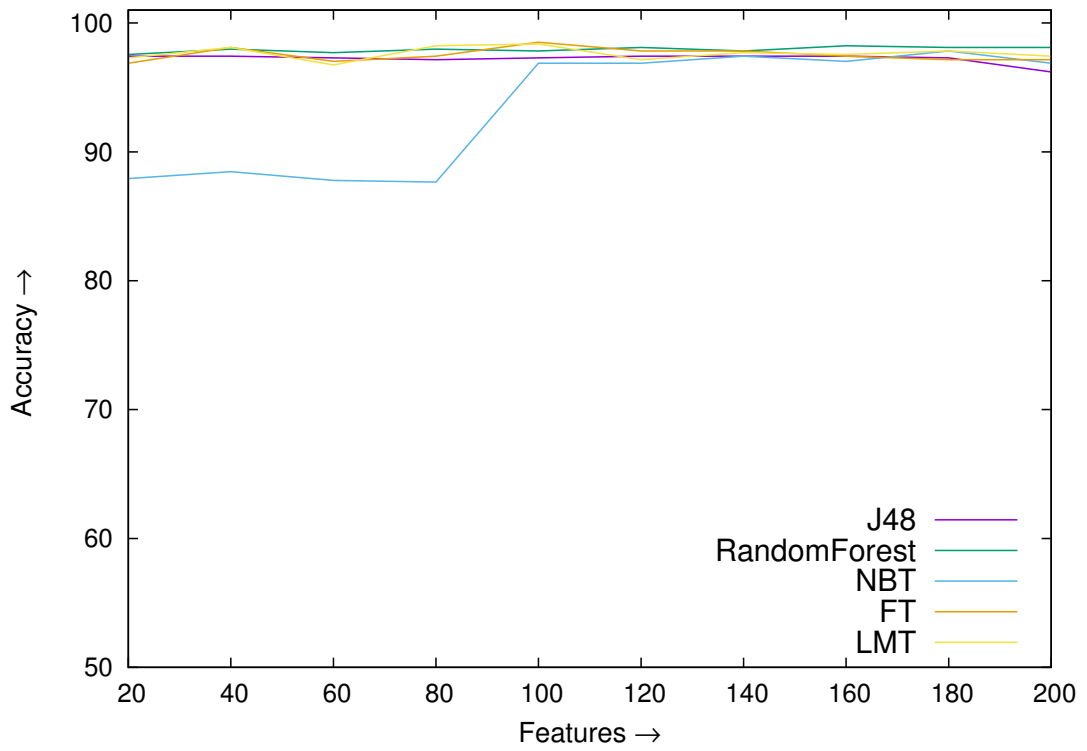


Figure 6.25: Detection accuracy of the classifiers for the SMS group.

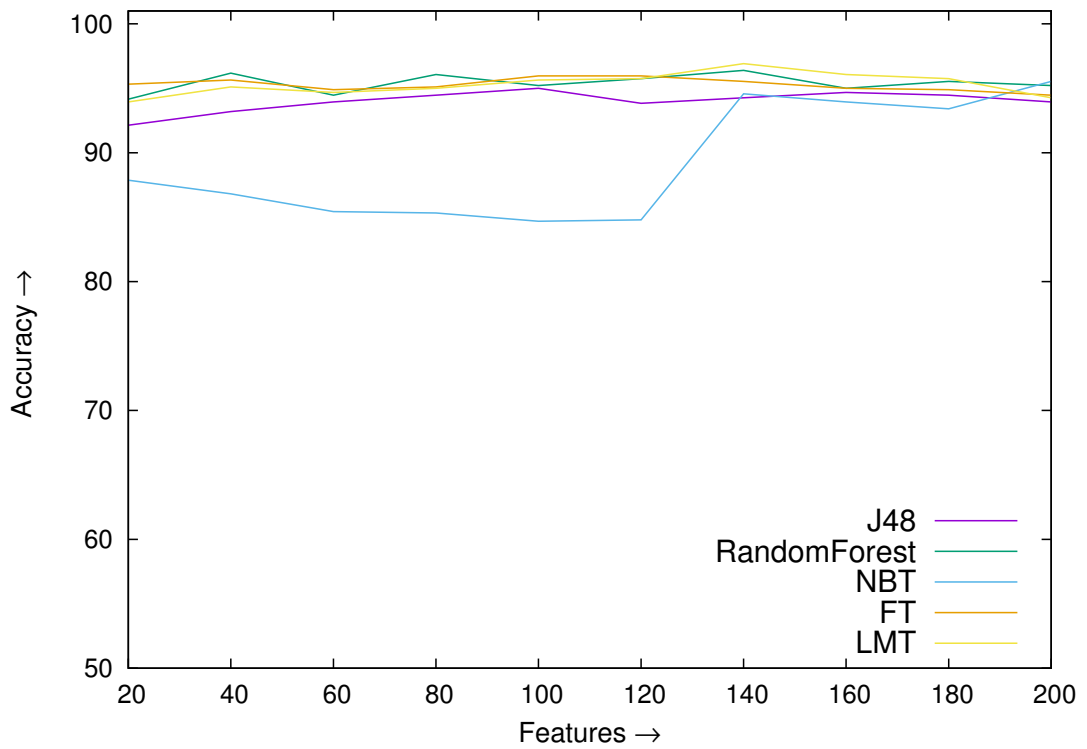


Figure 6.26: Detection accuracy of the classifiers for the STORAGE group.

No. of Features	J48	Random Forest	NBT	FT	LMT
20	93.69	95.01	90.37	93.32	94.28
40	95.28	96.26	92.26	93.78	93.45
60	95.51	96.10	94.24	94.01	94.31
80	94.83	<b>96.32</b>	94.44	95.38	95.46
100	<b>95.15</b>	96.24	94.41	<b>95.43</b>	<b>85.47</b>
120	94.48	95.96	92.96	94.57	94.23
140	95.12	96.08	93.68	93.53	94.76
160	95.39	95.16	<b>94.97</b>	95.16	94.29
180	94.94	95.73	93.93	95.18	94.56
200	94.71	95.78	93.24	94.98	94.71
<b>Maximum</b>	<b>95.51</b>	<b>96.32</b>	<b>94.97</b>	<b>95.43</b>	<b>95.47</b>
<b>Minimum</b>	<b>93.69</b>	<b>95.01</b>	<b>90.37</b>	<b>93.32</b>	<b>93.45</b>

Table 6.3: Average accuracy obtained by the classifiers.

The average accuracy obtained (here, the average accuracy means the sum of accuracy obtained by the classifier in the individual group with a fixed number of features divided by the total number of groups) by the selected classifier are shown in Table 6.3. The analysis shows that the RF average detection accuracy is best among the five classifiers and fluctuates least with the number of features, whereas NBT performance is worst and fluctuates maximum with the number of features. However, the maximum average accuracy obtained by the selected five classifiers does not fluctuate much (94.97% - 96.32%) but minimum average accuracy fluctuation is high (90.37% - 95.01%), and for the best performance one shall take top 80 - 100 features, for the training and testing. The best accuracy obtained by the classifier in all the groups are given in Table 6.4. We find that the detection accuracy is maximum in the CALENDAR group and

Groups	Best Classifier	Accuracy	Features Required	TN	TP
CALENDAR	RF	100.00	20	1.00	1.00
CAMERA	FT	96.67	40	0.93	0.98
CONTACTS	RF	96.08	120	0.99	0.89
LOCATION	FT	99.25	60	0.99	0.94
MICROPHONE	FT	93.59	120	0.87	0.96
OTHERS	LMT	96.80	160	0.85	0.98
PHONE	RF	96.54	60	0.98	0.92
SMS	FT	98.51	100	1.00	0.80
STORAGE	LMT	96.91	140	0.99	0.88

Table 6.4: Group-wise maximum accuracy, TP and TN of the classifiers.

minimum in the MICROPHONE group (Figure 6.27) obtained by FT and RF classifier

respectively. The overall average maximum accuracy comes to 97.15%, which is very much better than the obtained accuracy without grouping the dataset (Sec. 6.3), Bahman Rashidi et. al. [76], Annamalai et. al [70], Arp, et. al. [15] and Ali Feizollah, et. al. [36] (Figure 6.28). In terms of  $TP$  i.e. detection rate of malicious apps, the CALENDAR group are best classified by RF and SMS group are least by FT, while in terms of  $TN$  i.e. benign detection rate, CALENDAR and SMS group are best classified by RF and FT classifier respectively, while OTHER group containing normal permissions is best classified by the LMT classifier. The group-wise best results of  $TP$  and  $TN$  obtained by the classifiers which give the best accuracy is shown in Table 6.4 and is depicted in Figure 6.29.

