

# Android Malware Detection and Categorization using Machine Learning



By

**Mudassar Waheed**

**Fall 2018-MS(IS) - 00000276640**

Supervisor

**Dr. Sana Qadir**

**Department of Computing**

A thesis submitted in partial fulfillment of the requirements for the degree  
of Masters of Science in Information Security (MS IS)

In

School of Electrical Engineering and Computer Science,  
National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(June 2022)

## THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "Android malware detection and identification" written by Mudassar Waheed, (Registration No 00000276640), of SEecs has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: \_\_\_\_\_  \_\_\_\_\_

Name of Advisor: \_\_\_\_\_ **Dr. Sana Qadir** \_\_\_\_\_

Date: \_\_\_\_\_ **02-Jul-2022** \_\_\_\_\_

HoD/Associate Dean: \_\_\_\_\_

Date: \_\_\_\_\_

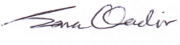
Signature (Dean/Principal): \_\_\_\_\_

Date: \_\_\_\_\_

## Approval

It is certified that the contents and form of the thesis entitled "Android malware detection and identification" submitted by Mudassar Waheed have been found satisfactory for the requirement of the degree

Advisor : Dr. Sana Qadir

Signature:  \_\_\_\_\_

Date: 02-Jul-2022

Committee Member 1: Dr. Dr Hasan Tahir

Signature:  \_\_\_\_\_

03-Jul-2022

Committee Member 2: Dr. Razi Arshad

Signature:  \_\_\_\_\_

Date: 02-Jul-2022

Signature: \_\_\_\_\_

Date: \_\_\_\_\_


# Dedication

I dedicate this research to my **family** and **parents**, who have made so many sacrifices for me during my life, believed in my choices, and allowed me the opportunity to live my own life.

## Certificate of Originality

I hereby declare that this submission titled "Android malware detection and identification" is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEecs or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEecs or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics, which has been acknowledged. I also verified the originality of contents through plagiarism software.

Student Name: Mudassar Waheed

Student Signature:  \_\_\_\_\_

# Acknowledgment

First of all, I am very grateful to Allah S.W.T for granting me the ability, skills and willpower to conclude this research work. After that, I want to thank my supervisor, Dr. Sana Qadir, for having confidence in me and providing me with constant direction and support from the beginning to the end of this work. Without her, this effort would not have been possible. I am also grateful to Dr. Hasan Tahir and Dr. Razi Arshad from my Guidance and Evaluation Committee for their important observations and valuable suggestions throughout this research project.

Finally, I also want to express my gratitude to my **parents**, family, and friends, who not only persistently prayed for me but also encouraged and supported me during my research.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Android Operating System Framework . . . . .	2
1.1.1	Application Layer . . . . .	4
1.1.2	API Framework . . . . .	4
1.1.3	Libraries . . . . .	5
1.1.4	Hardware Abstraction Layer . . . . .	5
1.1.5	Linux Kernel . . . . .	6
1.2	Problem Statement . . . . .	6
1.3	Research Objective . . . . .	7
1.4	Thesis Motivation . . . . .	8
1.5	Thesis Organization . . . . .	8
<b>2</b>	<b>Literature Review</b>	<b>10</b>
2.1	Introduction to Malware . . . . .	10
2.2	Malware Insertion Process in Android . . . . .	14
2.3	Malware Detection Technique . . . . .	15
2.3.1	Static Malware Detection . . . . .	16
2.3.2	Dynamic Malware Detection . . . . .	16

## TABLE OF CONTENTS

2.3.3	Hybrid Malware Detection . . . . .	17
2.3.4	Comparison of Detection Techniques . . . . .	18
2.4	Machine Learning (ML) . . . . .	18
2.4.1	Random Forest (RF) . . . . .	20
2.4.2	Decision Tree (DT) . . . . .	20
2.4.3	K-Nearest Neighbour Algorithm (KNN) . . . . .	21
2.4.4	Support Vector Machine (SVM) . . . . .	21
2.5	Related work . . . . .	22
2.5.1	Studies based on Static Malware Detection Approach .	22
2.5.2	Studies based on Dynamic Malware Detection Approach	23
2.5.3	Studies based on Hybrid Malware Detection Approach	25
2.5.4	Discussion . . . . .	25
2.6	Summary . . . . .	26
<b>3</b>	<b>Research Methodology</b>	<b>28</b>
3.1	Research Methodology . . . . .	29
3.1.1	Data Acquisition . . . . .	29
3.1.2	Data Preprocessing/Feature Engineering . . . . .	30
3.1.3	Feature Selection . . . . .	32
3.1.4	Model Training . . . . .	32
3.1.5	Model Evaluation and Tuning . . . . .	33
3.2	Tools and Technology . . . . .	33
3.3	Summary . . . . .	35
<b>4</b>	<b>Data Acquisition &amp; Preprocessing</b>	<b>36</b>
4.1	Data Acquisition . . . . .	36



*TABLE OF CONTENTS*

4.2	Data Preprocessing . . . . .	37
4.2.1	Exploratory Data Analysis (EDA) . . . . .	37
4.2.2	Data Cleaning & Integration . . . . .	39
4.2.3	Data Labeling . . . . .	41
4.2.4	Data Normalization/Scaling . . . . .	43
<b>5</b>	<b>Feature Selection &amp; Model Training</b>	<b>45</b>
5.1	Feature Selection . . . . .	45
5.2	Model Training . . . . .	50
5.3	Summary . . . . .	51
<b>6</b>	<b>Model Evaluation &amp; Tuning</b>	<b>52</b>
6.1	Evaluation Metrics . . . . .	52
6.1.1	Accuracy . . . . .	54
6.1.2	Precision . . . . .	56
6.1.3	Recall . . . . .	56
6.1.4	F-1 Score . . . . .	56
6.2	Results and Evaluation . . . . .	56
6.2.1	Initial Results . . . . .	57
6.2.2	Model Tuning & Final Results . . . . .	59
6.3	K-Fold Cross Validation for Results Validation . . . . .	63
6.4	Comparison with Existing Work . . . . .	65
6.5	Summary . . . . .	67
<b>7</b>	<b>Conclusion &amp; Future Work</b>	<b>69</b>
7.1	Conclusion . . . . .	69

*TABLE OF CONTENTS*

7.2	Thesis Contribution . . . . .	71
7.3	Future Work . . . . .	71
7.3.1	Adding more Balanced number of Categories . . . . .	71
7.3.2	Improvement in Categories Classification . . . . .	72
7.3.3	Performing Malware Family Classification . . . . .	72
	<b>Bibliography</b>	<b>73</b>

# List of Tables

2.1	Malware categories and families [48] . . . . .	11
2.2	Comparison of malware detection techniques . . . . .	18
3.1	Tools & Technologies used during this research work . . . . .	34
5.1	Listing of 12 generated features set . . . . .	47
5.2	Top50 features using ExtraTree classifier for Level 1 Classification . . . . .	48
5.3	Top50 features using ExtraTree classifier for Level 2 Classification . . . . .	49
5.4	Malware categories with number of samples . . . . .	51
6.1	Level 1 Binary Detection results(Accuracy) using ExtraTree classifier . . . . .	58
6.2	Level 1 Binary Detection results (Accuracy) using Mutual Information . . . . .	58
6.3	Level 2 Categories Classification results(Accuracy) using ExtraTree classifier . . . . .	58

*LIST OF TABLES*

6.4	Level 2 Categories Classification results(Accuracy) using Mutual Information . . . . .	59
6.5	Final results of Binary Detection . . . . .	60
6.6	Final results (Average) of Category Classification . . . . .	60
6.7	Comparison with existing work . . . . .	67
6.8	Comparison with existing work part 2 . . . . .	67

# List of Figures

1.1	Android Architecture [3]	3
2.1	Malware Detection Techniques [54]	15
3.1	Research Methodology (process flow chart) [56]	29
4.1	Exploratory analysis of data using different metrics	38
4.2	Checking missing values and showing dataset shape	39
4.3	Data Scaling / Normalization view	44
6.1	Confusion matrix for malware detection	55
6.2	Confusion matrix of RF for Binary Detection	61
6.3	Category classification Random Forest final results	62
6.4	K-Fold Cross Validation [61] for K = 5	64
6.5	K-Fold Cross Validation for Binary Detection	64
6.6	K-Fold Cross Validation for Categories Classification	65

# Abstract

Android became one of the most widely used mobile operating system, and the amount of malware targeting it is increasing at an alarming rate. Despite the fact that notable studies on malware detection and classification have been conducted in academia and industry, but a robust and efficient solution for detection of all types of Android malwares is still a challenge. Existing solutions do not adequately consider factors like concept drift and are often not based on a hybrid approach. Also they have been designed using information collected by running malware samples on virtual environment (and not on a real device). Thus, they are not able to detect sophisticated or new malwares. In this research work we have studied existing solutions and after finding their limitations we have proposed an effective and efficient hybrid Android malware detection solution based on machine learning to detect and categorize existing, emerging and behaviour evolving Android malwares.

# Chapter 1

## Introduction

In the past few years smart devices e.g smart phones has become the major source for digital information transfer and internet usage. There are different smart devices operating systems available for these devices which include Android, IOS, Blackberry, Windows etc. In the current era of technology people are using these smart phones as their one in all gadget for digital purposes. These smart devices are being used to perform Sensitive Banking Transaction, Secret and Private Communications, and to store Personal and Private Data of user's like Credit/Debit Card Numbers, Social Security Number, Passwords, Personal & Private Pictures and Personal/Business emails. If any of these data goes into unauthorized hands it can cause severe problems for the users. According to current stats [1] Android Holds 69.74% of market share's while IOS being the second market competitor hold 25.49% of market shares. Remaining 0.77% of market shares are held by other operating systems like Blackberry, Windows, Symbian etc. Recent stats show that new Android malwares are amounted 482,579 per month [2]. It is due

## *CHAPTER 1. INTRODUCTION*

to Android being open source, user friendly nature and being market leader is smart devices. There are three basic approaches for detecting Android malware: static malware detection, dynamic malware detection, and hybrid malware detection. Static approach uses different features extracted from applications without executing applications, while dynamic malware detection uses different features of applications obtained by running applications in emulated environment or on real devices. Hybrid malware detection uses features obtained in both static and dynamic detection stages. These all solutions have different advantages and limitations which will be discussed in chapter 2. To understand the malicious behaviour of malwares within Android platform it is essential to know the deep down working of different components of Android operating system. To achieve this objective, we shall briefly discuss the Android Operating System's platform architecture and important components.

### **1.1 Android Operating System Framework**

Android is built and defined in layered architecture [3]. It has five layers and Linux kernel is baseline of it. All other layers are architected on the top of Linux Kernel. Every layer in Android architecture has different level of abstraction in performing its tasks. Figure 1.1 shows the architecture of the Android platform.



CHAPTER 1. INTRODUCTION

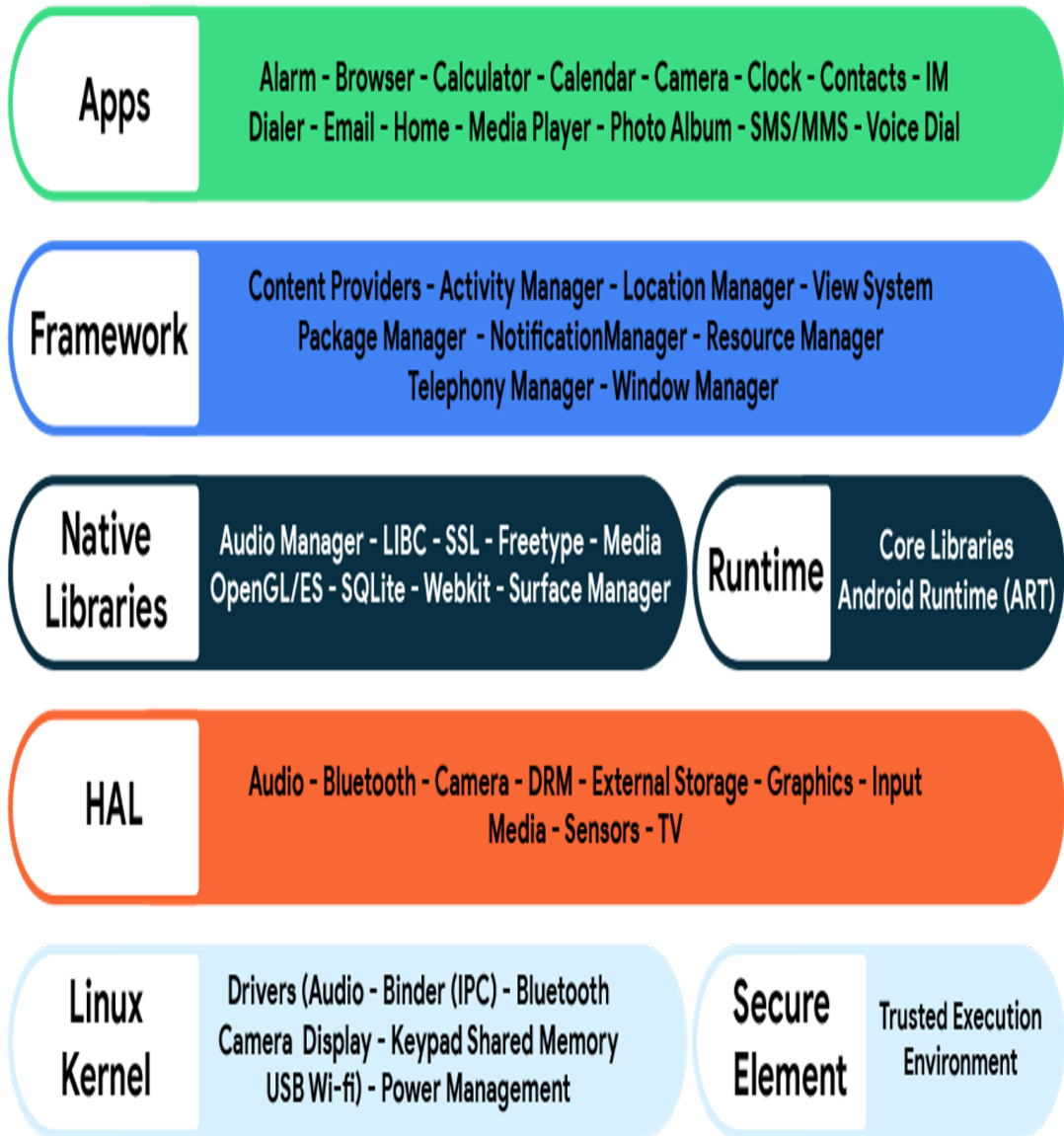


Figure 1.1: Android Architecture [3]

## *CHAPTER 1. INTRODUCTION*

### **1.1.1 Application Layer**

The application layer is the initial and fundamental layer of Android architecture. Application Layer is primarily accessible to end user's also by communication through Android Debug Bridge(ADB). There are two types of Applications in this layer: System applications and Third Party applications. System apps are those which are pre-installed in device by the Original Equipment Manufacturer(OEM) while the third party apps are those which are installed by user's from different available third party store's and websites. Android has few extra controls over system applications as these applications cannot be removed by end user's , but can only be disabled. In security sensitive devices enterprises equip their devices with their applications as system applications. System applications are also less vulnerable and exploitable to malwares as compared to third-party application in different aspects.

