ANDROID MALWARE DETECTION, CLASSIFICATION AND THREAT ASSESSMENT USING COMPUTATIONAL TECHNIQUES

Thesis submitted in fulfillment of the requirements for the Degree of

DOCTOR OF PHILOSOPHY

By

MEGHNA DHALARIA

Department of Computer Science & Engineering and Information Technology

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

Waknaghat, Solan-173234, Himachal Pradesh, INDIA

October, 2021

TABLE OF CONTENTS

DECLARATION BY THE SCHOLAR

I hereby declare that the work reported in the Ph.D. thesis entitled **"Android Malware Detection, Classification and Threat Assessment using Computational Techniques"** submitted at **Jaypee University of Information Technology, Waknaghat, India,** is an authentic record of my work carried out under the supervision of **Dr. Ekta Gandotra**. I have not submitted this work elsewhere for any other degree or diploma. I am fully responsible for the contents of my Ph.D. thesis.

Meghna Dhalaria
Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat, India
Date:

SUPERVISOR’S CERTIFICATE

This is to certify that the work in the thesis entitled **“Android Malware Detection, Classification and Threat Assessment using Computational Techniques”** submitted by **Meghna Dhalaria at Jaypee University of Information Technology, Waknaghat, India,** is a bonafide record of her original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma.

Dr. Ekta Gandotra
Assistant Professor (Senior Grade)
Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat, India
Date:

ACKNOWLEDGEMENT

With the providential grace of "**Almighty God**", the expedition of my Ph.D. came to an end and my heart is overflowing with appreciation towards each and every person who has lend a hand in the form of support, believe and efforts to accomplish this journey.

It is an immense pleasure to express my profound gratitude towards my supervisor **Dr. Ekta Gandotra**, Assistant Professor, Department of Computer Science & Engineering and Information Technology who graciously gave me the opportunity to work under her guidance. I am thankful for her patience, continuous support, optimistic approach, never-ending deliberations, time to time guidance and liberty throughout this course. Their valuable motivation, help, suggestion, affirmative vision, magnificent supervision and enormous confidence in my abilities made me face tough circumstances during the progress of the research work. It is a great honour to work under her supervision.

I would like to express my gratitude to our Honorable Vice Chancellor Prof. (Dr.) Rajendra Kumar Sharma and Dean (Academics and Research) Prof. (Dr.) Ashok Kumar Gupta and Head Department of CSE/IT Prof. (Dr.) Vivek Kumar Sehgal to promote the research and facilitate resources in the institution. I would also like to thank my doctoral committee members Dr. Pradeep Kumar Gupta, Dr. Ravindara Bhatt and Dr. Harsh Sohal for their valuable feedback and critical reviews during presentations and time to time help.

I am also grateful for the support received from JUIT, Waknaghat. In particular, I thank, Mr. Ravi Raina and all staff members at the Department of Computer Science and Engineering and Information Technology, JUIT, Waknaghat who has been extremely helpful at numerous occasions. I thank my fellow Ph.D. friends for their consistent help and valuable discussion. Last but not the least, I would like to thank my parents for supporting me spiritually throughout my life.

Meghna Dhalaria

ABSTRACT

In today's era, mobile devices are getting popular with a variety of applications (apps) to make our life easier. Several mobile Operating Systems (OS) are available in the market including iOS, Android, BlackBerry and Windows Phone. Android is a widely used mobile OS with a market share of more than 85%. It is based on Linux kernel specifically built for touchscreen devices such as tablets and smartphones etc. In the current era, there is an increase in the usage of smartphones for a variety of purposes like banking, social media, education etc. The growing popularity of Android apps has lured attackers to create malicious apps which pose several threats such as financial loss, information leakage etc. These malicious apps are becoming more sophisticated and using new ways to target mobile devices. These have the ability to evade detection and mitigation techniques that have already been developed. The traditional security systems like intrusion detection/prevention systems and Anti-Virus (AV) software rely on signature-based methods and thus are not able to identify new generation malware. Thus, there is a need to design techniques for better malware identification and classification. Furthermore, in a real-world scenario, the number of samples varies substantially among various malware families. Thus, it is important to build malware classification models which can take care of imbalanced classes. Additionally, there is a lack of adequate research on analyzing the threat or risk posed by Android apps. The main aim of this research is to address these problems and provide effective solutions.

Machine Learning (ML) techniques have been used to identify malware based on attributes mined using static and dynamic malware analysis. Through experiments, it is observed that both types of malware analysis have their pros and cons. The unknown malware use advanced obfuscation techniques to hide its presence, and it can detect the sandbox environment in which it is running. Thus, the single approach either static or dynamic is unable to identify and classify unknown malware. An integrated approach (an amalgamation of static and dynamic attributes) has been proposed in this work which can effectively analyze, detect and classify the malware. The two datasets i.e. Android malware detection and family classification are created using a comprehensive set of attributes acquired after performing static and dynamic analysis of malware. These datasets have been made public on GitHub and kaggle in order to assist anti-malware tool developers and researchers in developing new methodologies and tools for identifying and classifying malware. Six ML classifiers are used

to identify and classify Android malware using the attributes mined from static and dynamic malware analysis. The results demonstrate that the integrated approach improves the detection and classification accuracy of malware when compared with the approaches considering static or dynamic attributes alone.

The main problem with the existing malware detection systems is that they have a high False Negative (FN) and False Positive (FP) rate. An approach named as MalDetect has been proposed for enhancing the detection results of malware. The approach fuses the base classifiers on the basis of proposed ranking schemes defined on their error rate. These schemes are then used to generate a variety of combinations, with the best one being chosen to construct the final model. The proposed approach is tested on two datasets i.e. Drebin (benchmark) and AndroMD (self-created). The findings suggest that the proposed approach is more effective than conventional base classifiers and ensemble learning techniques.

In a real-world scenario, the number of samples varies substantially among various malware families which results in poor classification. For addressing this issue, a cost-sensitive learning (CSForest) approach has been proposed. The results of the proposed approach are compared with CSTree, Random Forest (RF) and C4.5 to identify its effectiveness in categorizing malicious app families. The findings suggest that the proposed approach is effective in determining the families of malicious apps.

Industries providing anti-malware solutions compute the risk associated with a piece of malware using the approaches involving human intervention along with a large number of resources. With the increase in the volume of malware, it is impossible to allocate a significant number of resources for analyzing the threat or risk posed by an Android app. To address this issue, a rule-based model has been designed. The proposed model assigns the risk levels (No, Low, Medium and High) to Android app features. The static features (permissions and Application Programming Interface (API) calls) in the data are examined statistically to come up with a hypothesis for identifying their risk factor. In order to test the hypothesis, Analysis of Variance (ANOVA) method is used. The results indicate that the mean values of different risk factors differ significantly. Afterward, a weight is assigned to the features under each category to compute the threat score of a particular app. This threat score can help the user to understand how risky it is to install an app on a mobile device. Moreover, the

computed threat score can assist in providing early warnings about a malicious app so that instant attention could be paid to with respect to assigning resources for deeper investigation.

The present research provides evidence based knowledge about emerging Android malware and is capable of generating actionable information in the form of threat intelligence.

LIST OF ACRONYMS & ABBREVIATIONS

AASandbox	Android Application Sandbox
AE	Average Error
ANOVA	Analysis of Variance
AOT	Ahead-Of-Time
Apps	Applications
API	Application Programming Interface
.apk	Android Package
ARM	Advanced RISC Machine
ART	Android Runtime
AUC	Area Under Curve
AVD	Android Virtual Device
AV	Antivirus
BERT	Bidirectional Representations for Transformers
CART	Classification and Regression Tree
CCR	Classification Cost Reduction
CED	Class Error Differential
CS	Cost-Sensitive
C2DM	Cloud to Device Messaging
DL	Deep Learning
DT	Decision Tree
DVM	Dalvik Virtual Machine
FAMD	Fast Android Malware Detector
FN	False Negative
FNR	False Negative Rate
FPGA	Field Programmable Gate Arrays
FP	False Positive
FPR	False Positive Rate
FT	Functional Tree
GCM	Goggle Cloud Messaging

HAL	Hardware Abstraction Layer
ICC	Inter-Component Call
ICFS	Iterative Classifier Fusion System
IDC	International Data Corporation
IDS	Intrusion Detection System
ID3	Iterative Dichotomiser3
IFS	Iterative Feature Selection
IG	Information Gain
IPS	Intrusion Prevention System
JSON	Java Script Object Notation
K-NN	K-Nearest Neighbors
LDA	Linear Discriminant Analysis
LR	Logistic Regression
LSTM	Long Short Term Memory
MBSS	Model Based Semi Supervised
MCC	Matthews Correlation Coefficient
MD5	Message-Digest
MIPS	Microprocessor without Interlocked Pipelined Stages
ML	Machine Learning
MLP	Multilayer Perceptron
NB	Naive Bayes
NN	Neural Network
OS	Operating Systems
PART	Partial Decision Tree
PCA	Principal Component Analysis
PPV	Positive Predicted Value
PSO	Particle Swarm Optimization
RAACED	Ranked Aggregate Average and Class Error Differential
RAPCE	Ranked Aggregate of Per Class Error
RF	Random Forest
Sens	Sensitivity
SigPID	Significant Permission Identification

SL	Simple Logistic
SMO	Sequential Minimal Optimization
SMOTE	Synthetic Minority Oversampling Technique
SSL	Secure Socket Library
SVM	Support Vector Machine
TF-IDF	Time Frequency-Inverse Document Frequency
TN	True Negative
TP	True Positive
WEKA	Waikato Environment for Knowledge Analysis
W-FM	Weighted F-measure

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Android is found to be the most well-known platform all over the world [1]. Andy Rubin is known as the “Father of Android” for his work on the “Camera” project, which the Symbian OS couldn't handle. In August 2005, Andy Rubin turned over ownership of Android to Google. Android OS relies on the Linux kernel. It is an open-source software and built for touchscreen devices such as tablets and smartphones etc. The first Android device named HTC Dream was released in 2008 [2]. According to the International Data Corporation (IDC) report, the worldwide smartphone market has achieved a shipment of 1.39 billion in 2019 [3].

1.1 BACKGROUND OF ANDROID

1.1.1 Architecture

The Android OS architecture consists of five main parts i.e. Linux kernel, Hardware Abstraction Layer (HAL), Libraries (including Android runtime), Application framework, Application [4] as demonstrated in Figure 1.1.

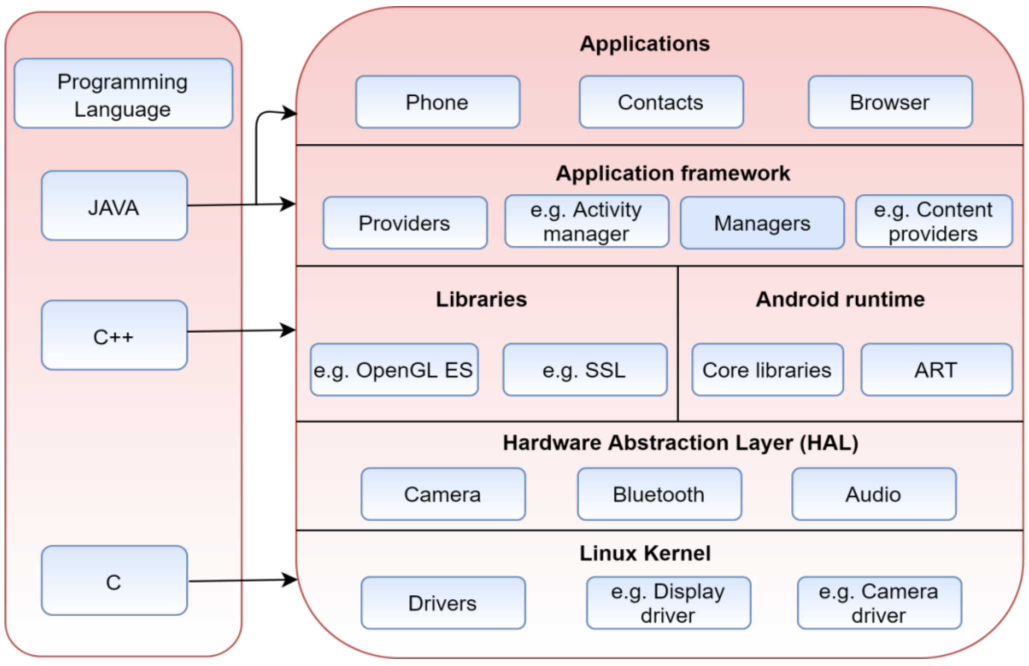


Figure 1.1: Architecture of Android [4]

A detailed explanation is given as follows:

- **Linux kernel**- During runtime, it manages all available drivers such as memory drivers, display and camera drivers, audio and Bluetooth drivers.
- **HAL-** It offers a standard interface that reveals hardware functionalities of the device to the higher-level app framework. The HAL comprises several library modules such as Bluetooth, audio, camera etc. When API makes a call to use hardware devices, the Android device loads the module for that hardware component.
- **Libraries-** Android system services and components are constructed from the native code that needs native libraries. The platform Android offers an application framework to reveal the capabilities of some of these libraries to apps. For example, Secure Socket Library (SSL) for Internet security and OpenGL ES used to generate 2D and 3D computer graphics.
- **Android Runtime-** The java classes are first converted to DEX Bytecode and then with the help of Dalvik Virtual Machine (DVM) and Android Runtime (ART), DEX Bytecode is converted to the machine level language. DVM has some limitations like low garbage collection etc. So to overcome these limitations, DVM was replaced by ART. From the Android 5.0 version onwards, ART was used as runtime. Some of the essential features of ART are as follows:
  - Optimized garbage collection
  - Uses Ahead-of-time (AOT) method and compiles the complete code at the time of installation.
- **Application framework-** It offers a variety of Android.* packages that serve as high-level building blocks for apps. On the mobile device, majority of the components in this layer are implemented as apps that operate in the background.
- **Applications-** It is the topmost layer of Android architecture. It consists of those apps which are already built into the device itself. Examples of the apps are contacts, camera, browser etc.

1.1.2 Components of an Android Application

Activities, Content Providers, Broadcast Receiver and Services are the four main components of an Android app [4]. Figure 1.2 shows the components of an Android app. These components are bound by the manifest file that holds the detail of each component. The description of the components is given as follows:

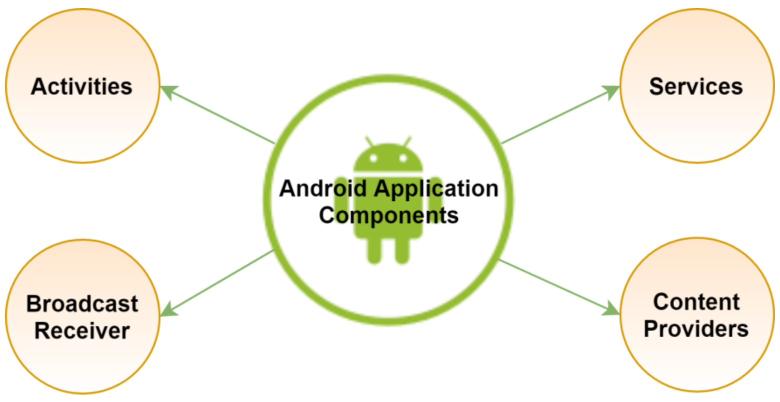


Figure 1.2: Components of Android apps [4]

- **Activities-** The user interface is directly linked by an activity. It is the visual representation of an Android app.
- **Services-** This component performs background tasks, triggering the notification, update your activities and data source. When the app is not active, it still performs some tasks.
- **Content Providers-** It can assist an app in managing access to data stored by it and by other users, as well as providing a way for data to be shared with other apps. They encapsulate data and give mechanisms for defining data protection.
- **Broadcast Receiver-** It is also called intent listeners. It allows your app to listen to the intents that meet the criteria prescribed by us. It carries out an action in reply to a message from other apps.

1.1.3 Features of Android

Lots of users use Android mobile devices because it offers an open platform facility and provides various functionalities [4]. Some of the features of Android are listed as follows:

- Storage
- Messaging: MMS, GCM (Goggle Cloud Messaging), SMS, C2DM (Cloud to Device Messaging)
- Multitouch

- Screen capture
- Connectivity: Bluetooth, GSM/EDGE, WIFI, GPS, LTE etc.
- Video calling
- Multilanguage Support

1.1.4 Versions of Android

The Android development was started in 2003, and was purchased in 2005 by Google [5]. The first version of Android 1.0 was released in October 2008 [3]. Table 1.1 shows the versions of Android with their names, API, release date.

Table 1.1: Android versions with their names, API and release date [6]

Version of Android	**Names**	**API**	**Release Date**
1.0	No name	1	23-09-2008
1.1		2	09-02-2009
1.5	Cupcake	3	27-04-2009
1.6	Donut	4	15-09-2009
2.0 – 2.1	Éclair	5 – 7	26-10-2009
2.2 – 2.2.3	Froyo	8	20-05-2010
2.3 – 2.3.7	Gingerbread	9 – 10	06-12-2010
3.0 – 3.2.6	Honeycomb	11 – 13	22-02-2011
4.0 – 4.0.4	Ice Cream Sandwich	14 – 15	18-10-2011
4.1 – 4.3.1	Jelly Bean	16 – 18	09-07-2012
4.4 – 4.4.4	KitKat	19 – 20	31-10-2013
5.0 – 5.1.1	Lollipop	21 – 22	12-11-2014
6.0 – 6.0.1	Marshmallow	23	05-10-2015
7.0 – 7.1.2	Nougat	24-25	22-08-2016
8.0 – 8.1	Oreo	26 – 27	21-08-2017
9.0	Pie	28	06-08-2018
10.0	Android 10	29	03-09-2019
11.0	Android 11		TBD

1.1.5 Android Applications

The structure of the Android app is described in this section. Android apps are packed in the Android Package (.apk) file format. A program in Android is compiled and its entire components are packed in a single file. This file is known as .apk which is saved into a zip file format. To open .apk file, it first needs to be unzipped or decompile [7]. The structure of the Android app is shown in Figure 1.3. This zipped file consists of the following folders and files as discussed below:

- **Assets-** It contains the media file which could be obtained by the Assets Manager.
- **Lib-** It comprises compiled code with respect to the software layer of a processor.
  - **Armeabi-** It contains the compiled code for processors i.e. Advanced RISC Machine (ARM)
  - **Armeabi-v7a-** It consists of the compiled code for processors i.e. armv7 and above.
  - **X86-** It contains compiled code for processors i.e. X86.
  - **MIPS-** It contains the compiled code for processors i.e. Microprocessor without Interlocked Pipelined Stages (MIPS).
- **Res-** It contains resources like icons, sting files, images, fonts etc.
- **META-INF-** This directory consists of the following such files:
  - **CERT.RSA-** It holds the app certificate.
  - **CERT.SF-** It holds the list of resources and the app security certificate.
  - **MANIFEST.MF-** It contains significant information about the app.
- **Resources.arsc-** It consists of pre-compiled resources.
- **Android Manifest.xml-** It characterizes the app functionalities and also contains meta-information about the app.
- **Classes.dex-** It holds the class name, package name, API calls or methods and class path.

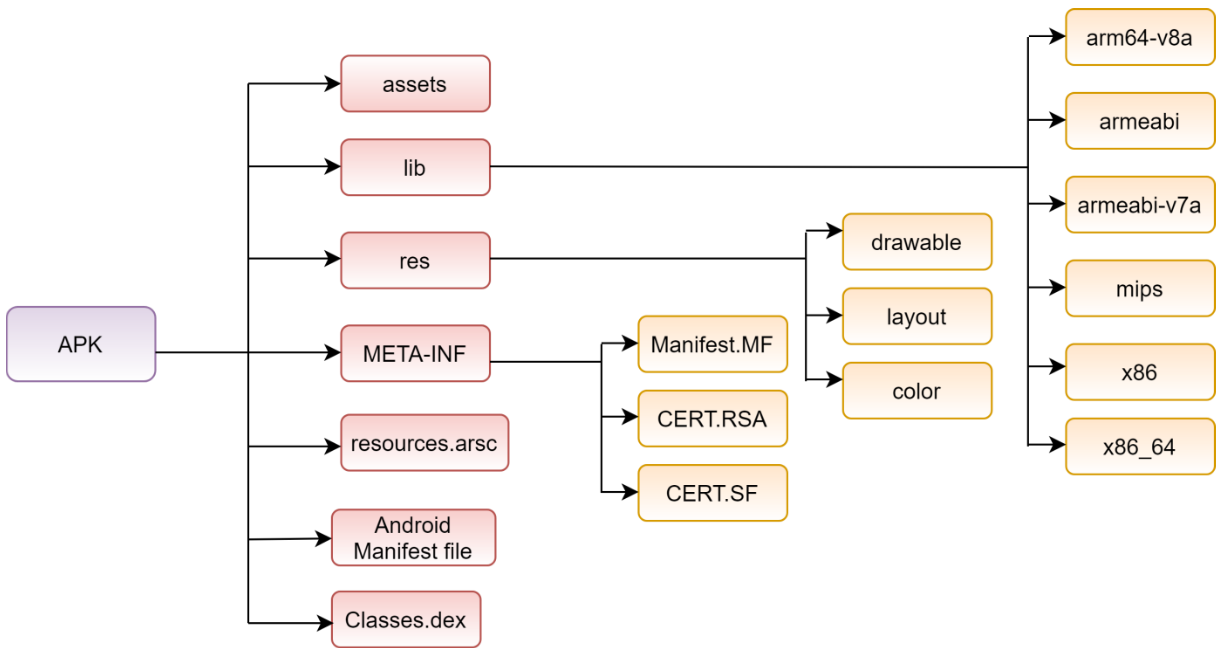


Figure 1.3: Architecture of apk [7]

1.2 EVOLUTION OF ANDROID MALWARE

The term malware refers to a distinct form of intrusive software or app. Some of the examples of malware are worms, ransomware, spyware, adware and trojan horses etc. It is also known as malicious software or malicious app. It is created to perform malicious activities such as stealing personal data, gaining access to the devices, making changes in the devices etc. A wide variety of malware has been identified till date. After 2009, there is a hazardous growth in mobile malware. This is because new technologies have opened new opportunities for beneficial exploitations [8, 9]. The first Trojans i.e. FakePlayer and DroidSMS were detected in the year 2010 [10]. Figure 1.4 shows the development of Android malware from 2010 to 2021. Table 1.2 shows the timeline of Android malware with its short description.

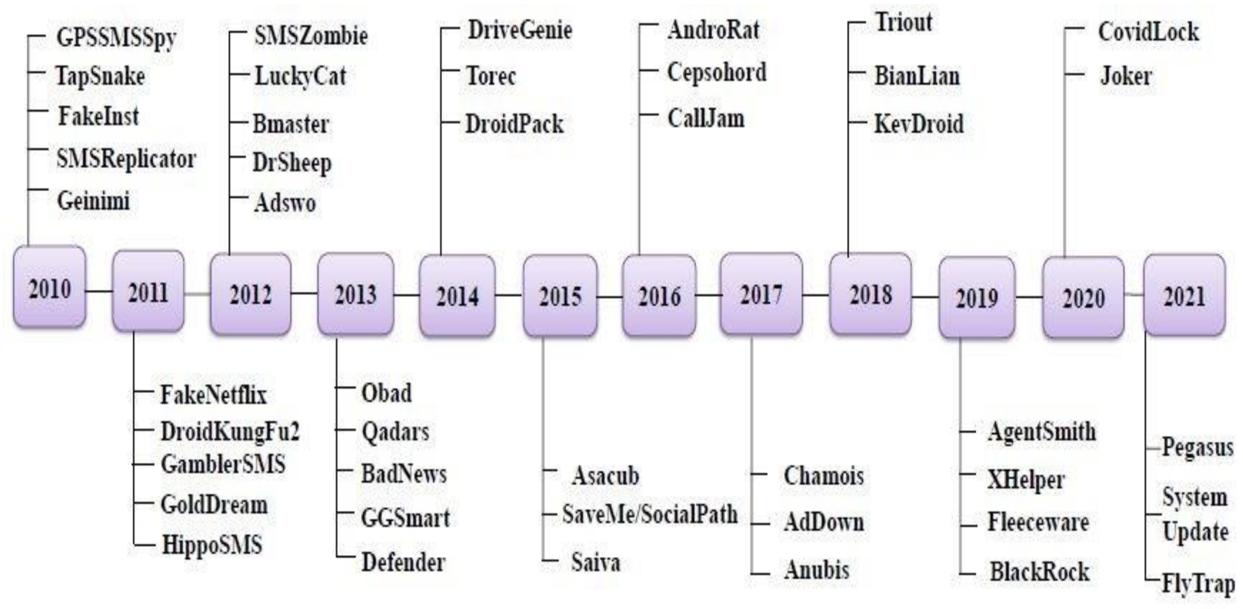


Figure 1.4: Android malware evolution

Table 1.2: Android malware with their short description

Released Year	Name of Android Malware	Short Description
2010	FakeInst	It is a kind of trojan that tries to send the SMS message to a preset number.
	SMS Replicator	It acts as a snoop that secretly sends the message to any phone number chosen by the user.
	Geinimi	It is a trojan that unlocks the backdoor and forwards the information to a specific URL from the mobile phone.
	GPSSMSSpy	It pays attention to SMS premised commands to record and forward the user's current location.
	TapSnake	It is a type of malware that sends the victim's location to a web service.
2011	FakeNetflix	This malware is designed to target Netflix users. It captured the personal data and is posted to a server.
	DroidKungFu2	Once installed on the mobile phone, it reads the entire data and writes it in a file and then posted it to the server.
	GoldDream	It is a trojan that examines the incoming and outgoing calls and gathers the information related to all messages and saved them in zjphonecall.txt file name.
	GamblerSMS	It acts as spyware and examines every call (incoming and outgoing) and records all calls and SMS messages.
	HippoSMS	It is a trojan that removes the incoming messages and forwards the SMS

		to premium numbers.
2012	DrSheep	It hijacks the accounts of social networking sites like Instagram, Facebook, LinkedIn via wifi connection.
	Bmaster	Once installed on the device, it collects sensitive information from the devices like DeviceID, GPS data, IMEI number etc.
	Adswo	It displays unwanted advertisements as notifications and monitors privacy-invasive.
	SMSZombie	This malware infects the devices and sends the SMS message to a preset number.
	LuckyCat	It can collect data on a mobile device and download and upload the files as guided by the C&C server.
2013	BadNews	The main aim of this malware is to send bogus messages to a server and permits the user to install apps.
	Qadars	It is a banking malware that prohibits the user from accessing their bank accounts. It is also known as SPY-ABN.
	Obad	It is a kind of multifunctional trojan that is liable for forwarding SMS to the premium rate numbers.
	GGSmart	Its main function is to gather significant information and sends it to a remote server, send the SMS messages to a premium rate number. It also contains access to read, delete and write privileges on the device.
	Defender	It is a kind of ransomware. Once it gets installed on the device, it infects the system and displays the messages demanding a fee to be paid to regain access.
2014	Torec	It is a malware that uses .onion domain as its command and control (C&C) server.
	DroidPack	It is a trojan that collects the login credentials information from the user device.
	DriveGenie	This malware is automatically downloaded on the mobile device without the user allowance. It gathers and sends the information of the victims to a server.
2015	Saiva	It is a trojan that has abilities to terminate processes, delete files and capture the input of the keyboard from the victim device.
	SaveMe/Social Path	This malware steals information such as SMS message, call logs etc and then upload the information to a server.
	Asacub	It is a banking trojan. It is developed to steal money from banking apps. It steals all SMS messages from the user's device and uploads them to the server.
	AndroRat	It can take a photo, steal browsing history.

2016	Cepsohord	It is a trojan horse that deletes information files and downloads other harmful malware like ransomware.
	CallJam	This malware comprises a premium dialer to make fraud calls and can display ads.
2017	Chamois	It downloads other apps on the victim's device that send fraud SMS.
	Anubis	This malware steals photos, contacts, SMS messages etc. It can also take a screenshot and record audio.
	AdDown	It is adware that stealthily installs apps in the device without the user's permission. Its main aim is to collect data, display ads etc.
2018	BianLian	It is a trojan that can send, read and receive messages. It also records the screen, locks the screen.
	Triout	This malware can record the calls, steal call logs and messages etc. It uploads the recorded call to the server.
	KevDroid	It is a RAT (Remote Access Trojan). It steals data like emails, calls logs and SMS etc. It also gathers information about the location of the device.
2019	Agent Smith	It hacks the apps and enforced them to show more ads to earn profit.
	XHelper	This malware offers a backdoor to the assailant. The assailants then steal, install other apps on the victim's device.
	Fleeceware	This malware comes with an unseen, unreasonable subscription fee.
	BlackRock	It can steal passwords and important information from the device.
2020	CovidLock	It is a ransomware that infect the users device by ensuring them to give information about COVID-19
	Joker	It steals money from victims by inadvertently enrolling them in premium memberships.
2021	FlyTrap	It is a trojan that hacks facebook account of the users and collects information from the users device
	System Update	Once it is installed on the mobile device, it steals videos, photos and location of the users.
	Pegasus	It is a spyware, once installed, it accesses all data including emails, whatsapp conversations and SMS.

The increasing demand for smartphones attracted many organizations to build various apps such as gaming, education, business, entertainment, banking, lifestyles, etc. The increasing use of Android apps has lured attackers to build malicious apps that pose several threats such as financial loss, information leakage etc.

1.3 MALWARE DETECTION METHODS

Malware detection methods can be broadly classified into two categories. These are as follows:

1.3.1 Signature based Method

It is the most popular technique used by all antiviruses. This method compares the app's signature to an already existing signature in the database. The limitation of this approach is that it cannot detect new (unknown) malware (also known as zero-day malware). To overwhelm the drawback of this approach, researchers start making use of ML approaches using static and dynamic malware analysis [11-13].

1.3.2 Machine Learning based Method

ML techniques have been used to detect malware based on attributes mined using static and dynamic malware analysis. This method makes use of features mined after performing static and dynamic malware analysis. The mined attributes are used to train the model for making predictions [12, 14, 15].

1.4 ANDROID MALWARE ANALYSIS

It is the process of investigating the apps to identify their functionalities and attacking techniques being used by malware creators [16]. It is carried out by using two approaches i.e. static malware analysis [17-20] and dynamic malware analysis [21-23].

1.4.1 Static Malware Analysis

It analyses the sample of malware without executing or running the code. It uses decompiling methods to decompile the app package and extract the features for the detection of malware. However, this approach has its constraints such as it is unable to examine the obfuscation code and morphed malware but it is faster in identifying malware [24, 25]. To overwhelm the constraints of the static approach, a dynamic approach is used. It can keep tracking the behavior or characteristic of the app and accurately identify the unknown malware [26-28].

1.4.2 Dynamic Malware Analysis

It examines the characteristics or behavior of an app while it is running in the sandbox (virtual environment). It is more effective as it keeps on tracking the behavior of the apps at the time of execution. The major constraints of this method are that it takes a long time because each app must run for at least one minute in the sandbox. Furthermore, if a malware is able to detect itself being executed in the virtual environment, it may become dead so that its behavior could not be monitored. Moreover, it could not explore all execution paths [29, 30].

From the above approaches, it is found that the static analysis is not that effective in detecting malicious contents. It omits code obfuscation and morphed malware but it is faster in identifying malware. While considering the dynamic approach, it is more effective than the static approach as it keeps the track of the behavior of apps at the time of execution. But still, some apps remain undetectable at the execution time. From here, it is concluded that a single approach is not capable of detecting malware more precisely. Thus to enhance the accuracy, the hybrid approach is being used which is the integration of both approaches. Table 1.3 illustrates the comparison between all three approaches.

Table 1.3: Comparison of malware analysis approaches

Features\Analysis Approach	**Static**	**Dynamic**	**Hybrid**
Time required	Low	High	High
Resource consumption	Less	More	More
Effectiveness	Less effective	More effective than static	More effective than both approaches
Merits	Low cost, takes less time, extracts features easily	Capable of identifying unknown malware	Give more accurate results
Demerits	Unable to examine obfuscated code	It takes more time and resources	High cost

1.5 TECHNIQUES AND TOOLS FOR ANALYSING ANDROID MALWARE

There exist a variety of techniques and tools for the analysis of malware. Gandotra *et al.* [26] have conducted a review based on several techniques and tools used for malware analysis. They incorporated a comparison of several tools and techniques for analysing malware. This section explains the numerous techniques and tools which are utilized for the execution of static and dynamic malware analysis as demonstrated in Figure 1.5.

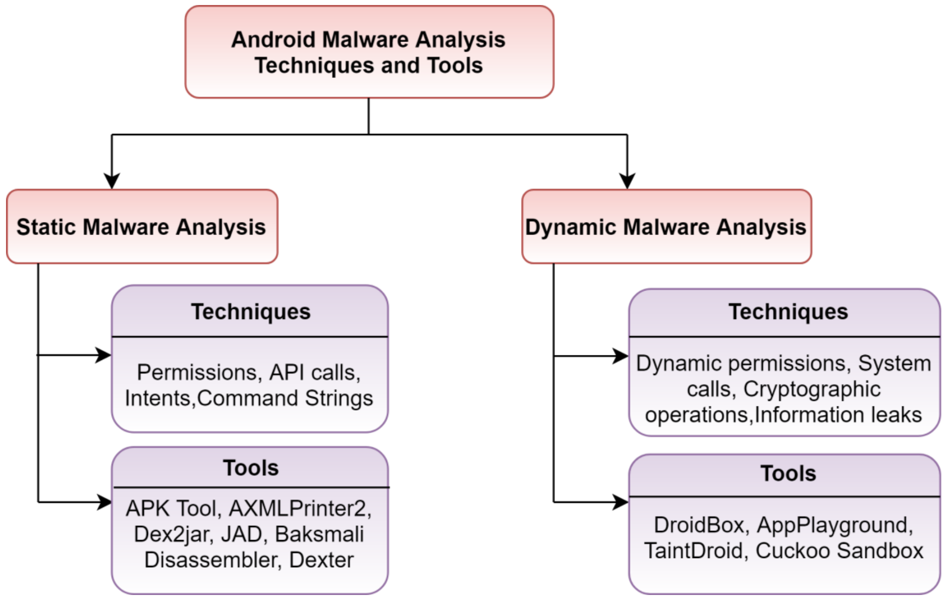

Android Malware Analysis Techniques and Tools
Static Malware Analysis
Dynamic Malware Analysis
Techniques
Permissions, API calls, Intents,Command Strings
Tools
APK Tool, AXMLPrinter2, Dex2jar, JAD, Baksmali Disassembler, Dexter
Techniques
Dynamic permissions, System calls, Cryptographic operations,Information leaks
Tools
DroidBox, AppPlayground, TaintDroid, Cuckoo Sandbox

Figure 1.5: Techniques and tools for analysing Android malware

1.5.1 Techniques and Tools for Static Malware Analysis

For static malware analysis, various techniques and tools are used. The most common are discussed as follows.

- **Permissions-** The security system in Android primarily relies on permissions. To access the user’s personal information (such as SMS and contacts) and specific features (such as Internet and camera) the mobile apps must request permissions. These are used to restrict or allow an app access to confined resources and API's. According to the features, the system might provoke the customer to accept the request or sometimes it might provide permission automatically. These permissions are present in the AndroidManifest.xml file**.** The Android OS defines various permissions which are stated as static string members in the manifest file [31-34]. Table 1.4 demonstrates the few examples of permissions with their description

Table 1.4: Few examples of permissions with their description

Permission Name	Description
READ_CONTACTS	It permits an app to read the contact of the user's data.
READ_SMS	It permits an app to read the SMS of the user's data.
MODIFY_PHONE_STATE	It permits an app to modify the data of the phone.
WRITE_SMS	It permits an app to write SMS messages.

These permissions are further categorized into four sub-categories that are described as follows:

- Normal permissions
- Dangerous permissions
- Signature permissions
- Special permissions

Normal permissions- A low hazard permission, which permits apps to access API calls (e.g. ACCESS_WIFI_STATE, CHANGE_WIFI_STATE) causing no harm to the Android users. The Android system directly assigns these kinds of permissions without any involvement of users.

Dangerous permissions- A high hazard permission that permits apps to access injurious API calls (e.g. CALL_LOG, READ_CONTACTS) causing harm such as stealing confidential information. These permissions are clearly displayed to the user before an app is installed. The user must select whether he/she accept or decline the permissions.

Signature permissions-These permissions are provided by the system itself while installing the app. The system provides it only when the app is signed by the same certificate as the app defining the permission e.g. BIND_INCALL_SERVICE.

Special permissions- These permissions do not behave like hazardous and normal permissions e.g. SYSTEM_ALERT_WINDOW, WRITE_SETTINGS etc.

- **API calls-** To interact with the devices, API calls are required. These include packages, methods and classes to support the developer to create apps. The java programming language is used to build Android apps, and the java compiler turns the source code into bytecode. It uses DVM after decompiling java bytecode to get the information of methods, packages and classes [34, 35]. For e.g. Telephony manager of OS to fetch user ID and a subscriber ID. Some of the API calls are there in Android apps are: getBinder, KeySpec, getBinder, Ljava.net.URLDecoder, android.os.Binder, Ljava.lang.Class.getMethods, ServiceConnection, onserviceConnected, Ljavax.crypto.spec.SecretKeySpec.
- **Intents-** Intents are present in Manifest.xml. It is an abstract description of an action to be carried out [36, 37]. It infers the app's intentions such as picking a contact. Some of the examples of intents are SET_WALLPAPER, SCREEN_OFF, ACTION_SHUTDOWN, CALL_BUTTON, PACKAGE_CHANGED, NEW_OUTGOING_CALL.
- **Command strings-** It is considered one of the most significant attributes for the recognition of malware [38]. These command strings are present in lib, res, assets folder. Some of the examples of command strings are /system/app, chown, mount, remount, /system/bin and chmod.

Various tools are used for static malware analysis to decompile the app package and to mine the attributes from the apk. Tools that are used for static malware analysis are shown in Table 1.5.

Table 1.5: Static malware analysis tools

Static Tools	Short Description
Apk tool [39]	It permits decoding an app to smali code and also it is a reverse engineering tool.
Dex2jar[40, 41]	It converts the .dex file into the .jar file.
String [26]	It finds an executable for the string.
Androguard [42, 43]	It is used for disassembling apps.
JAD [44]	It decompiles the .class file into the .jar file.
DED [45]	It is used for decompiling an app.
AXMLPrinter2 [46]	It is the library for decompiling manifest files.
Baksmali [47, 48]	Dex to smali translator

1.5.2 Techniques and Tools for Dynamic Malware Analysis

For dynamic malware analysis, various techniques and tools are used. The most common are discussed as follows.

- **Cryptographic operation-** These operations are accepted by malware to encrypt root exploits, target premium sms numbers, malicious payload etc. The features which are used to differentiate many cryptographic behaviors are represented as <action> <algorithm>. The <action> represents operations like decryption, generation and encryption whereas <algorithm> represents several cryptographic algorithms [49, 50].
- **Dynamic permission-** It is one of the significant dynamic attributes to examine the behavior of apps. These permissions are executed at the runtime environment [49, 50].
- **Information leaks-** Personal and confidential data has gained lots of attention [49, 50]. Android malware performs malicious activities such as stealing SMS content, important information associated with banking and social networking, contact information etc. The collected information is used for keeping track of users and making profits. These are defined as <source>_<sink>. The <source> represents operations that gained confidential data and the <sink> represents operations that leaked confidential data.

- **System calls-** It is a useful feature for mobile device intrusion detection. Android apps use the kernel's services via system calls [49-51]. The kernel provides apps with useful services such as operations-related processes, device security and power management etc. To alter the execution of other apps, this virus commonly uses sigprocmask, ptrace and getuid.

Various tools are used for dynamic malware analysis to examine the characteristics or behavior of an app while it is running in the sandbox (virtual environment). Tools that are used for dynamic malware analysis are shown in Table 1.6.

Table 1.6: Dynamic malware analysis tools

Dynamic Tools	Short Description
Cuckoo Sandbox [52]	It is a tool that monitors dangerous files on Android, window and Linux.
DroidBox [53]	It permits the execution of apps and gives information associated with the behavior of the app.
AppPlayground [54]	The dynamic analysis of the app is attempted to be automated with this tool.
TaintDroid [55]	It makes use of taint analysis to trace the data during the program execution.

The rapid increase in Android malware needs effective approaches or techniques to better detect malicious apps. Zhou and Jiang [56] demonstrated that Android malware is increasing continuously and the existing solutions are becoming ineffective. The traditional security analysis is based on the analysis of security incidents, but it fails to give protection at a proper time. As a result, users remain unsafe for a longer duration. For these problems, ML is the best solution. It automatically examines the data, assists in the early identification of threats and offers decisions on time. Most of the work based on mobile security using ML provides better results in detecting malware.

1.6 MACHINE LEARNING

"ML provides computers the capability to learn without being explicitly programmed [57]". It becomes a common field of research in recent years and has been applied to a variety of fields. Some of the examples of these fields are medical data processing [58], classification of false news [59] and voice analysis or recognition [60]. Researchers start making use of ML techniques using static and dynamic malware analysis. Malware analysis is performed in order to extract attributes and then these attributes are used to train the model for predictions.

1.6.1 Types of Machine Learning

ML is classified into three types: unsupervised learning, supervised learning and reinforcement learning. Figure 1.6 shows the various categories of ML.

