PEOPLES DEMOCRATIC REBUPLIC OF ALGERIA

MMINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCHE

UNIVERSITY MOHAMED BOUDIAF - M'SILA

FACULTY: MATHIMATICS AND COMPUTER SCIENCE

DEPARTEMENT: COMPUTER SCIENCE

N° :……..

DOMAIN: MATHIMATICS AND COMPUTER SCIENCE

BRANCH: COMPUTER SCIENCE

OPTION: RTIC

Dissertation submitted to obtain Master degree

By:

- **SOUFI ABOUBAKER SEDDIK**
- **BOUAZIZ AZEDDINE**

SUBJECT

Mobile malware detection

Publicly defended before the jury composed of

Ms .SAOUDI LALIA	University of M'sila	Supervisor
Mr. BENOUIS MOHAMED	University of M'sila	Chair
Mr. ATTIR AZZEDDINE	University of M'sila	Examiner

Academic year 2021/2022

Acknowledgment

First of all, we would like to thank Allah for giving us the strength and courage to carry out our project.

We would like to sincerely thank our supervisor Ms "SAOUDI LALIA" who was always attentive and very available throughout the realization of this project, as well as for the inspiration, the help and the time he kindly granted us. May she find in this work a living tribute to her lofty personality.

Our thanks also go to the honourable members of the jury, Mr "BENOUIS MOHAMED" and Mr "ATTIR AZEDDINE" who agreed to evaluate this modest work.

Finally, we would like to express our most sincere thanks to those who, from near or far, who have helped us and participated in the development of this work.

SOUFI ABOUBAKER SEDDIK / BOUAZIZ AZEDDINE

Abstract

With the advent of communication technologies and the extended availability of smart devices (smartphones, PDAs), mobiles applications are becoming part of our everyday life, due to its portability, availability and high performance services to cover most user's needs. Like all web applications, mobile system is exposed to malware attacks.

In this context, our project is aimed at developing a mobile malware detector (MobMal detector) on Android platform using machine learning approach to detect malware by mining the patterns of Permissions and API Calls.

Keywords:

Mobile application 's malware, Android Permission, API Calls, Malware Apps, Support Vector Machines.

ملخص

مع تقدم تكنولوجيا الاتصالات وانتشار الاجهزة الذكية، اصبحت التطبيقات الذكية المتنقلة جزءا لا يتجزأ من حياتنا اليومية. نظرًا لإمكانية النقل والتوافر والخدمات عالية الأداء التى تغطي معظم احتياجات المستخدمين. مثل جميع تطبيقات الويب ، يتعرض نظام الهاتف المحمول لـ هجمات البرمجيات الخبيثة.

في هذا السياق، يهدف مشروعنا إلى تطوير تطبيق للأجهزة الهاتف الذكية التي تعمل على نظام الأندرويد يعتمد على نهج التعلم الآلي (ماشين ليرنينغ) للكشف عن البرامج الضارة باستخدام أنماط الأذونات ومكالمات واجهة برمجة التطبيقات، ويعتمد على التحليل الثابت للكود المصدري وملفات الموارد لتطبيقات الأندرويد.

الكلمات المفتاحية:

تطبيق أندرويد الضار، أذونات الاندرويد، مكالمات واجهة برمجة التطبيقات ، التطبيقات الضارة، آلة المتجهات الداعمة.

Résumé:

Avec l'avènement des technologies de communication et la disponibilité étendue des mobiles (smartphones, PDA), les applications mobiles font désormais partie de notre quotidien, du fait à sa portabilité, disponibilité et les services de haute performance à couvrir la plupart des besoins des utilisateurs. Comme toutes les applications Web, le système mobile est exposé à attaques de logiciels malveillants.

Dans ce contexte, notre projet vise à développer un détecteur de malware mobile (MobMal détecteur) sur la plate-forme Android utilisant une approche de machine learning pour détecter les logiciels malveillants en exploitant le modèles de Permissions and API Calls.

Mots clés:

Logiciels malveillants d'applications mobiles, autorisation Android, appels API, applications malveillantes, assistance Machines vectorielles.

Table of content

List of figures and tables

List of figures

List of tables

GENERAL INTRODUCTION

Currently, most of the current daily tasks are entwined inexorably with our cell phones. In the last decade, Android kept its leading position as the world's widely used mobile operating system, Moreover, along with the continual development of Android OS, new and complex varieties of malware are appearing which cause malware detection to be increasingly challenging. A common feature of Malware is that they are specifically designed to damage, disrupt, steal, or in general inflict some other bad or illegitimate actions. Malware can literally infect any computing machines running user programs (or applications).

With the increase the use of the Android system and the emergence of millions of applications, the number of malicious applications has increased alarmingly and has become difficult to identify. The objective of our work is design and create a mobile malware detector under Android which we called "MobMal" to detect malicious apps.

In this study, we use a static analysis paradigm based on machine learning mechanism using API calls and permission features for detecting the Android malware.

Work plan:

We have four chapters:

Chapter 1: The first chapter is entitled "Android system and malware application" contains an overview of the Android system, the structures and components of Android applications, and finally, the malware applications.

Chapter 2: The second chapter is entitled "Methods used to detect mobile malware applications" In this chapter, we define the three types of malicious applications approaches and present some previous works in each type.

Chapter 3: The third chapter is entitled "Mobile Malware Detection based on Machine learning with permissions and API Calls as features "which is the core of our work and contains a detailed description of our application and its various components.

Chapter 4: The fourth chapter is entitled "Implementation and experimentation" in which we will define the development tools that we used. We will also illustrate some interfaces of the developed application.

Finally, we end with a general conclusion containing the results of our system and some perspectives for our future work.

CHAPTER 01

Android system and malware application

Introduction:

Based on the Linux kernel, Google developed an operating system for mobiles. They called it Android. The Android operating system is primarily designed for smartphone devices which implement a touch screen input interface. Not only that, it has also been developed for many devices such as smart watches (Android Wear), tablet computers and cars (Android Auto).

Android is known for its OS touch inputs that correspond to real-world actions such as swiping, tapping, pinching and reverse pinching among many mobile operating systems, Android is the most popular operating system, which is competing with IOS for Apple devices and Windows Phones. Android is the most used system in smartphones in the world for the year 2022 with 69.74% [1], the figure 1.1 shows the logo of the Android system.

Figure 1.1: Android System [2]

1.1. Architecture of android operating system:

Android operating system is a stack of software components. Main components of Android Operating system Architecture or Software Stack are Linux kernel, native libraries, Android Runtime, Application Framework and Applications, The figure 1.2 shows the Architecture of the Android system from the upper layer, "Applications", to the lower of the "Linux Kernel" layer.

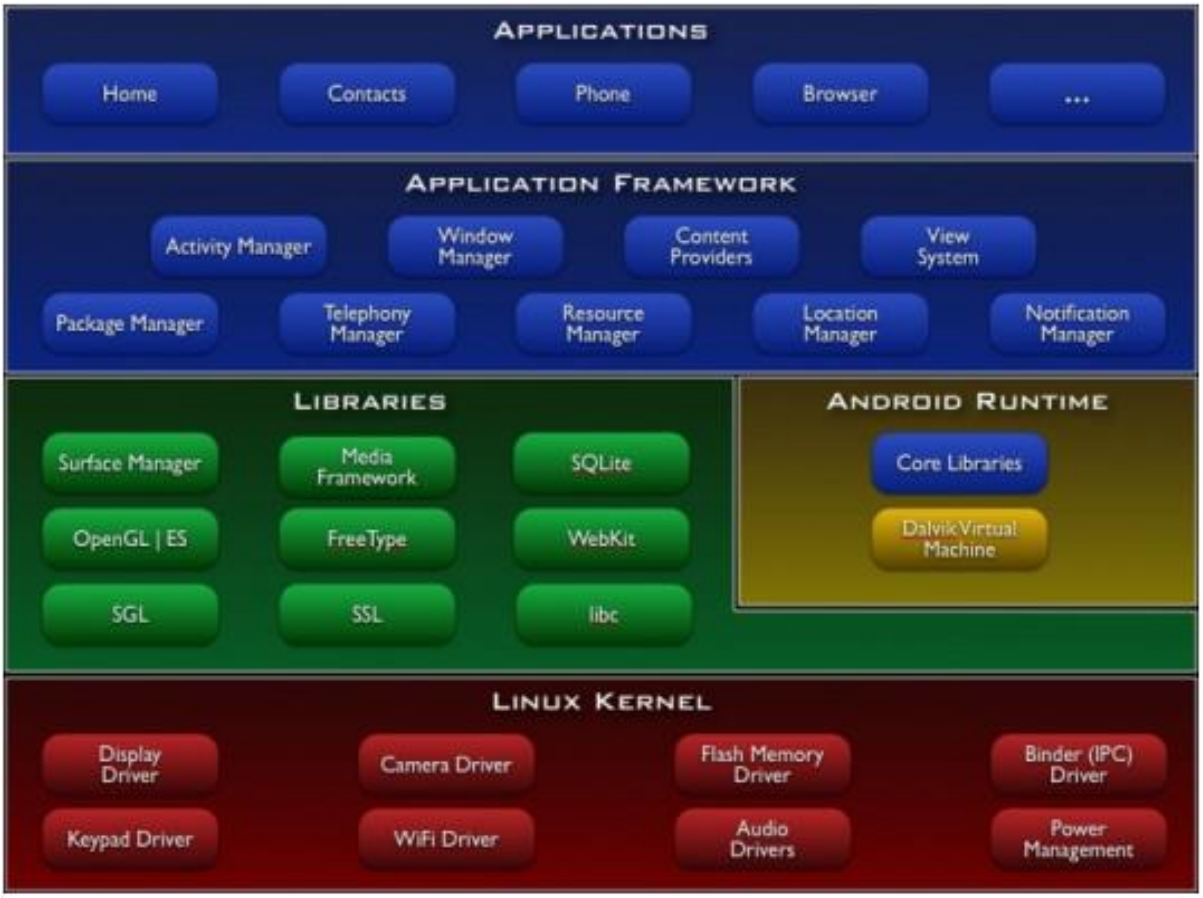

Figure 1.2: Architecture of Android [3]

1.1.1. Linux Kernel:

Kernel layer provides basic system functionalities like process management, memory management, device management including camera, display, keypad etc. Basically kernel used in android operating system is a Linux 2.6 series kernel. Reason of choosing Linux kernel for android OS is that Linux is really good at basic operation such as networking, vast array of device drivers which take the pain out of interfacing to peripheral hardware. For some special needs Google enhanced kernel to better address the needs of mobile platforms like memory management, Process management, power management, runtime environment special interprocess communication (IPC) mechanism and better handling of limited system resources [4].

1.1.2. Native Libraries Layer:

On the top of the Linux Kernel layer is Android's native libraries.

This layer enables the device to handle different types of data. Data is specific to hardware. All these libraries are written in C or C++ language. These libraries are called through java interface. Some important native libraries are:

Surface Manager: it is used to manage display of device. Surface Manager used for composing windows on the screen.

SQLite: SQLite is the database used in android for data storage. It is a relational database and available to all applications.

WebKit: It is the browser engine used to display HTML content.

Media framework: Media framework provides playbacks and recording of various audio, video and picture formats. (For example MP3, AAC, AMR, JPG, MPEG4, H.264, and PNG).

Free Type: Bitmap and Font Rendering OpenGL | ES: Used to render 2D or 3D graphics content to the screen.

Libc: It contains System related C libraries.

1.1.3. Android Runtime:

Android Runtime consists of Dalvik Virtual machine and Core Java libraries. It is located on the same level as the library layer. Dalvik Virtual Machine is a type of Java Virtual Machine used for running applications on Android device. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. The Dalvik VM allows multiple instance of Virtual machine to be created simultaneously providing security, isolation, memory management and threading support.

Unlike Java VM which is process-based, Dalvik Virtual Machine is register base. Dalvik Virtual Machine run ".dex" files which are created from .class file by dx tool."Dx" tool is included in Android SDK. DVM is optimized for low processing power and low memory environments. DVM is developed by Dan Bornstein from Google [5].

1.1.4. Application Framework:

The Application Framework layer provides many higher-level services or major APIs to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications. These are the blocks with which developer's applications directly interact. Important blocks of Application framework are:

Activity Manager: It manages the life cycle of applications.

Content Providers: It is used to manage the data sharing between applications, manages how to access data from other applications.

Telephony Manager: it manages all voice call related functionalities.

Location Manager: It is used for Location management, using GPS or cell tower.

Resource Manager: Manage the various types of resources used in Application [6].

1.1.5. Application Layer:

The Applications Layer is the top layer in the Android architecture. Some applications come preinstalled with every device, such as: SMS client app, Dialer, Web browser and Contact manager. A developer can write his own application and can replace it with the existing application.