### **1.1.2 API Framework**

As applications in Android reside on top layer and those applications need to communicate to Linux Kernel to perform different activities. Linux Kernel is the building block of Android operating system. This API Framework provides different set of APIs and services to allow applications to perform their activities and to utilize and communicate with underlying Linux Kernel. API Framework provides services (background processes), Activities Manager, Intent Filters and Content provider's. These all components have different key tasks in Android OS. Malwares also target the API Framework to obtain access to application data, hence malware detection and prevention

are required.

### 1.1.3 Libraries

This layer provide set of libraries to facilitate the communication between API Framework and lower layer's. It include different libraries like OpenGL for graphics, Web-kit for web interactions etc. This layer is mainly categorized into two sub categories.

- **Native C/C++ Libraries:** Native libraries includes core C/C++ libraries along with their Java wrappers which simplifies their calls from top layers. These libraries include Web-kit(for internet surfing), SSL(for Secure Communication), SQLite(for Data Storage and Management) and different other libraries which facilitate specific features.
- **Android Runtime:** Applications run in this layer and their execution and some of services execution takes place using Android Runtime. ART executes Dalvik execute-able format and Dex byte code by running multiple virtual machines. Since Android 5.0 ART runs each application into its own process with its own instance.

### 1.1.4 Hardware Abstraction Layer

Hardware Abstraction Layer is the separating point between Linux Kernel and Android Framework other layers. It reveals device hardware resources to the Java API Framework through an interface. HAL has several library modules, each with its own interface for that certain sort of hardware component, for example WIFI Card, Camera etc. Applications in Android access

## *CHAPTER 1. INTRODUCTION*

hardware by invoking a call through Framework API to the libraries of a specific module of a specific hardware.

### **1.1.5 Linux Kernel**

Linux kernel is main building block of Android OS. All other layers are built on top of it. Linux Kernel is an open source kernel and it has its own security and privacy controls built in it. Apps sandboxing and resource isolation in Android is also managed by Linux Kernel. Linux Kernel manages resource access, power management, memory management and Network stack for Android Runtime(ART). ART also depends on Linux Kernel for some of tasks like low level memory management and threading. Developing hardware driver's for Linux Kernel is also a plus point for device manufacturer as it is a well known kernel. Some of the key security features of Linux kernel Include user's based permissions model and process isolation. These key security features play important role in maintaining data security when a device gets infected with some malware.

## **1.2 Problem Statement**

As discussed above Android is main target of malwares and malicious applications due to its open nature framework and open source code. Android applications use different static and dynamic features to accomplish their key task. Malicious applications also use similar features but in different manner and amount. There are different existing solutions proposed and provided by both industry and research domain to detect and categorize malicious

## CHAPTER 1. INTRODUCTION

applications which are using these static (static malware detection) and dynamic features (dynamic malware detection) from benign applications. Both type of these solutions do not detect different type of malwares like static solutions [4], [5] are weak at detection of behavior changing malwares while dynamic detection [6], [7] has limitations when working on emulated environment. As the malwares are advancing therefore new emerging and behaviour evolving (concept drift) malwares are bypassing these solutions. There is very limited work done on hybrid detection using real devices. Therefore there is a dire need for an efficient solution to detect and categorize all type of malwares specifically including evolving malwares. We have proposed a solution to cater these behaviour changing malwares in Android using hybrid malware detection on real devices.

### 1.3 Research Objective

This research proposes a robust and efficient solution to handle all type of evolving malwares. Main objectives of the research are:

- Machine learning based Android malware detection solution using hybrid features.
- Machine learning based Android malware identification and categorization using hybrid features.
- Generation of high accuracy model to be able to be used by the industry and end-users.

## 1.4 Thesis Motivation

As it is highlighted in 1 malwares targeting Android are increasing rapidly. An example of it reported by zdnet [53] is a most notorious Android malware flubot which is active since November 2020. It steals passwords, bank details and other sensitive information from infected smart devices. Other than this it has ability to spread itself like a worm by accessing the contacts of infected devices and sending sms. This proves that cost associated to these malwares threat is much more then the cost associated with detection solutions. When a malware breach a device it not only harm the device but also obtain sensitive and private data of user. Early detection of malwares can prevent from these infections and safeguard the sensitive and personal data of users. Security experts also advice early detection and prevention of malwares. Therefore the motivation behind this research work is to provide an effective and efficient Android malware detection solution by covering the major highlighted limitations of existing solutions to limit and control the existing and upcoming threats of Android malwares.

## 1.5 Thesis Organization

Thesis organization is presented in following outline. Chapter 2 provides details of literature review by explaining about malware, malware insertion process, malware detection techniques, existing work on these techniques and their limitations. Chapter 3 provides proposed and implemented research methodology along with techniques and tools used. Whole experimental

## *CHAPTER 1. INTRODUCTION*

setup including data preprocessing/feature engineering, feature selection and model training process is discussed in Chapter 5. Chapter 6 continues the experimental setup by further explaining model evaluation and tuning. This chapter further explains results validation process and comparison of our work with existing similar published research work. Lastly, in Chapter 7, we wrap up this study and highlight possible future work that could be carried out on basis of the findings.

# Chapter 2

## Literature Review

All of the important research work mainly related to this study are discussed in this chapter. Main focus was put the cover the closely related and recent studies. In addition to that some extra focus has been put on incorporating additional studies to help the reader have a better understanding of the proposed technique as they read through this thesis.

### 2.1 Introduction to Malware

Malware-bytes define malware as [8] "Malware, or "malicious software," is a general phrase used to denote any malicious software or dangerous computer code". Mainly malware is short-term of malicious software which is any invasive program/software is a general phrase used to denote any malicious software or dangerous computer code.e developed by cyber-criminals for malicious intentions like to gain access to unauthorized devices and to steal data and to damage or destroy those computing devices. It is a collec-



## CHAPTER 2. LITERATURE REVIEW

tive name of different malicious software's including viruses, worms, trojans, ransomwares, adwares, and spywares etc. There are different categories and families of malwares. Some of most common categories and families are listed in table 2.1. Malwares are created for all type of devices but as discussed in first chapter Android is main target of malwares.

Table 2.1: Malware categories and families [48]

<i>Categories</i>	<i>Definition</i>	<i>Families</i>
<i>Adware</i>	It is a type of malware which shows intrusive advertisements while using web applications	Families include gexin, ewind, pandaad etc
<i>Backdoor</i>	It is a sort of malware that enters a device without being detected and maintains remote access to the device	Families are fobus, kmin and droidkungfu etc
<i>Scareware</i>	It is a malware which tricks mobile users by scaring them into visiting malware inserted websites	Example of families include av-pass and fakeapp
<i>PUA</i>	These applications may pose a significant risk or have a negative impact on the security and privacy of users. These applications also consume computational resources without any necessary need.	Families are apptrack,youmi, cauly etc

CHAPTER 2. LITERATURE REVIEW

Continuation of Table 2.1		
<i>Categories</i>	<i>Definition</i>	<i>Families</i>
<i>File Infector</i>	File infectors infect files in device to spread to others devices, removeable drives and networks.	Families include leech, tachi and many others.
<i>Riskware</i>	Riskware is basically a non malicious program/app but its installation and execution has risk associated with it due to some vulnerability or incompatibility.	List of families include triada, skymobi, sms-pay and many others
<i>Ransomware</i>	This malware takes complete control of device or users data and asks ransom to leave the control	Some famous families include slocker, congur and masnu.
<i>Trojan</i>	Trojans are a type of malware that caricature trustworthy programmes, files, or applications in order to deceive users into installing it and allowing it unrequested access to their devices. .	List of common families include lotoor, robtes and hqwar
<i>Trojan-SMS</i>	These Malware send and intercept messages via a mobile device's SMS (text) messaging services. In most cases, the user is completely oblivious of the activity.	Some common families are opfake, plankton and boxer

CHAPTER 2. LITERATURE REVIEW

Continuation of Table 2.1		
<i>Categories</i>	<i>Definition</i>	<i>Families</i>
<i>Trojan-SPY</i>	It is a sort of malware that has a wide range of functions, such as keystroke logging, spying operation of a device, and stealing data from stored files.	
<i>Trojan-Banker</i>	Trojan-Banker malwares are programmed to snatch data from clients' credit cards, online banking, and e-payments.	Example of families are bankbot, fake-token and minimob
<i>Trojan-Dropper</i>	Trojan dropper is a program that drops and install another programme into a computer or device, typically a malicious one. Basically it is a carrier for a malicious payload which it drops on targeted device for harmful actions.	Its families include cnzz, rooter and some other families
<i>Trojan/Riskware</i>	This category is basically those malwares which lie in both categories of torjans and riskware.	It has families of both categories

Continuation of Table 2.1		
<i>Categories</i>	<i>Definition</i>	<i>Families</i>
<i>Spyware</i>	It is a sort of malware which resides in your computing machine or mobile device, executes different tasks and gathers sensitive data , monetary information, user-names and passwords, and other individual data.	
End of Table		

## 2.2 Malware Insertion Process in Android

Malwares are mainly injected intentionally by applications developers or hacker's. They develop malicious applications and publish them on the play-store or other app stores. Another way of malware insertion [9] into Android market is through repackaging of applications. Cyber criminals download existing applications, decode them using different tools like APKTool, DextoJar etc, insert their malicious code into applications and then republish them to different apps store's. These application store's already have implemented different prevention and detection techniques for application publishers but cyber criminals still bypass these techniques using different strategies like code obfuscation. The third way of malwares insertion is when a developer or owner sell and application, the new owner infect it with malicious code and republish it to apps store's.

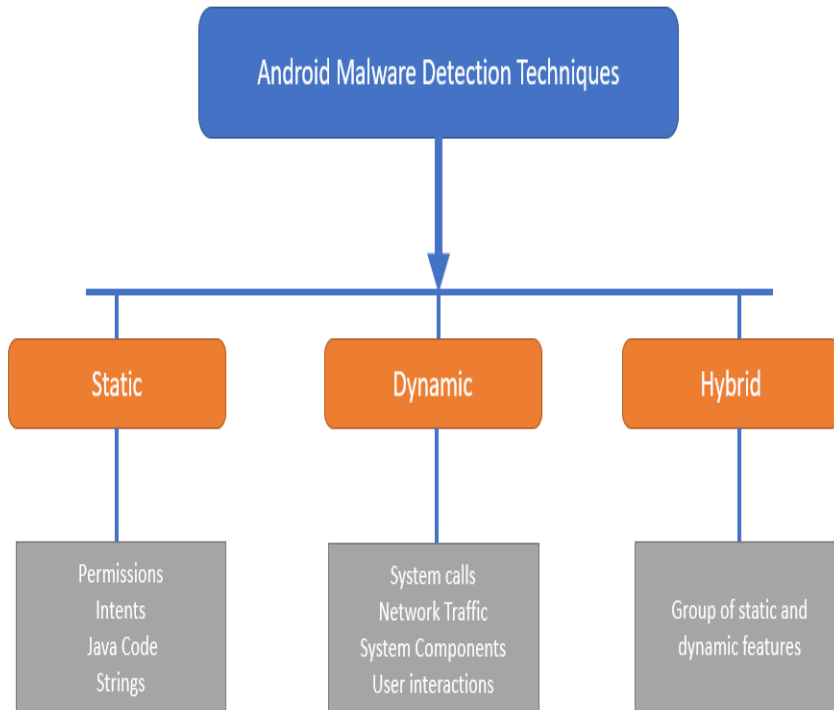


Figure 2.1: Malware Detection Techniques [54]

## 2.3 Malware Detection Technique

There are three different common techniques of malware detection : Dynamic Malware Detection, Static Malware Detection and Hybrid Malware Detection as shown in figure 2.1. These techniques have different advantages and disadvantages. We will discuss these techniques and existing work on them one by one.