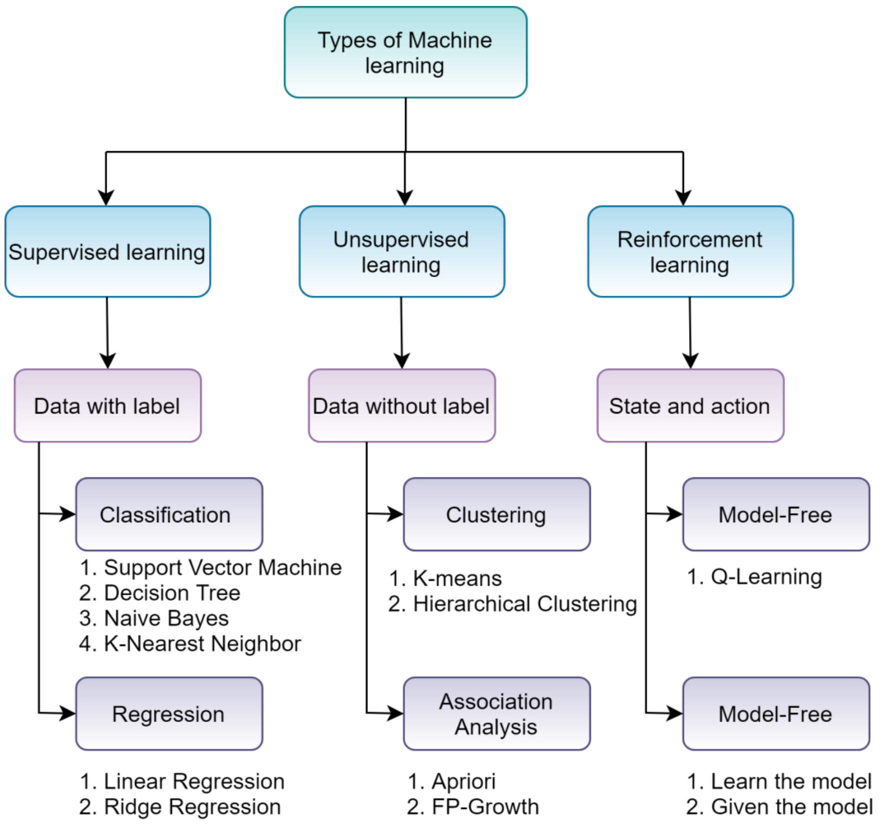


Figure 1.6: Various categories of ML

Supervised learning- It is also called "learn with examples". Prior knowledge of the predicted attribute (i.e. class attribute) is required for this learning. This learning is divided into two categories i.e. classification and regression [57].

- **Classification-** It is the process of distinguishing the instances into different groups. The output attribute consists of categorical values.
  - **Binary classification-** It classifies the instances of a given set into two groups for e.g. “malware” or “benign”.
  - **Multiclass classification-** It classifies the instances of a given set into two or more groups for e.g. “family classification of malware”.
- **Regression-** The output attribute consists of real values for e.g. “weights” and “heights”.

Some of the examples of supervised learning are RF, Neural Network (NN), Decision Tree (DT) and Support Vector Machine (SVM) etc [61, 62].

Unsupervised learning- It is also called “learn without examples”. No prior knowledge of the predicted attribute is required for this learning. This learning is divided into two categories i.e. Association and clustering [57].

- **Association-** It is a market-based analysis problem. It frames the association rule $(A \rightarrow B)$ between a set of items. For example, if someone buys item A, what is the probability that B also goes with it. One of the examples of association rule mining is the Apriori algorithm.
- **Clustering-** It is defined as the process of arranging data points into groups whose members are identical in some way. Some of the examples of clustering are hierarchical, K-means, partitioning clustering etc.

Reinforcement learning- It is a type of dynamic programming that trains the model using a system of punishment and rewards. It is used to define the best decision which permits the agent to solve a problem while maximizing a reward. Some of the examples of reinforcement learning are Q-learning and R-learning [57].

1.7 ANDROID MALWARE DATASETS

This section focuses on the importance of the datasets as well as the datasets available for the detection of malware. In classification or prediction, the dataset plays an important role while conducting any experiment. The dataset contains data that allows the model to have a high level of understanding. Only relevant information in the input dataset can be used to provide better training to the model, resulting in an optimal output. As a result, a dataset is crucial in the building and testing of the proposed methods.

For the identification and classification of malware, many researchers have proposed various approaches/methods and assessed these on data. According to the existing research, there are not enough datasets related to Android malware. As a result, it is necessary to create a dataset and make it publicly accessible so that researchers may compare their new techniques to previous ones.

Researchers collect the Android samples from the repositories that provide benign and malicious samples including Contagio Mini-Dump, Google Play Store, virusshare, EMBER, Apkpure and Apkmirror as shown in Table 1.7.

- **Contagio Mini-Dump-** It is the repository from where the users can upload or download the samples. It comprises 28,760 samples out of which 16,800 are benign samples and 11,960 are malicious ones.
- **Virusshare-** It is one of the most well-known websites that comprise 3,48,25,574 samples of both windows and Android malware.
- **Google Play Store** is one of the most renowned official app stores comprising 3.48 million benign apps.
- **EMBER-** It comprises of 1million records and carries malware and benign apps.
- **Apkpure-** It is a website from where the users can download benign samples.
- **Apkmirror-** It is a website from where the users can download benign samples.

Table 1.7: Sources of Android samples

Database	**Published year**	**Total samples**	**Available at**
Contagio Mini-Dump	2011	28,760	http://contagiominidump.blogspot.com/
Virusshare	2013	3,48,25,574	https://virusshare.com/
Google Play store	2012	3.48 million	https://play.google.com/store
Microsoft malware classification challenge	2015	20,000	https://arxiv.org/abs/1802.10135
EMBER	2018	1.1million	https://arxiv.org/abs/1804.04637
Apkpure	2014	---	https://apkpure.com/
Apkmirror	---	---	https://www.apkmirror.com/

--- Not specified

There are only few datasets that are publically available for comparing the techniques or methods with earlier ones. These are MalGenome, Drebin, CICAndMal2017 and AAGM as shown in Table 1.8.

- **MalGenome-** It comprises 3,799 observations out of which 2,539 are benign observations and 1,260 are malicious observations. Further, these malicious observations are classified into 49 malicious families. It has 215 attributes in all, grouped into four categories: permissions, command strings, intents and API calls.
- **Drebin-** It comprises 15,036 numbers of observations out of which 9,476 are benign observations and 5,560 are malicious observations. Further, these malicious observations are classified into 179 malicious families. It has 215 attributes in all, grouped into four categories: permissions, command strings, intents and API calls.
- **CICAndMal2017-** It comprises a total of 10,854 numbers of observations out of which 6,500 are benign and 4,354 are malicious observations. Further, these malicious observations are classified into 42 malicious families
- **AAGM-** It comprises a total of 1,900 observations out of which 1,500 are benign and 400 malicious observations. Further, these malicious observations are classified into 12 malicious families.

Table 1.8: Publically available Android benchmark datasets

Database	Published year	Malware	Benign	Malware families	Attributes	Total instances	Available at
MalGenome	2012	1,260	2,539	49	215	3,799	https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_1/5854590/1
Drebin	2014	5,560	9,476	179	215	15,036	https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653
CICAndMal2017	2017	4,354	6,500	42	---	10,854	https://www.unb.ca/cic/datasets/index.html
AAGM	2017	400	1,500	12	---	1,900	https://www.unb.ca/cic/datasets/android-adware.html

--- Not specified

1.8 PERFORMANCE PARAMETERS

The proposed approaches are examined based on different assessment parameters. These are discussed as follows:

Confusion matrix- It is a table that is frequently used to explain the performance of the classifier or classification model on the test data. Table 1.9 shows the confusion matrix for the binary classification problem.

Table 1.9: Confusion matrix for binary classification problem

		Predicted	
		Malware	**Benign**
	Malware	TP (True positive)	FN (False Negative)
Actual	**Benign**	FP (False positive)	TN (True Negative)

TP- Number of examples correctly predicted as the malware class.

TN- Number of examples correctly predicted as the benign class.

FP- Number of examples incorrectly predicted as malware class.

FN- Number of examples incorrectly predicted as benign class.

- **Accuracy (%) -** It is the ratio of correctly classified apps to the total number of apps. It is calculated as shown in Equation 1.1.

$$Accuracy\ (\%) = \frac{TP+TN}{TP+FP+FN+TN} \times 100 \tag{1.1}$$

- **Sensitivity (Sens) -** It is also called TP rate or recall. It is the rate of correctly detected malicious apps to the number of malicious apps. It is calculated as given in Equation 1.2.

$$TPR = \frac{TP}{TP+FN} \tag{1.2}$$

- **Positive Predicted Value (PPV) -** Precision is another name PPV. It is the ratio of correctly detected malicious apps to the total number of apps that are detected as malicious apps. It is calculated as given in Equation 1.3.

$$PPV = \frac{TP}{TP+FP} \tag{1.3}$$

- **F-measure -** It is the harmonic mean of both recall and precision. It is computed as given in Equation 1.4.

$$F - measure = \frac{2\times(Recall\times precision)}{Recall+precision} \tag{1.4}$$

- **False Positive Rate (FPR)** – It is the rate of incorrectly predicted benign observations. It is computed as given in Equation 1.5.

$$FPR = \frac{FP}{TN+FP} \tag{1.5}$$

- **Matthews Correlation Coefficient (MCC) -** It calculates the quality of binary classification problems. The value of MCC lies in between – 1 to +1. If the value of MCC is +1 then it means perfect prediction and if -1 then it means inverse prediction. It is computed as given in Equation 1.6.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FN)(TP+FP)(TN+FP)(TN+FN)}} \tag{1.6}$$

1.9 MOTIVATION

In the modern era, smartphones are becoming more prevalent in our daily lives. A Lot of users relies on smartphones for a variety of purposes including banking, shopping, gaming, entertaining etc. There is a variety of Operating Systems in the market including iOS, Android, BlackBerry and Windows Phone etc. Android is the most popular among these. It has 85% of market share with more than 3.04 million apps [63]. Till December 2019, the population of mobile users was 1 billion but throughout the pandemic covid-19 the population of mobile users is upto 7 billion [64]. The exponential growth in mobile technologies has made users to use smart devices to take advantage of various services. The increase of Android apps plays a significant role in the development of the forthcoming economy and mobile Internet. The increasing use of Android apps has lured the attention of the attackers. Recently, various threats (such as system damage, information leakage, financial loss) have been arisen due to explosive growth in mobile technologies. The report of MacAfee shows that the growth of Android malware is increased by approximately 121 million in the year 2020 [65]. The increase in the number of Android malware has become complex in manually handling the malware samples. To elude this problem, there is a need to develop an automated and effective technique to better detect malware. The traditional approaches for detecting malware are based on the Signature-based method and fails in detecting new malware.

Earlier, the malware was written or designed for simple purposes. Therefore, it was simpler to identify. This type of malware is known as conventional malware. The advanced malware being created by attackers has the ability to get executed in kernel mode and is harder to identify. This type of malware is known as new generation malware. Table 1.10 demonstrates the comparison of conventional malware and new generation malware.

Table 1.10: Comparison of conventional malware and new generation malware

Parameter	**Traditional**	**New Generation**
Level of implementation	Simple Coded	Hard Coded
Type of attack	General	Targeted
Targeted devices	Computers	Many different devices
Challenge	Easy	Difficult
Use of hiding technique	None	Yes
Permanency	Temporal	Persistent
Spreading through	.exe extension	Uses different extensions

The upcoming malware (new malware) is sophisticated and complex in nature. As a result, the traditional methods are incapable to detect complicated and sophisticated malware quickly and accurately. Therefore, there is a need to design techniques for better identification and classification of malware.

1.10 RESEARCH OBJECTIVES

The research objectives are framed as follows:

1. To propose a model by taking into account an integrated set of static and dynamic attributes for detection and classification of unknown malware.
2. To propose an approach based on the fusion of ML algorithms using ranking schemes for improving detection of Android malware.
3. To design and develop an approach for imbalanced classification of malware.
4. To design a rule based model for identifying the risk level of Android app features.

1.11 ORGANIZATION OF THESIS

The research work comprises seven chapters. **Chapter 1** demonstrates the importance of Android OS, its architecture, different versions, its evolution and methods for examining Android malware. Different tools and techniques for analysing Android malware, details of Android malware datasets and all evaluation parameters used for validation are discussed in this chapter.

Chapter 2 discusses the detailed literature survey of the state-of-the-art methods used for Android malware detection, Classification and Threat measurement. This chapter also presents the inferences drawn from the literature review.

Chapter 3 presents an integrated approach that can effectively analyze, detect and classify the malware. The dataset is created using static and dynamic malware analysis for both binary (named as Dataset-1) and multiclass classification (named as Dataset-2) dataset and made it publically accessible on kaggle and GitHub. Various ML algorithms are trained using static, dynamic and integrated attributes. The results demonstrate that the integrated approach performs better as compared to single approaches.

Chapter 4 presents an approach (MalDetect) for enhancing the detection results of Android apps. It is designed using the fusion of traditional ML algorithms on the basis of proposed ranking schemes. The ranking schemes are then used to generate various combination schemes, from which the best combination is chosen to construct the final model. The results indicate that MalDetect is more effective than ensemble learning techniques and traditional classifiers.

Chapter 5 presents an approach (CSForest) for the imbalanced family categorization of malware. The proposed technique results are compared with C4.5, CSTree and RF to see how good it is at categorizing malicious apps families. The findings suggest that the proposed approach is effective in determining malicious app families.

Chapter 6 describes a rule-based model to assign the risk levels (No, Low, Medium and High) to Android app features. The static features (permissions and API calls) in the data are examined statistically to come up with a hypothesis for identifying their risk factor. In order to test the hypothesis, ANOVA has been used. The results indicate that the mean values of different risk factors differ significantly. Afterward, a weight is assigned to the features under each category to compute the threat score of a particular app.

Chapter 7 presents the contributions of the research work carried out and concludes the study. It also highlights the scope of future work related to Android malware detection, classification and threat assessment.

CHAPTER 2

LITERATURE REVIEW

As discussed in chapter 1, the increasing demand for Android phones attracted many organizations to build various apps such as gaming, education, business, entertainment, banking, lifestyles etc. The increasing use of Android apps also lured attackers to build malicious apps that pose several threats such as financial loss, information leakage etc. Nowadays, cybercriminals are creating more innovative, complex, advanced and new varieties of malware due to which the detection, classification and threat assessment of malware is turning out to be a real-life challenge. In this chapter, a primary focus is given to all state-of-the-art approaches for identification, classification and threat assessment of Android malware.

2.1 RELATED RESEARCH WORK

The literature review conducted in context to malware detection, classification and threat assessment can be divided into two parts: Signature based and ML based methods.

2.1.1 Signature-based Method

Lots of research has been conducted in the area of detection, classification and threat measurement of Android malware. In the mid-1990s, signature-based approach for malware detection was developed. This approach extracts the malicious file patterns by matching the signatures of malicious apps present in the database. This technique is very efficient and reliable for identifying the known malware [67-76]. Figure 2.1 shows the traditional signature-based approach for malware identification. The work related to the signature-based approach for detection, classification and threat assessment of Android malware is described as follows:

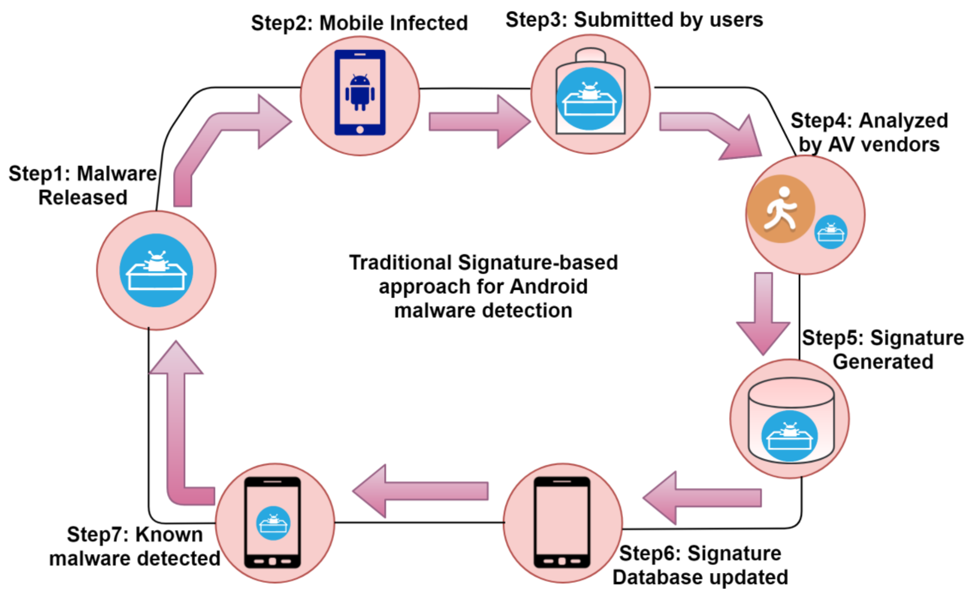

Step2: Mobile Infected
Step3: Submitted by users
Step4: Analyzed by AV vendors
Step1: Malware Released
Traditional Signature-based approach for Android malware detection
Step5: Signature Generated
Step7: Known malware detected
Step6: Signature Database updated

Figure 2.1: Process of signature-based approach for identification of Android malware

Venugopal and Hu [77] have proposed a mobile malware identification approach that needs less memory to scan the mobile devices. Furthermore, the authors compared their proposed technique with the renowned Clam-AV scanner. The findings illustrated that their approach takes 50% less memory than Clam-AV and provides a fast scanning rate. Faruki *et al.* [78] have introduced a method (AndroSimilar) that creates a signature by mining statistically unlikely attributes to identify malware apps. This method is very effective against repacking and code obfuscation mostly used to avoid AV signatures and to disseminate hidden variants of familiar malware. It is a mechanism that discovers areas of statistical similarity with familiar malware to identify those unknown samples. The results suggested that the AndroSimilar approach is very efficient and robust in comparison to fuzzy hashing approaches.

Zheng *et al.* [79] have presented a technique i.e. DroidAnalytics which is based on a signature-based approach that automatically collects, examines and analyses the mobile malware. The experimental findings demonstrated that DroidAnalytics is much more effective in examining the malware mutations and repacking and also it is a very efficient tool. Ngamwitroj and Limthanmaphon [80] have introduced a signature-based Android malware identification approach using broadcast-receiver data and permission from the manifest file. The evaluation findings showed that the accuracy acquired by the approach to detect malware app was 86.56%.

Feng *et al.* [81] have suggested a novel approach (Apposcopy) to determine malware. The signature matching technique of Apposcopy uses an Inter-Component Call (ICC) Graph and integration of static taint analysis to effectively determine apps that have properties like control and data flow. The findings indicated that it is very efficient and reliable approach for determining malware families. Tchakounte *et al.* [82] have proposed a system named as LimonDroid, a desktop security tool that contains various schemes. The proposed approach was tested on 300 benign and 341 malware apps on a database of 62 YARA malicious families patterns, 12,925 fuzzy hashed malware signatures and VirusTotal engine. The results demonstrated that the suggested approach is more effective for users and provides detection accuracy of 97.82%.

The major constraint of this technique is that it is incapable of detecting new malware (Zero-day). To overwhelm this constraint, the researchers used ML algorithms to develop several detection techniques (such as static and dynamic analysis). This method makes use of attributes mined after performing static and dynamic analysis of malware. The extracted attributes are used to train the model for making predictions.

2.1.2 Machine Learning Method

This method makes use of attributes mined after performing static or dynamic analysis of malware. The extracted features are used to train the ML models for making predictions pertaining to Android apps. The research work related to ML methods for malware detection, classification and threat assessment using static and dynamic features are described as follows:

2.1.2.1 Using Static Features

It is considered as one of the methods for the identification and classification of malware. It examines the sample of malware without executing or running the code. It uses decompiling methods to decompile the app package and mine the attributes for the detection of malware [83-114]. The following research work related to static malware analysis is addressed as follows:

Li *et al.* [115] have presented an approach (named Significant Permission Identification (SigPID) that uses permissions as features to determine malware. They mined the permission data to determine the important permissions that can efficiently characterize the apps as malicious or benign. The authors compared their results with the existing state of the art approaches. The results showed that SigPID is more efficient in detecting malware and the accuracy obtained by unknown malware was 91.4%. Zhu *et al.* [116] have proposed a technique (named as DroidDet) to mine APIs, permissions, permission rate and examine system events as a key feature. They applied ensemble rotation forest to develop a model for figuring out whether an app is infected with malware. The experimental outcomes suggested that the proposed technique obtained high accuracy i.e. 88.26% as compared to other existing approaches.

Kim *et al.* [117] have presented a framework to determine malware. The authors extract several types of features and these are refined using similarity-based or existence based feature extraction approach. Moreover, the authors also proposed a multimodal Deep Learning (DL) model to determine the app. The accuracy of the proposed model is compared with other deep neural network techniques. Feizollah *et al.* [118] have examined the efficiency of Android intents and permissions as a characterizing feature for determining the malware apps. The experiment was conducted using 7,406 apps out of which 1,846 apps are benign and 5,560 are malicious apps. The results indicated that the integration of both attributes results in a high identification rate i.e. 95.5% in comparison to individual features.

Wang *et al.* [119] has thoroughly investigated the permission risk in Android apps. They examine the risk of an individual and group of permissions. They then applied three distinct feature ranking algorithms such as T-test, mutual information and correlation coefficient to rank the permissions according to the risk factor. To determine the subsets of risky permissions they used Principal Component Analysis (PCA) as well as sequential forward selection. In the end, for the detection of malicious apps they compared their technique with conventional techniques such as RF, DT and SVM. The findings demonstrated that the detection accuracy attained by the proposed approach was 94.62% with a 0.6% FP rate. Alazab [120] has introduced a system for classifying Android apps using a real dataset premised on static analysis. The authors examined two attribute selection methods i.e. ANOVA and Chi-Square in combination with ten ML classifiers. They then evaluated the detection accuracy of each classifier to determine the best one for detecting malware using distinct attribute sets. It was found that Chi-Square had higher detection accuracy than

ANOVA. The proposed system achieved 98.1% detection accuracy and took 1.22 seconds to classify.

Arora *et al.* [121] have introduced PermPair, a novel detection model that builds and compares graphs for normal and malware by mining permission pairs from an application's manifest file. The authors when compared their proposed approach to other similar approaches and anti-malware apps, the outcomes revealed that the proposed approach is effective in identifying malware with an accuracy of 95.44%. Agrawal and Trivedi [122] have analyzed different malware identification techniques with different ML classifiers. The findings suggested that RF performed better than other ML classifiers.

Sahin *et al.* [123] have designed ML based system to distinguish malware from goodware apps. The proposed system aimed to eliminate unnecessary attributes by using linear regression based features selection method. The author employed seven ML classifiers such as Multilayer Perceptron (MLP), Sequential Minimal Optimization (SMO), Naive Bayes (NB), RF, C4.5, Logistic Regression (LR) and K-Nearest Neighbors (KNN) to identify malware. The findings demonstrated that the F-measure acquired by the proposed method was 0.961. Further, the authors claimed that this system is effective for detecting real time apps. Bai *et al.* [124] have suggested a system that could detect malware as well as classify it into families. The authors used permissions and opcode sequences as features that are acquired from Manifest.xml and classes.dex file. A fast correlation based filter algorithm was used for dimensionality reduction. The authors employed CatBoost classifier for classification purpose. The findings indicated that the accuracies achieved by both binary and family classification was 0.974 and 0.9738 respectively.

Yuan *et al.* [125] have proposed an algorithm for both identification and family classification of malware. The authors proposed Time Frequency-Inverse Document Frequency (TF-IDF) algorithm based on static permissions. This algorithm is used to compute the permission value of every permission and sensitivity value of apk of the app. After that, the authors used various classification algorithms such as Random Tree, NB, K-NN, C4.5, RF and Bayesian Network. The proposed approach was evaluated on 9,419 malware apps and 6,070 benign apps. The results demonstrated that the accuracy achieved in the case of malware detection was 99.5% whereas the accuracy achieved in the case of family classification was 99.6%. Sangal and Verna [126] have introduced an approach based on attributes to identify malware.

The authors worked on the ClclnvesAndMal2019 dataset and used intent and permissions as a feature set for detection. They used principal component analysis as a features selection technique. The well-known ML classifiers are employed to identify malware. The findings suggested that RF performed better with an accuracy of 96.05%.

Yerima *et al.* [127] have investigated an approach that was premised on parallel ML classifiers to detect malware. The authors first carried out the experiments with individual base classifiers such as Simple Logistic (SL), RIDOR, Partial Decision Tree (PART), NB and DT. Then, they carried out the experiments by combining different classifiers based on average of probabilities, maximum probabilities, majority voting and product of probabilities. The experimental consequences demonstrated that the product of probabilities performs better for identifying malware. Coronado-De-Alba *et al.* [128] have presented a meta-ensemble approach premised on static malware analysis to detect Android malware. Moreover, they introduced a comparative analysis of different Ensemble Learning methods to identify the best combination of classifiers premised on the evaluation of classification results.

Yerima and Sezer [129] have presented an approach named as DroidFusion. The fusion of classifiers was premised on different ranking combination schemes. They presented the experimental outcomes on four different datasets to show the effectiveness of the proposed model. The authors then compared the usefulness of the DroidFusion with the stacking ensemble method. The findings suggested that the DroidFusion was much more effective than the stacking technique to identify malware. Idrees *et al.* [130] have introduced a Plndroid that was premised on intents and permissions for the detection of Android apps. It makes use of a combination of intents and permissions with the ensemble approach for correctly classifying malware. The experiment was carried out on 1,745 apps to detect malware. The proposed framework provided 99.8% detection accuracy and also showed the effectiveness of the proposed framework.

Milosevic *et al.* [131] have introduced two approaches premised on permissions and code analysis using a bag-of-words demonstration model with ML The author employed C4.5, Random tree, SVM, JRip, RF and linear regression ML classifiers. The experimental findings suggested that the F-score acquired by both approaches were 89% and 95.1% for the permission based and source code based models. Wang *et al.* [132] have introduced an efficient and effective approach to managing the market to identify benign and malicious apps. The authors mined 11 different types of static attributes to describe the behaviors of the apps. They applied an ensemble of classifiers such as K-NN, Classification and Regression Tree (CART), NB, RF and SVM to identify the malicious apps. The proposed approach was tested on a dataset that contain 8,701 malicious apps and 1,07,327 benign apps. The outcomes illustrated that their model obtained better accuracy i.e. 99.39% for detecting malicious apps and obtained the accuracy of 82.93% in classifying benign apps.

Wang *et al.* [133] have proposed a novel method (i.e. Mlifdect) that used parallel ML and fusion techniques to better detect Android malware. The authors mined eight distinct types of static features. Then, they build a parallel ML identification model for spreading up the classification process. Furthermore, they investigated the probability analysis premised on Dempster-Shafer theory based fusion methods which obtained better detection results. The findings suggested that the Milfdect is much more capable of acquiring a higher detection rate than other solutions for detecting malware. Dehkordy and Rasoolzadegan [134] have applied Synthetic Minority Oversampling Technique (SMOTE), undersampling technique and their combination to balance the data. Then, the authors applied Iterative Dichotomiser3 (ID3), SVM and K-NN were used to identify Android malware. The findings indicated that the performance of K-NN with SMOTE was better than other classifiers. The accuracy obtained by K-NN with SMOTE was 99.49%.

Shrivastava and Kumar [135] suggested a framework named as SensDroid that assessed the performance of Android permissions and intents as a distinguishing trait to spot malware apps through sensitive analysis methods. The outcomes illustrated that the proposed approach was effective in distinguishing the clean and the infected apps. The accuracy acquired by the proposed framework was 98.65%. Wang *et al.* [136] have suggested a framework named as DroidRisk for quantitative security risk evaluation of Android apps based on permissions. The authors evaluated their framework on 27,274 benign and 1,260 malicious apps. The findings illustrated that the proposed approach is more reliable in providing the risk signal.

For malware detection, Onwuzurike *et al.* [137] developed MAMADROID, a static malware analysis system. Malware detection relies on static features like API calls and call graphs. The results were tested on 3.5 million malicious and 8.5 million benign apps. The proposed method yielded an F-measure of 0.99. Ye *et al.* [138] have introduced a method that calculates the risk of an app. The authors created and deployed a fuzzy logic system to calculate the total risk. They suggested a risk classification-based method for malware detection based on the quantitative estimation model. The experiments showed that the RF algorithm achieved high accuracy i.e. 93.2% with a low FP rate.

Xu *et al.* [139] have presented a new technique i.e. Fuzzy-SMOTE which was based on SMOTE and fuzzy set theory. The results showed that Fuzzy-SMOTE achieved the highest accuracy when compared to Borderline-SMOTE. Table 2.1 shows the comparative study of detection, classification and threat measurement of malware using static features.

Table 2.1: Comparative study for detection, classification and threat measurement of Android malware using static features

Authors	**Features**	**Data Source**			**Technique used**	**Results**
			Malicious	**Benign**		
Li *et al.* [115]	Permissions	Google Play and Anzhi store	5,494	3,10,926	Proposed SigPID and compared with Random Committee, Rotation Forest, Functional Tree (FT), PART, RF, SVM	SigPID is more efficient by identifying 93.63% of malware and 91.4% of new malware.
Zhu *et al.* [116]	API calls and Permissions	Official app store and virusshare	1,065	1,065	Ensemble Rotation Forest	Accuracy obtained by proposed approach was 88.26%
Kim *et al.* [117]	String, Permission, Shared library function opcode, API calls and Method	VirusShare, Google Play Store and Malgenome Project	13,075	19,747	Multimodal Neural Network	Accuracy acquired by proposed approach was 98%.

	opcode					
Feizollah *et al.* [118]	Permissions and Intents	Google Play store and Drebin	5,560	1,846	Perform analysis of features	Combination of attributes results in better detection i.e. 95.5%.
Wang *et al.* [119]	Permissions	Google Play store and Mal-com 1 and Mal-com 2 from antivirus companies	29,216	3,15,794	Mutual Information, correlation coefficient and t-test in combination with SVM, DT and RF	The detection rate of proposed approach was 94.62% with a 0.6% FP rate.
Alazab [120]	API calls	Play store, Androzoo, Contagion mobile, Malshare	17,915	19,000	Chi-square and ANOVA in combination with ten ML algorithm	The detection rate of proposed approach was 98.1%.
Arora *et al.* [121]	Permissions	Genome, Koodous, Drebin and Google Play Store	6,208	5,993	PermPair to detect Android malware	The detection rate of proposed approach was 95.44%.
Sahin *et al.* [123]	Permissions	APKPure, Android malware dataset	1,000	1,000	Linear regression with seven different ML algorithms	F-measure obtained by the proposed method was 0.961.
Bai *et al.* [124]	Dalvik opcode sequences and Permissions	Third-party markets and open source dataset	---	---	Fast Android Malware Detector (FAMD) with CatBoost	Accuracy achieved by both binary and family classification dataset was 0.974 and

						0.9738 respectively
Coronado-de-Alba *et al.* [128]	Permissions, hardware components and intents	Drebin project, Third party stores and Google play store	1,531	1,531	Random Committee and RF and Meta-ensembling RF in Random Committee	Accuracy obtained by the proposed model was 97.56%.
Yerima and Sezer [129]	API calls, command strings, intents and permissions	Drebin, MCAFEE-350, Malgenome, and MCAFEE-100	D1: 5,560 D2: 13,805 D3: 1,260 D4: 13,805	D1: 9,476 D2: 22,378 D3: 2,539 D4: 22,378	Random Tree, REPTree, RF, AdaBoost and J48 and uses multiranking algorithm	The proposed approach was far much effective in combining different classifiers
Wang *et al.* [133]	Permissions, intents, hardware Features and API calls	Anzhi	8,701	1,07,327	SVM, RF NB, CART and KNN and uses majority voting	Accuracy achieved by proposed approach was 95.39%.
Idrees *et al.* [130]	Permissions and intents	Google Playstore , Genome Contagio dump, VirusShare, Virus Total and MalShare	1,300	445	MLP, DT and Decision Table and uses Product of probabilities, Average of probabilities and majority vote	Accuracy obtained by the proposed approach was 99.8%.
Milosevic *et al.* [131]	Permissions	M0Droid	200	200	SVM, C4.5, JRip, LR, Random Tree, RF and DT and uses majority vote	Accuracy obtained by the best fusion model was 95.6%.

Shrivastava and Kumar [135]	Permissions and intents	Google Play Store and Drebin	5,680	2,973	SensDroid to detect the apps based on the risk of features	Accuracy obtained by SensDroid was 98.65%.
Dehkordy and Rasoolzadegan [134]	Intents, permissions, Hardware component and API calls	Third party apps, Drebin and AMD dataset	2,723	6,500	K-NN with a combination of SMOTE+ random undersampling	K-NN with a combination of SMOTE+ random undersampli-ng Provides better detection accuracy with 98.69%.
Wang *et al.* [136]	Permissions	Google Play store and Genome Project	1,260	27,274	DroidRisk for quantitative security risk evaluation	DroidRisk produced a reliable risk signal.
Xu *et al.* [139]	Permissions	Google play store and DroidDream	D1: 560 D2: 343 D3: 199 D4: 120 D5: 100 D6: 80 D7: 54 D8: 44 D9: 268 D10: 396	D1: 1,017 D2: 1,017 D3: 1,017 D4: 1,017 D5: 1,017 D6: 1,017 D7: 1,017 D8: 1,017 D9: 500 D10: 1,528	Fuzzy-SMOTE	Fuzzy–SMOTE achieves higher accuracy.

--- Not specified

This method has some limitations that it is ineffective to examine the code obfuscation and morphed malware [140, 141] though it is quicker in determining malware. To overwhelm the hindrances of the static method, the dynamic method is used. It can track the characteristic of the app and precisely determine the unknown malware.

2.1.2.2 Using Dynamic Features

It examines the characteristics or behavior of an app while it is running in the virtual environment. It is more effective as it keeps on tracking the behavior of the apps at the time of execution [142-160]. Some research works related to dynamic malware analysis are the following:

Feng *et al.* [50] have proposed an efficient dynamic framework named as EnDroid to determine highly accurate malware based upon dynamic behavior features. They employed a feature selection technique to eradicate irrelevant and noisy features and extract important features. Furthermore, EnDroid employed a stacking ensemble method to characterize the malicious app from benign apps. The experimental outcomes demonstrated that stacking obtained better performance rate and provided a promising solution for the detection of malware. Mahindru and Sangal [154] have presented a framework named as ML-Droid that identified the malware from mobile devices. This framework used dynamic analysis to identify mobile malware. Furthermore, various ML methods are employed using dynamic features to aid in the construction of a model. The experiment was carried out on a total of 5,00,000 Android apps. The results indicated that the accuracy attained by the proposed was 98.8%.

Cai *et al.* [161] have proposed a dynamic classification approach (named as DroidCat) to enhance the existing technique. The authors used a distinct set of dynamic features in accordance with ICC intents and method calls. Furthermore, the results illustrated that the DroidCat obtained high accuracy i.e. 97% in comparison to the state of the art methods to detect or distinguish malicious apps. Das *et al.* [162] have proposed a hardware architecture called GuardOL to carry out online malware detection. They built a multilayer perceptron in Field Programmable Gate Arrays (FPGA) to train classifiers using these dynamic features. The findings illustrated that the approach used less power consumption and provided a faster detection rate.

Enck *et al.* [55] have addressed an approach named as TaintDroid, an effective dynamic taint analysis system that continuously tracks sensitive data from different sources. The findings intimated that TaintDroid offered valuable input for mobile users and safety service firms seeking to determine the misbehaving apps. Chen *et al.* [163] have presented an architecture that utilizes model based semi-supervised classification technique based on dynamic API calls. Authors compared their methods with well-known classifiers like SVM, Linear Discriminant Analysis (LDA) and K-NN. The finding suggested that the proposed architecture obtained 98% detection accuracy which is higher than the other approaches.

Zheng *et al.* [164] have designed a system ("DroidTrace") based on dynamic analysis which permits analysts to carry out a systematic study of dynamic payloads with malware apps. It carried out forward implementation to trigger diverse dynamic loading performance. The authors showed their experiment on 50,000 benign apps and 294 malicious apps with ten families. Afonso *et al.* [165] have presented a framework to dynamically detect whether an app is malware or benign. The proposed framework was evaluated on 7,520 apps out of which 3,780 apps were used for training and the rest 3,740 apps were used for testing. The detection accuracy of the proposed framework was 96.66%.

Mahindru and Singh [166] have suggested a detection system based on dynamic permissions. The authors applied different ML classification algorithms such as RF, k-star, J48, SL and NB to detect malicious apps. The experimental results suggested that among all classification algorithms SL performed better. For dealing with imbalanced malware datasets, Oak *et al.* [167] employed a paradigm called Bidirectional Representations for Transformers (BERT). There are 1,80,000 apps in their dataset, with two-thirds of them being malicious. In spite of an extremely unbalanced dataset, BERT was able to detect malicious code with acceptable accuracy. The BERT model had an F1-score of 0.919.

Pang *et al.* [168] have suggested a novel method i.e. AWGSENN to address the problem of imbalanced classes. This method uses the Gaussian distribution probability density function to generate new instances. The results demonstrated that the proposed technique is effective than other resampling techniques. Table 2.2 illustrates the comparative study for detection and classification and threat measurement of malware using dynamic features.

Table 2.2: Comparative study for detection, classification and threat measurement of Android malware using dynamic features

Authors	**Features**	**Data Source**			**Technique used**	**Results**
			Malicious	**Benign**		
Feng *et al.* [50]	System calls, Cryptographic operation and Network Operation	Google Play Store, AndroZoo and Drebin	D1:5,213 D2:5,000	D1:8,806 D2:5,000	Stacking ensemble technique	Accuracy and F-measure obtained by stacking ensemble technique was 96.49% and 95.21% respectively
Mahindru and Sangal [154]	Permissions and API calls	Google Play Store, Virus Total, hiapk, AndroMalShare slidme	---	---	ML-Droid to identify malware from the device	Accuracy acquired by the proposed framework was 98.8 %
Cai *et al.* [161]	ICC Intents and Method calls	Google Play Store and MalGenome	D1:3,450 D2:3,190 D3:9,084 D4:1,254	D1:5,346 D2:6,545 D3:5,035 D4:439	DroidCat a dynamic classification technique	Accuracy obtained by DroidCat was 97%
Das *et al.* [162]	System calls	VX Heaven and Virusshare	472	371	Build Multilayer Perceptron in FPGA	Provides high accuracy for detecting new malware.
Chen *et al.* [163]	Dynamic API calls	Google Play and VirusTotal	31,777	24,217	Model Based Semi Supervised (MBSS) classification technique	Accuracy acquired by proposed approach was 98%.
Afonso *et al.* [165]	system call traces and API function calls	Malgenome Project, VirusShare and	4,552	2,968	Proposed a technique for dynamically	Accuracy acquired by proposed

		AndroidPIT market			identifying malware	approach was 96.66%.
Oak *et al.* [167]	Permissions and sequence of dynamic activities	Palo Alto Networks	1,20,780	60,390	BERT in order to deal with imbalanced dataset	F-score attained by BERT based model was 0.919.
Pang *et al.* [168]	Network traffic	VirusShare and 360zhushou	3,136	753	Proposed AWGSENN a resampling method for imbalance class problem	AWGSENN shows remarkable performance over the other seven resampling methods.

--- Not specified

The major constraint of this method is that it could not explore all execution paths. Sometimes, malware can detect that it is being carried out in the virtual environment then it will not show its characteristics. Due to executing stalling and obfuscation, Gandotra *et al.* [26] have concluded that individual static or dynamic methods are not suited for correctly classifying the malware. The researchers have therefore begun to use a hybrid approach to overcome this challenge.

2.1.2.3 Using Hybrid Features

It is an amalgamation of static and dynamic approaches. It takes advantage of the static and dynamic approaches [169-174].

Alzaylaee *et al.* [175] have proposed a DL approach (named as DL-Droid) using hybrid features to detect malicious apps. The experiment was conducted over 30,000 apps on real devices. The experimental outcomes showed that detection accuracy obtained by integrating both static and dynamic features is 99.6%, which is 1.8% higher than the accuracy obtained by the dynamic approach. Yuan *et al.* [176] have presented an online method i.e. DroidDetector based DL to detect whether an app is benign or malicious. The authors compared their proposed method with state of the art methods. The findings suggested that their method is more effective in characterizing malware in comparison to other methods. The detection accuracy obtained by the proposed method was 96.76%.

Tong and Yan [177] have introduced an integrated approach for the identification of mobile malware by considering both dynamic and static analysis. The author constructed the pattern of both malicious and benign sets by matching the pattern of both malware and benign app with one another. The findings of test set results suggested that their approach attained a better identification rate than other approaches. Martin *et al.* [49] have introduced OmniDroid, a massive dataset in which features are mined using dynamic and static methods. The authors introduced this to assist researchers and AV creators in building a new technique for identifying mobile malware. They proposed a detection method based upon both dynamic and static features using a combination of classifiers. The experimental findings showed the potential usability and feasibility of their framework.

Blasing *et al.* [178] have introduced an Android Application Sandbox (AASandbox) which make use of both static and dynamic approach to automatically identify the malicious file. Authors deployed both detection techniques and sandbox in the cloud for providing a fast detection rate. Further, the proposed method is much more effective in detecting mobile malware. Fu *et al.* [179] have presented an approach to detect mobile malware through static and dynamic attributes. The authors build and train Long Short Term Memory (LSTM) based model and then used a generative adversarial network to create augmented instances that mimic the behavior of newly emerged malware. The experimental results indicated that the classification accuracy attained by the proposed approach was 99.94% and the accuracy achieved by samples of newly emerged malware was 86.5%.