1.2. Android Application:

This section concerns the description of the structure of an Android application, the description of components included in an Android application, and tools used to dissemble and to implement Android applications.

Figure 1.3: Android Apps

1.2.1. Application Structure and Reversing Techniques:

We give an overview of the application structure and the contents of the corresponding package as well as a short introduction into the tools used for unpacking and decompiling of Android applications. Android Package (APK) files are unencrypted archives, which contain all the necessary files needed by the application to function. An APK contains the following files and folders:

- ✓ META-INF: It is the directory which contains the following files:
- ✓ MANIFEST.MF: The Manifest fie‹
- ✓ CERT.RSA: The application certificate‹
- ✓ CERT.SF: The list of resources and SHA-1 digest.
- ✓ Lib: This is the directory, which includes the compiled code specific to a software layer of a processor:
- ✓ armeabi: The compiled code for arm based processors‹
- ✓ armeabi-v7a: The compiled code for armv7 and above based processors‹
- ✓ x86: The compiled code for x86 processors‹
- ✓ mips: The compiled code for MIPS processors.
- ✓ Resources.arsc: This fie contains pre-compiled resources.
- ✓ Res: This directory contains resources not compiled into resources.arsc.
- ✓ AndroidManifest.xml: This is the Manifest fie. It includes meta information of the application and describes the application capabilities.
- ✓ Classes.dex: The compiled classes in the Dalvik Executable (DEX) file format.

The figure 1.4 shows the steps of building an Android application from the Java file to the APK file:

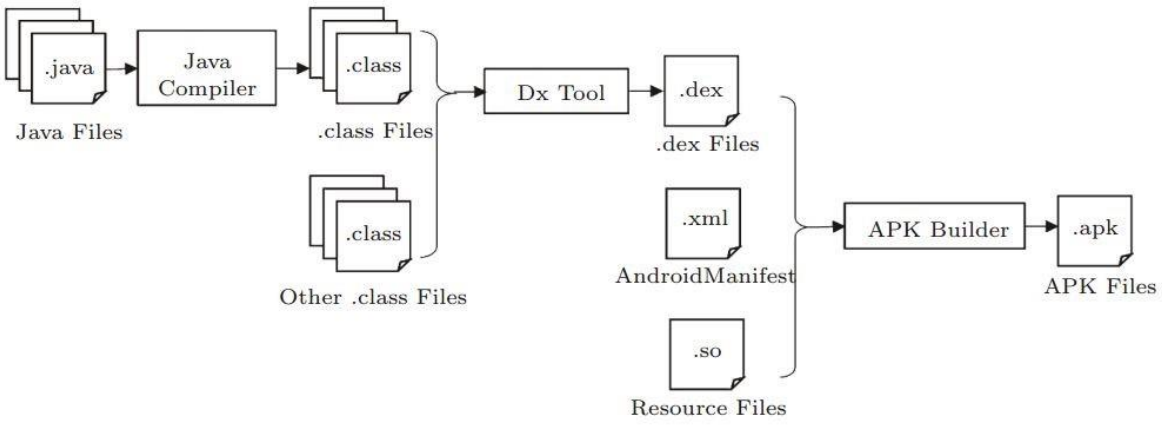

Figure 1.4: Android app building process [7]

Android applications themselves are mainly written in Java with support for their own native libraries written in C. The Java source code gets compiled to a DVM executable byte code, which is stored in a DEX file.

The Manifest, which is very important to execute the application, is called AndroidManifest.xml. It contains all permissions, listeners, receivers and the Meta information of the application.

The DEX file, the Manifest, all resources, the certificates and own libraries for the application are packaged to a ZIP archive file with the APK. This APK file is provided via the Google Play to the users. The source code of an Android application is not available in clear text when unpacking an APK file.

Tools are often needed for an analysis to get the source code, which is as close as possible to the original code. The tool dex2jar decompiles the Dalvik Virtual Machine (DVM) byte code to the Java byte code, which is easily readable with a graphical Java decompile like JD-GUI. Unfortunately, dex2jar has some limitations and is sometimes not able to retrieve the corresponding Java byte code. A second tool for disassembly is used for this reason: Baksmali. This tool disassembles DVM byte code to a new language called Smali, which is easily readable and which can be reassembled to an Android application.

We illustrate the source code of a simple “hello Android” application and the different disassembles in Figure 1.5 (a). The Smali output of the main Java class is shown in Figure 1.5 (b). The disassembly of IDA Pro is illustrated in Figure 1.5 (c) for a more comprehensive overview:

```
public class MainActivity extends Activity {
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       TextView tv = new TextView(this);
       tv.setText("Hello, Android");
       setContentView(tv);
   }
}
```

Figure 1.5 (a) Original source code

```
.class public Lcom/example/helloandroid/MainActivity;
.super Landroid/app/Activity;
.source "MainActivity.java"

# direct methods
.method public constructor <init>()V
    .registers 1
    .prologue
    .line 7
    invoke-direct {p0}, Landroid/app/Activity;-><init>()V
    return-void
.end method

# virtual methods
.method public onCreate(Landroid/os/Bundle;)V
    .registers 4
    .parameter "savedInstanceState"
    .prologue
    .line 11
    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
    .line 12
    new-instance v0, Landroid/widget/TextView;
    invoke-direct {v0, p0}, Landroid/widget/TextView;-><init>(Landroid/content/Context;)V
    .line 13
    .local v0, tv:Landroid/widget/TextView;
    const-string v1, "Hello, Android"
    invoke-virtual {v0, v1}, Landroid/widget/TextView;->setText(Ljava/lang/CharSequence;)V
    .line 14
    invoke-virtual {p0, v0}, Lcom/example/helloandroid/MainActivity;->setContentView(Landroid/view/View;)V
    .line 15
    return-void
.end method
```

Figure 1.5 (b) SMALI disassembly

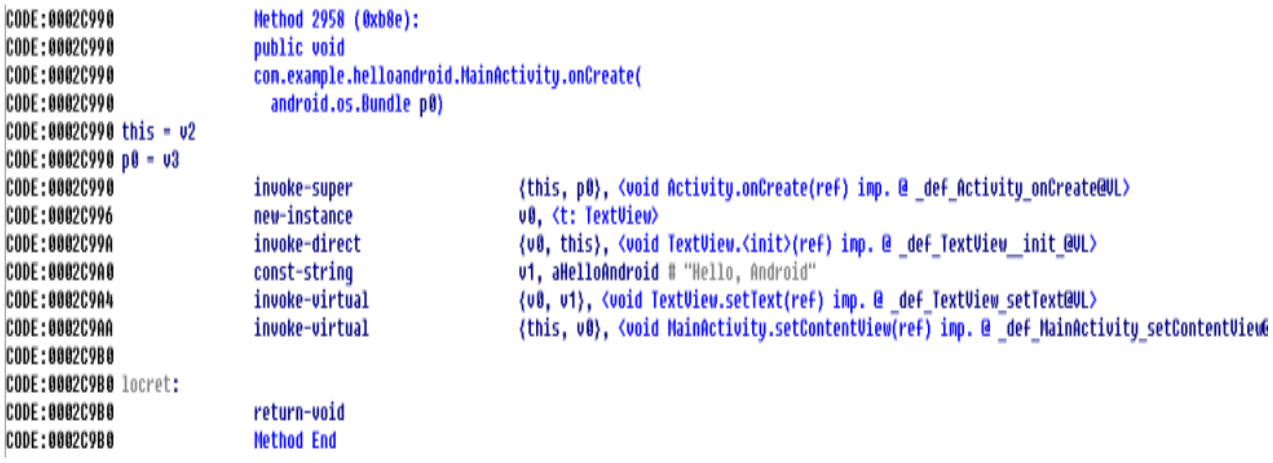

Figure 1.5 (c) IDA Pro disassembly

Figure 1.5(a, b and c): Application's source code and different disassembles [8]

1.2.2. Development:

The Android ecosystem is based on two pillars:

- ✓ The Java programming language،
- ✓ Software Development Kit (SDK), a tool box for the developer.

The development kit gives access to examples, documentation, an API, and an emulator to test applications. The plug-in Android Development Tool (ADT) integrates functionalities of the SDK. The developer installs it as a classical plug-in by specifying its URL. The next step consists of defining precisely the SDK location (already uploaded and unzipped) in the ADT preferences.

1.3. Application Malware:

1.3.1. Definition:

A malicious application or malware refers to an application that can be used to compromise an operation of a device, steal data, bypass access controls or otherwise cause damage on the host terminal.

Normal or benign applications or good software are in contrast those that do not perform any dangerous action on the system.

Application malware is malicious software on the Android platform [9].

1.3.2. Malware Life Cycle:

Their life cycle is structured around seven main phases:

Creation: stage during which the programmer designs and implements all the malicious code that will be included in the malware.

Gestation: Step during which the malicious application infiltrates and installs itself in the system it wants to infect. It remains inactive throughout this stage. This is why its presence remains completely unknown to the user.

Reproduction or infection: The malware reproduces a significant number of times before manifesting itself in this phase. The malware author seeks to remotely control devices and access private data. Malware spreads via file sharing or social engineering techniques on Android. It uses SMS, Bluetooth, Wi-Fi as a means of communication.

Activation: Some malware activates its destruction routine when certain conditions are met. Activation can also be done remotely. The purpose of this phase is to gradually appropriate all the resources of the device.

Discovery: The user notices strange behavior and suspects the presence of a malicious application. This strange behavior may include loss of performance, current changes in the homepage of the web browser, or the unavailability of certain system functions. Antiviruses often help the user detect malicious actions by sending alerts to the device owner. However, the stealth nature of some malware can prolong or even complicate this phase.

Assimilation: Antiviruses update their virus database after discovering new malware. If possible, a fi or antidote is also offered to eliminate this threat.

Elimination: This is the phase where the antivirus discovering the malware invites the user to delete it. It marks the death of the malware.

1.3.3. Types of Application Malware:

Cybercriminals use various tactics to infect mobile devices. If you're focused on improving your mobile malware protection, it's important to understand the different types of mobile malware threats. Here are some of the most common types:

Remote Access Tools (RATs): Remote Access Tools offer extensive access to data from infected victim devices and are often used for intelligence collection. RATs can typically access information such as installed applications, call history, address books, web browsing history, and SMS data. RATs may also be used to send SMS messages, enable device cameras, and log GPS data.

Bank trojans: Bank trojans are often disguised as legitimate applications and seek to compromise users who conduct their banking business — including money transfers and bill payments — from their mobile devices. This type of Trojan aims to steal financial login and password details.

Ransomware: Ransomware is a type of malware used to lock out a user from their device and demand a "ransom" payment — usually in untraceable Bitcoin. Once the victim pays the ransom, access codes are provided to allow them to unlock their mobile device.

Cryptomining Malware: Cryptomining Malware enables attackers to covertly execute calculations on a victim's device – allowing them to generate cryptocurrency. Cryptomining is often conducted through Trojan code that is hidden in legitimate-looking apps.

Advertising Click Fraud: Advertising Click Fraud is a type of malware that allows an attacker to hijack a device to generate income through fake ad clicks.

Conclusion:

In this chapter, we have seen the Architecture of the Android system and its components, Android applications and their development tools, malicious applications and their illegal uses. As a summary, we can retain the following elements:

- ✓ Android is the most popular mobile system.
- ✓ Mobile applications have many advantages.
- ✓ Mobile apps can be used for spying and other illegal purposes.

In the next chapter, we will discuss the methods used to detect malicious applications.

CHAPTER 02:

Methods used to detect mobile malware applications

Introduction:

In this chapter, we will see the types of mobile application analysis used in the Android system (dynamic analysis, static analysis, hybrid analysis) with some previous works and we will make a comparison between them.