### 2.3.1 Static Malware Detection

In this techniques malware detection and classification is performed without running malicious applications [10]. Features are collected without running the application using different tools like Apktool [11], Dex2Jar, and Androguard [12]. Android applications are compressed as zip files in apk format. These application are first decompressed using above mentioned tools and then their features like permissions, intents, services, and APIs are extracted to perform static analysis. Static analysis marks an application as benign or malicious based on proportion and usage of these features. Static analysis is rapid detection solution and less resource consumption process when compared with dynamic and hybrid solutions but previous research showed that it does not work well on all type of malwares [13]. As this analysis marks an application malicious on the basis of usage of different features of Android operating system, so a new malware can change the proportion of usage of those features. New malwares can attack with different signatures so they can bypass static malware detection techniques implemented using old signature. Other than this behaviour changing and sophisticated malwares can also bypass static analysis detection techniques. These malwares execute their malicious behaviour after getting installed on the targeted device.

### 2.3.2 Dynamic Malware Detection

In the process of dynamic malware detection malware behaviours are analyzed by running it on emulated environment or real devices [6]. Dynamic features are extracted during this process. This analysis is performed with

## *CHAPTER 2. LITERATURE REVIEW*

some human interaction or by using some automated scripts to capture the real-time behaviour of malicious applications and collect dynamic information like system calls, network activities, kernel calls and process execution. This collected information shows the exact behaviour and intentions of under observation Android applications. Dynamic malware analysis requires excessive resource utilization along with increased time as compared to static analysis. Dynamic analysis also has limitations if performed on emulators then sophisticated and advance malwares can detect emulated environment and do not execute in that environment. If dynamic analysis is performed on real devices it provides superior results as all type of malwares execute on real devices.

### **2.3.3 Hybrid Malware Detection**

Hybrid detection is a mixture of both dynamic and static detection. In this method both dynamic and static extracted features are used to detect and classify malicious applications. Hybrid analysis is most comprehensive analysis of Android applications as it involves both static features and run-time behaviour of applications. It yields superior accuracy compared to previous techniques and detects advanced and sophisticated malwares including signature based (static features like permissions) as well behaviour changing malwares (which bypass signature based detection techniques).

### 2.3.4 Comparison of Detection Techniques

All of these three techniques have their advantages and disadvantages but according to their results hybrid malware detection is considered as the best technique of detection as it give superior accuracy then other two techniques and also proves better in detection of sophisticated and advanced malwares. Comparison of malware detection techniques is carried out in table 2.2.

<i>Malware Detection Technique</i>	<i>Advantages</i>	<i>Disadvantages</i>
<i>Static Detection</i>	<ul style="list-style-type: none"> <li>• Time Efficient</li> <li>• Resource Efficient</li> </ul>	<ul style="list-style-type: none"> <li>• Not effective against behaviour changing malwares</li> <li>• Average Detection results</li> <li>• Not effective against new malwares</li> </ul>
<i>Dynamic Detection</i>	<ul style="list-style-type: none"> <li>• Effective against behaviour changing malwares</li> <li>• Efficient against new and advanced malwares</li> </ul>	<ul style="list-style-type: none"> <li>• High resource consumption</li> <li>• Not time efficient</li> <li>• Emulated environment can be detected by advanced malwares</li> </ul>
<i>Hybrid Detection</i>	<ul style="list-style-type: none"> <li>• Effective against signature based and behaviours based malwares</li> <li>• Yields high accuracy</li> </ul>	<ul style="list-style-type: none"> <li>• Requires more resource and time</li> </ul>

Table 2.2: Comparison of malware detection techniques

## 2.4 Machine Learning (ML)

Machine Learning is defined by SAS institute [55] as” machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn



## *CHAPTER 2. LITERATURE REVIEW*

from data, identify patterns and make decisions with minimal human intervention.” Machine learning handles two types of problems classification and regression. In classification problem the predicted variable is discrete variables and usually called labels or categories. A classification model predicts the test input to a specific label or category. While in regression problems the predicted variable is continuous real value like integer or floating point. These values are usually in quantities like amount and sizes. Malware detection is a binary classification problem as its either classify a sample into benign or malicious sample and malware category classification is a multi class classification problem as it classify different malwares into different categories. Machine Learning has important role in malware detection and categorization. There has been lot wide range of research efforts on malware detection using machine learning. Different malware detection solutions has been proposed by many researchers using different machine learning specifiers. Machine learning is classified into three types [14]: supervised, unsupervised and reinforced learning. In supervised learning before training to model, data must be labeled while in unsupervised learning data is not labeled. However, supervised learning produces greater results than unsupervised learning. Machine learning pipeline/process involve different steps to solve the any type of ML problem. These steps are data collection/data acquisition, data preprocessing/feature engineering (data cleaning, labeling, scaling), features selection, model training and evaluation. This whole process of machine learning is described in details in chapter 3. We have used supervised learning with most common algorithms in our proposed solution as our dataset was labeled in level 1 classification. We will give an overview

of most popular supervised machine learning algorithm which we have also used in our proposed malware detection approach.

### **2.4.1 Random Forest (RF)**

This is a supervised machine learning algorithm that is commonly employed in various classification and regression approaches. RF makes decision trees on different samples. It uses majority voting for the prediction of classification and uses average for the regression problems. It does not rely on one tree instead it takes prediction from all the trees and then make its final results based on those all predictions. RF utilizes decision trees as root classifier [15]. In random forest as the number of trees increase, accuracy increases and also it prevents the problem of over fitting. The ability of RF to handle both categorical and continuous data in both classification and regression problems is one of its most major characteristics. Generally random forest achieves superior results for classification problems.

### **2.4.2 Decision Tree (DT)**

This is a supervised learning technique that can be applied to both classification and regression. DT resembles a tree because it starts like a root of the tree and then develops branches to form a tree. Decision rules in DT are represented as branches, outputs as leaf nodes, and internal nodes as dataset features [16]. There are two different types of nodes in it: decision nodes, which are used to make decisions, and leaf nodes, which are the outcome of decision nodes. Decision nodes can contain further branches but leaf nodes

can't.

### 2.4.3 K-Nearest Neighbour Algorithm (KNN)

It is the widely used supervised machine learning algorithm. Based on the votes given through its K nearest neighbours during the testing phase and the similarity distance of the available instance that was used in the training phase, KNN classifies new instances. [17]. It is used for both regression and classification problem. Its also called lazy learner algorithm as it does not instantly learn from training data, rather its store that data and during the classification time it perform required actions on it. Since KNN just stores data, whenever it receives new data during the testing phase, it classifies it into a very similar type of stored data.

### 2.4.4 Support Vector Machine (SVM)

It is another well-known supervised machine learning algorithm. It is also used in both classification and regression but mainly it is used for classification problems. Malware based datasets are scattered and large [18] which create issues for classifying data using SVM. However it can be tuned to perform better and give better results. The primary objective of the SVM algorithm is to find the optimal line or decision boundary which can split n-dimensional space into classes such that subsequent data points could be easily placed in the relevant category. The best choice boundary is referred to as a hyperplane.

## 2.5 Related work

Other researchers have made efforts to detect Android malware utilising Static malware detection, Dynamic malware detection, and Hybrid malware detection.. These efforts are described below one by one.

### 2.5.1 Studies based on Static Malware Detection Approach

H. Bai et al. [19] performed static malware detection and family classification by using permissions and intents with the help of Cat-Boost as the algorithm. For benign apps, they used the Drebin dataset, while for malicious applications, they generated their own dataset. They obtained 97.40 percent accuracy in malware detection and 97.38 percent accuracy in family classifications. They were not getting good results on Linux-Looter family of ransomwares. Despite good results advanced malwares can bypass their solution as static detection is used as the detection technique. Similarly Abeer Rahali et.al. [20] did static malware detection categorization using deep image learning. They have used Activities, Services, Broadcast receivers/providers, Intent actions, Permissions, Metadata as static features. For the process of features selection they utilized ExtraTree Classifier as the algorithm. Their main contribution also include generation of large dataset CCS CICAND-MAL 2020 which have 200K malicious applications. This dataset has 12 malware categories and 191 malware families. Their results showed an accuracy of 93% and also good accuracy on categories classification. But this approach have similar above mentioned problem. Arora et.al [10] used per-

## CHAPTER 2. LITERATURE REVIEW

missions pairs extracted from Android applications manifest files to construct the graph of malicious and benign applications. They were successful in detecting malwares with an accuracy of 95.44% but as they have only used permission pairs, malwares can use other features and bypass this technique to perform malicious actions. Tao et al [21] examined permissions, APIs, and the connections between them to identify malware. They formed a dataset of around 30K benign apps and 15K malicious apps. First of all their dataset was imbalanced secondly their dataset is old(2015). Third limitation is the signature based detection can only detect applications using signature. Cen et al. [22] used a probabilistic discriminative model to detect malicious samples using decompiled source code and permissions. Their final analysis involved a dataset with around 11K samples with 9% of malicious sample. First of all their dataset is small, imbalanced and very old. Secondly this research work have used decompiled source code and permissions so if a malicious application is using byte code encryption or obfuscation this solution will not work.

### 2.5.2 Studies based on Dynamic Malware Detection Approach

Entropylyzer [6] is a dynamic analysis technique employed using Shanon entropy for feature's ranking and to analyze behavioural changes of malwares using six classes of features. Later they used Machine Learning algorithms to find out malwares categories and families. This analysis used CCS-CIC-AndMal2020 dataset which had 12 malware categories and 191 malware fam-

## CHAPTER 2. LITERATURE REVIEW

ilies. This analysis was performed on emulators in sandbox environment. They used 141 dynamic features for this malware analysis. Although they achieved good results on malware category and family identification but some of samples were not executed during this analysis. It emphasises the need of implementing dynamic analysis on real-mobile devices. Mahdavifar et al. [7] suggested a semi-supervised learning deep neural network approach for dynamic malware category categorization. They had used Copper Droid VMI based system which is also an emulated dynamic malware analysis platform. They used system calls, binder calls and composite behaviours as dynamic features. Their research work have only five malware categories. Further more analysis was performed so there is possibility that some malwares have not executed their malicious behaviour after detecting emulated environment. In conclusion, Droidbox [24], DroidMat [25], and AMAT [26], were presented as more advanced approaches to developing an emulator or sandbox. These techniques have used permissions, intents and API calls as the features set for their analysis process. All of these techniques, although significant, were shown to be unsatisfactory for dealing with anti-emulation completely. Several anti-emulation techniques have been described in subsequent works. According to Vidas and Nicolas [27] emulators are detected by many other applications by taking advantage of Android API . An example is when the API of the Telephony Manager method `TelephonyManager.getDeviceId()` reply with `0000000000000000`, it indicates the execution is being done on emulated environment rather than of a real device. It indicates that dynamic analysis needs to be performed on real devices to achieve satisfactory results.

### 2.5.3 Studies based on Hybrid Malware Detection Approach

There were research efforts carried out in hybrid malware detection but most of them performed dynamic detection part on emulators. A hybrid malware classification technique utilising pseudo label stacking auto encoders has been proposed by mahdavifar et.al [28] . To extract both system and dynamic features, they utilized a Virtual Machine Introspection (VMI)-based system. Their model could detect and classify malware with an accuracy of 98.28% but their dataset was small and also they have limited number of five categories only. They VMI based system is also emulator based environment which have similar problem mentioned in previous section of dynamic malware detection. Several research indicates that hybrid-based analysis significantly increases detection. Ali-Gombe and colleagues use both static and dynamic approaches. To identify resource misuse and look into questionable behaviour, which is then dynamically analyzed, it utilises byte code instrumentation [29]. Surendran et al. offered a novel hybrid technique for malware detection based on conditional interdependence between dynamic and static features (API calls, permissions, and system calls) employed in their machine learning classifiers [30]. Besides that, this research is still having performance problems since there are so many parameters that need to be evaluated.

### 2.5.4 Discussion

All of the above mentioned previous research work have used different techniques including static, dynamic, and hybrid malware detection. These all

## *CHAPTER 2. LITERATURE REVIEW*

techniques have some limitations and all are weak at detection of advance Android malwares. Therefore there is a dire need of a hybrid malware detection solution on real devices. It will solve problem of behaviour changing malwares, and anti emulator problem. Further if a complete and comprehensive dataset having malware sample's of all years with increase number of categories and families is used then it will also strengthen malware detection against those malwares which change their behaviour and working techniques over time. This behaviour of malwares changing with time is also known as concept drift. Concept drift is a machine learning terminology which is defined as change in relationship of input and target variable over time. This change negatively impact the accuracy of trained model. For example in case of Android malwares change their attacking techniques continuously. When a malware gets detected by an antivirus or anti malware solution it changes its signature or attacking behaviour to infect the devices without getting caught. This changing behaviour of malwares is called as concept drift. To handle the issue of concept drift in machine learning dataset should be extensive. We have proposed a malware detection solution with a complete, comprehensive and latest dataset using machine learning and real devices to fix all of above mentioned shortcomings of malware detection solutions.

### **2.6 Summary**

In this section, we have discussed malwares, malware insertion process in Android and malware detection techniques. Other than this we have highlighted the existing research work done on malware detection using different



## *CHAPTER 2. LITERATURE REVIEW*

techniques. At the completion of this section of this section we discussed the shortcomings of published research work in malware detection solutions and suggested the improvements needed to provide a more advanced and robust malware detection solution. In the coming chapter we will discuss the research methodology used during this research work.