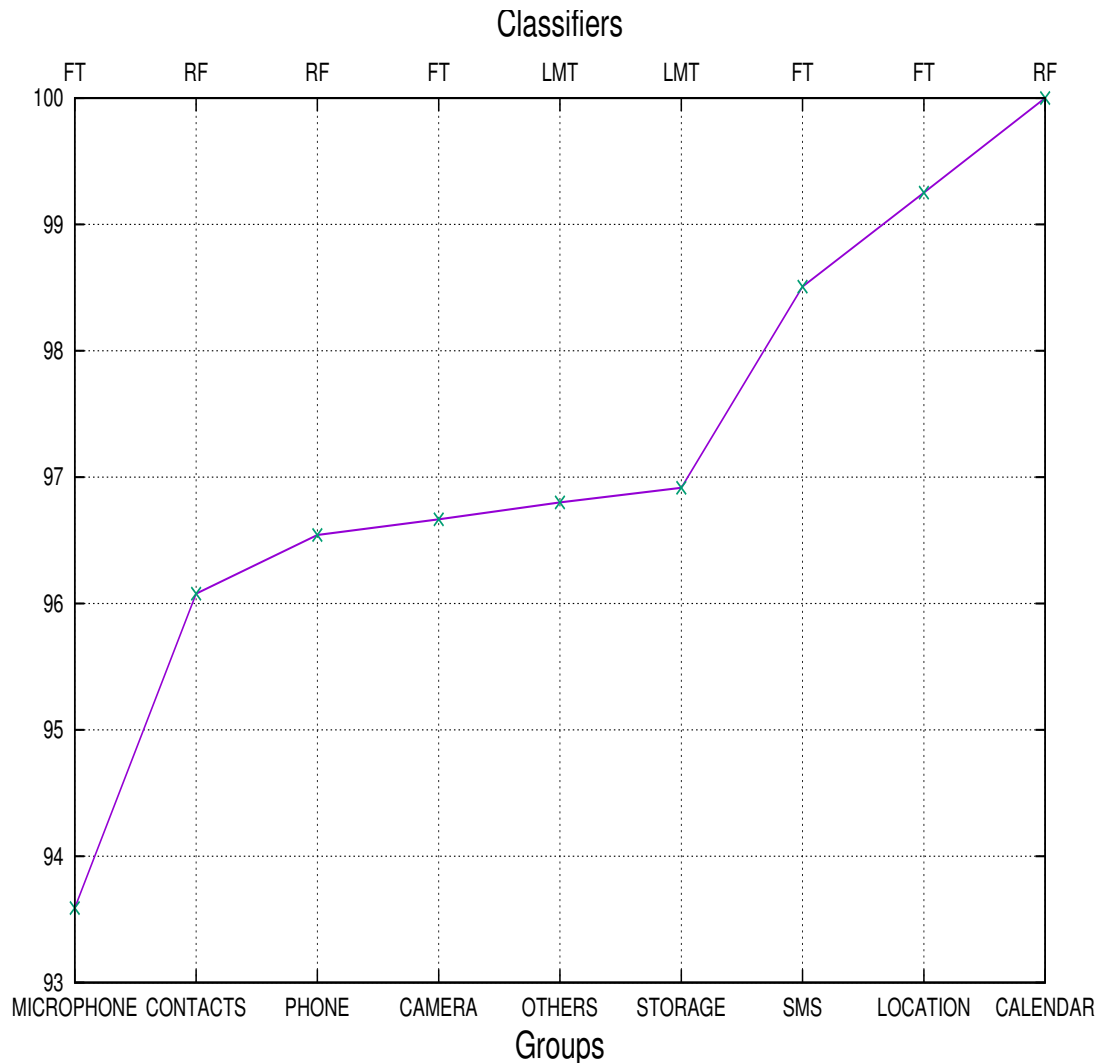


Figure 6.27: Group-wise best detection accuracy obtained by the classifiers.

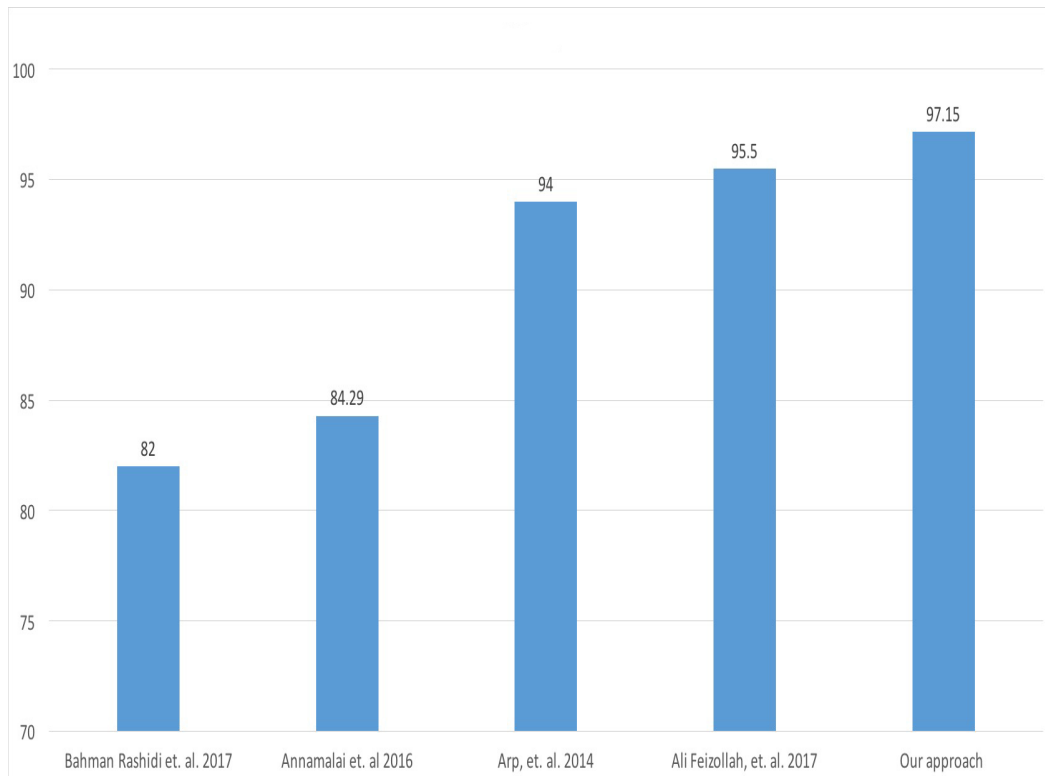


Figure 6.28: Comparisons of accuracy achieved by us with four other authors.

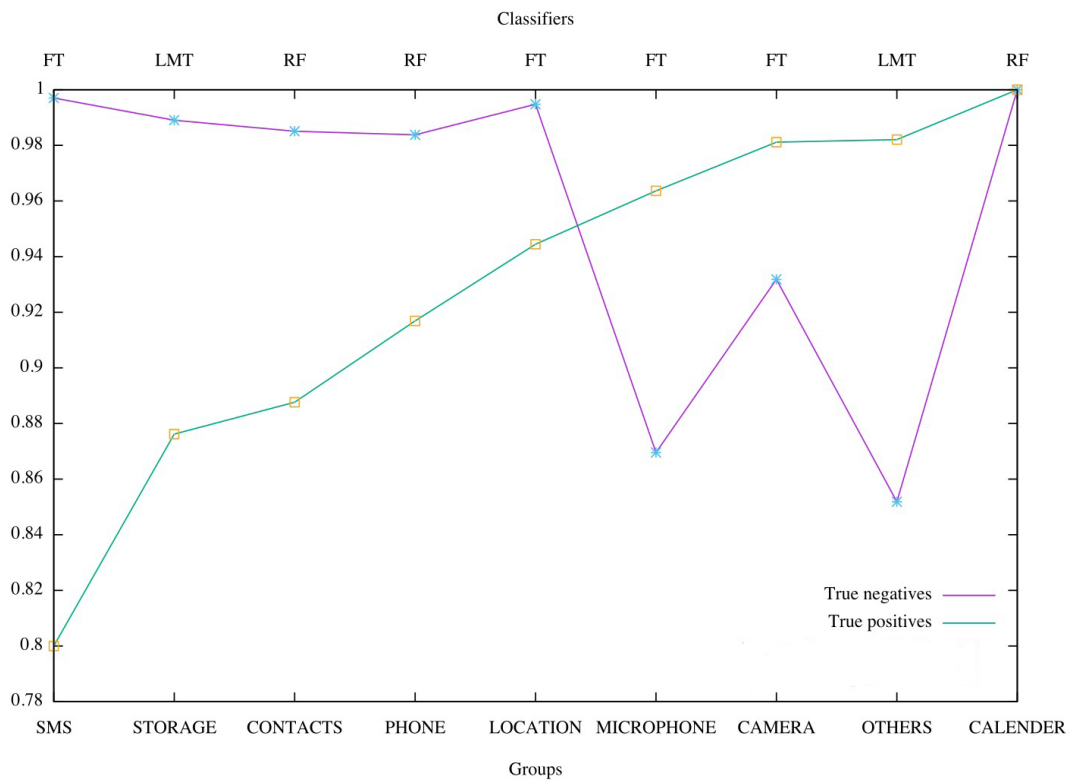


Figure 6.29: Group-wise best TP and TN of the classifiers.

## 6.6 Summary

The threats/attacks from the malicious apps in Android devices are now never seen at before levels, as millions of android apps are available officially and by the third party. Some of these available apps may be malicious, hence these devices are very much vulnerable to cyber threat/attack. The consequence will be devastating if in time counter-measures are not developed. Therefore, in this chapter, using *Drebin* benchmark malware dataset, first without grouping the data we investigated the five classifiers (FT, RF, J48, LMT, and NBT) for the detection of malicious apps. We found that among the studied classifiers, FT is the best classifiers to detect the malware ( $\sim 79.27\%$  accuracy). However, true positives i.e. malware detection rate is highest ( $\sim 99.91\%$ ) by RF and fluctuate least with the different number of prominent features compared to other studied classifiers, and is better than the BooJoong et. al., F-measure (98%) [52]. The analysis shows that overall accuracy is majorly affected by the false positives of the classifier.

Later, we group-wise analyzed the dataset based on permissions, and experimentally demonstrated how to improve the detection accuracy of Android malicious apps, and achieved up to 97.15% overall average accuracy. The obtained results outperformed the accuracy achieved by without grouping the data (79.27%, 2017), Arp, et. al. (94%, 2014), Annamalai et. al. (84.29%, 2016), Bahman Rashidi et. al. (82%, 2017)) and Ali Feizollah, et. al. (95.5%, 2017). Among these groups, the MICROPHONE group detection accuracy is least while CALENDAR group apps are detected with maximum accuracy and for the best performance, one shall take top 80 - 100 features for the training and testing. In terms of TP i.e. detection rate of malicious apps, CALENDAR group is best classified by RF, and SMS group is least by FT, while in terms TN i.e. benign detection rate, CALENDAR, and SMS group are best classified by RF, and FT classifier respectively, while OTHER group containing normal permissions is best classified by the LMT classifier.

## CHAPTER 7

# CONCLUSIONS AND FUTURE DIRECTIONS

Nowadays most of the computational devices are connected to the Internet, as a consequence, these devices are very much vulnerable to the cyber threat/attack from the advanced malware. It can penetrate networks, steal confidential information from desktops and smart devices, bring down servers and can cripple infrastructures etc. To combat the threat/attacks from the malware, anti-malware have been developed. The existing anti-malware are mostly based on the assumption that the malware structure does not change appreciably. But the recent advancement in second generation malware, which can create millions of its variants have posed challenges to the anti-malware developers, and it is an indisputable fact that the traditional approach to combat the threats/attack from today's highly sophisticated customized second generation malware with a technology-centric are ineffective.

For the detection of malware, feature selection plays a vital role, not only to represent the target concept but also to speed-up the learning and testing process. In this, often datasets are represented by many features, however, few of them are sufficient to improve the concept quality, and also limiting the features will speed-up the classification. Therefore, to find the prominent features which can represent the target concept, we studied the occurrence of opcodes in both benign and malware separately and found that the occurrence of many opcodes in malware and benign executables differ in large. Thus for the detection of malware, we selected the top

opcodes as features, whose occurrence significantly differ between the malware and benign executables.

The metamorphic malware variants lead to a huge signature database for the detection by traditional signature based techniques. Therefore, for the effective detection of unknown advanced malware for Windows Desktops, we present a novel approach by first investigating the variation in the size of malware generated by metamorphic malware generator kits and then group-wise classifying the collected dataset in 100 groups. We found that, if features are selected by partitioning dataset in the range of 5 KB and then classified, the malware are detected with 8.7% more accurate than the regular method.