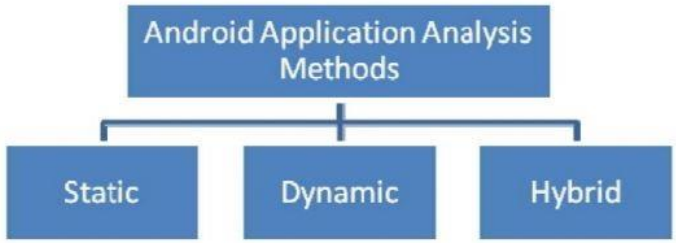

Figure 2.1: Android Application Analysis Methods [10]

Most of the methods depend on disassembling the application and extracting its files for analysis, we show in Figure 2.2 below the stages of disassembly and work on the application:

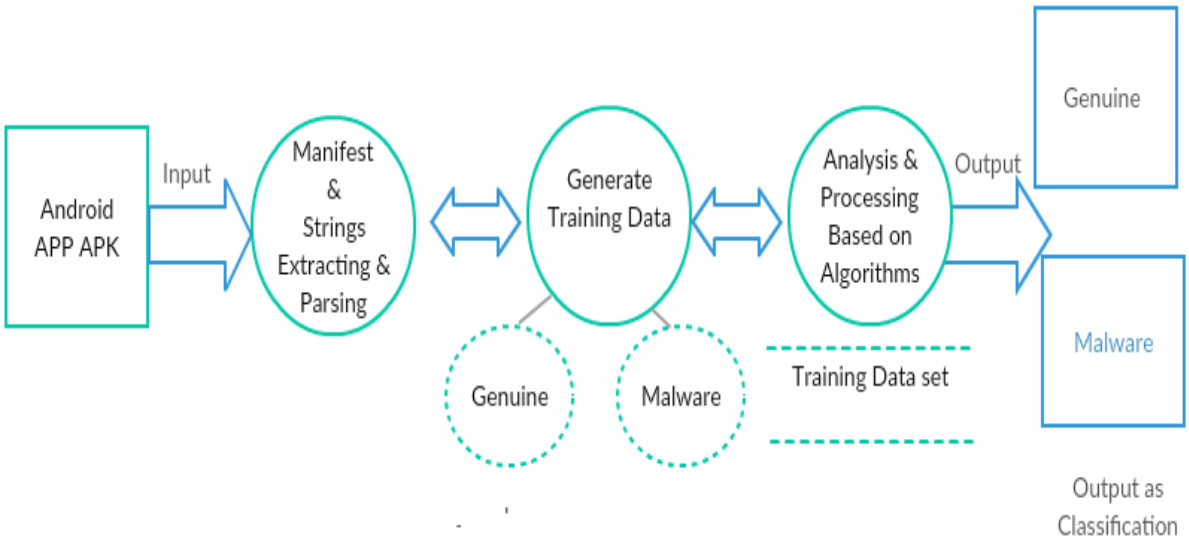

Figure 2.2: General architecture for statically Android malware detection

2.1. Static analysis:

The static analysis is the way to find malicious characteristics or bad code segments in an app without executing it. It is generally used in preliminary malware detection, when suspicious applications are first evaluated in order to find out any obvious security threats. Since it flags an app as malicious according to an over-approximation of its possible runtime behavior. As a consequence, any static analysis method must maximize effective detection while minimizing the risk of false positives.

2.1.1. Methods:

- TinyDroid:

TinyDroid is a static malware detection system for Android applications that is based on a two-step process: it first abstracts machine instructions, followed by a machine learning phase. A set of n-grams is computed for each app under consideration, and compared to the set of ngrams extracted from apps that are known to be either benign or malicious. created by T. Chen, Q. Mao, Y. Yang, M. Lv, J. Zhu, in 2018 using Opcodes as features and Random Forest,Naive Bayes, SVM,KNN as classifications methods with dataset estimated at 2400 App from Google Play,Drebin dataset, experimental results have shown that TinyDroid exhibits a high level of accuracy reaching 95% , and it's very effective [11].

- NSDroid:

It is a static malware detection system for Android apps based on the apps' similarity to known malware, By creating a graph for each application and comparing it with the graph of the malicious applications that it uses in its dataset, created by P. Liu, W. Wang, X. Luo, H. Wang, C. Liu, in 2020, using Function call graphs as features and SVM, Random Forest, Decision Tree as classifications methods with dataset estimated at 32190 App from Drebin dataset , the method is also highly effective, reaching 100% accuracy [12].

- DroidMOSS:

DroidMOSS is a static malware detection system for Android applications its relies upon the existence of the corresponding original applications in the data set to systematically detect and analyze repackaged apps its identifies the source of the applications, either from

the official Android Market or third party markets, and measures the similarity between pairs of application from the same market in order to detect repackaged applications. DroidMoss analysis is based on the totality of the code originating in every component of the app, created by W. Zhou, Y. Zhou, X. Jiang, P. Ning in 2012 using Dalvik bytecode as features with dataset estimated at 2400 App from Free apps, however it can be made more effective by disregarding inputs that originate in other components [13].

- DroidSieve:

DroidSieve collects static features that is include the list of API calls occurring in the code and permissions, and the set of all application components, then this data is entered into the classification algorithms. created by G. Suarez-Tangil, S.K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, L. Cavallaro, in 2017, using Metadata of the app, syntactic as features and Extra Trees, SVM, Random Forests, XGBoost as classifications methods with dataset estimated at 100000 App from Malgenome Project, Drebin dataset, PRAGuard dataset, McAfee Goodware , achieving a detection rate of 99.44%, with zero false positives. DroidSeive is also capable of classifying malware with high accuracy with tis wide dataset [14].

- Kirin:

Kirin is a static malware detection system for android applications, which examines the permissions requested by an app in order to determine if it meets a higher-level security policy, after extracts the permissions from the manifest file. It then compares these permissions to collection of rules, defined by the authors that conservatively overestimate templates of undesirable security properties needed by several types of common malware. If the configuration fails to pass all rules, the installation program can reject the application; if not the user may decide to proceed with the installation anyway, Created by W. Enck, M. Ongtang, P. McDaniel, in 2009 using Permissions as features, Kirin was tested on 311 apps downloaded from the official Android market [15].

2.1.2. Comparison between Previous static analysis methods:

We will see in the table 2.1 a summary of the previous static methods to compare the accuracy, rate, false positive, and the features used…..etc. To check the effectiveness of each static method.

Tool	Year	Rate				Database		Technique	Features	M. Learning
		FP	A	P	R	Nb	From			
NSDroid	2020	< 1%	95%	96%	95%	32190	Drebin,	Calculate the similarity between apps	Function call graphs	SVM, Random Forest, Decision Tree
TinyDroid	2018	5%	95%	92%	95%	2400	Google Play Drebin dataset	Disassembled APK file into smali codes	Opcodes	Random Forest, Naive Bayes, SVM, kNN
DroidSieve	2017	L	99%	99%	99%	100000	Malgenome Project, Drebin dataset, PRAGuard dataset, McAfee Goodware	Static analysis of resources	Metadata of the app, syntactic features	Extra Trees, SVM, Random Forests, XGBoost
DroidMOSS	2012	–	–	–	–	2400	Free apps	Similarity measure	Dalvik bytecode	–
Kirin	2009	–	–	–	–	311	Google Play	Verify perm. upon installation	Permissions	–

Table 2.1: Recapitulative table for Static methods [16]

2.1.3. Static code analysis advantages:

- ✓ It can find weaknesses in the code at the exact location.
- ✓ It can be conducted by trained software assurance developers who fully understand the code.
- ✓ It allows a quicker turn around for fixes.
- ✓ It is relatively fast if automated tools are used.
- ✓ Automated tools can scan the entire code base.
- ✓ Automated tools can provide mitigation recommendations, reducing the research time.

- ✓ It permits weaknesses to be found earlier in the development life cycle, reducing the cost to fix

2.1.4. Static code analysis limitations:

- ✓ It is time consuming if conducted manually.
- ✓ Automated tools do not support all programming languages.
- ✓ Automated tools produce false positives and false negatives.
- ✓ There are not enough trained personnel to thoroughly conduct static code analysis.
- ✓ Automated tools can provide a false sense of security that everything is being addressed.
- ✓ Automated tools only as good as the rules they are using to scan with.
- ✓ It does not find vulnerabilities introduced in the runtime environment.

2.2. Dynamic analysis:

Dynamic analysis involves executing mobile apps in a running environment, such as a virtual machine or emulator, so that researchers can monitor the app's dynamic running behaviors. Researchers primarily applied dynamic analysis in taint tracking or system call tracing. Unlike static analysis, it is late in that it only detects a violation right at the moment when it is about to occur.

2.2.1. Methods:

- ➢ Natural language processing:

It is a dynamic method that relies on creating and training two classifiers to sequence system calls from both malware and benign applications. Then the sequence of system calls to the application to be examined is compared with each of the two classifiers, and it is determined whether it is benign or malware according to the similarity with one of the classifiers, created by X. Xiao, S. Zhang, F. Mercaldo, G. Hu, A.K. Sangaiah, in 2019 using System calls as features and LSTM (Long Short-Term Memory)as classification method with dataset estimated at 7130 App from Google Play, Drebin dataset , the authors achieved an accuracy rate of 93, 7%, the model has been tested under different conditions and this method is effective[17].

- Crowdroid:

Crowdroid is a dynamic method that extracts all system calls from the running applications and sends them to a server, whose role is the classification to malware or benign, the latter creates a feature vector for each pair of user and application this feature vector is a listing of the number of times each of Linux's 250 system calls is called, created by I. Burguera, U. Zurutuza, S. Nadjm-Tehrani, in 2011 using System calls as features and k-means as classifications method with dataset estimated at 5 App from Google Play, Crowdroid was tested on 3 malware achieved between 85% and 100% detection rates. [18].

- RepassDroid:

RepassDroid is a tool adopts dynamic approach focuses on system-level information to detect malicious apps. It analyzes the Android application by synthesizing the API used in the application as a semantic function and the essential permissions as a syntactic function then, it uses learning to automatically determine whether an application is benign or malicious. Created by N. Xie, F. Zeng, X. Qin, Y. Zhang, M. Zhou, C. Lv, in 2018 using API, permissions as features and Decision Tree, Random Forest, SVM, KNN, Naive Bayes as classifications methods with database estimated at 24288 App (half were benign and the other half malicious) from Google PlayAndrozoo, Malgenome Project, VirusShare, Drebin dataset, reaching 97% accuracy[19].

- XManDroid:

TinyDroid is a security tool depends on dynamic malware detection system for Android apps that is based on monitoring the communications that occur between the different components, dynamically analyzes application permission usage to detect and prevent privilege escalation attacks at runtime. Created by S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, in 2011 using Perm., Reference Monitor, Decision Maker, System View, System Policy and Decisions as features with database estimated at 50 App from Google Play, sing 50 benign apps taken from the Android market showed that XManDroid exhibits a low false positive rate 3% out of 3824 ICC calls were flagged as false positives. [20].

2.2.2. Comparison between Previous dynamic analysis methods:

We will also see in this table a summary of the previous dynamic methods, which is to compare the accuracy, rate, false positive, and the features used…..etc. To check the effectiveness of each dynamic method.

Tool	Year	Rate				Database		Technique	Features	M. Learning
		FP	A	P	R	Nb	From			
Crowdroid	2011	–	100%	–	100%	5	Google Play	System call monitoring	System calls	k-means
XManDroid	2011	3%	–	–	–	50	Google Play	Monitoring of Perm. usage	Perm., Reference Monitor, Decision Maker, System View, System Policy and Decisions	–
Natural language processing	2019	9%	93%	91%	96%	7130	Google Play, Drebin dataset	System call trace	System calls	LSTM (Long Short-Term Memory)
RepassDroid	2018	< 1%	97%	99%	96%	24288	Google Play, Androzoo, Malgenome Project, VirusShare, Drebin dataset	Combines API calls and semantic information	API, permissions	Decision Tree, Random Forest, SVM, kNN, Naive Bayes

Table 2.2: Recapitulative table for Dynamic methods [21]

2.2.3. Dynamic code analysis advantages:

- ✓ It identifies vulnerabilities in a runtime environment.
- ✓ Automated tools provide flexibility on what to scan for.

- ✓ It allows for analysis of applications in which you do not have access to the actual code.
- ✓ It identifies vulnerabilities that might have been false negatives in the static code analysis.
- ✓ It permits you to validate static code analysis findings.
- ✓ It can be conducted against any application.

2.2.4. Dynamic code analysis limitations:

- ✓ Automated tools provide a false sense of security that everything is being addressed.
- ✓ Automated tools produce false positives and false negatives.
- ✓ Automated tools are only as good as the rules they are using to scan with.
- ✓ There are not enough trained personnel to thoroughly conduct dynamic code analysis [as with static analysis].
- ✓ It is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.

2.3. Hybrid analysis:

Hybrid analysis is a combination of static analysis and dynamic analysis to overcome the limitations of both and combine their advantages together.

- ➢ Toward extracting malware features for classification using static and dynamic analysis:

It is a hybrid method that identifies malware features using an approach called malware DNA (Mal-DNA). The heart of this technique is a debugging-based behavior monitor and analyzer that extracts dynamic characteristics. Created by Choi, Y.H. et al, in 2012 [22].

- ➢ HDM Analyzer:

HDM Analyzer uses hybrid analysis in the training phase, but performs only static analysis in the testing phase. The goal is to take advantage of the supposedly superior fidelity of dynamic analysis in the training phase, while maintaining the efficiency advantage of static detection in the scoring phase. It is shown that HDM Analyzer has better overall accuracy and time complexity than the static or dynamic analysis methods. The dynamic analysis is based on extracted API call sequences. Created by Eskandari, M., Khorshidpour, Z., Hashemi, S. in 2013 [23].