## Chapter 3

# Research Methodology

This section describes the methodology used throughout this research study. We will go over the steps that were taken in order to achieve the desired results. As this is a machine learning classification problem, the steps involved include all of the processes involved in a typical machine learning process, such as data collection, data cleaning and labeling, exploratory data analysis (EDA), data normalization, feature engineering, model evaluation and tuning. Whole process is illustrated in Figure fig 3.1 Following the process description, this section will also highlight the tools and technologies that are used in this research.

### 3.1 Research Methodology

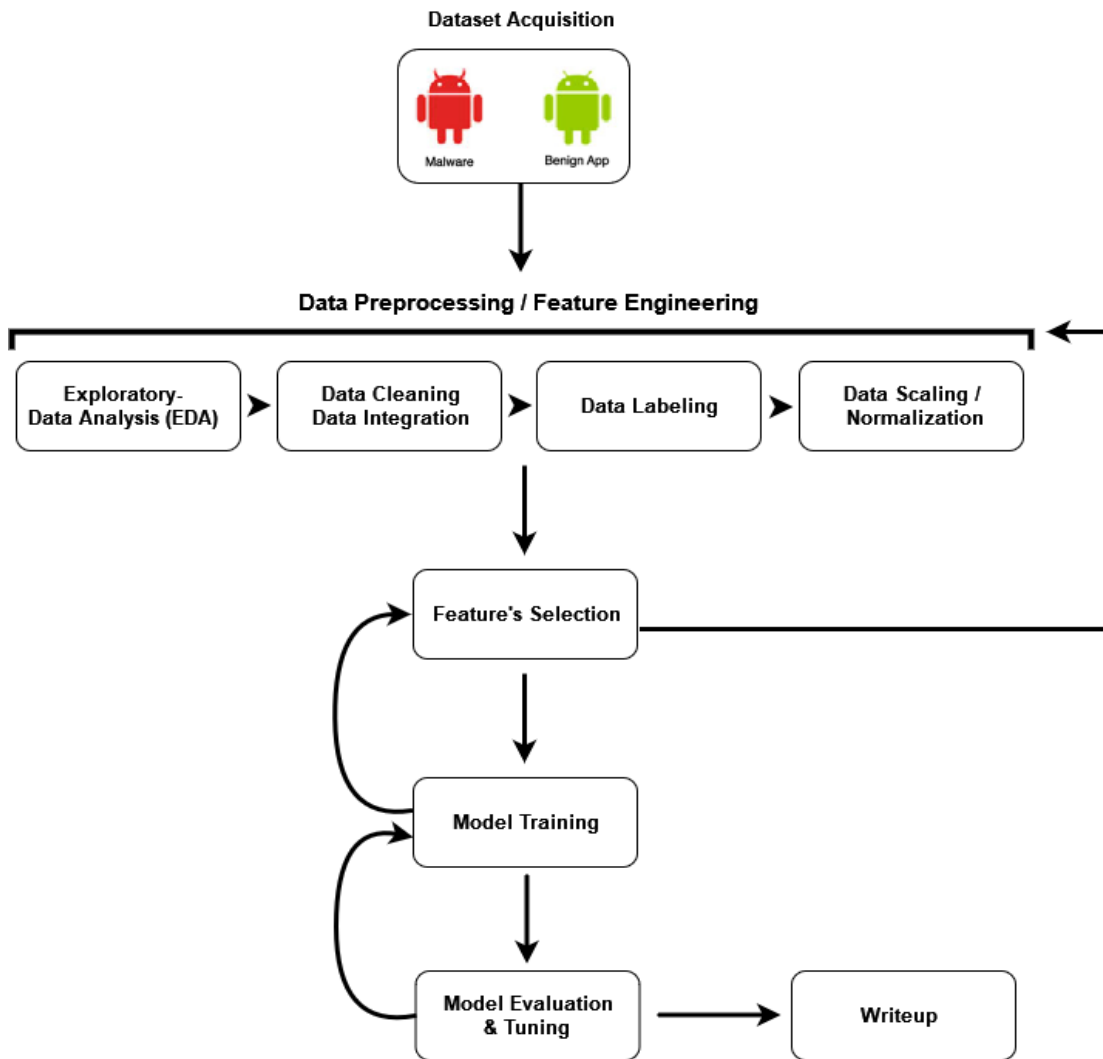


Figure 3.1: Research Methodology (process flow chart) [56]

#### 3.1.1 Data Acquisition

There are two ways of data acquisition: Collecting samples from different sources and generating a dataset for the research, or using an existing dataset published by some other research to perform intended research activities. In

our case we will be using existing dataset published by another research to perform our proposed solution of hybrid malware detection.

### **3.1.2 Data Preprocessing/Feature Engineering**

Data preprocessing is also called feature engineering and it has different steps, which are discussed one by one below.

#### **Exploratory Data Analysis (EDA)**

In machine learning datasets come in raw form and contain large number of samples and similarly those samples include large number of features or attributes. As a result, it is difficult to determine data characteristics by looking at a column of numbers or an entire spreadsheet. Also datasets are not easy to understand in their raw form. Exploratory data analysis is a technique for making judgments and gaining insights from data [57]. In this research phase, data is transformed into different graphical and statistical depictions that better represent the state of dataset. This research phase is generally carried out according to the data being used because datasets can only be represented according to applicable exploratory analysis techniques. To get an idea of the values within the data attributes, statistical methods – for example mean, median, standard deviation, and such are applied to numerical datasets. In addition, to visually inspect the data, researchers use various graphical presentation such as histograms, bar graphs, scatter plots, frequency distributions, and word clouds. Understanding of data, features and exploring it in details for further pre-processing is the primary objective

of this approach.

### **Data Cleaning & Labeling**

Throughout this process, data is cleaned to ensure that it is clean from outliers and observations which could lead to inaccurate results. This process employs a variety of techniques, including the removal of null, empty and duplicate records. After data cleaning, data labeling begins in which data is labelled across its entries. In our case, we performed data cleaning and data labeling respectively according to the our proposed solution. Data was already labelled as benign and malicious data but in the second step of categories classification we performed data labeling.

### **Data Normalization/Scaling**

It is a subpart of feature engineering stage in which attributes of dataset are converted in fine grained range to ease the process of feature engineering and model training for algorithms. The process of improving clean data is known as data normalisation. However, data normalisation is important for two reasons:

- Data normalisation is the process of transforming the data such that it looks the same across all records and fields
- It enhances entry type cohesiveness, which leads to data cleaning, lead creation, categorization, and greater data quality.

This step is not mandatory but it is very significant for some algorithms and improves the accuracy and detection rate impressively.

### 3.1.3 Feature Selection

Feature Selection is a machine learning technique which uses data to highlight features that were not known as the most impacting features. It has the potential to bring out most important features for both supervised and unsupervised learning, with the objective of facilitating and speeding up model performance also while improving model accuracy. Typically, exploratory data analysis, data normalisation and feature selection co - exist. The features upon which model is trained and the results are obtained are engineered based on information obtained in the previous two steps. This is a loop, as illustrated in the figure 3.1, in which researchers must switch between exploratory data analysis, data normalisation and feature selection steps. The main aim of these three steps is to create features that produce model with best results.

### 3.1.4 Model Training

The most important step in machine learning is training. During this step machine learning chosen model is trained with selected features set for getting the expected results. We input prepared data to a machine learning model during training, that searches for patterns and makes predictions. As a consequence, the model learns from the dataset and can predict outcomes for test data. The model improves at predicting over time as it is trained. In this step model is trained with chosen machine learning algorithm. Different algorithms perform differently on same features set. So before model training selection of right algorithm is also an important step.

### 3.1.5 Model Evaluation and Tuning

This is the final step of machine learning pipeline. After training your model, you must evaluate its performance. A machine learning model's evaluation is performed against the expected outcome or previously published similar work using the testing/unseen data. The testing set which divided data into earlier is the unseen data used. While performing testing make sure it is not on the same data that was used for training, if it is, it will not give accurate results since the model is already familiar with the information and finds the same patterns in it that it did before. This will provide with a biased high level of accuracy. By using the testing data, model gives precise results with justifiable performance and accuracy. If the results do not meet the evaluation metrics, the model is tuned. This model tuning is performed by using the right parameters of the selected algorithm. We can test different parameters on model and select the best parameters for model training. It will not only increase the performance of model but also enhance the detection results.

## 3.2 Tools and Technology

During the whole process of this research work we have used different tools and technologies. Since we are using an existing dataset published by [31] therefore it was not need to run and test the samples on real devices. This has already been done on the used dataset. We used machine learning on the available dataset to train the models for malware detection. Table 3.1 lists all of the tools and technologies used during this research.

CHAPTER 3. RESEARCH METHODOLOGY

<b>Tools / Technology</b>	<b>Usage in Implementation and Research</b>
Python [32]	Python programming language was used for writing different scripts of cleaning, labeling and compiling dataset.
Virustotal [33]	Virus total online repository was used for labeling malware categories.
Fsecure [34]	Fsecure online repository was used for labeling malware categories.
Fortiguard [35]	Fortiguard online repository was used for labeling malware categories.
Numpy [36]	Python library numpy is used for performing all mathematical operations. needed in the research.
Pandas [37]	Pandas is also a python library Its was used for all of data processing.
Matplotlib [38]	This python library was used for drawing different graphical representations e.g bar-graph, and confusion matrix.
Scikit-learn [39]	This python library was used for whole machine learning implementations.
Jupyter [40]	It is a web-based interactive computing platform and we have used it for the machine learning section.
Google Colab [41]	Google Colab is similar to jupyter but is an online environment and it was used to run jupyter notebooks machine learning tasks.
Pycharm [42]	It is a IDE for the development of python related tasks.
Latex [43]	Its is project by overleaf and an online type-setting system It was used for the whole write-up process .

Table 3.1: Tools & Technologies used during this research work



### **3.3 Summary**

Under this section, we described a graphical and textual overview of the entire research methodology used to conduct this research. All of the tools and technologies that have been used during this research work are also highlighted with the explanation for using each of those tools and technologies.

# Chapter 4

## Data Acquisition & Preprocessing

In this chapter process of data acquisition and preprocessing/feature engineering is explained. This chapter describes the in-depth details of all steps of data preprocessing stage.

### 4.1 Data Acquisition

Android applications include different set of features/attributes in them for performing the intended actions. These features include different static features like permissions, intent filters, strings, services, activities, dalvik opcodes, metadata and dynamic features like system calls, binders calls, API calls, network features, battery features and memory features. These features are key elements during the malware detection process. Therefore it is very important to select the right set of features while performing malware

detection. There are different Android malware datasets are published online by different research groups and individuals. These datasets differ in terms of the samples, the number of attributes, and the dates of sample publication and collection. Hence it is also important to choose the right dataset for providing better solution of malware detection. We choose real-devices subset of Kronodroid [44] malware dataset published by Alejandro et. al [31]. It is a comprehensive dataset that includes samples throughout all years of Android history between 2008 to 2020. This dataset has 200 static and 289 dynamic features making it more strong for malware detection due to these hybrid features. Static features include intents, permissions, files, services and timestamps while dynamic features include System calls. Furthermore, as claimed by the datasets creator, and to the best of our knowledge, this is the first dataset to take time variable/concept drift into account in malware identification. This dataset has 78137 of total real devices samples divide into 41382 malware sample's and 36755 benign samples. Other than benign and malware categorization this dataset has also further classified malware samples into their families, but malwares categories of the samples were not identified in this dataset.

## **4.2 Data Preprocessing**

### **4.2.1 Exploratory Data Analysis (EDA)**

EDA is an important step of data preprocessing/feature engineering phase. In this phase basically data is checked in more detailed. Data is checked on

## CHAPTER 4. DATA ACQUISITION & PREPROCESSING

number of samples, number of features and values of those features to better understand the structure of data and to highlight the important attributes of dataset. Different studies have used different EDA techniques and there are many python libraries to perform EDA on the data. We have used some of those libraries and inspected our dataset. We have first checked what is in data by using python libraries and then we checked the weightage and impact of attributes/features by using min, max and standard deviation etc. After this we checked for missing values, outliers etc. This helped us in data cleaning, data labeling, data scaling and feature selection phase. Figure 4.1 illustrates the weightage of different attributes of dataset on different mathematical scales and figure 4.2 shows the depiction of missing values and shape of dataset by displaying count of number of samples and features.

