

MALWARE DETECTION IN ANDROID USING
MACHINE LEARNING

MUHAMMAD HAZRIQ AKMAL BIN ZAIROL

BACHELOR'S OF COMPUTER SCIENCE
(COMPUTER SYSTEM & NETWORKING)
WITH HONOURS

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : MUHAMMAD HAZRIQ AKMAL BIN ZAIROL
Date of Birth :
Title : MALWARE DETECTION IN ANDROID USING
MACHINE LEARNING
Academic Session : SEMESTER 1 2022/2023

I declare that this thesis is classified as:

- CONFIDENTIAL (Contains confidential information under the Official Secret Act 1997)*
- RESTRICTED (Contains restricted information as specified by the organization where research was done)*
- OPEN ACCESS I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

(Student's Signature)

(Supervisor's Signature)

_ New IC/Passport
Number Date: 7/11/2022

□□ M□□□ □□□□□□ □□ □□ □□□□□

Name of Supervisor
Date: □□□□□□□□

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

THESIS DECLARATION LETTER (OPTIONAL)

Librarian,
Perpustakaan Universiti Malaysia Pahang,
Universiti Malaysia Pahang,
Lebuhraya Tun Razak,
26300, Gambang, Kuantan.

Dear Sir,

CLASSIFICATION OF THESIS AS RESTRICTED

Please be informed that the following thesis is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.

Author's Name
Thesis Title

- | | |
|---------|-------|
| Reasons | (i) |
| | (ii) |
| | (iii) |

Thank you.

Yours faithfully,

(Supervisor's Signature)

Date:

Stamp:

Note: This letter should be written by the supervisor, addressed to the Librarian, *Perpustakaan Universiti Malaysia Pahang* with its copy attached to the thesis.



SUPERVISOR’S DECLARATION

I/We* hereby declare that I/We* have checked this thesis/project* and in my/our* opinion, this thesis/project* is adequate in terms of scope and quality for the award of the degree of *Doctor of Philosophy/ Master of Engineering/ Master of Science in

(Supervisor’s Signature)

Full Name : DR MOHD FAIZAL BIN AB RAZAK

Position : SENIOR LECTURER

Date :

(Co-supervisor’s Signature)

Full Name :

Position :

Date :



STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

(Student's Signature)

Full Name : MUHAMMAD HAZRIQ AKMAL BIN ZAIROL

ID Number : CA20144

Date : 7 □□□2022

MALWARE DETECTION IN ANDROID USING MACHINE LEARNING

MUHAMMAD HAZRIQ AKMAL BIN ZAIROL

Thesis submitted in fulfillment of the requirements
for the award of the degree of
BACHELOR'S OF COMPUTER SCIENCE (COMPUTER SYSTEM &
NETWORKING) WITH HONOURS

Faculty of Computing
UNIVERSITI MALAYSIA PAHANG

□□□□ 202□

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to all those who have supported me throughout my final year project.

First and foremost, I would like to thank Allah for making it possible for me to finish this research without any problems. And I would like to thank my project supervisor Ts. Dr. Mohd Faizal bin Ab Razak, for the guidance, support, and encouragement throughout the project. His expertise and knowledge in the field of malware detection have been invaluable, and I am grateful for the time he dedicated to help me understand the technical aspects of the project.

I would also like to thank the rest of the Faculty of Computing members, for providing me with the resources and support I needed to complete my project. I am also grateful to my family, loved one, and friends for their unwavering support and encouragement throughout the completion of my research. Their understanding and patience have been a source of strength and motivation for me.

Finally, I would like to acknowledge the contributions of the open-source community, whose resources and tools have been essential in the development of this research. Thank you all for your invaluable support and guidance throughout this project.

ABSTRAK

Di era yang kian pesat dengan teknologi canggih, telefon pintar menjadi keutamaan dan keperluan bagi semua orang. Gajet ini berkembang setiap hari kearah yang lebih maju dan sesuai dengan cara penggunaannya. Namun, keselamatan menjadi salah satu punca yang menjadi kerisauan ramai pengguna telefon pintar ini. Keselamatan adalah aspek penting yang dipandang tinggi dan diambil berat oleh sesetengah pihak, dan sekiranya isu keselamatan ini dipandang remeh dan tidak diambil peduli, ia akan menyebabkan masalah kepada orang sekeliling. Sama seperti isu keselamatan pengguna telefon pintar, yang kini semakin berleluasa dengan salah satu ancaman terbesar bagi semua gajet, iaitu isu perisian perosak. Kajian telah menunjukkan bahawa terdapat peningkatan dari tahun ke tahun mengenai perisian perosak yang lebih tertumpu kepada menyerang dan merosakkan telefon pintar mangsa terutamanya kepada pengguna Android. Ramai pengguna Android telah terjejas dengan masalah perisian perosak ini dan juga pelbagai solusi sudah dijalankan. Kajian ini bertujuan untuk mengkaji cara dan kaedah pengesanan perisian perosak yang telah menyerang system operasi Android, dan mencadangkan pengesanan system pengesanan perisian perosak dengan menggunakan teknik pembelajaran mesin. Keputusan menunjukkan bahawa pembelajaran mesin adalah pendekatan yang lebih menjanjikan ketepatan 90% dalam eksperimen yang telah dijalankan bagi kaedah pembelajaran mesin untuk pengesanan perisian perosak yang lebih tinggi dan membuktikan bahawa sistem pengesanan perisian perosak ini dapat mengesan perisian perosak Android dengan lebih efisien.

ABSTRACT

In an era that is increasingly fast with advanced technology, smartphones are a priority and a necessity for everyone. These gadgets are developing every day towards more advanced and appropriate ways of use. However, security is one of the causes of concern for many smartphone users. Safety is an important aspect that is highly regarded and taken seriously by some parties, and if this safety issue is taken for granted and not taken care of, it will cause problems to the people surrounding. Just like the security issue of smartphone users, which is now increasingly prevalent with one of the biggest threats to all gadgets, which is the malware issue. Studies have shown that there is an increase from year to year regarding malware that is more focused on attacking and damaging the victim's smartphone, especially for Android users. Many Android users have been affected by this malware problem and various solutions have been implemented. This study aims to examine the ways and methods of detecting malware that has attacked the Android operating system, and suggest the detection of a malware detection system by using machine learning techniques. The results show that machine learning is a more promising approach with 90% accuracy in experiments that have been conducted for machine learning methods for higher malware detection and prove that this malware detection system can detect Android malware more efficiently.

TABLE OF CONTENT

DECLARATION	
TITLE PAGE	
ACKNOWLEDGEMENTS	ii
ABSTRAK	iii
ABSTRACT	iv
TABLE OF CONTENT	v
LIST OF TABLES	ix
LIST OF FIGURES	x
LIST OF SYMBOLS	xi
LIST OF ABBREVIATIONS	xii
LIST OF APPENDICES	xiii
CHAPTER 1 INTRODUCTION	14
1.1 Introduction	14
1.2 Problem Statement	16
1.3 Objectives	17
1.4 Scope	17
1.5 Thesis Organization	18
CHAPTER 2 LITERATURE REVIEW	20
2.1 Introduction	20
2.2 Malware	20
2.3 Types of malware attack	23
2.3.1 Adware and Backdoor	25
2.3.2 File infector and PUA	26
2.3.3 Ransomware and Riskware	27

2.3.4	Scareware, Spyware, and Trojan	28
2.4	Android Malware Detection Approaches	29
2.4.1	Signature-based Detection Approaches	30
2.4.2	Anomaly-based Detection Approaches	31
2.4.3	Specification-based Detection Approaches	32
2.4.4	Comparison of Android Malware Detection Approaches	33
2.5	Analysis Technique	34
2.5.1	Static Analysis	34
2.5.2	Dynamic Analysis	35
2.5.3	Hybrid Analysis	37
2.5.4	Comparison of Analysis Techniques	38
2.6	Machine Learning	38
2.6.1	Supervised machine learning	39
2.6.2	Unsupervised machine learning	40
2.6.3	Semi-supervised learning	40
2.6.4	Reinforcement learning	40
2.7	Previous Research Works	41
2.7.1	Comparison and description of previous research paper	42
2.8	Conclusion	44
CHAPTER 3 METHODOLOGY		45
3.1	Introduction	45
3.2	Research-Based	45
3.3	Planning and Reviewing Literature	47
3.4	Developing the Architecture	48
3.4.1	Procedure Description	50

3.4.2	Data Collection Phase	51
3.4.3	Decompiling the APK File	51
3.4.4	Features Selection	55
3.4.5	Machine Learning Classifiers	56
3.4.6	Machine Learning Tool	58
3.5	Design and Implementation	63
3.6	Hardware and Software	66
3.6.1	Hardware Requirement	66
3.6.2	Software Requirement	67
3.7	Testing and Evaluation	68
3.8	Conclusion	69
CHAPTER 4 RESULTS AND DISCUSSION		70
4.1	Introduction	70
4.2	Dataset Description	70
4.2.1	Used Dataset	70
4.2.2	Data Cleaning	72
4.2.3	Splitting Dataset	74
4.3	Machine Learning Approach	75
4.4	Evaluation and Results	79
4.4.1	Confusion matrix	83
4.4.2	Receiver operating characteristics curve (ROC)	86
4.4.3	Performance of Classifiers Build model	90
CHAPTER 5		92
CONCLUSION		92

5.1	Introduction	92
5.2	Research Objective Revisited	93
5.3	Achievement of the study	96
5.3.1	Choosing a relevant dataset to be used	96
5.3.2	Understand the detection approaches and techniques	97
5.3.3	Find the best machine learning classifiers	97
5.4	Limitations	98
5.4.1	Availability and Quality of Datasets	98
5.4.2	Imbalanced Sample Size	98
5.4.3	Interpretability and Explain ability of Machine Learning	98
5.5	Future Works	99
5.5.1	Incorporating New Malware Techniques	99
5.5.2	Deep Learning Approaches	99
5.5.3	Real-Time Detection on Mobile Devices	99
5.5.4	Adversarial Attack Detection	99
5.6	Conclusion	100
	REFERENCES	101

LIST OF TABLES

<i>Table 2.1: Type of Malware</i>	24
<i>Table 2.2: Comparison of malware detection approaches</i>	33
<i>Table 2.3: Comparison between analysis techniques</i>	38
<i>Table 3.1: Dataset Summary</i>	51
<i>Table 3.2: Top ten permission in Benign and Malware Applications</i>	53
<i>Table 3.3: Hardware requirement and description for this research</i>	66
<i>Table 3.4: Software requirement and description for this research</i>	67
<i>Table 4.1: Total AndroZoo Dataset</i>	70
<i>Table 4.2: List of permission features</i>	79
<i>Table 4.3: Results for each machine learning classifier</i>	79
<i>Table 4.4: Confusion matrix of classifiers</i>	85
<i>Table 4.5: AUC Classifiers Result</i>	87
<i>Table 4.6: Time taken to produce model (seconds)</i>	90
<i>Table 4.7: Time taken to test model (seconds)</i>	91

LIST OF FIGURES

<i>Figure 1.1: Overall Chapter</i>	18
<i>Figure 2.1: Total of Malware detected between years</i>	21
<i>Figure 2.2: Malware detected by categories</i>	22
<i>Figure 2.3: Malware Detection Approach</i>	29
<i>Figure 3.1: Main Phases of Research-Based</i>	46
<i>Figure 3.2: Malware Detection System Architecture</i>	48
<i>Figure 3.3: Data collection phase</i>	52
<i>Figure 3.4: Number of applications requesting Benign and Malware</i>	54
<i>Figure 3.5: Jupyter Notebook Tools</i>	59
<i>Figure 3.6: The main interface of the Jupyter Notebook</i>	61
<i>Figure 3.7: Option to start a new notebook</i>	62
<i>Figure 3.8: Running terminal in Jupyter Notebook</i>	62
<i>Figure 3.9: Jupyter Notebook cell</i>	63
<i>Figure 3.10: Flowchart of the procedure for improving detection method testing</i>	64
<i>Figure 4.1: Importing the Libraries in Jupyter Notebook</i>	71
<i>Figure 4.2: Reading the datasets</i>	71
<i>Figure 4.3: Display the datasets</i>	72
<i>Figure 4.4: Count the missing values in dataset</i>	72
<i>Figure 4.5: Data cleaning for checking the missing values</i>	73
<i>Figure 4.6: Separated the dataset into X and y</i>	74
<i>Figure 4.7: Splitting the dataset into Training and Testing</i>	74
<i>Figure 4.8: Accuracy testing for Naïve Bayes</i>	80
<i>Figure 4.9: Accuracy testing for DecisionTable</i>	80
<i>Figure 4.10: Accuracy testing for J48</i>	81
<i>Figure 4.11: Accuracy testing for Random Forest</i>	81
<i>Figure 4.12: Accuracy testing for Multi-Layer Perceptron</i>	82
<i>Figure 4.13: Level of accuracy for each classifier</i>	83
<i>Figure 4.14: Naïve Bayes ROC Curves</i>	87
<i>Figure 4.15: DecisionTable ROC Curves</i>	88
<i>Figure 4.16: J48 ROC Curves</i>	88
<i>Figure 4.17: Random Forest ROC Curves</i>	89
<i>Figure 4.18: Multi-Layer Perceptron ROC Curves</i>	89

LIST OF SYMBOLS

LIST OF ABBREVIATIONS

MLP	Multi-Layer Perceptron
RF	Random Forests

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Malware refers to any program designed to disrupt computer operations, steal data, or break into secure networks. Software that causes unintentional harm due to a deficiency is not considered malware since malware is defined by its malevolent purpose, which works against the requirements of the computer user. Aside from purposefully malicious software, the term "badware" is sometimes used to describe software that harms by accident. threaten the availability of the internet, the security of its hosts, and the privacy of its users by breaking into computer systems and network resources without the owner's permission, disrupting computer operations, and collecting personal information. Malware has had far-reaching effects, affecting everything from e-governance and social media to digital automation and mobile networks.

Each type of malware such as virus, worm, Trojan horse, rootkit, backdoor, botnet, spyware, or adware has its unique purpose and method of operation. Malware can exhibit traits from numerous categories at once since these categories are not exclusive to one another. Malware creators implement polymorphism into the harmful components as a means of evading detection. This implies that dangerous files within the same malware "family," exhibiting the same types of destructive behavior, are regularly updated and obfuscated using different techniques, making them appear to be a wide variety of distinct files.

When it comes to Internet security, malware is one of the biggest and worst risks currently available. A study performed by Symantec in February 2019, the poll found that 47% of firms had encountered malware security incidents/network breaches in the previous year.[1] Malware is expanding in three dimensions: quantity (widening scope of threats), diversity (evolving techniques of harm), and speed (fluidity of threats). These

are developing, gaining sophistication, and deploying novel techniques to attack computers and mobile devices. Over 100,000 new malware samples are added to McAfee's database every day.[2] This equates to roughly 69 new threats per minute, or one threat every second. There has been a rise in both the availability and sophistication of cybercriminal tools, leading to a new generation of threats and assaults that are more complex, persistent, and mysterious.

Since the number of Android devices and software applications known as apps (app stores) is continually growing, a large number of Android users have benefited from this. Concerns about safety and personal privacy are also gaining traction among a wide range of mobile users and stakeholders. For example, an increasing number of users are opting to keep their personal information on their mobile devices by using popular apps such as those for shopping, banking, and social networking. As a result, attackers have moved their emphasis to mobile applications during the last decade. As a result, malicious software for Android smartphones has emerged as one of the most serious security problems in the industry. That is, they believe the sources from which they obtain their programs are trustworthy and secure. Several approaches for detecting Android malware, including those based on signatures, behaviour, and data-flow analysis, have been developed.

The machine learning-based strategy is one of the most promising ways to detect out as one of the most promising ways for detecting Android malware. Because of the availability of vast data and the progress of hardware over the preceding decade, machine learning has proved incredibly effective in various cutting-edge fields, including Android virus detection. In practise, all of the aforementioned security precautions are primarily applied in app store backends. However, not all app shops are able to respond quickly when a new family of Android malware is discovered. The analysis approach now consists of three independent steps: investigating dangerous behaviour within applications, building detection models with the generated characteristics, and lastly running a detection on the whole app.

1.2 Problem Statement

With the rise of the new modern gadget era, people nowadays are having mobile phones for their daily use. According to Tech Crunch, mobility firm Ericsson predicts that by 2020, there will be more than six billion smartphone users worldwide, surpassing landlines. Because smartphones and tablets are quickly becoming more powerful as companies embrace the idea of bring your own device (BYOD) policies and allow users to access corporate networks with personal technology. However, with increased use emerges an increase in mobile malware, which is malicious code designed to target smartphones and tablets.

"Android malware is growing at an exponential rate, but I fear we won't see any major changes in user behaviour until a large and significant user base is affected by malware," Tim Armstrong, malware researcher for anti-virus firm Kaspersky Lab, told SecurityNewsDaily. Armstrong and Wisniewski were both commenting on a new Juniper Networks report showing that Android malware has increased 472 percent in the four months since July 2011. Despite this astounding increase in malicious, corrupt software, Android has captured 52.5 percent of the global smartphone market share, with more than 440.5 million units sold in the third quarter (July through September) alone, according to technology research firm Gartner.[3] As a result, Android malware is clearly a problem. However, as with car theft, it is not a serious issue until your vehicle is stolen.

In addition, UMP students keep browsing on their mobile phones to do simple research on what they learn in class. And the internet platform is the place where they may get android malware by clicking on a random advertisement and link. This may lead to downloading any files and leaking malicious android software on their mobile devices. Although, Zimperium Labs discovered earlier this year that 95 percent of Android devices could be hacked with a simple text message. Cyber criminals have come out with news-identified ways, methods, and tricks to launch an attack on android users. As opposed to classic malware, which was widespread, well-documented, and static, modern malware is targeted, unknown, stealthy, customized, and zero-day. After infiltrating a system, viruses and malware conceal, reproduce, and compromise security. This can put a risk for all android phone users.

1.3 Objectives

The objectives of this research are:

- i. To study the current issue with the Android malware detection system.
- ii. To analyze the technique for Machine Learning that will be used to construct an Android malware detection system.
- iii. To assess the effectiveness of the Android malware detection system in terms of its ability to identify malicious software.

1.4 Scope

The scope of this research:

- i. Platform:
 - This system is for Android packages only.
- ii. Development / Functionality:
 - The system is only able to identify malicious software and cannot completely remove it from infected devices.
 - The detection method is only effective on a mobile device that is powered by the Android operating system.
- iii. User:
 - Users of Android-based smartphones only.