Conclusion:

Static analysis is lightweight as compared to dynamic analysis as there is no need of Android application execution and run time monitoring of the application .Feature extraction from APK file and selection of subset of features is important for effective detection of Android malware, it can succeed for android even when confronted with obfuscation techniques such as reflection, encryption and dynamically-loaded native code. While fundamental changes in characteristics of malware remain a largely open problem.

On the other hand Dynamic analysis is characterized by revealing the real behavior of the program processing closer to the actual situation. But it is not widely used on account of a large number of resources and the slow detection speed while running the program.

In the next chapter, we explain the mechanism of our detector and select the features with API calls and Permissions.

CHAPTER 03:

Mobile Malware Detection based on Machine learning with permissions and API Calls as features

Introduction:

Dynamic analysis and static analysis are both effective and have their pros and cons. As we mentioned earlier, addressing the real behavior of the application in real time requires resources and high effort, so static analysis is effective in terms of extracting characteristics and classifying them accurately to give a satisfactory result. It's must maximize effective detection while minimizing the risk of false positives, especially if the examination method depends on the characteristics of the Permissions and API Calls as features.

The accuracy rate using this method in the previous works has reached up to 98%, of course using the wide and accurate Dataset, so the examination in this way is effective towards detecting malwares in android app.

After reviewing the previous works for detecting malwares with deferent methods, we suggest our strategy for developing our "Static analysis" detector in Android platform based on permissions and API Calls as features.

Our Mobile Malware Detector (MobMal) applies machine learning techniques in detecting malicious apps on Android environment.

First we need to make our work clear and simple in 4 major parts and some sub-parts, the MobMal detector considers the application features, including permissions and API calls, to characterize Android applications behaviors, the first part is an app analyzer that decompresses the APK file of an App and extracts AndroidManifest.xml and class files, which are necessary for characterizing Apps, the second part aims to characterize each App based on its requested permissions and API calls, the third part, feature generator, carries out feature extraction, which includes permission features extracted from AndroidManifest and API call features extracted from class files, each App is represented as a single instance with binary permission and API call features, and a class label indicates whether the App is a benign or a malicious, the last part includes the training of the classification models from the collected data. After we get Apps converted into generic instance feature format, one can simply use any learning algorithm to derive classification model from the training data. With classification method, which is Support Vector Machines (SVM).

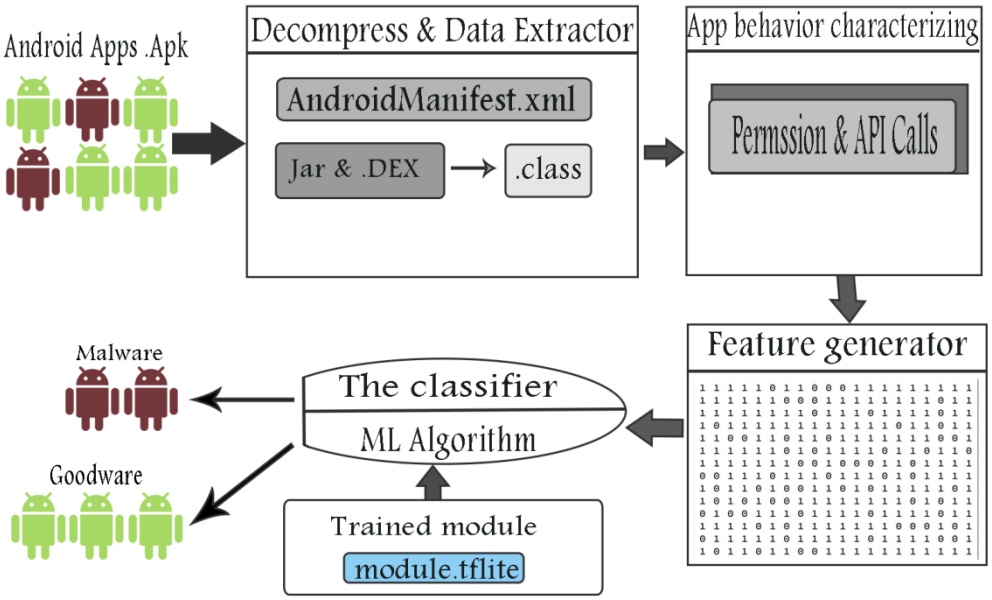

Figure 3.1: The proposed mobile malware detector MobMal

3.1. Features Definition:

We first summarize and propose 2 feature categories which are permissions and API calls according to the existing work to investigate their corresponding performances with the machine learning algorithm.

3.1.1. Android permissions:

Android is a privilege-separated system, which employs mandatory access control over all processes. Each Android application has a unique identity in the system, which comprises user ID and group ID. Android system use a default denial approach to control application executions, which means all operations are denied unless a specific permission is explicitly granted. A collection of permissions are defined and aligned to cover a variety of system resources. Each application must declare permissions in a manifest file in order to use corresponding APIs. The declaration of permissions is statistically done, so cannot be done dynamically at runtime.

Uses-Permissions:

The other are provided permissions where an application itself allows other applications entry to its data. The uses-permissions are stated permissions which the application wants to have from the Android system and are provided by the user at installation time. An Android application must state all preferred uses permissions inside the AndroidManifest.xml file. An example of a uses-permission is READ_CONTACTS, which if given, allows for reading all the contacts in the address book.

3.1.2. Android API Calls:

The Android platform provides a framework API that Apps can use to interact with the underlying Android system. The framework API consists of a core set of packages and classes. Because most Apps use a large number of APIs, it motivates us to use API calls of each application as feature to characterize and differentiate malware from benign Apps. To achieve the goal, one can create a framework to reverse engineer APK file and extract API calls of each application. After that each application is represented as a binary vector of API calls, namely A, where Ai = 1 if and only if the i-th API is used in the application and Ai = 0 if corresponding application does not use the API. The table 3.1 presents the list of malicious permissions and API calls [24].

Features type	Malicious Strings
Permissions	- android.permission.ACCESS_NETWORK_STATE - android.permission.INTERNET - android.permission.READ_PHONE_STATE - android.permission.ACCESS_WIFI_STATE - android.permission.WRITE_EXTENAL_STORAGE - android.permission.READ_SMS - android.permissions.ACCESS_COARSE_LOCATION android.permission.WRITE_SMS - android.permission.SEND_SMS - android.permission. RECEIVE_SMS …….66. (With sub-features extracted from our experiment).

API Calls	- android.content.pm.PackageInfo - TelephonyManager.GetDeviceId - android.intent.action.BOOT_COMPLETED - Java.lang.Object.getClass() - TelephonyManager.getSubscriberId() - Mount - Java.lang.Class.getResource - System.LoadLibrary - Runtime.GetRuntime - KeySpec ……203. (With sub-features extracted from our experiment).

Table 3.1: List of the main features

3.2. Feature extraction:

The first principal step in order to detect malwares is to extract features which the classification procedure will be based on, these features are permissions and API calls. And this step can be made by extracting features from the source code of the application without executing it, using the Manfiestfile.xml and the DEX file as we do in our project.

3.2.1. Android Permissions and API Calls Settings:

Android app is provided as a packed APK file, which contains the compiled binary files, XML files, like manifest file, and other resources, etc. In data processing progress, to get the features from the APK file, we first decode the package to separate files by using ApkTool [25] among the decoded application source files, we can extract three different feature types, which are manifest properties, API calls from the raw application data and class files. According to the different contents in each feature, we presented two different vector embedding methods to generate the inputs of the Support vector machines (SVM).

The AndroidManifest.xml file contains the permissions the application requested from the system, but the claimed permission may or may not be used by the application. If an

application request excess permission, which is known as overprovide problem, the application may be utilized to carry out some malicious activities after installation. As important features to characterize applications, we extract requested permissions from the manifest file to reveal the capability of using system resources.

3.3. Feature selection:

Feature selection phase is very important step in order to get efficient classification model due to the influence of the input data on building a strong classifier in addition to evade long execution time and even computational resources consuming, For this reason several approaches proposed new solutions to reduce feature sets dimensions and to select most informative and relevant features that increase the performance of the machine learning model and better classify an application to a specific class(benign or malware).

At the system level, Google announced that a security checking mechanism is applied to each application uploaded to their market. The open design of the Android operating system allows a user to install any applications downloaded from an untrusted source. Nevertheless, the permission list is still the minimal defense for a user to detect whether an application could be harmful. That way, users can choose to not install an App if they see the App unnecessarily requests permission to user's personal contact (e.g. phone book).

Google also categorizes Android permissions into four threat level:

- ✓ Normal permission: include lower-risk permissions which control access to API calls that are not particularly harmful, the system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval like SET ALARM.
- ✓ Dangerous permission: regulate access to the potential harmful API calls that would give access to private user data. For example, permissions to read the location of a user ACCESS FINE LOCATION or WRITE CONTACTS are classified as dangerous.

- ✓ Signature permission: protect access to the most dangerous privilege. The system grants the permission only if the requesting application is signed with the same certificate as the application that declared the permission.
- ✓ Signature/System permission: A permission that the system grants only to applications that are in the Android system.

To demonstrate that permission settings are indeed relevant to the behaviors of benign and malware, we have the top permissions requested by these malicious Apps in Figure 3.3 and top permissions requested by benign Apps Figure 3.2.

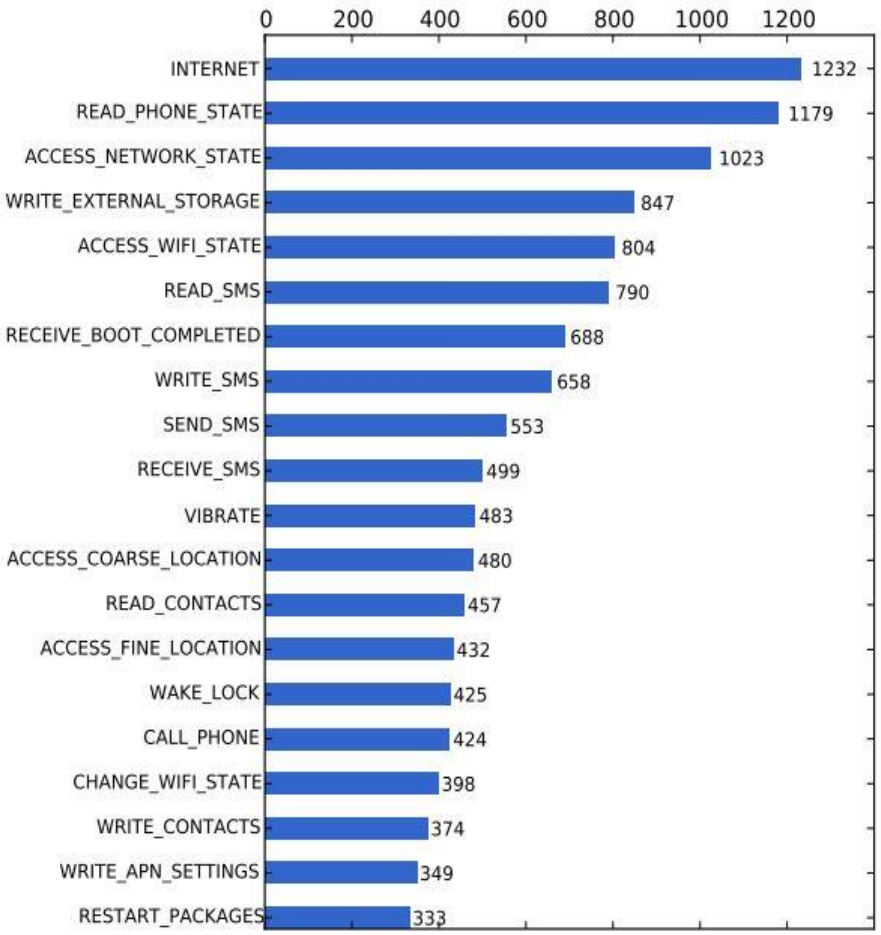

Figure 3.2: Top 20 Permissions Requested By 1260 Malware Samples [26]