	Malware	execve	getuid32	getgid32	geteuid32	getegid32	getresuid32
<b>count</b>	78137.000000	78137.0	78137.000000	78137.000000	78137.000000	78137.000000	78137.0
<b>mean</b>	0.529608	0.0	413.515966	0.008178	11.282619	0.010379	0.0
<b>std</b>	0.499126	0.0	3262.916606	0.134798	135.341399	0.273215	0.0
<b>min</b>	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0
<b>25%</b>	0.000000	0.0	26.000000	0.000000	0.000000	0.000000	0.0
<b>50%</b>	1.000000	0.0	86.000000	0.000000	0.000000	0.000000	0.0
<b>75%</b>	1.000000	0.0	251.000000	0.000000	0.000000	0.000000	0.0
<b>max</b>	1.000000	0.0	224928.000000	18.000000	9108.000000	29.000000	0.0

Figure 4.1: Exploratory analysis of data using different metrics

```

#-----Checking Missing Values-----
clasf_mal.isna().sum()

execve                0
getuid32              0
getgid32              0
geteuid32             0
getegid32            0
..
NrIntReceivers        0
NrIntReceiversActions 0
TotalIntentFilters    0
NrServices             0
Categories            0
Length: 474, dtype: int64

clasf_mal.shape

(41382, 474)

```

Figure 4.2: Checking missing values and showing dataset shape

### 4.2.2 Data Cleaning & Integration

This is the another needed step after data collection/data acquisition. In this step we have handled selected dataset for missing values and null values. There were some null values in the dataset which were causing problem for malware detection. We wrote some python scripts to remove and replace null

## CHAPTER 4. DATA ACQUISITION & PREPROCESSING

values in different attributes/features of dataset. This dataset was two different files of malware and benign samples. After data cleaning we merged them to make a final cleaned dataset having both malware and benign samples. Number of samples was still equivalent to original number of samples but features with missing or null values were fixed. Python library pandas [37] data-frame was used for merging and generating a finalized version of cleaned dataset. A little view of data cleaning code is shown below.

```
{
#Merging_dataset_as_consolidated_dataset
df_final=pd.concat([df_malware,df_legitimate])
#merging_String_values_to_Binary_values
df_final["Activities"].replace({"None":_0},inplace=True)
df_final["NrIntServices"].replace({"None":_0},inplace=True)
df_final["NrIntServicesActions"].replace({"None":_0},inplace=True)
df_final["NrIntActivities"].replace({"None":_0},inplace=True)
df_final["NrIntActivitiesActions"].replace({"None":_0},inplace=True)
df_final["NrIntReceivers"].replace({"None":_0},inplace=True)
df_final["NrIntReceiversActions"].replace({"None":_0},inplace=True)
df_final["TotalIntentFilters"].replace({"None":_0},inplace=True)
df_final["NrServices"].replace({"None":_0},inplace=True)
#Saving_after_removing_none_values&merging
df_final.to_csv('/content/drive/MyDrive/Android_Malware_Detection
_Work/Output_CSVs/Whole_BD_leg_mal_clean_v3.csv',index=False)
df_final.shape
}
```

### 4.2.3 Data Labeling

This dataset came with different malware and benign samples and with further details of describing malware families names across malicious samples. But this dataset was missing malware categories names. As our proposed solution was generating a machine learning model for detection of Android malwares and further classification of malwares into their categories to limit and understand the level of threat. Therefore it was needed to add malware categories across each of the malicious sample. We have used Virustotal [33], Fsecure [34], Fortigaard [35] online malware detection and classification repositories to add malware categories name's across each of malicious sample. We have achieved this using hash of malicious sample and their family name. After finding their categories we have labeled malicious samples with their categories name's using python and machine learning scripts on jupyter notebook. In this process there was over lapping in some categories with each other. Like some of samples were classified as Riskware by one Antivirus and Adware by other Antivirus. We solved this issue by first cross checking the categories of those samples from other published datasets and in case category or sample was not found it was labelled to that category to which it was identified by most number of Antiviruses. At this stage our data labeling was completed with 70% of confirmed categories labeling. There are 30% samples which still need confirmation in categories and it is included in future work of this study. After labeling malicious samples were assigned their respective categories. Below is a glimpse of data labeling with code.

{

## CHAPTER 4. DATA ACQUISITION & PREPROCESSING

```
#Adding_categories_across_families
csv_input["Categories"].replace({"LinuxLotoor":_ "Trojan"}
, inplace=True)
csv_input["Categories"].replace({"Hqwar":_ "Trojan"}
, inplace=True)
csv_input["Categories"].replace({"Robtes":_ "Trojan"}
, inplace=True)
csv_input["Categories"].replace({"Hiddad":_ "Backdoor"}
, inplace=True)
csv_input["Categories"].replace({"Fobus":_ "Backdoor"}
, inplace=True)
csv_input["Categories"].replace({"Kmin":_ "Backdoor"}
, inplace=True)
csv_input["Categories"].replace({"DroidKungFu":_ "Backdoor"}
, inplace=True)
csv_input["Categories"].replace({"FakeApp":_ "Scareware"}
, inplace=True)
csv_input["Categories"].replace({"DroidRooter":_ "Trojan-Dropper"}
, inplace=True)
csv_input["Categories"].replace({"BeanBot":_ "Trojan-SMS"}
, inplace=True)
csv_input["Categories"].replace({"Biige":_ "Trojan-SMS"
}, inplace=True)
csv_input["Categories"].replace({"FakeNotify":_ "Trojan-SMS"}
, inplace=True)
```



## CHAPTER 4. DATA ACQUISITION & PREPROCESSING

```
csv_input["Categories"].replace({"Leech":_ "File-Infector"}
, inplace=True)
csv_input["Categories"].replace({"Jifake":_ "Trojan-SMS"}
, inplace=True)
csv_input["Categories"].replace({"Nandrobox":_ "Trojan-SMS"}
, inplace=True)
csv_input["Categories"].replace({"Viser/Vserv":_ "Adware"}
, inplace=True)
csv_input["Categories"].replace({"Lovetrap/Luvrtrap":_ "Trojan"}
, inplace=True)

#-----Replacing_Nan_Values-----#

csv_input["Categories"].replace({np.nan:_ "Blank-Cat"}
, inplace=True)
#-----Saving_After_Adding_categories_names-----#
csv_input.to_csv('real_malware_after_Cat_fixing_v3.0.csv'
, index=False)
}
```

### 4.2.4 Data Normalization/Scaling

Data normalization/scaling is an optional but important step in machine learning pipeline. It not only increase the accuracy of algorithms but also boosts their performance. This research work have used MinMax Scaling [45]

CHAPTER 4. DATA ACQUISITION & PREPROCESSING

as data normalization/scaling technique after getting inspired from the Imtiaz et. al [46] work as by using same technique they achieved good results. We have tested our solution with and without data normalization. We trained our models on selected algorithms by first feeding the non scaled features and noted the results. After that we performed data scaling/normalization and then feed algorithms again with those features. There was significant improvements in results in some algorithms. Therefore we have selected normalized data and then applied feature selection to generate our final feature set for the development of malware detection model. Figure 4.3. shows the section of dataset before and after data normalization.

Before Data Normalization /Scaling												
Malware	execve	getuid32	getgid32	geteuid32	getegid32	getresuid32	getresgid32	readahead	...	NrIntActivities	NrIntActivitiesActions	NrIntReceiv
1	0	1845	0	0	0	0	0	0	0 ...	1	1	
1	0	1128	0	0	0	0	0	0	0 ...	1	1	
1	0	0	0	0	0	0	0	0	0 ...	1	1	
1	0	25	0	0	0	0	0	0	0 ...	1	1	
1	0	187	0	0	0	0	0	0	0 ...	1	1	

After Data Normalization/Scaling																					
	0	1	2	3	4	5	6	7	8	9	...	463	464	465	466	467	468	469	470	471	472
0	0.0	0.011325	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.051926	0.000000	0.000000	0.001825	0.001825	0.000000	0.000000	0.001757	0.000000
1	0.0	0.006924	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.006700	0.040816	0.046512	0.001825	0.001825	0.040816	0.000822	0.007030	0.012658
2	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.001675	0.000000	0.000000	0.001825	0.001825	0.000000	0.001645	0.005272	0.006329
3	0.0	0.000153	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.001675	0.000000	0.000000	0.001825	0.001825	0.000000	0.001645	0.003515	0.006329
4	0.0	0.001148	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.008375	0.000000	0.000000	0.001825	0.001825	0.000000	0.003289	0.003515	0.000000

Figure 4.3: Data Scaling / Normalization view

# Chapter 5

## Feature Selection & Model

### Training

Feature selection is basically the selection of most related features for algorithm to obtain finest results and also minimize the computational cost. This also improves the performance of algorithms by removing irrelevant or very low impact features. As mentioned in previous phase we have applied data scaling before final features selection but we have also tested feature's selected without applying data scaling. We will describe whole process of features selection phase carried out in our research work.

#### 5.1 Feature Selection

There are different techniques for features selection but they are mainly divided into three techniques: Filter methods, wrapper methods and embedded methods. Wrapper and embedded methods are computationally ex-

## CHAPTER 5. FEATURE SELECTION & MODEL TRAINING

expensive [58] and these methods are not suggested to use when data set is large and has wide dimensions. Therefore we choose filter methods for features selection. From filtering methods we have used ExtraTreeClassifier and Mutual-Information algorithms as our features selection algorithms. These algorithms were selected on the basis of good malware detection results of Abir et.al [47] and El Fiky et. al [48] research work obtained by using same algorithms. Features selected using both techniques showed good results but ExtraTreeClassifier's selected features outperformed Mutual Information's selected features. We have tried different types of features set to test that on which features set selected algorithm are giving best results with also minimizing the computational head. We select six types of features set through each of feature selection algorithm totaling in 12 different group/set of features. As we are performing malware detection and categories classification therefore 12 different group of features were selected for detection phase and similarly 12 different group of features were selected for categories classification phase. These group/set of features are listed in table 5.1.

In final selection we choose best 50 features selected using ExtraTreeClassifier on normalized data in both phases of malware detection and malware category classification. Table 5.2 describes the Top50 features selected by ExtraTree Classifier in detection phase. Similarly Top50 features for category classification phase selected using ExtraTree Classifier are listed in table 5.3.

CHAPTER 5. FEATURE SELECTION & MODEL TRAINING

No of Features	Feature Selection Algorithm
Top30	ExtraTree Classifier
Top30 minmax	Normalization and ExtraTree Classifier
Top50	ExtraTree Classifier
Top50 minmax	Normalization and ExtraTree Classifier
Top100	ExtraTree Classifier
Top100 minmax	Normalization and ExtraTree Classifier
Top30	Mutual Information
Top30 minmax	Normalization and Mutual Information
Top50	Mutual Information
Top50 minmax	Normalization and Mutual Information
Top100	Mutual Information
Top100 minmax	Normalization and Mutual Information

Table 5.1: Listing of 12 generated features set

CHAPTER 5. FEATURE SELECTION & MODEL TRAINING

Feature Name	Feature Type
455 (WRITE_VOICEMAIL)	Permission
461 (nr_custom)	Number of Custom Permissions
457 (normal)	Normal permissions
399 (READ_PHONE_NUMBERS)	Permission
48 (msync)	System Call
456 (nr_permissions)	Number of Permissions
470 (NrIntReceivers)	Number of intent receivers
472 (TotalIntentFilters)	Total Intent Filters
37 (acct)	System Call
235 (SYS_316)	System Call
132 (clock_adjtime)	System Call
228 (SYS_309)	System Call
32 (chroot)	System Call
298 (ACCESS_NOTIFICATION_POLICY)	Permission
38 (read)	System Call
55 (mincore)	System Call
420 (SEND_RESPOND_VIA_MESSAGE)	Permission
96 (fadvise64_64)	System Call
233 (SYS_314)	System Call
23 (getrusage)	System Call
251 (SYS_332)	System Call
202 (gettimeofday)	System Call
471 (NrIntReceiversActions)	Number of Intent Receivers Actions
170 (getcpu)	System Call
49 (mprotect)	System Call
254 (SYS_335)	System Call
151 (setsockopt)	System Call
224 (SYS_305)	System Call
458 (dangerous)	Dangerous permissions
229 (SYS_310)	System Call
93 (truncate64)	System Call
223 (SYS_304)	System Call
87 (isseek)	System Call
57 (readv)	System Call
61 (fchmod)	System Call
43 (pwritev)	System Call
288 (SYS_369)	System Call
406 (REBOOT)	Permission
1 (execv)	System Call
213 (pread)	System Call
203 (clone)	System Call
408 (RECEIVE_MMS)	Permission
86 (utimensat)	System Call
146 (getpeername)	System Call
449 (WRITE_CONTACTS)	Permission
150 (shutdown)	System Call
64 (dup3)	System Call
26 (ugetrlimit)	System Call
247 (SYS_328)	System Call
200 (set_thread_area)	System Call

Table 5.2: Top50 features using ExtraTree classifier for Level 1 Classification

CHAPTER 5. FEATURE SELECTION & MODEL TRAINING

Feature Name	Feature Type
420 (SEND_SMS)	Permission
408 (RECEIVE_SMS)	Permission
292 (ACCESS_COARSE_LOCATION)	Permission
296 (ACCESS_NETWORK_STATE)	Permission
406 (RECEIVE_BOOT_COMPLETED)	Permission
401 (READ_SMS)	Permission
293 (ACCESS_FINE_LOCATION)	Permission
459 (custom_yes)	System Call
298 (ACCESS_WIFI_STATE)	Permission
343 (CALL_PHONE)	Permission
351 (CHANGE_WIFI_STATE)	Permission
451 (WRITE_SECURE_SETTINGS)	Permission
395 (READ_EXTERNAL_STORAGE)	Permission
461 (total_perm)	Total Permissions
457 (dangerous)	Dangerous permissions
384 (MOUNT_UNMOUNT_FILESYSTEMS)	Permission
365 (GET_TASKS)	Permission
362 (GET_ACCOUNTS)	Permission
456 (normal)	Normal Permissions
435 (SYSTEM_ALERT_WINDOW)	Permission
455 (nr_permissions)	Number of Permission
176 (sysinfo )	System Call
470 (NrIntReceiversActions)	Number of Intent receiver actions
462 (FilesInsideAPK)	Total Files inside APK
443 (VIBRATE)	Permission
407 (RECEIVE_MMS)	Permission
258 (SYS_340)	System Call
399 (READ_PHONE_STATE)	Permission
444 (WAKE_LOCK)	Permission
460 (nr_custom)	Number of custom features
472 (NrServices)	Number of Services
314 (BIND_DEVICE_ADMIN)	Permission
339 (BROADCAST_SMS)	Permission
458 (signature)	Signature permissions
394 (READ_CONTACTS)	Permission
348 (CHANGE_CONFIGURATION)	Permission
464 (Activities)	Number of Activities
418 (RESTART_PACKAGES)	Permission
349 (CHANGE_NETWORK_STATE)	Permission
251 (SYS_333)	System Call
340 (BROADCAST_STICKY)	Permission
55 (ioctl)	System Call
288 (SYS_362)	System Call
213 (getrlimit)	System Call
26 (setrlimit)	System Call
257 (SYS_339)	System Call
200 (clock_gettime)	System Call
397 (READ_LOGS)	Permission
452 (WRITE_SETTINGS )	Permission
449 (WRITE_EXTERNAL_STORAGE)	Permission

Table 5.3: Top50 features using ExtraTree classifier for Level 2 Classification

## 5.2 Model Training

This is another crucial step of machine learning pipeline. In this step, we choose machine learning algorithm and then input it selected features to train the model. We chose four popular supervised learning algorithms: Random Forest, Decision Tree, K-Nearest Neighbour and State Vector Machine which were discussed in chapter 3. We split our dataset into two parts: 70% for training and 30% for testing. We trained each of the algorithm with 12 of each feature group/set. KNN and SVM take little longer but their performance is also good like other two algorithms. In initial phase of malware detection we had total **samples of 78137** including **36755 benign** and **41382 malicious** samples. But in the second stage of categories classification we only had **41382 malicious** samples as the objective of this phase was further classification of malicious samples into their categories. These all malicious samples had different set of samples of each category of malwares. We trained all algorithms with the features of all of these samples. We had **14 categories** of malwares and **1 Unknown/Blank** category. Number of samples of each category and all different categories are shown in table 5.4 .