Qaisar and Li [180] have presented a multimodal analysis of malicious apps. The authors exploited dynamic, static and visual features of apps to detect the malware apps using information fusion. Their approach used semi-supervised technique to detect and classify malware. The findings suggested that their approach obtained 95% accuracy which was better than other traditional approaches. Kabakus and Dogru [181] have proposed a hybrid malware analysis technique named mad4a. This technique takes advantage of both dynamic and static methods. The importance of this approach is to reveal the unknown behavior of Android malware.

Abawajy and Kelarev [182] have introduced a system named as Iterative Classifier Fusion System (ICFS). The authors carried out the empirical study to identify the best option to be applied to ICFS and then compared the effectiveness of the proposed technique with existing ML classifiers. The consequences demonstrated that ICFS provided better results using a combination of NB, MLP, Lib SVM with polynomial kernel and applied Iterative Feature Selection (IFS) premised on wrapper subset with Particle Swarm Optimization (PSO). Gupta and Rani [183] have presented two approaches premised on ensemble learning and big data to enhance malware detection accuracy. The first approach is premised on the weighted voting scheme of ensemble learning, and the next approach selects an optimum set of ML classifiers for stacking purposes. The proposed technique was conducted using Apache Spark and the performance is evaluated and tested on a large dataset containing 1,98,350 files out of which 98,150 benign and 1,00,200 malware apps. The findings illustrated the effectiveness and better generalization of the proposed technique in identifying malware.

Sharma and Gupta [184] proposed the RNPDroid technique for risk mitigation using permissions. The proposed technique was evaluated on the M0Droid dataset which consists of 400 Android apps. The authors applied ANOVA test to check whether the null hypothesis was accepted or rejected. The experimental results demonstrated that the computed value of F i.e. 517.3 was significantly greater than the tabulated value of F is 2.61 at level of significance 5%. Table 2.3 illustrates the comparative study for detection and classification and threat measurement of malware using integrated features.

Table 2.3: Comparative study for detection, classification and threat measurement of Android malware using integrated features

Authors	**Features**	**Data Source**			**Technique used**	**Results**
			Malicious	**Benign**		
Alzaylaee *et al.* [175]	Permissions, application attributes and actions/events	Intel Security (McAfee Labs).	11,505	19,620	DL-Droid for Android malware detection	Detection accuracy of proposed technique was 99.6%.
Yuan *et al.* [176]	Permissions and sensitive API	Google Play Store, Genome	1,760	20,000	Droid-Detector for Android	Detection accuracy achieved by

		Project and Contagio Community			malware detection	Droid-Detector was 99.6%.
Tong and Yan [177]	System calls related to network and file access	Malgenome	D1: 147 D2: 195 D3: 195 D4: 195	D1: 126 D2: 187 D3: 195 D4: 195	Proposed a hybrid approach for malware detection	Proposed approach showed the feasibility and potential usability for malware detection
Martin *et al.* [49]	Permissions, services, system calls, receivers, activities, Opcodes, FlowDroid and API calls	AndroZoo and Koodous	21,018	11,973	AndroPyTool that automatically perform static and dynamic analysis of Android apps	Fusion of features performed well
Fu *et al.* [179]	Permissions, receivers action and system call	Apkpure, Android wake lock research project and Virusshare	3,090	3,090	LSTM based model for detection of malware	Accuracy obtained by the proposed model was 99.94%
Kabakus and Dogru [181]	Permissions and network traffic	Play store, Drebin, ASHISHB malware, Genome project and Contagio Mobile	2,999	2,809	Mad4a for analysing the characteristic of malware	This approach was more effective in detecting unknown characteristic of malware
Sharma and Gupta [184]	Permissions	m0droid dataset	200	200	RNPDroid to detect the apps based on the risk factor of features	Accuracy obtained by RNPDroid was 97.48%.

From here, it is concluded that a single approach is not capable of detecting malware more precisely. Thus to enhance the accuracy, the hybrid approach is being used which is the integration of both approaches. Through a comprehensive literature review, we are able to discover research gaps for this work.

2.2 INFERENCES DRAWN FROM LITERATURE REVIEW

The review of the literature shows that a significant research has been carried out in context to Android malware. However, there are many areas which are unexplored and need immediate attention. Following inferences are drawn from the elaborative literature review.

1. Most of the existing research relies on either static or dynamic malware features for building ML models to detect and classify malware [11, 12, 115, 118].
2. As there is a huge difference between the rate of infection and actual detection of malware, there is a scope of improvement in designing the methods for their better detection and classification [6, 11, 12].
3. In a real-world scenario, the number of samples differs greatly among various malware families. Thus, there is a need to build malware classification models which can take care of imbalanced classes [113, 115, 119].
4. There is a lack of adequate research on analyzing the threat or risk posed by Android apps [11, 12, 135].
5. With the increasing use of mobile apps, the volume and variety of mobile malware have increased significantly which requires the development of algorithms for malware detection and classification using big data tools [6, 11, 12].

Based on these inferences, the research objectives are framed as discussed in chapter 1.

2.3 SUMMARY

This chapter discusses the literature review conducted in context to malware detection, classification and threat assessment. The inferences are drawn from the elaborative literature review. Based on the inferences, the research objectives are framed. In the next chapters, the effective techniques are built to deal with these problems.

CHAPTER 3

PROPOSED INTEGRATED APPROACH FOR DETECTION AND CLASSIFICATION OF UNKNOWN MALWARE

As discussed in the previous chapters, the traditional defenses like AV and Intrusion Detection System (IDS)/Intrusion Prevention System (IPS) rely on signature-based methods and are therefore unable to identify zero-day malware. In order to address this problem, static and dynamic malware analysis is being used along with ML algorithms for malware detection. Single approach either static or dynamic is not able to accurately detect and classify the malicious apps because of obfuscation and execution-stalling techniques being used by attackers.

This chapter proposes an integrated set of static and dynamic attributes that can effectively analyze, detect and classify unknown malware. The detection refers to binary classification which comprises of two categories i.e. “benign” and “malware”. The multi-class classification is referred to as the family classification. Here, a malware family for Android refers to a collection of malware programs that exhibit similar characteristics and share a common set of source codes.

3.1 PROPOSED METHODOLOGY

The workflow of the proposed model used for the detection and classification of unknown malware is discussed in this section. This process comprises three phases: (1) data gathering, (2) data preparation and (3) identification and classification of families. Data is gathered from various sources like apkmirror [185], apkpure [186] and virusshare [187] in the initial step. The duplicate apps are removed using the Message-Digest (MD5) hash algorithm in the second phase, and then these apps are scanned with Avira AV [188] tool. After that, the static and dynamic analysis approaches are used to mine the attributes from Android apps. A self-developed python script is used to extract static features, which makes use of various automated tools like strings [26], Baksmali Disassembler [47, 48] and AXMLPrinter2 [46]. The attributes such as intents, API calls, command strings and permissions are mined through a static approach. CuckooDroid [52] is used to extract dynamic features. Features including dynamic permissions, information leakage cryptographic operation and system calls are mined through dynamic malware analysis. In order to remove the redundant and irrelevant attributes, an Information Gain (IG) feature ranking algorithm [189] is used. Several ML classifiers like K-NN, DT, PART, SVM, RF and NB are applied to detect and classify the apps. Figure 3.1 illustrates the workflow of the proposed approach used for the identification and classification of unknown malware. The detail description of different steps is given below.

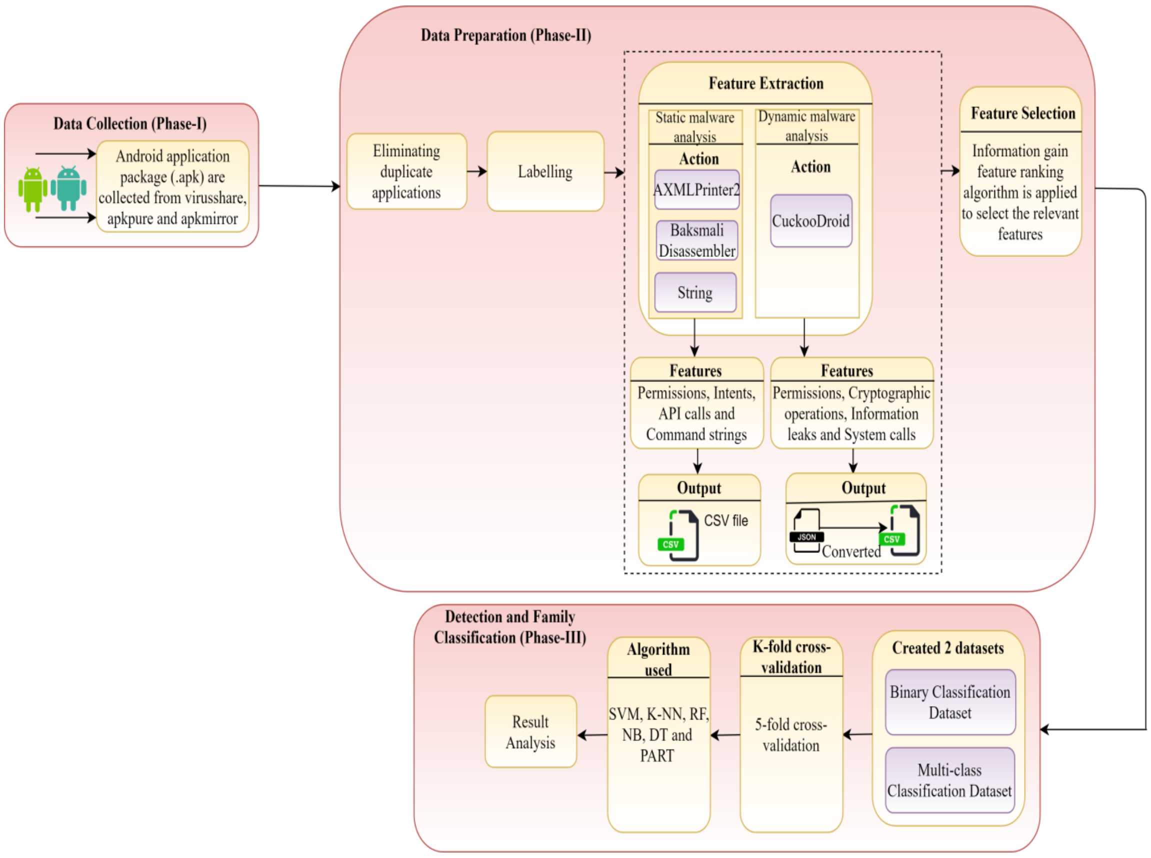

Data Collection (Phase-I)
Android application package (.apk) are collected from virusshare, apkpure and apkmirror
Data Preparation (Phase-II)
Eliminating duplicate applications
Labelling
Feature Extraction
Static malware analysis
Action
AXMLPrinter2
Baksmali Disassembler
String
Dynamic malware analysis
Action
CuckooDroid
Features
Permissions, Intents, API calls and Command strings
Features
Permissions, Cryptographic operations, Information leaks and System calls
Output
CSV file
CSV
Output
JSON
CSV
Converted
Feature Selection
Information gain feature ranking algorithm is applied to select the relevant features
Detection and Family Classification (Phase-III)
Created 2 datasets
Binary Classification Dataset
Multi-class Classification Dataset
K-fold cross-validation
5-fold cross-validation
Algorithm used
SVM, K-NN, RF, NB, DT and PART
Result Analysis

Figure 3.1: Workflow of the methodology used for detection and classification of unknown malware

3.1.1 Data Collection

Data gathering is the first step in the proposed methodology. Android apps are gathered from various sources including apkmirror, virusshare, and apkpure. The benign apps are gathered from apkpure and apkmirror. The malware apps are gathered from virusshare after registering on virusshare website and receiving permission from the administrator. A total of 4,400 Android apps are gathered from the different sources.

3.1.2 Data Preparation

This subsection describes the several steps used for preparing the data. It comprises eliminating duplicate apps, labelling, feature extraction and feature selection.

3.1.2.1 Eliminating Duplicate Applications

To remove the duplicate apps, the MD5 hash algorithm is applied. After eliminating the duplicates, we are left with 3,547 Android apps.

3.1.2.2 Labelling

Avira AV is used to label the Android apps which are left after removing the duplicates. After labelling, it is found that there are 1,747 malware and 1,800 benign apps. 13 distinct malware families are identified in 1,747 malicious apps. The name of the families along with the corresponding number of apps is shown in Figure 3.2.

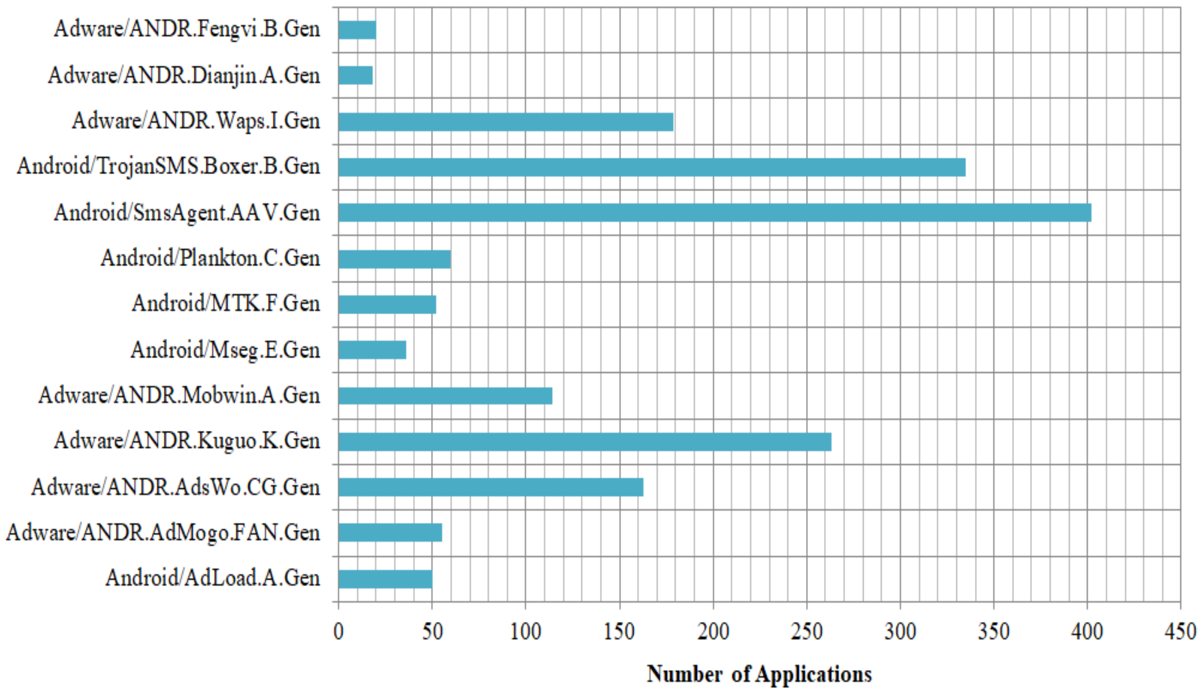


Figure 3.2: Android malware families

3.1.2.3 Feature Extraction

With the use of static and dynamic malware analysis, different attributes are mined. Four different categories of static attributes including intents, API calls, command strings and permissions are mined through static malware analysis. A self-developed python script is used to extract static features, which makes use of various automated tools like strings, Baksmali Disassembler and AXMLPrinter2. Four different categories of dynamic features including dynamic permissions, information leakage, cryptographic operation and system calls are mined through dynamic malware analysis using CuckooDroid (a tool for analysis of Android malware). The description associated with feature extraction through static and dynamic analysis is discussed as follows.

- **Static malware analysis-** It analyses the sample of malware without executing or running the code. A variety of disassembling techniques are employed to decompile the app's source code. Using Baksmali Disassembler, AXMLPrinter2, and string tools, a python script is constructed to mine static attributes. Features such as permissions, command strings, API calls and intents are mined using these tools. The procedure of mining static features is demonstrated in Figure 3.3. Firstly, the .apk file is unzipped or unpacked. The .apk file comprises of Android Manifest file, res, assets, classes.dex file and lib folder (as discussed in chapter 1). Using these files/folders, four different categories of static features are mined through various tools. AndroidManifest.xml file comprises information about permissions, classes.dex file comprises information about API calls and the rest all comprises information about command strings.

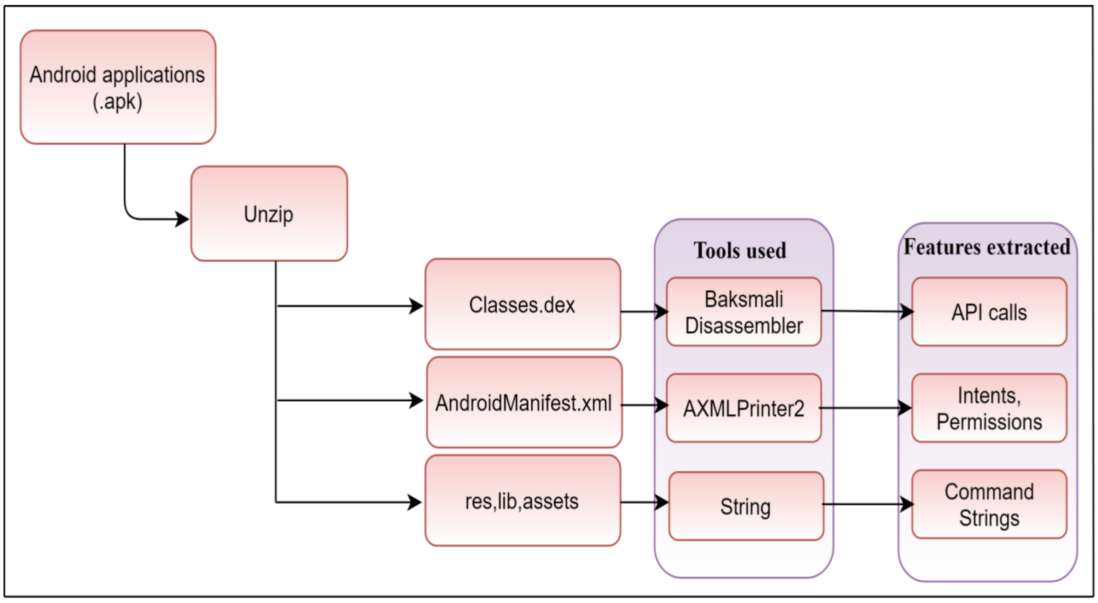


Figure 3.3: Process of mining static attributes

- **Dynamic malware analysis-** It is carried out while the code is being executed in the runtime environment. CuckooDroid is used to gather runtime behavior information of particular app. It is a continuation of the cuckoo sandbox, software for examining and executing the apps. CuckooDroid is responsible for handling Android emulator and generate report at the end of the analysis. The infrastructure of Cuckoo comprises the host machine (i.e. management software) and the guest machine (virtual machine that performs analysis). The main function of the host is to run the core sandbox components that control the entire analysis process, while the guest machine is the isolated environment where malware samples are executed. The guest comprises Linux virtual machine running Android emulator, which is supervised by the machinery module. Emulator for Android is primarily responsible for executing apps and returning data to CuckooDroid. A timeout of 180 seconds is set for each Android malicious file, meaning an Android sample has a maximum of 180 seconds to be examined before it expires. When the analysis of each sample is completed, the results are saved in Java Script Object Notation (JSON) format. In this process, the guest is to be rooted Android Virtual Device (AVD) with Xposed framework [190] and with its modules i.e. Droidmon and Emulator Anti-Detection. The python agent and the analyzer code operate on the guest machine using Python 2.7. When an APK file is received, the python agent's job is to perform an analysis on it. When the python analyzer runs an app, it returns screenshots and any dropped files to the host. After the procedure is completed, the log reports are collected and stored in JSON format. Reports of various apps are parsed and saved in CSV format in the database using Python scripts. Then these files are used to detect and classify malware. The procedure of mining dynamic features is demonstrated in Figure 3.4. Features including dynamic permissions, information leakage cryptographic operation and system calls are mined through dynamic malware analysis. Table 3.1 shows the description of the extracted features.

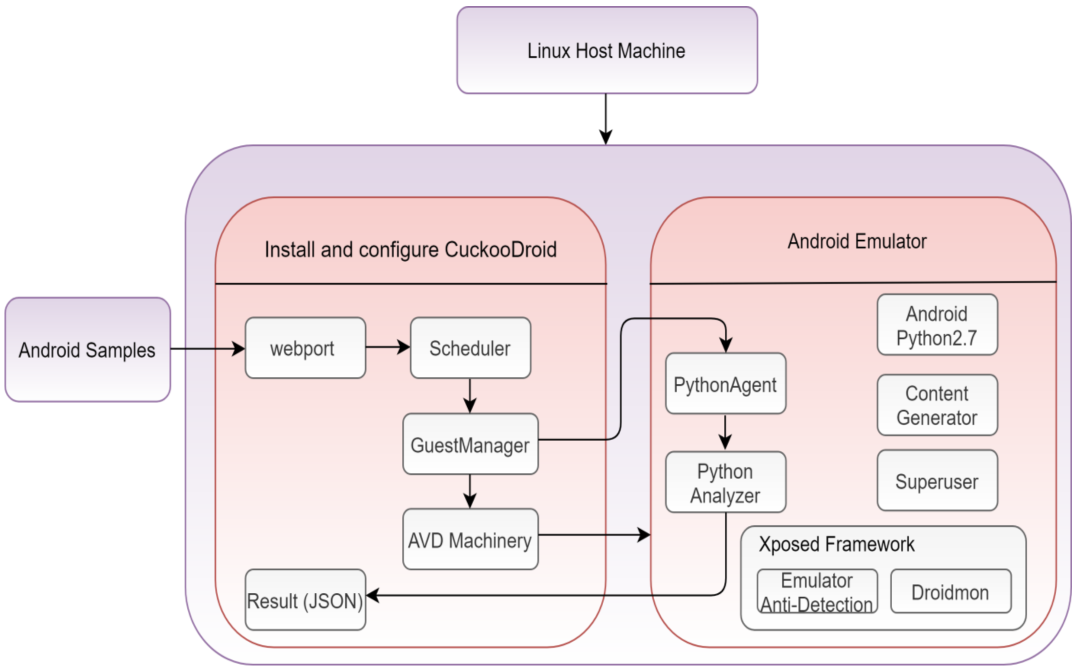


Figure 3.4: General framework of CuckooDroid for extracting dynamic features

Table 3.1: Description of mined attributes

Methods	Features	Tools Used	Examples	Number of attributes	Total attributes
Static	Permissions	AXMLPrinter2	READ_PHONE_STATE, RECEIVE_SMS, ACCESS_WIFI_STATE, READ_SMS, ACCESS_FINE_LOCATION	277	352
	Command Strings	String	Chown, chmod, remount, mount	6	
	Intents	AXMLPrinter2	ACTION_SHUTDOWN, SET_WALLPAPER, CALL_BUTTON, PACKAGE_CHANGED, NEW_OUTGOING_CALL	22	
	API calls	Baksmali Disassembler	PackageInstaller, GetCallingUid, Runtime.exec, getBinder, TelephonyManager.getCallState	47	
Dynamic	Information leaks	CuckooDroid	IMEI_Network, PHONE_NUMBER_File, IMEI_File	123	323
	Dynamic Permissions		ACCESS TO PASSWORDS FOR GOOGLE ACCOUNTS, WRITE CONTACT DATA, READ CONTACT DATA, AUDIO FILE ACCESS	71	
	System calls		PTRACE, RECVMSG, GETPID, SIGPROCMASK, SENDMSG, WRITE, SENDTO	50	
	Cryptographic operations		encryption_AES, keyalgo_AES, Decryption_AES	79	

3.1.2.4 Feature Selection

Variable selection is also known as Feature selection. It is carried out to reduce the dimensionality of data and helps in selecting the appropriate features. Irrelevant features lead to a decrease in the quality of the model. Moreover, it increases time and space complexity [191]. Choosing the appropriate attributes will help in minimizing the time and space complexity. It also helps in improving classification accuracy. IG feature ranking algorithm is employed to choose the appropriate attributes to better detect and classify malware.

IG computes the information a feature provides about the class. Entropy is used by IG to determine how homogeneous a sample is. The entropy $E(Z)$ of the dataset with k classes is computed as shown in Equation 3.1.

$$E(Z) = \sum_{i=1}^{k} -p_i log_2 \, p_i \tag{3.1}$$

Here p_i represents the probability of class i in dataset Z. Afterwards, the dataset is then divided on the various attributes X. Equation 3.2 calculates the entropy of a dataset in relation to the variable X.

$$H(Z,X) = \sum_{c \in X} P(k)H(k) \tag{3.2}$$

c denotes the possible values of the attributes X. IG is attained by a variable is calculated as given in Equation 3.3. More the IG of a specific attribute, more significant the attribute is.

$$IG = E(Z) - H(Z,X) \tag{3.3}$$

The IG technique allocates weight and rank to every feature. The attributes having a weight of 0 are ignored in this study. As a result, 110 and 47 static attributes are selected from detection dataset (named as Dataset-1) and multi-class classification dataset (named as Dataset-2) respectively. Figure 3.5 and Figure 3.6 demonstrate the top 20 attributes selected for binary and family classification of malware respectively.

Out of 323 dynamic attributes, 99 and 35 are selected from Dataset-1 and Dataset-2 respectively. Figure 3.7 and Figure 3.8 demonstrate the top 20 attributes selected for binary and family classification of malware respectively.

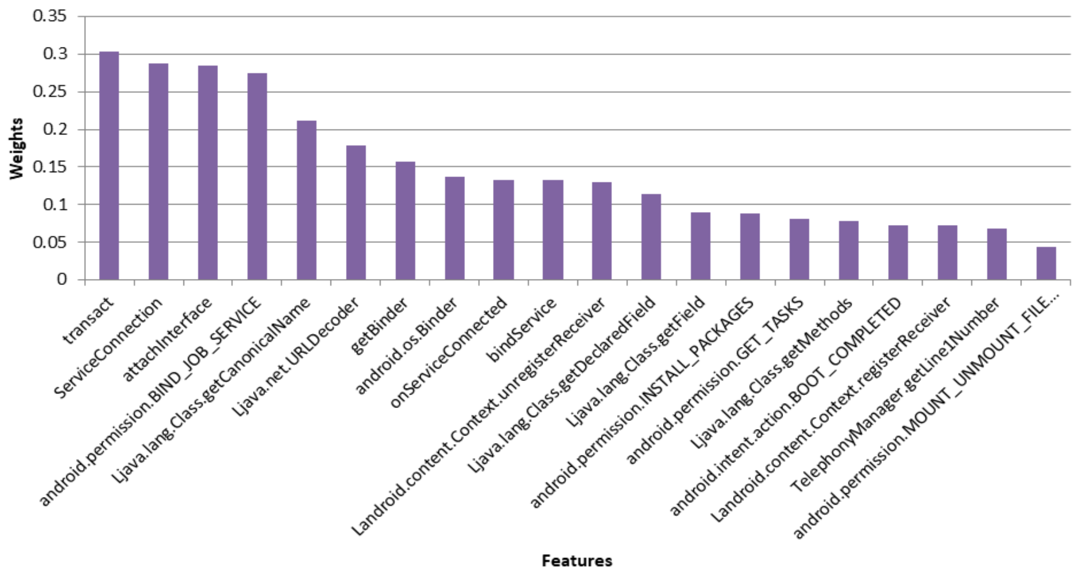


Figure 3.5: Top 20 static attributes of detection dataset (Dataset-1)

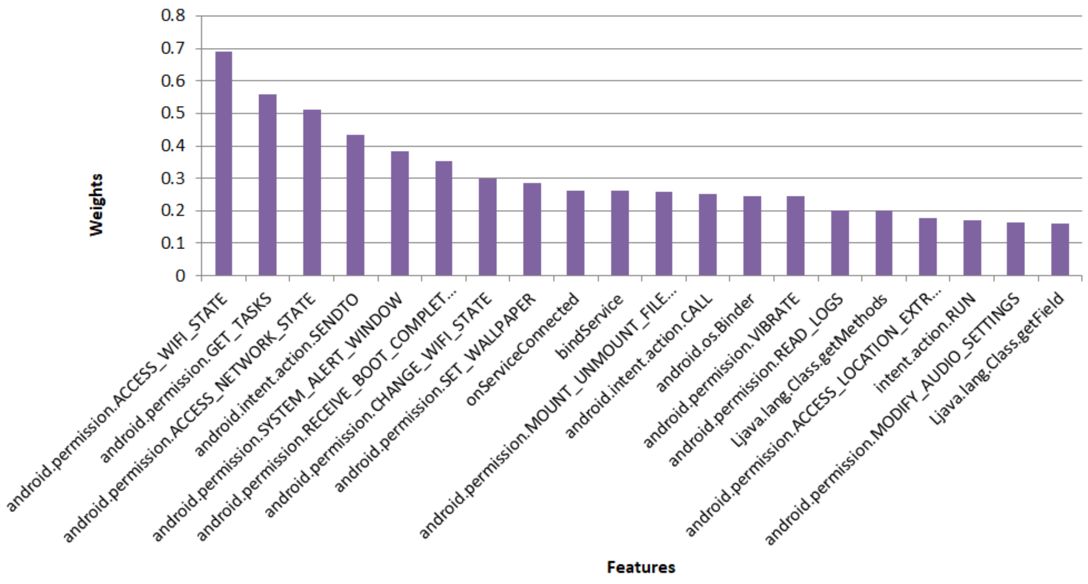


Figure 3.6: Top 20 static attributes of multi-class classification dataset (Dataset-2)

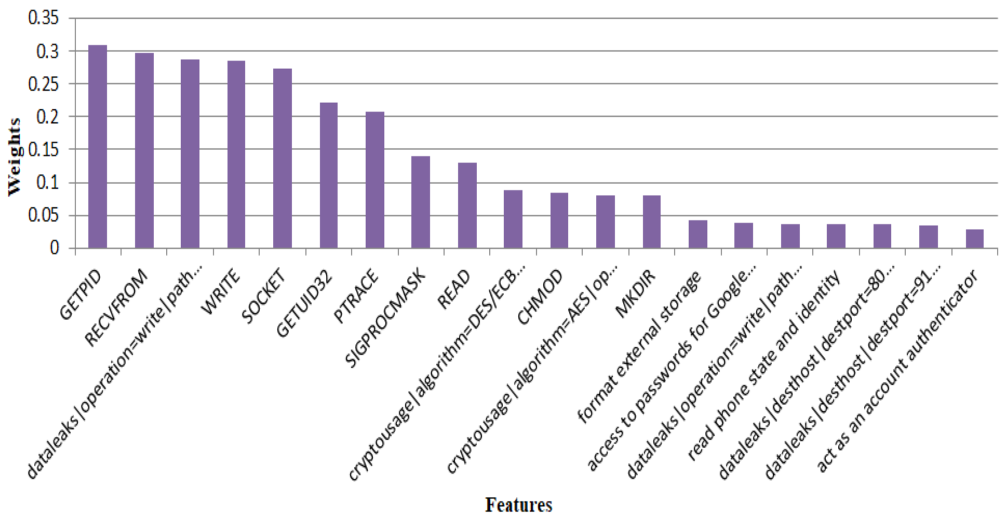


Figure 3.7: Top 20 dynamic attributes of detection dataset (Dataset-1)

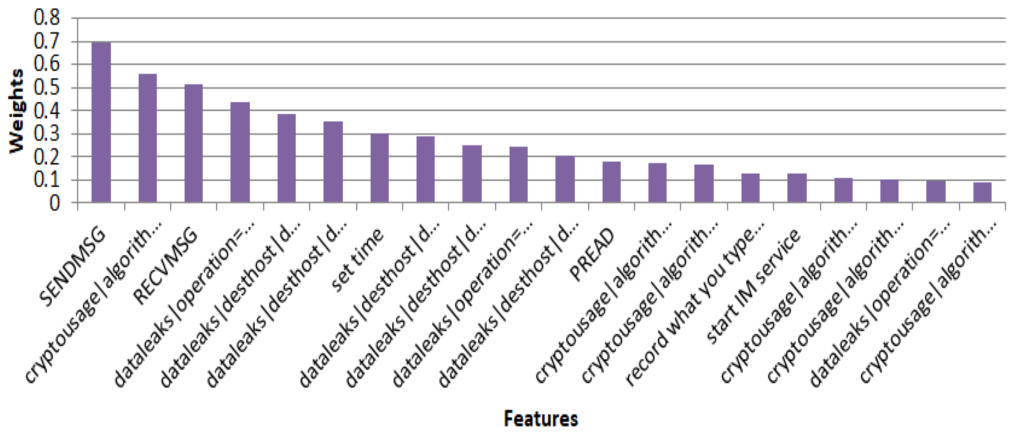


Figure 3.8: Top 20 dynamic attributes of multi-class classification dataset (Dataset-2)

Table 3.2 demonstrates the summary of both the datasets (i.e. Dataset-1 and 2) before and after selection of features.

Table 3.2: Detail description of datasets (Where # represents number of)

Datasets	**#Malware apps**	**#Benign apps**	**#Attributes extracted**		**#Attributes selected**	
			Static	**Dynamic**	**Static**	**Dynamic**
Detection (Dataset-1)	1747	1800	352	323	110	99
Multi-class Classification (Dataset-2)	1747 (with 13 families)	-----	352	323	47	35

Both static and dynamic malware analysis datasets are made available on GitHub (Link: https://github.com/Meghna-Dhalaria/Android-malware-dataset) and Kaggle (Link: https://www.kaggle.com/meghnadhalaria/android-malware-detection-and-classification). The process for preparing these two datasets is depicted in Figure 3.9.

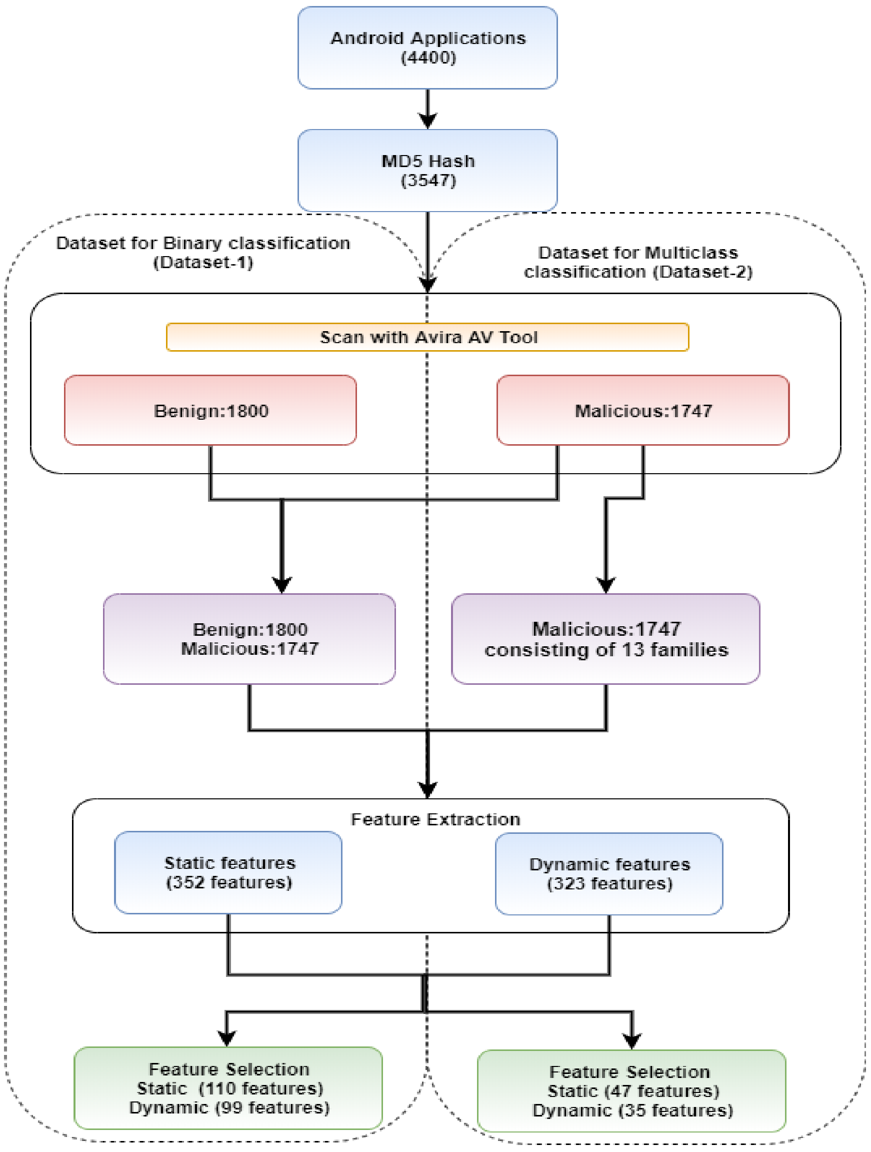

Android Applications (4400)
MD5 Hash (3547)
Dataset for Binary classification (Dataset-1)
Dataset for Multiclass classification (Dataset-2)
Scan with Avira AV Tool
Benign:1800
Malicious:1747
Benign:1800 Malicious:1747
Malicious:1747 consisting of 13 families
Feature Extraction
Static features (352 features)
Dynamic features (323 features)
Feature Selection Static (110 features) Dynamic (99 features)
Feature Selection Static (47 features) Dynamic (35 features)

Figure 3.9: Steps of data preparation

3.1.3 Detection and Family Classification

The third phase is to detect and classify Android malware. Several ML classifiers like DT, RF, NB, SVM, PART and K-NN are applied for the identification and classification of malware. The classifiers are trained using 5-fold cross-validation. This technique divides the dataset into five equal portions, out of which four portions are used for training and one portion is used for testing at every run. The description of ML classifiers is given below:

- **K-NN-** It is sometimes called a lazy learner [192]. It identifies the class label of a new observation on the basis of the similarity measure. This algorithm computes the distance between each row of training data and test data with the aid of Euclidean distance (as shown in Equation 3.4). After that, the distance values are then sorted in ascending order. It then selects the top k rows from the sorted list. At the end, it allocates a class to the new data based on the most often class of these rows.

$$d(a, b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2} \qquad (3.4)$$

Here a and b represents the two points in Euclidean n-space, b_i and a_i are the Euclidean vectors and n represents the n-space.

- **DT-** It is considered as the fundamental ML algorithm which is used for both regression and classification tasks. It has a tree like structure. It consists of an internal node (also known as non-leaf node), root node and the leaf node (also known as terminal node). The root node is the topmost node of the tree. The internal node and the leaf node show a test on the variable. The leaf or terminal nodes show the label class [193]. The purpose of using DT is to develop a training model that can be used to predict the class label by learning basic decision rules on the basis of previous data (training data). It uses a variety of algorithms (i.e. ID3, C4.5 and CART) to split a node into two or more sub-nodes. In this study, C4.5 algorithm is used.
- **RF-** It is based on the idea of ensemble learning, which is a method of integrating several classifiers to solve a complicated problem and to increase the model's performance. This combines several DT on different subsets of a dataset and takes the average to increase the predictive accuracy of the given dataset. Figure 3.10 depicts a graphical representation of the RF. On a large dataset, it is a highly effective and efficient approach [194].

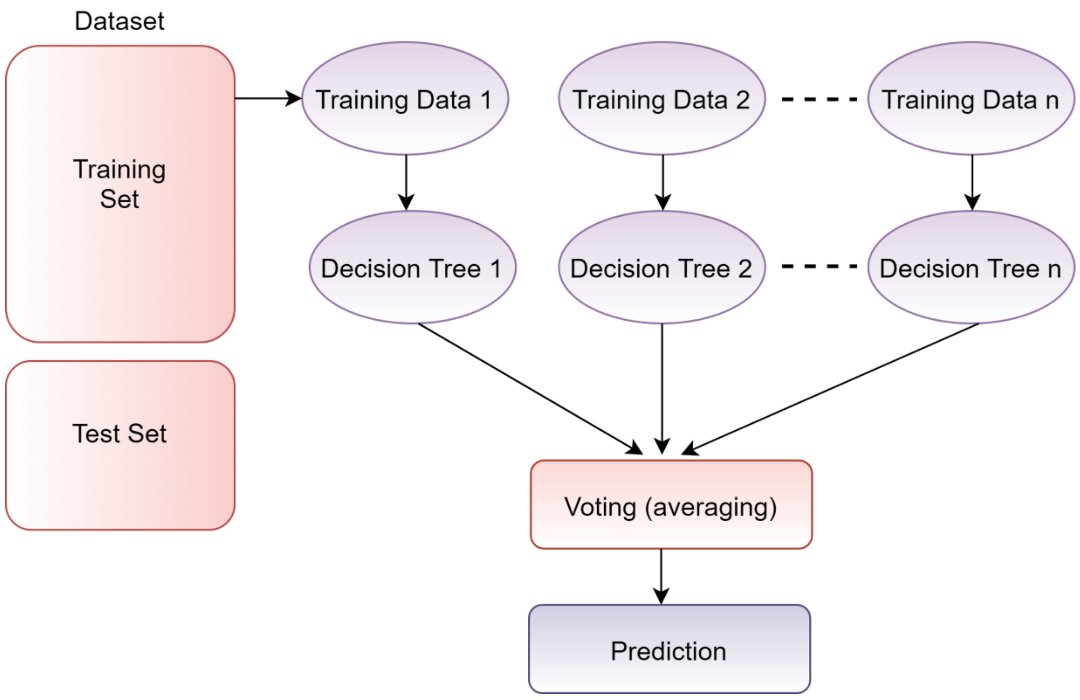


Figure 3.10: General framework of RF algorithm [194]

- **SVM-** It uses the decision surface to solve a 1-n class classification problem. The support vectors that are the nearest and equidistant points to this plane make up this decision surface [195]. Figure 3.11 depicts a graphical representation of the SVM.