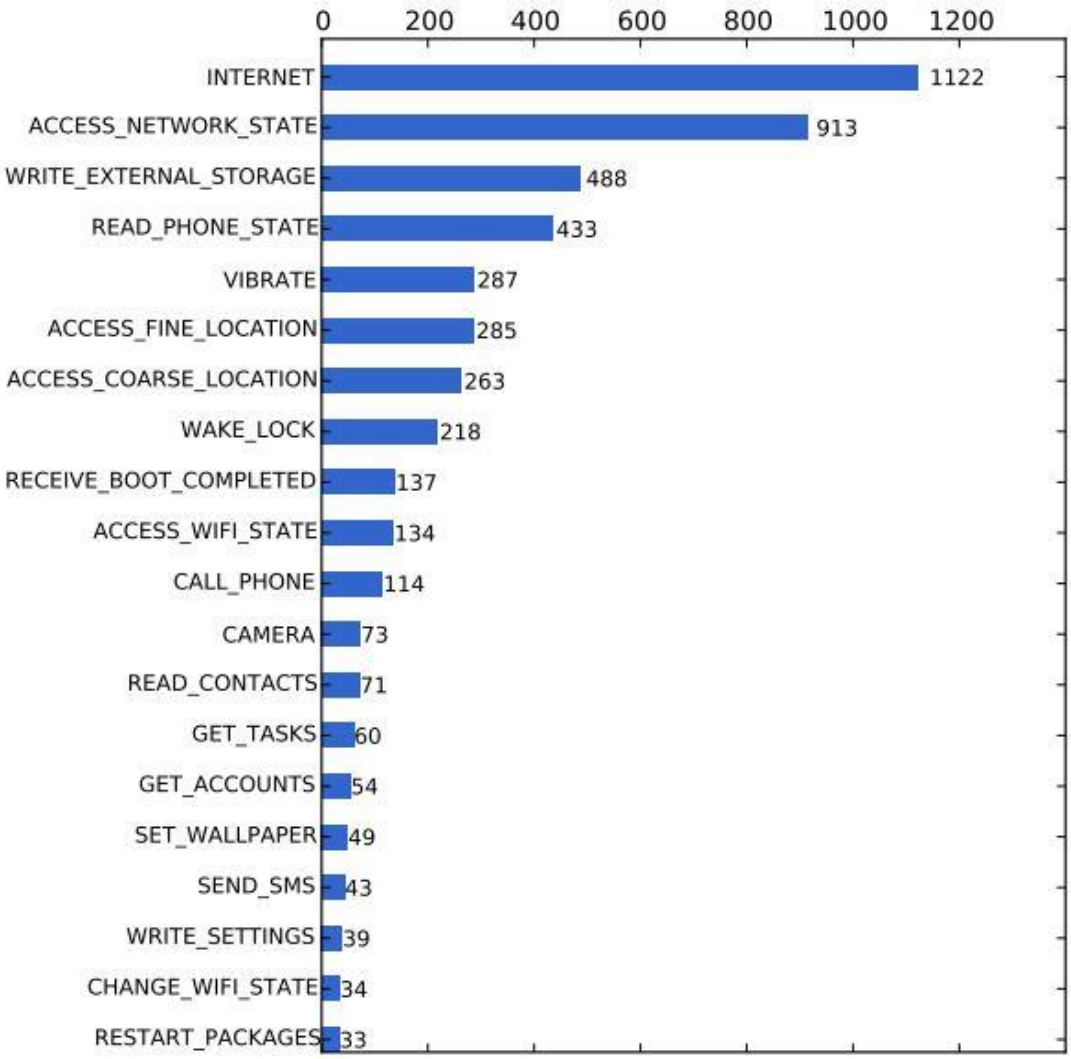

Figure 3.3: Top 20 Permissions Requested by 1260 Top Free (Benign) Apps on the Official Android Market [26]

With the Comparison of top most frequently Permissions used in malware and benign applications, where y-axis lists the name of the permissions and the x-axis lists the number of Apps. The results show that, on average, malware Apps intend to request more permissions than benign Apps.

On the other hand to demonstrate that API calls are indeed helpful for differentiating benign and malware applications, we report the top 10 API calls used in malicious and benign applications in Figure 3.5 and 3.4, the results clearly show that benign applications use more API calls than malware applications.

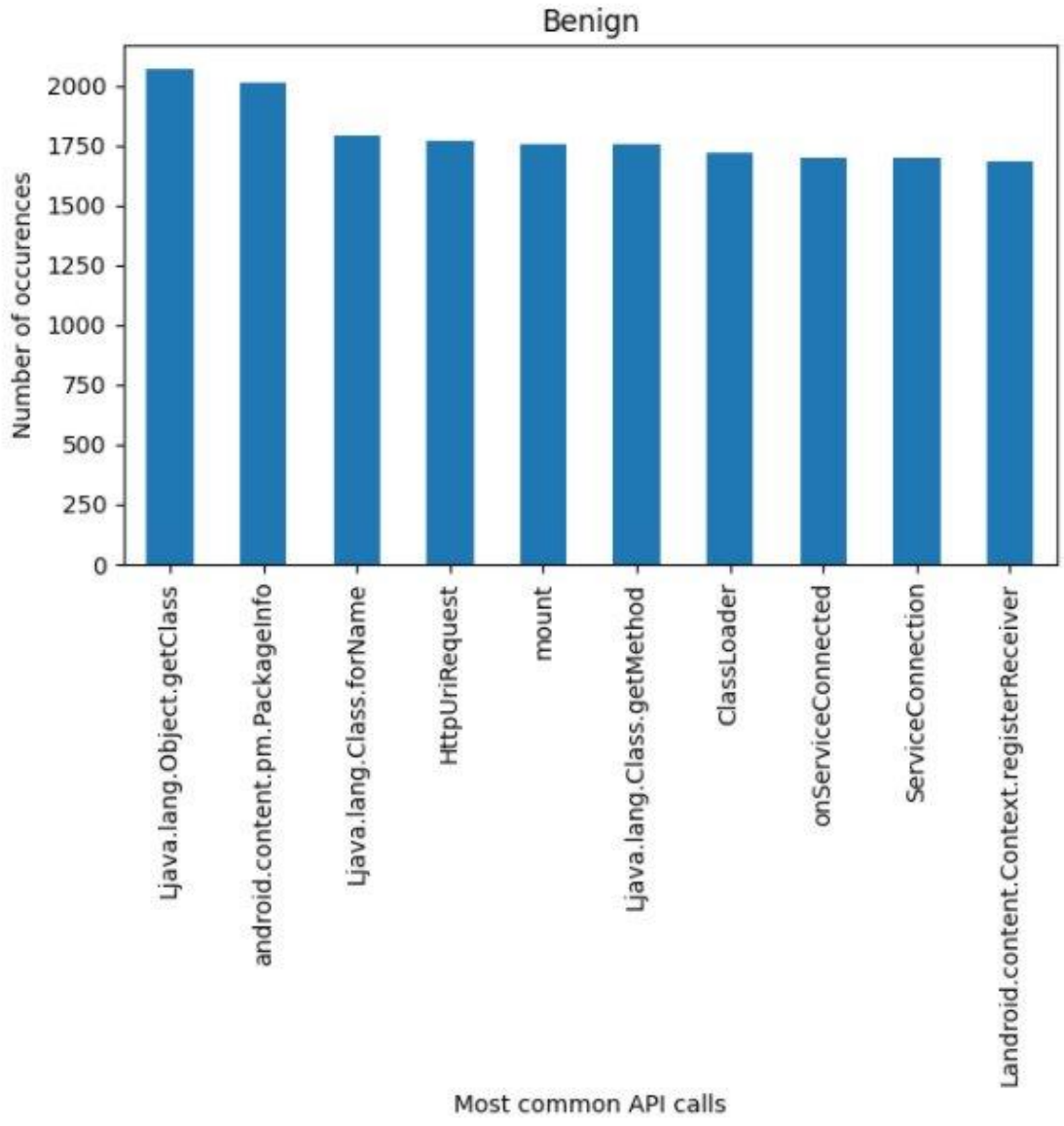

Figure 3.4: Most common API calls in benign apps [27]

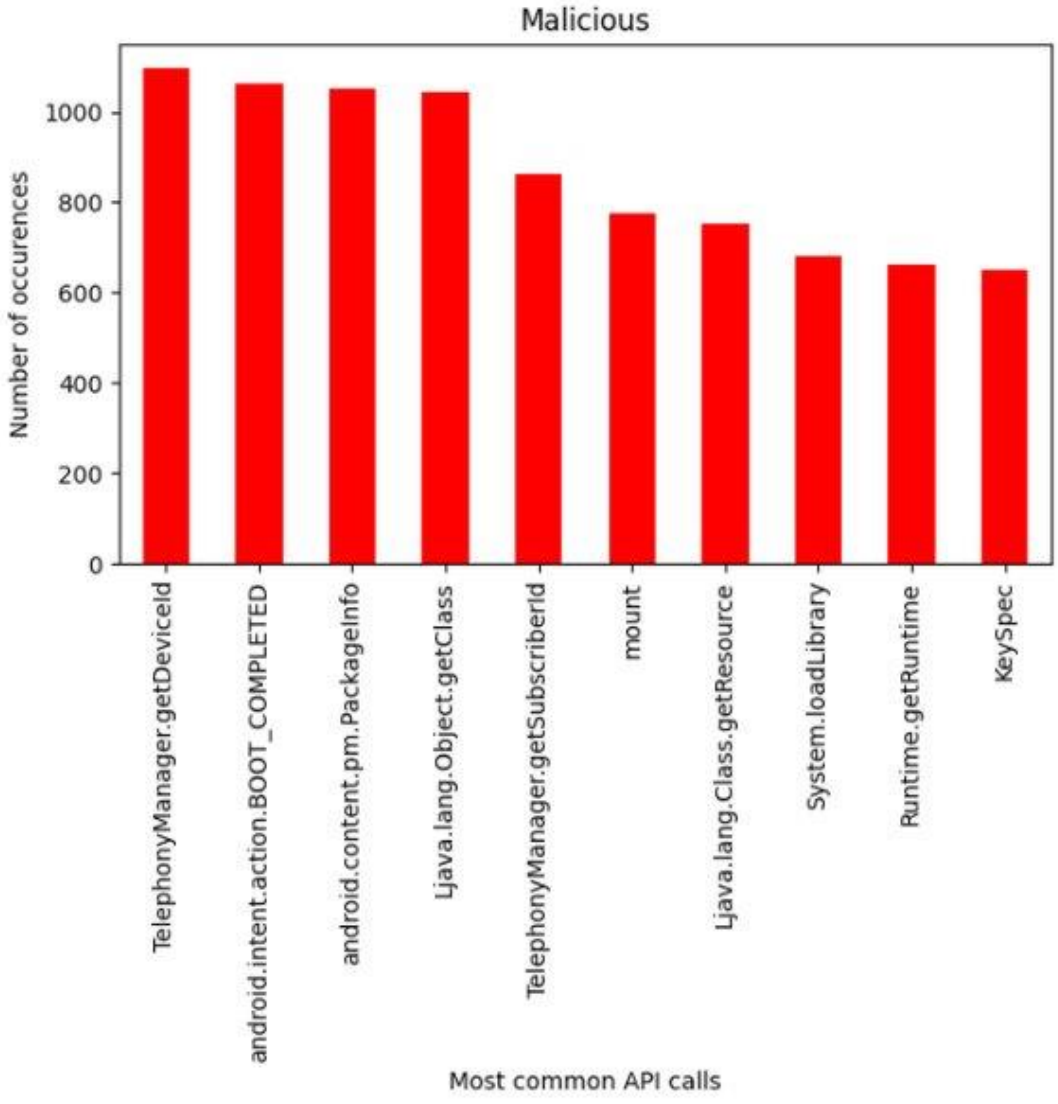

Figure 3.5: Most common API calls in malicious apps [27]

With the Comparison of top most frequently used API calls in malware and benign applications. x-axis lists the name of the API calls and the y-axis lists the number of Apps. The results show clear patterns/differences of API call usages: benign Apps tend to use more API classes than malware Apps.

This section aims to determine the most significant permissions that can be used for distinguishing Apps from malicious and benign ones. To select the significant key for permissions and API calls for malware detection, we have explored and evaluated the dangerous permission list from previous study [24].

Even though the combined feature set provide more accuracy classifications, it also introduce great overhead in computation and possibly some redundancies in features. To enhance the efficiency the classification process, we apply a feature selection method SVM to filtering features. From the last comparison between figure 3.2 withe figure 3.3 and figure 3.4 with figure 3.5 show the top ten features with highest F-statistic values, which are considered as more discriminative features. We pick top 300 features from API set and top 64 features from permission set, which achieve higher efficiency without signification impact on classification accuracy.

To select the minimum set of permissions and API calls we need to filter out the permissions and API calls that are less impactful for the detection. For this, we employed different combinations of dangerous permission and API calls list [24] (from Table 3.1), while also incorporating the feature importance property. Feature importance is a measure that helps generate simpler and faster prediction models using fewer inputs [28] [29]. To enlist the important permissions (Table3.1). We have defined a threshold measure of 0.1 to select the most impactful ones by ignoring the permissions and API calls that show importance lower than 0.01 and those who have 0.00 degree never mention it. Through the various combinations of feature set experiments, we finally identify the most significant permission and API calls list as shown in Table 3.2.