Malware Category	No of Sample's
Adware	11185
Trojan	6690
Trojan-SMS	6475
Riskware	5340
Trojan-Spy	4281
Ransomware	1964
Trojan-Banker	1681
Backdoor	1095
Scareware	1036
PUA	657
File-Infector	436
Spyware	260
Trojan-Dropper	62
Trojan/Riskware	54
Blank-Category	166

Table 5.4: Malware categories with number of samples

### 5.3 Summary

In this chapter we have explained all steps of feature selection and model training. We have also described algorithms used for features selection and model training. In table of top features we have highlighted top features along with their type. In upcoming chapter we will discuss evaluation results and evaluation metrics used for testing and results collection.

# Chapter 6

## Model Evaluation & Tuning

This chapter will be detailing about the trained model's evaluation/testing and further about improving the results by optimization of algorithms. Model evaluation is performed across different evaluation metrics which will also be described in this chapter. At the end we will discuss obtained results and compare these results with other similar existing research works. Lets start with evaluation process step by step.

### 6.1 Evaluation Metrics

As the evaluation process is critical so the evaluation metrics are. We chose below mentioned evaluation metrics but accuracy is our main targeted evaluation metric:

- Accuracy
- Precision

## CHAPTER 6. MODEL EVALUATION & TUNING

- F1-Score
- Recall

These metrics with their formulas are defined [59] below but before that we need to understand some of abbreviations and important elements of these metrics. Following are the different metrics with their abbreviations.

- True Positive= TP
- False Positive= FP
- True Negative= TN
- False Negative= FN

Before defining these parameters lets consider malware = negative and benign = positive

### **True Positive**

True positive is the equivalence/match of actual value with the predicted value. In case of malware true positive is: The actual value was benign and model predicted benign.

### **True Negative**

True negative is the equivalence/match of actual value with the predicted value. In case of malware true negative is: The actual value was malware and model also predicted malware.

## CHAPTER 6. MODEL EVALUATION & TUNING

### False Positive

False positive is defined as the expected value was incorrectly predicted. In case of malware false positive is: The actual value was malware but the model predicted benign.

### False Negative

False negative is defined as the expected value was incorrectly predicted. In case of malware false negative is: The actual value was benign and model predicted malware.

### Confusion Matrix

Confusion matrix is  $c \times c$  matrix used for the evaluation of machine learning model where  $c$  is number of classes. This matrix compares predicted values of ML model with actual values. The confusion matrix formed for malware detection is shown in figure 6.1.

#### 6.1.1 Accuracy

Accuracy is best described as the quality of measurement or data of being true, correct, or exact and freedom from mistakes and errors. Formula of accuracy is given below:

$$Accuracy(Acc) = \frac{TP + TN}{TP + TN + FP + FN}$$

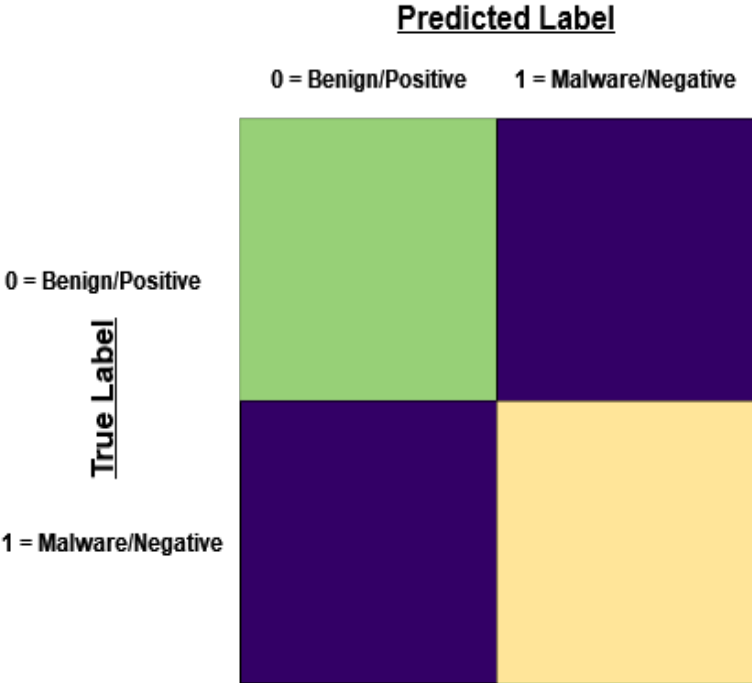


Figure 6.1: Confusion matrix for malware detection

### 6.1.2 Precision

Precision is termed as the percentage of correct positive predictions.

$$Precision = \frac{TP}{TP + FP}$$

### 6.1.3 Recall

Recall is what percentage of actual positives were correctly classified.

$$Recall = \frac{TP}{TP + FN}$$

### 6.1.4 F-1 Score

The F1-score computes a single score by taking the harmonic mean of a classifier's precision and recall. It is usually used to make a comparison of the performance of two classifiers. Assume classifier A has more recall and classifier B has greater precision. Then in this scenario F1-scores of both classifiers will be used to determine whichever classifier outperforms in this scenario.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

## 6.2 Results and Evaluation

As we have discussed in chapter 2 our selected machine learning algorithms for model training and now we will discuss their results.

### 6.2.1 Initial Results

As mentioned in chapter 5 we split data into 70/30 and trained the Random Forest (RF), Decision Tree (DT) , K-Nearest Neighbour (KNN) and Support Vector Machines (SVM) classifiers with the 70% of the training data. Remaining 30% was used for the evaluation (testing) of trained models. As we choose 6 different group/set of features for using each of feature selection algorithms (ExtraTree Classifier and Mutual Information), we evaluated results of all features set. From the evaluation we observed that Top50 features selected using ExtraTree Classifier is giving best accuracy, Therefore we chose **Top50 features selected using ExtraTree Classifier on normalized data**. In comparison of different classifiers model trained using Random forest outperformed other classifiers by giving an accuracy of **97.98%** in malware detection and **87.24%** accuracy on malware category classification. Other classifiers also performed with good results.

Table 6.1 summarizes results on accuracy metric of binary malware detection using ExtraTree Classifier as feature selection algorithm and Random forest(RF), Decision Tree(DT), K-Nearest Neighbour(KNN) and State Vector Machine (SVM) as machine learning classifier's on different set/group of features while table 6.2 summarizes same accuracy metric results using Mutual Information as feature engineering algorithm. Similarly results on accuracy metric of category classifications using ExtraTree classifier and Mutual Information as feature selection algorithm are listed in table 6.3 and 6.4 respectively.

CHAPTER 6. MODEL EVALUATION & TUNING

<i>Features</i>	<i>RF</i>	<i>DT</i>	<i>KNN</i>	<i>SVM</i>
<i>Top30</i>	95.99%	96.82%	91.53%	84.28%
<i>Top30 Minmax</i>	96.33%	95.58%	95.62%	95.25%
<i>Top50</i>	97.72%	95.88%	86.66%	68.66%
<i>Top50 Minmax</i>	<b>97.98%</b>	96.44%	96.78%	95.26%
<i>Top100</i>	97.72%	95.58%	89.90%	69.17%
<i>Top100 Minmax</i>	97.73%	96.10%	96.85%	95.47%

Table 6.1: Level 1 Binary Detection results(Accuracy) using ExtraTree classifier

<i>Features</i>	<i>RF</i>	<i>DT</i>	<i>KNN</i>	<i>SVM</i>
<i>Top30</i>	96.57%	94.88%	90.47%	82.03%
<i>Top30 Minmax</i>	96.63%	94.94%	95.28%	91.06%
<i>Top50</i>	96.85%	95.14%	88.97%	73.08%
<i>Top50 Minmax</i>	96.84%	95.21%	95.66%	92.03%
<i>Top100</i>	97.51%	81.78%	89.66 %	74.14%
<i>Top100 Minmax</i>	97.55%	82.55%	96.77%	95.54%

Table 6.2: Level 1 Binary Detection results (Accuracy) using Mutual Information

<i>Features</i>	<i>RF</i>	<i>DT</i>	<i>KNN</i>	<i>SVM</i>
<i>Top30</i>	85.13%	82.55%	77.47%	43.53%
<i>Top30 Minmax</i>	85.67%	83.48%	82.95%	79.35%
<i>Top50</i>	87.06%	81.78%	68.28%	39.91%
<i>Top50 Minmax</i>	<b>87.24%</b>	82.47%	84.29%	81.10%
<i>Top100</i>	86.99%	81.78%	72.36%	41.18%
<i>Top100 Minmax</i>	86.79%	82.55%	84.72%	80.78%

Table 6.3: Level 2 Categories Classification results(Accuracy) using Extra-Tree classifier



<i>Features</i>	<i>RF</i>	<i>DT</i>	<i>KNN</i>	<i>SVM</i>
<i>Top30</i>	83.89%	77.87%	71.92%	40.93%
<i>Top30 Minmax</i>	84.01%	77.78%	79.75%	60.53%
<i>Top50</i>	85.71%	79.64%	72.62%	41.19%
<i>Top50 Minmax</i>	85.70%	80.35%	81.69%	61.48%
<i>Top100</i>	86.54%	81.49%	72.40%	40.71%
<i>Top100 Minmax</i>	86.81%	82.35%	83.70%	77.30%

Table 6.4: Level 2 Categories Classification results(Accuracy) using Mutual Information

## 6.2.2 Model Tuning & Final Results

Random forest produced satisfactory results but there was gap in results improvements which could be filled by a process in machine learning called hyper-parameters tuning. Hyper parameters [60] are classifier specific parameters that control the learning rate during training and are set before the model is trained. Initially we trained all of machine learning classifiers with default parameters and obtained results. For final results selection we applied hyper parameter tuning using GridSearchCV [49]. It is a python library which facilitate the process of selecting best parameters for any machine learning algorithm. There are also other techniques like random search and brute force method but GridSearchCV performs better as it selects grid of hyper parameter values and compares them to get the best set of grid. After applying hyper-parameter tuning on Top50 features selected using extra-tree classifier on normalized data our final results improved on all machine learning classifiers. Table 6.5 lists the final results of all classifiers with hyper-parameter tuning in binary detection phase on different evaluation metrics

CHAPTER 6. MODEL EVALUATION & TUNING

while figure 6.2 shows the confusion matrix of RF for binary detection phase. For the category classification phase we also tested hyper-parameter tuning on all classifiers and their results also improved, but Random Forest was still giving best results. Figure 6.3 shows the final results of Random Forest algorithm on different metrics with an average accuracy of **87.56%** and table 6.6 shows the final results of all classifiers in category classification phase. It shows that most of categories are classified correctly.