To find the best classifier for the detection of unknown/advanced malware, we studied the performance of the popular thirteen classifiers using N-fold cross-validation available in machine learning tool WEKA with the Malicia data set. Among these thirteen classifiers, we did an extensive experiment to study the performance of the top five classifiers viz. RF, LMT, NBT, J48, and FT in terms of TPR, TNR, FPR, FNR, and accuracy by analyzing benchmark Malicia dataset and benign programs collected from different systems. By our approach, all five classifiers are able to uncover unknown malware with more than 96.28% accuracy, which is better than the detection accuracy (95.9%) reported by Santos et. al. (2013). Among these classifiers, we found that RF is the best (97.95%) classifier to detect the unknown malware. Thus, our approach outperforms to detect the unknown malware. Hence, it can be an effective technique to complement the signature based mechanism or dynamic approach for the detection of unknown/advanced malware.

According to our study, group-wise classification of data improves the detection accuracy of unknown malware. Therefore, we partitioned the executables in nine groups using Optimal k-Means Clustering algorithm and then features are selected separately from each formed groups by finding the difference of opcode occurrence between benign and malware executables. Then we used the top five studied classifiers viz. RF, J48, LMT, FT, and NBT for the detection of malware and found that all of them give more than 98% of detection accuracy, whereas NBT gives the highest accuracy (99.11%).

The threat/attack from the malicious apps in Android based devices is now never seen at before levels, as millions of Android apps are available officially and unofficially. Some of these available apps may be malicious, hence these devices are very much vulnerable to cyber threat/attack. The consequence will be devastating if in time counter-measures are not developed. Therefore, similar to Windows Desktops malware analysis, first without making the groups, we investigated the five classifiers (FT, RF, J48, LMT, and NBT) for the detection of malicious apps using *Drebin* benchmark dataset. We found that among the five studied classifiers, FT is the best classifier and detect the malware with 79.27% accuracy. However, highest TP (99.91) is obtained by RF, and it fluctuates least with the number of prominent features compared to other classifiers. The obtained result is better than the BooJoong et. al. F-measure (98%), and the analysis shows that overall accuracy is majorly affected by the FP of the classifiers.

The experimental analysis of the Windows Desktops executables shows that group-wise classification improves the detection accuracy. Therefore we group-wise analyzed the collected dataset based on permissions, and experimentally demonstrated how to improve the detection accuracy of Android malicious apps. The obtained results (97.15% average accuracy) outperformed the accuracy achieved by without grouping the data (79.27%, 2017), Arp, et. al. (94%, 2014), Annamalai et. al. (84.29%, 2016), Bahman Rashidi et. al. (82%, 2017)) and Ali Feizollah, et. al. (95.5%, 2017). Among the groups, the MICROPHONE group detection accuracy is least while CALENDAR group apps are detected with maximum accuracy. In term of TP i.e. detection rate of malicious apps, CALENDAR group is best classified with RF, and SMS group is least by FT, while in terms TN i.e. benign detection rate, CALENDAR, and SMS group are best classified with RF and FT classifier respectively, while OTHER group containing normal permissions is best classified by the LMT classifier.

To detect the advanced malware generally machine learning techniques are used. However, recent development in deep learning, which has been proved very successful in other fields can be applied for the effective classification of advanced malware to decrease the FP and FN to improve the overall detection accuracy. In this, different architectures and combinations of the algorithms, using different parameters can be



investigated. Grid Search Cross-Validation or other similar methods shall be studied with different combinations of the architectures and then select the best algorithm to design a cognitive system for the detection of advanced malware. Also, the feasibility of integrating our solution can be explored with dynamic detection techniques by profiling dynamic features like system calls, network connections, resources usage, etc.

From the investigations, we concluded that the group-wise detection of malware is more effective than without grouping the data. However, if the attacker knows the internal parameters, then it will be prone to adversary attacks because the approach of the malware detection uses the size of binaries or dangerous permissions as a criterion for building the group based models. The attacker can change these parameters in the adversary sample, which may reduce the accuracy of the classifiers. Also, in the individual group's number of training samples may not be sufficient, which may effect the accuracy of the results. Further, for the efficient classification of malware, in-depth study is required to optimize the feature selection, identifying the best-suited classifier for the group-by-group analysis, and to implement the developed approach together with the traditional technique in general-purpose graphics processing unit can make the detection more efficient. The work presented in the thesis can be extended to the other growing operating systems (IOS, LINUX etc) for the detection of malware.

# APPENDIX A

## OBFUSCATION TECHNIQUES

To evade the traditional signature based malware detection technique, the Polymorphic and Metamorphic malware uses obfuscation techniques to change their internal structure which makes them look different, but the functionality remains same. Below we discuss some of the obfuscation techniques.

### A.1 Register Renaming

In this technique after each infection, the mutation engine renames the registers or memory variables used in the malware, which changes in the internal structure of the malware variant and can bypass the detection technique.

```
a)
5A                pop  edx
BF04000000       mov  edi,0004h
8BF5             mov  esi,ebp
B80C000000       mov  eax,000Ch
81C288000000     add  edx,0088h
8B1A             mov  ebx,[edx]
899C8618110000   mov  [esi+eax*4+00001118],ebx

b)
58                pop  eax
BB04000000       mov  ebx,0004h
8BD5             mov  edx,ebp
BF0C000000       mov  edi,000Ch
81C088000000     add  eax,0088h
8B30             mov  esi,[eax]
89B4BA18110000   mov  [edx+edi*4+00001118],esi
```

Figure A.1: A variant of RegWswap virus code by register renaming.

This technique was first used in the Win95/Regswap virus, which was developed by Vecna. Whenever this virus propagates, the mutation engine transforms the code by using different register names in its variant. An example (RegSwap virus) of register renaming by mutation engine is shown in Figure A.1. The bold hexadecimal code in the figure shows the similarity between two the variants.

## A.2 Subroutine Permutation

In this obfuscation technique, the malware code is first divided into independent subroutines (blocks of codes). Then the mutation engine changes the order of these subroutines without modifying the functionality of code (Figure A.2). This technique can generate  $n!$  numbers of variants where  $n$  is the number of independent subroutines. For e.g. Win32/Ghost have 10 subroutines and can make  $10!$  different permutation. Thus generates a large number (3628800) of variants of the Win32/Ghost malware.

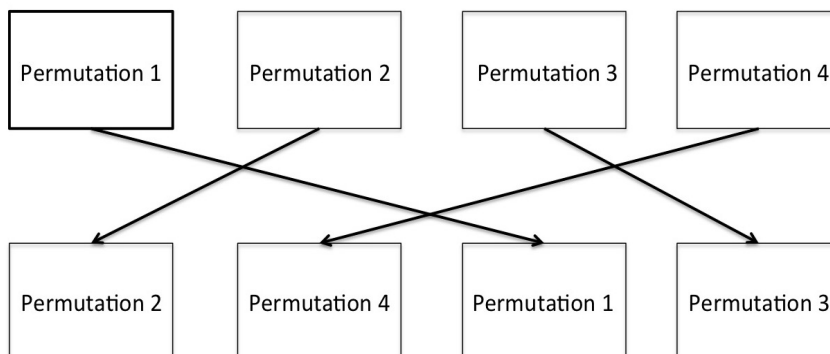


Figure A.2: Subroutines permutation.

## A.3 Instruction Level Permutation

In this technique, the malware variant code is created by changing the order of instructions, such that there is no dependency among the instructions, e.g. as shown in Figure A.3 and A.4, the result of final computation of the variant of malware does not change.

---

```
int x=5;
int y=2;
int z=x+y;
```

---

Figure A.3: Simple C code.

---

```
int y=2;
int x=5;
int z=x+y;
```

---

Figure A.4: Variation in the C code after using instruction level permutation.

## A.4 Insertion of Jump Instructions

This obfuscation technique is used in many metamorphic malware which inserts jump statements after a block of instructions such that the control flow of the program remains same (e.g. Figure A.5). WIn95/Zperm is one of the malware which uses this technique.

---

```
[Original Program]
instruction 1 ; entry point
instruction 2
instruction 3
instruction 4
instruction 5
```

---

---

```
[Transformed program]
instruction 4
jump 5
instruction 2
jump 3
instruction 3
jump 4
instruction 1 ; entry point
jump 2
instruction 5
```

---

Figure A.5: Insertion of jump instructions to create new variants.

## A.5 Subroutine Inlining and Outlining

In this technique, calls of the subroutines are replaced by its codes, e.g. an inline of the subroutines are shown in Figure A.6, in which code variant A is transformed to code variant B and vice-versa for subroutine outlining.

---

<pre>/* some instructions */ call Function 1 call Function 2 /* some instructions */ Function 1: mov eax, ebx add eax, 12h push eax ret Function 2: mul ecx mov edx, eax ret</pre>	<pre>/* some instructions */ mov eax, ebx add eax, 12h push eax mul ecx mov edx, eax /* some instructions */</pre>
--	--

(a) Code variant A                      (b) Code variant B

---

Figure A.6: Inlining and outlining of subroutines.

## A.6 Dead Code Insertion

Dead code/garbage insertion is a technique in which ineffective instructions/subroutine are inserted in the malware code to create a new variant. Adding the dead codes in malware is an easy way to transform it to different variant without changing its functionality, e.g. NOPs opcodes, the addition of 0 value to a variable, assigning same value to register, using jump instruction for next instruction, etc. obfuscate the malware and a create new variant. Win32/Evol virus (Figure A.7 shows its snippet) is a metamorphic malware which used this technique.

---

```

1 C7060F000055  MOV [esi], 5500000Fh
2 C746048BEC5151  MOV [esi+0004], 5151EC8Bh
3 BF0F00055      MOV edi, 5500000Fh
4 893E          MOV [esi], edi
5 5F           POP edi           ; garbage
6 52           PUSH edx          ; garbage
7 B640         MOV dh, 40        ; garbage
8 BA8BEC5151   MOV edx, 5151EC8Bh
9 53           PUSH ebx          ; garbage
10 8BDA        MOV ebx, edx
11 895E04      MOV [esi+0004], ebx

```

---

Figure A.7: Snippet of Evol virus using dead code insertion.

## A.7 Equivalent Code Substitution

In this technique, the variants are created by exchanging some instructions with other equivalent set of instructions so that the functionality of code remains same for e.g. below given instructions

- mov ebx, 0
- xor ebx, ebx
- and ebx, 0
- sub ebx, ebx

are similar in operation i.e. assigning ebx register to value 0.