Type	**Number**	**Name**
Permissions	1	READ.PHONE.STATE
	2	READ.EXTERNAL.STORAGE
	3	WRITE.EXTERNAL.STORAGE
	4	ACCESS_COARSE_LOCATION
	5	ACCESS_FINE_LOCATION
	6	READ_CONTACTS
	7	READ_SMS
	8	RECEIVE_SMS
	9	ACCESS_WIFI_STATE
	10	ACCESS_NETWORK_STATE
	11	KILL_BAKGROUND_PROCESSES
	12	ACCESS_LOCATION_EXTRA_COMMANDS
	13	RECEIVE_BOOT_COMPLETED
	14	SYSTEM_ATERT_WINDOW

	15	RECEIVE_WAP_PUSH ……..;
API Calls	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	android.net.Proxy android.telephony android.app.PendingIntent android.content.Intene.ShortcutIconResource android.Annotation.TargetApi android.app.Service android.content.pm.ApplicationInfo android.content.pm.PackageInfo android.Provider.Browser android.content.IntentFilter android.content.pmServiceInfo android.content.pm.Signature android.os.IBinder android.content.ContentUris android.net.wifi …… ;

Table 3.2: The proposed identified parameter of the main features

3.4. Feature vector generation:

To represent the features extracted from raw data as a computable format, we have a dictionary for direct matching on permission and API call features, it is difficult to represent them through direct matching. The dictionary for features in manifest are generated from the Android source code which are predefined by Google developers. Totally, there are 170 permissions listed in dictionary. By studying the API calls from more than 500 real world Android applications, we find that there are a lot of API calls are not related to any sensitive components in Android OS. For example, the View loading API is widely used in most applications, but it cannot participate in any kinds of attacks. Meanwhile, among our dataset, most API calls from the third-party packages appeared only few times. In other words, a third-party API call in an application may never appear in others. Thus, we remove all uncommon third-party packages and the API calls which are not related to any sensitive components manually by experience and get the API call dictionary, which contains 2525 APIs According to the combined dictionary of manifest permissions and API call features, the size of vector is

determined by the dictionary size of features. For each Android app, the vector is represented by mapping the retrieved values to the dictionary size dimension vector space.

After that we present a Support vector machine (SVM) to train the classifier for malicious and benign Android apps, with our generated input feature vector, as showing figure 3.6.

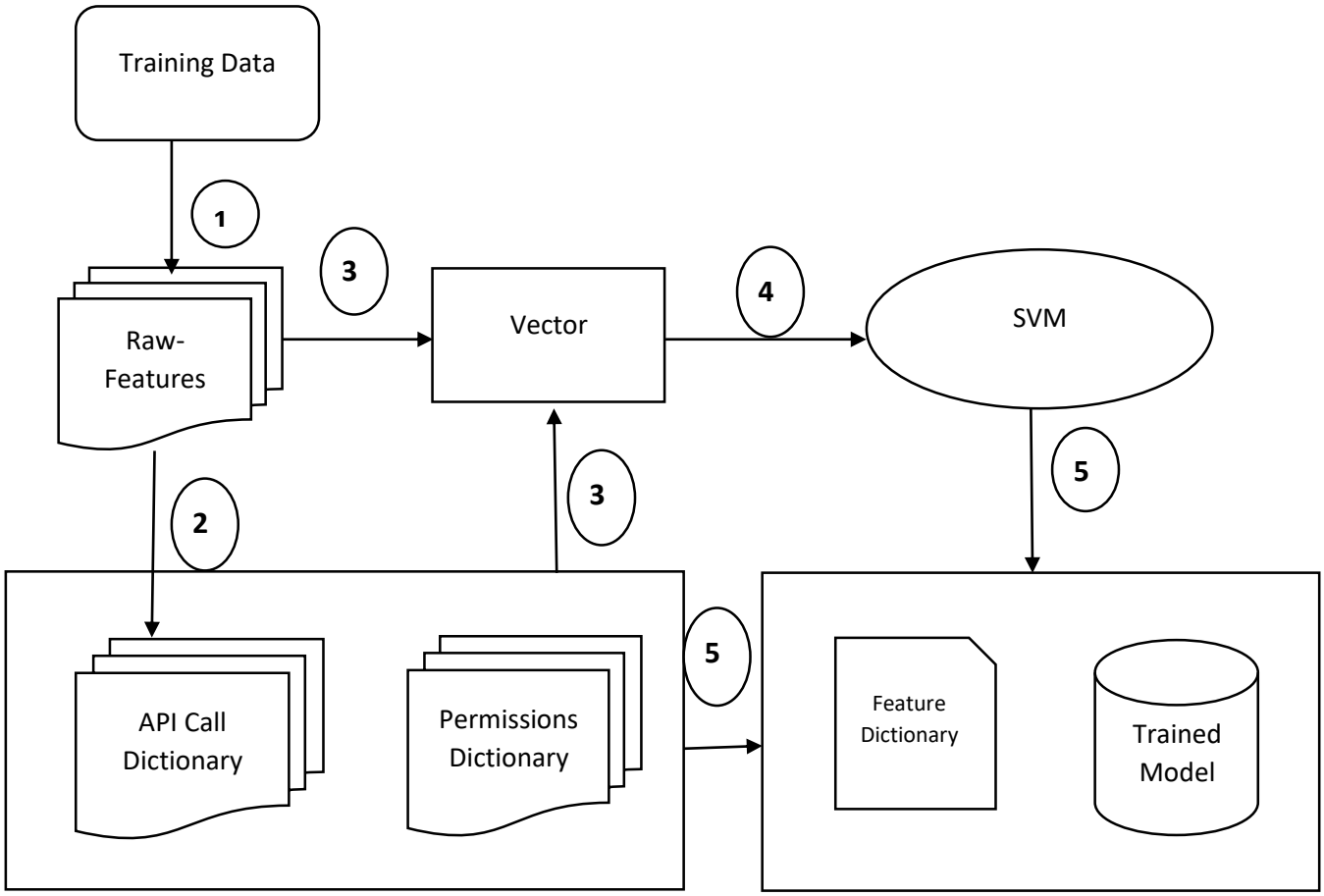

Figure 3.6: the processes of feature preparation and machine learning model training

Actually, machine learning-based approaches have achieved better performance compared with other approaches in Android malware detection. We intend to deploy the trained machine learning (ML) model on server-side to Android devices. While a computational intensive machine learning software could be executed efficiently on server-side with the GPU support, such machine learning or deep learning models usually cannot be

directly deployed and executed on other platforms supported by small Android devices due to various computation resource limitations, such as the computation power, memory size, and energy. Therefore, we use TENSORFLOW LITE [30] for Android to migrate the machine learning trained models. Due to the performance limitations of Android devices.

3.5. ML Model Quantization:

To deploy our pre-trained ML model on Android platform, we convert the model from the server-side platform to TensorFlowlite model, which is supported by Android operating system. To achieve this target, we convert pre-trained model to a TensorFlow model first. Following the Google TensorFlow guidance, and then migrate the TensorFlow model to a mobile readable TensorFlow-lite model. As we see in Figure 3.7.

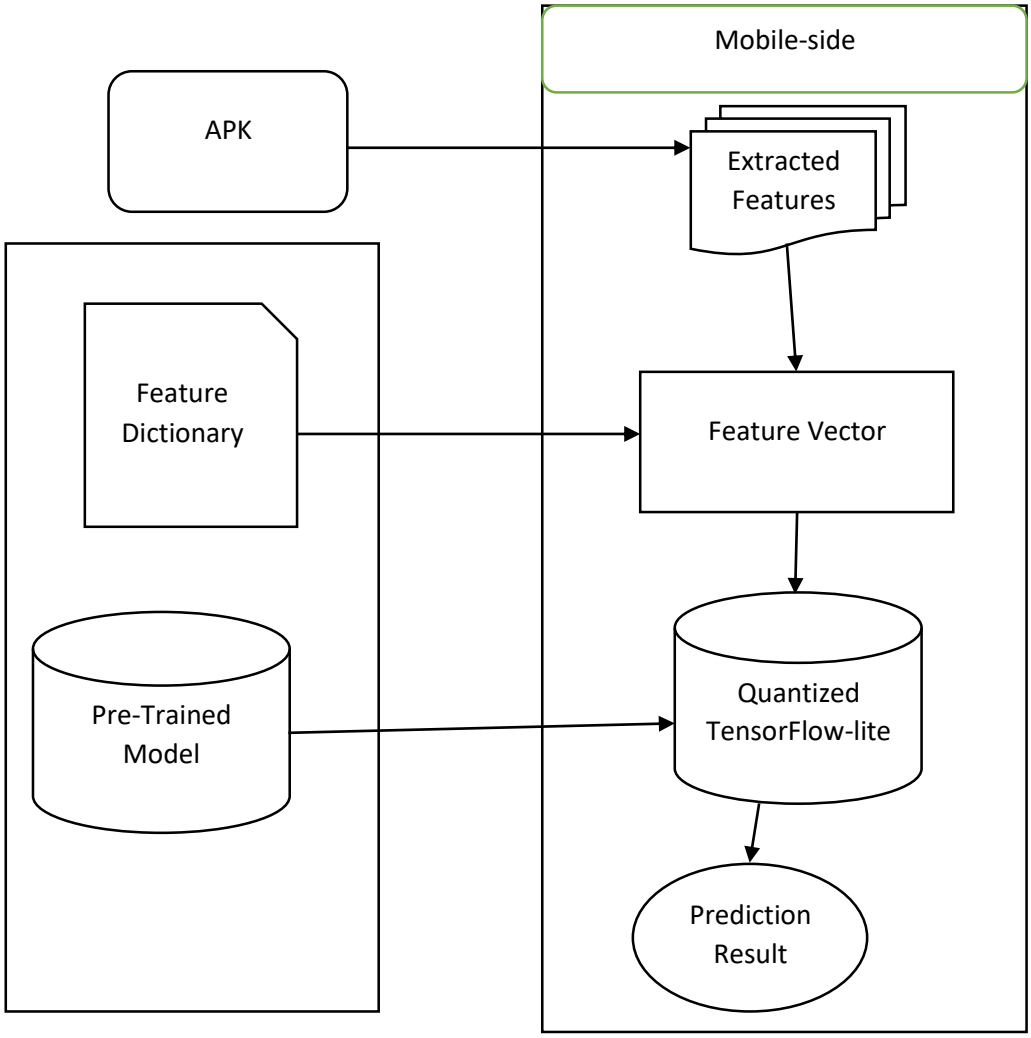

Figure 3.7: Deploying the pre-trained ML model on Android platform and workflow of mobile-side

3.6. Data set:

To collect benchmark data, we have gathered 285 APK files where 175 are malicious Apps and the remaining 110 are benign Android APK files. Which have been validated in a previous study [31] from Maldroid data set [32], The Maldroid data set comprises 11599 programs categorized as benign, adware, banking malware, and mobile riskware.

3.7. Classification Method:

The classification phase consist on building a model of machine learning algorithm that get the input data which is the selected features and learn from a training set how to predict new input, in this case, classifiers are built using selected features extracted from android applications whether from its source code or by observing its behavior and learn from a training set to detect malwares of a new input data, several machine learning algorithms are used for this purpose.

In this phase a subset of selected features is given to a machine learning classifier that is responsible to classify applications basing on the features used by those applications.

3.7.1. Support vector machine (SVM):

We use a SVM as a classifier relying on the accuracy and consistency of previous works.

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text.

Compared to newer algorithms like neural networks, they have two main advantages: higher speed and better performance with a limited number of samples (in the thousands). This makes the algorithm very suitable for text classification problems, where it's common to have access to a dataset of at most a couple of thousands of tagged samples.

SVM algorithms divides the n-dimensional space representation of the data into two regions by using a hyperplane, which intends to maximize the margin between the two regions separating two classes of samples. The margin is defined by the distance between the

examples of the two classes and is computed based on the distance between the closest instances of both classes, which are called support vectors [33].

Conclusion:

In this chapter, we proposed to use permissions and API calls of Android applications to detect malware and malicious codes in Android based mobile platform. With giving the features definition and selection, the proposed architecture extracts permissions from Android applications and further combines the API calls to characterize each application as a high dimension feature vector. By applying learning method to the collected datasets, we can derive classification models to classify Apps as benign or malware.

In the future, our number of samples will be extended to thousands, instead of limited samples as of now. This may help us to obtain more precise results in the evaluation experiments. On the other hand, the Sensitive Permissions and API calls should be monitored and modified regularly to cope with the recent Android malware trend. Then the SVM may not be the optimal classifier. Nevertheless, this static method should combine with another method, which is based on dynamic mechanism, so as to diversify and optimize performance.

CHAPTER 04:

IMPLEMENTATION AND EXPERIMENTATION

Introduction:

To be able to carry out an IT project, it is necessary to choose technologies that simplify its implementation. For this, after completing the conceptual study in the previous chapter, we will address the implementation part in what follows.