1.5 Thesis Organization

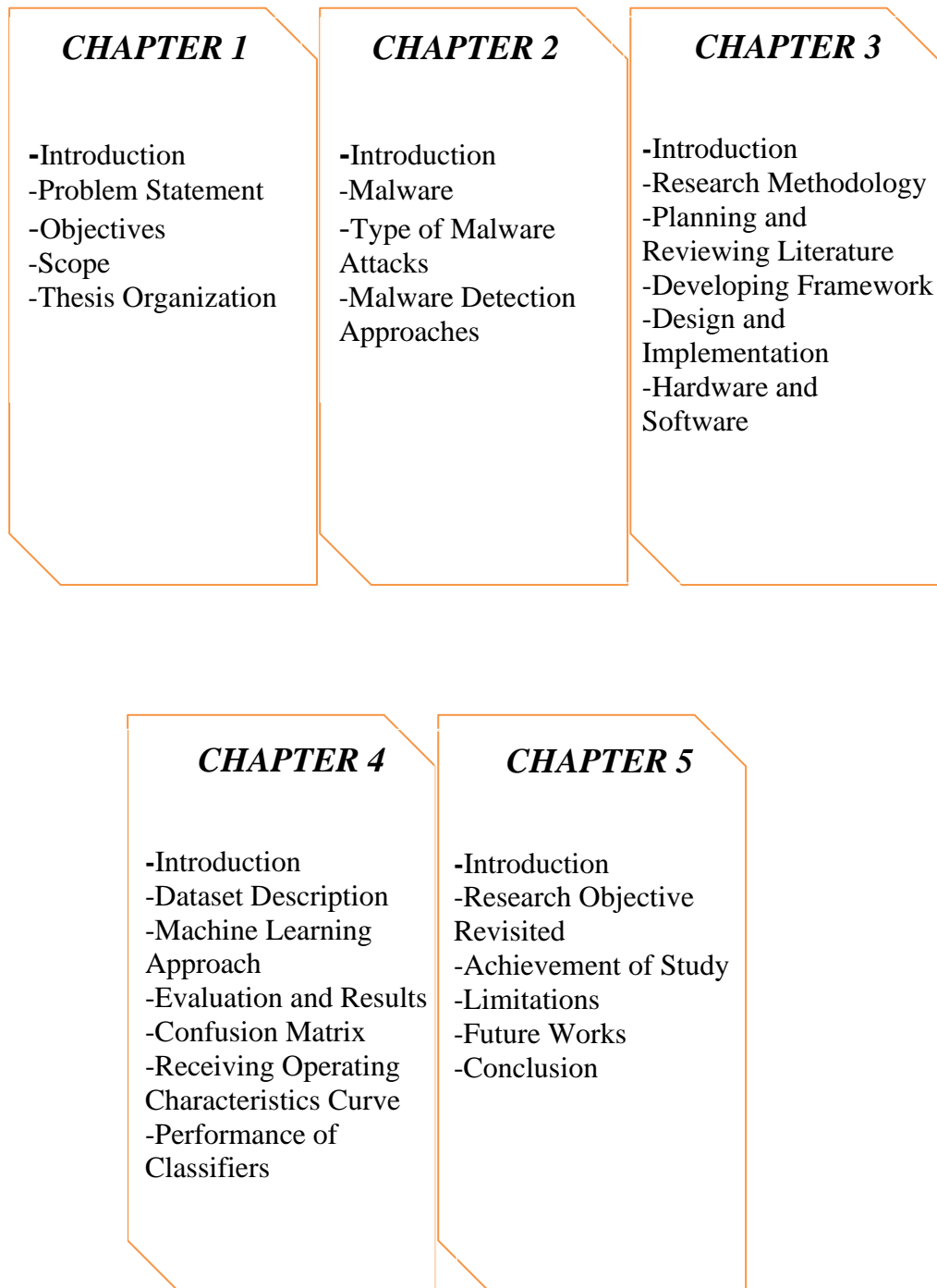


Figure 1.1: Overall Chapter

In chapter 1, there are an introduction, problem statement, objectives, scope, and thesis organization. Furthermore, chapter 2 discusses more literature reviews of the research topic such as what is the definition of malware, the background of android malware detection, and the current approach solution comparisons to earlier research on the research topic. Besides, chapter 3 highlights the methodology that is utilized throughout this study on the approach that was used. The gathering of data, the standardization of such data, and the software that was utilized in this experiment are all topics that are covered in this section of the study. Furthermore, Chapter 4 provides a detailed exploration for Training and Testing of the machine learning approach for malware detection in Android to achieve a highest results accuracy. It presents the dataset, discusses the chosen algorithms, evaluates their performance using various metrics, and visualizes the results through the confusion matrix and ROC curve. Lastly, Chapter 5 of the project serves as a reflective and conclusive section. It reviews the research objective, evaluates the achievement of the study, examines its limitations, recommends future improvements, and wraps up the project. The results of the study are thoroughly summarised in this chapter, which also paves the way for future developments in machine learning-based malware detection for Android.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter will be more focusing on the literature review which can describe more about malware detection, especially android malware. Otherwise, the malware detection approaches, techniques, and features. This chapter also determines how malware can be found in infected devices. Furthermore, will be studying the previous research's comparison with the related review on related research that suggested different techniques to detect the malware in order to further enhance the current work.

2.2 Malware

Malware is a catch-all term for computer viruses, trojan horses, and other destructive computer programs that threat actors use to infect systems and networks in order to gain access to sensitive information. Malware is defined as software that is designed to interfere with the normal functioning of a computer. Malware is an abbreviation for "malicious software," which refers to a file or piece of code that, when transmitted typically across a network, can infect, investigate, steal, or conduct nearly any other behaviour that an attacker desire.[3]

Hostile, intrusive, and intentionally nasty, malware seeks to invade, damage, or disable computers, computer systems, networks, tablets, and mobile devices, often by taking partial control over a device's operations. Because malware comes in so many varieties, there are countless ways to infect computers. Malware, while varying in type and capability, typically has a goal such as providing a remote control for an attacker to use an infected machine, sending spam from the infected machine to unsuspecting targets,

investigating the infected user's local network, and, most importantly, stealing sensitive data.

Android malware is malicious software that targets Android-powered smartphone devices. It functions similarly to other malware variants that run on desktop or laptop PCs. Android malware, often known as mobile malware, is any malicious software designed to harm a mobile device by doing unauthorized activities such as installing infected apps from unofficial app stores, visiting hacked websites, or receiving infected email attachments. Malware is categorized into several types, including adware, backdoors, file infectors, potentially unwanted applications (PUA), ransomware, riskware, scareware, spyware, and trojans. Each malware type has some distinguishing traits that set it apart from the others. Android malware develops in the same way that humans do.

Users have access to high-performance platforms thanks to Android, which is the industry-leading operating system. Android is expected to maintain its dominant position in the industry with an 85 percent share of the global market in the final quarter of 2020, as stated in a report that was issued by the International Data Corporation (IDC). In addition, it is anticipated that the yearly shipment rate of Android would expand by 150 million units in the year 2021. In conjunction with the skyrocketing demand for Android in the global market, the difficulties that are related to malicious software for Android are also increasing at a breakneck speed. Its solutions identified over 3,5 million malicious installation packages in 2021, which is roughly the same amount as in 2019, but 2.2 million fewer than in 2020. In 2021, the number of malicious installation packages was about the same as in 2019.

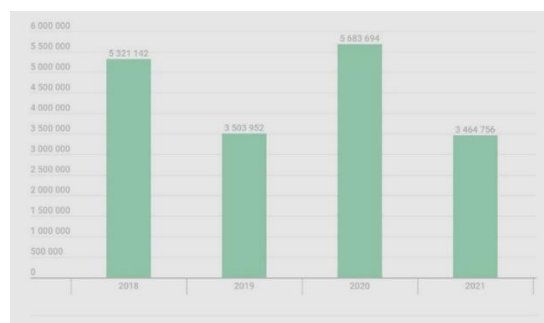


Figure 2.1: Total of Malware detected between years

The number of attacks that were identified continued to fall throughout 2021, going from 5.5 million in January 2021 to 2.2 million by the end of the year. However, according to Kaspersky, attacks on mobile have become more complex in terms of both the functionality of the virus and the vectors that are being used. Adware and potentially unwanted applications (PUA) typically use business models that facilitate spreading at a large scale on as many devices as possible. These findings should not come as a surprise given that adware and PUA tend to use business models that facilitate spreading at a large scale on as many devices as possible with adware representing as much as 42% of all detected mobile malware and PUA representing 35% of all detections.

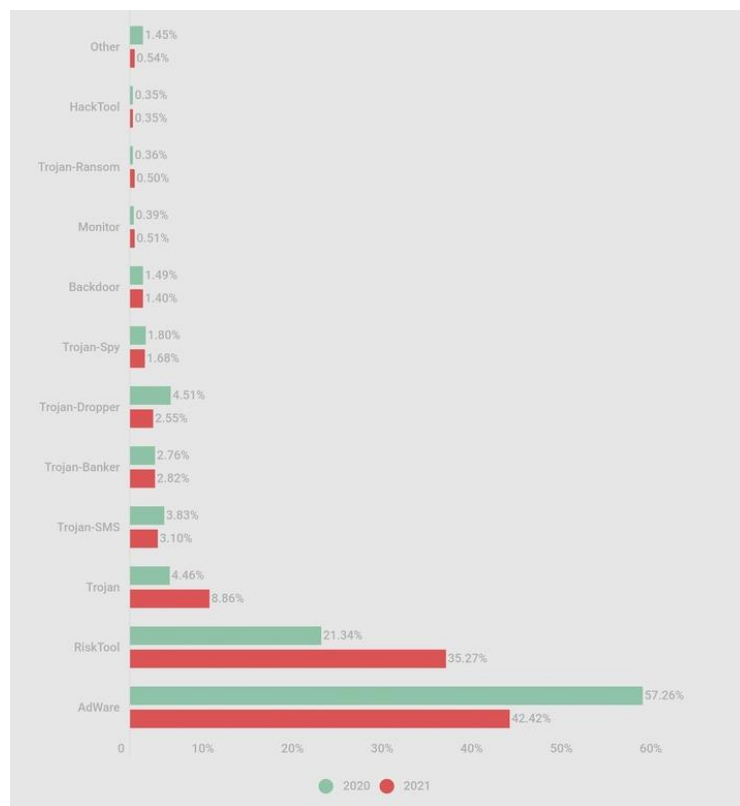


Figure 2.2: Malware detected by categories

The Trojan malware category is the third most common one to be found, and despite the fact that it only accounts for 8.86% of all detections, it is regarded as being far more dangerous than the first two categories. It is also important to note that the number of detected Trojans nearly doubled between the years 2020 and 2021.

2.3 Types of malware attack

Researchers and cybersecurity professionals are faced with an open challenge as a result of the unrivalled hazards provided by malicious Android software. This software is the root cause of a wide array of security vulnerabilities that are currently plaguing the internet. The speed with which malware samples may be identified and fixed is a critical factor in determining whether or not this threat can be eradicated. It is impossible to do anything else. Having a solid understanding of the many different families and types of Android malware is essential to accomplishing this goal. Adware, backdoors, file infectors, potentially unwanted applications (PUAs), ransomware, riskware, scareware, spyware, trojans, trojan-sms, trojan-spies, trojan-bankers, and trojan-droppers are some of the most frequent forms of harmful software for Android devices. [4]

<i>Android Malware Category</i>	<i>General Description of Behaviour</i>	<i>Common Malware Families</i>
<i>Adware</i>	Displays adverts in an unpleasant pop-up window to the user.	gexin, batmobi, ewind, shedun, and adcolony
<i>Backdoor</i>	Stealthily utilizes the device by remaining in the background.	mobby, kapuser, hiddad, dendroid, and droidkungfu
<i>File Infector</i>	Files are contaminated particularly executable (APK) files.	leech, tachi, commplat, gudex, and aqplay
<i>PUA</i>	Acts as an annoying thing that stops the device from doing what it should be doing.	apptrack, secapk, wiyun, youmi, scamapp, utchi, cauly, and umpay
<i>Ransomware</i>	Performs the function of a crypto locker, which encrypts the user's files and directories and then requests a ransom payment from	congur, masnu, fusob, jisut, koler, lockscreen, slocker, and smsspy

	the user in order to decrypt his or her own data.	
<i>Riskware</i>	Potentially endangers the smartphone's weak spots that could be exploited by attackers.	badpac, mobilepay, wificrack, and tordow
<i>Scareware</i>	Performs the function of a fear coaxer, which causes the user to experience dread and drives them to download malicious applications.	avpass, mobwin, and fakeapp
<i>Spyware</i>	Spy activities are used to get useful information from the device and send it to a server that is controlled from afar.	spynote, qqspy, spydealer, smsthief, spyagent, spyoo, smszombie, and smforw
<i>Trojan</i>	Performs actions in the background similar to those of an imposter, which continually steals information from the device. It can appear in a variety of myriad forms, such as the trojan-banker, trojan-dropper, trojan-sms, and trojan-spy.	gluper, lotoor, rootnik, guerrilla, gugi, hqwar, obtes, and hypay

Table 2.1: Type of Malware

2.3.1 Adware and Backdoor

Adware is short for the term "advertising malware". It's a malicious application that floods the user's screen with unwanted ads, usually when they're trying to use online services. Adware is software that displays intrusive advertisements, often with enticing offers, in the hopes that the user will click on them. The maker of this obnoxious app makes money whenever a user interacts with the in-app advertising. [5]

Typical forms of adware include software that claims to assist users in avoiding screen warnings about phony infections, losing weight, or making more money in less time. Many forms of adware can be downloaded onto a smartphone after any program or application is installed. Information such as phone number, email address, application accounts, IMEI number of the device, device ID, and device status may be gathered by the adware. The adware that gains access to a device's camera can steal personal data. On occasion, the adware will try to encrypt data on infected devices and install further malicious software, code, or files.

Backdoors are essentially secret entrances into a mobile device. Simply said, backdoors are a method through which an attacker can bypass a smartphone's authentication measures and get root access to the device. The term "trap door" is commonly used to describe a back entrance. The use of backdoors enables attackers to start attacks from a distance without physically possessing the target system. They might be completely new programs, or they could be a part of an existing program.

For example, attackers carefully conceal malicious code within legal programs, ensuring that it is only activated in specific circumstances. It has been noticed that in some cases, malicious code can be injected into a device and used to take control of it remotely if the user has not changed the default passwords of any accounts they create on the device. Malware that gains access to a device through a backdoor can steal sensitive data, send and receive messages, make and receive phone calls, record call history, compile a list of installed and running applications, and allocate memory.

2.3.2 File infector and PUA

Malware that adds itself as an attachment to APK files is referred to as a file infector. The Android Package Kit, or APK, is a file that contains all of the information about an Android app. The APK files are used to install the file infector. Malicious code is then executed anytime an APK file is installed. APK files can represent any type of Android program, from games to word processors to GPS applications.

As a result of recent events, Google has removed a number of apps from the Play Store after discovering they may contain malware. When executed, file infectors slow down the device and drain the battery significantly. These capture information regarding the device ID, IMEI number, and the status of the phone. They might disable, damage, or utilize the programs on your phone. They have the capacity to access, modify, and collect data from the settings and files of the device. In the worst-case scenario, malicious files infect the device's system files and take control of it.

PUAs are possibly undesirable apps that are packaged with genuine software that is supplied for free. PUAs spread alongside legitimate programs because they travel in the same channels. Potentially Unwanted Programs (PUPs) are another name for them[6]. Despite common belief, there are some situations in which a PUA would be beneficial. It is contingent on how they are put to use.

When an application is installed that includes a PUA, the PUA is also installed automatically. This risk can take several forms, including adware, malware, and browser hijackers. Memory-hogging PUAs slow down the device. Spyware applications are designed to acquire sensitive data from the device that is the target of the attack and pass it to the attacker, and PUPs might lead to more of the same. Using GPS, they are able to track the user's location, display unwanted advertisements, alerts, and links, and create shortcuts on the user's home screen.

2.3.3 Ransomware and Riskware

Ransomware is a form of malicious software that encrypts files and directories on a computer system, making those files and directories inaccessible to users of the system. For the purpose of delivering the decryption key that may be used to gain access to the data, it asks a significant sum of money to be paid as a type of ransom. Bitcoins are regularly utilized as a form of payment for ransomware demands.

However, some incidents have shown that some consumers have been unable to regain access to their data even after paying the necessary quantity of money. Incomplete files were reportedly received by some of them. At times, data would mysteriously vanish. The evolution of Android ransomware has been dramatic, and new strains are constantly being created.[7] Some ransomware strains pose as popular apps and manage to avoid detection. The sending and receiving of SMS messages, the locking of SIM cards and cell phones, the theft of network information such as Wi-Fi connection details, and communication with the remote server that controls the ransomware attack are all activities that are carried out by ransomware.

A program is considered to be riskware if it is completely legal but nevertheless has the ability to compromise the system's security in some way. To steal information from users' devices and direct them to malicious websites, hackers are exploiting a genuine piece of software.

It may also be called malicious software if it compromised the security of the device while performing its intended tasks. Riskware has the ability to snoop on users' data, including phone numbers and contact lists, send and receive text messages, steal network data, direct users to malicious websites, download, and install malicious software, display malicious advertisements, and alter the device's settings and files.

2.3.4 Scareware, Spyware, and Trojan

Scareware is software that produces fear in computer users in the hopes that they would download or purchase apps that contain malicious code. Convincing consumers to install a false application that promises to protect the device. Scareware is software that, in addition to installing dangerous programs on a device, attempts to gather information about the device, including its GPS location.

When installed on a computer, spyware can monitor user activity and steal sensitive data. Spyware collects data and sends it to a variety of commercial and non-profit entities. In the future, this data is put to use in the commission of criminal activities. Android users are requested to grant permission before any spyware is installed, but spyware can still access the phone's location, camera, and settings without the user's awareness or approval. Spyware can send and receive text messages, track a device's location and phone number, steal data about the networks to which a device is connected (including the Wi-Fi networks to which it is connected), modify system files and settings, and steal personally identifiable information.

When they are executed, Trojan's masquerade as legitimate software so that they can steal information and do damage. They are able to obtain sensitive data from the device while remaining undetected in the background[8]. It is the most widespread form of malicious software, and it incorporates a number of subtypes of malware as well, such as trojan-banker, trojan-dropper, trojan-sms, and trojan-spy. It is also the most dangerous. Trojans will typically engage in activities such as deleting, modifying, blocking, or copying data in an effort to disrupt the operations that are carried out by the operating system.

2.4 Android Malware Detection Approaches

Over the span of the last several years, there has been a significant rise in the number of academic research focused on the detection of android malware. In the beginning, the approach of signature-based detection was implemented quite frequently. This approach is quick and effective when used to counter previously discovered malware, however it is not as effective when used to resist zero-day malware[9]. Over the course of time, researchers have started employing methods such as anomaly-based, specification-based, and model-checking-based detection. Also, completely new methods of detection such as those based on deep learning, the cloud, mobile devices, and the internet of things.