## APPENDIX B

### WINDOWS DESKTOPS OPCODE LIST

ID	Opcode	ID	Opcode	ID	Opcode
1	.byte	20	addr32	39	addr64
2	(bad)	21	addsubps	40	addsd
3	aaa	22	addr32csrex.B	41	addss
4	aad	23	addr32csrex.RXB	42	addsubpd
5	aam	24	addr32fsrex.RX	43	addw
6	aas	25	addr32gsrex.R	44	aesdec
7	adc	26	addr32gsrex.RXB	45	aesdeclast
8	adcb	27	addr32gsrex.W	46	aesenc
9	adcl	28	addr32gsrex.WRB	47	aesenclast
10	adcq	29	addr32gsrex.WRX	48	aesimc
11	adcw	30	addr32rex	49	aeskeygenassist
12	add	31	addr32rex.B	50	and
13	addb	32	addr32rex.R	51	andb
14	addl	33	addr32rex.RXB	52	andl
15	addpd	34	addr32rex.WR	53	andnpd
16	addps	35	addr32rex.WRB	54	andnps
17	addq	36	addr32rex.WRX	55	andpd
18	addr16	37	addr32rex.WXB	56	andps
19	adcx	38	addr32rex.X	57	andq

ID	Opcode	ID	Opcode	ID	Opcode
58	andw	90	cmovg	122	comisd
59	arpl	91	cmovge	123	comiss
60	blefill	92	cmovl	124	cpuid
61	bound	93	cmovle	125	cqto
62	bsf	94	cmovne	126	cs
63	bsr	95	cmovno	127	cvtpd2dq
64	bswap	96	cmovnp	128	csrex.B
65	bt	97	cmovns	129	csrex.R
66	btc	98	cmovo	130	csrex.RB
67	btcl	99	cmovp	131	csrex.RXB
68	btl	100	cmovs	132	csrex.W
69	btr	101	cmp	133	csrex.WB
70	btrl	102	cmpb	134	csrex.WR
71	bts	103	cmpeqsd	135	csrex.WRB
72	btsl	104	cmpl	136	csrex.WRX
73	btsq	105	cmpltpd	137	csrex.WX
74	call	106	cmpltsd	138	csrex.WXB
75	callq	107	cmpltss	139	csrex.X
76	callw	108	cmpneqpd	140	csrex.XB
77	cbtw	109	cmpneqps	141	cvtdq2pd
78	clc	110	cmpnlepd	142	cvtdq2ps
79	cld	111	cmppd	143	cvtpd2ps
80	cli	112	cmpps	144	cvtpi2ps
81	cltd	113	cmpq	145	cvtps2pd
82	cltq	114	cmpsb	146	cvtps2pi
83	clts	115	cmpsl	147	cvtsd2si
84	cmc	116	cmpsq	148	cvtsd2ss
85	cmova	117	cmpsw	149	cvtsi2sd
86	cmovae	118	cmpunordps	150	cvtsi2sdq
87	cmovb	119	cmpw	151	cvtsi2ss
88	cmovbe	120	cmpxchg	152	cvtss2sd
89	cmove	121	cmpxchg8b	153	cvttpd2dq



ID	Opcode	ID	Opcode	ID	Opcode
154	cvttpd2pi	185	divb	216	extrq
155	cvttps2pi	186	divl	217	f2xm1
156	cvttss2si	187	divpd	218	fabs
157	cvttss2si	188	divps	219	fadd
158	cwtd	189	divq	220	faddl
159	cwtl	190	divsd	221	faddp
160	daa	191	divss	222	fadds
161	das	192	divw	223	fbld
162	data16	193	d3d8	224	fbstp
163	data16addr32rex.WRB	194	ds	225	fchs
164	data16data16rex.R	195	dsrex	226	fclex
165	data16data16rex.WXB	196	dsrex.R	227	fcmovb
166	data16gsrex.RXB	197	dsrex.RXB	228	fcmovbe
167	data16gsrex.WRB	198	dsrex.WB	229	fcmove
168	data16rex	199	dsrex.WR	230	fcmovnb
169	data16rex.B	200	dsrex.WRB	231	fcmovnbe
170	data16rex.R	201	dsrex.WRX	232	fcmovne
171	data16rex.W	202	dsrex.WRXB	233	fcmovnu
172	data16rex.WB	203	emms	234	fcmovu
173	data16rex.WRB	204	enter	235	fcom
174	data16rex.WRX	205	enterq	236	fcomi
175	data16rex.WRXB	206	enterw	237	fcomip
176	data16rex.WXB	207	es	238	fcoml
177	data16rex.XB	208	encls	239	fcomp
178	data32	209	esrex.RB	240	fcompl
179	dec	210	esrex.RX	241	fcompp
180	decb	211	esrex.W	242	fcomps
181	decl	212	esrex.WB	243	fcoms
182	decq	213	esrex.WRX	244	fcos
183	decw	214	esrex.X	245	fdecstp
184	div	215	esrex.XB	246	fdiv

ID	Opcode	ID	Opcode	ID	Opcode
247	fdivl	278	fistpll	309	fnsave
248	fdivp	279	fisttp	310	fnsaves
249	fdivr	280	fisttpl	311	fnsetpm(287
250	fdivrl	281	fisttpll	312	fnstcw
251	fdivrp	282	fisub	313	fnstenv
252	fdivrs	283	fisubl	314	fnstenvs
253	fdivs	284	fisubr	315	fnstsw
254	femms	285	fisubrl	316	fpatan
255	ffree	286	fld	317	fprem
256	ffreep	287	fld1	318	fprem1
257	fiadd	288	fldcw	319	fptan
258	fiaddl	289	fldenv	320	frndint
259	ficom	290	fldenvs	321	frstor
260	ficoml	291	fldl	322	frstors
261	ficomp	292	fldl2e	323	frstpm(287
262	ficompl	293	fldl2t	324	fs
263	fidiv	294	fldlg2	325	fstp1
264	fidivl	295	fldln2	326	fsave
265	fidivr	296	fldpi	327	fscale
266	fidivrl	297	flds	328	fsfsrex
267	fld	298	fldt	329	fsfsrex.RX
268	fldl	299	fldz	330	fsfsrex.WRB
269	fldll	300	fmul	331	fsfsrex.WXB
270	fimul	301	fmull	332	fsgsrex.W
271	fimull	302	fmulp	333	fsgsrex.WB
272	fincstp	303	fmuls	334	fsgsrex.WR
273	finit	304	fnclx	335	fsin
274	fist	305	fndisi(8087	336	fsincos
275	fistl	306	fneni(8087	337	fsqrt
276	fistp	307	fninit	338	fsrex
277	fistpl	308	fnop	339	fsrex.B

ID	Opcode	ID	Opcode	ID	Opcode
340	fsrex.R	371	ftst	402	gsgsrex.R
341	fsrex.RB	372	fucom	403	gsgsrex.W
342	fsrex.RX	373	fucomi	404	gsrex
343	fsrex.RXB	374	fucomip	405	gsrex.B
344	fsrex.W	375	fucomp	406	gsrex.R
345	fsrex.WB	376	fucompp	407	gsrex.RB
346	fsrex.WR	377	fwait	408	gsrex.RXB
347	fsrex.WRB	378	fxam	409	gsrex.W
348	fsrex.WRX	379	fxch	410	gsrex.WB
349	fsrex.WRXB	380	fxrstor	411	gsrex.WR
350	fsrex.WX	381	fxsave	412	gsrex.WRB
351	fsrex.WXB	382	fxtract	413	gsrex.WRX
352	fsrex.XB	383	fyl2x	414	gsrex.WRXB
353	fst	384	fyl2xp1	415	gsrex.WXB
354	fstcw	385	getsec	416	gsrex.X
355	fstenv	386	gs	417	gsrex.XB
356	fstl	387	gdipus	418	hlt
357	fstp	388	gsaddr32rex.R	419	hsubps
358	fstpl	389	gsaddr32rex.WXB	420	icebp
359	fstps	390	gscsrex.RXB	421	idiv
360	fstpt	391	gsdata16rex	422	idivb
361	fstps	392	gsdsrex.R	423	idivl
362	fstsw	393	gsdsrex.RXB	424	idivq
363	fsub	394	gsdsrex.WRB	425	idivw
364	fsubl	395	gsdsrex.WRX	426	imul
365	fsubp	396	gsesrex.W	427	imulb
366	fsubr	397	gsfsrex.B	428	imull
367	fsubrl	398	gsfsrex.RB	429	imulq
368	fsubrp	399	gsfsrex.W	430	imulw
369	fsubrs	400	gsfsrex.WR	431	in
370	fsubs	401	gsfsrex.WRB	432	inc

ID	Opcode	ID	Opcode	ID	Opcode
433	incb	464	jecxz	495	jo,pn
434	incl	465	jecxz,pn	496	jo,pt
435	incq	466	jecxz,pt	497	jp
436	incw	467	jg	498	jp,pn
437	insb	468	jg,pn	499	jp,pt
438	insl	469	jg,pt	500	jrcxz
439	insw	470	jge	501	jrcxz,pn
440	int	471	jge,pn	502	jrcxz,pt
441	int3	472	jge,pt	503	js
442	into	473	jl	504	js,pn
443	invd	474	jl,pn	505	js,pt
444	invlpg	475	jl,pt	506	lahf
445	iret	476	jle	507	lar
446	iretq	477	jle,pn	508	lcall
447	iretw	478	jle,pt	509	lcallq
448	ja	479	jmp	510	lcallw
449	ja,pn	480	jmpq	511	lddqu
450	ja,pt	481	jmpw	512	ldmxcsr
451	jae	482	jne	513	lds
452	jae,pn	483	jne,pn	514	lea
453	jae,pt	484	jne,pt	515	leave
454	jb	485	jno	516	leaveq
455	jb,pn	486	jno,pn	517	leavew
456	jb,pt	487	jno,pt	518	les
457	jbe	488	jnp	519	lfs
458	jbe,pn	489	jnp,pn	520	lgdt
459	jbe,pt	490	jnp,pt	521	lgdtl
460	jcxz	491	jns	522	lgs
461	je	492	jns,pn	523	lidt
462	je,pn	493	jns,pt	524	lidtl
463	je,pt	494	jo	525	ljmp

ID	Opcode	ID	Opcode	ID	Opcode
526	ljmpq	556	lretq	586	movntq
527	ljmpw	557	lretw	587	movq
528	lldt	558	lsl	588	movsb
529	lmsw	559	lss	589	movsbl
530	lock	560	ltr	590	movsbq
531	lock	561	maskmovq	591	movsbw
532	lockrex	562	maxps	592	movsd
533	lockrex.B	563	minpd	593	movsl
534	lockrex.WB	564	minps	594	movsldup
535	lockrex.WR	565	minss	595	movslq
536	lockrex.X	566	montmul	596	movsq
537	lockrex.XB	567	mov	597	movss
538	lods	568	movabs	598	movsw
539	loop	569	movapd	599	movswl
540	loop,pn	570	movaps	600	movswq
541	loop,pt	571	movb	601	movupd
542	loope	572	movd	602	movups
543	loope,pn	573	movdq2q	603	movw
544	loope,pt	574	movdqa	604	movzbl
545	loopel	575	movdqu	605	movzbq
546	loopew	576	movhlps	606	movzbw
547	loopew,pn	577	movhps	607	movzwl
548	loopl	578	movl	608	movzwq
549	loopne	579	movlhps	609	movzww
550	loopne,pn	580	movlpd	610	mul
551	loopne,pt	581	movlps	611	mulb
552	loopnel	582	movmskps	612	mull
553	loopnew	583	movntdq	613	mulpd
554	loopw	584	movnti	614	mulps
555	lret	585	movntps	615	mulq