In this chapter, we start by presenting the work environment and then we present some graphical interfaces of the application. Finally, we end the chapter by presenting some tests of the application.

4.1 Development environments:

For the realization and implementation of our project, we need specific hardware and software that we will describe in the following.

4.1.1 Java:

Java is an object-oriented language, developed by SUN (1991). The goal of Java at the time was to constitute a programming language that could be integrated into household appliances in order to be able to control them, to make them interactive, and above all to allow communication between the devices [34].

Java technology is the foundation of most networked applications and is leveraged around the world to develop and deliver mobile and embedded applications, games, web content and enterprise software. Used by millions of developers around the world, Java technology makes it possible to efficiently develop, deploy and use fascinating applications and services [35].

Software developers choose Java technology because it has been tested, tuned, extended and proven by a dedicated community of Java developers, architects and enthusiasts. It was designed to enable the development of high-performance portable applications on a wide range of computing platforms. By delivering applications in heterogeneous environments, enterprises can deliver more services and boost end-user productivity, communication, and collaboration, while dramatically reducing the cost of ownership of enterprise applications. And general public. Java has now become an indispensable tool that allows developers to:

- write software on one platform and run it on virtually any other platform،
- create programs that can be run in a web browser and access available web services،
- develop server-side applications for forums، online stores and surveys، for processing HTML forms، etc.
- to combine applications or services based on the Java language to create very personalized applications or services،
- Write powerful and efficient applications for cell phones, remote processors, microcontrollers, wireless modules, sensors, gateways, consumer products and all other types of electronic device.

4.1.2 XML*:*

Extensible Markup Language (XML) is a markup language and file format for storing, transmitting, and reconstructing arbitrary data. It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The World Wide Web Consortium's XML 1.0 Specification [36] of 1998 [37] and several other related specifications —all of them free open standards—define XML [38]**.**

The design goals of XML emphasize simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures [39] such as those used in Android apps.

Several schema systems exist to aid in the definition of XML-based languages, while programmers have developed many application programming interfaces (APIs) to aid the processing of XML data.

4.1.3 Android Studio:

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development [40].It is available for download on Windows, macOS and Linux based operating systems or as a subscription-based service in 2020 [41].It is a

replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development.

Android Studio was announced on May 16, 2013, at the Google I/O conference. It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014 [42]. The first stable build was released in December 2014, starting from version 1.0 .

Android Studio supports all the same programming languages of IntelliJ (and CLion) e.g. Java, C++, and more with extensions, such as Go [43]. And Android Studio 3.0 or later supports Kotlin [44]. And "all Java 7 language features and a subset of Java 8 language features that vary by platform version." External projects backport some Java 9 features. While IntelliJ states that Android Studio supports all released Java versions, and Java 12, it's not clear to what level Android Studio supports Java versions up to Java 12 (the documentation mentions partial Java 8 support). At least some new language features up to Java 12 are usable in Android.

Once an app has been compiled with Android Studio, it can be published on the Google Play Store. The application has to be in line with the Google Play Store developer content policy.

4.1.3.1 Android Studio Settings:

- ✓ Android Gradle Plugin Version (4.1.2).
- ✓ Gradle Version(6.5).
- ✓ AVD Xiaomi M2004J19C.
- ✓ API (v29).
- ✓ Compile Sdk Version 30.
- ✓ Build Tools Version "30.0.3."
- ✓ Min Sdk Version 19.

4.1.4. TensorFlow:

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets

researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications [45].

- Easy model building:

Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

- Robust ML production anywhere:

Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

- Powerful experimentation for research:

A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.

4.1.5. Basic materials:

For the realization and the implementation of our system, we need the following materials:

- Laptop: to run Android studio and deferent tools.
- A smartphone: to test the mobile application of our system.

We run all our tests on a custom built computer:

- CPU: Intel(R) Celeron (R) CPU 1037U @ 1.80GHz (2 CPUs).
- RAM: 4 GB.
- Operating System: Microsoft Windows 10 Pro 64 bit.

4.1.6. Dataset:

A dataset is a collection of data. In the case of tabular data, a data set corresponds to one or more database tables, where every column of a table represents a particular variable, and each row corresponds to a given record of the data set in question. The data set lists values for each of the variables, such as for example height and weight of an object, for each member of the data set. Data sets can also consist of a collection of documents or files or APKs [46].

Collecting data from applications is an important and sensitive step, because it is necessary to choose applications that have a direct relationship with the goal of the research, and it is preferable to choose modern applications that keep pace with time to get the best result.

4.1.7. Apktool:

A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications. It also makes working with an app easier because of the project like file structure and automation of some repetitive tasks like building apk, etc. [25]

It is not intended for piracy and other non-legal uses. It could be used for localizing, adding some features or support for custom platforms, analyzing applications and much more.

4.1.7.1. Features:

- Disassembling resources to nearly original form (including **resources.arsc**, **classes.dex**, **9.png.** and **XMLs**).
- Rebuilding decoded resources back to binary APK/JAR
- Organizing and handling APKs that depend on framework resources
- Smali Debugging (Removed in **2.1.0** in favor of IdeaSmali)
- Helping with repetitive tasks

4.1.7.2. Requirements:

- Java 8 (JRE 1.8).
- Basic knowledge of Android SDK, AAPT and smali.

For extracting features list we need to get access into decompiled files and get the API calls and permissions who matching the features dictionary as showed in figure 4.3 and 4.4.

4.2. Project general architecture:

Our MobMal Detector project is composed of three modules:

Browsing module: this module aims to browse and the select the APK file from android URIs.

Extractor module: this module extracts features and generates features vector, the last one is a test input for SVM classifier.

Decoder Module: it uses Apktool code for decoding resources.

Training module: it contains features dictionary and the trained model.

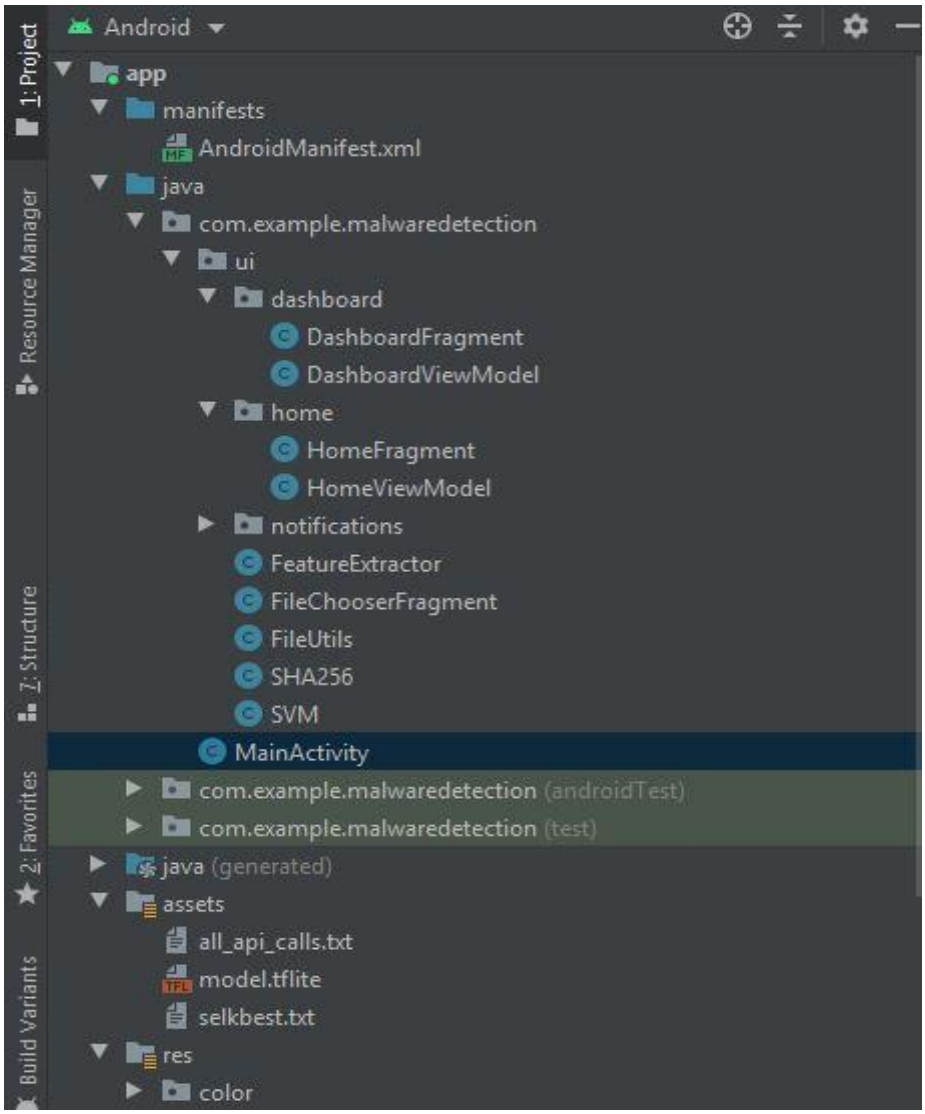

Figure 4.1: The main architecture of the project on android studio

From the last figure we have some sub-classes and main classes which including FileUtils.java and FileChooserFragment.java those two classes control the browsing and the selection of APK file from android URIs.

FeatureExtractor.java this class extract features and generate features vector, the last one is an input for SVM.java classifier class.

DashboardFragment.java contain Apktool code for decoding resources and this class control the GUIs of the figure 4.9 and call the classification method with getting and matching the results of the prediction.

assets resource, contain features dictionary's and the trained model.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Figure 4.2: The permissions requested by MobMal detector

```
public String readPermission(){
    try{
        Matcher m1;
        Matcher m2;
        Matcher m3;

        String line , per;
        String allPers="";
        Pattern p1 = Pattern.compile("uses-permission");
        Pattern p2 = Pattern.compile("action");
        Pattern p3 = Pattern.compile("category");

        if(main!=null)
            main.updateProgress( num: 95, text: "Extracting Permission");
        BufferedReader bf = new BufferedReader(new FileReader( fileName: path+"/AndroidManifest.xml"));
        line = bf.readLine();
        while (line!=null) {
            m1 = p1.matcher(line);
            m2 = p2.matcher(line);
            m3 = p3.matcher(line);

            if(m1.find() || m2.find() || m3.find()){
                per = line.substring(line.indexOf("android:name")+14,line.indexOf( ch: '"', fromIndex: line.
                if(all_api_calls_selkbest.contains(per)) {
                    allPers = allPers + api_index.get(per) +",";
                }
            }
```

Figure 4.3: Read and extract permissions from AndroidManifest.xml

```
        /*Extracting feature*/
        AtomicReference<String> feature = new AtomicReference<>( initialValue: "");
        AtomicInteger count = new AtomicInteger( initialValue: 1);
        int size = listFile.size();
        int k =size/8;
        for(int i=0 ; i <8 ;i ++){
            int finalI = i;
            Thread background = new Thread(() -> {
                String content;
                Scanner scanner;
                String line=null;
                Matcher m1;
                Matcher m2;
                Matcher m3;
                Matcher m4;
                Matcher m5;

                Pattern p1 = Pattern.compile(";->");
                Pattern p2 = Pattern.compile("/*Manager");
                Pattern p3 = Pattern.compile("\\(.*\\)");
                Pattern p4 = Pattern.compile("Layout");
                Pattern p5 = Pattern.compile("Landroid");

                String api;
```

Figure 4.4: Read and extract API calls from files classes.

```
public void readSBAPI(List<String> all_api_calls_selkbest){
    BufferedReader reader;
    try{
        final InputStream file = getActivity().getAssets().open( fileName: "selkbest.txt");
        reader = new BufferedReader(new InputStreamReader(file));
        String line = reader.readLine();
        while(line != null){
            all_api_calls_selkbest.add(line);
            line = reader.readLine();
        }
        file.close();
        reader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void readAllAPI(Hashtable<String, String> api_index){
    BufferedReader reader;
    int index = 1;
    try{
        final InputStream file = getActivity().getAssets().open( fileName: "all_api_calls.txt");
        reader = new BufferedReader(new InputStreamReader(file));
        String line = reader.readLine();
        while(line != null){
            api_index.put(line,String.valueOf(index));
```

Figure 4.5: Read and matching features dictionary to generate features vector.

4.3. Application GUIs:

To start working with MobMal Detector you can move from the main interface of our application "MobMal" to a list containing all the installed applications to perform the examination.