<i>Algorithm</i>	<i>Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-Score</i>
<b>RF</b>	<b>98.03%</b>	<b>98.51%</b>	<b>97.73 %</b>	<b>98.12%</b>
<b>DT</b>	96.60%	97.12%	96.45%	96.78%
<b>KNN</b>	97.16%	97.24%	97.44%	97.34%
<b>SVM</b>	96.39%	97.65%	95.54%	96.58%

Table 6.5: Final results of Binary Detection

<i>Algorithm</i>	<i>Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-Score</i>
<b>RF</b>	<b>87.56%</b>	<b>90.14%</b>	<b>74.20 %</b>	<b>81.39%</b>
<b>DT</b>	83.09%	74.14 %	70.87 %	72.47%
<b>KNN</b>	84.92%	77.27 %	73.67%	75.20%
<b>SVM</b>	84.78%	78.40%	71.34 %	74.70%

Table 6.6: Final results (Average) of Category Classification

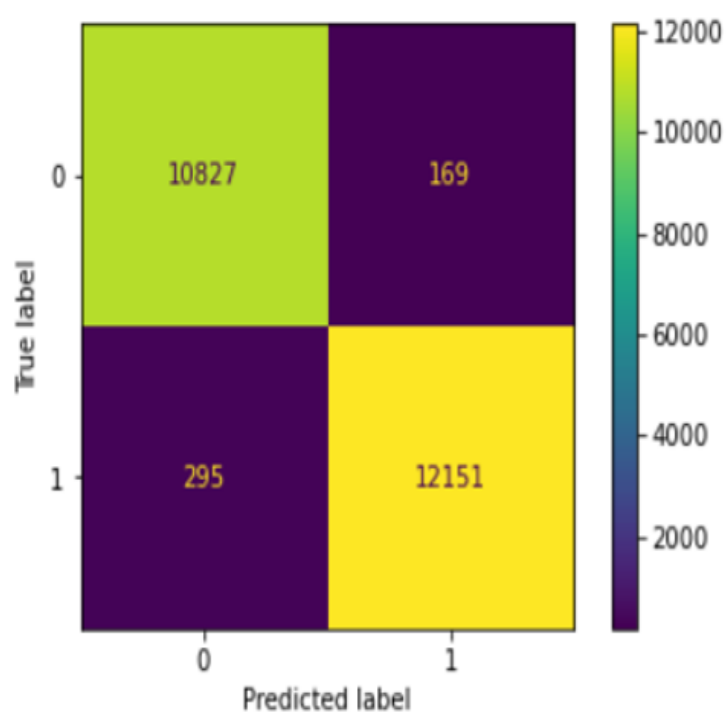


Figure 6.2: Confusion matrix of RF for Binary Detection

```
# Random Forest Classifier Results
```

```
Accuracy: 0.8756343133306485
```

	precision	recall	f1-score
Adware	0.87	0.92	0.89
Backdoor	0.96	0.87	0.91
Blank-Cat	1.00	0.06	0.11
File-Infector	0.88	0.72	0.79
PUA	0.76	0.55	0.64
Ransomware	0.96	0.92	0.94
Riskware	0.93	0.92	0.93
Scareware	0.94	0.87	0.91
Spyware	0.87	0.72	0.79
Trojan	0.74	0.80	0.77
Trojan-Banker	0.92	0.82	0.86
Trojan-Dropper	0.86	0.32	0.46
Trojan-SMS	0.93	0.91	0.92
Trojan-Spy	0.90	0.88	0.89
Trojan/Riskware	1.00	0.85	0.92

Figure 6.3: Category classification Random Forest final results

## 6.3 K-Fold Cross Validation for Results Validation

We tested the effectiveness of our proposed research work using several machine learning classifiers. Although these classifiers have produced good results but to eradicate the problem of over-fitting and to showcase the stability of final model we had to validate the results. We used the K-Fold Cross Validation approach to accomplish this. The K-Fold Cross-Validation method divides the data for training into K folds, with one fold acting as the test set and the other k-1 folds serving as the training set. This cycle is continued K times until each fold in the validation set has played a role. The results of all folds are then averaged. An example of 5-Fold Cross Validation is depicted in Figure 6.4.

We have applied K-Fold cross validation with k=5 value in both stages of binary detection and category classification on all selected machine learning classifiers and on Top50 features selected using ExtraTree Classifier. The validation results of level 1 classification (binary detection) of all classifiers are depicted in figure 6.5 Similarly validation results of level 2 classification (category classification) are shown in figure 6.6 . These results show that our models are accurate and even applied in production environment our final model will produce similar detection and classification results.

CHAPTER 6. MODEL EVALUATION & TUNING

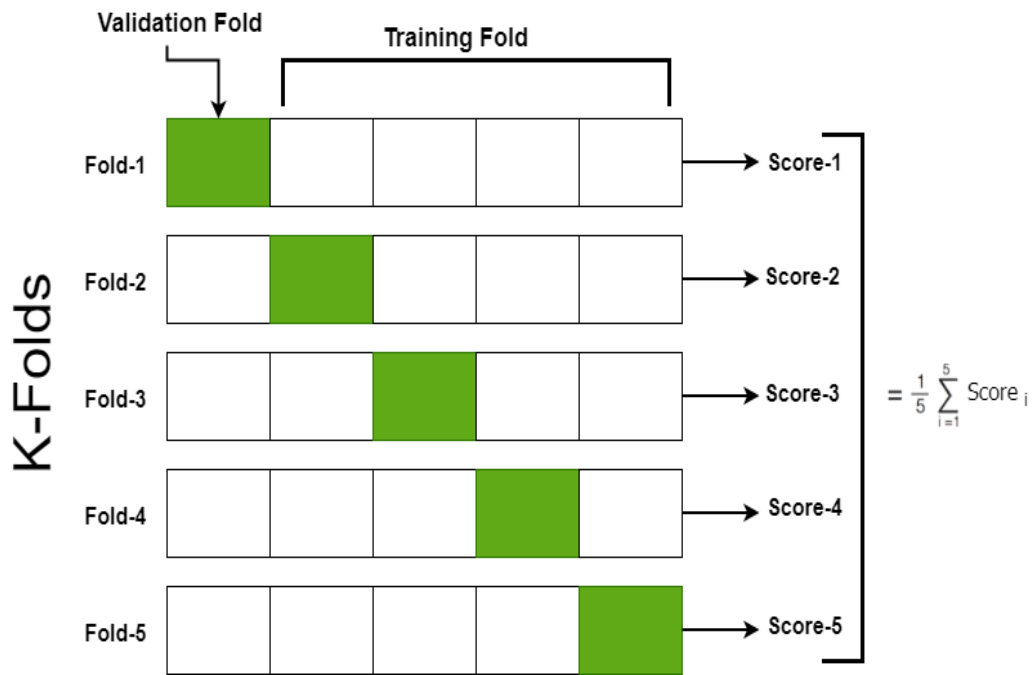


Figure 6.4: K-Fold Cross Validation [61] for K = 5

K-Fold Cross Validation Binary Detection (ETC Top50 Minmax)				
Features	RF	DT	KNN	SVM
Fold-1	98.08%	96.71%	97.01%	96.30%
Fold-2	98.00%	96.73%	97.03%	96.31%
Fold-3	97.97%	96.62%	97.04%	96.36%
Fold-4	97.93%	96.64%	97.06%	96.51%
Fold-5	97.95%	96.28%	97.10%	96.64%
Avg	<b>97.97%</b>	<b>96.60%</b>	<b>97.05%</b>	<b>96.42%</b>

Figure 6.5: K-Fold Cross Validation for Binary Detection

K-Fold Cross Validation Categories Classification (ETC Top50 Minmax)				
Features	RF	DT	KNN	SVM
Fold-1	87.01%	82.55%	84.63%	84.06%
Fold-2	87.13%	82.90%	82.46%	84.02%
Fold-3	87.43%	83.77%	85.10%	84.46%
Fold-4	87.24%	82.82%	84.95%	84.32%
Fold-5	87.21%	82.91%	84.77%	84.37%
Avg	<b>87.20%</b>	<b>82.99%</b>	<b>84.78%</b>	<b>84.25%</b>

Figure 6.6: K-Fold Cross Validation for Categories Classification

## 6.4 Comparison with Existing Work

In the research domain, there has been substantial progress in malware detection. Several researchers have proposed various malware detection solutions based on machine learning, deep learning, and other approaches. When it comes to using any technique of malware detection, selection of appropriate and quality dataset is very important. We have listed research works closely related to our proposed solution with their results in table 6.7 and table 6.8. From all Several researchers have proposed various malware detection solutions based on machine learning, deep learning, and other approaches. of these research work most of the researchers have performed dynamic analysis on emulators which cannot prevent advanced and sophisticated malwares to bypass the malware detection solution developed on the basis of emulated

## *CHAPTER 6. MODEL EVALUATION & TUNING*

analysis process because those malwares can detect emulated environment and do not execute their malicious behaviours or intents. Only DL-Droid [52] have performed dynamic analysis on real devices and they got good results on malware detection using deep learning. Their dataset has 30000 total samples and it did not include time-effect/time-frame of malware samples. Therefore this solution will not be very effective against those malwares which change their behaviour with time which is also called concept drift. Our proposed solution has used Kronodroid [31] dataset which is almost balanced by having 41382 malware samples and 36735 benign samples. Moreover it has samples from complete Android history from 2008-2020 making it effective against concept-drift which is the change in malwares behaviour over time. Further more this dataset has performed dynamic analysis on both emulators and real devices and we choose dataset of real devices as it has been noted in their analysis that some malwares did not execute in emulated environment. These all inclusion of Kronodroid dataset make it efficient for the detection of all type of malwares. To the best of our knowledge currently no malware detection solution has used this dataset. Our solution have used machine learning on Kronodroid dataset by carefully and thoroughly implementing process of machine learning. So to the best of author's knowledge all most relevant and latest research work have above highlighted gaps in malware detection which we have tried to fill in our research work.



## CHAPTER 6. MODEL EVALUATION & TUNING

<i>Research work</i>	<i>Detection Approach and algorithm</i>	<i>Dataset &amp; Year</i>	<i>Features Type</i>	<i>Time Frame</i>
<i>Atzeni et.al [50]</i>	Semi Supervised Learning	Dataset creation 2016	Hybrid(Dynamic on emulator)	N/A
<i>DeepAmd [4]</i>	Deep Artificial Neural Network	CICANDMAL2019%	Static & Dynamic on real device	N/A
<i>Aktas et al [51]</i>	Machine Learning	Updroid	Hybrid (Dynamic on Emulator)	2014-18
<i>DL-Droid [52]</i>	Deep Learning	DL Droid Dataset 2019	Hybrid (Dynamic on real devices)	N/A
<i>MahdaviFar et .al [7]</i>	Deep Neural Network with psuedo label	CICMALDroid2020	Dynamic(Emulator)	2017-18
<i>EntropLyzer [23]</i>	Machine Learning	CCCS-CIC-AndMal2020	Dynamic (Emulator)	N/A
<i>PLSAE [28]</i>	Deep Neural Network with Psuedo Label Stack Auto Encoder	CICMALDroid2020	Hybrid (Dynamic on Emulator)	2017-18
<i>Our Approach</i>	Machine Learning	KronoDroid 2020	Hybrid Dynamic on real devices	2008-2020

Table 6.7: Comparison with existing work

<i>Research work</i>	<i>Binary Detection Results</i>	<i>Categorization Results</i>
<i>Atzeni et.al [50]</i>	91.23% Accuracy	Family classification 95% Homogeneity score
<i>DeepAmd [4]</i>	Static: 93.40% Accuracy	Static 92.5% Accuracy, Dynamic 80.3% Accuracy, 4 Categories
<i>Aktas et al [51]</i>	Detection as categorization	96.37% Accuracy
<i>DL-Droid [52]</i>	98.5% Accuracy	N/A
<i>MahdaviFar et .al [7]</i>	Detection as categorization	97.8% f1 score on 5 categories
<i>EntropLyzer [23]</i>	Detection as categorization	98.4% Precision on 12 categories
<i>PLSAE [28]</i>	98.28% Accuracy	5 Categories
<i>Our Approach</i>	98.03% on detection	87.56% on 15 categories

Table 6.8: Comparison with existing work part 2

## 6.5 Summary

This chapter details the process of malware detection model evaluation and tuning. This chapter first described the evaluation metrics and then discussed the initial and final results in details along with model tuning. Lastly k-fold cross validation results has been discussed along with comparison of proposed

*CHAPTER 6. MODEL EVALUATION & TUNING*

research work with existing closely related studies.

# Chapter 7

## Conclusion & Future Work

### 7.1 Conclusion

As it has been pointed out that Android is market leader in smart devices meanwhile it is the main target of malwares. Android malwares are not only security threat to smart devices using Android operating system but its also a severe danger to its users because users store their personal, private and commercial data on these devices. Despite many efforts in malware detection there are still loop holes for malwares. There is a dire need of more efficient solution to fill the gap in the domain of malware detection. In this research work we have highlighted all deficiencies and proposed a more efficient Android malware detection solution which will work against different type of advanced and behaviour evolving malwares. We have provided a malware detection and categorization solution using Kronodroid dataset on different classifiers of supervised machine learning including random forest, decision tree, k-nearest neighbour and state vector machine. Kronodroid

## *CHAPTER 7. CONCLUSION & FUTURE WORK*

dataset is latest, large, balanced, has include malware samples from all of Android history and has collected dynamic features on real devices. This dataset does not included malware categories. We have first added malware categories in dataset by using different online anti-malware repositories and then performed malware detection and categories classification using machine learning. Our final model yields good results in malware detection with an accuracy of 98.03% and an accuracy of 87.56% in malware category classification through random forest with only top50 features selected using ExtraTree classifier. This selection of best minimal number of features not only enhanced our results but also reduced the computational overhead. Other machine learning classifiers also yield good results in both phases. We have also checked precision, recall and f1-Score in both phases of malware detection and category classifications. Lastly we have performed K-fold cross validation to validate the results of final model. K-fold cross validation also gave almost similar results on average of 5-folds using different classifiers. This makes our solution efficient against all type of malwares either using signature or using behaviour based techniques. Hence this research work provides a machine learning based malware detection and categorization model which can be used in production environment. Novel contribution of this research work are described below.

## 7.2 Thesis Contribution

This research work makes the following novel contributions:

- Machine learning based hybrid malware detection solution. This Malware detection solution is using a comprehensive data-set which has following advantages:
  - Handling concept-drift as it have malware samples from 2008 to 2021.
  - Captured dynamic features on real devices.
- Labeling of dataset by adding malware categories.
- Classification of malwares into their categories.
- Reducing computational overhead by choosing best minimal number of features (top50)

## 7.3 Future Work

There are still some improvements which could be made in this research work in malware categorization and further family classification.