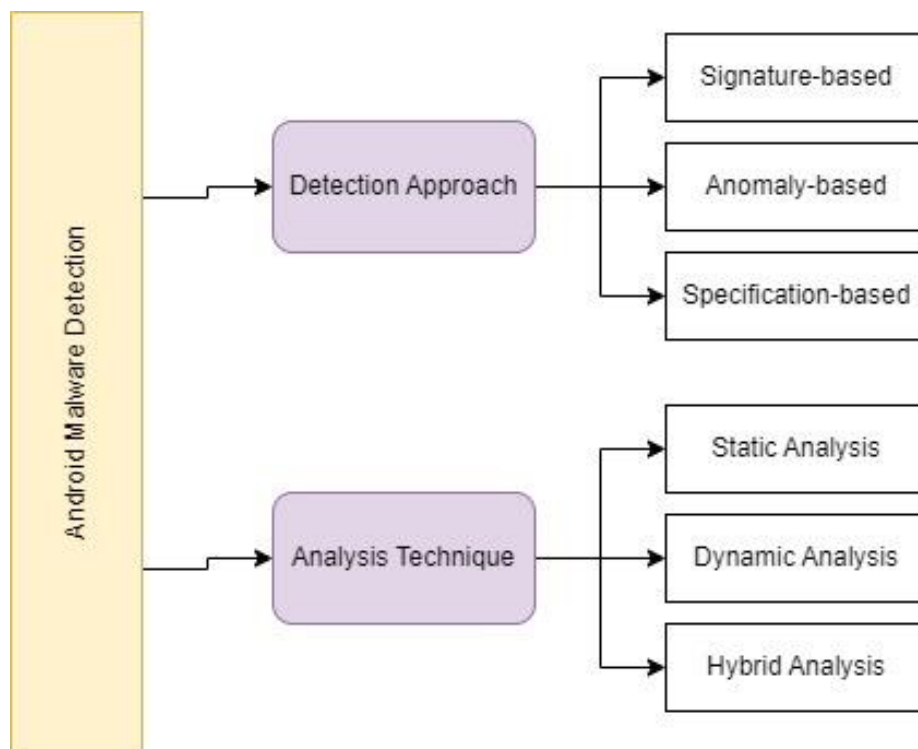


Figure 2.3: Malware Detection Approach

2.4.1 Signature-based Detection Approaches

A signature is a feature of malware that encapsulates the program structure and identifies each piece of malware in a way that is unique to that virus. Signature-based detection is a common method utilized by commercially available antivirus software. This method is quick and effective for identifying known forms of malware, however it is insufficient for identifying unknown forms of malware. In addition, malicious software that comes from the same family has a much better chance of evading signature-based detection if it makes use of obfuscation techniques.

When a sample of a program needs to be determined whether it is malicious or not, the signature of the sample in question is extracted in the same approach as it was before, and then it is compared with signatures that are stored in the database. The sample program is classified as either malware or benign, depending on the results of the comparison. Creating a signature can be accomplished using a wide variety of methods, including string scanning, top-and-tail scanning, entry-point scanning, and integrity verification, among others.

Signature-based detection schema has been used for many years by antivirus manufacturers since it is a very efficient method for detecting known malware, and it has been utilized for those years.[10] In most cases, this method is utilized to identify malicious software that is a member of the same family. On the other hand, it is unable to detect malware of a newer generation since it employs strategies like obfuscation and polymorphism. The signature should be as short as possible and can represent many different types of malwares with a single signature, an effective automatic signature generation mechanism must be built, data mining and machine learning techniques should be used more frequently during the generation of the signature, and the signature should be resistant to packing and obfuscation techniques. These requirements must be met for a signature to be considered effective.

2.4.2 Anomaly-based Detection Approaches

Anomaly-based detection is one type of intrusion detection system, and it works by keeping tabs on system activity and labelling it as normal or abnormal. This enables the system to identify malicious activity on a network or computer and prevent further abuse of the system. Instead of looking for patterns or signs, the classification, which is based on heuristics or rules, makes an effort to identify any kind of improper use that deviates from the typical functioning of the system. Signature-based systems, on the other hand, are limited in their ability to detect assaults because they can only identify those for which a signature has already been developed.

In addition, it is also able to recognize unknown attacks by basing its analysis on the behaviour of past intrusions that are similar. The strategy of anomaly-based detection is one that seeks to find incidences of malware by modelling what is considered normal. Therefore, everything that is not consistent with this model is regarded as unusual. This method is useful for identifying previously unknown forms of malware. In order to construct the model using anomaly-based detection, many characteristics had to be extracted from the Manifest file of Android applications. These features included uses-permission and uses-features. These features were utilized to develop the usual model of multiple legitimate applications, which was then used to identify malicious programs.

Other efforts, such as the model that was developed, made use of entropy-based anomaly detection to identify distinct abnormalities in the manner in which Android applications behaved on the system. To detect malicious software on Android devices, they made use of two popular entropy measures: sample entropy and changed sample entropy. Researchers interested in computing systems and network traffic have begun to focus on anomaly-based malware detection. Various technologies, such as those based on data mining and machine learning, have been utilized in the process of detecting mobile malware. These include statistically-based strategies, methods, and approaches. The researchers merged permissions and API calls in a machine-learning approach to malware, which allowed machine-learning methods to be applied in anomaly-based malware detection.

2.4.3 Specification-based Detection Approaches

Specification-based detection approaches monitor applications for normal and inappropriate behaviour. Heuristic-based detection uses machine learning and AI to recognize legitimate software's valid and invalid activities, while specification-based detection analyses the system specification's behaviour.

Otherwise, the specification-based detection determines whether or not a program is malicious based on the possibility that it violates a specified set of rules by referring to a rule set that specifies what kinds of behaviours are recognized to be normal. Malicious software is defined as any application that operates in violation of a predefined set of rules. In specification-based malware detection, a detection method that addresses the limitation of pattern-matching was developed. This was implemented so that malware might be detected more effectively. This technique takes advantage of instruction semantics in order to identify instances of malware. The method has a high degree of resistance to conventional methods of obfuscation.

This approach does not permit the proper specification of an attribute of a program, which is one of its limitations. The concept of anomaly-based detection provided rise to the concept of specification-based detection. When using specification-based detection, an approximation of the system's or application's requirements is created rather than an estimation of the system's or application's implementation. A training phase is present in a specification-based approach. During this phase, an attempt is made to learn all of the valid behaviour of a program or system that has to be inspected. The primary drawback of a system that is based on specifications is that it is extremely challenging to provide an accurate description of the operation of the system or the software.

2.4.4 Comparison of Android Malware Detection Approaches

Many different kinds of malware detection have been developed, each of which has seen its own set of capabilities becoming more sophisticated.[11] The following table compares and contrasts the advantages and disadvantages of three different approaches of detection: signature-based, anomaly-based, and specification-based detection approaches.

<i>Android Malware Detection</i>	<i>Advantages</i>	<i>Disadvantages</i>
Signature-based	<ul style="list-style-type: none"> - Can detect known attacks accurately. - Less amount of system resource is required to detect intrusion, - Focus on attack behaviour. 	<ul style="list-style-type: none"> - It cannot detect new, and unknown intrusion methods. - Ineffective against previously unseen attacks, as no signatures are available for such attacks.
Anomaly-based	<ul style="list-style-type: none"> - It can detect new intrusion methods and novel attacks. - Focus on normal behaviour to overcome undetected unknown attacks. 	<ul style="list-style-type: none"> - It needs to update the data describing the user's behavior and the statistics in normal usage and tends to be large. - Problem to select the appropriate set of features to be able to detect potential attacks. - Need more resources like CPU time, memory, and disk space.
Specification-based	<ul style="list-style-type: none"> - Attacks can be detected even though they may not previously encounter 	<ul style="list-style-type: none"> - It is not as effective in detecting novel attacks, especially in network probing and DOS attacks. - Development of detailed specifications is time-consuming.

Table 2.2: Comparison of malware detection approaches

2.5 Analysis Technique

Analysis of malware is the first step in identifying malicious software. In order to detect malware, we must first investigate how malware possesses its function and what the motivation is behind the development of malware. Having this level of understanding about malware makes it much simpler for the developers of malware detectors to put defensive features into their products. The processes involved in analyzing malware can be categorized into three distinct groups according to the amount of time and effort required to complete the analysis.[12]

These techniques can be divided into three different categories: static features, dynamic features, and hybrid features, depending on whether they are generated by running an Android application.

2.5.1 Static Analysis

The technique of static analysis for the identification of malware does not execute or run the code of the malware, but it relies exclusively on the properties of the malicious abstraction. When utilizing this method to detect malware, the most reliable features for detection come from the application's byte code or its manifest file.

Applications for Android are stored in an archive format known as APK. This comes in a zipped-up package the majority of the time. Included in this package are each and every one of Android's files, directories, and other resources. Most of the time, the process of reverse engineering is applied to the apk files in order to do feature mining. When searching for the extraction of key features, the manifest file known as "AndroidManifest.xml" is the first thing that should be examined. This manifest file includes permission vector features for access to the installation, locations, battery optimization, and phone state permissions. These features may be found in this manifest file.

Ankita used 103 malware datasets and 97 benign application datasets correspondingly, and it was able to detect malware on a Nexus 5 device with API level 19. It also found that the malware was performing high-level unauthorized permission assaults. The XML parser extracted the permission request, which then generated binary features of the malware. These features were stored in Attribute Relation File Format (ARFF). The results showed a detection rate of 96.6% when the random forest algorithm was implemented, with just a minimal difference of 0.069% from the algorithm with the worst detection rate.[13]

Once the code that is operating the malicious program has been thoroughly reviewed by trained systems, the appearance of the malicious application will be much easier to spot. The power of DSA was characterized when it was applied at the input and extraction layer of the model, and it was characterized by the different detections that were achieved by the trained algorithms. The detection accuracy of the random forest method was found to be 97% better compared to the other algorithms.

2.5.2 Dynamic Analysis

System calls to make it possible for applications that are based on Android to communicate with the operating system of the device, which in turn makes it possible to view the events that take place between the two parties. Android malware is monitored in a controlled environment during runtime by the dynamic detection analysis. This is accomplished by making a record of the malware pointers and deciding which detection signatures may be modelled using them. In order to accomplish this, it is necessary to take into consideration the dynamic behaviour of the malware. It investigates the ways in which malicious software interacts with mobile resources and services, including location, networks, packages, and actions carried out by the operating system.

The research demonstrated the effectiveness of this detection method on 4034 malicious datasets and 10024 benign datasets, respectively. The malware was found on those applications by the random forest classification algorithm with a success rate of 96% when the ServiceMonitor methodology was applied. The accuracy of the information that the malware collected, such as the phone's IMEI, was determined to be 67%. 17% of the malicious applications that were found to be running on the device were found to have attached their payload there in exchange for a premium service rating. The overhead device performance of the mobile utilities such as CPU and Memory was detected to be infected with a value of 0.8 percent and 2 percent, correspondingly.

After being downloaded and installed, certain malicious software will remain hidden on the device until it is prompted to perform an action. While some execute their payload in this approach, others do so neither during download nor during installation or runtime. Even if the default permissions are constantly encountered throughout the application download and installation process on Android devices, access authorization that is routinely authorized by users provides a vast area in the attack vector for the device. During those tests, malicious code is attached to programs that aren't malicious at all. It is necessary to do critical monitoring at each of these stages in order to improve the mobile platforms' level of security.

2.5.3 Hybrid Analysis

This analysis combines both the features of dynamic and static analysis to provide a more robust detection result when analyzing malware. The basic aspects of training and detection, which can be carried out by dynamic and static analysis, are included in the hybrid detection approach to the detection of malicious software. Because the benefits of both approaches are combined, this seems to result in a higher detection rate than either the dynamic or the static procedures individually. The hybrid analysis gathered a total of 192 examples for training, including both malicious and benign android software. The model produced detection results with an accuracy of 96.60%, with only a 0.0021% difference in accuracy amongst the various techniques that were utilized.

A comparison of the accuracy of the static and dynamic detection rates can be made with the assistance of the hybrid technique. As an analyzer, Android Buster Sandbox was employed, which allowed for the definition and establishment of the maliciousness and benignness of an application. However, Android malware detection using API call sequence was unable to circumvent the issue of malware obfuscation. This approach is unable to build the first malware distribution state in the call graph and sequence respectively because the observational sequence of the malware features does not produce a relational correlation to the HMM's distinct states. Even though this strategy was successful in preventing evasion assaults in Android malware, machine learning was unable to solve the problem of Android malware being poisoned.

2.5.4 Comparison of Analysis Techniques

There is a comparison between static, dynamic, and hybrid analysis techniques in analyzing and detecting android malware. These are the advantages of three analysis techniques:

<i>Approach</i>	<i>Advantages</i>	<i>Disadvantages</i>
Static Analysis	<ul style="list-style-type: none"> - Fast, safe, and low resource consumption. - Multipath malware analysis and more secure than dynamic analysis. 	<ul style="list-style-type: none"> - Can't analyze obfuscated and encryption malware. - Can't detect unknown malware.
Dynamic Analysis	<ul style="list-style-type: none"> - Can analyze obfuscated and encryption malware. - Can detect both known and unknown malware. 	<ul style="list-style-type: none"> - Slow, unsafe, and high resource consumption. - Time-consuming and vulnerable.
Hybrid Analysis	<ul style="list-style-type: none"> - Better than static and dynamic analysis. - Have the highest accuracy among the three analyses. 	<ul style="list-style-type: none"> - More time and resources consuming. - Highest complexity.

Table 2.3: Comparison between analysis techniques

2.6 Machine Learning

In machine learning, data and algorithms are used to model the way humans learn, with the ultimate goal of improving the realism of the model over time. This area of AI and computer science is rapidly growing in popularity. Early AI researcher Arthur Samuel coined the phrase "the branch of research that provides computers with the ability to learn without explicitly being programmed" in the 1950s.

Storage and processing capacity improvements over the past few decades have paved the way for a plethora of innovative machine learning-based solutions. Such items include driverless vehicles and Netflix's recommendation engine, to name a few examples.

A key topic of data science is machine learning, which is expanding rapidly. Algorithms are frequently trained in data mining projects to produce classifications or predictions and to discover key insights by making use of statistical methods. These discoveries inform the ensuing application and enterprise decision-making process and, ideally, influence key growth KPIs.[14]. As the big data industry continues to develop and thrive, it is reasonable to anticipate that there will be an increased demand for data scientists in the market. They will be expected to lend a hand in deciding the most pertinent business questions, as well as the data that is necessary to answer those questions, which will be one of the responsibilities placed on them.

There are many different ways in which machine learning algorithms can be trained, and each of these ways has both advantages and disadvantages. Machine learning may be roughly broken down into four distinct subfields, each of which is characterized by a distinct set of learning strategies and techniques.

2.6.1 Supervised machine learning

Supervised learning, or supervised machine learning, is the practice of teaching an algorithm how to accurately classify data or make predictions using only examples from labelled datasets. No matter what information is fed into it, the model will keep adjusting its weights until it is properly matched. In order to ensure that the model is not overfitting or underfitting, this procedure is performed as part of the cross-validation process. Supervised learning allows businesses to expand their efforts to solve a wide range of real-world problems, such as identifying and deleting spam emails. Supervised learning allows for the implementation of a wide variety of methods, including neural networks, naïve bayes, linear regression, logistic regression, random forest, and support vector machines (SVM).

2.6.2 Unsupervised machine learning

Unsupervised machine learning, or unsupervised learning, is the application of machine learning algorithms to data without the benefit of a labelled training set. This kind of learning is employed to rank and classify data that has not been labelled. Unlike human researchers, these algorithms can find new patterns and clusters in data on their own. This strategy excels in applications where similarities and differences need to be uncovered, such as exploratory data analysis, cross-selling strategies, consumer segmentation, and picture and pattern recognition. As a by-product, it can be used to reduce the number of features in a model through a technique called dimensionality reduction. Principal component analysis (PCA) and singular value decomposition are two methods that are commonly employed for this purpose (SVD). Unsupervised learning can also make use of other kinds of algorithms, such as neural networks, k-means clustering, and probabilistic clustering techniques.

2.6.3 Semi-supervised learning

The benefits of both supervised and uncontrolled learning can be seen in a semi-supervised setting. During the training phase, it employs a smaller labelled data set to guide the analysis of a larger unlabelled data set and the extraction of relevant features. Semi-supervised learning can be used to address situations where a supervised learning system does not have access to enough labelled data. It's also helpful when it would cost too much to classify a large amount of data manually.

2.6.4 Reinforcement learning

The supervised learning model and the machine learning paradigm known as reinforcement learning are quite similar. In contrast, the algorithm in reinforcement learning is not learned through the use of examples. This model learns as it goes by making mistakes. As a problem's solution, counsel, or policy is developed, it will be built upon and supported by a series of successes.

2.7 Previous Research Works

My research project and study on "MALWARE DETECTION IN ANDROID USING MACHINE LEARNING," must begin with a review of prior research articles. Engaging with earlier works has several benefits and can be essential for determining the purpose and contributions of the research.

First of all, reading through earlier research papers enables to develop a thorough awareness of the state of current knowledge and developments in the field of Android malware detection. Able to recognise gaps, restrictions, and areas that need more research by looking at the approaches, procedures, and experimental designs used in previous research.

Additionally, reviewing previous research articles assists in clarifying and identifying the research objectives and hypotheses of the study. It enables to recognise the gaps in the corpus of knowledge and create research questions that add to it. By evaluating the strengths and weaknesses of prior research, we can design an improved experimental setup and choose appropriate methodologies and techniques to achieve higher accuracy and reliability in the testing results.

Moreover, reviewing earlier research publications gives a platform for comparing and validating the individual findings. We can compare the results to previously published research to acquire a greater understanding of the merits and drawbacks of the suggested method that can be apply in my research. This comparative analysis contributes to the overall credibility and impact of the research, as it demonstrates the novelty and advancements in study brings to the field of Android malware detection.

2.7.1 Comparison and description of previous research paper

Research Paper	Summary	Technique Used	Dataset	Advantages	Disadvantages
"A Comparative Study of Machine Learning Techniques for Android Malware Detection"	This paper compares different machine learning techniques for Android malware detection. It evaluates classifiers using a large dataset.	Decision trees, SVM, NN	A dataset of over 100,000 Android applications	Provides insights into the effectiveness of various techniques	Limited discussion on feature selection and model evaluation methods
"Deep Learning-Based Android Malware Detection Using Recurrent Neural Networks"	The research focuses on using recurrent neural networks (RNNs) for Android malware detection. It explores LSTM and	Recurrent Neural Networks	A dataset of over 10,000 Android applications	Captures temporal dependencies in app behavior	Limited discussion on dataset characteristics and preprocessing techniques

	GRU architectures.				
"Feature Selection Techniques for Android Malware Detection using Machine Learning"	This paper investigates feature selection techniques for Android malware detection. It compares mutual information, chi-squared, and RFE.	Mutual information, Chi-squared, RFE	A dataset of over 10,000 Android applications	Identifies informative features for accurate detection	Limited exploration of other feature selection algorithms
"Ensemble Learning Approaches for Android Malware Detection"	The research explores ensemble learning approaches for Android malware detection. It investigates bagging, boosting, and stacking methods.	Bagging, Boosting, Stacking	A dataset of over 100,000 Android applications	Improves detection accuracy through ensemble methods	Limited discussion on the specific ensemble configurations and their impact on performance

"Android Malware Detection using Hybrid Machine Learning Models"	The paper proposes a hybrid machine learning approach for Android malware detection, combining decision trees, random forests, and SVM.	Decision trees, Random forests, SVM	A dataset of over 100,000 Android applications	Leverages strengths of multiple models for improved accuracy	Limited discussion on the hybrid model architecture and training process
--	---	-------------------------------------	--	--	--

2.8 Conclusion

This chapter makes various comparisons between past solutions that have been suggested by another researcher and the current solution, which is Machine Learning. In addition to that, this chapter demonstrates the many methods that the researchers utilized to accomplish their goal of developing an analysis method that can detect malware. Over the course of recent years, a great number of different approaches have been suggested. And finally, we get an overview of the many methods used by android malware detection to combat previously known malware. However, those methods required further development to achieve more satisfactory outcomes in the future. The current solution, which was described in Chapter 3, is intended to assist Android users in identifying malware that may be present in their respective devices

CHAPTER 3

METHODOLOGY

3.1 Introduction

The definition of malware as well as the tools that it possesses to confine itself was covered in the chapter before this one. In Chapter 2, we've previously talked about a few of the existing research projects that have been suggested as ways to detect malware. As a result, the particulars regarding the strategy, method, and characteristics that will be utilized in the course of this research will be explained in this chapter, along with the methodology that will be utilized in the course of the experiment that will be carried out.