ID	Opcode	ID	Opcode	ID	Opcode
616	mulsd	647	paddd	678	pi2fw
617	mulss	648	paddq	679	pinsrb
618	mulw	649	paddsb	680	pinsrw
619	neg	650	paddsw	681	pmaddubsw
620	negb	651	paddusb	682	pmaddwd
621	negl	652	paddusw	683	pmaxsw
622	negq	653	paddw	684	pmaxub
623	negw	654	palignr	685	pminsw
624	nop	655	pand	686	pminub
625	nopq	656	pandn	687	pmovmskb
626	nopl	657	pause	688	pmulhuw
627	nopw	658	pavgb	689	pmulhw
628	not	659	pavgw	690	pmullw
629	notb	660	pcmpeqb	691	pmuludq
630	notl	661	pcmpeqd	692	pop
631	notw	662	pcmpeqw	693	popa
632	or	663	pcmpgtb	694	popaw
633	orb	664	pcmpgtd	695	popf
634	orl	665	pcmpgtw	696	popfq
635	orpd	666	pcmpistri	697	popfw
636	orps	667	pextrw	698	popl
637	orq	668	pf2id	699	popq
638	orw	669	pf2iw	700	popw
639	out	670	pfcmpeq	701	por
640	outsb	671	pfcmpgt	702	prefetch
641	outsl	672	pfrsqit1	703	prefetchnta
642	outsw	673	phaddbd	704	prefetcht0
643	packssdw	674	phaddd	705	prefetcht1
644	packsswb	675	phadduwq	706	prefetcht2
645	packuswb	676	phsubbq	707	prefetchw
646	paddb	677	pi2fd	708	psadbw

ID	Opcode	ID	Opcode	ID	Opcode
709	pshufb	740	pushf	771	repnzrex.WRXB
710	pshufd	741	pushfq	772	repnzrex.XB
711	pshuflw	742	pushfw	773	repz
712	pshufw	743	pushl	774	repz
713	pslld	744	pushq	775	repzcsrex.XB
714	psllq	745	pushw	776	repzrex.WRB
715	psllw	746	pxor	777	repzrex.WX
716	psrad	747	rcl	778	repzrex.WXB
717	psraw	748	rclb	779	repzrex.X
718	psrld	749	rcll	780	repzrex.XB
719	psrldq	750	rclq	781	ret
720	psrlq	751	rclw	782	retq
721	psrlw	752	rcpps	783	retw
722	psubb	753	rcpss	784	rex
723	psubd	754	rcr	785	retnw
724	psubq	755	rcrb	786	rex.B
725	psubsb	756	rcrl	787	rglpsz
726	psubsw	757	rcrq	788	rex.R
727	psubusb	758	rcrw	789	rex.RA
728	psubusw	759	rdmsr	790	rex.RB
729	psubw	760	rdpmc	791	rex.RBA
730	punpckhbw	761	rdtsc	792	rex.RX
731	punpckhdq	762	rep	793	rglpsz
732	punpckhdq	763	repe	794	rex.RXB
733	punpckhwd	764	repnz	795	rguid
734	punpcklbw	765	repnz	796	rex.W
735	punpckldq	766	repnzrex.R	797	riid
736	punpcklwd	767	repnzrex.RX	798	rex.WB
737	push	768	repnzrex.RXB	799	rsts
738	pusha	769	repnzrex.W	800	rex.WR
739	pushaw	770	repnzrex.WRX	801	roundpd

ID	Opcode	ID	Opcode	ID	Opcode
802	rex.WRB	833	sarw	864	shr
803	rsldt	834	sbb	865	shrb
804	rex.WRX	835	sbbb	866	shrd
805	roundsd	836	sbbi	867	shrl
806	rex.WRXB	837	sbbq	868	shrq
807	rsldt	838	sbbw	869	shrw
808	rex.WX	839	scas	870	shufps
809	rsqrtss	840	seta	871	sidt
810	rex.WXB	841	setae	872	siddt
811	rsts	842	setb	873	sldt
812	rex.X	843	setbe	874	smsw
813	rex.RXB	844	sete	875	sqrtpd
814	rex.XB	845	setg	876	sqrtps
815	rueu	846	setge	877	ss
816	rol	847	setl	878	sz
817	rolb	848	setle	879	ssfsrex.W
818	roll	849	setne	880	ssrex.B
819	rolq	850	setno	881	ssrex.RXB
820	rolw	851	setnp	882	ssrex.WB
821	ror	852	setns	883	ssrex.WRB
822	rorb	853	seto	884	ssrex.WX
823	rorl	854	setp	885	ssrex.WXB
824	rorq	855	sets	886	ssrex.X
825	rorw	856	sgdt	887	stc
826	rsm	857	sgdtl	888	std
827	rsqrtps	858	shl	889	sti
828	sahf	859	shlb	890	stmxcsr
829	sar	860	shld	891	stos
830	sarb	861	shll	892	str
831	sarl	862	shlq	893	sub
832	sarq	863	shlw	894	subb



ID	Opcode	ID	Opcode	ID	Opcode
895	subl	926	vandnps	957	vdivpd
896	subpd	927	vandpd	958	vdivps
897	subps	928	vandps	959	vdivsd
898	subq	929	vcmltpd	960	vdivss
899	subsd	930	vcmpngess	961	vdpps
900	subss	931	vcmppd	962	verr
901	subw	932	vcmpsps	963	verw
902	syscall	933	vcmpsd	964	vfmadd213pd
903	sysenter	934	vcmpss	965	vfmadd231pd
904	sysexit	935	vcomisd	966	vfmadd231ss
905	sysret	936	vcomiss	967	vfnmsubpd
906	test	937	vcvtdq2pd	968	vfrzsd
907	testb	938	vcvtdq2ps	969	vfrzss
908	testl	939	vcvtpd2dq	970	vgatherqpd
909	testq	940	vcvtpd2dqx	971	vhaddpd
910	testw	941	vcvtpd2dqy	972	vhaddps
911	ucomisd	942	vcvtpd2psx	973	vhsubpd
912	ucomiss	943	vcvtps2dq	974	vhsubps
913	ud1	944	vcvtps2pd	975	vlddqu
914	ud2	945	vcvtps2ph	976	vmaskmovdqu
915	unpckhpd	946	vcvtsd2si	977	vmaxpd
916	unpckhps	947	vcvtsd2ss	978	vmaxps
917	unpcklpd	948	vcvtsi2sd	979	vmaxsd
918	unpcklps	949	vcvtsi2ssl	980	vmaxss
919	vaddpd	950	vcvtss2sd	981	vminpd
920	vaddps	951	vcvtss2si	982	vminps
921	vaddsd	952	vcvttd2dq	983	vminsd
922	vaddsubpd	953	vcvttd2dqy	984	vminss
923	vaddsubps	954	vcvttps2dq	985	vmload
924	vaesdec	955	vcvttsd2si	986	vmmcall
925	vandnpd	956	vcvtts2si	987	vmovapd

ID	Opcode	ID	Opcode	ID	Opcode
988	vmovaps	1019	vpackssdw	1050	vphsubwd
989	vmovd	1020	vpacksswb	1051	vpinsrw
990	vmovddup	1021	vpackuswb	1052	vpmacsdqh
991	vmovdqa	1022	vpaddb	1053	vpmacsdql
992	vmovdqu	1023	vpaddd	1054	vpmacssdql
993	vmovhpd	1024	vpaddq	1055	vpmacsswd
994	vmovhps	1025	vpaddsb	1056	vpmacssww
995	vmovlpd	1026	vpaddusb	1057	vpmacswd
996	vmovlps	1027	vpaddusw	1058	vpmaddwd
997	vmovmskpd	1028	vpaddw	1059	vpmaxsw
998	vmovmskps	1029	vpand	1060	vpmaxub
999	vmovntdq	1030	vpandn	1061	vpminsw
1000	vmovntpd	1031	vpavgb	1062	vpminub
1001	vmovntps	1032	vpavgw	1063	vpmovmskb
1002	vmovq	1033	vpcmpeqb	1064	vpmulhuw
1003	vmovsd	1034	vpcmpeqd	1065	vpmulhw
1004	vmovshdup	1035	vpcmpeqw	1066	vpmulld
1005	vmovsldup	1036	vpcmpgtb	1067	vpmullw
1006	vmovss	1037	vpcmpgtd	1068	vpmuludq
1007	vmovupd	1038	vpcmpgtw	1069	vpord
1008	vmovups	1039	vpcomud	1070	vpperm
1009	vmptird	1040	vpcomw	1071	vprotb
1010	vmptrst	1041	vpextrw	1072	vprotd
1011	vmread	1042	vphaddbd	1073	vpasdbw
1012	vmulpd	1043	vphaddbw	1074	vpshab
1013	vmulps	1044	vphaddubq	1075	vpshld
1014	vmulsd	1045	vphaddubw	1076	vpshlw
1015	vmulss	1046	vphadduwq	1077	vpshufb
1016	vmwrite	1047	vphaddwq	1078	vpshufhw
1017	vorpd	1048	vphsubbw	1079	vpshufw
1018	vorps	1049	vphsubdq	1080	vpshld

ID	Opcode	ID	Opcode	ID	Opcode
1081	vpsllq	1105	vrsqrtps	1129	xcrypt-cbc
1082	vpsllw	1106	vrsqrtss	1130	xcrypt-cfb
1083	vpsrad	1107	vshufpd	1131	xcrypt-ctr
1084	vpsraw	1108	vshufps	1132	xcrypt-ecb
1085	vpsrld	1109	vsqrtpd	1133	xcrypt-ofb
1086	vpsrlq	1110	vsqrtps	1134	xgetbv
1087	vpsrlw	1111	vsqrtsd	1135	xlat
1088	vpsubb	1112	vsqrtss	1136	xor
1089	vpsubq	1113	vsubpd	1137	xorb
1090	vpsubsb	1114	vsubps	1138	xorl
1091	vpsubsw	1115	vsubsd	1139	xorpd
1092	vpsubusb	1116	vsubss	1140	xorps
1093	vpsubusw	1117	vucomisd	1141	xorq
1094	vpsubw	1118	vucomiss	1142	xorw
1095	vpunpckhbw	1119	vunpckhps	1143	xrstor
1096	vpunpckhdq	1120	vunpcklpd	1144	xsave
1097	vpunpckhqdq	1121	vunpcklps	1145	xsaveopt
1098	vpunpckhwd	1122	vxorpd	1146	xsha1
1099	vpunpcklbw	1123	vxorps	1147	xsha256
1100	vpunpckldq	1124	vzeroupper		
1101	vpunpcklqdq	1125	wbinvd		
1102	vpunpcklwd	1126	wrmsr		
1103	vrcpps	1127	xadd		
1104	vrcpss	1128	xchg		