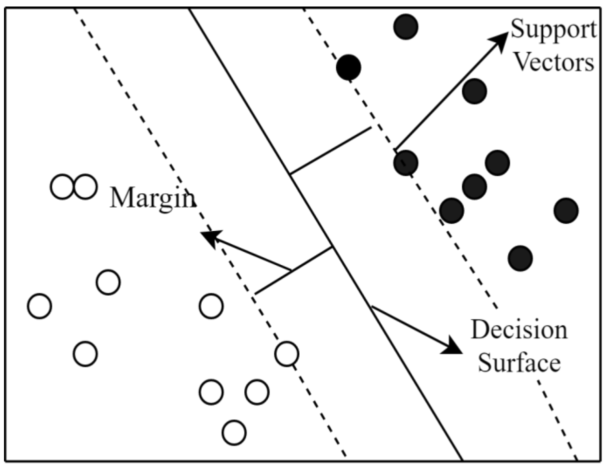


Figure 3.11: General framework of SVM algorithm [195]

The distance of a point (x_i, y_i) from decision boundary is referred to as functional margin as calculated in Equation 3.5.

$$\gamma^i = y_i(w^T x_i + b) \tag{3.5}$$

Here w^T is a hyperplane parameter normal to the decision boundary's surface, b is a constant and the point x_i is mapped onto a higher-dimensional space. If the point is far away from the surface, then it means there is the higher confidence in classifying the point. As a result, a higher functional margin indicates greater confidence in the predicted class of that point.

- **NB-** Bayes theorem (as calculated in Equation 3.6) is used to build the NB classifier, which is built on strong independent assumptions. It calculates the chances of a given occurrence in a dataset belonging to a particular class. It considers that the presence of an attribute in a class is independent of the occurrence of any other characteristic, i.e. all attributes contribute independently in computing the likelihood of data categorization. This model is suitable for very big datasets and is simple to construct [196].

$$P\left(\frac{X}{Y}\right) = \frac{P\left(\frac{Y}{X}\right) .\ P(X)}{P(Y)} \tag{3.6}$$

Here $P\left(\frac{X}{Y}\right)$ represents the probability of X occurring given evidence Y has already occurred, $P\left(\frac{Y}{X}\right)$ symbolizes the probability of Y occurring given evidence X has already occurred, $P(X)$ is the probability of X occurring and $P(Y)$ is the probability of Y occurring.

- **PART-** It is also known as a partial decision tree. The divide and conquer principle is used in this algorithm. It creates a decision list, which is a collection of rules. Each new instance is compared to every rule, and the class of the first matching rule is assigned to it [197].

3.2 EXPERIMENTAL RESULTS

This section summarises the experimental outcomes on the basis of static, dynamic, and integrated attributes. Six ML algorithms are employed and executed on python 3.7 on an Intel Core i5 64-bit processor with 8GB of memory. The experiments are carried out using a 5-fold cross validation technique. The ML algorithms used here are evaluated on different evaluation parameters like Sens, Accuracy, MCC, FPR, AUC, PPV and F-measure.

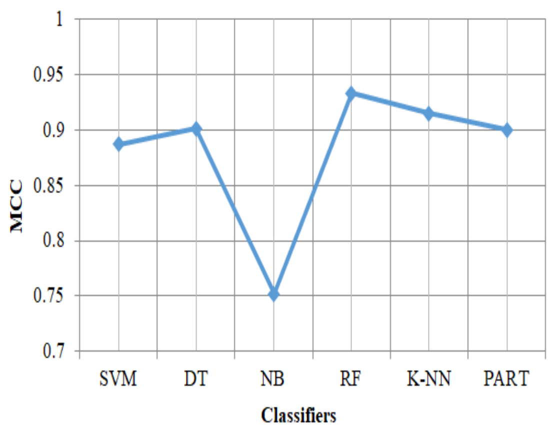
3.2.1 Results of Classification Using Static Features

Six ML techniques are employed for the identification and classification of malware using static features. These algorithms are implemented using Sklearn library [198].

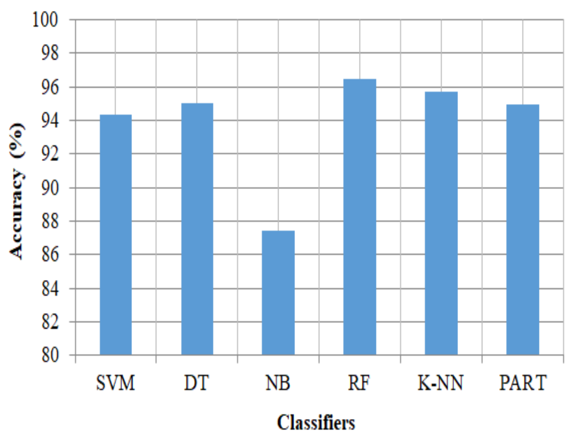
Table 3.3: Comparison of ML techniques using static attributes for detection dataset (Dataset-1)

ML model	Sens	FPR	PPV	F-measure	AUC	MCC	Accuracy (%)
DT	0.950	0.050	0.950	0.950	0.970	0.901	95.03
SVM	0.943	0.057	0.943	0.943	0.943	0.887	94.33
RF	0.965	0.035	0.965	0.965	0.990	0.933	**96.50**
NB	0.874	0.124	0.878	0.874	0.948	0.752	87.42
K-NN	0.957	0.042	0.958	0.957	0.989	0.915	95.74
PART	0.950	0.050	0.950	0.950	0.975	0.900	94.98

Table 3.3 indicates the results of ML algorithms using static attributes for detection dataset. It is observed that RF provides the best detection accuracy of 96.50% followed by K-NN which provides a detection accuracy of 95.74%.



(a)



(b)

Figure 3.12: Comparative analysis of various classifiers using static approach based on (a) MCC (b) Accuracy for detection dataset (Dataset-1)

Figure 3.12 compares the MCC and accuracy of several classifiers for detection dataset. It shows that RF obtains better results than other classifiers. The MCC and accuracy attained by RF are 0.933 and 96.50% respectively.

Table 3.4: Comparison of ML techniques based on static attributes for multi-class classification dataset (Dataset-2)

ML model	Sens	FPR	PPV	F-measure	AUC	Accuracy (%)
DT	0.848	0.023	0.852	0.847	0.949	84.77
SVM	0.859	0.023	0.863	0.857	0.962	85.86
RF	0.867	0.024	0.870	0.866	0.982	**86.72**
NB	0.751	0.032	0.792	0.756	0.967	75.10
K-NN	0.845	0.024	0.847	0.843	0.966	84.48
PART	0.840	0.024	0.842	0.839	0.947	84.02

Table 3.4 indicates the results of ML algorithms based on static attributes for multi-class classification dataset. It is observed that RF provides the best classification accuracy of 86.72% followed by DT and SVM which provide classification accuracy of 84.77% and 85.86% respectively. The Sens, F-measure and PPV acquired by RF are 0.867, 0.866 and 0.870 respectively which is higher than other ML classifiers.

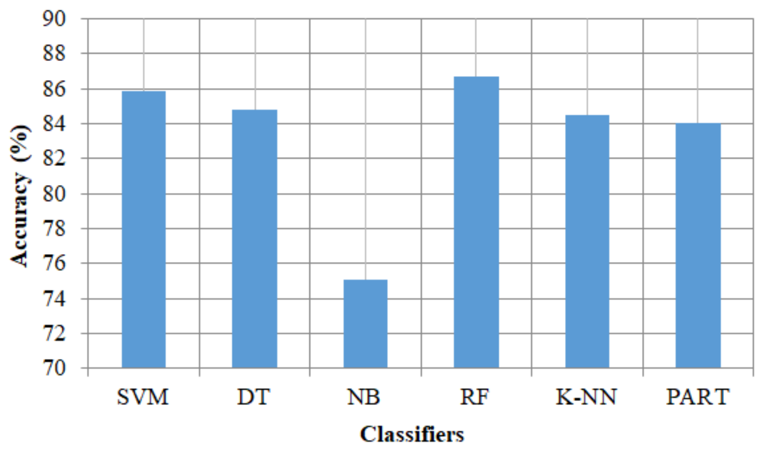


Figure 3.13: Comparative analysis of various classifiers using static approach based on accuracy for multi-class classification dataset (Dataset-2)

Figure 3.13 compares the accuracy of different ML classifiers for multi-class classification dataset. The accuracy attained by RF for Dataset-2 is 86.72% which is smaller than the accuracy attained by RF for Dataset-1.

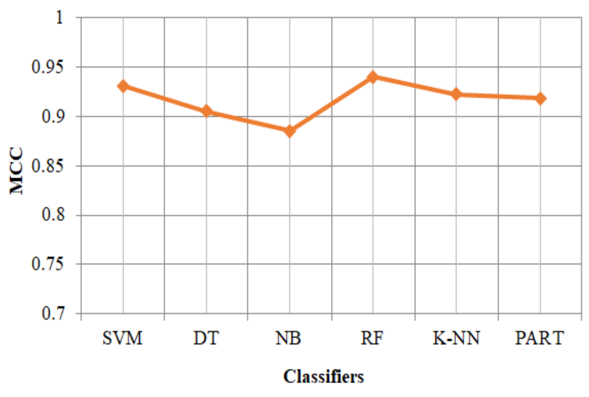
3.2.2 Results of Classification Using Dynamic Features

Six ML techniques are employed for the identification and classification of malware using dynamic features.

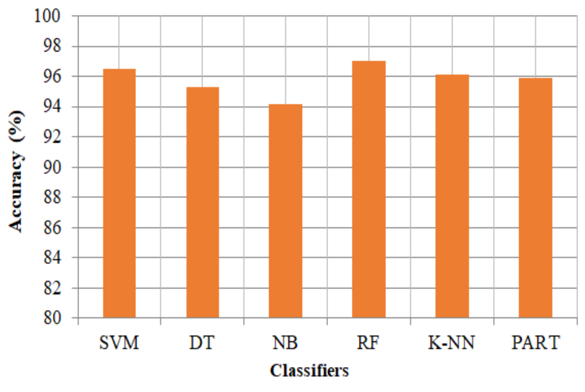
Table 3.5: Comparison of ML techniques based on dynamic attributes for detection dataset (Dataset-1)

ML model	Sens	FPR	PPV	F-measure	AUC	MCC	Accuracy (%)
DT	0.953	0.048	0.953	0.953	0.973	0.905	95.26
SVM	0.965	0.035	0.965	0.965	0.965	0.931	96.53
RF	0.970	0.030	0.970	0.970	0.996	0.940	**97.01**
NB	0.942	0.057	0.943	0.942	0.989	0.885	94.19
K-NN	0.961	0.039	0.961	0.961	0.990	0.922	96.08
PART	0.959	0.041	0.959	0.959	0.970	0.918	95.88

Table 3.5 indicates the results of ML algorithms using dynamic attributes for detection dataset. It is observed that RF provides the best detection accuracy of 97.01% followed by SVM which provides a detection accuracy of 96.53%.



(a)



(b)

Figure 3.14: Comparative analysis of various classifiers using dynamic approach based on (a) MCC (b) Accuracy for detection dataset (Dataset-1)

Figure 3.14 compares the MCC and accuracy of several classifiers for detection dataset. It shows that RF obtains better results than other classifiers. The MCC and accuracy attained by RF are 0.933 and 96.50% respectively. The MCC and accuracy attained by RF are 0.940 and 97.01% respectively.

Table 3.6: Comparison of ML techniques based on dynamic attributes for multi-class classification dataset (Dataset-2)

ML model	Sens	FPR	PPV	F-measure	AUC	Accuracy (%)
DT	0.843	0.026	0.843	0.841	0.947	84.25
SVM	0.864	0.021	0.871	0.866	0.985	86.85
RF	0.886	0.018	0.888	0.885	0.991	**88.60**
NB	0.800	0.029	0.805	0.795	0.951	79.96
K-NN	0.839	0.025	0.842	0.837	0.967	83.91
PART	0.841	0.026	0.838	0.836	0.950	84.08

Table 3.6 indicates the results of ML algorithms based on dynamic attributes for multi-class classification dataset. It is observed that RF provides the best classification accuracy of 88.60% followed by SVM which provides classification accuracy of 86.85%. The Sens, F-measure and PPV acquired by RF are 0.886, 0.885 and 0.888 respectively which is higher than other ML classifiers.

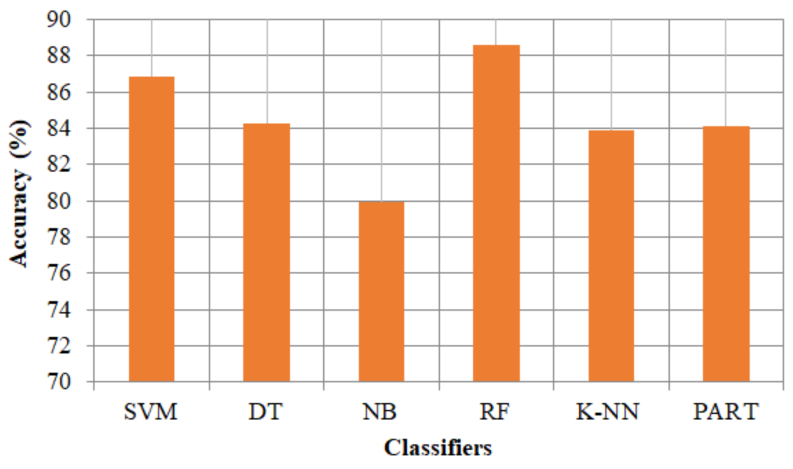


Figure 3.15: Comparative analysis of various classifiers using dynamic approach based on accuracy for multi-class classification dataset (Dataset-2)

Figure 3.15 compares the accuracy of different ML classifiers for multi-class classification dataset. The accuracy attained by RF for Dataset-2 is 88.60% which is smaller than the accuracy acquired by RF for Dataset-1.

3.2.3 Results of Classification Using Integrated Features

Individual approach either dynamic or static is insufficient for better classifying Android malware due to execution stalling and obfuscation techniques being used by malware authors. The hybrid approach is used to overwhelm this problem. In this approach, both static and dynamic features are integrated. Six ML techniques are employed for the identification and classification of malware using integrated features.

Table 3.7: Comparison of ML techniques based on integrated attributes for detection dataset (Dataset-1)

ML model	Sens	FPR	PPV	F-measure	AUC	MCC	Accuracy (%)
DT	0.970	0.030	0.970	0.970	0.980	0.941	97.03
SVM	0.983	0.017	0.983	0.983	0.983	0.966	98.30
RF	0.985	0.015	0.985	0.985	0.999	0.971	**98.53**
NB	0.956	0.043	0.957	0.956	0.993	0.913	95.60
K-NN	0.982	0.018	0.982	0.982	0.994	0.963	98.16
PART	0.971	0.029	0.971	0.971	0.983	0.942	97.09

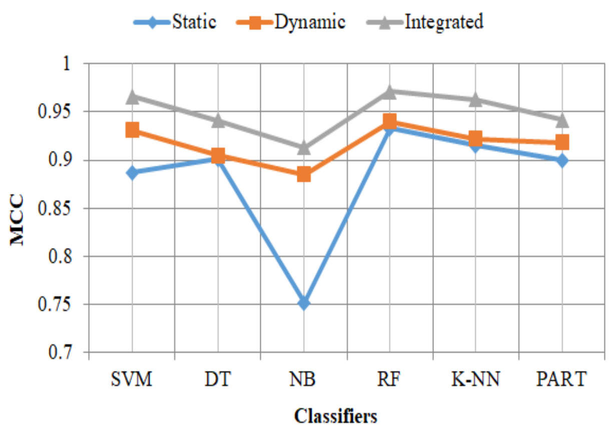
Table 3.7 indicates the results of ML algorithms using integrated attributes for detection dataset. It is observed that RF provides the best detection accuracy of 98.53% followed by K-NN and SVM with the detection accuracy of 98.16% and 98.30% respectively.

Table 3.8: Comparison of ML techniques based on integrated attributes for multi-class classification dataset (Dataset-2)

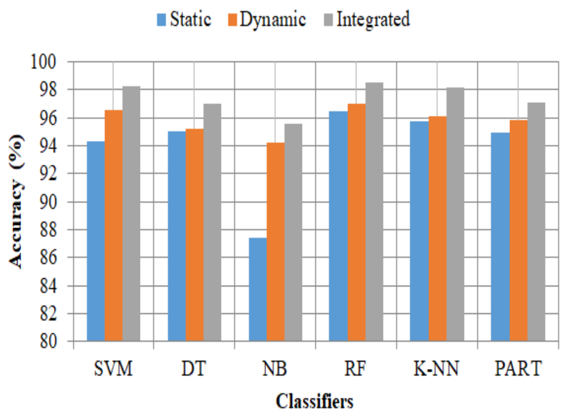
ML model	Sens	FPR	PPV	F-measure	AUC	Accuracy (%)
DT	0.846	0.024	0.851	0.845	0.949	84.60
SVM	0.870	0.020	0.875	0.871	0.987	87.06
RF	0.901	0.016	0.902	0.901	0.995	**90.10**
NB	0.783	0.027	0.814	0.784	0.970	78.30
K-NN	0.854	0.022	0.857	0.854	0.966	85.40
PART	0.833	0.024	0.837	0.833	0.946	83.34

Table 3.8 indicates the results of ML algorithms based on integrated attributes for multi-class classification dataset. It is observed that RF provides the best classification accuracy of 90.10% followed by SVM which provides classification accuracy of 87.06%. The Sens, F-

measure and PPV acquired by RF are 0.901, 0.901 and 0.902 respectively which is higher than other ML classifiers.



(a)



(b)

Figure 3.16: Comparison of ML classifiers for all three approaches based on (a) MCC (b) Accuracy for detection dataset (Dataset-1)

Figure 3.16 illustrates the MCC and accuracy comparison of six classifiers for detection dataset using static, dynamic and integrated features. It shows that there is an improvement in the MCC and accuracy for all the classifiers when both static and dynamic attributes are combined. It indicates that using both types of attributes aids in the better identification of malware.

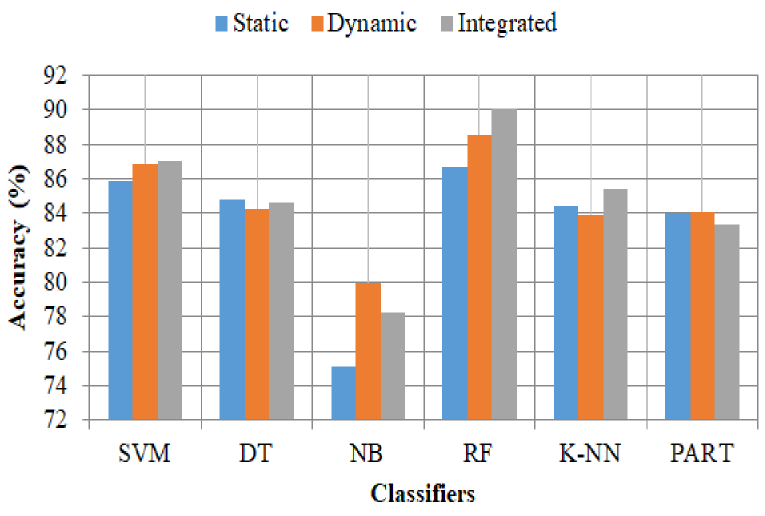


Figure 3.17: Comparison of ML classifiers for all three approaches based on accuracy for multi-class classification dataset (Dataset-2)

Figure 3.17 illustrates the accuracy comparison of six classifiers for multi-class classification dataset using static, dynamic and integrated features. It shows that the integrated approach performs better as compared to static and dynamic approach for all the classifiers except NB. The accuracy attained by multi-class classification dataset is smaller than detection dataset it might be due to imbalanced classes.

Table 3.9: Comparison results of static, dynamic and integrated approach

Dataset	**Approach**	**ML model**	**Sens**	**FPR**	**PPV**	**MCC**	**F-measure**	**Accuracy (%)**
Detection (Dataset-1)	**Static**	**RF**	0.965	0.035	0.965	0.933	0.965	96.50
	Dynamic		0.970	0.030	0.970	0.940	0.970	97.01
	Integrated		0.985	0.015	0.985	0.971	0.985	**98.53**
Multi-class Classification (Dataset-2)	**Static**	**RF**	0.867	0.024	0.870	**--**	0.866	86.72
	Dynamic		0.886	0.018	0.888	**--**	0.885	88.60
	Integrated		0.901	0.016	0.902	**--**	0.901	**90.10**

*MCC -- not applicable to datasets with multiple classes

Table 3.9 shows the comparison of all three approaches for the best classifier i.e. RF for both the datasets. The results show that the integrated approach performs better for malware identification and classification for both the datasets. The accuracy obtained by RF is 98.53% and 90.10% for detection and multi-class classification dataset respectively.

3.3 DISCUSSIONS

The proposed approach makes use of integrated set of features which are obtained after combining static and dynamic features. A total of 352 static and 323 dynamic attributes are mined from Android samples. To get rid of noisy and unnecessary attributes, the IG feature selection method is used. Through this technique, 110 static and 99 dynamic attributes are selected for Dataset-1 and 47 static attributes and 35 dynamic attributes are selected for Dataset-2. To detect and identify Android malware, various classifiers are used. The results demonstrate that the integrated approach performs well as compared to when static and dynamic attributes are examined alone. In the case of static attributes, RF gives better detection and classification accuracy i.e. 96.5% and 86.72% for detection and multi-class classification dataset respectively. In the case of dynamic attributes, RF gives better detection and classification accuracy i.e. 97.01% and 88.6% for both datasets. RF offers the maximum detection and classification accuracy in the integrated approach for both datasets its value is 98.53% and 90.1% for detection and multi-class classification dataset respectively. From the experimental results, it is found that the classification results in case of static, dynamic and integrated features are not so good as compared to detection results. It might be due to the imbalanced classes in the classification dataset.

3.4 SUMMARY

This chapter presented an integrated approach for identification and classification of Android malware. The two datasets (i.e. Dataset-1 and Dataset-2) are created for detection and multi-class classification of malware. These datasets have been made public on kaggle and GitHub in order to aid anti-malware tool developers and researchers in improving or developing new methodologies and tools for identifying and classifying malware. These datasets can be used as benchmark datasets by various researchers to validate their proposed techniques. Various classifiers are used to detect and classify malware based static, dynamic and integrated approaches. The results demonstrated that the integrated approach performs better than individual approaches as it overcomes the constraints of both static and dynamic malware analysis. Chapter 4 and chapter 5 present the proposed techniques to improve the Android malware detection and classification results respectively.

CHAPTER 4

PROPOSED APPROACH FOR IMPROVING DETECTION OF ANDROID MALWARE

As discussed in chapter 3, malware developers create new malware to threaten the security of the system and privacy of users. The security of mobile devices has motivated researchers in employing ML techniques to improve the detection of Android malware as the conventional approaches are not effective in recognizing unknown malware. ML-based approaches are increasingly being used to detect malware on Android devices. The main problem with the existing malware detection systems is that they have a high FP and FN rate. Thus, there is a need to design methods for better identification and classification of malware. This chapter presents an approach named as MalDetect for enhancing the detection results of Android malware. The approach fuses the base classifiers on the basis of proposed ranking schemes defined on their error rate. These schemes are then used to generate a variety of combinations, with the best one being chosen to construct the final model. The proposed approach is evaluated on two datasets i.e. Drebin (benchmark) and AndroMD (self-created).

4.1 PROPOSED METHODOLOGY

The proposed classifier fusion approach for the identification of Android malware is described in this section. Its architecture consists of 2-layers. It is developed in such a way that it can be employed to both ensemble and traditional classifiers. In layer-1, after acquiring both the datasets, six base classifiers are trained using 5-fold cross-validation technique to find the error rates. In layer-2, it uses various ranking schemes defined based on the predictive error rate of base classifiers. The ranking schemes are then used to derive various combination schemes out of which the best combination is selected to build the final model. The architecture of MalDetect is shown in Figure 4.1.

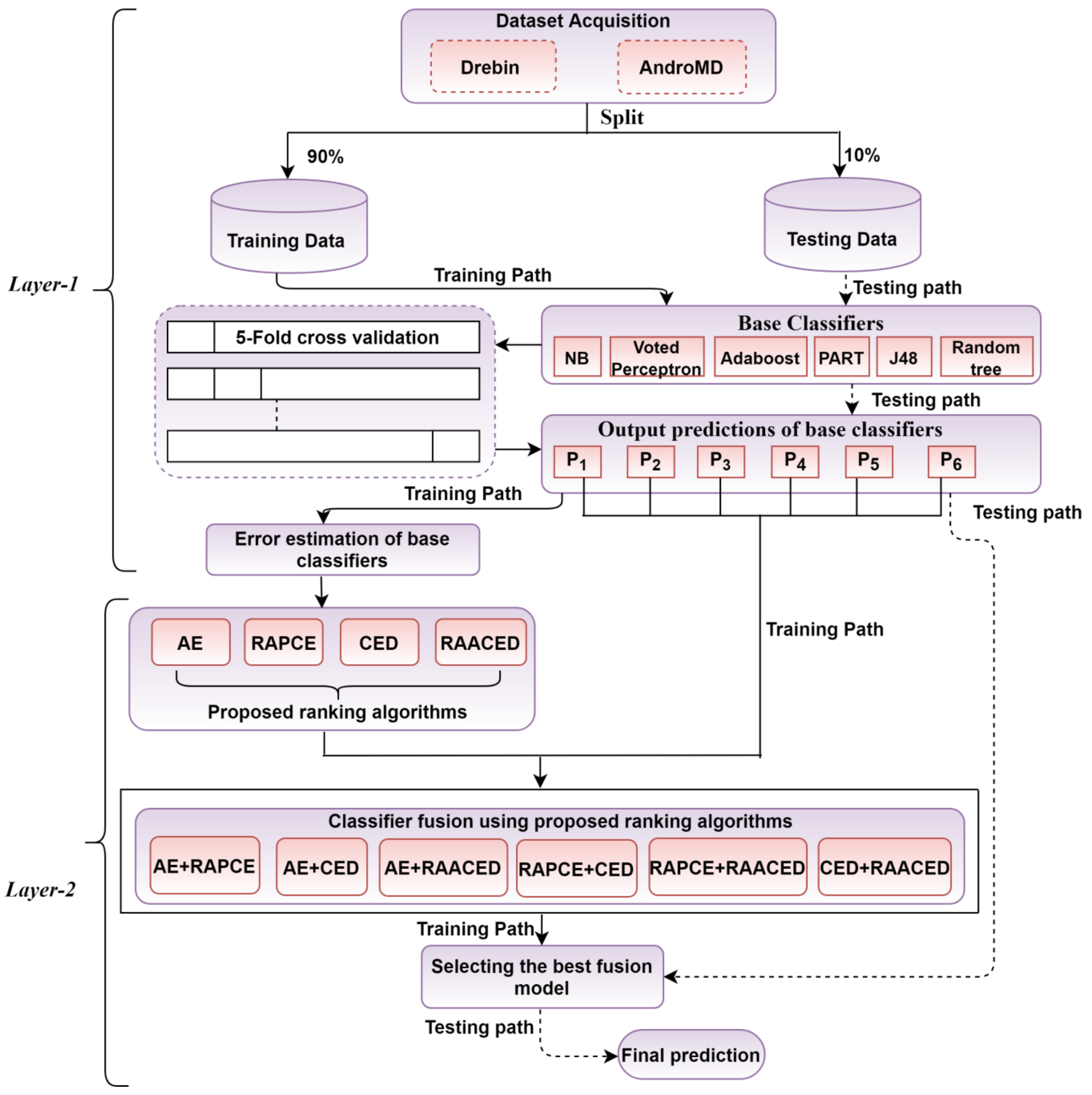


Figure 4.1: Proposed approach for improving the detection of malware

4.1.1 Data Acquisition

Two benchmark datasets i.e. Drebin [87] and AndroMD used in this work are acquired from figshare and Kaggle respectively. Drebin dataset consists of only static features whereas AndroMD dataset consists of both static and dynamic features. Drebin dataset contains 15,036 instances out of which 9,476 are benign and 5,560 are malware. It contains 215 static features. AndroMD dataset contains 3,547 instances out of which 1,800 are benign and 1,747 are malware (as discussed in chapter 3). It contains 352 static and 323 dynamic features. Table 4.1 describes the summary of both datasets.

Table 4.1: Summary of datasets used (where # represents the number of)

Dataset	#Instances	#Malware	#Benign	#Attributes
Drebin	15,036	5,560	9,476	215
AndroMD	3,547	1,747	1,800	675

4.1.2 Data Splitting

Both datasets are randomly divided into two parts i.e. training and testing. 90% of the data is used for training purposes, while 10% is used for testing purposes.

4.1.3 Classification Algorithms

The six different base classifiers are used in this study such as NB, Random Tree, PART, J48, AdaBoost and Voted perceptron. The description of these classifiers are as follows:

- **NB-** It is a type of ML technique that is used to solve classification problems [196]. It is based on the Bayes theorem as discussed in chapter 3.
- **Voted Perceptron-** Frank Rosenblatt's perceptron algorithm is used to construct the voted perceptron. This algorithm takes advantage of data that can be linearly separated by a wide margin. This method is easier to implement and is more effective in terms of computation time [199].
- **AdaBoost-** Adaptive Boosting, also known as AdaBoost, is a well-known boosting technique. Its main purpose is to create strong classifiers by combining several weak classifiers [200]. The pseudocode for AdaBoost algorithm is shown in Algorithm 4.1. This algorithm takes a training set (p_1, q_1),…,(p_m, q_m) where each $p_i(i = 1, 2, \dots, m)$ belongs to some domain P and each label $q_i(i = 1, 2, \dots, m)$ belongs to Q = {-1,1}. A distribution of weights Y_j is set over the training sample at each iteration j, and a weak classifier is created on the training set according to Y_j. The algorithm begins by assigning all weights to the same value, but at each round, the weights of misclassified instances are increased, forcing the weak learner to focus on the most difficult cases to categorize. After a set number of iterations, the procedure ends. In proportion to their accuracy, all of the weak classifiers contribute to the prediction of new unlabelled cases.

Algorithm 4.1: Pseudocode for AdaBoost algorithm

```
    Given: $(p_1, q_1)$,…,$(p_m, q_m)$ $p_i \in P$, $q_i \in Q$ ={-1,1}
01: Initialize $Y_1\ (i) = \frac{1}{m}$
02: for j =1 to J:
03: Train weak classifier using distribution $Y_j$
04: Get weak hypothesis $h_j : P \rightarrow \{-1,1\}$ with error $\in_j = \sum_{i:h_j(p_i)\neq q_i} Y_j\ (p_i)$
05: Choose $\alpha_j = \frac{1}{2}\log(\frac{1-\in_j}{\in_j})$
06: Update:
    $Y_{j+1}(i) = \frac{Y_j(i)}{Z_j}$ $\{e^{-\alpha_j}$ if instance $i$ is classified correctly, $e^{\alpha_j}$ if instance $i$ is not classified correctly
    where $Z_j$ is a normalization factor ($\sum_{i=1}^{m} Y_{j+1} = 1$)
    Output: Final hypothesis: $H(p) = sign(\sum_{j=1}^{J} \alpha_j\, h_j(p))$
```

- **PART-** The divide and conquer principle is used in this algorithm [197]. It is discussed in chapter 3.
- **J48-**The J48 algorithm is also called as C4.5 algorithm [193]. It is discussed in chapter 3.
- **Random Tree-** It works like a decision tree (as discussed in chapter 3) with the exception that it chooses random attributes for each split [201].

4.1.4 5-Fold Cross Validation

5-fold cross validation is used in the training phase, that splits the training data into five subsets and the hold-out approach is repeated five times. One subset is used for testing every time, and the rest four subsets are used for training purposes.

Using 5-fold cross-validation method, the classifier's error rate and performance prediction probabilities for both classes are calculated. Then, based on the classifier error rate the ranks are assigned to the classifiers. The ranks are allocated on the basis of proposed ranking algorithms that are discussed in the subsequent sub-section.

4.1.5 Proposed Ranking Schemes

Four ranking methods are proposed for allocating rank to the base classifiers. These are discussed as follows:

1. **Average Error (AE) Based Ranking Method:** In this method, the ranks are allocated to the base classifiers based on average error prediction of both the classes. It assigns a higher rank to those classifiers which have smaller average error rate as shown in Algorithm 4.2. Let $x_1, x_2, \dots, x_c$ be the c base classifiers being used to classify the instances into benign and malware class. If E_{m,x_i} and E_{b,x_i} represent the error rates of malware and benign class for a classifier x_i respectively, the average error rate E_{x_i} of each classifier is computed as

$$E_{x_i} = \frac{E_{m,x_i} * z_m + E_{b,x_i} * z_b}{z_m + z_b} \; where \; i \in \{1,2,\dots,c\} \tag{4.1}$$

Here z_b and z_m denote the number of benign instances and malware instances respectively. Let $E = \{E_{x_1}, E_{x_2}, \dots, E_{x_c}\}$ be the set of average error predictions for all the classifiers, then the rank $\bar{E}$, defined in Equation 4.2, is assigned using $Rank_{ascen}(\,)$ function on the basis of average error prediction for both the classes. It assigns a higher rank to those classifiers which has smallest average error rate.

$$\bar{E} = Rank_{ascen}(E) \tag{4.2}$$

Algorithm 4.2: Algorithm of AE Based Ranking Method

Input: Number of Base classifiers (c), number of malware instances (z_m), number of benign instances (z_b), error rate of malware (E_{m,x_i}) and error rate of benign (E_{b,x_i})

Output: AE based Rank ($\bar{E}$)

```
01: for i=1:c
02:     $E_{x_i} = \frac{E_{m,x_i} * z_m + E_{b,x_i} * z_b}{z_m + z_b}$          # $E_{x_i}$ is the average error rate of ith classifier
03: end for
04: $\bar{E}$ = Rank_ascen(E)          # Here, $E = \{E_{x_1}, E_{x_2}, \dots, E_{x_c}\}$
05: Return ($\bar{E}$)
```

2. **Ranked Aggregate of Per Class Error (RAPCE) Method:** In this method, firstly the ranks are allocated to each classifier on the basis of class error rate and then the final ranks are computed by adding the per class ranking as shown in Algorithm 4.3. Let $M = \{M_{x_1}, M_{x_2}, \dots, M_{x_c}\}$ and $B = \{B_{x_1}, B_{x_2}, \dots, B_{x_c}\}$ are the set of error rates of all

base classifiers for malware and benign class respectively. The rank to each classifier is allocated individually based on the class error rate using $Rank_{ascen}(\,)$ function (i.e. smaller the value of M_{x_i} or B_{x_i}, higher is the rank) as represented in Equations 4.3 and 4.4.

$$Rank\ (x_i, m) = Rank_{ascen}(M) \tag{4.3}$$

$$Rank\ (x_i, b) = Rank_{ascen}(B) \tag{4.4}$$

The aggregate of per class rank S_{x_i} of each classifier is computed using Equation 4.5.

$$S_{x_i} = Rank\ (x_i, m) + Rank\ (x_i, b)\ where\ i \in \{1, 2, \dots, c\} \tag{4.5}$$

Let $S = \{S_{x_1}, S_{x_2}, \dots, S_{x_c}\}$ be the set of values of aggregate rank for all base classifiers. The final rank is allocated to each classifiers using $Rank_{desc}(\,)$function on the basis of aggregating the per class rank as represented in Equation 4.6. It assigns a higher rank to those classifiers which have higher aggregated rank.

$$\bar{S} = Rank_{desc}(S) \tag{4.6}$$

Algorithm 4.3: Algorithm of RAPCE based ranking method

Input: Number of Base classifiers (*c*), error rates of all base classifiers for malware (*M*) and benign (*B*).

Output: RAPCE based Rank ($\bar{S}$)

```
01: for i=1:c
02:     $Rank\ (x_i, m) = Rank_{ascen}(M)$
03:     $Rank\ (x_i, b) = Rank_{ascen}(B)$
04: end for
05: for i=1:c
06:     $S_{x_i} = Rank\ (x_i, m) + Rank\ (x_i, b)$          # $S_{x_i}$ is the aggregate per class rank of $i^{th}$ classifier
07: end for
08: $\bar{S} = Rank_{desc}(S)$                               # Here, $S = \{S_{x_1}, S_{x_2}, \dots, S_{x_c}\}$
09: Return ($\bar{S}$)
```

3. **Class Error Differential (CED) Method:** In this method, the ranks are allocated to each base classifier based on the average error rate and the absolute difference between the class errors as shown in Algorithm 4.4. Let Y_{x_i} is the ratio of average error rate (E_{x_i}) and the error difference of both classes ($|E_{m,x_i} - E_{b,x_i}|$) for each classifiers. Y_{x_i} is computed as

$$Y_{x_i} = \frac{E_{x_i}}{|E_{m,x_i} - E_{b,x_i}|} \; where \; i \in \{1,2,\dots,c\} \tag{4.7}$$

If $Y = \{Y_{x_1}, Y_{x_2}, \dots, Y_{x_c}\}$ is the set of class error differentials for all the classifiers, then the rank is assigned using $Rank_{ascen}(\,)$ function. It assigns a higher rank to those classifiers which have smallest class error differential value.

$$\bar{Y} = Rank_{ascen}(Y) \tag{4.8}$$

Algorithm 4.4: Algorithm of CED based ranking method

Input: Number of Base classifiers (c), average error rate (E_{x_i}), error rate of malware (E_{m,x_i}) and error rate of benign (E_{b,x_i})

Output: CED based Rank ($\bar{Y}$)

```
01: for i=1:c
02:     $Y_{x_i} = \frac{E_{x_i}}{|E_{m,x_i} - E_{b,x_i}|}$   # $Y_{x_i}$ is the ratio of average error rate and the error difference of both classes of $i^{th}$ classifier
03: end for
04: $\bar{Y} = Rank_{ascen}(Y)$          # Here, $Y = \{Y_{x_1}, Y_{x_2}, \dots, Y_{x_c}\}$
05: Return ($\bar{Y}$)
```

4. **Ranked Aggregate Average and Class Error Differential (RAACED) Method:** In this method, the final ranks of the base classifiers are computed on the basis of values obtained after aggregating the ranks obtained from average error rate and class error differential methods as shown in Algorithm 4.5. The ranked aggregate average and class error differential T_{x_i} of each classifier is computed by using Equation 4.9.

$$T_{x_i} = Rank_{ascen}(E) + Rank_{ascen}(Y) \tag{4.9}$$

If $T = \{T_{x_1}, T_{x_2}, \dots, T_{x_c}\}$ is the set of values obtained after aggregating the ranks obtained using average error rate and class error differential methods, then the final rank is allocated using $Rank_{desc}(\,)$ function as shown in Equation 4.10. It assigns highest rank to those classifiers which have higher aggregated rank.

$$\bar{T} = Rank_{desc}(T) \tag{4.10}$$

Algorithm 4.5: Algorithm of RAACED based ranking method

Input: Number of Base classifiers (c), AE based Rank ($\bar{E}$), CED based Rank ($\bar{Y}$)
Output: RAACED based Rank ($\bar{T}$)

```
01: for i=1:c
02:     $T_{x_i} = \bar{E} + \bar{Y}$                # $T_{x_i}$ is the aggregate rank of ith classifiers
03: end for
04: $\bar{T} = Rank_{desc}(T)$              # Here, $T = \{T_{x_1}, T_{x_2}, \dots, T_{x_c}\}$
05: Return ($\bar{T}$)
```

4.1.6 Classifier Fusion using Proposed Ranking Algorithms

The proposed ranking algorithms are used to fuse the classifiers by considering their pairwise combinations. All the training instances are re-classified by using the output prediction of classifiers and the pairwise combination of the proposed ranking algorithms. Every ranking scheme draws a set of D ranks that is employed with output prediction of classifier for each instance during the process of reclassification. Let the set of four ranking scheme is denoted by $R = \{R_1, R_2, R_3, R_4\}$. The pairwise combinations of element of R results in six possibilities i.e. $\Phi = \{ R_1 R_2, R_1 R_3, R_1 R_4, R_2 R_3, R_2 R_4, R_3 R_4\}$.

To identify the performance of every pairwise combination, assume $w_n, n \in \{1, 2, \dots, D\}\ where\ D \le c$ as the ranks acquired from the first ranking in the pair and $u_n, n \in \{1, 2, \dots, D\}\ where\ D \le c$ as the ranks acquired from the second ranking in the pair. If q_n is the output prediction of each instance using base classifier, then the class prediction $H_{Ri,Rj}(z)$ of every instance z is calculated as shown in Equation 4.11.

$$H_{Ri,Rj}(z) = \begin{cases} 1: \quad if \ \frac{\sum_{n=1}^{D} w_n q_n + \sum_{n=1}^{D} u_n q_n}{\sum_{n=1}^{D} w_n + \sum_{n=1}^{D} u_n} \geq 0.5 \\ 0: \quad otherwise \\ \forall\, i \in \{1,2,3,4\}, \ \forall\, j \in \{1,2,3,4\} \\ where\ i \neq j,\ R_i R_j \ \equiv \ R_j R_i \end{cases} \tag{4.11}$$

Here, 1 and 0 represent the malware and benign class respectively. $R_i R_j$ represent the pairwise ranking combination of all four ranking schemes.