3.2 Research-Based

The research-based method comprises four main phases, which are the study of existing literature, the development of new architecture, the design of the system, its implementation, and finally, testing and evaluation. The adoption of this methodology in this research is effective since the phases may be continuously examined to ensure high-quality outcomes. This research-based system development life cycle is distinct from other life cycles for system development that have already been proposed. It is for this reason that this strategy will concentrate on managing and observing each and every detail of the research conducted on this research title.

The review of previous research is the initial step in this research-based process. During this stage, the prior studies that have been done on the topic of the research will be examined and reviewed in detail. Following that, the problem statement and the aims of this research are characterized as the definition of this research. The subsequent stage will involve the creation of new design requirements. During this development, a critical study of past studies will be examined as an appropriate algorithm and method to be used

in this research. This will be done in order to ensure that this research has the most accurate results possible.

After the concept for the research has been formed, the next steps in the research process will be to design and carry out the research. Therefore, the program, hardware, and language are the technical requirements that are needed for this research. When all of the needs have been planned for and are available, the research will then incorporate the design model and the detection model. As soon as the implementation has been finished, the research experiment will go through a process of testing and evaluation to determine the limitations of the research and the ways in which it may be improved in the future.



Figure 3.1: Main Phases of Research-Based

The research base is modified in this research because it is possible to revert to earlier stages with small amounts of data loss when implementing the new and improved research[15]. And beyond that, research-based approaches can be adjusted as needed to address pressing issues at hand. In conclusion, research-based also allows researchers to readily modify their methods to suit the specifics of any given project's investigation.

3.3 Planning and Reviewing Literature

The first step in this research-based approach is a complete review of relevant previous projects. The conceptualization is finished to find the appropriate kind of research question before the previous studies are analyzed. When a topic is settled for investigation, a collection of relevant literature is compiled. Existing research can be understood by analyzing the data already collected on the topic. In light of this, we are able to refine the problem description, the study goal, and the research scope. It has been recommended that the n-gram opcode be used to locate data regarding malicious code detection and the existing methods for detecting it.

For my study, I consulted a variety of online journals and articles as well as student references from the past. There is also a thorough examination and filtering of the current project studies to ensure they are relevant to the research question. What's more, the data collected should be used for further study and applied to the advancement of the research.

Studies of various approaches and techniques have the objective of determining which type of approach and technique is most suited to the task of resolving the issue that occurred on Android devices, particularly the issue that was associated with the malware. Malware detection on smartphones has to be the primary focus of this research given that Android's security flaws cause the most concern overall.

The existing research projects that are relevant to the detection of malware are evaluated critically and categorized according to the location where the malware code structure was carried out. Each of the different methods that have been suggested for detecting malware is investigated in order to determine its strengths and weaknesses. Consequently, it is necessary to have this kind of information to determine the approach that the researchers employed for the experiment testing. As a result, the limitations of previous research will not be replicated in this study.

3.4 Developing the Architecture

Based on the research done, it was determined to employ Machine Learning to advance malware detection. The Android malware detection system may be separated into five different phases. They consist of raw data collection, data analysis, feature selection and transformation, classification algorithm, and malware detection system[16]. The malware detection system's architecture is depicted in the figure below.

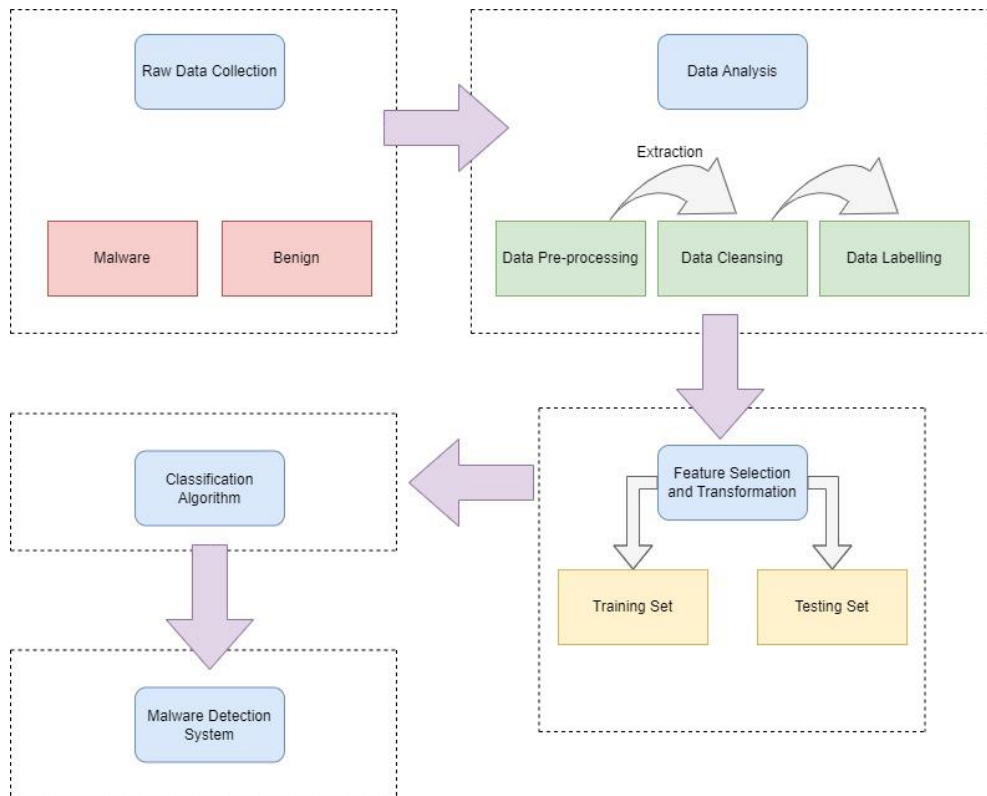


Figure 3.2: Malware Detection System Architecture

- 1) Raw data collection: This is the first phase of the Android malware detection system. During this phase, raw data is collected from a variety of sources, including the logs of Android devices, network traffic, and APK files. The data can be collected through various methods such as a device's APIs, network sniffing, or by scanning file systems. The data can also be collected by monitoring the device's behavior and activities, such as the installed apps, usage patterns, and network connections.

- 2) **Data Analysis:** During this phase, the collected raw data is analyzed in order to recognize patterns and features that can be utilized to differentiate between malicious and benign software. Techniques such as static analysis, dynamic analysis, and machine learning can all be utilized during this part of the process. The goal of this phase is to extract relevant information and features from the raw data that can be used to train a classification algorithm.
- 3) **Feature selection and transformation:** In this phase, the most relevant features are selected from the data obtained in the previous phase, and then they are transformed into a format that the classification algorithm can utilize which is the Training Set and Testing Set. This phase occurs after the data has been collected. In this stage, there is also a reduction in the dimensionality of the data, which is conducted in order to improve the classification algorithm's overall performance.
- 4) **Classification algorithm:** At this stage, a classification algorithm is trained using the features that have been identified and changed in the previous phase. The attributes that the algorithm has been trained on are utilized to determine if the software is malicious or benign. Different classification algorithms can be used such as decision trees, Naïve Bayes, Random Forest, and Artificial Neural Networks.
- 5) **Malware Detection System:** This is the final phase of the Android malware detection system, and it involves integrating the trained classification algorithm into a system that is capable of detecting malware on a real-time basis. It is possible for this system to have a user interface, reporting, and alerting capabilities, and it is also possible for it to be connected with other security systems such as firewalls and intrusion detection systems.

It is essential to keep in mind that this is but one of many conceivable architectures, and the manner in which these phases are actually implemented can change considerably depending on the kind of malware detection system that is being designed.

3.4.1 Procedure Description

During the process of developing the architecture for the Android malware detection system, a machine-learning technique was developed. This technique can train a dataset sample to learn the behaviour of both benign and malicious applications. The purpose of the implementation of this architecture is to identify whether or not new applications contain malware or are completely benign. In addition, this design is comprised of three components which are a data connection, machine learning, and a database. Each of these components is equally important.

The gathering of data began with the acquisition of all the permissions, which included both malicious and innocuous software applications. The decompilation of an APK file and its subsequent extraction for use in a data cleansing process that can filter permissions are both part of this data Pre-processing step. Before loading it, the next step in the data labelling process involves storing all of the permissions that have been constructed in a format that is legible and then saving the file as an Attribute-Relation File Format (x.arff) file. This arff file is where the feature attributes that are being utilized for the purpose of the approach to feature optimization can be accessed.

During this step, the information gain and bio-inspired algorithms that are employed for feature selection are utilized in order to locate and pick the characteristics that are of the highest quality. One of the most important functions of this technique for optimizing feature sets is determining the differences between a non-bio-inspired algorithm and a bio-inspired algorithm.

The gathering of data and the process of optimizing features are also essential components of the malware detection process. This is due to the fact that the procedure of collecting data obtained the malware characteristics and benign from the process of data cleansing. After that, this operation will notify the database of the change by sending an alert. At this rate, the data filtering will depend on the authorization and its package names in order to ensure that the same apps and features are separated from the database. After that, the filtered features are transferred to the Machine Learning process so that they can be used to optimize the features.

3.4.2 Data Collection Phase

Dataset	Source	Total Use in Experiment
Malware	AndroZoo	10,000
Benign		10,000
TOTAL		20,000

Table 3.1: Dataset Summary

During this stage, the process of data collection was used to assemble the malware applications and the benign datasets. In this stage of the process, the random samples originate from AndroZoo[17], which is split into two datasets, Benign and Malware. The data can be studied to discover patterns and characteristics that can be utilized to differentiate between malicious and benign software. The data that was obtained may be put to use for a variety of reasons, including reverse engineering, behavioural analysis, or code analysis.

It is essential to keep in mind that the type of malware detection system that is being developed as well as the particular dangers that it is intended to identify will determine the precise data that is collected as well as the methods that are utilized to collect it.

3.4.3 Decompiling the APK File

Android Application Package is what is meant by the abbreviation APK. It is the extension of a file that is compatible with Android devices and can be installed on those devices. After compiling a large number of files in Android Studio, an APK is an executable file that is produced for use on an Android device[18]. This file is part of the overall project. The process of creating source files from their compiled form is a highly difficult and time-consuming procedure. Android Studio creates the APK file by compiling several different types of files, including AndroidManifest.xml, java, or .kt

files, layout files, various media files, and many more types of files. The act of turning the code of an Android application's compiled version back into the application's original source code is referred to as decompiling an APK file.

The collection of benign and malicious applications is the initial stage of this procedure. A total of 20,000 samples of the dataset are collected, with 10,000 examples of benign software and 10,000 examples of malware applications. The Benign dataset is downloaded from AndroZoo, which is part of the Google Play store. The Malware dataset is acquired from a variety of different markets. The procedures involved in the collection of data are depicted in the figure that may be found below.

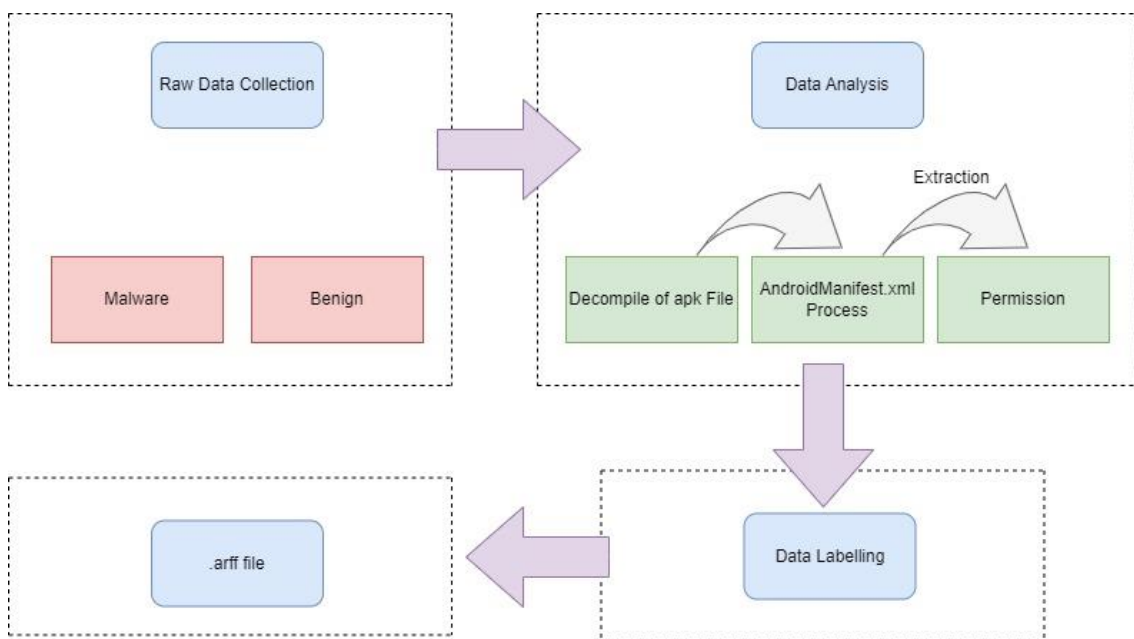


Figure 3.3: Data collection phase

An Android application's AndroidManifest.xml file is an important file that provides important information about the app's components, permissions, and other configurations. The file may be found in the app's root directory. The Android operating system looks to it to figure out the app's capabilities, requirements, and overall structure. It may be found at the very top of the application's project hierarchy.

The AndroidManifest.xml file contains a variety of elements that explain the app's components, such as activities, services, broadcast receivers, and content providers. It also contains information about the rights that the app has been granted, such as the permissions that are required for the app to access specific capabilities of the device, such as the camera, microphone, or internet access.

The AndroidManifest.xml file is used to retrieve important information such as the permissions and activities of an application. Before the permission can be saved in the database as an x.arff file, it is necessary for all of the permission that has been extracted to be labelled.

Benign Applications		Malware Applications	
Permission	Frequency	Permission	Frequency
INTERNET	1121	INTERNET	1199
ACCESS_NETWORK_STATE	663	ACCESS_COARSE_LOCATION	1146
READ_PHONE_STATE	391	VIBRATE	994
WRITE_EXTERNAL_STORAGE	362	WRITE_EXTERNAL_STORAGE	823
ACCESS_COARSE_LOCATION	236	READ_SMS	779
VIBRATE	210	WRITE_SMS	762
WAKE_LOCK	188	READ_CONTACTS	680
ACCESS_FINE_LOCATION	162	BLUETOOTH	633
GET_TASK	125	WRITE_CONTACTS	542
SET_WALLPAPER	102	DISABLE_KEYGUARD	491

Table 3.2: Top ten permission in Benign and Malware Applications

After comparing the top ten permission for benign and malicious applications (as mentioned in Table 1), I discovered that Malware applications sought a total of 8,049 permissions, which was much greater than Benign applications (3,560 permissions)[19]. The following four most often requested permissions by both clean and malicious applications are the INTERNET, ACCESS_COARSE_LOCATION, WRITE_EXTERNAL_STORAGE, and VIBRATE.

The top five permission for the Benign Application the INTERNET, ACCESS_NETWORK_STATE, READ_PHONE_STATE, WRITE_EXTERNAL_STORAGE, and ACCESS_COARSE_LOCATION. While for the Malware Applications, the INTERNET, ACCESS_COARSE_LOCATION, VIBRATE, WRITE_EXTERNAL_STORAGE, and READ_SMS. The figure below shows the number of requesting permission for benign and malware applications.

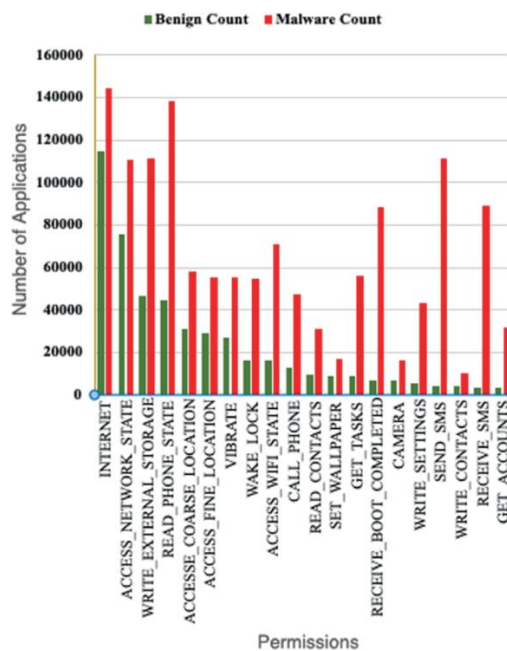


Figure 3.4: Number of applications requesting Benign and Malware

According to the graph above, benign applications are issued fewer permissions than malicious applications. The INTERNET, READ_PHONE_STATE, and WRITE_EXTERNAL_STORAGE proved that it has the greatest number of permissions for Malware Applications. This demonstrates that the attacker used permissions to propagate the malware among Android devices.

3.4.4 Features Selection

A key phase in machine learning is feature selection, which aims to find and choose the dataset's most pertinent and instructive features. The objective of the features selection is to keep the features that are most helpful for the prediction task while reducing the dimensionality of the data. It is necessary to think over and evaluate which qualities are most crucial before making a decision for the dataset.

Wrapper methods in feature selection involve training and evaluating a machine learning model with different subsets of features. It treats the feature selection process as a search problem, where different combinations of features are evaluated based on their impact on the model's performance. The algorithm evaluates the performance of different subsets of features by repeatedly training and testing the model. The key advantage of wrapper methods is that they consider the interaction and dependency between features. By evaluating feature subsets based on the performance of the model, wrapper methods can potentially identify relevant features that work well together, leading to improved predictive accuracy. The feature selection process is integrated into embedded approaches, such as decision tree-based feature importance or regularisation methods like L1 regularisation (Lasso)[20].