## APPENDIX C

### ANDROID OPCODE LIST

ID	Opcode	ID	Opcode	ID	Opcode
0	nop	14	const	28	goto
1	move	15	const/high16	29	goto/16
2	move/from16	16	const-wide/16	2A	goto/32
3	move/16	17	const-wide/32	2B	packed-switch
4	move-wide	18	const-wide	2C	sparse-switch
5	move-wide/from16	19	const-wide/high16	2D	cmpl-float
6	move-wide/16	1A	const-string	2E	cmpg-float
7	move-object	1B	const-string-jumbo	2F	cmpl-double
8	move-object/from16	1C	const-class	30	cmpg-double
9	move-object/16	1D	monitor-enter	31	cmp-long
A	move-result	1E	monitor-exit	32	if-eq
B	move-result-wide	1F	check-cast	33	if-ne
C	move-result-object	20	instance-of	34	if-lt
D	move-exception	21	array-length	35	if-ge
E	return-void	22	new-instance	36	if-gt
F	return	23	new-array	37	if-le
10	return-wide	24	filled-new-array	38	if-eqz
11	return-object	25	filled-new-array-range	39	if-nez
12	const/4	26	fill-array-data	3A	if-ltz
13	const/16	27	throw	3B	if-gez

ID	Opcode	ID	Opcode	ID	Opcode
3C	if-gtz	5B	iput-object	7A	unused_7A
3D	if-lez	5C	iput-boolean	7B	neg-int
3E	unused_3E	5D	iput-byte	7C	not-int
3F	unused_3F	5E	iput-char	7D	neg-long
40	unused_40	5F	iput-short	7E	not-long
41	unused_41	60	sget	7F	neg-float
42	unused_42	61	sget-wide	80	neg-double
43	unused_43	62	sget-object	81	int-to-long
44	aget	63	sget-boolean	82	int-to-float
45	aget-wide	64	sget-byte	83	int-to-double
46	aget-object	65	sget-char	84	long-to-int
47	aget-boolean	66	sget-short	85	long-to-float
48	aget-byte	67	sput	86	long-to-double
49	aget-char	68	sput-wide	87	float-to-int
4A	aget-short	69	sput-object	88	float-to-long
4B	aput	6A	sput-boolean	89	float-to-double
4C	aput-wide	6B	sput-byte	8A	double-to-int
4D	aput-object	6C	sput-char	8B	double-to-long
4E	aput-boolean	6D	sput-short	8C	double-to-float
4F	aput-byte	6E	invoke-virtual	8D	int-to-byte
50	aput-char	6F	invoke-super	8E	int-to-char
51	aput-short	70	invoke-direct	8F	int-to-short
52	iget	71	invoke-static	90	add-int
53	iget-wide	72	invoke-interface	91	sub-int
54	iget-object	73	unused_73	92	mul-int
55	iget-boolean	74	invoke-virtual/range	93	div-int
56	iget-byte	75	invoke-super/range	94	rem-int
57	iget-char	76	invoke-direct/range	95	and-int
58	iget-short	77	invoke-static/range	96	or-int
59	iput	78	invoke-interface-range	97	xor-int
5A	iput-wide	79	unused_79	98	shl-int

ID	Opcode	ID	Opcode	ID	Opcode
99	shr-int	B8	shl-int/2addr	D7	xor-int/lit16
9A	ushr-int	B9	shr-int/2addr	D8	add-int/lit8
9B	add-long	BA	ushr-int/2addr	D9	sub-int/lit8
9C	sub-long	BB	add-long/2addr	DA	mul-int/lit8
9D	mul-long	BC	sub-long/2addr	DB	div-int/lit8
9E	div-long	BD	mul-long/2addr	DC	rem-int/lit8
9F	rem-long	BE	div-long/2addr	DD	and-int/lit8
A0	and-long	BF	rem-long/2addr	DE	or-int/lit8
A1	or-long	C0	and-long/2addr	DF	xor-int/lit8
A2	xor-long	C1	or-long/2addr	E0	shl-int/lit8
A3	shl-long	C2	xor-long/2addr	E1	shr-int/lit8
A4	shr-long	C3	shl-long/2addr	E2	ushr-int/lit8
A5	ushr-long	C4	shr-long/2addr	E3	unused_E3
A6	add-float	C5	ushr-long/2addr	E4	unused_E4
A7	sub-float	C6	add-float/2addr	E5	unused_E5
A8	mul-float	C7	sub-float/2addr	E6	unused_E6
A9	div-float	C8	mul-float/2addr	E7	unused_E7
AA	rem-float	C9	div-float/2addr	E8	unused_E8
AB	add-double	CA	rem-float/2addr	E9	unused_E9
AC	sub-double	CB	add-double/2addr	EA	unused_EA
AD	mul-double	CC	sub-double/2addr	EB	unused_EB
AE	div-double	CD	mul-double/2addr	EC	unused_EC
AF	rem-double	CE	div-double/2addr	ED	unused_ED
B0	add-int/2addr	CF	rem-double/2addr	EE	execute-inline
B1	sub-int/2addr	D0	add-int/lit16	EF	unused_EF
B2	mul-int/2addr	D1	sub-int/lit16	F0	invoke-direct-empty
B3	div-int/2addr	D2	mul-int/lit16	F1	unused_F1
B4	rem-int/2addr	D3	div-int/lit16	F2	iget-quick
B5	and-int/2addr	D4	rem-int/lit16	F3	iget-wide-quick
B6	or-int/2addr	D5	and-int/lit16	F4	iget-object-quick
B7	xor-int/2addr	D6	or-int/lit16	F5	iput-quick

ID	Opcode	ID	Opcode	ID	Opcode
F6	iput-wide-quick	FA	invoke-super-quick	FE	unused_FE
F7	iput-object-quick	FB	invoke-super-quick/range	FF	unused_FF
F8	invoke-virtual-quick	FC	unused_FC		
F9	invoke-virtual-quick/range	FD	unused_FD		



## APPENDIX D

# BRIEF DESCRIPTION OF THE MAJOR CLASSIFIERS

Decision Trees are one of the machine learning techniques for the supervised classification [47]. It uses the tree structure, which has a root (starting point) and nodes (containing branches and leaves). The tree is built, starting from the root and grown by adding branches until the leaf nodes are reached. These branches are the segment, which connects root to leaves, and these branches (non-terminal nodes) represent the decision tests/conditions on one or more attributes to traverse the tree to reach leaf (terminal node), which are the outcome of the classifier. Here, finally, all leaf nodes represents a certain characteristic or outcome class and the branches represent a range of values. Below are the brief descriptions of the five major tree-based classifiers.

### D.1 J48

J48 is a C4.5 decision tree classification algorithm implemented in Java and is available in WEKA tool, and can be applied only for the numerical data [19]. It constructs a decision tree using the information entropy. In this at every node

of the tree, data points are divided into multiple subsets on the basis of one of the attributes in the data. The attribute is selected by evaluating its information gain. At every split the new child subsets will always have low entropy than its parent and this splitting continues until the entropy of the child subset becomes minimum or less than a threshold set by the programmer. This condition depends on the information gain of the attribute. This algorithm has the limitation of handling numeric data only.

## D.2 Random Forest

Random forest is an ensemble method decision tree classifier [44]. In this classifier individual decision trees are generated using a random vector sampled independently and with the same distribution for all trees in the forest. During the classification, each tree votes and the most popular class is returned. A Random forest can be built using bagging in tandem with random attribute selection. It generally exhibits a substantial performance improvement over the single tree classifier such as CART and C4.5 and the accuracy is comparable to AdaBoost, yet is more robust to errors and outliers. The accuracy of the Random forest depends on the strength of the individual classifiers and a measure of the dependencies between them. The generalization error for a forest converges as long as the number of trees in the forest is large. Thus, overfitting is not a problem [21].

## D.3 Naive Bayes Trees

The Naive Bayesian Tree (NBT) is a classifier, which combines the two classification methods (Decision Tree and Naive Bayes classifier). It uses decision trees of a certain height, and then in the bottom of tree leaves, Naive Bayes models are implemented [55]. The tree is grown in a top-down fashion by splitting the data points according to the information gain value. During the construction of the tree, at each node a decision is taken, whether the data at that node should be divided or not. If not divided then the node will be the leaf node and will be trained accordingly to the data at the node. Here, in the pruning phase at each node is done by comparing the accuracy of the Naive

Bayes model implemented at that node to the sum of the accuracy of its leaves. The split is done on any node if there are at least 30 data samples at the node and the error reduced by splitting is more than 5%. It has been seen in general that Naive Bayes trees often perform better than the decision trees.

## D.4 Logistic Model Trees

Logistic Model Tree classifier combines the logistic regression models and decision tree [58]. It basically consists of a standard decision tree structure with logistic regression functions at the leaves similar to the Naive Bayes classifier in NBT. Also, as in ordinary decision trees, a test on one of the attributes is associated with every inner node. For a nominal attribute of 'k' values, the node will have 'k' number of child, and depending on the attribute values the instance is sorted down to one of the branches. The test for the numerical data is done by comparing the attribute value to a set threshold, and if the attribute value is smaller than the set threshold, the instance is sorted down to the left branch else to the right branch.

## D.5 Functional Tree

The functional tree is a multivariate tree for regression and classification problems [39]. It can deal with binary and multi-class target variables, numeric and nominal attributes. The nodes in this tree are built while growing the tree and leaves are built during the pruning of the tree. Hence, in this model, it is possible to derive algorithms able to use functional decision nodes and functional leaf nodes for the classification.

## REFERENCES

- [1] The need for speed: 2013 incident response survey. Technical report, FireEye, 2013.
- [2] Red alert: Kaspersky lab reviews the malware situation in q3. Technical report, Kaspersky Lab, 2014.
- [3] Continued rise in mobile threats for 2016, Nov 2015.
- [4] Quick heal quarterly threat report q2 2015. Technical report, Quick Heal, February 2015.
- [5] Securelist: Mobile malware evolution. Technical report, Kaspersky Lab, 2015.
- [6] Threat report 3rd quarter, 2015, 2015.
- [7] Virustotal - free online virus, malware and url scanner, june 2016.
- [8] 9apps. Free android apps download, August 2016.
- [9] Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, and Moutaz Alazab. Zero-day malware detection based on supervised learning algorithms of api call signatures. In *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*, pages 171–182. Australian Computer Society, Inc., 2011.
- [10] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérôme, Jacques Klein, Yves Le Traon, et al. Large-scale machine learning-based malware detection: confronting the 10-fold cross validation scheme with reality. In *Proceedings of the 4th ACM conference on Data and application security and privacy*, pages 163–166. ACM, 2014.