The detection rate of malicious class ($P_{Ri,Rj}^{malware}$) is calculated as given in Equation 4.12.

$$P_{Ri,Rj}^{malware} = \frac{\sum_{z=1}^{Z} H_{Ri,Rj}(z) | H_{Ri,Rj}(z)=1, l(z)=1}{m} \tag{4.12}$$

Here $l(z)$ represents the class label of each instance. The detection rate of benign class ($P_{Ri,Rj}^{benign}$) is calculated as shown in Equation 4.13.

$$P_{Ri,Rj}^{benign} = \frac{\sum_{z=1}^{Z} (H_{Ri,Rj}(z)+1) | H_{Ri,Rj}(z)=0, l(z)=0}{b} \tag{4.13}$$

Then calculate the average detection rate for the pairwise ranking scheme using Equation 4.14.

$$P_{Ri,Rj} = \frac{m \,.\, P_{Ri,Rj}^{malware} + b \,.\, P_{Ri,Rj}^{benign}}{Z} \tag{4.14}$$

Similarly, the value of average precision is computed. After computing the average pairwise combination detection rate of proposed ranking schemes on training data, the best fusion model is selected and evaluated on test data set.

4.2 EXPERIMENTAL RESULTS

This section presents the results of two datasets to evaluate the performance of the proposed approach. To implement and test the proposed approach, an open-source software called Waikato Environment for Knowledge Analysis (WEKA) [44] is used. Both datasets are divided into two parts, one for training and the other for testing. The ratio of training and the testing portion is 90 % and 10% respectively. 5-fold cross-validation is used to build a fusion model using the training data. Afterward, it is evaluated on test data.

4.2.1 Experimental Results for Drebin Dataset

Six base classifiers i.e. voted perceptron, J48, Adaboost, NB, PART and Random tree using 5-fold cross-validation method for the training data. Table 4.2 demonstrates the classification results.

Table 4.2: Results of base classifiers on the basis of different parameters on Drebin training data

Classifiers	FNR	FPR	Recall$_m$	Precision$_m$	Recall$_b$	Precision$_b$	W-FM
NB	0.059	0.229	0.941	0.707	0.771	0.957	0.8489
Voted perceptron	0.046	0.017	0.954	0.971	0.983	0.973	0.9723
Adaboost	0.098	0.063	0.902	0.894	0.937	0.942	0.9242
PART	0.027	0.012	0.973	0.979	0.988	0.984	0.9823
J48	0.043	0.019	0.957	0.967	0.981	0.975	0.9721
Random tree	0.042	0.027	0.958	0.953	0.973	0.975	0.9672

Table 4.2 shows that PART outperforms other base classifiers. It provides a precision of 0.979 and 0.984 and a recall of 0.973 and 0.988 for malware and benign class respectively.

The comparison of ML classifiers based on False Positive Rate (FPR) and False Negative Rate (FNR) is shown in Figure 4.2. The value of both FPR and FNR are minimum for PART i.e. 0.012 and 0.027 respectively. NB provides the maximum value of FPR i.e. 0.229 and Adaboost provides the maximum value of FNR i.e. 0.098. Figure 4.3 shows the comparison of ML classifiers on the basis of W-FM. The value of W-FM is maximum for PART i.e. 0.9823 and minimum for NB i.e. 0.8489.

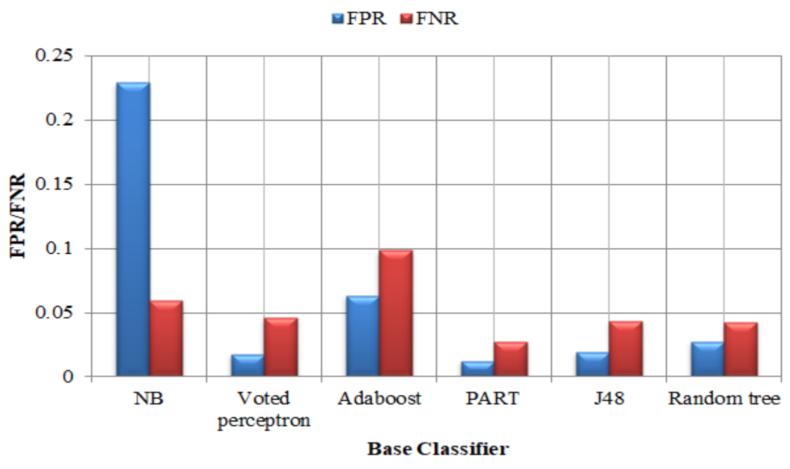


Figure 4.2: Comparison of ML classifiers on the basis of FPR and FNR on Drebin training data

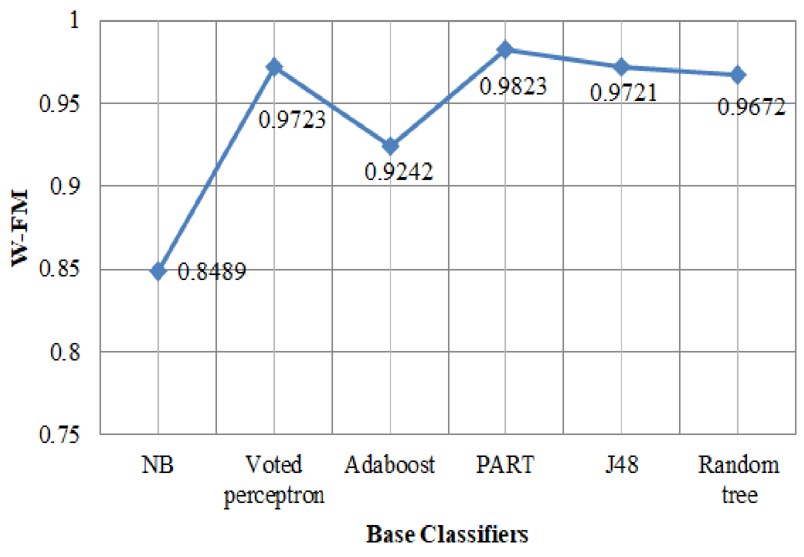


Figure 4.3: Comparison of ML classifiers on the basis of W-FM on Drebin training data

In order to compute the rank for each classifier, the proposed ranking algorithms are employed on the classification results obtained for the training data. The computations are done using FPR (error rate in detecting benign apps) and FNR (error rate in detecting malicious apps). Table 4.3 shows the class error rate and the ranks computed using proposed ranking algorithms i.e. AE, RAPCE, CED and RAACED for each classifier.

Table 4.3: Rank of base classifiers using proposed ranking algorithms on Drebin training data

(Lower Rank=1 and Highest Rank=6)

Classifiers	FNR	FPR	RAPCE	AE	RAACED	CED
NB	0.059	0.229	1	1	3	6
Voted perceptron	0.046	0.017	3	5	6	5
Adaboost	0.098	0.063	1	2	1	1
PART	0.027	0.012	6	6	5	3
J48	0.043	0.019	3	4	4	4
Random tree	0.042	0.027	3	3	2	1

From Table 4.3, it is found that PART gets the highest rank for AE and RAPCE ranking methods. After applying Equation 4.11 to the instances in the training data and then compute the pairwise combination values using Equations 4.12-4.14. The results of fusion on training data are shown in Table 4.4.

Table 4.4: Fusion results on Drebin training data

Combination	$Recall_m$	$Precision_m$	$Recall_b$	$Precision_b$	W-FM
AE+RAPCE	0.979	0.982	0.987	0.985	0.9840
AE+CED	0.973	0.971	0.979	0.980	0.9767
AE+RAACED	0.973	0.979	0.984	0.981	0.9801
RAPCE+CED	0.972	0.969	0.979	0.978	0.9751
RAPCE+RAACED	0.973	0.980	0.984	0.980	0.9800
CED+RAACED	0.972	0.967	0.974	0.977	0.9733

The results of Table 4.4 demonstrate the performance improvement in layer-2 combination schemes. The best combination is found to be AE+RAPCE on the training data. It provides a precision of 0.982 and 0.985 and a recall of 0.979 and 0.987 for malware and benign class respectively. The W-FM of AE+RAPCE combination is 0.9840.

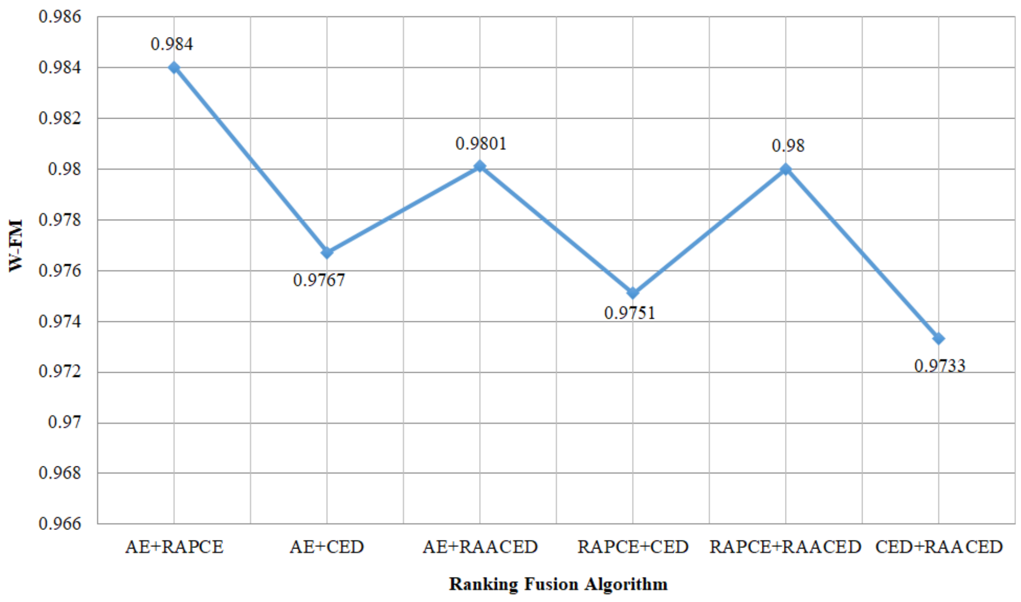


Figure 4.4: Comparison of fusion of ranking algorithms based on W-FM on Drebin training data

Figure 4.4 illustrates the comparison of combinations of ranking algorithms based on W-FM. It shows that the combination of AE+RAPCE performs better than other combinations of ranking algorithms. The W-FM acquired by AE+RAPCE is 0.9840.

From the training data results, it is observed that the combination of AE+RAPCE performs best on training data. This model is further used to evaluate the performance on the test data.

Table 4.5 demonstrates the comparison of the proposed technique with the conventional combination techniques and the ML classifiers on the test data.

Table 4.5: Comparison of proposed technique with ML classifiers and conventional combination techniques on Drebin test data

Classifiers	Recall$_m$	Precision$_m$	Recall$_b$	Precision$_b$	W-FM
NB	0.941	0.730	0.796	0.958	0.8615
Voted perceptron	0.955	0.962	0.978	0.974	0.9695
Adaboost	0.885	0.893	0.938	0.933	0.9183
PART	0.962	0.978	0.987	0.978	0.9779
J48	0.960	0.966	0.980	0.977	0.9728
Random tree	0.959	0.960	0.977	0.976	0.9702
Majority voting	0.966	0.978	0.987	0.980	0.9792
Average probabilities	0.966	0.978	0.987	0.980	0.9792
Maximum probabilities	0.986	0.934	0.959	0.991	0.9695
Multischeme	0.955	0.948	0.969	0.974	0.9641
Proposed approach	**0.987**	**0.975**	**0.985**	**0.992**	**0.9857**

From Table 4.5, it is concluded that the proposed approach performs best among all base classifiers and the conventional combination techniques (such as majority voting, average probabilities, multischeme and maximum probabilities).

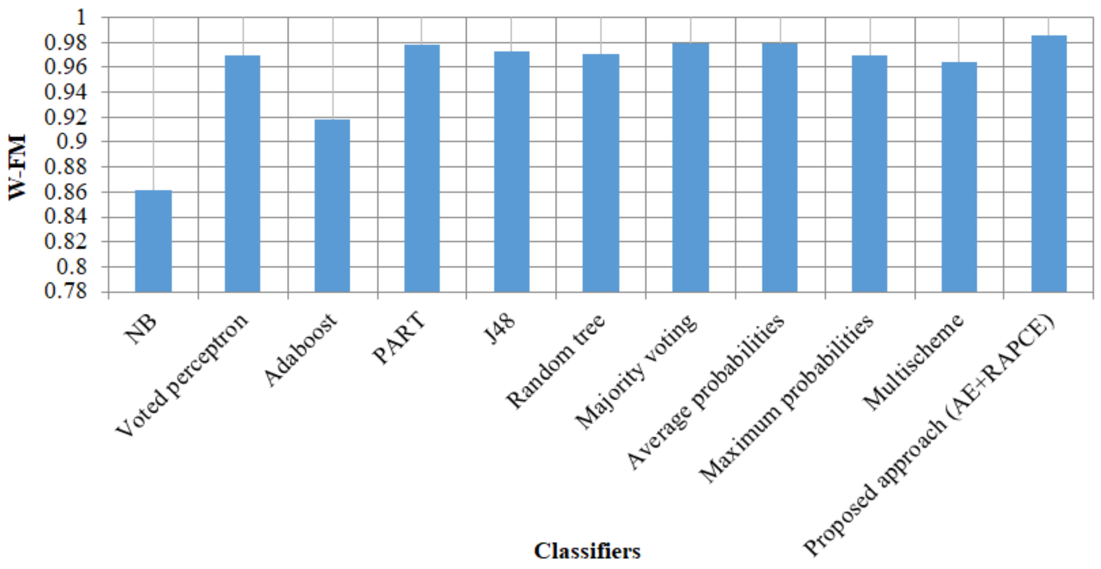


Figure 4.5: Comparison of various techniques on the basis of W-FM on Drebin test data

Figure 4.5 demonstrates the comparison analysis of different techniques based on W-FM. The findings suggest that the proposed approach is more superior than other traditional methods. It obtains the highest value of W-FM i.e. 0.9857 followed by majority voting and average probabilities which achieve a W-FM value of 0.9792.

4.2.2 Experimental Results for AndroMD Dataset

The proposed approach is also evaluated on AndroMD dataset using 5-fold cross-validation method for the training data. The classification results are shown in Table 4.6.

Table 4.6: Results of base classifiers on the basis of different parameters on AndroMD training data

Classifiers	FNR	FPR	Recall_m	Precision_m	Recall_b	Precision_b	W-FM
NB	0.029	0.063	0.971	0.937	0.937	0.971	0.9540
Voted perceptron	0.022	0.028	0.978	0.971	0.972	0.978	0.9748
Adaboost	0.036	0.059	0.964	0.940	0.941	0.965	0.9525
PART	0.025	0.028	0.975	0.971	0.972	0.976	0.9735
J48	0.025	0.030	0.975	0.970	0.970	0.976	0.9728
Random tree	0.092	0.089	0.908	0.908	0.911	0.911	0.9095

From Table 4.6, the results demonstrate that voted perceptron performs better than other base classifiers. It provides a precision of 0.971 and 0.978 and a recall of 0.978 and 0.972 for malware and benign class respectively.

Figure 4.6 shows the comparison of ML classifiers on the basis of FPR and FNR. The value of both FPR and FNR is minimum for voted perceptron i.e. 0.028 and 0.022 respectively. Random tree provides the maximum value of FPR and FNR i.e. 0.089 and 0.092 respectively. Figure 4.7 shows the comparison of ML classifiers based on W-FM. The value of W-FM is maximum for voted perceptron i.e. 0.9748 and minimum for Random tree i.e. 0.9095.

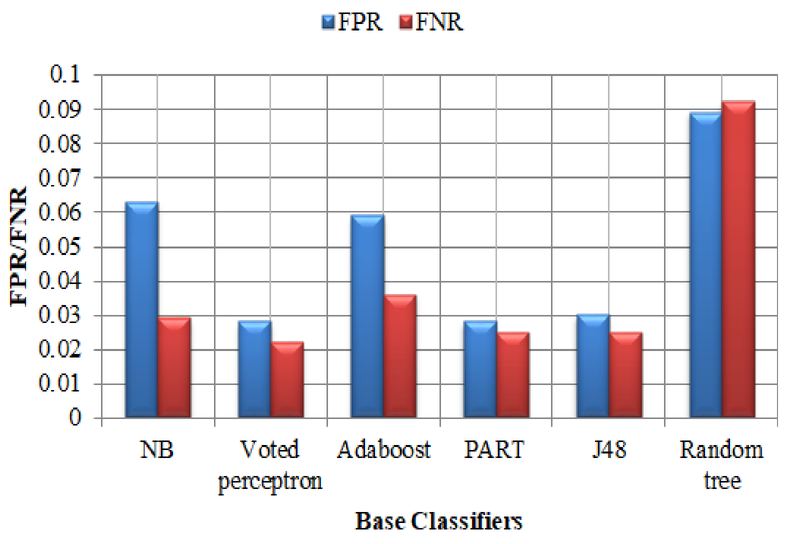


Figure 4.6: Comparison of ML classifiers based on FPR and FNR on AndroMD training data

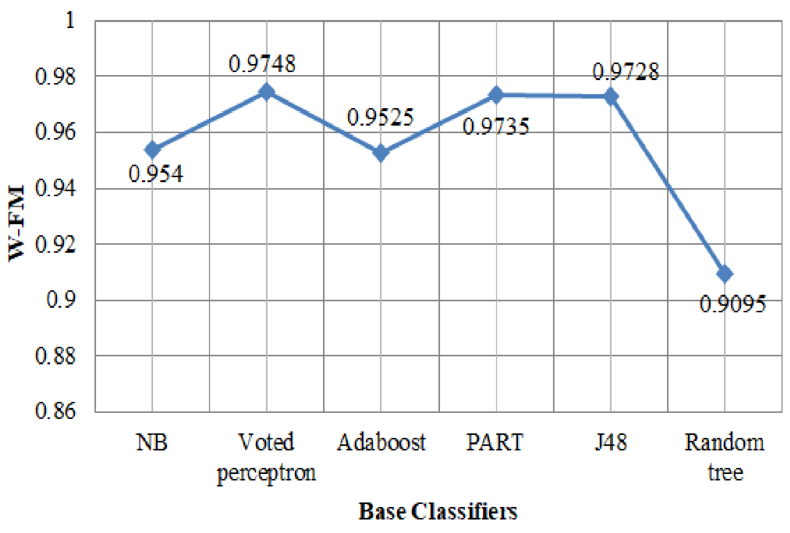


Figure 4.7: Comparison of ML classifiers on the basis of W-FM on AndroMD training data

In order to compute the rank for each classifier, the proposed ranking algorithms are applied on the classification results obtained for the training data. Table 4.7 shows the class error rate and the ranks computed using proposed ranking algorithms i.e. AE, RAPCE, CED and RAACED for each classifier.

Table 4.7: Rank of base classifiers using proposed ranking algorithms on AndroMD training data

(Lower Rank=1 and Highest Rank=6)

Classifiers	FNR	FPR	RAPCE	AE	RAACED	CED
NB	0.029	0.063	2	3	5	6
Voted perceptron	0.022	0.028	6	6	6	4
Adaboost	0.036	0.059	2	2	2	5
PART	0.025	0.028	5	5	2	2
J48	0.025	0.030	4	4	2	3
Random tree	0.092	0.089	1	1	1	1

From Table 4.7, it is found that voted perceptron gets the highest rank for AE, RAPCE and RAACED ranking methods. After applying Equation 4.11 to the instances in the training data and then compute the pairwise combination values using Equations 4.12-4.14. The results of fusion training data are demonstrated in Table 4.8.

Table 4.8: Fusion results on AndroMD training data

Combination	Recall_m	Precision_m	Recall_b	Precision_b	W-FM
AE+RAPCE	0.987	0.985	0.990	0.991	0.9888
AE+CED	0.984	0.982	0.987	0.988	0.9858
AE+RAACED	0.986	0.980	0.988	0.991	0.9871
RAPCE+CED	0.984	0.982	0.987	0.988	0.9858
RAPCE+RAACED	0.986	0.981	0.988	0.991	0.9873
CED+RAACED	0.982	0.978	0.985	0.988	0.9841

The results of Table 4.8 demonstrate the performance improvement in the layer-2 combination schemes. The best combination is found to be AE+RAPCE on the training data. It provides a precision of 0.985 and 0.991 and recall of 0.987 and 0.990 for malware and benign class respectively. The W-FM of AE+RAPCE combination is 0.9888.

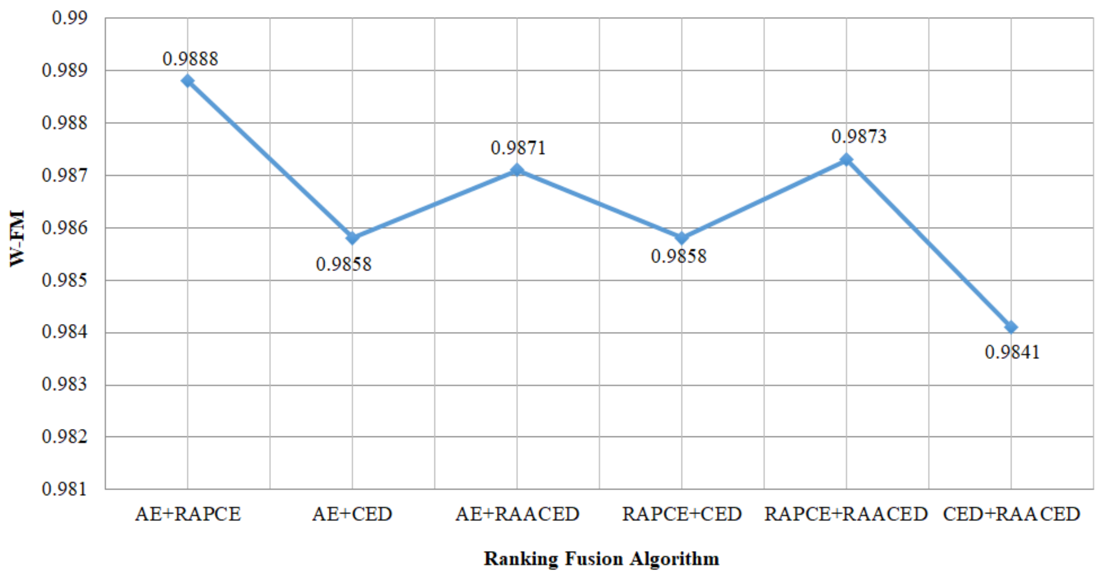


Figure 4.8: Comparison of fusion of ranking algorithms based on W-FM on AndroMD training data

Figure 4.8 shows the comparison of combinations of ranking algorithms based on W-FM. It shows that the combination of AE+RAPCE performs better than other combinations of ranking algorithms. The W-FM acquired by AE+RAPCE is 0.9888.

From the training data results, it is observed that the combination of AE+RAPCE performs best on training data. This model is further used to evaluate the performance of the test data. Table 4.9 shows the comparison of fusion approach with the conventional combination techniques and the ML classifiers on the test data.

Table 4.9: Comparison of proposed technique with ML classifiers and conventional combination techniques on AndroMD test data

Classifiers	Recall$_m$	Precision$_m$	Recall$_b$	Precision$_b$	W-FM
NB	0.903	0.919	0.922	0.907	0.9128
Voted perceptron	0.971	0.955	0.956	0.972	0.9635
Adaboost	0.960	0.960	0.961	0.961	0.9605
PART	0.931	0.982	0.983	0.937	0.9583
J48	0.949	0.988	0.989	0.952	0.9695
Random tree	0.903	0.913	0.917	0.907	0.9100
Majority voting	0.966	0.971	0.972	0.967	0.9690
Average probabilities	0.966	0.971	0.972	0.967	0.9690
Maximum probabilities	0.994	0.906	0.900	0.994	0.9485
Multischeme	0.891	0.923	0.928	0.898	0.9100
Proposed approach	**0.966**	**0.992**	**0.990**	**0.967**	**0.9790**

From Table 4.9, it is concluded that the proposed approach performs best among all base classifiers and the conventional combination techniques (such as majority voting, average probabilities, multischeme and maximum probabilities).

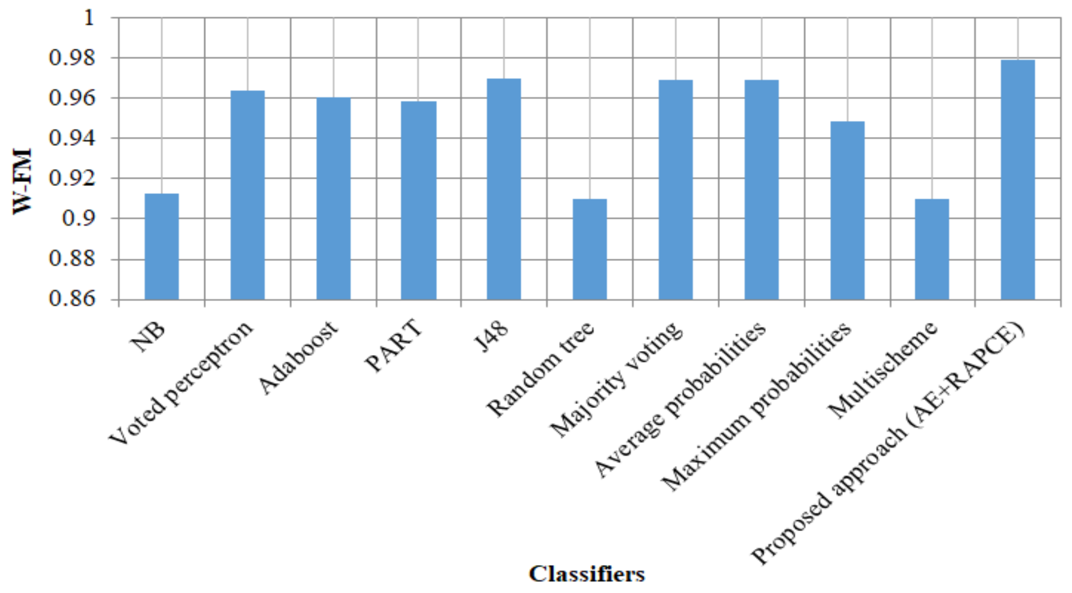


Figure 4.9: Comparison of various techniques on the basis of W-FM on AndroMD test data

Figure 4.9 demonstrates the comparison of different techniques based on W-FM. The results suggest that the proposed approach is more superior to other conventional techniques. It obtains the highest W-FM i.e. 0.9790 followed by majority voting and average probabilities which achieve a W-FM value of 0.9690.

4.2.3 Comparison of Fusion Approach with Stacking Ensemble Method

Stacking ensemble method [202] combines various classifiers via meta-classifiers. It is one of the most common methods for classifiers fusion and employed to various ML algorithms. So we are comparing our proposed approach with the stacked ensemble method. In our proposed approach various ranking algorithms are used to combine the results of ML classifiers. In the stacking ensemble method, the base classifiers are combined with meta logistic regression to detect Android malware. Figure 4.10 demonstrates the general framework of the stacking ensemble learning technique. Table 4.10 shows the comparison results of the proposed approach with the stacking ensemble method on both the dataset.

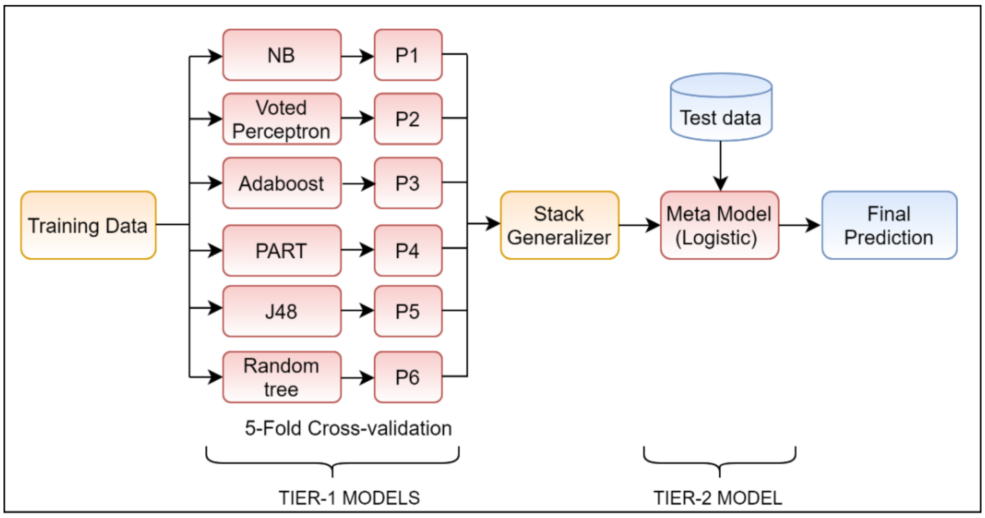


Figure 4.10: Framework of the stacking ensemble learning method

Table 4.10: Comparison of the proposed technique with stacking ensemble method

Datasets	Method	Precision$_m$	Recall$_m$	Precision$_b$	Recall$_b$	W-FM
Drebin dataset	**Stacking**	0.978	0.971	0.983	0.987	0.9811
	Proposed approach	**0.975**	**0.987**	**0.992**	**0.985**	**0.9857**
AndroMD dataset	**Stacking**	0.988	0.966	0.967	0.989	0.9775
	Proposed approach	**0.992**	**0.966**	**0.967**	**0.990**	**0.9790**

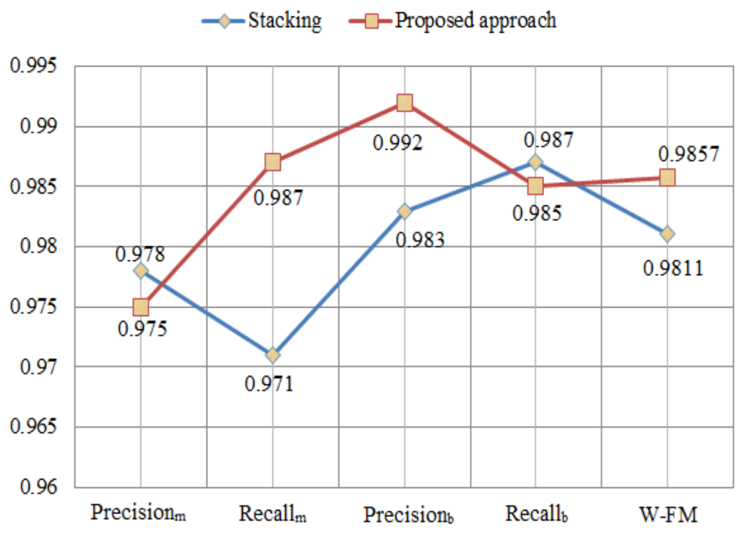


Figure 4.11: Comparison of proposed technique with stacking ensemble technique on Drebin test data

Figure 4.11 shows the comparison of the proposed technique with stacking ensemble technique on Drebin dataset based on W-FM. It is found that the proposed fusion approach performs better than the stacking ensemble learning technique. The W-FM obtained by the proposed approach and stacking approach is 0.9857 and 0.9811 respectively.

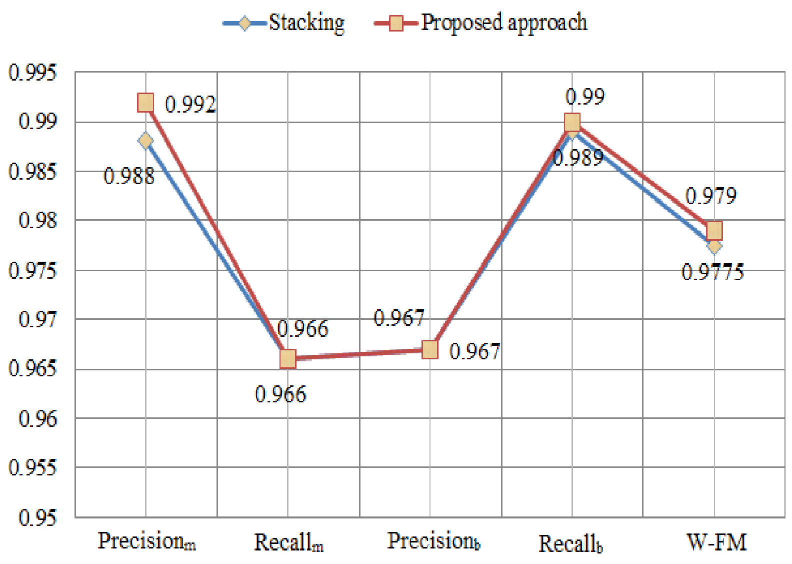


Figure 4.12: Comparison of proposed technique with stacking ensemble technique on AndroMD test data

Figure 4.12 shows the comparison of the proposed technique with stacking ensemble technique on AndroMD dataset based on W-FM. It is found that the fusion approach performs better than the stacking ensemble learning technique. The W-FM obtained by the proposed approach and stacking approach is 0.9790 and 0.9775 respectively.

4.3 DISCUSSIONS

The proposed approach (MalDetect) makes use of four different ranking schemes which are proposed to assign ranks to the base classifiers using training data. The ranking schemes are then used to derive various combinations out of which the best one is selected to build the final model. The findings suggest that MalDetect is more effective than traditional classifiers and ensemble learning techniques. In addition, the comparison of the proposed approach with the stacking ensemble learning technique is also demonstrated. The W-FM obtained from the stacking ensemble technique for both datasets i.e. Drebin and AndroMD dataset is 0.9811 and 0.9775 respectively whereas the W-FM obtained from MalDetect is 0.9857 and 0.9790 respectively.

4.4 SUMMARY

This chapter presented the proposed fusion approach “MalDetect” for detecting Android malware. It fuses the ML algorithms on the basis of proposed ranking schemes for improving the detection of Android malware. The proposed fusion method is tested on a benchmark dataset and self-created dataset. It is compared with the existing techniques to infer its outperformance. This approach is suitable only for binary classification problems for detecting malware. Thus, an approach is designed and presented in chapter 5 for multi-class classification of Android malware.

CHAPTER 5

PROPOSED APPROACH FOR IMBALANCED FAMILY CLASSIFICATION OF MALWARE

In a real-world scenario, the number of samples differs greatly among various malware families which makes classification processing more challenging and has a significant impact on the performance of classifiers. Malware detection, fault detection, fraud detection are some of the examples which are inherently imbalanced [203-205]. The distribution of classes in a dataset is important for constructing effective models. The distribution of classes is almost equal in most circumstances, however this is unattainable in all real-life problems. An imbalance classification problem occurs when one of the classes has a large number of observations (majority class) relative to the other classes, which have a small number of observations (minority class) [206]. The categorization becomes more difficult when a dataset contains imbalanced classes [207]. Thus, there is a need to build malware classification models which can take care of imbalanced classes.

In inductive learning and ML, classification is a critical task [208-210]. Models are trained using a group of training cases that have been labelled with their respective classes [211-213]. Predictive accuracy is used to evaluate the quality and effectiveness of ML approaches, but it is insufficient when the data is excessively skewed. Rather, evaluation metrics like recall, F-measure and precision are used. Many methods like data level, Cost-Sensitive (CS) learning and algorithm level are there to address this problem [134]. This chapter proposes a cost-sensitive learning (CSForest) approach for the imbalanced family categorization of malware.

5.1 PROPOSED METHODOLOGY

This section presents the proposed method (CSForest) for the imbalanced family categorization of malware. The data samples are first obtained from virusshare [187]. To eliminate duplicate samples, these samples are given hash values using the MD5 hash algorithm and examined with the Avira AV software [188] to discover the names of their families. After that, distinct attributes are mined using dynamic and static malware analysis. A feature reduction technique is used to pick appropriate features, after feature extraction. The resulting dataset is fed into CSForest, which predicts the malicious app families. Figure 5.1 shows the workflow of the methodology used for imbalanced classification of malware.

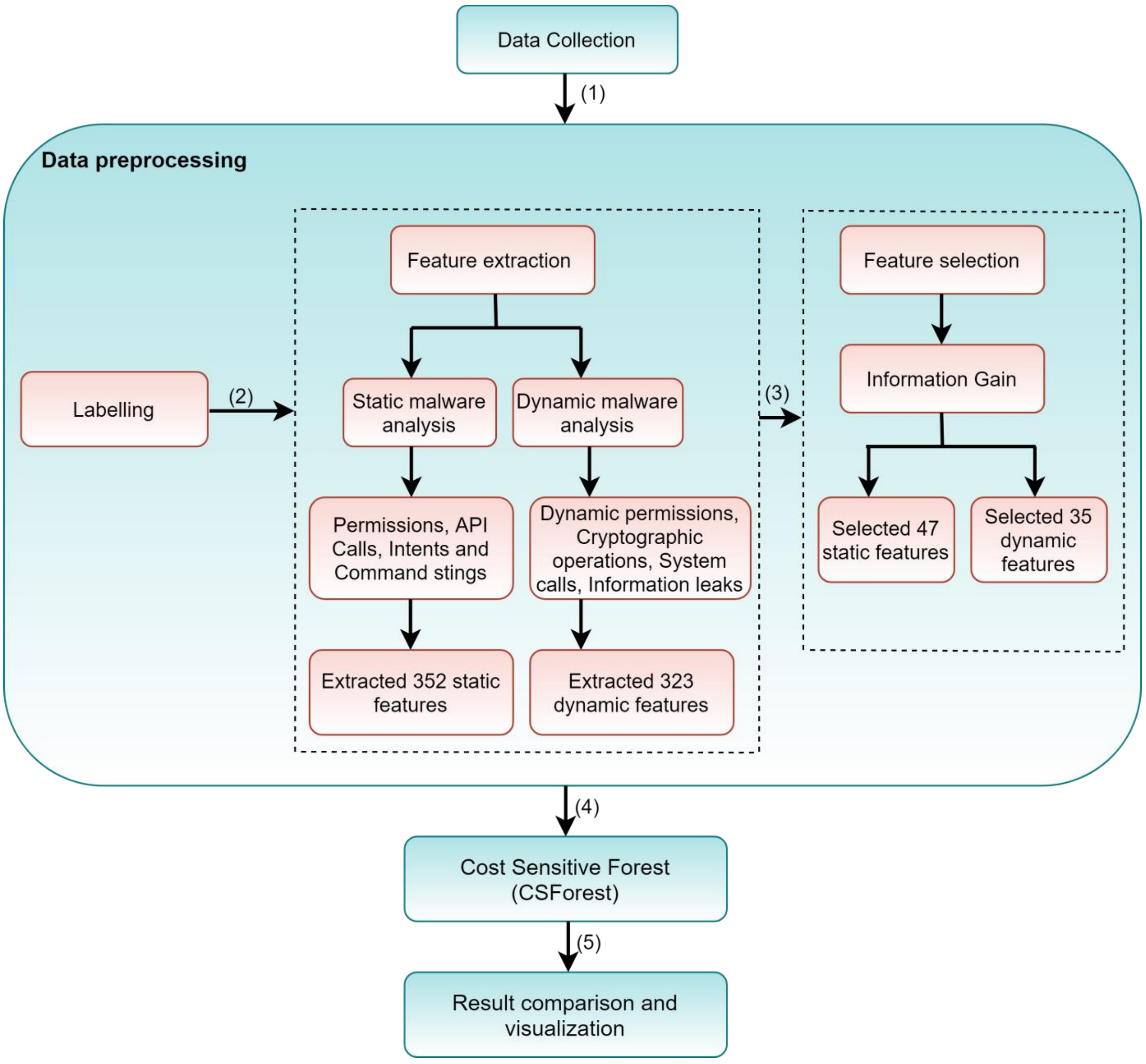

Data Collection
(1)
Data preprocessing
Labelling
(2)
Feature extraction
Static malware analysis
Dynamic malware analysis
Permissions, API Calls, Intents and Command stings
Dynamic permissions, Cryptographic operations, System calls, Information leaks
Extracted 352 static features
Extracted 323 dynamic features
(3)
Feature selection
Information Gain
Selected 47 static features
Selected 35 dynamic features
(4)
Cost Sensitive Forest (CSForest)
(5)
Result comparison and visualization

Figure 5.1: Workflow of the methodology used for imbalanced classification of malware

5.1.1 Data Collection and Data Pre-processing

Data collection and Data pre-processing steps are discussed in chapter 3 under subsection 3.1. In this work, a total of 1,747 apps containing 13 malware families are considered for classifying the families of malware.

5.1.2 Cost-Sensitive Forest

CS algorithms are employed to minimize the classification cost because if an app is dangerous yet appears to be goodware then their consequences can be severe. The expected Classification Cost Reduction (CCR) is used by the CS learning method [214]. The approach initially calculates the total expected cost of the complete dataset using Equation 5.1 before constructing a tree.

$$E = \frac{2\times C_P\times C_N}{C_P+C_N} \tag{5.1}$$

C_N and C_p are the labelling cost of the negative and positive examples. These are calculated as shown in Equations 5.2 and 5.3.

$$C_N = N_{FN}\times C_{FN} + N_{TN}\times C_{TN} \tag{5.2}$$

$$C_P = N_{FP}\times C_{FP} + N_{TP}\times C_{TP} \tag{5.3}$$

$C_{FN}\ \times N_{FN}$ is the product of the cost and number of *FN* predictions and $C_{TN}\ \times N_{TN}$ is the product of the cost and number of *TN* predictions. $C_{FP}\times N_{FP}$ is the product of the cost and number of *FP* predictions. $C_{TP}\ \times N_{TP}$ is the product of the cost and number of *TP* predictions.