To achieve the highest level of testing accuracy, it is crucial to take certain factors into account while choosing features. I will ensure all of this steps to achieve high accuracy results during testing. First, there should be a significant correlation between the attributes and the target variable. Features that have a strong correlation to the target have a higher likelihood of making a substantial contribution to the prediction challenge. Secondly, in order to prevent multicollinearity, characteristics should demonstrate low redundancy among one another. Overfitting can result from redundant features, which increase the model's needless complexity. Third, take into account the features' interpretability and domain relevance. Better generalisation and simpler interpretation of the model's predictions are frequently brought on by intuitively significant features.

To find the best feature subset, a thorough investigation of the effects of the chosen features on the model's performance through experimentation and validation can be test repeatedly.

3.4.5 Machine Learning Classifiers

Machine learning is a rapidly growing field that has the potential to revolutionize many industries, including cybersecurity. One area where machine learning has shown promise is in the detection of malware. Malware, or malicious software, is a major threat to computer systems and networks, and traditional methods of detection are often not enough to keep up with the constantly evolving nature of malware.

Traditionally, malware detection has relied on signature-based detection, which looks for specific patterns or characteristics that are known to be associated with malware. This method is limited by the fact that it can only detect malware that is already known and that it can't detect new or unknown malware. Machine learning algorithms, on the other hand, can be trained to detect malware based on patterns or characteristics that are not known in advance. This is done by analyzing large sets of benign and malware samples and using this data to train the algorithm to identify patterns or features that are unique to malware. Once the algorithm is trained, it can be used to detect new or unknown malware in real-time.[21]

One of the key advantages of machine learning for malware detection is that it can adapt to changing malware threats. As new malware is discovered, the algorithm can be retrained on new data, which allows it to continue detecting new and unknown malware. Another advantage of machine learning for malware detection is that it can be used to detect malware that is designed to evade traditional detection methods. For example, machine learning can be used to detect malware that uses obfuscation techniques to hide its code or that uses legitimate system calls to avoid detection.

In this study, five different classifiers were used so that the researchers could compare and contrast the results obtained from the various machine learning classifiers. The names of the five classifiers are as follows: Naive Bayes, DecisionTable, J48, Random Forest (RF), and Multi-Layer Perceptron (MLP).

3.4.5.1 Random Forest Classifier

Random Forest is an ensemble learning method that uses multiple decision trees to make predictions. In this method, a large number of decision trees are created and their outputs are combined to make a final prediction. Each tree is built using a random subset of the data and a random subset of the features, which helps to reduce overfitting and improve the generalization of the model. The final prediction is based on the majority vote of the trees, or by averaging the predictions of the individual trees. Random Forest is known for its high accuracy, robustness to overfitting, and ability to handle high-dimensional data. It's also known for its ability to handle missing data and categorical variables.

3.4.5.2 J48 Classifier

J48 is an implementation of the C4.5 algorithm, which is a decision tree algorithm. J48 creates a decision tree by recursively partitioning the data into smaller subsets based on the values of the input features. At each partition, J48 selects the feature that maximizes the information gain, which is a measure of how much the data is reduced in uncertainty by the partition. J48 is known for its simplicity, interpretability, and ability to handle both categorical and continuous input features.

3.4.5.3 Multi-Layer Perceptron Classifier

Multi-Layer Perceptron (MLP) is a type of artificial neural network that is used for supervised learning. An MLP consists of one or more layers of artificial neurons and is trained using backpropagation. Each neuron in an MLP takes the input, applies a set of weights and biases, and then applies an activation function to produce an output. The output of one layer becomes the input for the next layer, and this process continues until the final output is produced. MLPs are known for their ability to model non-linear relationships and their ability to learn complex patterns in the data. However, MLPs can be sensitive to the choice of the activation function, the number of hidden layers, and the number of neurons per layer. They also require a large amount of labeled data to train, and they can be computationally expensive.

3.4.5.4 Decision Table Classifier

A Decision Table is a rule-based classifier that generates a set of if-then rules by analyzing the data. Each rule represents a decision made based on the input features and the corresponding output class. The decision Table is known for its interpretability, as the rules generated by the algorithm can be easily understood by humans. It's also able to handle missing data, but it's limited by the fact that it can only produce simple linear decision rules. It's also computationally efficient.

3.4.5.5 Naïve Bayes Classifier

Naïve Bayes is a probabilistic classifier that uses Bayes' theorem to make predictions. It is called "naive" because it makes the assumption that all input features are independent of each other, which is often not the case in practice. Despite this assumption, Naive Bayes is known for its simplicity, speed, and ability to handle large amounts of data. It's also able to handle missing data and categorical variables. It's commonly used for text classification, spam detection, and sentiment analysis.

3.4.6 Machine Learning Tool

The functionality of machine learning tools is for the analysis of data, which automates the model construction process. When judgments or predictions are made throughout the learning process, this model enables the system to gain knowledge from either historical or current data sets. The analytical processes can be simplified and sped up when a system is equipped with tools that are capable of machine learning. It is also able to automatically apply sophisticated mathematical calculations in order to answer problems, and this ability does not require any machine learning techniques or experience on the part of the user. Jupyter Notebook was the tool for machine learning that was implemented in this research.

3.4.6.1 Jupyter Notebook



Figure 3.5: Jupyter Notebook Tools

Data scientists can use the open-source web tool Jupyter Notebook to create and share documents that incorporate live code, equations, computational output, visualizations, and other multimedia elements alongside descriptive writing. Jupyter Notebook's ability to incorporate all of these components into a single file makes this possible. Jupyter Notebooks are flexible and may be used for a wide range of data science tasks, from preparing and manipulating data through numerical simulation, exploratory data analysis, data visualization, statistical modelling, machine learning, deep learning, and beyond.

Originally known as the IPython Notebook, the Jupyter Notebook was renamed to reflect its expanded language support beyond Python to include R, Julia, and others. It's a handy tool for data exploration and experimentation because users can combine code, output, and markdown text on a single page. The notebook interface allows users to run code blocks, visualize data, and display results all in the same document, making it easy to keep track of the development process and share the results with others. Jupyter Notebook also supports the use of interactive widgets, which allows users to interact with the data and the code in a more intuitive way.

Jupyter Notebook is widely used in the data science and machine learning communities and it's supported by a large number of libraries and frameworks. It can be run locally or on cloud-based platforms, and it's also supported by many popular machine learning frameworks such as Tensorflow, PyTorch, and scikit-learn. Jupyter Notebook has become a popular tool for data analysis, visualization, and machine learning, it's widely adopted in the industry, research, and education. Here are some steps that can be taken to use Jupyter Notebook for malware detection:

- 1) Install the necessary libraries: Jupyter Notebook runs on Python, so the first step is to install the necessary libraries for machine learning and malware detection. Some commonly used libraries include *pandas* for data manipulation, *scikit-learn* for machine learning, and *pefile* for analyzing Portable Executable (PE) files.
- 2) Prepare the dataset: The next step is to prepare the dataset for training and testing the model. This may involve collecting benign and malware samples, pre-processing the data and splitting the data into a training set and a test set.
- 3) Train and test the model: Once the dataset is prepared, it can be used to train and test a machine-learning model. This can be done using Jupyter Notebook by creating a new notebook, importing the necessary libraries, and writing code to train and test the model.
- 4) Evaluate the model: After the model is trained and tested, it's important to evaluate its performance. This can be done by using metrics such as accuracy, precision, recall, and F1-score. It's also recommended to use a confusion matrix to understand the model's performance.
- 5) Fine-tune the model: If the model's performance is not satisfactory, it can be fine-tuned by adjusting the parameters or by using a different algorithm.
- 6) Deploy the model: Once the model is fine-tuned, it can be deployed to detect malware in real time. Jupyter Notebook can be used to create a script that takes the input files and returns the prediction made by the model.

3.4.6.2 Jupyter Notebook Interface

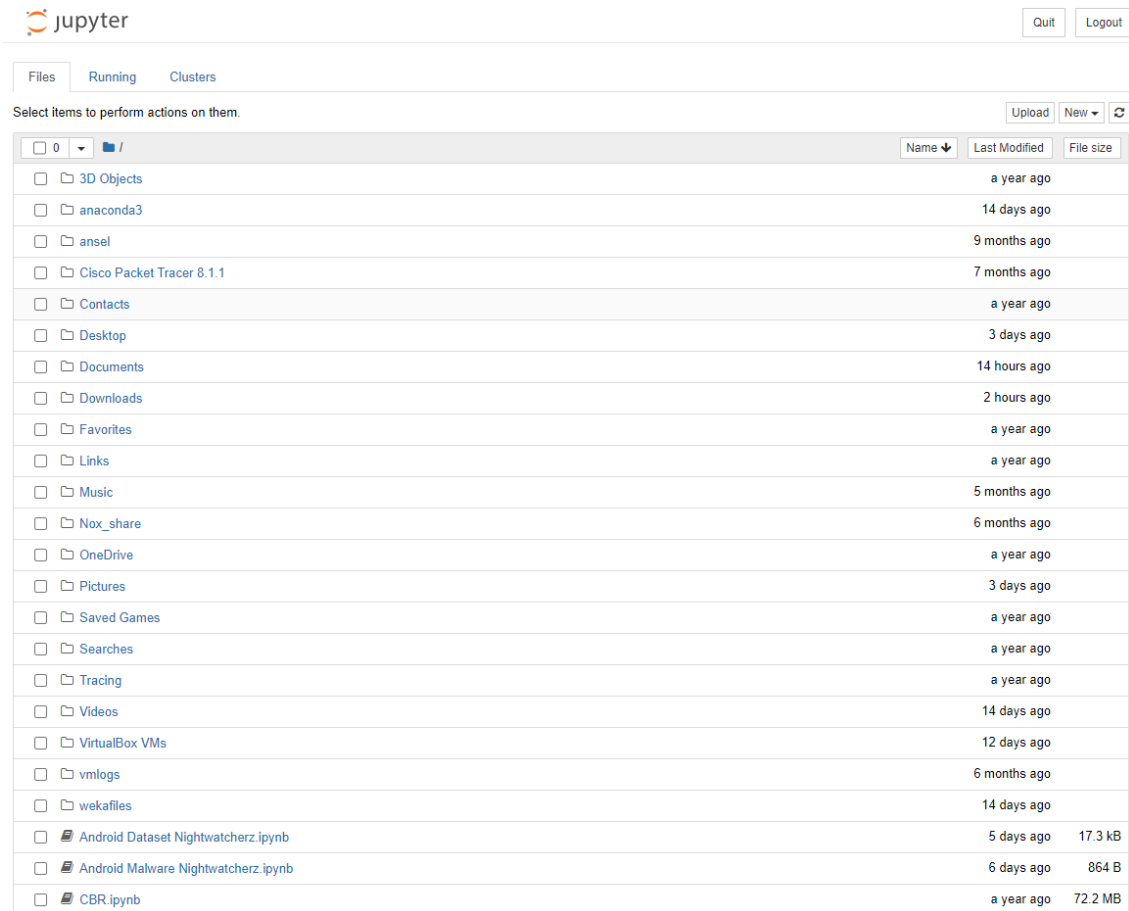


Figure 3.6: The main interface of the Jupyter Notebook

This is the interface for the Jupyter Notebook, and we can view all of the files in the directory that is currently active. The notebook icon that is located next to the name of each Jupyter Notebook makes it easy to recognize that notebook. If we already have a Jupyter Notebook in this directory that we would want to view, we can locate it in the list of files and then click on it to open the notebook.

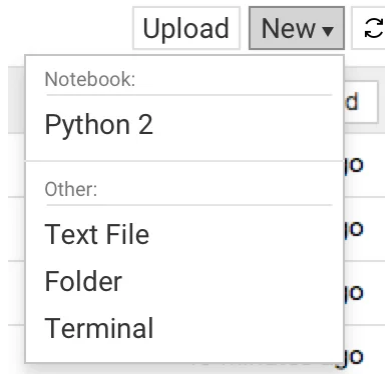


Figure 3.7: Option to start a new notebook

To start a new notebook, pick New and then Notebook - Python 2 from the drop-down menu. If there are any other Jupyter Notebooks on the system that we would want to use, all we need to do is click the Upload button and then navigate to the specific file that we would like to use.

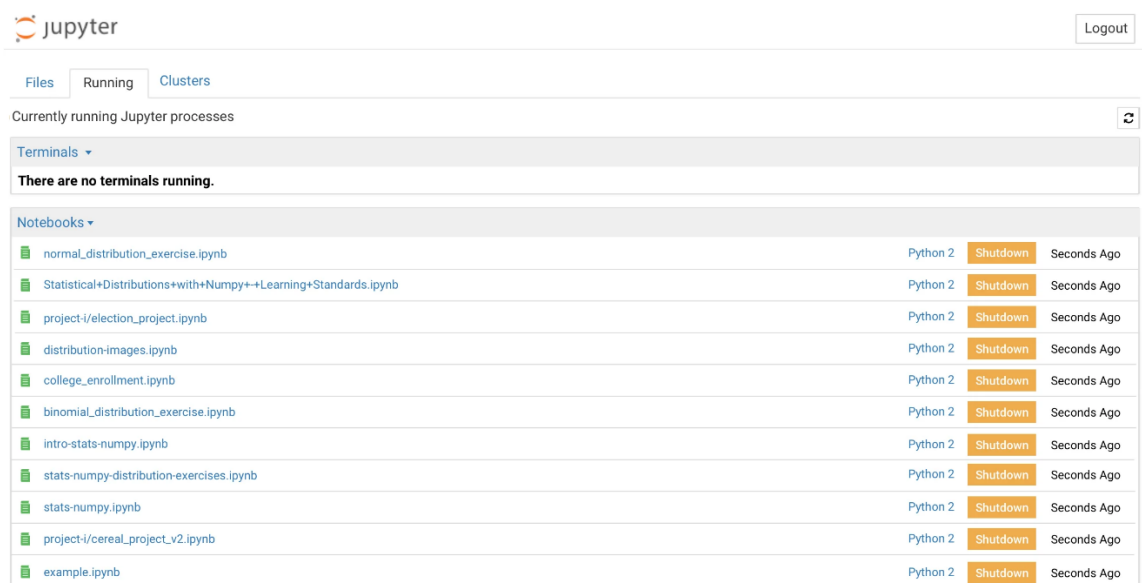


Figure 3.8: Running terminal in Jupyter Notebook

The icon for notebooks that are actively running will be green, while the icon for notebooks that are not currently running will be grey. Simply select the Running tab to bring up a list of all of the notebooks that are presently being used.



Figure 3.9: Jupyter Notebook cell

When we open a brand-new Jupyter notebook, we are going to see that it already has a cell in it. Notebooks are organized using cells, which also serve as the spaces in which we compose the code for our projects. To execute some code, first select the cell by clicking on it to make it active, then either hit the SHIFT+ENTER key combination or click the play button in the toolbar that is located above the worksheet.

3.5 Design and Implementation

After the framework has been designed, it will be necessary to demonstrate that the proposed framework is acceptable. Before the execution of the system, the generated is drafted in order to verify the accuracy of the anomaly detection. The proposed technique is developed to test the idea before proceeding to the malware detection system for Android mobile devices, as can be seen in the figure below.

The process of collecting raw datasets, defining the database, developing a system, testing the system, and, as the final step, comparing the results obtained are the five components that make up the design model.

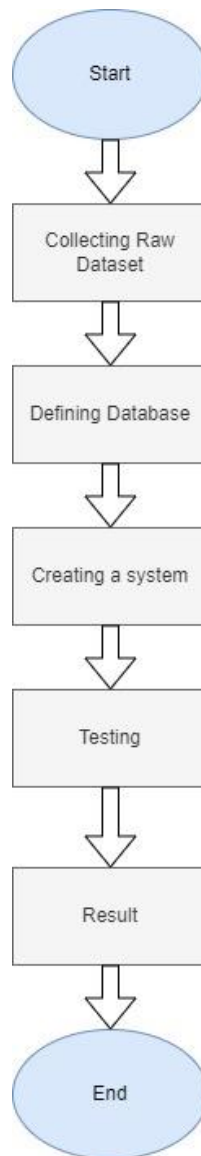


Figure 3.10: Flowchart of the procedure for improving detection method testing

The design and implementation phase in a malware detection system is a critical step in the development process, as it involves creating and implementing the solution that will be used to detect malware. The first step in the design and implementation phase is to clearly define the problem that the malware detection system is intended to solve. This may involve identifying the types of malwares that the system should be able to detect, the platforms and devices that it should run on, and the specific requirements of the users.[22]

Once the problem is defined, the next step is to select a machine-learning algorithm that will be used to detect malware. This may involve evaluating different algorithms, such as Random Forest, J48, Multi-Layer Perceptron, Decision Table, and Naive Bayes, to determine which one is the most suitable for the problem at hand.

The next step is to prepare the dataset that will be used to train and test the model. This may involve collecting benign and malware samples, pre-processing the data and splitting the data into a training set and a test set. Once the dataset is prepared, the model can be trained and tested. This may involve writing code to train the model on the training set, evaluating the model's performance on the test set, and making adjustments to the model as necessary. This step can be done using Jupyter Notebook or other programming environments.

After the model is trained and tested, it's important to evaluate its performance. This can be done by using metrics such as accuracy, precision, recall, and F1-score. It's also recommended to use a confusion matrix to understand the model's performance. If the model's performance is not satisfactory, it can be fine-tuned by adjusting the parameters or by using a different algorithm. Once the model is fine-tuned, it needs to be integrated into the malware detection system. This may involve writing code to take the input files, run the model, and return the predictions.

Finally, the malware detection system should be thoroughly tested and evaluated to ensure that it is working correctly and that it meets the requirements identified in the first step.

3.6 Hardware and Software

During the process of developing the project, it is required to compile a list of the necessary specifications. In order to carry out this research, the requirements for the experiment's hardware and software must first be determined. These requirements need to be met before the experiment can be set up. Due to the fact that both hardware and software are utilized in the process of carrying out this research as well as testing and assessing the experiment in preparation for the subsequent phase, this stage is essential.

3.6.1 Hardware Requirement

HARDWARE	DESCRIPTION
Processor: Intel® Core™ i5-5200U CPU @ 2.20GHz - RAM: 8.0 GB - System type: 64-bit Operating System, x64-based processor	Utilized for the purpose of carrying out the resource finding, implementation, testing, and documentation for the complete research study.

Table 3.3: Hardware requirement and description for this research

3.6.2 Software Requirement

SOFTWARE	DESCRIPTION
Windows 10	An operating system used to complete this research
Jupyter Notebook	To analyze, train, and test the dataset
Microsoft Word 2021	To write documentation for this report follow the guidelines and format
Microsoft Excel 2021	To review the dataset for Benign and Malware applications
Microsoft Edge	To discover and collect the information related to the topic of research
Draw.io	To create a related diagram and flowchart for the android malware detection system

Table 3.4: Software requirement and description for this research

3.7 Testing and Evaluation

The phase of testing and analyzing the results of this investigation is the final step. At this point, the experiment will be evaluated after all of its parts have been brought together. This testing and assessment process is carried out so that the problem statement can be resolved, and it can be determined whether or not the limitation posed by previously published journals can be circumvented.