- [11] Android-developers. Normal and dangerous permissions requesting permissions. Technical report, Android labs, 2017.
- [12] Ladkat Anita, Zure Dipali, and Mathew Lishoy. Annual threat report 2017. Technical report, Quick Heal, 2017.
- [13] Ladkat Anita, Zure Dipali, Mathew Lishoy, More Pranali, Moon Prashil, Dhasade Priyanka, Kadam Sagar, Khedkar Shraddha, Girme Tejas, Chaudhari Leena, Suddame Prachi, Temgire Sanket, Borse Sandip, and Pharate Swati. Quick heal quarterly threat report — q1 2017. Technical report, Quick Heal, 2017.
- [14] Seyed Emad Armoun and Sattar Hashemi. A general paradigm for normalizing metamorphic malwares. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 348–353. IEEE, 2012.
- [15] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, pages 1–15, 2014.
- [16] Thomas H Austin, Eric Filiol, Sebastien Josse, and Mark Stamp. Exploring hidden markov models for virus analysis: a semantic approach. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 5039–5048. IEEE, 2013.
- [17] Philippe Beaucamps. Advanced polymorphic techniques. *International Journal of Computer Science*, 2(3):194–205, 2007.
- [18] Boldizsár Bencsáth, Gábor Pék, Levente Buttyán, and Márk Félegyházi. Duqu: A stuxnet-like malware found in the wild. *CrySyS Lab Technical Report*, 14, 2011.
- [19] Neeraj Bhargava, Girja Sharma, Ritu Bhargava, and Manish Mathuria. Decision tree analysis on j48 algorithm for data mining. *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering*, 3(6), 2013.
- [20] Daniel Bilar. Opcodes as predictor for malware. *International Journal of Electronic Security and Digital Forensics*, 1(2):156–168, 2007.

- [21] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [22] Michael W Browne. Cross-validation methods. *Journal of mathematical psychology*, 44(1):108–132, 2000.
- [23] Gerardo Canfora, Francesco Mercaldo, and Corrado Aaron Visaggio. An hmm and structural entropy based detector for android malware: An empirical study. *Computers & Security*, 61:1–18, 2016.
- [24] Julio Canto, Marc Dacier, Engin Kirda, and Corrado Leita. Large scale malware collection: lessons learned. In *IEEE SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems*, 2008.
- [25] Scott Shaobing Chen and Ponani S Gopalakrishnan. Clustering via the bayesian information criterion with applications in speech recognition. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 2, pages 645–648. IEEE, 1998.
- [26] Beek Christiaan, Frosst Douglas, Greve Paula, Gund Yashashree, and Moreno Francisca. McAfee threats report. Technical report, McAfee, June 2012.
- [27] Beek Christiaan, Frosst Douglas, Greve Paula, Gund Yashashree, and Moreno Francisca. McAfee labs threats report. Technical report, McAfee, June 2014.
- [28] Beek Christiaan, Frosst Douglas, Greve Paula, Gund Yashashree, and Moreno Francisca. McAfee labs threats report. Technical report, McAfee, May 2015.
- [29] Beek Christiaan, Frosst Douglas, Greve Paula, Gund Yashashree, and Moreno Francisca. McAfee Labs Threats Report. Technical report, 2017.
- [30] Mihai Christodorescu, Johannes Kinder, Somesh Jha, Stefan Katzenbeisser, and Helmut Veith. Malware normalization. Technical report, University of Wisconsin, 2005.
- [31] Fred Cohen. Computer viruses: theory and experiments. *Computers & security*, 6(1):22–35, 1987.
- [32] Michael K Daly. Advanced persistent threat. *Usenix, Nov*, 4, 2009.

- [33] Sanjeev Das, Yang Liu, Wei Zhang, and Mahintham Chandramohan. Semantics-based online malware detection: Towards efficient real-time protection against malware. *IEEE Transactions on Information Forensics and Security*, 11(2):289–302, 2016.
- [34] Ddcreateur. Antivirus 2004, [database on the internet], March 2014.
- [35] F-Secure. Threat report. Technical report, F-Secure labs, 2013.
- [36] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Guillermo Suarez-Tangil, and Steven Furnell. Androdialysis: Analysis of android intent effectiveness in malware detection. *Computers & Security*, 65:121–134, 2017.
- [37] Adam P Fuchs, Avik Chaudhuri, and Jeffrey S Foster. Scandroid: Automated security certification of android. Technical report, University of Maryland Department of Computer Science, 2009.
- [38] Johannes Fürnkranz, Dragan Gamberger, and Nada Lavrač. Rule learning in a nutshell. In *Foundations of Rule Learning*, pages 19–55. Springer, 2012.
- [39] João Gama. Functional trees. *Machine Learning*, 55(3):219–250, 2004.
- [40] Nisarg Gandhewar and Rahila Sheikh. Google android: An emerging software platform for mobile devices. *International Journal on Computer Science and Engineering*, 1(1):12–17, 2010.
- [41] Aditya Govindaraju. Exhaustive statistical analysis for detection of metamorphic malware. 2010.
- [42] Kent Griffin, Scott Schneider, Xin Hu, and Tzi-Cker Chiueh. Automatic generation of string signatures for malware detection. In *Recent advances in intrusion detection*, pages 101–120. Springer, 2009.
- [43] N. Del Grosso. It’s time to rethink your corporate malware strategy. White paper, SANS Institute Reading Room site, 2002.
- [44] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

- [45] David Harley and Andrew Lee. Heuristic analysis—detecting unknown viruses. <http://www.eset.com/download/whitepapers/HeurAnalysis> (Mar2007) Online.pdf, 2007.
- [46] Olivier Henchiri and Nathalie Japkowicz. A feature selection and evaluation scheme for computer virus detection. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 891–895. IEEE, 2006.
- [47] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE, 1994.
- [48] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in computer Virology*, 4(3):251–266, 2008.
- [49] Jae-wook Jang, Hyunjae Kang, Jiyoung Woo, Aziz Mohaisen, and Huy Kang Kim. Andro-dumpsys: anti-malware system based on the similarity of malware creator and malware centric information. *computers & security*, 58:125–138, 2016.
- [50] Quentin Jerome, Kevin Allix, Radu State, and Thomas Engel. Using opcode-sequences to detect malicious android applications. In *2014 IEEE International Conference on Communications (ICC)*, pages 914–919. IEEE, 2014.
- [51] Alan Jović, Karla Brkić, and Nikola Bogunović. A review of feature selection methods with applications. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*, pages 1200–1205. IEEE, 2015.
- [52] B. Kang, S. Y. Yerima, K. Mclaughlin, and S. Sezer. N-opcode analysis for android malware classification and categorization. In *2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security)*, pages 1–7, June 2016.
- [53] Ankita Kapratwar. Static and dynamic analysis for android malware detection. Master’s thesis, San Jose State University, 2016.



- [54] Md Enamul Karim, Andrew Walenstein, Arun Lakhotia, and Laxmi Parida. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, 1(1-2):13–23, 2005.
- [55] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, volume 96, pages 202–207. Citeseer, 1996.
- [56] Jeremy Z Kolter and Marcus A Maloof. Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 470–478. ACM, 2004.
- [57] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. In *Machine Learning: ECML 2003*, pages 241–252. Springer, 2003.
- [58] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, 2005.
- [59] Neal Leavitt. Mobile phones: the next frontier for hackers? *Computer*, 38(4):20–23, 2005.
- [60] R. Lehtinen and G.T. Gangemi. *Computer Security Basics: Computer Security*. O’Reilly Media, 2006.
- [61] Chatchai Liangboonprakong and Ohm Sornil. Classification of malware families based on n-grams sequential pattern features. In *Industrial Electronics and Applications (ICIEA), 2013 8th IEEE Conference on*, pages 777–782. IEEE, 2013.
- [62] Kirti Mathur and Saroj Hiranwal. A survey on techniques in detection and analyzing malware executables. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(4):422–428, 2013.
- [63] Bilal Mehdi, Faraz Ahmed, Syed Ali Khayyam, and Muddassar Farooq. Towards a theory of generalizing system call representation for in-execution malware detection. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5. IEEE, 2010.
- [64] Syed Bilal Mehdi, Ajay Kumar Tanwani, and Muddassar Farooq. Imad: in-execution malware analysis and detection. In *Proceedings of the 11th Annual*

- conference on Genetic and evolutionary computation*, pages 1553–1560. ACM, 2009.
- [65] Tom M Mitchell. *Machine learning*. wcb, mcgraw-hill boston, ma, 1997.
- [66] Robert Moskovitch, Yuval Elovici, and Lior Rokach. Detection of unknown computer worms based on behavioral classification of the host. *Computational Statistics & Data Analysis*, 52(9):4544–4566, 2008.
- [67] Robert Moskovitch, Clint Feher, Nir Tzachar, Eugene Berger, Marina Gitelman, Shlomi Dolev, and Yuval Elovici. Unknown malware detection using opcode representation. In *Intelligence and Security Informatics*, pages 204–215. Springer, 2008.
- [68] Robert Moskovitch, Clint Feher, Nir Tzachar, Eugene Berger, Marina Gitelman, Shlomi Dolev, and Yuval Elovici. Unknown malware detection using opcode representation. In *Intelligence and Security Informatics*, pages 204–215. Springer, 2008.
- [69] Antonio Nappa, M Zubair Rafique, and Juan Caballero. Driving in the cloud: An analysis of drive-by download operations and abuse reporting. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 1–20. Springer, 2013.
- [70] Annamalai Narayanan, Liu Yang, Lihui Chen, and Liu Jinliang. Adaptive and scalable android malware detection through online learning. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 2484–2491. IEEE, 2016.
- [71] Smita Naval, Vijay Laxmi, Muttukrishnan Rajarajan, Manoj Singh Gaur, and Mauro Conti. Employing program semantics for malware detection. *IEEE Transactions on Information Forensics and Security*, 10(12):2591–2604, 2015.
- [72] K Olmstead and M Atkinson. Apps permissions in the google play store. Technical report, Pew Research Center, 2016.
- [73] Gabor Paller. Dalvik opcodes, 2017.