The algorithm then computes the ability of every variable to decrease the classification costs. The feature with the maximum CCR is considered as a root node of the tree.

Consider F_i the *i*th attribute and $F_i \in F$ (where *F* is the set of variables used in X_T (dataset)). If F_i is a numerical attribute, the optimal splitting point of F_i is used to divide X_T into two subsets. X_T is partitioned into *m* subsets if F_i is a categorical attribute with *m* distinct values. The expected cost of each variable is computed as shown in Equation 5.4.

$$E_{F_i} = 2\times\sum_{y=1}^{k}\frac{C_P^y\times C_N^y}{C_P^y+C_N^y} \tag{5.4}$$

Here C_P^y and C_N^y represent the labelling cost of observations within y^{th} subset as positive and negative respectively. The CCR of an attribute is determined by $E - E_{F_i} - T_c^i$ where T_c^i is the total test cost for all instances on F_i. CSTree iterates over all possible splitting attributes

$F_i \in F, \forall_i$ and chooses the one that results in the greatest expected cost reduction. The splitting attribute must satisfy $E - E_{F_i} - T_c^i > 0$, otherwise no splitting will take place.

CSForest uses CS pruning [193]. It permits the tree to reach its full potential before being pruned. A modified version of SysFor [215] is used in the CSForest method. It computes CCR as opposed to gain ratio as a splitting criterion. Firstly, it calculates the CCR ability of each attribute $F_i \in F$ where F refers to the set of variables in X_T. After calculating the CCR ability of each attribute, it then selects the good set of attributes (F^n) on the basis of variables whose CCR ability falls inside the goodness threshold (τ)set by the user.

The splitting point S_i and the root variable F_i are used by CSForest to divide the dataset into various subgroups. The feature with the best CCR value is chosen as the test attribute for continuing dividing each sub-dataset. This procedure will be repeated until no further CCR is possible. Finally, CSForest forms the tree comprised of logic rules. Every tree utilizes the pruning confidence factor c and the minimal records x_m in a leaf. As long as $|Y| < T$ and $F^n < T$ where T specifies the total number of trees, the tree T_i with the attribute F_i at the root node is added to the collection of trees Y. If the number of trees formed is less than the number of trees set by the user (i.e. $|Y| < T$), CSForest builds more trees by utilizing the identical approach in Level 1 of the trees built so far, similar to SysFor.

CSForest employs the CSVoting method to classify the new instance P_i. Assume that Y contains n trees and that P_i falls into n leaves $M = \{M_1, M2, \dots, M_n\}$. For each leaf M_j, CSVoting calculates the labelling cost of examples that belong to M_j as Android/AdLoad.A.Gen (C_1) $C_{C_1}^j$ and the labelling cost of examples that belong to Adware/ ANDR.AdMogo.FAN.Gen (C_2) $C_{C_2}^j$ and so forth. It then computes the total Android/AdLoad.A.Gen classification cost for all n leaves as $C_{C_1} = \sum_{j=1}^{n} C_{C_1}^j$ and for ANDR.AdMogo.FAN.Gen $C_{C_2} = \sum_{j=1}^{n} C_{C_2}^j$ and so on. Finally, P_i is classified as Android/AdLoad.A.Gen if $C_{C_1} < C_{C_2}$, otherwise ANDR.AdMogo.FAN.Gen. CSForest algorithm is described in Algorithm 5.1.

Algorithm 5.1: Algorithm for Cost-Sensitive Forest

```
Input:   Dataset ($X_T$), Trees defined by the users ($T$), pruning confidence factor ($c$), goodness threshold
         ($\tau$), minimum number of records in a leaf ($x_m$)
Output:  Set of trees ($Y$), prediction
01:      Calculate the CCR of each attribute.
02:      Sort the attributes according to CCR values in descending order.
03:      Find a set of good attributes ($F^n$) based on goodness threshold ($\tau$)
04:      Pick the best attributes one by one to build number of trees (at level-0)
05:      If the number of trees formed from good set of the attribute is less than userdefined number of
         trees then it generates more trees using alternative good attributes (at level 1) of the trees.
06:      Return the set of trees ($Y$)
07:      Employ CSVoting to classify the new instance.
```

5.2 EXPERIMENTAL RESULTS

The performance of the proposed method for the imbalanced categorization of malware is evaluated in this section. The method is run on an i5 processor with a 64-bit OS and 8 GB RAM, and it uses Python 3.7. The entire dataset is partitioned into five equal sections and tested using a 5-fold cross-validation approach. Four portions are used for training and one part is used for testing throughout each run. Sens, FPR, F-measure and precision are all used to judge the algorithm's performance. These performance metrics are based on four prospects i.e. FN, TN, FP and TP. Here, FN is the most important because if an app is dangerous yet appears to be goodware then consequences can be severe. Thus, a cost matrix is designed in which the weights are assigned based on hyperparameter tuning. The outcomes of cost-sensitive classifiers (CSForest and CSTree) are compared with those of cost-insensitive classifiers (RF and C4.5) using static, dynamic and hybrid features. The parameters values employed in CSForest are c = 0.25, $x_m = 2$, τ = 0.2, $\in$ = 0.3, T = 30 with T_C^i set to be 0 [214].

Table 5.1: Results of malware family classification of cost-sensitive and cost insensitive classification algorithms

Approach	**Evaluation Parameters**	**Sens**	**Precision**	**FPR**	**F-measure**
Static	**RF**	0.867	0.870	0.024	0.866
	C4.5	0.848	0.852	0.023	0.847
	CSTree	0.852	0.858	0.023	0.851
	CSForest	0.890	0.893	0.016	**0.890**
Dynamic	**RF**	0.886	0.888	0.018	0.885
	C4.5	0.843	0.843	0.026	0.841
	CSTree	0.839	0.837	0.026	0.835
	CSForest	0.905	0.911	0.013	**0.907**
Hybrid	**RF**	0.901	0.902	0.016	0.901
	C4.5	0.846	0.851	0.024	0.845
	CSTree	0.852	0.859	0.023	0.851
	CSForest	0.919	0.922	0.011	**0.919**

Table 5.1 illustrates the family classification results of RF, CSTree, C4.5 and CSForest on the basis of various assessment parameters for all three approaches (i.e. static, dynamic and hybrid). It demonstrates that CSTree performs better than C4.5 and CSForest performs better than RF to classify malicious apps. Furthermore, it demonstrates that CSForest outperforms the other methods in all three approaches. Among these approaches, the hybrid approach outperforms the static and dynamic approaches. The Sens, F-measure and precision attained by CSForest in case of hybrid approach are 0.919, 0.919 and 0.922 respectively.

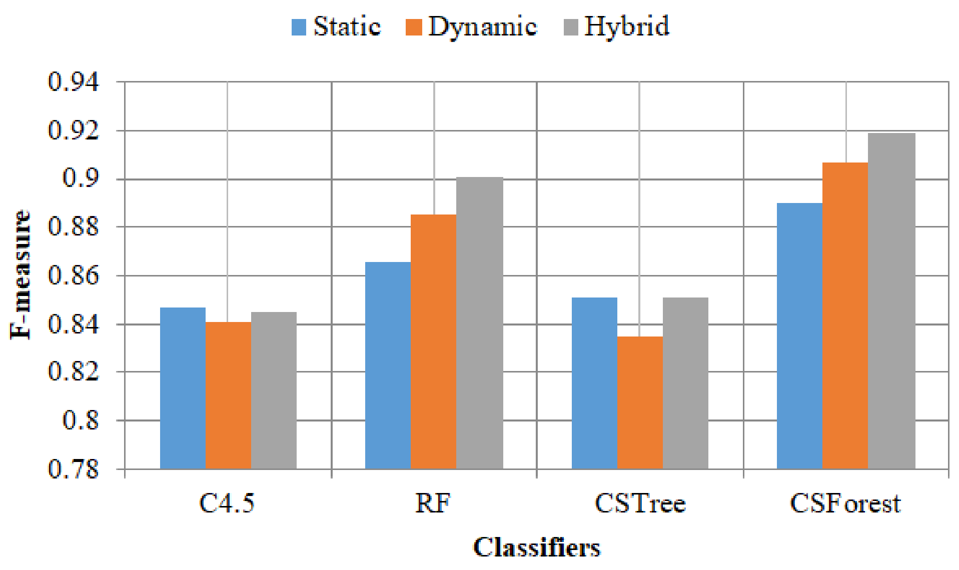


Figure 5.2: Comparison of various classification algorithms on the basis of F-measure

Figure 5.2 shows the comparison of various classification algorithms on the basis of f-measure for all three approaches. It shows that for both cost insensitive and sensitive classifiers, the integrated set of attributes (hybrid approach) offers the optimum f-measure. Furthermore, it demonstrates that the CSForest approach performs better in case of all three approaches as compared to other existing classifiers. The f-measure attained by CSForest in case of a hybrid set of features is 0.919 followed by RF which obtains 0.901 f-measure.

5.3 DISCUSSIONS

The proposed CSForest method is used to take care of imbalanced classes of Android malware. The experimental results are compared with RF, CSTree and C4.5 to determine the effectiveness of the proposed approach using static, dynamic and hybrid features. The results show that for both cost insensitive and sensitive classifiers, the integrated set of attributes (hybrid approach) provides the optimum f-measure. Furthermore, CSForest outperforms the other algorithms in all three approaches (static, dynamic and hybrid). The f-measure attained by CSForest in case of a hybrid set of features is 0.919 followed by RF which obtains 0.901 f-measure.

5.4 SUMMARY

This chapter presented CSForest technique to cope with the imbalanced classes of Android malware apps. This method is applied on the self-created dataset and the results are compared with CSTree, RF and C4.5 to identify the effectiveness of the proposed approach using static, dynamic and hybrid features. The results demonstrate that when hybrid set of features is considered, CSForest performs best for family classification of Android malware. The risk posed by malware necessitates the development of dependable and precise methods for assessing the risk in Android apps. This challenge is addressed by designing a rule-based model for identifying the risk of Android app features which is presented in Chapter 6.

CHAPTER 6

PROPOSED MODEL FOR IDENTIFYING THE LEVEL OF ANDROID APPLICATION FEATURES

Malware developers generate new malware on regular basis which poses several threats to the system's security and the privacy of users. People don't have much awareness and knowledge to determine whether the app is harmful or not. Typically, while downloading an app from the Android app store, customers overlook or fail to read the terms and conditions. Unfortunately, attackers take advantage of this tendency and attack mobile systems. In this case, a user downloads an app from a third-party store which requires certain permissions to be granted before it can be installed on the mobile device. Because of the unawareness about hazardous permissions, the user grants all rights and installs the app. This situation exemplifies the mobile device's susceptibility (shown in Figure 6.1).

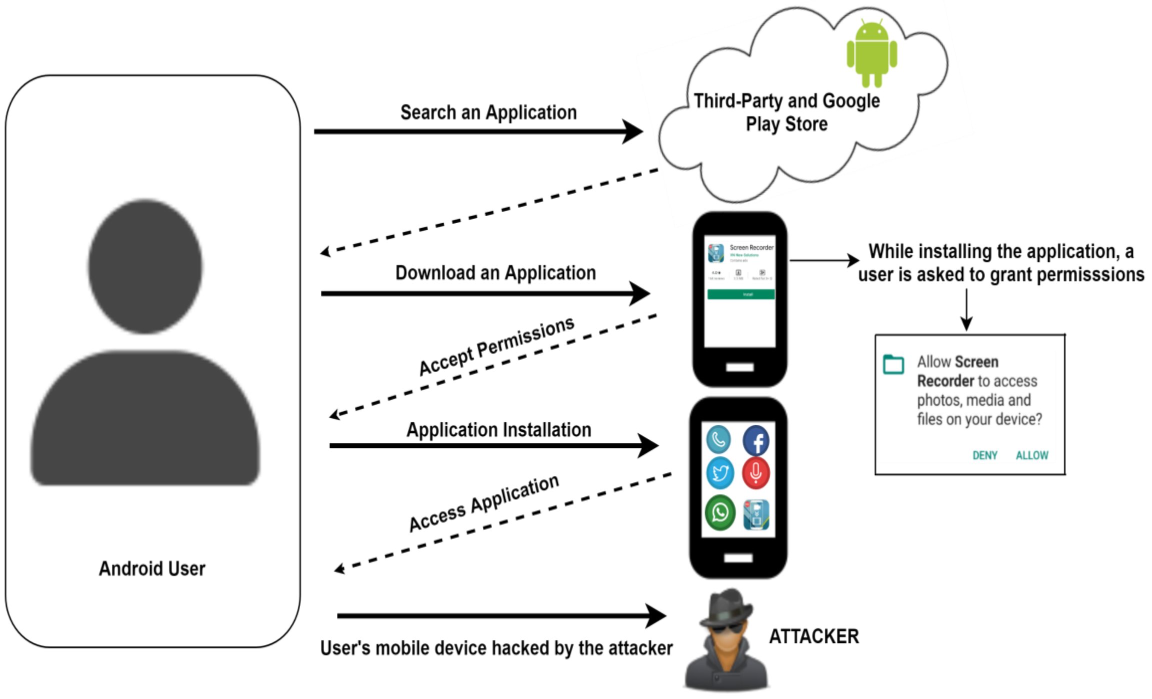

Third-Party and Google Play Store
Search an Application
Download an Application
Accept Permissions
Application Installation
Access Application
While installing the application, a user is asked to grant permisssions
Allow Screen Recorder to access photos, media and files on your device?
DENY
ALLOW
Android User
User's mobile device hacked by the attacker
ATTACKER

Figure 6.1: Process of targeting mobile device using permissions

The risk posed by Android malware necessitates the development of dependable and precise methods for assessing the risk in Android apps [216]. The goal of risk management is to anticipate potential problems before they actually happen and cause the damage. [217, 218]. Industries providing anti-malware solutions, compute the risk associated with a piece of malware using the approaches involving human intervention along with a large number of resources [219, 220]. With the increase in the volume of malware, it is impossible to allocate a significant number of resources for analyzing the threat or risk posed by an Android app. To address this issue, this chapter presents a rule-based model to identify the risk level of Android app features. The proposed model assigns the risk levels (No, Low, Medium and High) to Android app features. The static features (permissions and API calls) in the data are examined statistically to come up with a hypothesis for identifying their risk factor. In order to test the hypothesis, ANOVA method is used. Afterward, a weight is assigned to the features under each category to compute the threat score of a particular app.

6.1 PROPOSED METHODOLOGY

The workflow of the proposed approach for the quantitative threat assessment of an Android app is discussed in this section. The Android samples are gathered from various sources including apkmirror [185], apkpure [186] and virusshare [187]. All the apps are labelled as benign or malignant based on their scanning results. The static malware analysis is performed to extract static permissions and API calls. The dataset created in this step is analyzed statistically to frame a set of rules (hypothesis) to identify the risk factor of permissions and API calls. These risk factors are then used to compute the threat score of Android apps. Figure 6.2 depicts the workflow of the methodology used for the identification of risk factor of Android app features. The detail description of different steps is given below.

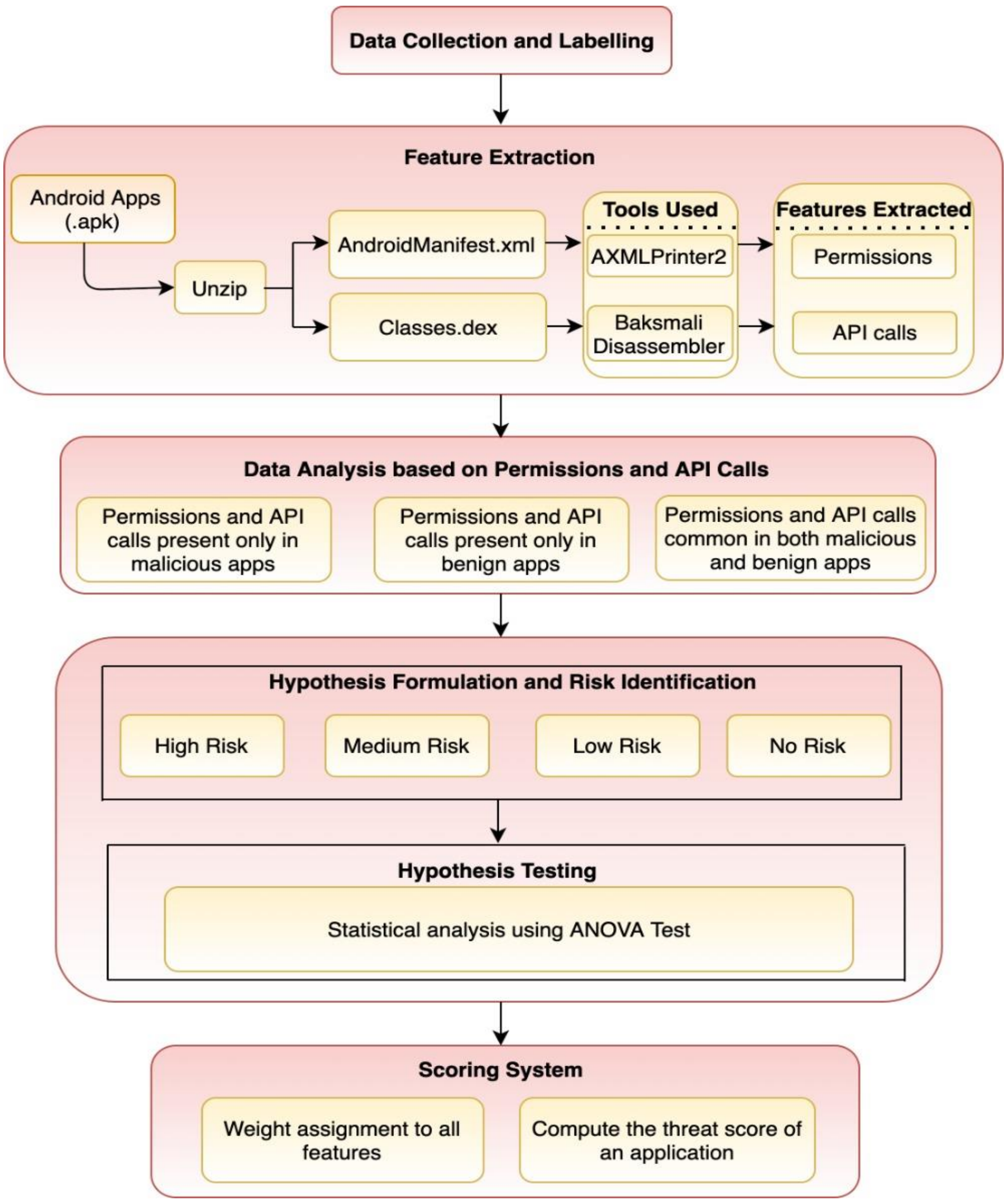

Data Collection and Labelling
Feature Extraction
Android Apps (.apk)
Unzip
AndroidManifest.xml
Classes.dex
Tools Used
AXMLPrinter2
Baksmali Disassembler
Features Extracted
Permissions
API calls
Data Analysis based on Permissions and API Calls
Permissions and API calls present only in malicious apps
Permissions and API calls present only in benign apps
Permissions and API calls common in both malicious and benign apps
Hypothesis Formulation and Risk Identification
High Risk
Medium Risk
Low Risk
No Risk
Hypothesis Testing
Statistical analysis using ANOVA Test
Scoring System
Weight assignment to all features
Compute the threat score of an application

Figure 6.2: Workflow of the methodology used for identification of risk factor of Android app features

6.1.1 Data Collection and Feature Extraction

Data collection and Labelling as well as feature extraction steps are discussed in chapter 3 under subsection 3.1. In this work, static API calls and permissions are considered for identifying the risk factor of Android app features. A total of 47 API calls and 277 permissions are mined from these apps.

6.1.2 Data Analysis based on Android Permissions and API calls

In this step, the created dataset is analyzed based on static permissions and API calls. Based on the analysis, these permissions and API calls are grouped in three categories i.e. API calls and permissions that are found only in malicious apps, only in benign apps and common in both malware and benign apps. Out of 277 permissions, 22 are found only in malicious apps, 61 are found only in benign apps and 82 are common in both malware and benign apps. In case of API calls out of 47, 6 are found only in malicious apps, 10 are found only in benign apps and 31 are common in both malware and benign apps. The detailed analysis of the dataset is given as follows.

- **API calls and permissions found in benign apps**- The entire dataset is analysed to filter those API calls and permissions which are present only in benign apps. A total of 10 API calls and 61 permissions are found only in benign apps. Table A.1 and Table A.2 (presented in Appendix A) show the names of these API calls and permissions respectively.
- **API calls and permissions common in both benign and malware apps**- The entire dataset is analysed to filter those API calls and permissions which are common in both malware and benign apps. A total of 31 API calls and 82 permissions are found in both malware and benign apps. Table A.3 and Table A.4 (presented in Appendix A) show the names of these API calls and permissions respectively.
- **API calls and permissions found in malware apps**- The entire dataset is analysed to filter those API calls and permissions which are present only in malicious apps. A total of 6 API calls and 22 permissions are found only in malware apps. Table A.5 and Table A.6 (presented in Appendix A) show the names of these API calls and permissions respectively.

6.1.3 Hypothesis Formulation and Risk Identification

After filtering out the permissions and API calls into three categories, we frame a set of rules (hypothesis) to identify the risk factor of each feature (as shown in Figure 6.3). API calls and permissions found exclusively in malicious apps (during the data analysis step) are classified as high risk, while API calls and permissions found only in benign apps are classified as no risk. Algorithm 6.1 shows the steps for the identification of the risk factor of each API call and permission. Z is a set of benign (X_{ben}) and malicious (X_{mal}) apps. Assume there are a total of n apps containing p benign and q malicious apps. $X_{ben} = \{X_{ben1}, X_{ben2}, \dots X_{benp}\}$ and $X_{mal} = \{X_{mal1}, X_{mal2}, \dots X_{malq}\}$ is the set of benign and malware apps respectively. A is a set of all features.

Algorithm 6.1: Algorithm for identification of the risk factor of each permission and API call

Input: $Z = \{X_{ben1}, X_{ben2}, \dots X_{benp},\ X_{mal1}, X_{mal2}, \dots X_{malq}\}$, $p + q = n$, A: set of all features $A = \{A_1, A_2, \dots, A_k\}$.

Output: Risk factor of API calls and permissions.

```
    # Apply reverse engineering to all Android apps    (to extract API calls and permissions)
01: for j =1to n do
02:     reverse engineering ($X_{ben}$, $X_{mal}$)
03: end for
    # Identify the risk factor of API calls and permissions
    # Check $A_i$ in $X_{ben}$ and in $X_{mal}$
04: for i=1 to k
05: if ($A_i$ is present in $X_{mal}$ and not present in $X_{ben}$)
06:     then high risk
07:    else if ($A_i$ is present in both $X_{mal}$ and $X_{ben}$)
08:        if ($A_i$ in $X_{mal}$ >= $A_i$in $X_{ben}$)
09:            then Medium Risk
10:          else if ($A_i$ in $X_{mal}$ < $A_i$ in $X_{ben}$)
11:            then Low Risk
12:        end if
13:    else if ($A_i$ is not present in $X_{mal}$ and present in $X_{ben}$)
14:            then No Risk
15:    else
16:        Invalid feature
17: end if
18: end for
```

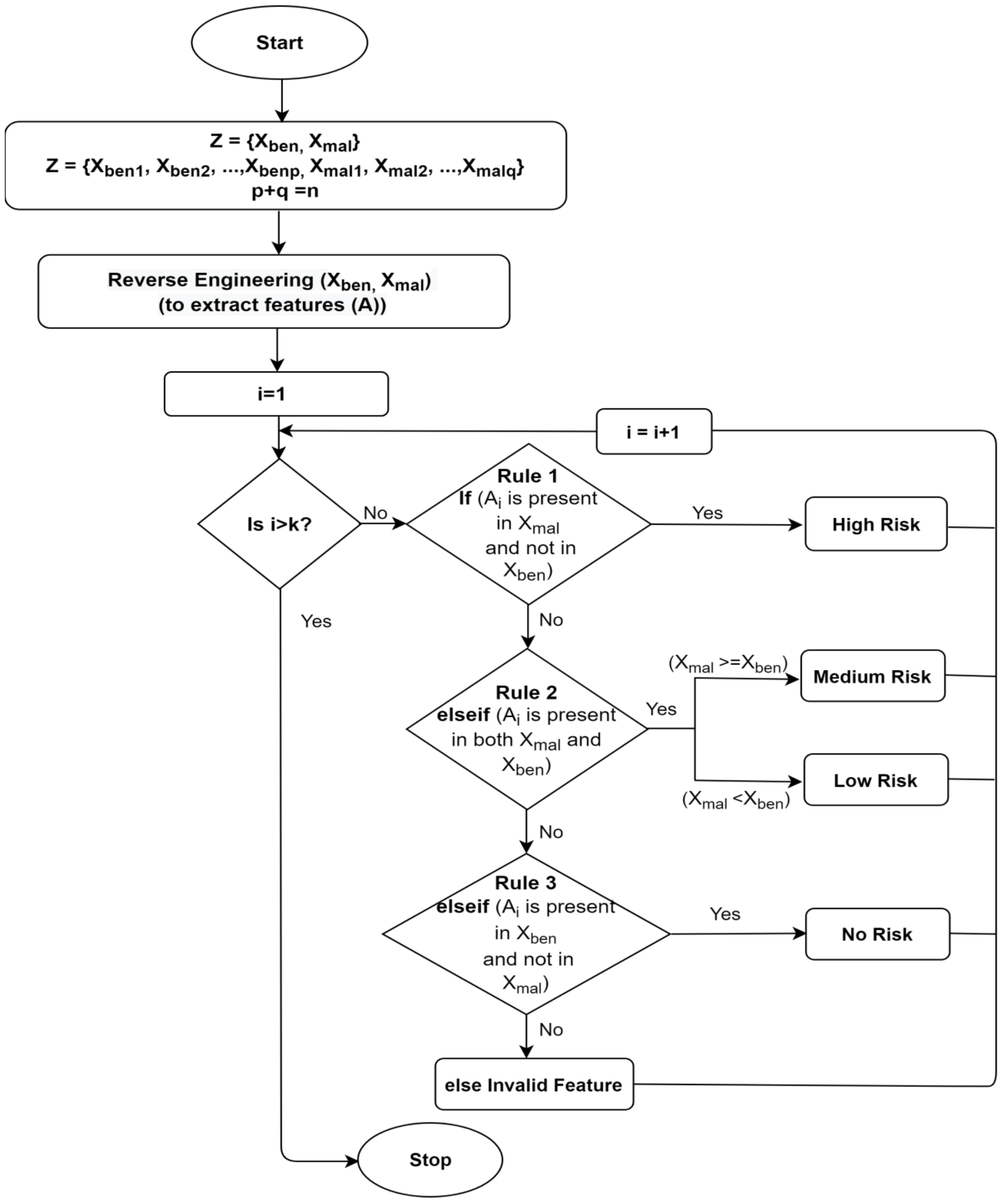

Start
Z = {Xben, Xmal}
Z = {Xben1, Xben2, ...,Xbenp, Xmal1, Xmal2, ...,Xmalq}
p+q =n
Reverse Engineering (Xben, Xmal)
(to extract features (A))
i=1
i = i+1
Is i>k?
No
Yes
Rule 1
If (Ai is present in Xmal and not in Xben)
Yes
High Risk
No
Rule 2
elseif (Ai is present in both Xmal and Xben)
Yes
(Xmal >=Xben)
Medium Risk
(Xmal <Xben)
Low Risk
No
Rule 3
elseif (Ai is present in Xben and not in Xmal)
Yes
No Risk
No
else Invalid Feature
Stop

Figure 6.3: Flowchart for identifying risk factor of features

Figure 6.3 shows the flowchart for identifying risk factors of app features. *Z* represents a set of benign (X_{ben}) and malicious (X_{mal}) apps and *A* represents a set of all features.

- **High Risk Factor-** This category includes permissions and API calls that are only seen in malware apps and not in benign apps. Figure 6.4 illustrates the percentage of high risk permissions in Android apps. It shows that in Android apps, the percentage of ACCESS_LOCATION_EXTRA_COMMANDS permission is greater (i.e. 11.22%). Figure 6.5 illustrates the graphical representation of percentage of high risk API calls in Android apps. It shows that the percentage of Runtime.exec API call in benign apps is the greater i.e. 26.39%. Table B.1 and Table B.2 (presented in Appendix B) demonstrate the percentage of high risk API calls and permissions respectively in Android apps.

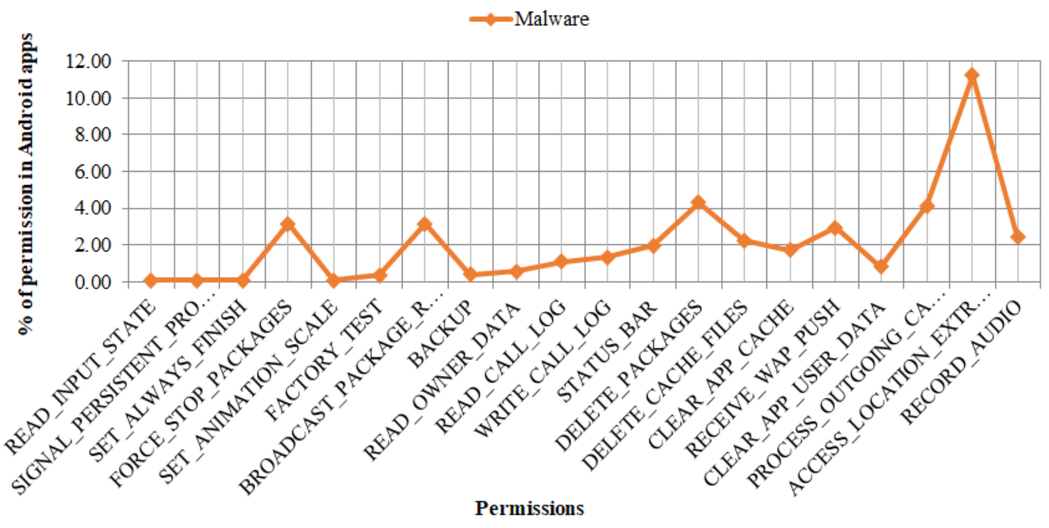


Figure 6.4: High risk permissions in Android apps

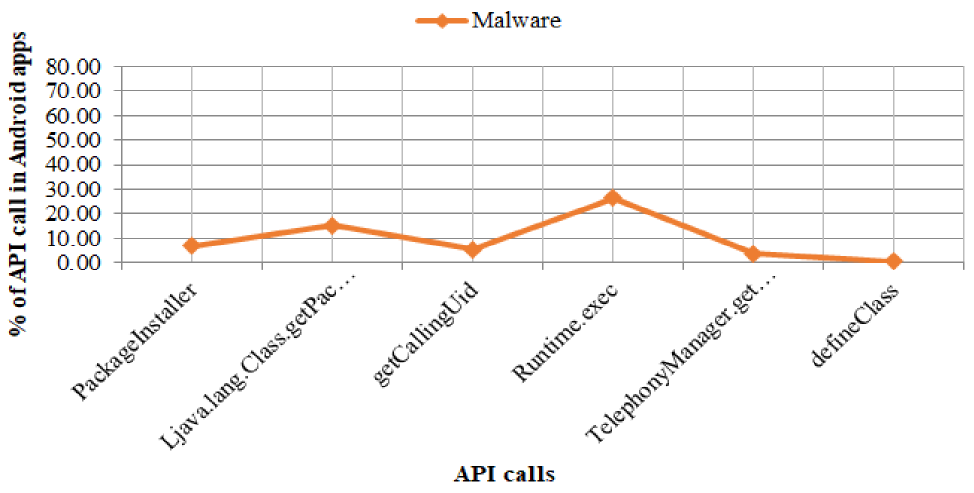


Figure 6.5: High risk API calls in Android apps

- **Medium Risk Factor-** This category includes API calls and permissions that are common in benign and malware apps. If the percentage of API calls and permissions in malware apps is greater than or equal to benign apps then it is considered in a medium risk category. Figure 6.6 shows a graphical representation of the percentage of few medium risk permissions in Android apps. It shows that percentage of

INTERNET permission is higher in both benign and malicious apps i.e. 82.17% and 95.94% respectively. Figure 6.7 shows a graphical representation of the percentage of medium risk API calls in Android apps. It shows that in both benign and malware apps, the percentage of android.os.IBinder API call is greater i.e. 0.85% and 0.92% respectively. Table B.3 and Table B.4 (presented in Appendix B) demonstrate the percentage of medium risk API calls and permissions respectively in Android apps.

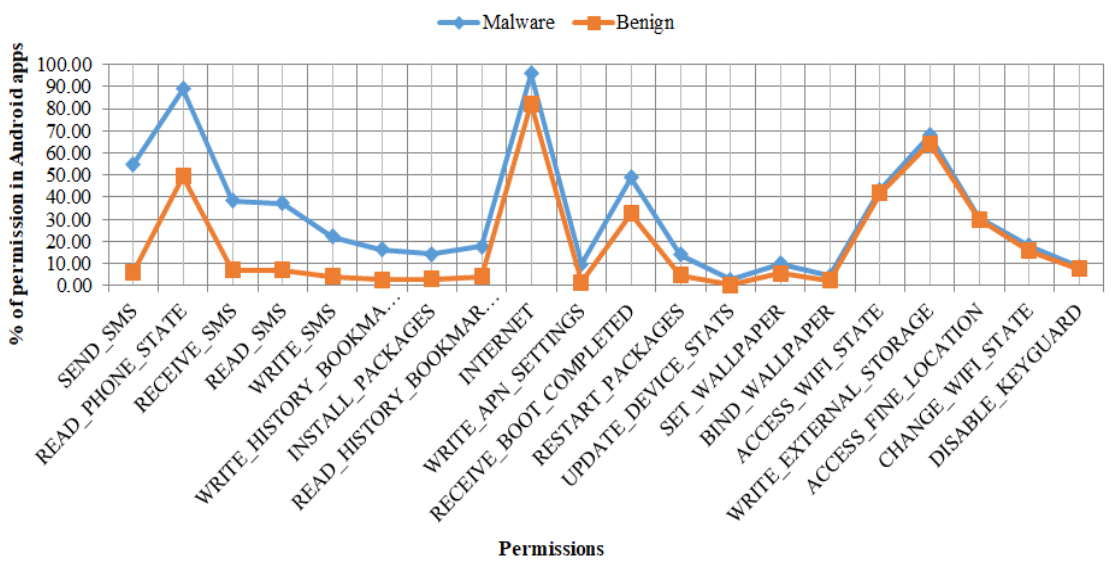


Figure 6.6: Medium risk permissions in Android apps

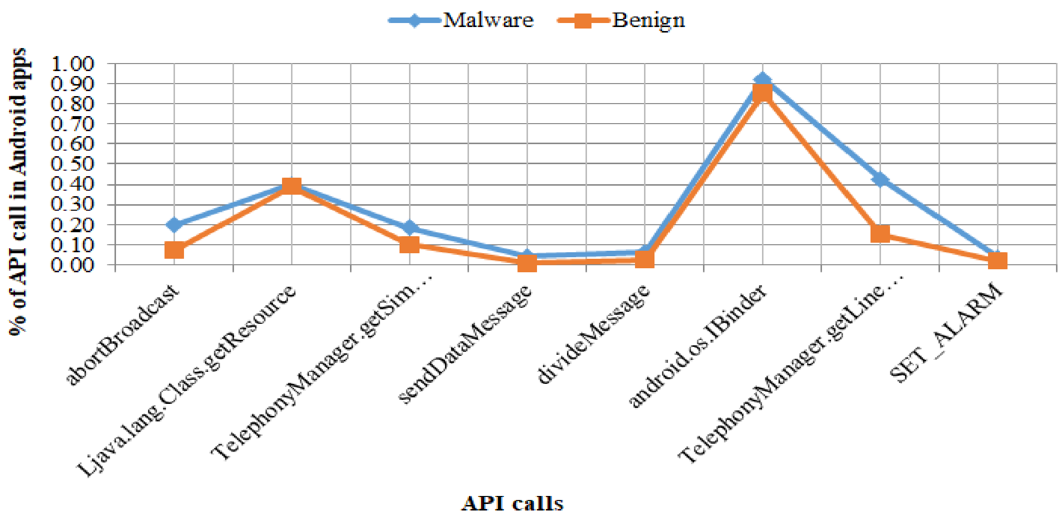


Figure 6.7: Medium risk API calls in Android apps

- **Low Risk Factor-** This category includes API calls and permissions that are common in benign and malware apps. If the percentage of API calls and permissions in malware apps is less than benign apps then it is considered in a low risk. Figure 6.8 shows a graphical representation of the percentage of few low risk permissions in Android apps. It shows that percentage of ACCESS_NETWORK_STATE permission is higher in both malicious and benign apps i.e. 65.66% and 76.33% respectively. Figure 6.9 illustrates the graphical representation of percentage of low risk API calls in Android apps. It shows that in both malware and benign apps, the percentage of onBind API call is greater i.e. 0.64% and 0.72% respectively. Table B.5 and Table B.6 (presented in Appendix B) demonstrate the percentage of low risk API calls and permissions respectively in Android apps.

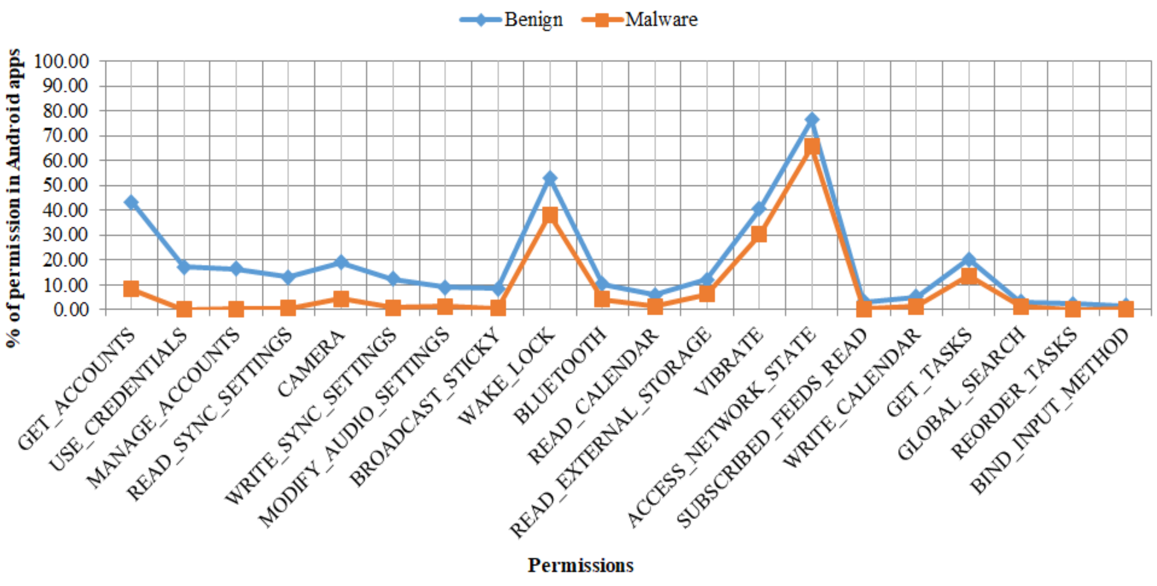


Figure 6.8: Low risk permissions in Android apps

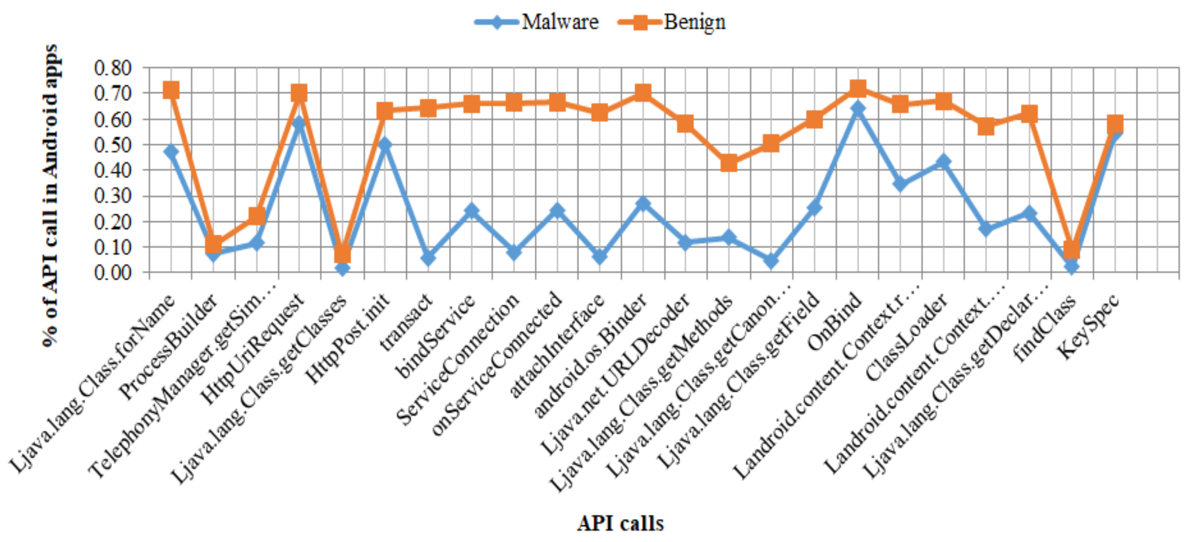


Figure 6.9: Low risk API calls in Android apps

- **No Risk Factor-** This category includes permissions and API calls that are only seen in benign apps and not in malware apps. Figure 6.10 illustrates the graphical representation of percentage of few no risk permissions in Android apps. It shows that in Android apps, the percentage of BIND_JOB_SERVICE permission is greater i.e. 45.39%. Figure 6.11 illustrates the graphical representation of percentage of no risk API calls in Android apps. It shows that the percentage of getBinder API call in benign apps is greater i.e. 0.31%. Table B.7 and Table B.8 (presented in Appendix B) demonstrate the percentage of no risk API calls and permissions respectively in Android apps.