The first step in the testing and evaluation phase is to thoroughly test the system using a variety of test cases. This may include testing the system with known benign and malware samples, as well as new or unknown malware. It's important to use a diverse set of test cases to ensure that the system is able to detect a wide range of malware. Once the system has been tested, the next step is to evaluate its performance. This can be done by using metrics such as accuracy, precision, recall, and F1-score. It's also recommended to use a confusion matrix to understand the system's performance.

After the system's performance has been evaluated, any issues that are identified need to be addressed. This may involve fine-tuning the model, adjusting the parameters, or making changes to the system's architecture. After the issues have been addressed, the system should be retested and re-evaluated to ensure that the changes have improved its performance. It's important to compare the performance of the developed malware detection system with other systems, this comparison should be done using the same dataset and the same evaluation metrics.

Besides, collecting feedback from users and stakeholders who will be using the system, will help identify any additional issues or areas for improvement. And once the system has been thoroughly tested and evaluated, it can be deployed in a production environment. This may involve installing the system on the appropriate devices, training users on how to use the system, and monitoring the system's performance in the production environment.

The assertions that an accurate result may be obtained by using a detection system that uses a real-time process are one of the primary reasons for doing this test. Another reason is to prove the claimed optimum detection method. In addition, this stage gives researchers the opportunity to recognize the shortcomings and mistakes made during the research, which ultimately helps them achieve their goal of making additional advancements.

In addition, after carrying out the system with the detection technique geared toward authorization, a thorough discussion is carried out based on the result acquired from the experiment. This discussion is based on the findings. After that, a conclusion will be drafted based on the results, and it will be decided whether or not the hypothesis that this study was testing may be accepted.

The research that comprehensively details the entire procedure of carrying out this project has finally been finished. In order to demonstrate that the study objectives have been met, the outcome is analyzed and its significance is noted. In the following chapter, we will go over an additional description of the implementation step in full depth.

3.8 Conclusion

One of the conclusions that can be drawn from this chapter is that it was one of the subjects that assisted the researcher in deciding which model should be used in the investigation. In addition, the sort of methodology and the instruments that will be used have been discussed in this chapter so that the objectives of this thesis can be accomplished. Moreover, the full explanation of the method that will be utilized when conducting the research can be found in this chapter. In order for the researcher to accomplish what they set out to do with this investigation, they will need some essential components, such as computer hardware and software that can aid in the detection of malicious software. The next chapter will detail the implementation as well as the testing and evaluation that took place.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

This chapter will demonstrate how to put into action the methodology, planning, analysis, and design that were drafted in the previous chapter. The stage of implementation is the most important step throughout the entirety of the process of building the tools. This is due to the fact that we will now begin the process of identifying malicious software on Android devices by making use of the tools.

4.2 Dataset Description

4.2.1 Used Dataset

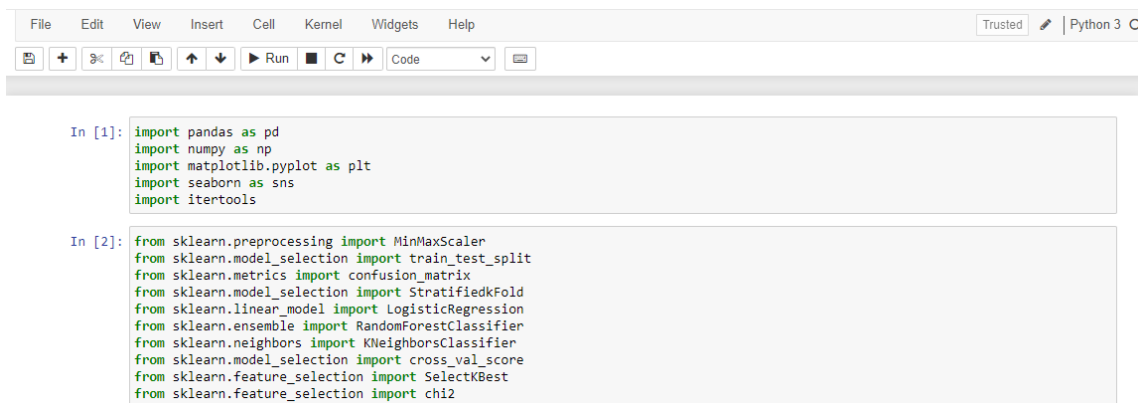
The collection of raw datasets is the initial step in the execution of this solution. It is necessary to have the dataset in order to guarantee the accuracy of the results. The dataset will provide additional details, explanations, and comprehension regarding the operations of the malware. The dataset is going to be analyzed, and the results are going to be used to try to anticipate or guess what will happen in future instances.

Labels	Size
Malware	10,000
Benign	10,000
Total	20,000

Table 4.1: Total AndroZoo Dataset

AndroZoo was the source of all of the data that was obtained. This chapter has compiled a total of 20,000 malicious features for android software, making it one of the largest collections of its kind. The information for this dataset came from a well-known

AndroZoo website as well as various Google search operators.[23] These category values, which are "Malware" and "Benign," have been converted to numerical values by substituting the values "1" and "0" for the values "Benign" and "Malware" correspondingly in the collected dataset. The categorical values are "Malware" and "Benign."



```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 C
+ ⌂ ↺ ↻ ⬆ ⬇ ▶ Run ⏹ ⌂ ▶ Code
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import itertools

In [2]: from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

Figure 4.1: Importing the Libraries in Jupyter Notebook

Accessing the datasets is accomplished with the help of *panda's* library, and graphs are produced with the use of visualization tools such as *matplotlib* and *seaborn*. The predictive modeling and other procedures that are connected to this will make use of the *sklearn* package.

```
df = pd.read_csv('AndroZooDataset.csv', engine='python')
```

Figure 4.2: Reading the datasets

The Jupyter Notebook needs to read the dataset on permissions. The permissions data will show total of 20,000 row and 31 columns in 'AndrooZooDataset.csv'.

```
print('The permissions data has {} row and {} columns'.format(df.shape[0], df.shape[1]))
The permissions data has 20000 row and 31 columns
```

```

pd.options.display.max_columns = 31
df.head()

```

	ACCESS_FINE_LOCATION	ACCESS_WIFI_STATE	CHANGE_WIFI_STATE	GET_ACCOUNTS	GET_TASKS	INSTALL_SHORTCUT	INTERNET	MEDIA_CONTENT_C
0	0	1	1	0	1	1	1	
1	1	0	0	1	1	0	1	
2	1	1	1	0	0	0	1	
3	1	1	0	0	0	0	1	
4	1	1	1	0	1	1	1	

Figure 4.3: Display the datasets

As the figure above, it showed that the dataset has been successfully read and can be displayed in a table using the Jupyter Notebook tool with a maximum 31 columns.

4.2.2 Data Cleaning

There shouldn't be any missing values in the data. If so, it is necessary to either delete the missing data or perform some kind of missing value imputation. In order to guarantee accurate and trustworthy findings during testing, data cleaning is a necessary phase in the machine learning process.

```

print(df.isnull().sum())

```

ACCESS_FINE_LOCATION	0
ACCESS_WIFI_STATE	0
CHANGE_WIFI_STATE	0
GET_ACCOUNTS	0
GET_TASKS	0
INSTALL_SHORTCUT	0
INTERNET	0
MEDIA_CONTENT_CONTROL	0
READ_PHONE_STATE	0
READ_SMS	0
RECEIVE_SMS	0
SEND_SMS	0
SET_ALARM	0
WAKE_LOCK	0
WRITE_EXTERNAL_STORAGE	0
ACCESS_NETWORK_STATE	0
BLUETOOTH	0
BLUETOOTH_ADMIN	0
BROADCAST_STICKY	0
CALL_PHONE	0
CAMERA	0
ACCESS_COARSE_LOCATION	0
READ_CALENDAR	0
READ_CALL_LOG	0
READ_CONTACTS	0
READ_EXTERNAL_STORAGE	0
RECORD_AUDIO	0
VIBRATE	0
WRITE_CALENDAR	0
WRITE_CALL_LOG	0
Class	0
dtype: int64	

Figure 4.4: Count the missing values in dataset

The process of finding and eliminating mistakes, inconsistencies, missing numbers, and outliers from the dataset is a part of it. Data quality and integrity significantly affect the effectiveness and validity of machine learning models, which highlights the need of data cleaning.

'**df.isnull().**' is used to identify and count the number of missing values in each column of a DataFrame, '**df**', is determined and counted using '**sum()**'. Each element in the DataFrame returned by the '**isnull()**' method is a boolean value that indicates whether it is a missing value (True) or not (False), and it has the same shape as the original DataFrame. The total number of missing values in each column can be determined by using the '**sum()**' function on the resulting DataFrame.

This code is particularly useful in data analysis and pre-processing tasks because missing values can have a significant impact on the accuracy and reliability of the results. Understanding the presence and distribution of missing values is crucial for deciding how to handle them effectively.[24]

```
print('The maximum number of missing values in any column is: {}'.format(df.isnull().sum().max()))  
The maximum number of missing values in any column is: 0
```

Figure 4.5: Data cleaning for checking the missing values

Then, we need to re-check again the data cleaning process for the dataset. By applying the code, '**print('The maximum number of missing values in any column is: {}'.format(df.isnull().sum().max()))**' that is used to determine the maximum number of missing values present in any column of a DataFrame, **df**, and display the result as a formatted string.

The maximum number of missing values in any column of the DataFrame **df** will be shown by executing this code. This information can be used to assess the data's general completeness and spot columns that have a lot of missing data. It enables researchers and data analysts to weigh the potential effects of missing values on the analysis and choose the best course of action, such as imputation or removing specific columns from further analysis.

4.2.3 Splitting Dataset

We separate the feature variables from the target variable through these processes, which is essential for supervised learning tasks like classification. In a typical machine learning scenario, the objectives are to create a model that can identify patterns and relationships between the features (X) and the target variable (y) in order to make predictions on new, unaltered data.

```
X = df.drop('Class', axis=1)
y = df['Class']
```

Figure 4.6: Separated the dataset into X and y

We establish the input and output elements necessary for training a machine learning model by putting the feature variables on X and the target variable on y. The remaining processes of model training and evaluation are made easier by this subsection of the model.

Following the data's division into X and y, model fitting can be done using algorithms that need distinct input features and target variables. During the training phase, we are able to give the model X as input and y as the desired outcome. Additionally, by contrasting the projected outputs of the model with the actual target values (y), this separation enables us to assess the model's effectiveness.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Figure 4.7: Splitting the dataset into Training and Testing

This code plays an essential part in the process of machine learning by partitioning the dataset into training and testing groups. By dividing the dataset into subgroups for training and testing, we can use the '**X_train**' and '**y_train**' datasets, train the machine learning model so that it can discover patterns and connections between the input features and the target variable. And contrasting the model's projected outputs with the actual target values (y_test), we can evaluate the model's effectiveness and generalizability using the X_test dataset.[25]

'X_train, X_test, y_train, y_test' This line of code assigns the split datasets to separate variables for further use:

- **'X_train'**: This variable contains the training set of feature variables. It represents the input data used to train the machine learning model.
- **'X_test'**: This variable contains the testing set of feature variables. It represents the unseen input data used to evaluate the trained model's performance.
- **'y_train'**: This variable contains the corresponding target variable for the training set. It represents the expected output values associated with the training set.
- **'y_test'**: This variable contains the corresponding target variable for the testing set. It represents the ground truth output values against which the model's predictions on the testing set will be compared.

Furthermore, the **'X'** is the feature variables or the input data and **'y'** is the target variable or the class labels associated with the feature variables. The **'test_size=0.3'** specifies the proportion of the data that should be allocated to the testing dataset. In this research, I have separated **30%** of the data will be used for testing, while the remaining **70%** will be used for training. And the **'random_state=42'** is a parameter that sets a random seed value to ensure reproducibility. By using the same random seed, we obtain the same random split each time we run the code. And this helps to achieve and produce a new results accuracy each time of testing.

4.3 Machine Learning Approach

To ensure that Android users are able to optimize the malware through the usage of the malware permission features method, a strategy based on machine learning is utilized. This strategy reduces the amount of time required for training and testing in order to identify malware.

First, the permission features of the malware detection were trained, and then, utilizing key features, they were categorized. In this study, the features selection method is utilized so that the significant features that are necessary for efficient malware detection can be chosen. Methods of feature selection are utilized to recognize and eliminate unnecessary and duplicated attribute data, both of which do not contribute in any way, shape, or form to the accuracy of a predictive model. As a direct result of this, the number of malicious software features has been cut down from the 107 permissions to just the Top 30 permission.[26] This is done in order to confirm that there is a distinct pattern emerging between the benign and the malicious. The list of authorization features that were employed in the study may be seen in the Table below.

Permission	Description
INTERNET	To allow an application to access a network socket
ACCESS_WIFI_STATE	To allow an application access to data stored in Wi-Fi networks
READ_PHONE_STATE	To allow read-only permission to the current phone state
WRITE_EXTERNAL_STORAGE	To allow all application access to writing on the external storage
GET_ACCOUNTS	To allow an access to the account list in the Account Service
GET_TASKS	To allow the application about the currently running and recently completed task
SEND_SMS	To allow an application to send Short Message Service (SMS)

RECEIVE_SMS	To allow an application to receive SMS messages
READ_SMS	To allow an application to read SMS messages
MEDIA_CONTENT_CONTROL	To allow an application to see what is currently being played
CHANGE_WIFI_STATE	To allow an application to toggle Wi-Fi on and off
SET_ALARM	To allow an application to set an alarm for the user
ACCESS_FINE_LOCATION	To allow an application to access the specific location
WAKE_LOCK	To allow the use of PowerManager WakeLocks to control hibernating processor or dimming screen
INSTALL_SHORTCUT	To allow an application to install a shortcut in the Launcher
ACCESS_NETWORK_STATE	To allow an application to access information about networks
BLUETOOTH	To allow an application connect to paired Bluetooth devices
BLUETOOTH_ADMIN	To allow applications to discover and pair Bluetooth devices and to make the device discoverable to other Bluetooth devices.

BROADCAST_STICKY	To allow an application to broadcast sticky intents
CALL_PHONE	To allow an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call.
CAMERA	Required to access the camera device
ACCESS_COARSE_LOCATION	To allow an application to access approximate location derived from network location sources such as cell towers and Wi-Fi
READ_CALENDAR	To allow an application to read the user's calendar data
READ_CALL_LOG	To allow an application to read the user's call log
READ_CONTACTS	To allow an application read the user's contacts data
READ_EXTERNAL_STORAGE	To allow grants read access to external storage
RECORD_AUDIO	To allow an application to record audio
VIBRATE	To allow an access to the vibrator
WRITE_CALENDAR	To allow an application to write the user's calendar data

WRITE_CALL_LOG	To allow an application to write (but not read) the user's call log data
----------------	--

Table 4.2: Top 30 of permission features

4.4 Evaluation and Results

The initial results display the outcomes that were produced by three different machine learning classifiers. These classifiers are Naïve Bayes, DecisionTable, J48, Random Forest, and Multi-Layer Perceptron. In addition, this research utilized the metrics of accuracy, FPR, precision, recall, and f-measure to investigate the various measurements that each classifier possessed. The results obtained from the 15 permission features of the testing set that made use of the three specified classifiers are presented in the table below.

Classifiers	Accuracy (%)	FPR	Precision	Recall	F-measure
Naïve Bayes	81.06%	0.323	0.746	0.943	0.833
DecisionTable	89.20%	0.100	0.899	0.884	0.891
J48	89.33%	0.098	0.900	0.885	0.893
Random Forest	90.45 %	0.095	0.905	0.901	0.903
Multi-Layer Perceptron	90.25 %	0.084	0.913	0.889	0.901

Table 4.3: Results for each machine learning classifier

The table presents the performance metrics of different classifiers, including Naïve Bayes, DecisionTable, J48, Random Forest, and Multi-Layer Perceptron, for malware detection in Android devices. Each classifier is evaluated based on its accuracy, false positive rate (FPR), precision, recall, and F-measure.

```

classifier = GaussianNB()
classifier.fit(X_train, y_train)

GaussianNB()

y_pred=classifier.predict(X_test)

# Evaluate the model on the testing set
accuracy = classifier.score(X_test, y_test)
print("Accuracy:", accuracy)

Accuracy: 0.8106666666666666

```

Figure 4.8: Accuracy testing for Naïve Bayes

Naïve Bayes achieved an accuracy of 81.06%, with a relatively high false positive rate of 0.323. It exhibited good precision (0.746) and recall (0.943), indicating that it can effectively classify malicious samples while having a relatively higher rate of false positives. A probabilistic classifier called Naive Bayes operates under the presumption of feature independence. It functions effectively in settings with a lot of features and uses computational resources effectively. However, it could have trouble capturing intricate connections and interactions between characteristics, increasing the potential of false positives.

```

classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)

DecisionTreeClassifier()

y_pred=classifier.predict(X_test)

# Evaluate the model on the testing set
accuracy = classifier.score(X_test, y_test)
print("Accuracy:", accuracy)

Accuracy: 0.892

```

Figure 4.9: Accuracy testing for DecisionTable

DecisionTable demonstrated an accuracy of 89.20%, with a lower false positive rate of 0.100. It achieved a balanced precision of 0.899 and recall of 0.884, resulting in an F-measure of 0.891. DecisionTable, sometimes referred to as decision rules or decision lists, classifies instances using a collection of if-then rules. It functions well in scenarios with discrete and categorical features and is interpreted and easily understood. Its great precision and recall are a result of the decision rules' direct feature interpretation capability.

```

classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)

DecisionTreeClassifier()

y_pred=classifier.predict(X_test)

# Evaluate the model on the testing set
accuracy = classifier.score(X_test, y_test)
print("Accuracy:", accuracy)

Accuracy: 0.8933333333333333

```

Figure 4.10: Accuracy testing for J48

J48, a decision tree algorithm, achieved an accuracy of 89.33% with a low false positive rate of 0.098. It demonstrated good precision (0.900) and recall (0.885), resulting in an F-measure of 0.893. J48 builds a model that resembles a tree to make choices based on feature values. Both cases with categorical and numerical characteristics work well, and it can manage intricate feature interactions. The algorithm's excellent accuracy, balanced precision and recall, and capacity to capture complex decision boundaries are all benefits.

```

classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)

RandomForestClassifier()

y_pred=classifier.predict(X_test)

# Evaluate the model on the testing set
accuracy = classifier.score(X_test, y_test)
print("Accuracy:", accuracy)

Accuracy: 0.9045

```

Figure 4.11: Accuracy testing for Random Forest

Random Forest achieved the highest accuracy among the classifiers, at 90.45%, with a low false positive rate of 0.095. It exhibited a precision of 0.905, recall of 0.901, and an F-measure of 0.903. Multiple decision trees are combined in the ensemble learning technique known as Random Forest. It makes use of decision trees' advantages while minimising their specific drawbacks, such as overfitting. It achieves excellent accuracy and a balanced trade-off between precision and recall by combining predictions from various trees.