- [74] Babak Bashari Rad, Maslin Masrom, and Suhaimi Ibrahim. Evolution of computer virus concealment and anti-virus techniques: a short survey. *arXiv preprint arXiv:1104.1070*, 2011.
- [75] Babak Bashari Rad, Maslin Masrom, and Suhaimi Ibrahim. Camouflage in malware: from encryption to metamorphism. *International Journal of Computer Science and Network Security*, 12(8):74–83, 2012.
- [76] Bahman Rashidi, Carol Fung, and Elisa Bertino. Android resource usage risk assessment using hidden markov model and online learning. *Computers & Security*, 65:90–107, 2017.
- [77] Chandrasekar Ravi and R Manoharan. Malware detection using windows api sequence and machine learning. *International Journal of Computer Applications*, 43(17):12–16, 2012.
- [78] Richardson Robert. 14th annual csi/fbi computer crime and security survey-2009. Technical report, 2019.
- [79] Sanjay K Sahay and Ashu Sharma. Grouping the executables to detect malwares with high accuracy. *Procedia Computer Science*, 78:667–674, 2016.
- [80] Zahra Salehi, Ashkan Sami, and Mahboobe Ghiasi. Using feature generation from api calls for malware detection. *Computer Fraud & Security*, 2014(9):9–18, 2014.
- [81] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G Bringas. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences*, 231:64–82, 2013.
- [82] Igor Santos, Javier Nieves, and Pablo G Bringas. Semi-supervised learning for unknown malware detection. In *International Symposium on Distributed Computing and Artificial Intelligence*, pages 415–422. Springer, 2011.
- [83] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, and Pablo Garcia Bringas. On the automatic categorisation of android applications. In *2012 IEEE Consumer communications and networking conference (CCNC)*, pages 149–153. IEEE, 2012.

- [84] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli. Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1, 2017.
- [85] Matthew G Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J Stolfo. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49. IEEE, 2001.
- [86] Seung-Hyun Seo, Aditi Gupta, Asmaa Mohamed Sallam, Elisa Bertino, and Kangbin Yim. Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications*, 38:43–53, 2014.
- [87] Abhishek Shah. *Approximate Disassembly using Dynamic Programming*. PhD thesis, Citeseer, 2010.
- [88] Ashu Sharma and Sanjay Sahay, K. An investigation of the classifiers to detect android malicious apps. In *Information and Communication Technology, Proceedings of ICICT 2016*, volume 625. Springer (in press), 2017.
- [89] Ashu Sharma and Sanjay K Sahay. An effective approach for classification of advanced malware with high accuracy. *International Journal of Security and Its Applications*, 10(4):249–266, 2016.
- [90] Ashu Sharma and Sanjay K Sahay. Group-wise classification to improve the detection accuracy of android malicious apps. *International Journal of Network Security*, 2018 (in press).
- [91] Ashu Sharma, Sanjay K Sahay, and Abhishek Kumar. Improving the detection accuracy of unknown malware by partitioning the executables in groups. In *Advanced Computing and Communication Technologies*, pages 421–431. Springer, 2016.
- [92] Ashu Sharma and Sanjay Kumar Sahay. Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey. *International Journal of Computer Applications*, 90(2):7–11, March 2014.

- [93] Aimoto Shaun, AlKhatib Tareq, Coogan Peter, Corpin Mayee, and DiMaggio Jon. Internet security threat report. Technical report, Symantec Corporation, April 2012.
- [94] Aimoto Shaun, AlKhatib Tareq, Coogan Peter, Corpin Mayee, and DiMaggio Jon. Internet security threat report. Technical report, Symantec, April 2014.
- [95] Aimoto Shaun, AlKhatib Tareq, Coogan Peter, Corpin Mayee, and DiMaggio Jon. Internet security threat report. Technical report, Symantec Corporation, 2015.
- [96] Aimoto Shaun, AlKhatib Tareq, Coogan Peter, Corpin Mayee, and DiMaggio Jon. Internet security threat report 2016. Technical report, Symantec, 2016.
- [97] Aimoto Shaun, AlKhatib Tareq, Coogan Peter, Corpin Mayee, and DiMaggio Jon. Internet security threat report 2016. Technical report, Symantec Corporation, 2016.
- [98] Aimoto Shaun, AlKhatib Tareq, Coogan Peter, Corpin Mayee, and DiMaggio Jon. Internet security threat report. Technical report, Symantec Corporation, 2017.
- [99] Aimoto Shaun, AlKhatib Tareq, Coogan Peter, Corpin Mayee, and DiMaggio Jon. Internet security threat report 2017. Technical report, Symantec, 2017.
- [100] Muazzam Siddiqui, Morgan C Wang, and Joochan Lee. Detecting internet worms using data mining techniques. *Journal of Systemics, Cybernetics and Informatics*, 6(6):48–53, 2008.
- [101] Muazzam Siddiqui, Morgan C Wang, and Joochan Lee. A survey of data mining techniques for malware detection using file features. In *Proceedings of the 46th annual southeast regional conference on xx*, pages 509–510. ACM, 2008.
- [102] Michael Spreitzenbarth, Felix Freiling, Florian Echtler, Thomas Schreck, and Johannes Hoffmann. Mobile-sandbox: having a deeper look into android applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1808–1815. ACM, 2013.

- [103] William Stallings. *Network security essentials: applications and standards*. Pearson Education India, 2007.
- [104] Statista. Number of available applications in the google play store from december 2009 to february 2016, August 2016.
- [105] Richard Stone. A call to cyber arms. *Science*, 339(6123):1026–1027, 2013.
- [106] Peter Szor. *The art of computer virus research and defense*. Pearson Education, 2005.
- [107] Peter Szor and Peter Ferrie. Hunting for metamorphic. In *Virus bulletin conference*, pages 123–144, 2001.
- [108] S Momina Tabish, M Zubair Shafiq, and Muddassar Farooq. Malware detection using statistical analysis of byte-level file content. In *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, pages 23–31. ACM, 2009.
- [109] Ronghua Tian, Lynn Margaret Batten, and SC Versteeg. Function length as a tool for malware classification. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, pages 69–76. IEEE, 2008.
- [110] Annie H Toderici and Mark Stamp. Chi-squared distance and metamorphic virus detection. *Journal of Computer Virology and Hacking Techniques*, 9(1):1–14, 2013.
- [111] Nhat-Phuong Tran and Myungho Lee. High performance string matching for security applications. In *ICT for Smart Society (ICISS), 2013 International Conference on*, pages 1–5. IEEE, 2013.
- [112] Ashwini Venkatesan. *Code Obfuscation and Virus Detection*. PhD thesis, San Jose State University, 2008.
- [113] Timothy Vidas, Nicolas Christin, and Lorrie Cranor. Curbing android permission creep. In *Proceedings of the Web*, volume 2, pages 91–96, 2011.
- [114] webmaster@vxheaven.org. Viruses don't harm, ignorance does. <http://vx.netlux.org>, 2017.

- [115] R Winsniewski. Android–apktool: A tool for reverse engineering android apk files, 2012.
- [116] Wing Wong and Mark Stamp. Hunting for metamorphic engines. *Journal in Computer Virology*, 2(3):211–229, 2006.
- [117] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, pages 62–69. IEEE, 2012.
- [118] J-Y Xu, Andrew H Sung, Patrick Chavez, and Srinivas Mukkamala. Polymorphic malicious executable scanner by api sequence analysis. In *Hybrid Intelligent Systems, 2004. HIS’04. Fourth International Conference on*, pages 378–383. IEEE, 2004.
- [119] Ke Xu, Yingjiu Li, and Robert H Deng. Iccdetector: Icc-based malware detection on android. *IEEE Transactions on Information Forensics and Security*, 11(6):1252–1264, 2016.
- [120] Ming Xu, Lingfei Wu, Shuhui Qi, Jian Xu, Haiping Zhang, Yizhi Ren, and Ning Zheng. A similarity metric method of obfuscated malware using function-call graph. *Journal of Computer Virology and Hacking Techniques*, 9(1):35–47, 2013.
- [121] Yanfang Ye, Dingding Wang, Tao Li, Dongyi Ye, and Qingshan Jiang. An intelligent pe-malware detection system based on association mining. *Journal in computer virology*, 4(4):323–334, 2008.
- [122] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE, 2010.
- [123] Min Zheng, Mingshen Sun, and John CS Lui. Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 163–171. IEEE, 2013.

## LIST OF PUBLICATIONS

### INTERNATIONAL JOURNALS

1. Ashu Sharma, Sanjay K. Sahay, *Group-wise Classification to Improve the Detection Accuracy of Android Malicious Apps*, International Journal of Network Security, 2018 (in press).
2. Ashu Sharma, Sanjay K. Sahay, *An Effective Approach for Classification of Advanced Malware with High Accuracy*, International Journal of Security and Its Applications, Vol. 10, No. 4, pp. 249-266, 2016.
3. Ashu Sharma, Sanjay K. Sahay, *Evolution and Detection of Polymorphic and Metamorphic malware: A Survey*, International Journal of Computer Applications, Vol., 90, No. 2, pp. 7-11, 2014.

### INTERNATIONAL CONFERENCES

1. Ashu Sharma, Sanjay K. Sahay, *Investigation of the classifiers to detect android malicious apps*, Springer, Information and Communication Technology, pp. 207-217, 2017, Proceedings ICICT-2016.
2. Sanjay K. Sahay, Ashu Sharma, *Grouping the Executables to Detect malware with High Accuracy*, Elsevier, Procedia Computer Science, Vol. 78, pp. 667-674, 2016, Proceedings ICISP-2015.
3. Ashu Sharma, Sanjay K. Sahay, Abhishek Kumar, *Improving the detection accuracy of unknown malware by partitioning the executables in groups*, Springer, Advanced Computing and Communication Technologies, pp. 421-431, 2016, Proceedings 9th ICACCT-2015.



## **BRIEF BIOGRAPHY OF THE CANDIDATE**

Ashu Sharma was born in Jhansi, Uttar Pradesh, India. He received his Bachelor's degree in Computer Science and Engineering from Uttar Pradesh Technical University and Master's degree in Information Security from Atal Bihari Vajpayee Indian Institute of Information Technology and Management, Gwalior. In 2012 he joined the Department of Computer Science and Information Systems, BITS, Pilani, K.K. Birla Goa Campus, India as a full-time research scholar for the Ph.D. degree under the supervision of Dr. Sanjay K. Sahay.

## **BRIEF BIOGRAPHY OF THE SUPERVISOR**

Dr. Sanjay Kumar Sahay is working as an Assistant Professor in the Department of Computer Science and Information Systems, BITS, Pilani, K.K. Birla Goa Campus. His research interests are Data Science, Information Security, and Gravitational Waves. After submitting his Ph.D. thesis on “Studies in Gravitational Wave Data Analysis” during 2002-2003, he continued his work on Data Analysis of Gravitational Waves as a Project Scientist at Inter-University Centre for Astronomy and Astrophysics, Pune, India. In 2003-2005 at Raman Research Institute, Bangalore, India he worked as Project Associate on the multi-wavelength astronomy project (ASTROSAT). In 2005 he worked as Post Doctoral Fellow at Tel Aviv University, Israel, and since 2006, he is working as Assistant Professor in the Department of Computer Science and Information Systems, BITS, Pilani, K.K. Birla Goa Campus, India. He has published many papers in reputed journals and conferences and supervised Ph.D. student and many Postgraduate and Undergraduate projects.