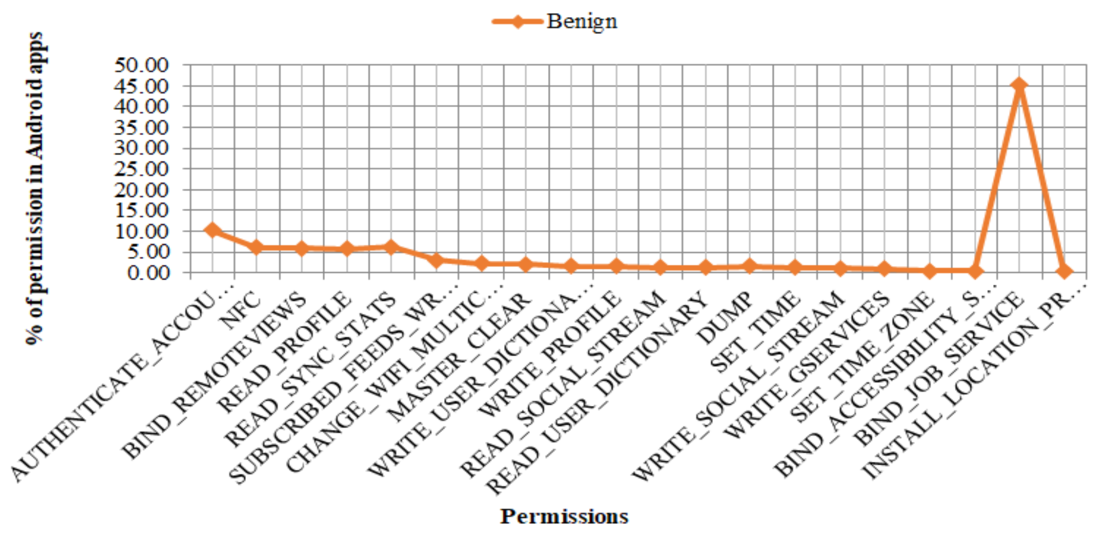


Figure 6.10: No risk permissions in Android apps

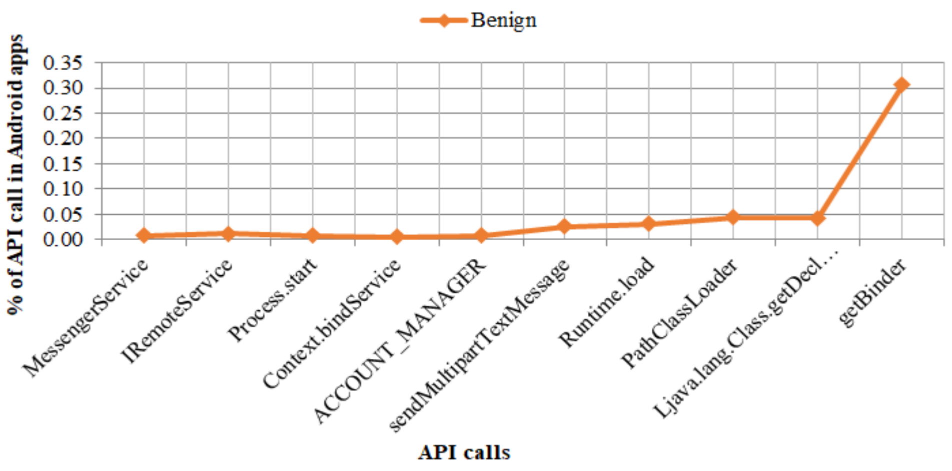


Figure 6.11: No risk API calls in Android apps

6.1.4 Statistical test using ANOVA

After identifying the risk factor (i.e. No, Low, Medium and High risk) of each feature, the statistical analysis test is perform to examine whether these risk factor varies from one another and mined the explicit decision rule. The ANOVA statistical test is used. It is a statistical technique which is used to check whether the experimental or survey results are significant and it also helps to find out whether the null hypothesis is accepted or rejected. ANOVA technique evaluates the difference in a scale level and dependent variable by a nominal-level variable having more than two categories. The *z* and *t*-test permits the nominal level variable to have only two categories. So to overcome the constraints of both the tests, ANOVA test came into existence. One-way ANOVA is used which relates to the number of independent variable rather than categories in each variable. It has one independent variable [221]. Table 6.1 shows the variability within the groups and between the groups.

Table 6.1: ANOVA source

Source of Variation	Sum of Squares (SS)	Degree of freedom (df)	Mean Square (MS)
Between Samples	$SSB = \sum n_i(\bar{X}_i - \bar{X})^2$	$K - 1$	$MSB = \frac{SSB}{K-1}$
Within Samples	$SSE = \sum\sum(X - \bar{X}_i)^2$	$N - K$	$MSE = \frac{SSE}{N-K}$
Total	$SST = \sum\sum(X - \bar{X})^2$	$N - 1$	

Here *N* represents the total number of observations, *X* represents the individual observation, $\bar{X}$ represents the overall sample mean, $\bar{X}_i$ denotes the sample mean of the i^{th} group and *K* denotes the number of independent comparison groups.

SS stands for Sum of Squares in the source (of variability) column. SST measures the variation of the data around the $\bar{X}$, SSB measures variation of the group means around the $\bar{X}$ and SSE measures the variation of each observation around its $\bar{X}_i$.

The next column degree of freedom (df) is the number of independent variables that a statistical analysis can estimate. The last column mean square is an estimate of population variance, and it is calculated by dividing the total squares by the number of degrees of freedom.

F statistic is the ratio of variability between and within the groups. The value of *F* statistic is computed as shown in Equation 6.1.

$$F = \frac{MS\_between}{MS\_within} \tag{6.1}$$

6.1.5 Scoring System

Based on the set of rules, API calls and permissions are categorized into four risk categories. With the help of experts, a weight is assigned to the features (API call and permission) that fall into distinct categories after filtration. All features (permissions and API calls) in the high risk category have a weight of 3. This category includes 28 features, including 22 permissions and 6 API calls. A weight of 2 is assigned to API calls and permissions in the medium risk category. This category includes a total of 39 features out of which 31 permissions and 8 API calls. The features (permissions and API calls) in the low risk category have a weight of 1. This category includes 74 features, including 51 permissions and 23 API calls. A weight of 0 is assigned to API calls and permissions in the no risk category. This category includes a total of 71 features out of which 61 permissions and 10 API calls. Table 6.2 shows the number of attributes belonging to different risk categories.

Table 6.2: Number of attributes belonging to different risk categories and assigned weights

Risk Category	Explanation	Number of attributes		Weight Assigned
		API calls	Permissions	
High	Feature that can only be seen in malware apps	6	22	3
Medium	Feature that is present in both benign and malware apps but the percentage of feature in malware apps is more than or equal to benign apps	8	31	2
Low	Feature that is present in both benign and malware apps but the percentage of feature in malware apps is less than benign apps.	23	51	1
No	Feature that can only be seen in benign apps	10	61	0

After assigning weight to each permission and API call falling under different risk categories, total threat score of an app *(T)* is calculated using Equation 6.2.

$$T = \sum_i A_i \; W_i \tag{6.2}$$

Where A_i represents the i^{th} feature and W_i is the corresponding weight with respect to that feature. If a feature is available in an app, its value is 1 otherwise it is 0. The impact of an app on the victim device is represented by its threat score.

After calculating the overall threat score for all apps, the value is normalized on a scale of 1 to 10 using min-max normalization as given in Equation 6.3.

$$T_{normalized} = \frac{(T - T_{min})}{(T_{max} - T_{min})} \times 10 \tag{6.3}$$

Here, T stands for the input value, T_{max} and T_{min} stand for the maximum and minimum value in the data respectively.

Figure 6.12 depicts the threat score of all apps on a scale of one to ten. It reveals that there are 431 apps in our dataset with a threat score of less than 2 and 100 apps with a threat value of more than 7. It shows that the higher the threat score, the higher the app's risk. This threat score assessment can also be used to deliver early alerts about a specific Android malware sample, allowing for immediate attention and resource allocation for a more thorough investigation.

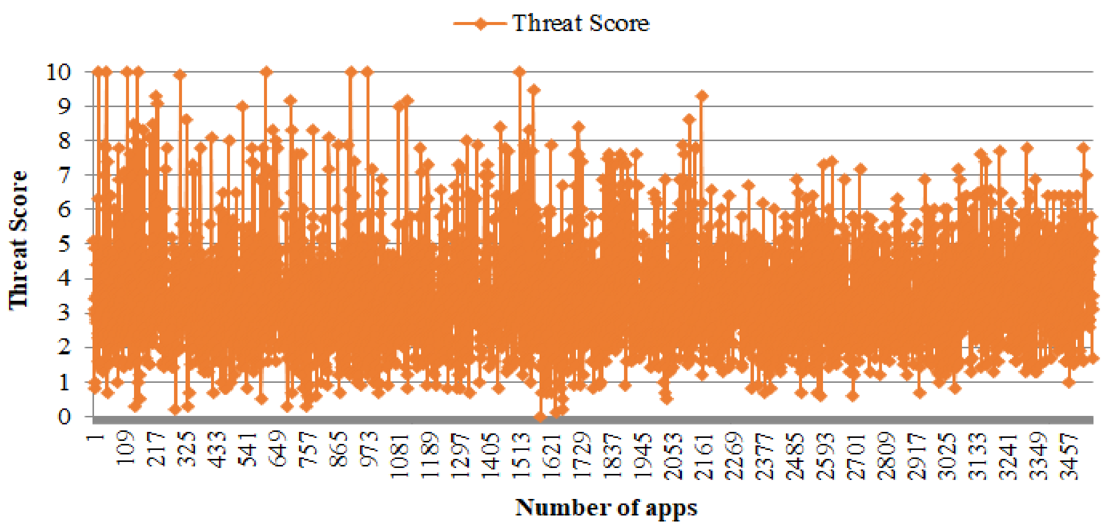


Figure 6.12: Threat score of all apps

6.2 RESULTS OF ANOVA

This section summarises the results of the ANOVA technique and shows the visualization of distinct risk factors using API calls and permissions. ANOVA is used to compare distinct risk factors (i.e. no, low, medium, and high risk). Because our analysis consisted of four factors so the ANOVA test is used in this study. The null hypothesis asserts that the mean values of various risk factors do not differ significantly.

Table 6.3: ANOVA: single factor (summary)

Groups	Count	Sum	Average	Variance
High Risk	3547	10125.074	2.855	28.777
Medium Risk	3547	74714.339	21.064	82.475
Low Risk	3547	83312.400	23.488	197.543
No Risk	3547	6163.025	1.738	9.339

Table 6.4: ANOVA statistical comparison

Source of Variation	SS	Df	MS	F	P-value	F_{crit}
Between Groups	1428608	3	476202.683	5987.439	0	2.606
Within Groups	1128105	14184	79.534			
Total	2556713	14187				

The ANOVA single factor summary is shown in Table 6.3. It calculates the sum, count, variance and average of each group. Table 6.4 shows a statistical comparison of several risk factors using the ANOVA test. Table 6.4 shows that at the 5% level of significance, the computed value (F) i.e. 5987.439 is greater than the tabulated value (Fcrit) i.e. 2.606. The null hypothesis is thus rejected in this case. As a result, we can conclude that the mean values of various risk factors are significantly different. Figures 6.13 and 6.14 illustrate the percentage of high, medium, low and no risk API calls and permissions respectively in an app.

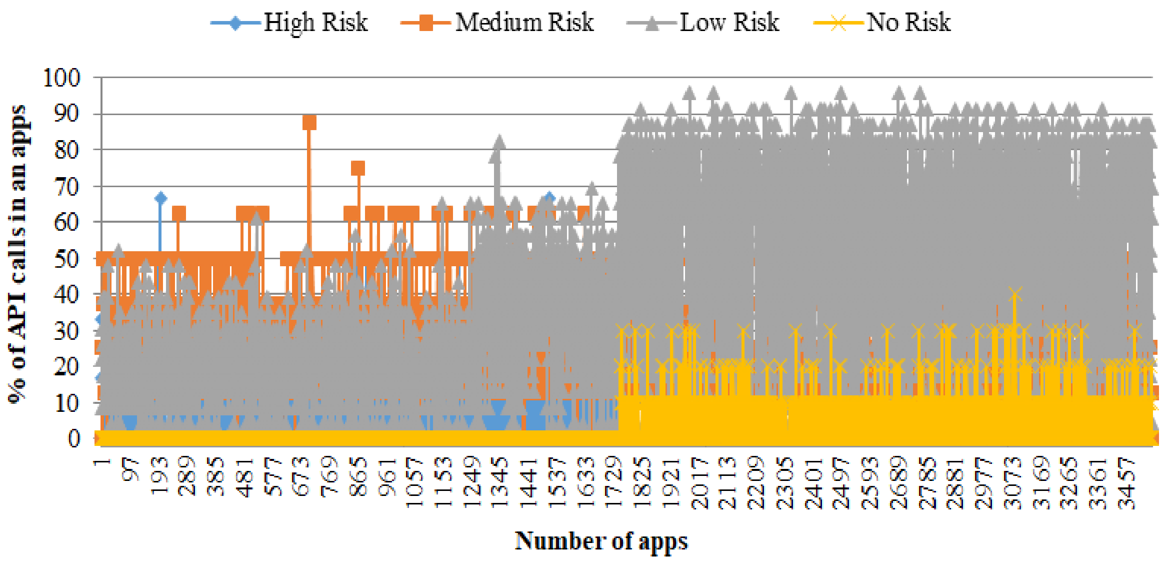


Figure 6.13: Visualization of four risk factors in an app based on API calls

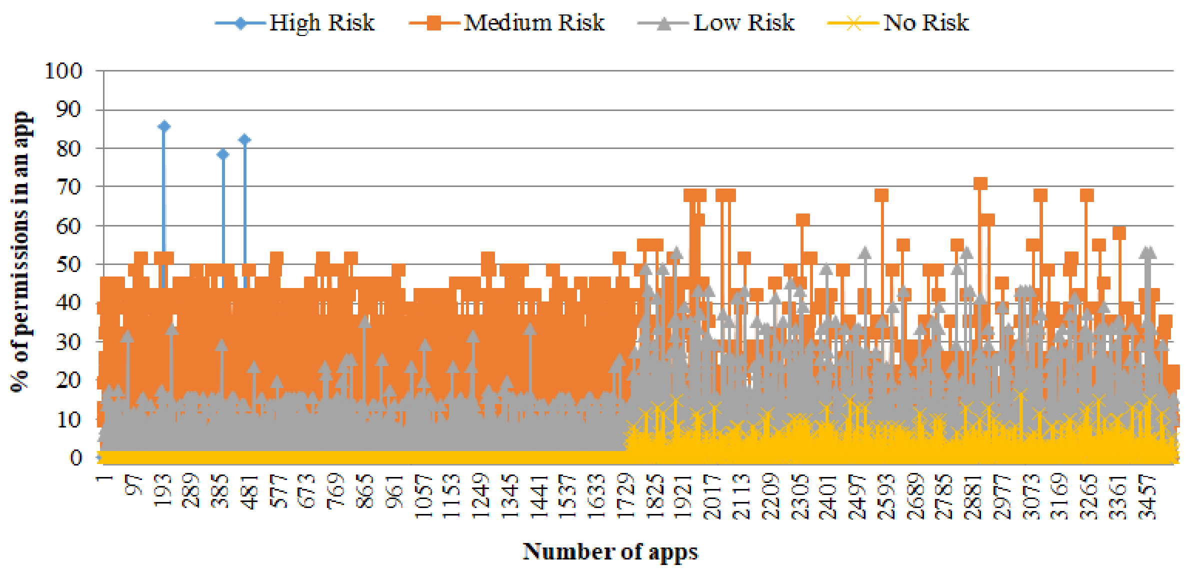


Figure 6.14: Visualization of four risk factors in an app based on permissions

Figures 6.13 and 6.14 depict all four risk factors in an app based on API calls and permissions respectively. The findings show that the majority of apps fall into the low risk category when it comes to API calls, and the majority of apps fall into the medium and low risk categories when it comes to permissions.

6.3 DISCUSSIONS

The proposed approach makes use of permissions and API calls for determining the risk factor of Android app features. There are a total of 3,547 instances in the dataset, with 277 permissions and 47 API calls as features. While analysing the data, it is found that out of 277 permissions, 22 permissions are found in only malicious apps, 61 are found in only benign apps and 82 are common in both malware and benign apps. Out of 47, 6 of the API calls are found only in malicious apps, 10 are found only in benign apps and 31 are common in both benign and malicious apps. The data analysis results are used to frame a hypothesis for the identification of risk factor in each feature present in an app. ANOVA technique is used to test the hypothesis. The results demonstrate that the computed value of F is 5987.439, which is significantly greater than the tabulated value Fcrit i.e. 2.606 at 5% level of significance. A weight is assigned to all permissions and API calls on the basis of the risk category to which they belong. Finally, a threat score is computed for each app. The computed threat score can assist in providing early warnings about a malicious app so that instant attention could be paid with respect of assigning resources for deeper investigation.

6.4 SUMMARY

This chapter introduced a rule based model for determining the risk factor of Android app features using permissions and API calls. First and foremost, a dataset is developed based on static malware analysis and was made publicly available on kaggle. It is then analyzed to frame the hypothesis for the identification of risk factor of each feature. ANOVA technique is used to test the hypothesis. The results indicate that the mean values of different risk factors differ significantly. Afterward, a weight is assigned to the features under each category to compute the threat score of a particular app. This threat score can help the user to understand how risky it is to install an app on a mobile device. In the next chapter, conclusion of the entire work is presented. The future research areas for malware detection and classification are also mentioned.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This chapter concludes research work regarding Android malware detection, classification, and threat assessment using computational techniques. The major research contributions and future research directions are discussed as follows.

7.1 MAJOR RESEARCH CONTRIBUTIONS

The following are the major contributions of this research work.

- The two datasets (Android malware detection and family classification) are created using a comprehensive set of attributes acquired after performing static and dynamic malware analysis. These datasets have been made public on kaggle and GitHub. The goal behind creating these datasets is to aid researchers and anti-malware tool developers in improving and developing new methodologies and tools for identifying and classifying malware. These datasets can be used as benchmark datasets by various researchers to validate their proposed techniques.
- An approach, making use of integrated set of static and dynamic features is proposed. It has the ability to effectively analyze, detect and classify unknown malware. Individual approach either dynamic or static is insufficient due to obfuscation and execution stalling techniques being used by malware authors. The findings demonstrate that the proposed integrated approach performs better as compared to the approaches which use only static or dynamic features.
- An approach named as MalDetect has been proposed to improve the detection (binary classification) of Android malware. The approach fuses the base classifiers on the basis of proposed ranking schemes defined on their error rate. The proposed approach is evaluated on two datasets i.e. Drebin (benchmark) and AndroMD (self-created). The findings suggest that the proposed approach is effective than traditional base classifiers and ensemble learning techniques.

- A cost-sensitive learning approach is proposed for imbalanced family categorization of Android malicious applications. The proposed approach is compared with CSTree, RF and C4.5 to identify its effectiveness in categorizing malicious app families. The findings show that the proposed approach is effective in identifying the families of malicious apps.
- A rule-based approach is proposed to compute the threat score of real Android apps. The data analysis results are used to frame a set of rules for the identification of risk factor of the features of Android apps. ANOVA is used to test the hypothesis. Afterward, a weight is assigned to the features under each category to compute the threat score (on a scale of 1-10) of a particular app.

Using the proposed techniques, a threat intelligence platform for Android systems can be developed which can act as an early warning system. The intelligent information generated from the system can be shared with security experts and other stakeholders so that they can issue the early warnings and advisories about emerging malicious threats and developing solutions to minimize the risks posed by changing threat landscape.

7.2 SCOPE FOR FUTURE WORK

This section briefly discusses the future directions and improvements in the present research.

- Due to the increased use of mobile apps, the variety, volume and velocity of malware targeting mobile devices has increased. It is necessary for the research community to build and develop malware classification methods that make use of big data analytics in order to increase prediction accuracy. Furthermore, the steadily streaming data on the network motivates researchers to develop unique approaches and principles for extracting useful information from raw data using data mining and ML algorithms.
- To overcome the constraints of signature based method, this research uses ML techniques that learn from instances using a set of attributes mined through static and dynamic analysis of malware. Only a limited set of attributes is used in the existing research. To deal with the growing complexity and sophistication of malicious apps, security researchers need to explore more robust set of attributes obtained using both static and dynamic analysis of malware.

- The more emphasis could be laid on deep learning methods for identification and classification of malicious apps.
- One of the contributions of this work is that it presents a rule based model to identify the risk level of Android features. These risk levels are used to compute the threat score of Android apps which in turn helps in prioritizing resource assignments for conducting a closer manual analysis of suspicious apps and to warn the mobile users before installing these apps. This method makes use of only static features. In future, more emphasis could be laid on identifying the dynamic features for computing the threat score. A fuzzy logic method could also be explored for risk identification of an Android app.
- Sharing of intelligent information obtained is one of the important aspects related to the present work. In future, development of platforms for sharing or exchanging intelligent information related to cyber threats can be explored.

REFERENCES

[1] Mobile Operating System Market Share Worldwide (accessed October, 2019) https://gs.statcounter.com/os-market-share/mobile/worldwide.

[2] M. Wilson, T-mobile g1: Full details of the htc dream android phone, Gizmodo-We come from the future, 2008.

[3] Smartphone Market Share (accessed December, 2019) https://www.idc.com/promo/smartphone-market-share/os.

[4] J. DiMarzio, Beginning Android Programming with Android Studio, *John Wiley & Sons*, 2016.

[5] Google Android (accessed December, 2020) https://www.android.com/.

[6] 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, pp. 1-41, 2017.

[7] Y. S. Yen, and H. M. Sun, "An Android mutation malware detection based on deep learning using visualization of importance from codes," *Microelectronics Reliability*, vol. 93, pp. 109-114, 2019.

[8] McAfee, Threats report, 2013. http://www.mcafee.com/uk/resources/reports/rp-quarterlythreatq1-2013.pdf.

[9] McAfee, Mobile security report, 2014. http://www.mcafee.com/uk/resources/reports/rp-mobilesecurityconsumer-trends.pdf.

[10] NJCCIC Android Malware variants (accessed February, 2019) https://www.cyber.nj.gov/threat-center/threat-profiles/android-malware-variants/.

[11] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, and S. Chen, "A survey on mobile malware detection techniques," *IEICE Transactions on Information and Systems*, vol. 103, no. 2, pp. 204-211, 2020.

[12] O. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249-6271, 2020.

[13] D. Gupta and R. Rani, "Big data framework for zero-day malware detection," *Cybernetics and Systems*, vol. 49, no. 2, pp. 103-121, 2018.

[14] J. Senanayake, H. Kalutarage, and M. O. A. Kadri, “Android Mobile Malware Detection Using Machine Learning: A Systematic Review,” *Electronics,* vol. 10, no. 13, p. 1606, 2021.

[15] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, “Constructing features for detecting Android malicious applications: Issues, taxonomy and directions,” *IEEE Access,* vol. 7, pp. 67602–67631, 2019.

[16] K. Dunham, S. Hartman, M. Quintans, J. A. Morales, and T. Strazzere, “Android malware and analysis,” CRC Press, 2014.

[17] S. Alam, S. A. Alharbi, and S. Yildirim, “Mining nested flow of dominant APIs for detecting android malware,” *Computer Networks*, vol. 167, 2020.

[18] O. Olukoya, L. Mackenzie, and I. Omoronyia, “Towards using unstructured user input request for malware detection,” *Computers & Security*, vol. 93, 2020.

[19] F. Shen, J. D. Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek, “Android malware detection using complex-flows,” *IEEE Transactions on Mobile Computing*, vol. 18, no. 6, pp. 1231–1245, 2019.

[20] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, “Android malware familial classification and representative sample selection via frequent subgraph analysis,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 1890–1905, 2018.

[21] X. Xiao, Z. Wang, Q. Li, S. Xia, and Y. Jiang, “Back-propagation neural network on Markov chains from system call sequences: A new approach for detecting Android malware with system call sequences,” *IET Information Security*, vol. 11, no. 1, pp. 8–15, 2017.

[22] W. Zhang, H. Wang, H. He, and P. Liu, “DAMBA: Detecting Android malware by ORGB analysis,” *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 55–69, 2020.

[23] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, “MADAM: Effective and efficient behavior-based Android malware detection and prevention,” *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2018.

[24] S. Singla, E. Gandotra, D. Bansal, and S. Sofat, “Detecting and classifying morphed malwares: A survey,” *International Journal of Computer Applications,* vol. 122, no. 10, 2015.

[25] M. Irshad, H. M. A. Khateeb, A. Mansour, M. Ashawa, and M. Hamisu, “Effective methods to detect metamorphic malware: a systematic review,” *International Journal of Electronic Security and Digital Forensics,* vol. 10, no. 2, pp. 138-154, 2018.

[26] E. Gandotra, D. Bansal, and S. Sofat, “Tools & Techniques for Malware Analysis and Classification,” *International Journal of Next-Generation Computing,* vol. 7, no. 3, 2016.

[27] E. Gandotra, D. Bansal, and S. Sofat, “Malware analysis and classification: A survey,” *Journal of Information Security,* vol. 5, no. 2, p. 56, 2014.

[28] Y. Pan, X. Ge, C. Fang, and Y. Fan, “A systematic literature review of android malware detection using static analysis,” *IEEE Access,* vol. 8, pp. 116363-116379, 2020.

[29] E. Gandotra, D. Bansal, and S. Sofat, “Malware intelligence: beyond malware analysis,” *International Journal of Advanced Intelligence Paradigms,* vol. 13, no. 1-2 , pp. 80-100, 2019.

[30] P. Yan and Z. Yan, “A survey on dynamic mobile malware detection,” *Software Quality Journal,* vol. 26, no. 3, pp. 891-919, 2018.

[31] F. Tchakounte, “Permission-based malware detection mechanisms on android: Analysis and perspectives,” *Journal of Computer Science,* vol. 1, no. 2, 2014.

[32] S. Verma and S. K. Muttoo, “An android malware detection framework-based on permissions and intents,” *Defence Science Journal,* vol. 66, no. 6, pp. 618-623, 2016.

[33] F. Shang, Y. Li, X. Deng, and D. He, “Android malware detection method based on naive Bayes and permission correlation algorithm,” *Cluster Computing,* vol. 21, no. 1, pp. 955-966, 2018.

[34] G. Tao Z. Zheng, Z. Guo, and M. R. Lyu, “MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs,” *IEEE Transactions on Reliability,* vol. 67, no. 1, pp. 355-369, 2017.

[35] V. Rastogi, Y. Chen, and X. Jiang, “Catch me if you can: Evaluating android anti-malware against transformation attacks,” *IEEE Transactions on Information Forensics and Security,* vol. 9, no. 1, pp. 99-108, 2013.

[36] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck. “Mast: Triage for market-scale mobile malware analysis,” In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks,* pp. 13-24, 2013.

[37] J. W. Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, "Andro-AutoPsy: Anti-malware system based on similarity matching of malware and malware creator-centric information," *Digital Investigation*, vol. 14, pp. 17-35, 2015.

[38] Y. Zhang, Y. Yuexiang and X. Wang, "A novel android malware detection approach based on convolutional neural network," In *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy*, pp. 144-149. 2018.

[39] Z. Wang, J. Cai, S. Cheng, and W. Li, "DroidDeepLearner: Identifying Android malware using deep learning," *in Proc. the 37th Sarnoff Symposium*, pp. 160-165, 2018.

[40] K. Allix, T. F. Bissyande, Q. Jérome, J. Klein, and Y. L. Traon, "Empirical assessment of machine learning-based malware detectors for Android," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 21, no. 1, pp. 183-211, 2016.

[41] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1-9, 2018.

[42] N. Xie, X. Wang, W. Wang, and J. Liu, "Fingerprinting android malware families," *Frontiers of Computer Science*, vol. 13, no. 3, 2018.

[43] Y. Liu, L. Zhang, and X. Huang, "Using g features to improve the efficiency of function call graph based android malware detection," *Wireless Personal Communications*, pp. 2947-2955, 2018.

[44] JAD (accessed February, 2019) http://www.javadecompilers.com/jad.

[45] DED (accessed February, 2019) http://siis.cse.psu.edu/ded/.

[46] Android4me: J2ME port of Google's Android (2011) https://code.google.com/p/android4me/downloads/list.

[47] Q. Jerome, K. Allix, R. State, and T. Engel, "Using opcode-sequences to detect malicious Android applications," *in 2014 IEEE International Conference on Communications (ICC)*, pp. 313-320, 2014.

[48] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," *IEEE Access*, vol. 6, pp. 4321-4339, 2018.

[49] A. Martín, R. L. Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," *Information Fusion*, vol. 52, 2019, pp. 128-142.

[50] P. Feng, J. Ma, C.Sun, X. Xu, and Y. Ma, "A Novel Dynamic Android Malware Detection System With Ensemble Learning," *IEEE Access*, vol. 6, 2018, pp. 30996-31011.

[51] V. Sihag, M. Vardhan, P. Singh, G. Choudhary, and S. Son, "De-LADY: Deep learning based Android malware detection using Dynamic features," *Journal of Internet Services and Information Security (JISIS)*, vol. 11, no. 2, pp. 34-45, 2021.

[52] CuckooDroid. (accessed October, 2019). https://cuckoo-droid.readthedocs.io/en/latest/installation/.

[53] A. Desnos, and P. Lantz, "Droidbox: An android application sandbox for dynamic analysis," *Lund Univ. Tech. Rep: Lund*, Sweden, 2011.

[54] AppPlayground (accessed February, 2019). https://github.com/keymetrics/app-playground.

[55] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1-29, 2014.

[56] Y. Zhou, and X. Jiang, "Dissecting android malware: Characterization and evolution," In *2012 IEEE symposium on security and privacy*, pp. 95-109, 2012.

[57] Tom Mitchell, Machine Learning, *McGraw Hill*, 1997.

[58] J. Latif, C. Xiao, S. Tu, S. U. Rehman, A. Imran, and A. Bilal. "Implementation and use of disease diagnosis systems for electronic medical records based on machine learning: A complete review," *IEEE Access*, vol. 8, pp. 150489-150513, 2020.

[59] S. Hakak, M. Alazab, S. Khan, T. R. Gadekallu, P. K. R. Maddikunta, and W. Z. Khan, "An ensemble machine learning approach through effective feature extraction to classify fake news," *Future Generation Computer Systems*, vol. 117, pp. 47-58, 2021.

[60] J. Padmanabhan, and M. J. J. Premkumar, "Machine learning in automatic speech recognition: A survey," *IETE Technical Review*, vol. 32, no. 4, pp. 240-251, 2015.

[61] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504-518, 2015.

[62] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-centric Computing and Information Sciences*, vol. 8, pp. 1-22, 2018.

[63] Mobile Development (accessed August, 2021) https://www.mindinventory.com/blog/android-app-development-trends/.

[64] Mobile Applications in Corona Virus Pandemic (accessed December, 2020) https://scientificwebs.com/importance-of-mobile-applications-in-corona-virus-pandemic/.

[65] McAfee Labs (2020) Threat Predictions Report, McAfee Labs, Santa Clara, CA, USA.

[66] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: A behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161-190, 2012.

[67] J. O. Kephart, "Automatic extraction of computer virus signatures," *In Proc. 4th Virus Bulletin International Conference, Abingdon*, England, pp. 178-184, 1994.

[68] Y. Tang, B. Xiao, and X. Lu, "Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms," *computers & security*, vol. 28, no. 8, pp. 827-842, 2009.

[69] H. R. Borojerdi, and M. Abadi, "MalHunter: Automatic generation of multiple behavioral signatures for polymorphic malware detection," In *ICCKE 2013*, pp. 430-436. IEEE, 2013.

[70] T. A. Assegie, "An Optimized KNN Model for Signature-Based Malware Detection," *International Journal of Computer Engineering In Research Trends (IJCERT)*, pp. 2349-7084, 2021.

[71] N. K. Sreelaja, "Ant Colony Optimization based Light weight Binary Search for efficient signature matching to filter Ransomware," *Applied Soft Computing*, vol. 111, p. 107635, 2021.

[72] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pp. 226-241. IEEE, 2005.

[73] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," In *NSDI*, vol. 10, p. 14. 2010.

[74] P. Liu, W. Wang, X. Luo, H. Wang, and C. Liu, "NSDroid: efficient multi-classification of android malware using neighborhood signature in local function call graphs," *International Journal of Information Security*, vol. 20, no. 1, 2021.

[75] P. Gopalakrishnan, R. S. Narayanan, R. Kamath, and A. Ramani, “Analyzing Diverse Data Mining Techniques to Detect the Malware based on Signature,” *Test Engineering and Management*, vol. 83, pp. 17717-17724, 2020.

[76] J. Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, “Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information,” *computers & security*, vol. 58, pp. 125-138, 2016.

[77] D. Venugopal, and G. Hu, “Efficient signature based malware detection on mobile devices,” *Mobile Information Systems*, vol. 4, no. 1, pp. 33-49, 2008.

[78] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, “AndroSimilar: robust statistical feature signature for Android malware detection,” In *Proceedings of the 6th International Conference on Security of Information and Networks*, pp. 152-159. 2013.

[79] M. Zheng, M. Sun, and J. C. Lui, “Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware,” In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 163-171. IEEE, 2013.

[80] S. Ngamwitroj, and B. Limthanmaphon, “Adaptive Android malware signature detection,” In *Proceedings of the 2018 International Conference on Communication Engineering and Technology*, pp. 22-25. 2018.

[81] Y. Feng, S. Anand, I. Dillig, and A. Aiken, “Apposcopy: Semantics-based detection of android malware through static analysis,” In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pp. 576-587. 2014.

[82] F. Tchakounte, R. C. N. Ngassi, V. C. Kamla, and K. P. Udagepola, “LimonDroid: A system coupling three signature-based schemes for profiling Android malware,” *Iran Journal of Computer Science*, vol. 4, no. 2, pp. 95-114, 2021.

[83] Z. U. Rehman, S. N. Khan, K. Muhammad, J. W. Lee, Z. Lv, S. W. Baik, P. A. Shah, K. Awan, and I. Mehmood, “Machine learning-assisted signature and heuristic-based detection of malwares in Android devices,” *Computers and Electrical Engineering*, vol. 69, pp. 828-841, 2018.

[84] J. Yu, Q. Huang, and C. Yian, “DroidScreening: a practical framework for real-world Android malware analysis,” *Security and Communication Networks*, vol. 9, no. 11, pp. 1435–1449, 2016.

[85] T. Ban, T. Takahashi, S. Guo, D. Inoue, and K. Nakao, "Integration of multimodal features for android malware detection using linear svm," *in Proc. the 11th Asia Joint Conference on Information Security (AsiaJCIS)*, pp. 141-146, Aug. 2016.

[86] S. Morales-Ortega, P. J. Escamilla-Ambrosio, A. Rodriguez-Mota, and L. D. Coronado-De-Alba, "Native malware detection in smartphones with android OS using static analysis, feature selection and ensemble classifiers," *in Proc. the 11th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 1-8, 2016.

[87] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. E. R. T. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," *in Proc. NDSS*, vol. 14, pp. 23-26, 2014.

[88] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of android malware," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 26, no. 3, pp. 11, 2018.

[89] A. Firdaus, N. B. Anuar, A. Karim, and M. F. AbRazak, "Discovering optimal features using static analysis and a genetic search based method for Android malware detection*," Frontiers of Information Technology and Electronic Engineering*, vol. 19, no. 6, pp. 712-736, 2018.

[90] A. Bhattacharya and R. T. Goswami, "A Hybrid Community Based Rough Set Feature Selection Technique in Android Malware Detection," *Smart Trends in Systems, Security and Sustainability*, pp. 249-258, 2018.

[91] X. Yang, D. Lo, L. Li, X. Xia, T. F. Bissyandé, and J. Klein, "Characterizing malicious android apps by mining topic-specific data flow signatures," *Information and Software Technology*, vol. 90, pp. 27-39, 2017.

[92] I. Martin, J. A. Hernandez, A. Munoz, and A. Guzman, "Android malware characterization using metadata and machine learning techniques," *Security and Communication Networks*, 2018.

[93] W. Chen, D. Aspinall, A. D. Gordon, C. Sutton, and I. Muttik, "More semantics more robust: Improving android malware classifiers," *in Proc. The 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 147-158, 2017.

[94] S. Sen, E. Aydogan, and A. I. Aysan, "Coevolution of mobile malware and anti-malware," *IEEE Transactions on Information Forensics and Security*, pp. 2563-2574, 2018.

[95] K. Riad and L. Ke, "Roughdroid: Operative scheme for functional android malware detection," *Security and Communication Networks,* 2018.

[96] P. Palumbo, L. Sayfullina, D. Komashinskiy, E. Eirola, and J. Karhunen, "A pragmatic android malware detection procedure," *Computers and Security,* vol. 70, pp. 689-701, 2017.

[97] Q. Li, B. Sun, M. Chen, and H. Dong, "Detection malicious Android application based on simple-Dalvik intermediate language," *Neural Computing and Applications*, vol. 31, no. 1, pp. 185-194, 2018.

[98] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *computers and security,* vol. 73, no. 1, pp. 326-344, 2018.

[99] L. Chen, C. S. 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,* vol. 12, no. 4, pp. 400-412, 2014.

[100] Y. Xu, C. Wu, K. Zheng, X. Wang, X. Niu, and T. Lu, "Computing adaptive feature weights with PSO to improve android malware detection," *Security and Communication Networks*, 2017.

[101] X. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-Based Malware Detection on Android," *IEEE Transactions on Information Forensics and Security,* vol. 11, pp. 1252-1264, 2016.

[102] D. Hu, Z. Ma, X. Zhang, P. Li, D. Ye, and B. Ling, "The Concept Drift Problem in Android Malware Detection and Its Solution," *Security and Communication Networks,* 2017.

[103] H. J. Zhu, T. H. Jiang, B. Ma, Z. H. You, W. L. Shi, and L. Cheng, "HEMD: a highly efficient random forest-based malware detection framework for Android," *Neural Computing and Applications*, vol. 30, no. 11, pp. 3353-3361, 2018.

[104] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for android malware detection using deep learning," *Digital Investigation,* vol. 24, pp. 48-59, 2018.

[105] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient Android malware detection system based on method-level behavioral semantic analysis," *IEEE Access*, vol. 7, pp. 69246-69256, 2019.

[106] S. Turker and A. B. Can, "AndMFC: Android Malware Family Classification Framework," *In 2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops),* pp. 1-6, 2019.

[107] G. Shrivastava, and P. Kumar, "Android application behavioural analysis for data leakage," *Expert Systems*, 2019.

[108] L. Nguyen-Vu, J. Ahn, and S. Jung, "Android fragmentation in malware detection," *Computers & Security*, vol. 87, p. 101573, 2019.

[109] S. Badhani, and S. K. Muttoo, "CENDroid—A cluster-ensemble classifier for detecting malicious Android applications," *Computers & Security,* vol. 85, pp. 25-40, 2019.

[110] A. Alotaibi, "Identifying Malicious Software Using Deep Residual Long-Short Term Memory," *IEEE Access,* vol. 7, pp. 163128-163137, 2019.

[111] S. Y. Yerima, and S. Khan, "Longitudinal performance analysis of machine learning based Android malware detectors," *In 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security),* pp. 1-8, 2019.

[112] J. McGiff, W. G. Hatcher, J. Nguyen, W. Yu, E. Blasch, and C. Lu, "Towards multimodal learning for android malware detection," *In 2019 International Conference on Computing, Networking and Communications (ICNC),* pp. 432-436, 2019.

[113] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based Android malware detection using Hamming distance of static binary features," *Future Generation Computer Systems,* 105, 230-247, 2020.

[114] F. Mercaldo and A. Santone, "Audio signal processing for Android malware detection and family identification," *Journal of Computer Virology and Hacking Techniques,* vol. 17, no. 2, pp. 139-152, 2021.

[115] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisaan, and Y. Heng, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216-3225, 2018.

[116] H. J. Zhu, Z. H. You, Z. X. Zhu, W. L. Shi, X. Chen, and L. Cheng, "DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model," *Neurocomputing,* vol. 272, pp. 638-646, 2018.

[117] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, “A multimodal deep learning method for Android malware detection using various features,” *IEEE Transactions on Information Forensics and Security*, vol.14, no. 3, pp. 773-788, 2018.

[118] A. Feizollah, N. B. Anuar, R. Salleh, G. S. Tangil, and S. Furnell, “Androdialysis: Analysis of android intent effectiveness in malware detection,” *computers & security*, vol. 65, pp. 121-134, 2017.

[119] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, “Exploring permission-induced risk in android applications for malicious application detection,” *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869-1882, 2014.