```
classifier = MLPClassifier()
classifier.fit(X_train, y_train)

C:\Users\Riqy\anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:614: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
MLPClassifier()

y_pred=classifier.predict(X_test)

# Evaluate the model on the testing set
accuracy = classifier.score(X_test, y_test)
print("Accuracy:", accuracy)

Accuracy: 0.9025
```

Figure 4.12: Accuracy testing for Multi-Layer Perceptron

Multi-Layer Perceptron attained an accuracy of 90.25%, with a relatively low false positive rate of 0.084. It demonstrated the highest precision among the classifiers, at 0.913, and a recall of 0.889, resulting in an F-measure of 0.901. An artificial neural network called a multi-layer perceptron has numerous layers of interconnected neurons. It can recognise nonlinear correlations between features and learn complex patterns. The model's excellent precision and comparatively low false positive rate are both a result of its capacity to extract complex feature representations.

Based on the results, Random Forest achieved the highest accuracy (90.45%), making it the top-performing classifier in terms of overall classification performance. It balances precision, recall, and false positive rate effectively. Naïve Bayes, on the other hand, achieved the lowest accuracy (81.06%) due to its higher false positive rate. Despite its lower accuracy, Naïve Bayes demonstrated a higher recall, indicating its effectiveness in detecting true positive instances. However, for malware detection purposes, a higher false positive rate can be undesirable, making Random Forest the preferred choice due to its superior overall performance and lower false positive rate.

PERCENTAGE OF ACCURACY

for each machine learning classifiers

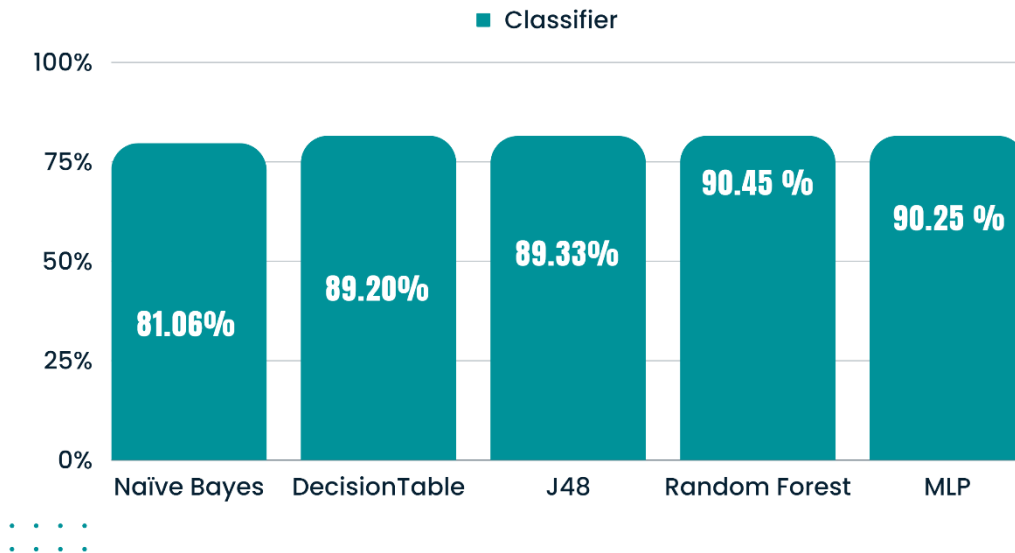


Figure 4.13: Level of accuracy for each classifier

According to the figure above, Random Forest classifiers produced the best accuracy result, which was 90.45 percent, in comparison to MLP, which achieved 90.25 percent, J48 get 89.33 percent, DecisionTable achieved 89.20 percent, and Naïve Bayes, which achieved the lowest accuracy result, which was 81.06 percent.

4.4.1 Confusion matrix

A method for analysing and summarising the performance of a classification model is known as a confusion matrix. The table that follows outlines two categories of probable predictions: benign and malicious software. The confusion matrix consists of four categories based on the predictions made by the classifier and the actual labels of the test dataset:

- True Positives (TP): The classifier correctly predicted that an app is malicious.
- True Negatives (TN): The classifier correctly predicted that an app is benign.

- False Positives (FP): The classifier incorrectly predicted that an app is malicious, when it is actually benign. This is also known as a Type I error or a false positive.
- False Negatives (FN): The classifier incorrectly predicted that an app is benign, when it is actually malicious. This is also known as a Type II error or a false negative.

The information in the confusion matrix can be used to calculate various performance metrics for the classifier, such as accuracy, precision, recall, and F1-score. These metrics can help to determine whether the classifier is performing well or not, and can be used to improve the performance of the model.[27] For instance, if a model predicts the presence of malware activity, the result will display "malware" and "benign". The results of the five different classifiers are presented.

Classifiers	Actual	Prediction	
		Malware	Benign
Naïve Bayes	Actual Malware	8326	1674
	Actual Benign	1406	8594
DecisionTable	Actual Malware	8873	1127
	Actual Benign	969	9031
J48	Actual Malware	9088	912
	Actual Benign	852	9148
Random Forest	Actual Malware	9328	672
	Actual Benign	696	9304

Multi-Layer Perceptron	Actual Malware	9173	827
	Actual Benign	815	9185

Table 4.4: Confusion matrix of classifiers

Looking at the confusion matrix classifiers table, we can see that each row represents the instances in the actual class, while each column represents the instances in the predicted class. The cells of the matrix represent the count of true positives, false positives, false negatives, and true negatives.

For the actual malware, all the classifiers correctly identified a large number of malware instances, as shown by the high true positive counts. However, there are also some false negatives, indicating that some malware instances were incorrectly classified as benign. Naïve Bayes had the highest false negative count with total of 1674, while DecisionTable get 1127, J48 get 912, Random Forest with 672, and Multi-Layer Perceptron get 827 number of false negatives.

Besides, for the actual benign instances, all of the classifiers correctly identified a significant number of benign instances, as shown by the high true negative counts. However, there are also some false positives, indicating that some benign instances were incorrectly classified as malware. Random Forest had the highest false positive count with total of 9304, while Multi-Layer Perceptron get 9185, J48 get 9148, DecisionTable with 9031, and Naïve Bayes get the lowest with 8594 number of false positives.

All of the classifiers perform fairly well when identifying occurrences as malware or benign. However, among the classifiers, Random Forest and Multi-Layer Perceptron show the highest accuracy. In comparison to the other classifiers, Random Forest has the best accuracy (90.4%) and the lowest false positive rate. This shows that it can correctly identify a greater variety of cases as malware or benign. The Random Forest model performs well in properly forecasting both positive and negative instances, as seen by its excellent precision, recall, and F1-score values.

Naïve Bayes, on the other hand, has a larger false positive rate and a lower accuracy of 81.06%. This implies that it might mistakenly label some innocent occurrences as malware. The fact that it still exhibits a respectable level of precision, recall, and F1-score values, demonstrating its capacity to accurately identify a sizable number of cases, says a lot about its accuracy.

4.4.2 Receiver operating characteristics curve (ROC)

In this particular investigation, the processes were separated into two categories: benign and malware based on the authorization features. Along with the performance matrix, we are also calculating the receiver operating characteristics (ROC) curve for each of the machine learning classifiers. During this stage of the process, the TPR is considered to be the detection rate that accurately predicted the malware process, whereas the FPR was chosen as the detection rate that incorrectly predicted normal as malware. The curves of five different machine learning classifiers were displayed in Figure 4.1.

The ROC curve helps to evaluate the trade-off between the sensitivity (TPR) and specificity ($1 - \text{FPR}$) of a classifier system. A perfect classifier would have a ROC curve that passes through the point (0,1), indicating that it has a TPR of 1 and an FPR of 0, regardless of the threshold setting. A random classifier, on the other hand, would have a ROC curve that is a straight line passing through the point (0,0) and (1,1).[28] The AUC results determined that it was possible to measure whether the detection strategy was effective or ineffective. A score of 1 represented an absolutely accurate prediction, whereas a score 0.5 suggested an inaccurate prediction. The results are summarised in Table 4.5.

Classifiers	AUC	Prediction
Naïve Bayes	0.918	Accurate Prediction
DecisionTable	0.949	Accurate Prediction
J48	0.954	Accurate Prediction
Random Forest	0.979	Accurate Prediction
Multi-Layer Perceptron	0.961	Accurate Prediction

Table 4.5: AUC Classifiers Result

4.4.2.1 Naïve Bayes (ROC)

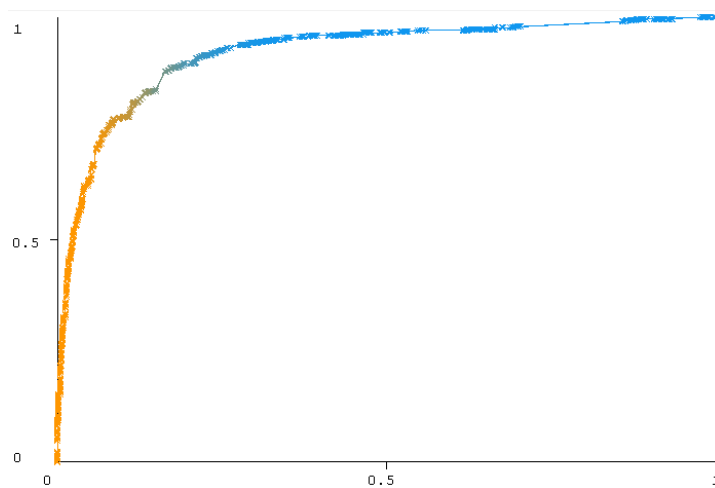


Figure 4.14: Naïve Bayes ROC Curves

The Naive Bayes classifier has an AUC value of 0.918, which indicates that it performs better than random but is not a perfect classifier. This means that it can correctly classify a relatively high percentage of instances, but it may still make some errors.

4.4.2.2 DecisionTable (ROC)

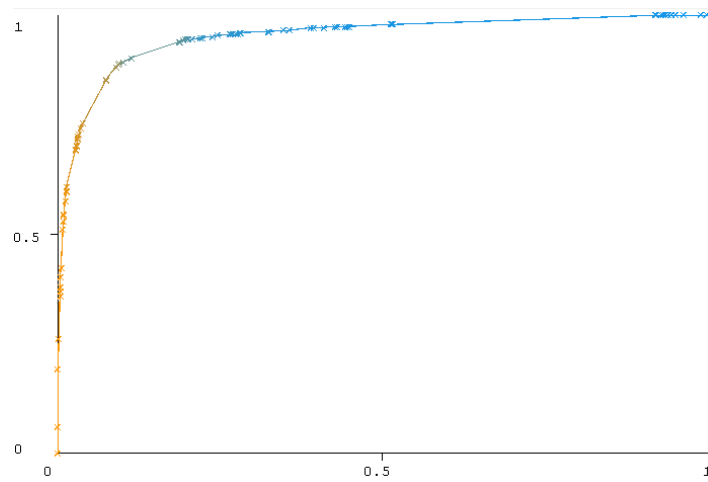


Figure 4.15: DecisionTable ROC Curves

The DecisionTable classifier has an AUC value of 0.949, which is higher than the Naive Bayes classifier. This indicates that it performs better than the Naive Bayes classifier and is closer to being a perfect classifier.

4.4.2.3 J48 (ROC)

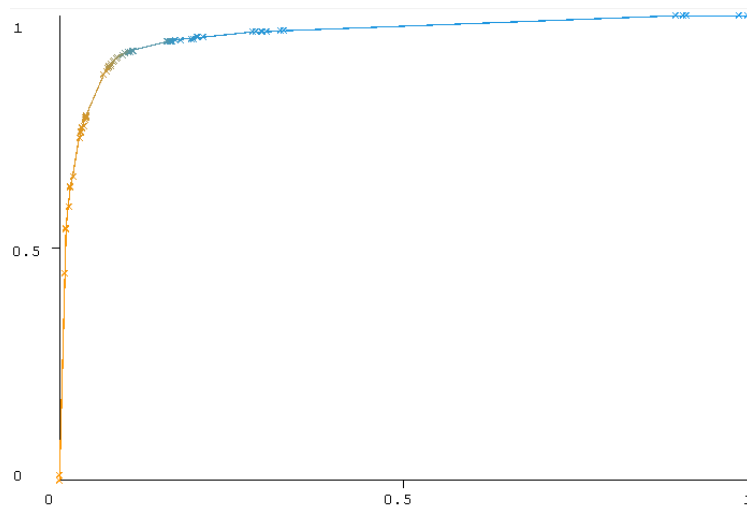


Figure 4.16: J48 ROC Curves

The J48 classifier has an AUC value of 0.954, which is lower than the DecisionTable classifier. This suggests that it performs worse than the DecisionTable classifier, but it is still better than a random classifier.

4.4.2.4 Random Forest (ROC)

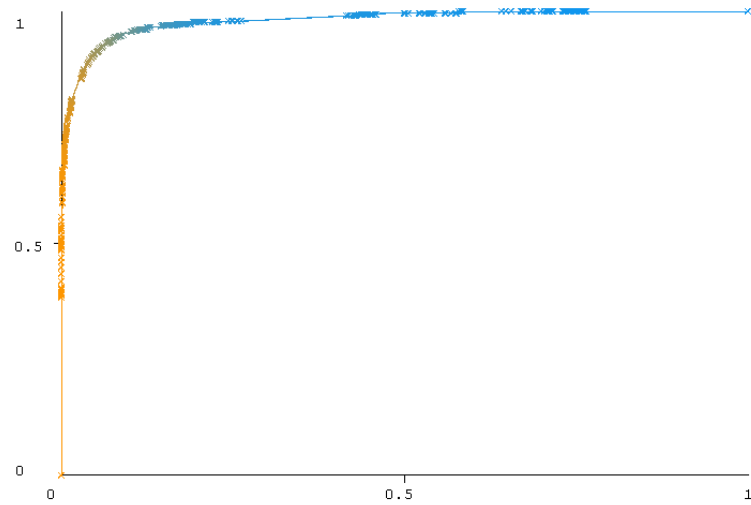


Figure 4.17: Random Forest ROC Curves

The Random Forest classifier has an AUC value of 0.979, which is similar to the DecisionTable classifier. This indicates that it performs well and is closer to being a perfect classifier.

4.4.2.5 Multi-Layer Perceptron (ROC)

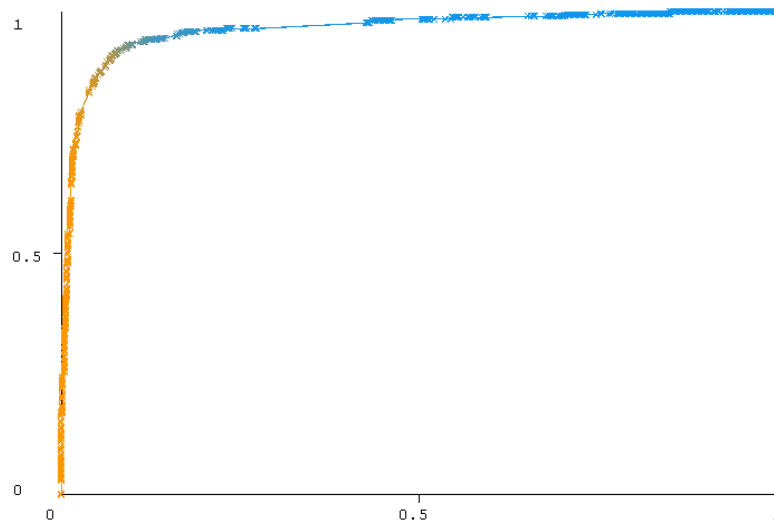


Figure 4.18: Multi-Layer Perceptron ROC Curves

The Multi-Layer Perceptron classifier also has an AUC value of 0.961, which is similar to the Random Forest classifier. This suggests that it also performs well and is closer to being a perfect classifier.

In general, we can see that the DecisionTable, Random Forest, and Multi-Layer Perceptron classifiers have higher AUC values than the Naive Bayes and J48 classifiers, which indicates that they perform better in general when applied to this particular problem.

4.4.3 Performance of Classifiers Build model

During the training phase, the performance of the classifiers is evaluated on a training dataset to ensure that they are learning and improving over time. This is typically done by measuring metrics such as accuracy, precision, recall, F-measure, and AUC. These metrics help to assess the quality of the classifiers and identify any areas for improvement.

And for the testing phase, the performance of the classifiers is evaluated on a separate testing dataset to ensure that they can generalize to new, unseen data. This is important because classifiers that perform well on the training dataset may not necessarily perform well on new data.

Classifiers	Time Taken to Build Model
Naïve Bayes	0.13 seconds
DecisionTable	4.36 seconds
J48	2.45 seconds
Random Forest	9.98 seconds
Multi-Layer Perceptron	67.23 seconds

Table 4.6: Time taken to produce model (seconds)

In the table provided, we can see that the Naïve Bayes classifier has the shortest build time of 0.13 seconds, while the Multi-Layer Perceptron classifier has the longest build time of 67.23 seconds. This suggests that Naïve Bayes may be more efficient and scalable for smaller datasets or less complex models, while Multi-Layer Perceptron may be more suitable for larger datasets or more complex models. However, the specific trade-offs between build time and performance will depend on the specific requirements and constraints of the application.

Classifiers	Time Taken to Test Model
Naïve Bayes	0.32 seconds
DecisionTable	0.17seconds
J48	0.14 seconds
Random Forest	2.02 seconds
Multi-Layer Perceptron	0.15 seconds

Table 4.7: Time taken to test model (seconds)

The time taken to test the model is also an important factor to consider in machine learning. It indicates how quickly the model can make predictions on new data. In practical applications, the time taken to make predictions can be a critical factor, especially in real-time systems where predictions need to be made quickly.

By looking at the table above, we can see that the J48 classifier has the shortest time taken to test the model, followed by DecisionTable and Multi-Layer Perceptron. The Naive Bayes and Random Forest classifiers have slightly longer times to test the model, but they are still relatively fast.

CHAPTER 5

CONCLUSION

5.1 Introduction

Smartphones powered by Google's Android operating system are now practically indispensable in the highly sophisticated world of today.[29] The way in which we engage with technology is being fundamentally altered as a result of the diversity of capabilities and pursuits offered by these gadgets. Smartphones have become vital tools for personal and professional convenience, being used for everything from socialising and online shopping to accessing the internet for information, downloading files and games, providing financial services, and entertaining oneself.

Our day-to-day lives have been completely transformed by the simplicity and convenience that our cell phones and the internet. Effortless and speedy completion of tasks that, in the past, demanded a large investment of both time and energy is now possible. The capabilities of smartphones seem to have almost no ceiling, whether we're talking about the ability to interact with friends and family through social media, stream our favourite films and television shows, or complete financial transactions online.

However, despite the fact that smartphones provide users with an extensive range of advantages and benefits, there is also an inherent security risk associated with their use that the majority of users frequently fail to recognise. Malicious software, more frequently referred to as malware, poses a substantial risk to the safety and privacy of those who use smartphones. Malware refers to a broad category of malicious software that can take many different forms, including viruses, worms, Trojan horses, adware, spyware, and ransomware. These dangerous programmes are able to penetrate cell phones through applications that are infected, websites that have been compromised, or other means. Once inside, they can compromise important information, interrupt device performance, and even cause financial losses.

5.2 Research Objective Revisited

It is absolutely necessary, in order to guarantee the safety of Android devices, to have an understanding of the nature of malware and to create efficient solutions for the detection and prevention of it. The overarching goal of this project is to make a contribution to the field of cybersecurity by concentrating on the particular challenge of "Malware Detection in Android Using Machine Learning."

In order to find a solution to this problem, the research will compile a complete dataset of malware. Permissions are the privileges provided to programmes on a smartphone in order for them to access particular resources or carry out particular operations on the device. By analysing these permissions, it is possible to uncover patterns and characteristics that can differentiate between benign and malicious software. This is an important step in preventing security breaches.

Objective 1: To study the current issue with the Android malware detection system.

The first objective of my research is to investigate the current issues and challenges in the Android malware detection system. Through an in-depth study, I aimed to gain a comprehensive understanding of the existing problems and explore potential solutions to enhance the effectiveness of malware detection on Android devices. To achieve this objective, I extensively studied and analyzed the works published in online scholarly journals and some research paper.

Chapter 2 of my research served as a critical component in accomplishing this objective. Within this chapter, I presented a detailed overview of the malware detection system, covering various aspects such as the classification of malware detection techniques and the utilization of machine learning approaches. Additionally, I delved into the different algorithms employed in the field.

By conducting this thorough review, my research aimed to gather valuable insights and knowledge about the current state of the Android malware detection system. This served as the foundation for further investigation and analysis in subsequent chapters of the study.

In essence, Chapter 2 played a crucial role in achieving the objective of reviewing the issues surrounding the Android malware detection system. It served as a comprehensive resource, presenting key information and insights regarding the classification of malware detection techniques, the utilization of machine learning approaches, and the algorithms employed in the field. This comprehensive review laid the groundwork for the subsequent phases of my research and contributed to a deeper understanding of the challenges and opportunities in Android malware detection.

Objective 2: To analyze the technique for Machine Learning that will be used to construct an Android malware detection system.

The second objective of my research was to analyze the technique for Machine Learning that will be used to construct an Android malware detection system. I aimed to explore and evaluate various machine learning approaches and algorithms to identify the most suitable ones for detecting and classifying malware on Android devices. To accomplish this objective, I utilized the Jupyter Notebook software to conduct an experiment and testing thorough the evaluation of the system.

Chapter 4 of my research was crucial for accomplishing this objective. In this chapter, I conducted a step-by-step experiment in Jupyter Notebook to evaluate the usefulness of the Android malware detection system. Accuracy, False Positive Rate (FPR), True Positive Rate (TPR), precision (ability to correctly classify malware instances), recall (ability to identify all malware instances), and f-measure (a combined measure of precision and recall) were used to gain a solid experiment result.

The findings of my research, presented in Chapter 4, offered a spotlight on its effectiveness of the Android malware detection system. By analyzing the results obtained from the evaluation measures, I was able to determine the system's accuracy, its ability to minimize false positives, and its capability to correctly identify malware instances. This evaluation process provided a quantitative assessment of the system's performance and served as a basis for further improvements and enhancements.