### 7.3.1 Adding more Balanced number of Categories

As balanced dataset gives more accurate results and it has been highlighted in table 5.4 that number of samples of categories are not balanced in dataset used in this research work. It shows that Adware have 11185 samples while

## *CHAPTER 7. CONCLUSION & FUTURE WORK*

Trojan have 6690 samples. This impacts accuracy of the model which could be significantly improved by add more balanced number of samples for all categories.

### **7.3.2 Improvement in Categories Classification**

We have labeled kronodroid dataset of real devices with different malware categories. These malware categories were added by scanning samples on different online repositories. There were some malware categories which were over lapping with each other. Like some of samples were classified as Trojan by one Antivirus and Riskware by other Antivirus. We have confirmed categories of around 70% malware samples. Remaining 30% samples categories still needs a more detailed look. As stated above we are getting 87.56% accuracy on categories classification we believe that this could be improve significantly by validating remaining 30% samples of categories.

### **7.3.3 Performing Malware Family Classification**

Malwares families classification is another next step in malware detection process. This step enhances the malware detection solutions by identifying malwares of similar families. We have not implemented this in our research work due to limited time. We plan to make this step of malware identification part of our research work in near future. We believe that malware detection, malware categories classification and malware family identification gives a complete picture of robust malware detection solution.

# Bibliography

- [1] “Android Global Market Shares.” <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/> (accessed April. 24, 2022).
- [2] “Development of new Android Malware Per Month” <https://www.statista.com/statistics/680705/global-android-malware-volume/> (accessed May. 05, 2022).
- [3] “Android Platform Architecture” <https://developer.android.com/guide/platform> (accessed May. 09, 2022).
- [4] Imtiaz, Syed Ibrahim, et al. ”DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network.” *Future Generation computer systems* 115 (2021): 844-856.
- [5] Rahali, Abir, et al. ”DIDroid: android Malware classification and characterization using deep image learning.” 2020 the 10th International Conference on Communication and Network Security. 2020.
- [6] Keyes, David Sean, et al. ”EntropLyzer: Android Malware Classification and Characterization Using Entropy Analysis of Dynamic Character-

## BIBLIOGRAPHY

- istics.” 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS). IEEE, 2021.
- [7] Mahdavifar, Samaneh, et al. ”Dynamic android malware category classification using semi-supervised deep learning.” 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCCom/CyberSciTech). IEEE, 2020.
- [8] “Malware Definition By Malware Bytes” <https://www.malwarebytes.com/malware> (accessed May. 12, 2022).
- [9] “Malware insertion process” <https://theconversation.com/explainer-how-malware-gets-inside-your-apps-79485> (accessed May. 12, 2022).
- [10] Arora, Anshul, Sateesh K. Peddoju, and Mauro Conti. ”Permpair: Android malware detection using permission pairs.” IEEE Transactions on Information Forensics and Security 15 (2019): 1968-1982.
- [11] apktool; A tool for reverse engineering Android Apks <https://ibotpeaches.github.io/Apktool/> (Accessed December 23, 2021),
- [12] Androguard: Reverse engineering, Malware analysis of Android applications. <https://github.com/androguard/androguard> (Accessed December 23, 2021)



## *BIBLIOGRAPHY*

- [13] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of android malware and android analysis techniques," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 76, 2017.
- [14] S. Das and M. J. Nene, "A survey on types of machine learning techniques in intrusion prevention systems," *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 2017, pp. 2296-2299, doi: 10.1109/WiSPNET.2017.8300169.
- [15] Zhu, H. J., Jiang, T. H., Ma, B., You, Z. H., Shi, W. L., & Cheng, L. (2018).HEMD: a highly efficient random forest-based malware detection framework for Android. *Neural Computing and Applications.*" <https://doi.org/10.1007/s00521-017-2914-y>"
- [16] Nikola Milosevic, Ali Dehghantanha, Kim-Kwang Raymond Choo, Machine learning aided Android malware classification, *Computers & Electrical Engineering*, Volume 61, 2017, Pages 266-274, ISSN 0045-7906, <https://doi.org/10.1016/j.compeleceng.2017.02.013>.
- [17] G. Baldini and D. Geneiatakis, "A Performance Evaluation on Distance Measures in KNN for Mobile Malware Detection," *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2019, pp. 193-198, doi: 10.1109/CoDIT.2019.8820510.
- [18] T. Ban, T. Takahashi, S. Guo, D. Inoue and K. Nakao, "Integration of Multi-modal Features for Android Malware Detection Using Linear SVM," *2016 11th Asia Joint Conference on Information Security (AsiaJCIS)*, 2016, pp. 141-146, doi: 10.1109/AsiaJCIS.2016.29.

## BIBLIOGRAPHY

- [19] H. Bai, N. Xie, X. Di and Q. Ye, "FAMD: A Fast Multifeature Android Malware Detection Framework, Design, and Implementation," in *IEEE Access*, vol. 8, pp. 194729-194740, 2020, doi: 10.1109/ACCESS.2020.3033026.
- [20] Abir Rahali, Arash Habibi Lashkari, Gurdip Kaur, Laya Taheri, FRANCOIS GAGNON, and Frédéric Massicotte. 2020. DIDroid: Android Malware Classification and Characterization Using Deep Image Learning. In 2020 the 10th International Conference on Communication and Network Security (ICCNS 2020). Association for Computing Machinery, New York, NY, USA, 70–82. <https://doi.org/10.1145/3442520.3442522>
- [21] G. Tao, Z. Zheng, Z. Guo, and M. Lyu, MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs, *IEEE TRANSACTIONS ON RELIABILITY*, 67(1), 355-369, 201
- [22] L. Cen, C. Gates, L. Si, and N. Li, A Probabilistic Discriminative Model for Android Malware Detection with Decompiled Source Code, *IEEE Transactions On Dependable And Secure Computing*, 12(4), 400-412, 2015
- [23] D. S. Keyes, B. Li, G. Kaur, A. H. Lashkari, F. Gagnon and F. Massicotte, "EntropLyzer: Android Malware Classification and Characterization Using Entropy Analysis of Dynamic Characteristics," 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS), 2021, pp. 1-12, doi:10.1109/RDAAPS48126.2021.9452002.

## BIBLIOGRAPHY

- [24] A. Desnos and P. Lantz. Droidbox: An android application sandbox for dynamic analysis. Lund Univ., Lund, Sweden, Tech. Rep, 2011.
- [25] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.- P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In 2012 Seventh Asia Joint Conference on Information Security, pages 62–69. IEEE, 2012.
- [26] Android malware toolkit. <http://dunkelheit.com.br/amat/analysis/index.en.php>, accessed Mar, 2018.
- [27] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou. Deep ground truth analysis of current android malware. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 252–276. Springer, 2017
- [28] Mahdavifar, Samaneh, Dima Alhadidi, and Ali Ghorbani. "Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder." *Journal of Network and Systems Management* 30.1 (2022): 1-34.
- [29] Ali-Gombe, Aisha I., et al. "Toward a more dependable hybrid analysis of android malware using aspect-oriented programming." *computers & security* 73 (2018): 235-248.
- [30] Surendran, Roopak, Tony Thomas, and Sabu Emmanuel. "A TAN based hybrid model for android malware detection." *Journal of Information Security and Applications* 54 (2020): 102483

## BIBLIOGRAPHY

- [31] Alejandro Guerra-Manzanares, Hayretdin Bahsi, Sven Nõmm, KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization, Computers & Security, Volume 110, 2021, 102399, ISSN 0167-4048, <https://doi.org/10.1016/j.cose.2021.102399>.
- [32] “Welcome to Python.org.” <https://www.python.org/> (accessed May,10, 2022).
- [33] “Virus Total.” <https://www.virustotal.com/gui/home/search> (accessed May,10, 2022).
- [34] “Fsecure.” <https://www.f-secure.com/v-descs/virus.shtml/> (accessed May,10, 2022).
- [35] “Fortiguard.” <https://www.fortiguard.com/search?q=TrojanSMS.Stealer&engine=1/> (accessed May,10, 2022).
- [36] “NumPy.” <https://numpy.org/> (accessed May,10, 2022).
- [37] “Pandas - Python Data Analysis Library.” <https://pandas.pydata.org/> (accessed May,10, 2022).
- [38] “Matplotlib: Python plotting — Matplotlib 3.4.3 documentation.” <https://matplotlib.org/> (accessed May,10, 2022).
- [39] “Scikit-learn: machine learning in Python — scikit-learn 1.0.1 documentation.” <https://scikit-learn.org/stable/> (accessed May,10, 2022).
- [40] “Project Jupyter — Home.” <https://jupyter.org/> (accessed May,10, 2022).

## BIBLIOGRAPHY

- [41] “Welcome to Colaboratory - Colaboratory.”  
<https://colab.research.google.com/> (accessed May,10, 2022).
- [42] “PyCharm: the Python IDE for Professional Developers by JetBrains.”  
<https://www.jetbrains.com/pycharm/> (accessed May,10, 2022).
- [43] “Latex by overleaf” <https://www.overleaf.com/> (accessed May,10, 2022).
- [44] “Latex by overleaf” <https://github.com/aleguma/kronodroid/tree/main/realdevice/>  
(accessed May,10, 2022).
- [45] “MinMax Scaling By Scikit-Learn” <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>  
(accessed May,12, 2022).
- [46] Imtiaz, Syed Ibrahim et al. “DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network.” *Future Gener. Comput. Syst.* 115 (2021): 844-856.
- [47] Abir Rahali, Arash Habibi Lashkari, Gurdip Kaur, Laya Taheri, FRANCOIS GAGNON, and Frédéric Massicotte. 2020. DIDroid: Android Malware Classification and Characterization Using Deep Image Learning. In 2020 the 10th International Conference on Communication and Network Security (ICCNS 2020). Association for Computing Machinery, New York, NY, USA, 70–82. <https://doi.org/10.1145/3442520.3442522>

## BIBLIOGRAPHY

- [48] El Fiky, Ahmed & Elshenawy Elsefy, Ayman & Madkour, Mohamed. (2021). Android Malware Category and Family Detection and Identification using Machine Learning.
- [49] “GridSearchCV ” [https://scikitlearn.org/stable/modules/generated/sklearn.model\\_selection](https://scikitlearn.org/stable/modules/generated/sklearn.model_selection) (accessed May,13, 2022).
- [50] A. Atzeni, F. Díaz, A. Marcelli, A. Sánchez, G. Squillero and A. Tonda, ”Countering Android Malware:A Scalable SemiSupervised Approach for FamilySignature Generation,” in *IEEE Access*,vol. 6, pp. 5954059556, 2018, doi: 10.1109/ACCESS.2018.2874502.
- [51] Aktas, K., Sen, S.2018. UpDroid: Updated Android Malware and Its Familial Classification. In: Gruschka, N. eds *Secure IT Systems. NordSec 2018. Lecture Notes in Computer Science()*, vol 11252. Springer, Cham. [https://doi.org/10.1007/9783030036386\\_22](https://doi.org/10.1007/9783030036386_22)
- [52] Mohammed K. Alzaylaee, Suleiman Y. Yerima, Sakir Sezer,DL-Droid: Deep learning based android malware detection using real devices, *Computers & Security*,Volume 89,2020,101663,ISSN 0167-4048, <https://doi.org/10.1016/j.cose.2019.101663>.
- [53] “Flubot ” <https://www.zdnet.com/article/passwordstealingandroid-malwaretricksyouintodownloadingitbyclaimingyourphoneinalready-infected/> (accessed May,25, 2022).
- [54] W. Wang et al., ”Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions,” in *IEEE Access*, vol. 7, pp. 67602-67631, 2019, doi: 10.1109/ACCESS.2019.2918139.

## BIBLIOGRAPHY

- [55] “Machine Learning Definition ” [https://www.sas.com/en\\_us/insights/analytics/machine-learning.html](https://www.sas.com/en_us/insights/analytics/machine-learning.html) (accessed May,25, 2022).
- [56] Mohammed K. Alzaylaee, Suleiman Y. Yerima, and Sakir Sezer. 2017. EMULATOR vs REAL PHONE: Android Malware Detection Using Machine Learning. In Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics (IWSPA '17). Association for Computing Machinery, New York, NY, USA, 65–72. <https://doi.org/10.1145/3041008.3041010>
- [57] “Exploratory Data Analysis and ML process ” <https://www.analyticsvidhya.com/blog/2020/12/understandmachine-learninganditsendtoendprocess/> (accessed May,25, 2022).
- [58] Biswas, Saroj,et al. ”Review on Feature Selection and Classification using Neuro-Fuzzy Approaches.” IJAEC vol.7, no.4 2016: pp.28-44. <http://doi.org/10.4018/IJAEC.2016100102>
- [59] “Confusion Matrix ” <https://towardsdatascience.com/understanding-confusionmatrix-a9ad42dcfd62> (accessed May,27, 2022).
- [60] “Hyper Parameters Tuning ” [https://scikit-learn.org/stable/modules/grid\\_search.html](https://scikit-learn.org/stable/modules/grid_search.html) (accessed May,27, 2022).
- [61] “K Fold Cross Validation ” [https://scikit-learn.org/stable/modules/cross\\_validation.html?highlight=kfold%20cross%20validation](https://scikit-learn.org/stable/modules/cross_validation.html?highlight=kfold%20cross%20validation) (accessed May,27, 2022).