[120] M. Alazab, “Automated Malware Detection in Mobile App Stores Based on Robust Feature Generation,” *Electronics*, vol. 9, no. 3, p. 435, 2020.

[121] A. Arora, K. P. Sateesh, and C. Mauro, “Permpair: Android malware detection using permission pairs,” *IEEE Transactions on Information Forensics and Security,* vol. 15, pp. 1968-1982, 2019.

[122] P. Agrawal, and T. Bhushan, “Machine learning classifiers for Android malware detection,” In *Data Management, Analytics and Innovation*, pp. 311-322. Springer, 2021.

[123] D. O. Şahin, O. E. Kural, S. Akleylek, and E. Kılıc, “A novel permission-based Android malware detection system using feature selection based on linear regression,” *Neural Computing and Applications*, pp. 1-16, 2021.

[124] H. Bai, N. Xie, X. Di, and Q. Ye, “Famd: A fast multifeature android malware detection framework, design, and implementation,” *IEEE Access,* vol. 8, pp. 194729-194740, 2020.

[125] H. Yuan, T. Yongchuan, S. Wenjuan, and L. Liu, “A detection method for android application security based on TF-IDF and machine learning,” *Plos one,* vol. 15, no. 9, 2020.

[126] A. Sangal, and H. K. Verma, “A static feature selection-based android malware detection using machine learning techniques,” In *2020 International conference on smart electronics and communication (ICOSEC)*, pp. 48-51. IEEE, 2020.

[127] S. Y. Yerima, S. Sezer, and I. Muttik, “Android malware detection using parallel machine learning classifiers,” In *2014 Eighth international conference on next generation mobile apps, services and technologies*, pp. 37-42. IEEE, 2014.

[128] L. D. Coronado-De-Alba, A. Rodríguez-Mota, and P. J. Escamilla-Ambrosio, “Feature selection and ensemble of classifiers for Android malware detection,” In *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*, pp. 1-6. IEEE, 2016.

[129] S. Y. Yerima, and S. Sezer, “Droidfusion: A novel multilevel classifier fusion approach for android malware detection,” *IEEE transactions on cybernetic,* vol. 49, no. 2 2018, pp. 453-466.

[130] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, “PIndroid: A novel Android malware detection system using ensemble learning methods,” *Computers & Security,* vol. 68, pp. 36-46, 2017.

[131] N. Milosevic, A. Dehghantanha, and K. K. R. Choo, “Machine learning aided Android malware classification,” *Computers & Electrical Engineering,* vol. 61, pp. 266-274, 2017.

[132] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, “Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers,” *Future generation computer systems,* vol. 78, pp. 987-994, 2018.

[133] X. Wang, D. Zhang, X. Su, and W. Li, “Mlifdect: android malware detection based on parallel machine learning and information fusion,” *Security and Communication Networks*, 2017.

[134] D. T. Dehkordy and A. Rasoolzadegan, “A new machine learning-based method for android malware detection on imbalanced dataset,” *Multimedia Tools and Applications,* pp.1-22, 2021.

[135] G. Shrivastava, and P. Kumar, “SensDroid: analysis for malicious activity risk of Android application,” *Multimedia Tools and Applications*, vol. 78, no. 24, pp. 35713-35731, 2019.

[136] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala, “Quantitative security risk assessment of android permissions and applications,” In *IFIP Annual Conference on Data and Applications Security and Privacy,* pp. 226-241. Springer, 2013.

[137] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, “Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version),” *ACM Transactions on Privacy and Security (TOPS),* vol. 22, no. 2, pp. 1-34, 2019.

[138] Y. Ye, L. Wu, Z. Hong, and K. Huang, “A risk classification based approach for android malware detection,” *KSII Transactions on Internet and Information Systems (TIIS),* vol. 11, no. 2, pp. 959-981, 2017.

[139] Y. Xu, C. Wu, K. Zheng, X. Niu, and Y. Yang, “Fuzzy–synthetic minority oversampling technique: Oversampling based on fuzzy set theory for Android malware detection in imbalanced datasets,” *International Journal of Distributed Sensor Networks,* vol. 13, no. 4, 2017.

[140] Y. T. Ling, N. F. M. Sani, M. T. Abdullah, and N. A. W. A. Hamid, “Structural features with nonnegative matrix factorization for metamorphic malware detection,” *Computers & Security,* vol. 104, p. 102216, 2021.

[141] S. Alam, R. N. Horspool, I. Traore, and I. Sogukpinar, “A framework for metamorphic malware analysis and real-time detection,” *computers & security,* vol. 48, pp. 212-233, 2015.

[142] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, “Copperdroid: automatic reconstruction of android malware behaviors,” In *Ndss*. 2015.

[143] M. Bierma, E. Gustafson, J. Erickson, D. Fritz, and Y. R. Choe, “Andlantis: Large-scale Android dynamic analysis,” *arXiv preprint arXiv:1410.7751*, 2014.

[144] M. N. P. Pravin, “Vetdroid: Analysis using permission for vetting undesirable behaviours in android applications,” *International Journal of Innovative and Emerging Research in Engineering,* vol. 2, pp. 31-136, 2015.

[145] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, “Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing,* vol. 20, no. 1, pp. 343-357, 2016.

[146] S. Dai, T. Wei, and W. Zou, “DroidLogger: Reveal suspicious behavior of Android applications via instrumentation,” In *2012 7th international conference on computing and convergence technology (ICCCT),* pp. 550-555. IEEE, 2012.

[147] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, “Mobile malware detection through analysis of deviations in application network behaviour,” *Computers & Security,* vol. 43, pp. 1-18, 2014.

[148] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey on automated dynamic malware-analysis techniques and tools,” *ACM computing surveys (CSUR),* vol. 44, no. 2, pp. 1-42, 2008.

[149] P. S. Chen, S. C. Lin, and C. H. Sun, "Simple and effective method for detecting abnormal internet behaviors of mobile devices," *Information Sciences,* vol. 321, pp. 193-204, 2015.

[150] D. F. Guo, A. F. Sui, Y. J. Shi, J. J. Hu, G. Z. Lin, and T. Guo, "Behavior Classification based Self-learning Mobile Malware Detection," *Journals of Computers,* vol. 9, no. 4, pp. 851-858, 2014.

[151] F. Maggi, M. Matteucci, and S. Zanero, "Detecting intrusions through system call sequence and argument analysis," *IEEE Transactions on Dependable and Secure Computing,* vol. 7, no. 4, pp. 381-395, 2008.

[152] Z. Salehi, A. Sami, and M. Ghiasi, "MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values," *Engineering Applications of Artificial Intelligence,* vol. 59, pp. 93-102, 2017.

[153] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, "MalInsight: A systematic profiling based malware detection framework," *Journal of Network and Computer Applications* vol. 125, pp. 236-250, 2019.

[154] A. Mahindru and A. L. Sangal, "MLDroid—framework for Android malware detection using machine learning techniques," *Neural Computing and Applications,* vol. 33, no. 10, pp. 5183-5240, 2021.

[155] A. Yan, Z. Chen, H. Zhang, L. Peng, Q. Yan, M. U. Hassan, C. Zhao, and B. Yang, "Effective detection of mobile malware behavior based on explainable deep neural network," *Neurocomputing,* vol. 453, pp. 482-492, 2021.

[156] Y. Park, S. R. Douglas, and M. Stamp, "Deriving common malware behavior through graph clustering," *computers & security,* vol. 39, pp. 419-430, 2013.

[157] Ahmadi, Mansour, Ashkan Sami, Hossein Rahimi, and Babak Yadegari. "Malware detection by behavioural sequential patterns," *Computer Fraud & Security,* no. 8, pp. 11-19, 2013.

[158] M. Ghiasi, A. Sami, and Z. Salehi, "DyVSoR: dynamic malware detection based on extracting patterns from value sets of registers," *The ISC International Journal of Information Security,* vol. 5, no. 1, pp. 71-82, 2013.

[159] C. Ravi, and R. Manoharan, "Malware detection using windows api sequence and machine learning," *International Journal of Computer Applications,* vol. 43, no. 17, pp.12-16, 2012.

[160] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security,* vol. 19, no. 4, pp. 639-668, 2011.

[161] H. Cai, N. Meng, B. Ryder, and D. Yao, "Droidcat: Effective android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security,* vol.14, no. 6, pp. 1455-1470, 2018.

[162] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE transactions on information forensics and security,* vol. 11, no. 2, pp. 289-302, 2015.

[163] L. Chen, M. Zhang, C. Y. Yang, and R. Sahita, "Semi-supervised classification for dynamic Android malware detection," arXiv preprint arXiv:1704.05948, 2017.

[164] M. Zheng, M. Sun, and J. C. S. Lui, "DroidTrace: A ptrace based Android dynamic analysis system with forward execution capability," *In 2014 international wireless communications and mobile computing conference (IWCMC),* pp. 128-133. IEEE, 2014.

[165] V. M. Afonso, M. F. D. Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. D. Geus, "Identifying Android malware using dynamically obtained features," *Journal of Computer Virology and Hacking Techniques,* vol. 11, no. 1, pp. 9-17, 2015.

[166] A. Mahindru and P. Singh, "Dynamic permissions based android malware detection using machine learning techniques," In *Proceedings of the 10th innovations in software engineering conference*, pp. 202-210, 2017.

[167] R. Oak, M. Du, D. Yan, H. Takawale, and I. Amit, "Malware detection on highly imbalanced data through sequence modelling," *In Proceedings of the 12th ACM Workshop on artificial intelligence and security,* pp. 37-48, 2019.

[168] Y. Pang, L. Peng, Z. Chen, B. Yang, and H. Zhang, "Imbalanced learning based on adaptive weighting and Gaussian function synthesizing with an application on Android malware detection," *Information Sciences,* vol. 484, pp. 95-112, 2019.

[169] H. Chen, H. F. Leung, B. Han, and J. Su, "Automatic privacy leakage detection for massive android apps via a novel hybrid approach," In *2017 IEEE International Conference on Communications (ICC),* pp. 1-7. IEEE, 2017.

[170] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, and J. Hoffmann, "Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques," *International Journal of Information Security,* vol. 14, no. 2, pp. 141-153, 2015.

[171] Y. Liu, Y. Zhang, H. Li, and X. Chen, "A hybrid malware detecting scheme for mobile Android applications," *In 2016 IEEE International Conference on Consumer Electronics (ICCE),* pp. 155-156. IEEE, 2016.

[172] S. Kandukuru and R. M. Sharma, "Android malicious application detection using permission vector and network traffic analysis," *In 2017 2nd International Conference for Convergence in Technology (I2CT),* pp. 1126-1132. IEEE, 2017.

[173] A. Altaher and O. M. Barukab, "Intelligent hybrid approach for Android malware detection based on permissions and API calls," *International Journal of Advanced Computer Science and Applications,* vol. 8, no. 6, pp. 60-67, 2017.

[174] F. Afifi, N. B. Anuar, S. Shamshirband, and K. K. R. Choo, "DyHAP: Dynamic hybrid ANFIS-PSO approach for predicting mobile malware," *PloS one,* vol. 11, no. 9, 2016.

[175] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep learning based android malware detection using real devices," *Computers & Security* vol. 89, p. 101663, 2020.Z.

[176] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology,* vol. 21, no. 1, pp. 114-123, 2016.

[177] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in Android," *Journal of Parallel and Distributed computing,* vol.103, pp. 22-31, 2017.

[178] T. Bläsing, L. Batyuk, A. D.Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," *In 2010 5th International Conference on Malicious and Unwanted Software,* pp. 55-62. IEEE, 2010.

[179] Z. Fu, Y. Ding, and M. Godfrey, "An LSTM-Based Malware Detection Using Transfer Learning," *Journal of Cybersecurity,* vol. 3, no. 1, p.11, 2021.

[180] Z. H. Qaisar and R. Li, "Multimodal information fusion for android malware detection using lazy learning," *Multimedia Tools and Applications,* pp. 1-15, 2021.

[181] A. T. Kabakus and I. A. Dogru, "An in-depth analysis of Android malware using hybrid techniques," *Digital Investigation,* vol. 24, pp. 25-33, 2018.

[182] J. H. Abawajy and A. Kelarev, "Iterative classifier fusion system for the detection of Android malware," *IEEE Transactions on Big Data,* vol. 5, no. 3, pp. 282-292, 2017.

[183] D. Gupta and R. Rani, “Improving malware detection using big data and ensemble learning,” *Computers & Electrical Engineering*, vol. 86, p. 106729, 2020.

[184] K. Sharma and B. B. Gupta, “Mitigation and risk factor analysis of android applications," *Computers & Electrical Engineering*, vol. 71, pp. 416-430, 2018.

[185] APKMirror (accessed March, 2019). https://www.apkmirror.com/.

[186] Apkpure (accessed March, 2019). https://apkpure.com/.

[187] Virusshare (accessed March, 2019). https://virusshare.com/.

[188] Avira (accessed April, 2019). https://www.avira.com/.

[189] J. Han, J. Pei, and M. Kamber, “Data mining: concepts and techniques,” Elsevier, 2011.

[190] “Xposed module repository.” Xposed Module. (accessed October, 2019) http://repo.xposed.info/module/de.robv.android.xposed.installer.

[191] B. Chizi and O. Maimon, “Dimension reduction and feature selection,” *In Data mining and knowledge discovery handbook*, pp. 83-100. Springer, 2009.

[192] G. Shakhnarovish, T. Darrell, and P. Indyk, “Nearest-neighbor methods in learning and vision,” In *MIT Press*, p. 262, 2005.

[193] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, 1993.

[194] A. Liaw and M. Wiener, “Classification and regression by randomForest,” *R news*, vol. 2, no. 3, pp. 18-22, 2002.

[195] V. Vapnik, E. Levin, and Y. L. Cun, “Measuring the VC-dimension of a learning machine,” *Neural computation*, vol. 6, no. 5, pp. 851-876, 1994.

[196] P. Domingos and M. Pazzani, “On the optimality of the simple Bayesian classifier under zero-one loss,” *Machine learning*, vol.29, no. 2-3, pp. 103-130, 1997.

[197] F. Eibe, I. H. Witten, “Generating Accurate Rule Sets Without Global Optimization,” *In: Fifteenth International Conference on Machine Learning*, pp. 144-151, 1998.

[198] Scikit-Learn Machine Learning in Python. (accessed June, 2019). https://scikit-learn.org/stable/.

[199] F. Rosenblatt, The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

[200] Y. Freund and R. E. Schapire, “Experiments with a new boosting algorithm.” *In icml*, vol. 96, pp. 148-156, 1996.

[201] D. Aldous, “The continuum random tree. I,” *The Annals of Probability,* pp.1-28, 1991.

[202] K. M. Ting and I. H. Witten, “Issues in stacked generalization,” *Journal of artificial intelligence research,* vol.10, pp. 271-289, 1999.

[203] L. I. Kuncheva, A. A. Gonzalez, J. F. D. Pastor, and I. A. D. Gunn, “Instance selection improves geometric mean accuracy: a study on imbalanced data classification,” *Progress in Artificial Intelligence,* vol. 8, no. 2, pp. 215-228, 2019.

[204] J. Liu and E. Zio, “Integration of feature vector selection and support vector machine for classification of imbalanced data,” *Applied Soft Computing,* vol.75, pp. 702-711, 2019.

[205] P. Lopez-Garcia, A. D. Masegosa, E. Osaba, E. Onieva, and A. Perallos, “Ensemble classification for imbalanced data based on feature space partitioning and hybrid metaheuristics,” *Applied Intelligence,* vol. 49, no. 8, pp. 2807-2822, 2019.

[206] H. He and E. A. Garcia, “Learning from imbalanced data,” *IEEE Transactions on knowledge and data engineering,* vol. 21, no. 9, pp. 1263-1284, 2009.

[207] Y. Sun, A. K. C. Wong, and M. S. Kamel, “Classification of imbalanced data: A review,” *International journal of pattern recognition and artificial intelligence,* vol. 23, no. 04, pp. 687-719, 2009.

[208] N. Hatami, Y. Gavet, and J. Debayle, “Classification of time-series images using deep convolutional neural networks,” *In Tenth international conference on machine vision (ICMV 2017),* vol. 10696, p. 106960Y, International Society for Optics and Photonics, 2018.

[209] V. Kumar, R. Kumar, and A. Sharma, “Maintainability prediction from project metrics data analysis using artificial neural network: an interdisciplinary study,” *Middle-East Journal of Scientific Research,* vol. 19, no. 10, pp. 1412-1420, 2014.

[210] R. Kumar, P. S. Grover, and A. Kumar, “A Fuzzy Logic Approach to Measure Complexity of Generic Aspect Oriented Systems,” *Journal of Object Technology,* vol. 9, no. 3, pp. 59-77, 2010.

[211] K. Rai, M. S. Devi, and A. Guleria, “Decision tree based algorithm for intrusion detection,” *International Journal of Advanced Networking and Applications,* vol. 7, no. 4, p. 2828, 2016.

[212] L. H. M. Sepulvene, I. N. Drummond, B. T. Kuehne, R. M. D. Frinhani, F. Petri, S. Reiff-Marganiec, and B. G. Batista, “Analysis of machine learning techniques in fault

diagnosis of vehicle fleet tracking modules," *In 2019 8th Brazilian Conference on Intelligent Systems (BRACIS),* pp. 759-764. IEEE, 2019.

[213] S. Kumari and V. Bhattacharjee, "An indepth experimentation with classifiers for prediction of Diabetes," *International Journal of Engineering Sciences & Research Technology,* vol. 10, no. 6, pp. 28-33, 2021.

[214] V. S. Sheng, B. Gu, W. Fang, and J. Wu, "Cost-sensitive learning for defect escalation," *Knowledge-Based Systems,* vol. 66, pp. 146-155, 2014.

[215] R. Barandela, R. M. Valdovinos, J. S. Sanchez, and F. J. Ferri, "The imbalanced training sample problem: Under or over sampling?," In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR),* pp. 806-814. Springer, 2004.

[216] A. S. Sohal, R. Sandhu, S. K. Sood, and V. Chang, "A cybersecurity framework to identify malicious edge device in fog computing and cloud-of-things environments," *Computers & Security,* vol. 74, pp. 340-354, 2018.

[217] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra, "Risk analysis of Android applications: A user-centric solution," *Future Generation Computer Systems,* vol.80, pp. 505-518, 2018.

[218] K. Sharma, M. K. Ghose, D. Kumar, R. P. K. Singh, and V. K. Pandey, "A comparative study of various security approaches used in wireless sensor networks," *International journal of advanced science and technology,* vol. 17, no. 2, pp. 31-44, 2010.

[219] K. Rai, A. Guleria, and M. Syamala Devi, "Integrated Intrusion Detection Scheme using Agents," *International Journal of Cyber-Security and Digital Forensics,* vol. 9, no. 1, pp. 26-42, 2020.

[220] U. Gupta, S. Gupta, and V. Bhattacharjee, "Intrusion detection system using outlier analysis," *Advancement of Computer Technology and its Applications,* vol. 2, no. 2, 2019.

[221] A. Parchami, M. Nourbakhsh, and M. Mashinchi, "Analysis of variance in uncertain environments," *Complex & Intelligent Systems,* vol. 3, no. 3, pp. 189-196, 2017.

APPENDICES

APPENDIX A

Based on the data analysis, the API calls and permissions are divided into three categories i.e. API calls and permissions that are found only in the benign app sample, API calls and permissions found in both the benign and malicious samples and API calls and permissions found only in the malicious app sample.

Table A.1: API calls found only in benign apps

API calls only in benign apps	
MessengerService	SendMultipartTextMessage
IRemoteService	Runtime.load
Process.start	PathClassLoader
Context.bindService	Ljava.lang.Class.getDeclaredClasses
ACCOUNT_MANAGER	GetBinder

Table A.2: Permissions found only in benign apps

Permissions only in benign apps		
AUTHENTICATE_ACCOUNTS	BIND_APPWIDGET	CHANGE_WIMAX_STATE
NFC	SET_PROCESS_LIMIT	GET_DETAILED_TASKS
BIND_REMOTEVIEWS	BIND_TEXT_SERVICE	GET_INTENT_SENDER_INTENT
READ_PROFILE	INSTALL_LOCATION_PROVIDER	GLOBAL_SEARCH_CONTROL
READ_SYNC_STATS	MOUNT_FORMAT_FILESYSTEMS	INTERACT_ACROSS_USERS_FULL
CAPTURE_VIDEO_OUTPUT	SET_ACTIVITY_WATCHER	MANAGE_DOCUMENTS
SUBSCRIBED_FEEDS_WRITE	BIND_VPN_SERVICE	MANAGE_USERS
CHANGE_WIFI_MULTICAST_STATE	ACCESS_BLUETOOTH_SHARE	MEDIA_CONTENT_CONTROL
MASTER_CLEAR	ACCESS_CACHE_FILESYSTEM	MOVE_PACKAGE
CAPTURE_SECURE_VIDEO_OUTPUT	ACCESS_NOTIFICATION_POLICY	OVERRIDE_WIFI_CONFIG
WRITE_USER_DICTIONARY	BLUETOOTH_PRIVILEGED	PACKAGE_USAGE_STATS
WRITE_PROFILE	BIND_DIRECTORY_SEARCH	READ_INSTALL_SESSION

		S
READ_SOCIAL_STREAM	BIND_DREAM_SERVICE	REAL_GET_TASKS
READ_USER_DICTIONARY	BIND_INCALL_SERVICE	REQUEST_INSTALL_PACKAGES
DUMP	BIND_JOB_SERVICE	SEND_DOWNLOAD_COMPLETED_INTENTS
SET_TIME	BIND_QUICK_SETTINGS_TILE	SEND_RESPOND_VIA_MESSAGE
WRITE_SOCIAL_STREAM	BIND_SCREENING_SERVICE	STATUS_BAR_SERVICE
WRITE_GSERVICES	BIND_TELECOM_CONNECTION_SERVICE	TETHER_PRIVILEGED
SET_TIME_ZONE	WRITE_MEDIA_STORAGE	TRANSMIT_IR
BIND_ACCESSIBILITY_SERVICE	BIND_VOICE_INTERACTION	UPDATE_APP_OPS_STATS
ADD_VOICEMAIL		

Table A.3: API calls found in both malware and benign apps

Common API calls in both malicious and benign apps	
AbortBroadcast	Transact
Ljava.lang.Class.getResource	BindService
TelephonyManager.getSimSerialNumber	Ljava.lang.Class.getCanonicalName
KeySpec	OnServiceConnected
SendDataMessage	AttachInterface
DivideMessage	android.os.Binder
android.os.IBinder	Ljava.net.URLDecoder
TelephonyManager.getLine1Number	Ljava.lang.Class.getMethods
SET_ALARM	ServiceConnection
HttpPost.init	Ljava.lang.Class.getField
ProcessBuilder	Landroid.content.Context.registerReceiver
Ljava.lang.Class.getClasses	ClassLoader
Ljava.lang.Class.forName	Landroid.content.Context.unregisterReceiver
HttpUriRequest	FindClass
TelephonyManager.getSimCountryIso	Ljava.lang.Class.getDeclaredField
OnBind	

Table A.4: Permissions found in both malware and benign apps

Common permissions in both malicious and benign apps		
SEND_SMS	HARDWARE_TEST	READ_CONTACTS
WRITE_SMS	ACCESS_WIFI_STATE	BIND_DEVICE_ADMIN
RECEIVE_SMS	ACCESS_SURFACE_FLINGER	KILL_BACKGROUND_PROCESSES
READ_SYNC_SETTINGS	ACCESS_FINE_LOCATION	CALL_PRIVILEGED
BATTERY_STATS	CAPTURE_AUDIO_OUTPUT	SET_PREFERRED_APPLICATIONS
WRITE_HISTORY_BOOKMARKS	INJECT_EVENTS	CHANGE_NETWORK_STATE
INSTALL_PACKAGES	GET_ACCOUNTS	BROADCAST_WAP_PUSH
READ_HISTORY_BOOKMARKS	USE_CREDENTIALS	FLASHLIGHT
INTERNET	MANAGE_ACCOUNTS	SYSTEM_ALERT_WINDOW
DISABLE_KEYGUARD	READ_SMS	GET_PACKAGE_SIZE
WRITE_APN_SETTINGS	CAMERA	READ_FRAME_BUFFER
RECEIVE_BOOT_COMPLETED	WRITE_SYNC_SETTINGS	WRITE_EXTERNAL_STORAGE
RESTART_PACKAGES	BIND_NOTIFICATION_LISTENER_SERVICE	BROADCAST_SMS
UPDATE_DEVICE_STATS	MODIFY_AUDIO_SETTINGS	CHANGE_COMPONENT_ENABLED_STATE
ACCOUNT_MANAGER	BROADCAST_STICKY	WRITE_SETTINGS
SET_WALLPAPER_HINTS	WAKE_LOCK	WRITE_CONTACTS
SET_WALLPAPER	BLUETOOTH	ACCESS_MOCK_LOCATION
SET_DEBUG_APP	READ_CALENDAR	MODIFY_PHONE_STATE
BIND_WALLPAPER	READ_EXTERNAL_STORAGE	EXPAND_STATUS_BAR
READ_PHONE_STATE	VIBRATE	SET_ORIENTATION
ACCESS_COARSE_LOCATION	ACCESS_NETWORK_STATE	BLUETOOTH_ADMIN
RECEIVE_MMS	SUBSCRIBED_FEEDS_READ	DEVICE_POWER
CONTROL_LOCATION_UPDATES	WRITE_CALENDAR	INTERACT_ACROSS_USERS
CALL_PHONE	GET_TASKS	PERSISTENT_ACTIVITY

READ_LOGS	GLOBAL_SEARCH	WRITE_SECURE_SETTINGS
MOUNT_UNMOUNT_FILESYSTEMS	REORDER_TASKS	ACCESS_DOWNLOAD_MANAGER
CHANGE_CONFIGURATION	BIND_INPUT_METHOD	ACCESS_WIMAX_STATE
CHANGE_WIFI_STATE		

Table A.5: API calls found only in malicious apps

API calls only in malware apps	
PackageInstaller	DefineClass
Ljava.lang.Class.getPackage	GetCallingUid
Runtime.exec	TelephonyManager.getCallState

Table A.6: Permissions found only in malicious apps

Permissions only in malware apps	
READ_INPUT_STATE	FORCE_STOP_PACKAGES
SIGNAL_PERSISTENT_PROCESSES	SET_ANIMATION_SCALE
SET_ALWAYS_FINISH	FACTORY_TEST
BROADCAST_PACKAGE_REMOVED	BACKUP
READ_OWNER_DATA	READ_CALL_LOG
WRITE_CALL_LOG	STATUS_BAR
DELETE_PACKAGES	DELETE_CACHE_FILES
CLEAR_APP_CACHE	RECEIVE_WAP_PUSH
CLEAR_APP_USER_DATA	PROCESS_OUTGOING_CALLS
ACCESS_LOCATION_EXTRA_COMMANDS	RECORD_AUDIO
REBOOT	INTERNAL_SYSTEM_WINDOW

APPENDIX B

This section shows the percentage of API calls and permissions present in Android apps belonging to different risk categories.

Table B.1: High risk API calls

API call	Malware	API call	Malware
PackageInstaller	6.70%	DefineClass	0.52%
Ljava.lang.Class.getPackage	15.28%	GetCallingUid	5.44%
Runtime.exec	26.39%	TelephonyManager.getCallState	3.55%

Table B.2: High risk permissions

Permission	Malware	Permission	Malware
READ_INPUT_STATE	0.06%	FORCE_STOP_PACKAGES	3.15%
SIGNAL_PERSISTENT_PROCESSES	0.06%	SET_ANIMATION_SCALE	0.06%
SET_ALWAYS_FINISH	0.06%	FACTORY_TEST	0.34%
BROADCAST_PACKAGE_REMOVED	3.15%	BACKUP	0.40%
READ_OWNER_DATA	0.57%	READ_CALL_LOG	1.09%
WRITE_CALL_LOG	1.32%	STATUS_BAR	1.95%
DELETE_PACKAGES	4.29%	DELETE_CACHE_FILES	2.23%
CLEAR_APP_CACHE	1.72%	RECEIVE_WAP_PUSH	2.92%
CLEAR_APP_USER_DATA	0.80%	PROCESS_OUTGOING_CALLS	4.12%
ACCESS_LOCATION_EXTRA_COMMANDS	11.22%	RECORD_AUDIO	2.40%
REBOOT	0.52%	INTERNAL_SYSTEM_WINDOW	0.06%

Table B.3: Medium risk API calls

API call	Malware	Benign	API call	Malware	Benign
AbortBroadcast	0.20%	0.07%	divideMessage	0.06%	0.02%
Ljava.lang.Class.getResource	0.40%	0.39%	android.os.IBinder	0.92%	0.85%
TelephonyManager.getSimSerialNumber	0.18%	0.10%	TelephonyManager.getLine1 Number	0.42%	0.15%
SendDataMessage	0.04%	0.01%	SET_ALARM	0.03%	0.02%

Table B.4: Medium risk permissions

Permission	Malware	Benign	Permission	Malware	Benign
SEND_SMS	54.84%	6.11%	BIND_WALLPAPER	4.58%	2.50%
WRITE_EXTERNAL_STORAGE	68.00%	64.22%	READ_HISTORY_BOOKMARKS	17.74%	4.06%
RECEIVE_SMS	38.29%	7.17%	ACCESS_COARSE_LOCATION	31.94%	27.06%
ACCESS_WIFI_STATE	43.39%	41.83%	RECEIVE_MMS	3.26%	1.44%
WRITE_SMS	22.15%	4.22%	CONTROL_LOCATION_UPDATES	0.63%	0.44%
WRITE_HISTORY_BOOKMARKS	16.26%	2.83%	CALL_PHONE	13.97%	11.06%
INSTALL_PACKAGES	14.25%	2.89%	READ_LOGS	8.59%	7.94%
ACCESS_FINE_LOCATION	30.57%	29.94%	MOUNT_UNMOUNT_FILESYSTEMS	4.41%	2.61%
INTERNET	95.94%	82.17%	CHANGE_CONFIGURATION	3.26%	2.33%
READ_PHONE_STATE	89.01%	49.39%	CHANGE_WIFI_STATE	18.20%	15.78%
WRITE_APN_SETTINGS	9.79%	1.61%	DISABLE_KEYGUARD	8.59%	7.78%
RECEIVE_BOOT_COMPLETED	48.94%	32.67%	GET_PACKAGE_SIZE	1.89%	1.67%
RESTART_PACKAGES	13.91%	4.61%	CAPTURE_AUDIO_OUTPUT	0.57%	0.28%
UPDATE_DEVICE_STATS	2.75%	0.44%	HARDWARE_TEST	0.34%	0.22%
INJECT_EVENTS	0.06%	0.06%	READ_SMS	37.32%	6.94%
SET_WALLPAPER	10.07%	5.61%			

Table B.5: Low risk API calls

API call	Benign	Malware	API call	Benign	Malware
Ljava.lang.Class.forName	0.71%	0.47%	AttachInterface	0.63%	0.06%
ProcessBuilder	0.11%	0.07%	android.os.Binder	0.70%	0.27%
ServiceConnection	0.66%	0.08%	Landroid.content.Context.unregisterReceiver	0.57%	0.17%
OnBind	0.72%	0.64%	Ljava.lang.Class.getMethods	0.43%	0.14%
KeySpec	0.58%	0.54%	Ljava.lang.Class.getCanonicalNa	0.50%	0.05%

			me		
TelephonyManager.getSimCountryIso	0.22%	0.11%	Ljava.lang.Class.getField	0.60%	0.25%
OnServiceConnected	0.67%	0.24%	Landroid.content.Context.registerReceiver	0.66%	0.34%
HttpUriRequest	0.70%	0.58%	ClassLoader	0.67%	0.43%
Ljava.lang.Class.getClasses	0.07%	0.02%	Ljava.net.URLDecoder	0.58%	0.12%
HttpPost.init	0.63%	0.50%	FindClass	0.09%	0.02%
Transact	0.64%	0.06%	Ljava.lang.Class.getDeclaredField	0.62%	0.23%
BindService	0.66%	0.24%			

Table B.6: Low risk permissions

Permission	Benign	Malware	Permission	Benign	Malware
GET_ACCOUNTS	43.17%	8.24%	WRITE_SECURE_SETTINGS	5.28%	3.84%
USE_CREDENTIALS	17.00%	0.06%	PERSISTENT_ACTIVITY	1.78%	0.29%
MANAGE_ACCOUNTS	16.44%	0.17%	CHANGE_NETWORK_STATE	11.44%	6.47%
READ_SYNC_SETTINGS	13.00%	0.46%	BROADCAST_WAP_PUSH	0.67%	0.11%
CAMERA	19.00%	4.24%	FLASHLIGHT	3.28%	1.55%
WRITE_SYNC_SETTINGS	12.28%	0.63%	SYSTEM_ALERT_WINDOW	8.11%	5.27%
BIND_NOTIFICATION_LISTENER_SERVICE	3.33%	0.06%	BIND_DEVICE_ADMIN	2.56%	0.29%
MODIFY_AUDIO_SETTINGS	8.94%	1.26%	READ_FRAME_BUFFER	1.00%	0.46%
BROADCAST_STICKY	8.33%	0.52%	CHANGE_COMPONENT_ENABLED_STATE	1.00%	0.69%
WAKE_LOCK	53.11%	38.18%	BROADCAST_SMS	1.17%	0.40%
BLUETOOTH	10.22%	4.01%	READ_CONTACTS	24.33%	23.58%
READ_CALENDAR	5.72%	1.14%	KILL_BACKGROUND_PROCESSES	5.61%	2.80%
READ_EXTERNAL_STORAGE	11.94%	6.18%	WRITE_CONTACTS	12.44%	9.85%
VIBRATE	40.50%	30.17%	ACCESS_MOCK_LOCATION	1.67%	1.03%

ACCESS_NETWORK_STATE	76.33%	65.66%	MODIFY_PHONE_STATE	2.17%	2.00%
SUBSCRIBED_FEEDS_READ	3.00%	0.17%	ACCESS_SURFACE_FLINGER	0.33%	0.06%
WRITE_CALENDAR	5.06%	1.20%	SET_ORIENTATION	0.83%	0.23%
GET_TASKS	20.11%	13.57%	EXPAND_STATUS_BAR	2.50%	1.72%
GLOBAL_SEARCH	3.11%	1.32%	DEVICE_POWER	1.78%	1.37%
REORDER_TASKS	2.17%	0.06%	SET_DEBUG_APP	0.17%	0.06%
BIND_INPUT_METHOD	1.56%	0.11%	SET_PREFERRED_APPLICATIONS	1.06%	0.80%
CALL_PRIVILEGED	1.44%	0.06%	INTERACT_ACROSS_USERS	0.33%	0.06%
BATTERY_STATS	3.22%	0.92%	ACCESS_DOWNLOAD_MANAGER	0.78%	0.46%
WRITE_SETTINGS	17.89%	12.36%	ACCESS_WIMAX_STATE	0.22%	0.11%
SET_WALLPAPER_HINTS	1.67%	1.49%	ACCOUNT_MANAGER	0.17%	0.06%
BLUETOOTH_ADMIN	6.11%	3.49%			

Table B.7: No risk API calls

API call	Benign	API call	Benign
MessengerService	0.01%	sendMultipartTextMessage	0.03%
IRemoteService	0.01%	Runtime.load	0.03%
Process.start	0.01%	PathClassLoader	0.04%
Context.bindService	0.01%	Ljava.lang.Class.getDeclaredClasses	0.04%
ACCOUNT_MANAGER	0.01%	GetBinder	0.31%

Table B.8: No risk permissions

Permission	Benign	Permission	Benign
AUTHENTICATE_ACCOUNTS	10.28%	BIND_DIRECTORY_SEARCH	0.06%
NFC	6.06%	BIND_DREAM_SERVICE	0.22%
BIND_REMOTEVIEWS	5.89%	SUBSCRIBED_FEEDS_WRITE	3.06%
READ_PROFILE	5.72%	BIND_JOB_SERVICE	45.39%
READ_SYNC_STATS	6.22%	BIND_QUICK_SETTINGS_TILE	2.61%
WRITE_MEDIA_STORAGE	0.94%	BIND_SCREENING_SERVICE	0.11%

BIND_INCALL_SERVICE	0.22%	BIND_TELECOM_CONNECTION_SERVICE	0.06%
CHANGE_WIFI_MULTICAST_STATE	2.22%	CAPTURE_SECURE_VIDEO_OUTPUT	0.17%
MASTER_CLEAR	2.11%	BIND_VOICE_INTERACTION	0.06%
CAPTURE_VIDEO_OUTPUT	0.17%	BLUETOOTH_PRIVILEGED	0.06%
WRITE_USER_DICTIONARY	1.67%	CHANGE_WIMAX_STATE	0.22%
WRITE_PROFILE	1.56%	GET_DETAILED_TASKS	0.06%
READ_SOCIAL_STREAM	1.28%	GET_INTENT_SENDER_INTENT	0.06%
ADD_VOICEMAIL	0.33%	GLOBAL_SEARCH_CONTROL	0.06%
DUMP	1.61%	INTERACT_ACROSS_USERS_FULL	1.00%
SET_TIME	1.28%	MANAGE_DOCUMENTS	1.11%
WRITE_SOCIAL_STREAM	1.11%	MANAGE_USERS	0.33%
WRITE_GSERVICES	0.94%	MEDIA_CONTENT_CONTROL	1.06%
SET_TIME_ZONE	0.56%	MOVE_PACKAGE	0.11%
BIND_ACCESSIBILITY_SERVICE	0.50%	OVERRIDE_WIFI_CONFIG	0.06%
READ_USER_DICTIONARY	1.33%	PACKAGE_USAGE_STATS	2.89%
INSTALL_LOCATION_PROVIDER	0.39%	READ_INSTALL_SESSIONS	0.06%
SET_PROCESS_LIMIT	0.28%	REAL_GET_TASKS	0.50%
BIND_TEXT_SERVICE	0.17%	REQUEST_INSTALL_PACKAGES	4.28%
BIND_APPWIDGET	0.89%	SEND_DOWNLOAD_COMPLETED_INTENTS	0.22%
MOUNT_FORMAT_FILESYSTEMS	0.56%	SEND_RESPOND_VIA_MESSAGE	0.94%
SET_ACTIVITY_WATCHER	0.33%	STATUS_BAR_SERVICE	0.17%
BIND_VPN_SERVICE	0.17%	TETHER_PRIVILEGED	0.06%
ACCESS_BLUETOOTH_SHARE	0.11%	TRANSMIT_IR	0.28%
ACCESS_CACHE_FILESYSTEM	0.11%	UPDATE_APP_OPS_STATS	0.11%
ACCESS_NOTIFICATION_POLICY	2.17%		

LIST OF PUBLICATIONS

Journals

Published

1. M. Dhalaria, and E. Gandotra, "A Hybrid Approach for Android Malware Detection and Family Classification," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 6, no. 6, 2021, pp. 174-188. DOI: http://dx.doi.org/10.9781/ijimai.2020.09.001 **(SCIE Indexed, IF= 3.137)**

2. M. Dhalaria, and E. Gandotra, "CSForest: an approach for imbalanced family classification of android malicious applications," *International Journal of Information Technology*, vol. 13, no. 3, 2021, pp. 1-13. DOI: https://doi.org/10.1007/s41870-021-00661-7 **(SCOPUS Indexed)**

3. M. Dhalaria and E. Gandotra, "Android malware detection techniques: a literature review," Recent Patents on Engineering, vol. 15, no. 2, 2021, pp. 225-245. DOI: https://doi.org/10.2174/1872212114999200710143847 **(SCOPUS Indexed)**

4. M. Dhalaria and E. Gandotra, "Binary and Multi-class Classification of Android Applications using Static Features," *International Journal of Applied Management Science*. 2021 **(Accepted, SCOPUS Indexed)**

Communicated

1. M. Dhalaria and E. Gandotra, "MalDetect A Classifier Fusion Approach for Detection of Android Malware," *Expert Systems with Applications*. **(SCIE Indexed, IF=6.954)**

2. M. Dhalaria and E. Gandotra, "Quantitative Threat Assessment of Android Applications using Permissions and API Calls," *Multimedia Tools and Applications.* **(SCIE Indexed, IF=2.757)**

Conferences

Published

1. M. Dhalaria, E. Gandotra, and S. Saha, "Comparative Analysis of Ensemble Methods for Classification of Android Malicious Applications," *In International Conference on Advances in Computing and Data Sciences*, pp. 370-380. Springer, Singapore, 2019. DOI: https://doi.org/10.1007/978-981-13-9939-8_33.

2. M. Dhalaria, and E. Gandotra, "A Framework for Detection of Android Malware using Static Features," *In 2020 IEEE 17th India Council International Conference (INDICON)*, pp. 1-7. IEEE, 2020. DOI: 10.1109/INDICON49873.2020.9342511.

3. M. Dhalaria, and E. Gandotra, "Android Malware Detection using Chi-Square Feature Selection and Ensemble Learning Method," *In 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pp. 36-41. IEEE, 2020. DOI: 10.1109/PDGC50313.2020.9315818.

4. M. Dhalaria, and E. Gandotra, "Risk Detection of Android Applications using Static Permissions," *In International Conference on Advances in Data Computing, Communication and Security (I3CS)*, pp. 591–600. Springer, 2021. DOI: https://doi.org/10.1007/978-981-16-8403-6_54.

Book Chapter

Communicated

1. M. Dhalaria, and E. Gandotra, "Analysis of Feature Selection Methods for Detection of Android Malware," *In Deep Learning Empowered Security in Cyber Physical Systems*.