I conducted experiments and evaluations to assess the performance and accuracy of these machine learning algorithms in detecting Android malware. By utilizing a carefully curated dataset of malware permissions, I trained and tested the algorithms to measure their effectiveness in accurately identifying malicious software.

Objective 3: To assess the effectiveness of the Android malware detection system in terms of its ability to identify malicious software.

The third objective of my research was to assess the effectiveness of the Android malware detection system in terms of its ability to identify malicious software. This evaluation involved assessing the performance of five different classifiers. Based on the results obtained, it was found that the random forest classifier achieved the highest accuracy percentage in detecting Android malware.

During my research, I conducted a thorough evaluation of the performance in terms of its ability to accurately detect malware. This evaluation process involved comparing the performance of multiple classifiers to identify the most effective one.

By analyzing the results obtained from the evaluation, it was determined that the random forest classifier demonstrated the highest level of accuracy in detecting Android malware. This precision and recall allowed me to evaluate the system's performance in terms of minimizing false positives (identifying benign apps as malware) and false negatives (failing to detect actual malware), respectively. These metrics provided insights into the system's ability to strike a balance between accurate malware detection and avoiding false alarms. This finding highlights the efficacy of the random forest algorithm in accurately identifying malicious software on Android devices.

Furthermore, I explored the receiver operating characteristic (ROC) curve and calculated the area under the curve (AUC) to assess the system's performance across different thresholds. The ROC curve provided a visual representation of the system's ability to trade-off between true positive rate and false positive rate, allowing for a more comprehensive understanding of its performance characteristics.

Through my research, I obtained valuable findings regarding the effectiveness of the Android malware detection system. I observed that the system demonstrated promising performance in accurately identifying malicious software, with high accuracy rates and satisfactory precision and recall scores. The AUC analysis also indicated that the system had a favourable trade-off between true positive rate and false positive rate, indicating its capability to effectively distinguish between malware and benign applications.

5.3 Achievement of the study

Throughout the course of my research, I have achieved several significant milestones and made noteworthy contributions in the field of Android malware detection using machine learning. I've been conducted an extensive review of the existing literature and scholarly works related to Android malware detection. This review provided a solid foundation of knowledge and helped identify the key issues and challenges in the field. And help me gained a deep understanding of the various techniques and methodologies employed in malware detection, particularly in the context of Android devices. This understanding allowed me to identify the limitations of current approaches and explore potential solutions. These are some of the achievements during my research:

5.3.1 Choosing a relevant dataset to be used

Using a relevant dataset allows for accurate evaluation of the performance of the malware detection system. By employing a dataset that closely resembles the distribution of malware in the wild, researchers can assess the system's accuracy, precision, recall, and other performance metrics with greater confidence.

This study teaches me to get a relevant dataset that reflects real-world scenarios and encompasses a wide range of Android applications, including both legitimate and malicious ones. By using a dataset that accurately represents the diversity of applications encountered by users, the research findings can be more applicable and reliable.

5.3.2 Understand the detection approaches and techniques

Understanding detection approaches and techniques enables me to properly evaluate the performance of my models. By comparing different algorithms and methodologies, I can successfully assess the strengths and weaknesses of each approach and make informed decisions about which techniques are most effective on the specific testing and experiments.

The field of machine learning is constantly evolving, with new algorithms and techniques being developed regularly. Having a deep understanding of detection approaches and techniques allows me to stay abreast of the latest advancements and adapt the detection systems accordingly. This adaptability ensures that the system remains effective and relevant as new types of malware and attack techniques emerge.

5.3.3 Find the best machine learning classifiers

Start by defining the evaluation metrics that will be used to assess the performance of the classifiers. This selection needs to be based on prior research, expert knowledge, or commonly used classifiers in the field. The dataset will be divided into training, validation, and testing sets. The training set is used to train the classifiers, the validation set is used to fine-tune hyperparameters and optimize the model, and the testing set is used to evaluate the final performance.

Different classifiers have varying performance characteristics. By identifying the best classifiers, I can definitely improve the overall performance of my machine learning models. The best classifiers are those that demonstrate higher accuracy, precision, recall, or other relevant metrics, leading to more reliable and effective predictions or classifications.[30] This helps in advancing the field of machine learning by establishing standards and best practices, enabling researchers and practitioners to build upon previous work and make informed decisions about the most effective techniques.

5.4 Limitations

Limitations are important to acknowledge as they can impact the validity, reliability, generalizability, and applicability of the research findings. During the progress of my research on Android malware detection using machine learning, there are several limitations that have been often face:

5.4.1 Availability and Quality of Datasets

Obtaining high-quality and diverse datasets for malware detection can be challenging. Limited access to comprehensive and up-to-date datasets may restrict the generalizability and effectiveness of your research. Additionally, ensuring the accuracy and reliability of labelled malware samples can be difficult, as manual labelling may be subjective and prone to errors.

5.4.2 Imbalanced Sample Size

Imbalance between the number of malware samples and benign samples that created a total of 10,000 in the dataset is a common issue in malware detection research. Imbalanced datasets can lead to biased model performance, where the classifier may favor the majority class (benign) and struggle to accurately detect the minority class (malware).

5.4.3 Interpretability and Explain ability of Machine Learning

Machine learning models, particularly complex ones, often lack interpretability and explain ability. Understanding the decision-making process of these models and providing insights into why a certain classification is made can be difficult. Interpretable models may sacrifice some accuracy, while highly accurate models may lack transparency, making it challenging to gain user trust and acceptance.

5.5 Future Works

Future works are crucial as they contribute to the advancement of knowledge, address existing gaps, and pave the way for further research and development. These are the areas to identify further investigation, improvement, or expansion of the existing work as follow:

5.5.1 Incorporating New Malware Techniques

As malware techniques continue to evolve, it is essential to stay updated and adapt the detection system accordingly. Investigate emerging malware trends and explore innovative approaches to effectively detect and mitigate new malware threats.

5.5.2 Deep Learning Approaches

Explore the application of deep learning techniques, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), for Android malware detection. Deep learning models have shown promising results in various domains and can potentially capture intricate patterns and representations within malware samples.

5.5.3 Real-Time Detection on Mobile Devices

Develop efficient and lightweight models suitable for real-time detection on resource-constrained mobile devices. Consider optimizing the models for low power consumption and minimal computational requirements without compromising the detection accuracy.

5.5.4 Adversarial Attack Detection

Investigate the robustness of the malware detection system against adversarial attacks. Adversarial attacks involve manipulating or crafting malicious inputs specifically designed to evade detection. Develop techniques to detect and defend against such attacks to ensure the reliability and effectiveness of the system.

5.6 Conclusion

The elimination of malicious software often depends on the application of machine learning methods. This research makes use of a number of well-known classifiers, such as Naïve Bayes, DecisionTable, J48, Random Forest, and Multi-Layer Perceptron. These classifiers are successfully able to recognise and categorise dangerous software because they are trained with the optimised dataset and take into account a variety of characteristics.

The results of this study have the potential to make important contributions to the field of cybersecurity when taken together with its other findings. We can better safeguard Android users from the ever-evolving threats posed by malicious software if we improve our understanding of malware and enhance the capabilities of detection systems that are based on machine learning. Not only will the development of more efficient detection methods protect personal data and privacy, but it will also ensure the smooth and secure operation of Android smartphones in our day-to-day life.

Moreover, the broad adoption of Android smartphones has offered an incredible amount of ease and completely revolutionised the way in which we live our lives and engage with technology. Having said that, this ease does come with security dangers, most notably those posed by malware. The objective of the research project titled "Malware Detection in Android Using Machine Learning" is to find a solution to this problem by making use of machine learning algorithms and improving the quality of datasets in order to increase the reliability and efficiency of malware detection. This project aims to make a contribution to the field of cybersecurity and provide users with a safer and more secure experience when using their smartphones by concentrating on the detection and prevention of dangers posed by malware on Android devices.

By continuously refining detection techniques, incorporating new malware trends, and considering real-world deployment scenarios, the development of more effective and reliable malware detection systems becomes possible, ultimately ensuring the security and integrity of Android users' devices and data.

REFERENCES

- [1] S. Talukder and Z. Talukder, “A Survey on Malware Detection and Analysis Tools,” *International Journal of Network Security & Its Applications*, vol. 12, no. 2, pp. 37–57, Mar. 2020, doi: 10.5121/IJNSA.2020.12203.
- [2] “What is malware and how cybercriminals use it | McAfee.” <https://www.mcafee.com/en-my/antivirus/malware.html> (accessed Jan. 22, 2023).
- [3] “What is malware? Definition and how to tell if you’re infected | Malwarebytes.” <https://www.malwarebytes.com/malware> (accessed Jan. 22, 2023).
- [4] J. Senanayake, H. Kalutarage, and M. O. Al-Kadri, “Android Mobile Malware Detection Using Machine Learning: A Systematic Review,” *Electronics 2021, Vol. 10, Page 1606*, vol. 10, no. 13, p. 1606, Jul. 2021, doi: 10.3390/ELECTRONICS10131606.
- [5] “Understanding Android Malware Families: Adware and Backdoor (Article 5) - IT World Canada.” <https://www.itworldcanada.com/blog/understanding-android-malware-families-part-5-adware-backdoor/447798> (accessed Jan. 22, 2023).
- [6] “PUP: Potentially unwanted program / PUA: potentially unwanted application – How you can protect yourself.” <https://www.kaspersky.com/resource-center/definitions/what-is-pup-pua> (accessed Jan. 22, 2023).
- [7] “Ransomware explained: How it works and how to remove it | CSO Online.” <https://www.csoonline.com/article/3236183/what-is-ransomware-how-it-works-and-how-to-remove-it.html> (accessed Jan. 22, 2023).
- [8] “What Is a Trojan Horse? Trojan Virus and Malware Explained | Fortinet.” <https://www.fortinet.com/resources/cyberglossary/trojan-horse-virus> (accessed Jan. 22, 2023).
- [9] O. Aslan and R. Samet, “A Comprehensive Review on Malware Detection Approaches,” *IEEE Access*, vol. 8, pp. 6249–6271, 2020, doi: 10.1109/ACCESS.2019.2963724.
- [10] “What is Signature-Based Detection? — Techslang.” <https://www.techslang.com/definition/what-is-signature-based-detection/> (accessed Jan. 22, 2023).

- [11] M. Norouzi, A. Souri, and M. Samad Zamini, “A Data Mining Classification Approach for Behavioral Malware Detection,” *Journal of Computer Networks and Communications*, vol. 2016, 2016, doi: 10.1155/2016/8069672.
- [12] M. Alhebsi, “Android Malware Detection using Machine Learning Techniques,” *Theses*, Apr. 2022, Accessed: Jan. 22, 2023. [Online]. Available: <https://scholarworks.rit.edu/theses/11178>
- [13] J. Mohamad Arif, M. F. Ab Razak, S. Awang, S. R. Tuan Mat, N. S. N. Ismail, and A. Firdaus, “A static analysis approach for Android permission-based malware detection systems,” *PLoS One*, vol. 16, no. 9, p. e0257968, 2021, doi: 10.1371/JOURNAL.PONE.0257968.
- [14] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, “A Review of Android Malware Detection Approaches Based on Machine Learning,” *IEEE Access*, vol. 8, pp. 124579–124607, 2020, doi: 10.1109/ACCESS.2020.3006143.
- [15] “(PDF) Android Malware Detection System using Machine Learning.” https://www.researchgate.net/publication/348150130_Android_Malware_Detection_System_using_Machine_Learning?enrichId=rgreq-7ae277dd6fb411bdcf2e219bf47305df-XXX&enrichSource=Y292ZXJQYWdlOzM0ODE1MDEzMDtBUzo5NzU0NDQyMzcxMTUzOTRAMTYwOTU3NTQzNzYyNg%3D%3D&el=1_x_3&_esc=publicationCoverPdf (accessed Jan. 22, 2023).
- [16] Y. Lin *et al.*, “Dataset Bias in Android Malware Detection,” May 2022, doi: 10.48550/arxiv.2205.15532.
- [17] “AndroZoo: Collecting Millions of Android Apps for the Research Community | IEEE Conference Publication | IEEE Xplore.” <https://ieeexplore.ieee.org/document/7832927> (accessed Jan. 22, 2023).
- [18] A. Mahindru and A. L. Sangal, “MLDroid—framework for Android malware detection using machine learning techniques,” *Neural Comput Appl*, vol. 33, no. 10, pp. 5183–5240, May 2021, doi: 10.1007/S00521-020-05309-4/TABLES/37.
- [19] Q. E. Alahy, M. N. U. R. Chowdhury, H. Soliman, M. S. Chaity, and A. Haque, “Android Malware Detection in Large Dataset: Smart Approach,” *Advances in Intelligent Systems and Computing*, vol. 1129 AISC, pp. 800–814, 2020, doi: 10.1007/978-3-030-39445-5_58.

- [20] S. Ledesma, M. A. Ibarra-Manzano, E. Cabal-Yepez, D. L. Almanza-Ojeda, and J. G. Avina-Cervantes, "Analysis of Data Sets With Learning Conflicts for Machine Learning," *IEEE Access*, vol. 6, pp. 45062–45070, Aug. 2018, doi: 10.1109/ACCESS.2018.2865135.
- [21] M. R. Keyvanpour, M. Barani Shirzad, and F. Heydarian, "Android malware detection applying feature selection techniques and machine learning," *Multimed Tools Appl*, vol. 82, no. 6, pp. 9517–9531, Mar. 2023, doi: 10.1007/S11042-022-13767-2.
- [22] M. F. A. Razak, N. B. Anuar, R. Salleh, and A. Firdaus, "The rise of 'malware': Bibliometric analysis of malware study," *Journal of Network and Computer Applications*, vol. 75, pp. 58–76, Nov. 2016, doi: 10.1016/J.JNCA.2016.08.022.
- [23] Y. N. Maddineni, C. Nagesh, B. Prasad, C. L. Reddy, and M. Tech Scholar, "A GENETIC ALGORITHM FOR ANDROID MALWARE DETECTION WITH OPTIMIZED FEATURE SELECTION USING MACHINE LEARNING APPROACH," vol. 12, 2021, Accessed: Jun. 12, 2023. [Online]. Available: www.jespublication.com
- [24] J. Li *et al.*, "Feature selection: A data perspective," *ACM Comput Surv*, vol. 50, no. 6, Dec. 2017, doi: 10.1145/3136625.
- [25] F. Yang, Y. Zhuang, and J. Wang, "Android malware detection using hybrid analysis and machine learning technique," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10603 LNCS, pp. 565–575, 2017, doi: 10.1007/978-3-319-68542-7_48.
- [26] O. Yildiz and I. A. Doğru, "Permission-based Android Malware Detection System Using Feature Selection with Genetic Algorithm," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 2, pp. 245–262, Feb. 2019, doi: 10.1142/S0218194019500116.
- [27] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science (1979)*, vol. 349, no. 6245, pp. 255–260, Jul. 2015, doi: 10.1126/SCIENCE.AAA8415.
- [28] B. Amro, "Malware Detection Techniques for Mobile Devices," *International Journal of Mobile Network Communications & Telematics*, vol. 7, no. 4/5/6, pp. 01–10, Dec. 2017, doi: 10.5121/IJMNCT.2017.7601.
- [29] K. H. Tae, Y. Roh, Y. H. Oh, H. Kim, and S. E. Whang, "Data Cleaning for Accurate, Fair, and Robust Models: A Big Data - AI Integration Approach," *Proceedings of the 3rd International*

Workshop on Data Management for End-to-End Machine Learning, Jun. 2019, doi:
10.1145/3329486.3329493.

- [30] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, “A Survey of Android Malware Detection with Deep Neural Models,” *ACM Comput Surv*, vol. 53, no. 6, Feb. 2021, doi:
10.1145/3417978.