Figure 4.6: logo of Our App "MobMal"

4.3.1. The main interface:

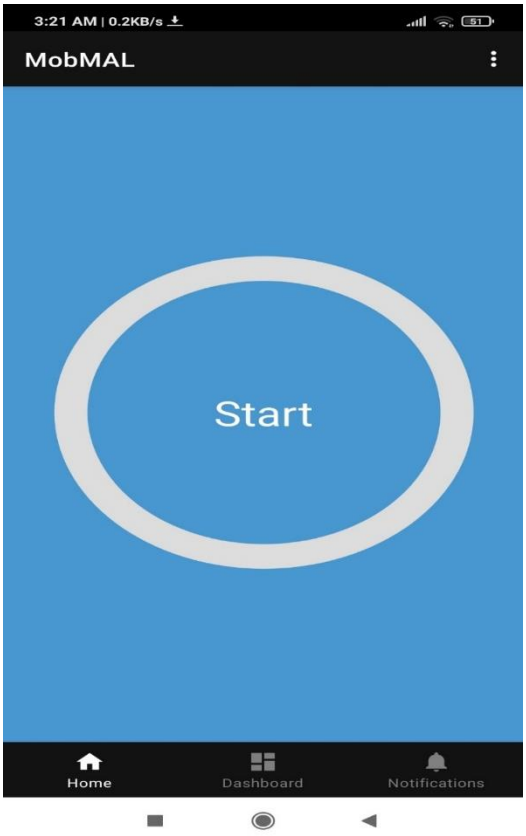

Figure 4.7: main interface of our App

After pressing the button "Start", the application displays Figure 4.8.

4.3.2. The second interface:

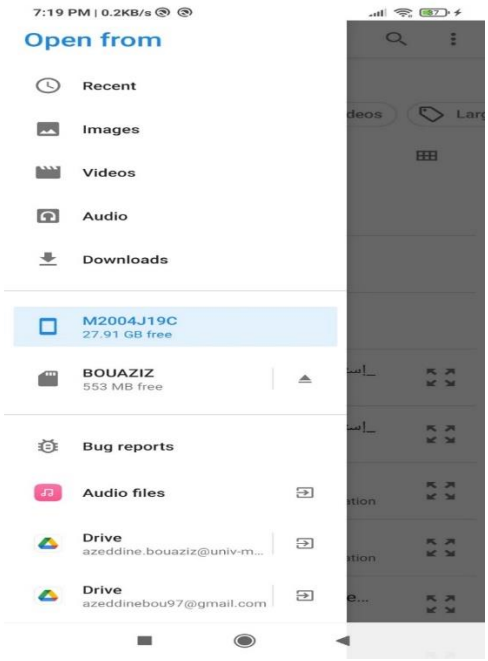

Figure 4.8: Select APK file path in android URI

In this case I choose facebook lite ".apk".

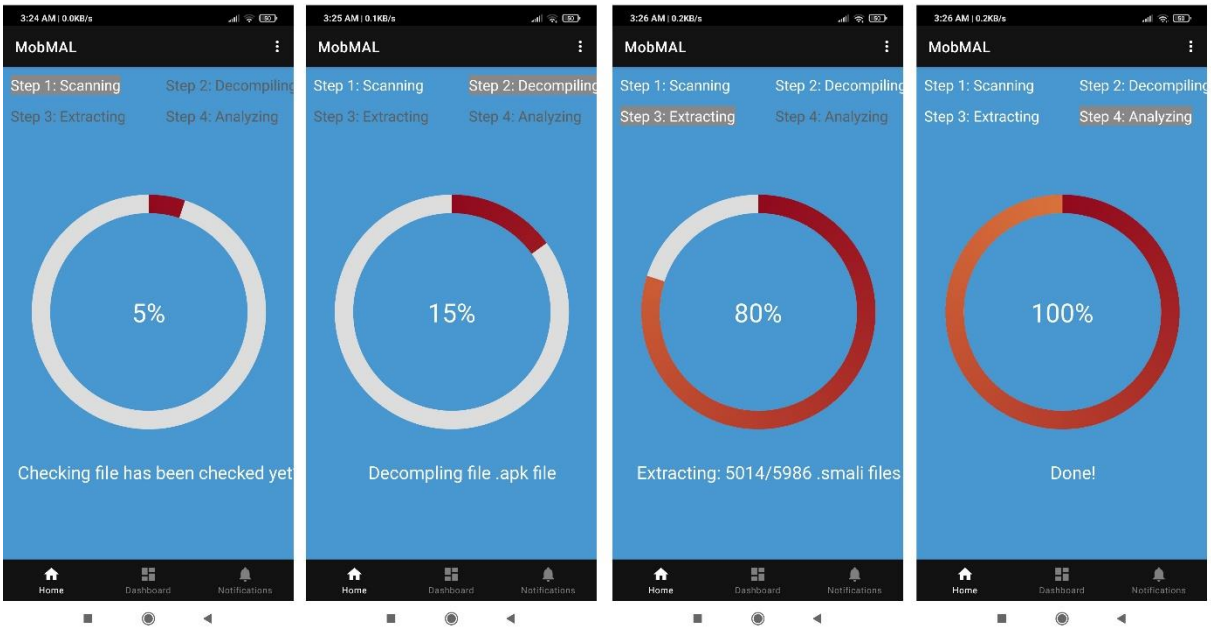

Figure 4.9: The 4 major processes to making prediction

4.3.3. Result interface:

The result of scanning facebook lite as an example with another malicious app named benews.apk is showed in Figure 4.10.

Application is secure (benign) or insecure (malicious):

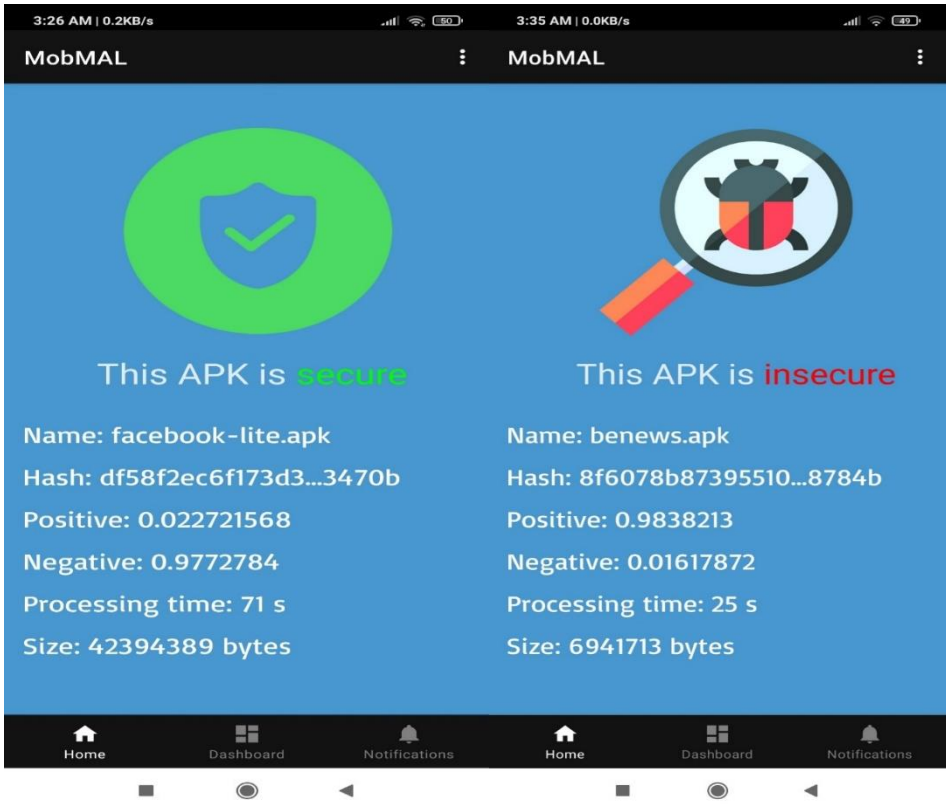

Figure 4.10: Result of scanning.

4.4. General results:

As a final test we test more than 20 malwares sample from different categories downloaded from github [47] of malwares and more than 20 benign apps (randomly downloaded from each category on the Google Play official Android market) so As a result, we arrived at an estimated accuracy 83%. This is a good thing for an initial version, and we also aspire in the future to integrate more than one method and more than two features categories for detection and use a wide and reliable dataset to obtain the best accuracy and analysis.

Conclusion:

In this chapter we have shown our working environment which contains the hardware and software part used to implement our Detector, and moreover the architecture and the experimentation of our mobile application.

General conclusion:

Since the arrival of smartphones, thousands of mobile applications, free or paid, that can be downloaded from our telephone or our touch pads, have appeared. Many of them are of interest to the field of cyber security and protection against viruses and malicious applications.

The objective of our work is to develop a Mobile Malware Detector "MobMal" under Android plateform, to detect installed malware application on the tested smartphone using machine learning algorithms with API calls and Permissions dataset features, it is implemented using java language under the Android studio environment.

for an initial version result, we arrived at an estimated accuracy 83%, and we aspire in the future to integrate more than one method and more than two features categories for detection and use a wide and reliable dataset to obtain the best accuracy and analysis.

This project was an interesting experience, which allowed us to learn and improve our knowledge and skills in the field of mobile programming security and machine learning.

Perspectives:

The application we have developed could be enriched by several advanced features. Among the perspectives adopted to improve the functioning of the system, we cite in particular:

- ✓ Enrich the application with more efficient classification methods.
- ✓ Enrich the learning base.
- ✓ Add templates for each malicious application to make classification more reliable.
- ✓ Make the application compatible with several mobile platforms (IOS, Windows Phone).

Finally, we want our efforts to bring this project to life because there are currently malicious application detection applications in the market.

REFERENCE:

[1] Statistics company: Statista https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[2] Seek logo: https://seeklogo.com/free-vector-logos/android

[3] Stefan Brahler :Analysis of the Android Architecture (figure 1.1)

[4] Umer Farooq Computer Science Department, Virtual University of Pakistan, Lahore -- Pakistan: Android Operating System Architecture

[5] Andrei Frumusanu (July 1, 2014). "A Closer Look at Android RunTime (ART) in Android L". AnandTech. Retrieved July 5, 2014.

[6] Tutorials point: https://www.tutorialspoint.com/android/android_architecture.htm

[7] Chen J, Alalfi MH, Dean TR et al. Detecting Android malware using clone detection. JOURNAL OF COMPUTER, SCIENCE AND TECHNOLOGY 30(5): 942–956 Sept. 2015. DOI 10.1007/s11390-015-1573-7 (figure 2)

[8] A Malware Detection System For Android by M.Sc. Franklin Tchakounté (p 19)

[9] A Malware Detection System For Android by M.Sc. Franklin Tchakounté (p 33)

[10] Shrikant Korke, Arjun Mane, Mahendra Dhore: Comparative Study fo the Android Malware Analysis Techniques (figure 1)

[11] T. Chen, Q. Mao, Y. Yang, M. Lv, J. Zhu, TinyDroid: A lightweight and efficient model for Android malware detection and classification, Mob. Inf. Syst. 2018 (2018) 4157156:1–4157156:9, http://dx.doi.org/10.1155/2018/4157156.

[12] P. Liu, W. Wang, X. Luo, H. Wang, C. Liu, NSDroid: Efficient multiclassification of android malware using neighborhood signature in local function call graphs, Int. J. Inf. Secur. (2020) http://dx.doi.org/10.1007/s10207-020-00489-5

[13] W. Zhou, Y. Zhou, X. Jiang, P. Ning, Detecting repackaged smartphone applications in third-party android marketplaces, in: Second ACM Conference on Data and Application

Security and Privacy, CODASPY 2012, San Antonio, TX, USA, February 7-9, 2012, 2012, pp. 317–326, http://dx.doi.org/10.1145/2133601.2133640.

[14] G. Suarez-Tangil, S.K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, L. Cavallaro, Droidsieve: Fast and accurate classification of obfuscated Android malware, in: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017, 2017, pp. 309–320, http://dx.doi.org/10.1145/3029806.3029825.

[15] W. Enck, M. Ongtang, P. McDaniel, On lightweight mobile phone application certification, in: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, ACM, New York, NY, USA, 2009, pp. 235–245, http://dx.doi.org/10.1145/1653662.1653691.

[16] Asma Razgallah Raphaël Khoury Sylvain Hallé Kobra Khanmohammadi: A survey of malware detection in Android apps: Recommendations and perspectives for future research (Table 05)

[17] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, A.K. Sangaiah, Android malware detection based on system call sequences and LSTM, Multimedia Tools Appl. 78 (4) (2019) 3979–3999 https://link.springer.com/article/10.1007/s11042-017-5104-0.

[18] I. Burguera, U. Zurutuza, S. Nadjm-Tehrani, Crowdroid: Behavior-based malware detection system for Android, in: SPSM'11, Proceedings of the 1st ACM Workshop Security and Privacy in Smartphones and Mobile Devices, Co-Located with CCS 2011, October 17, 2011, Chicago, IL, USA, 2011, pp. 15–26, http://dx.doi.org/10.1145/2046614.2046619.

[19] N. Xie, F. Zeng, X. Qin, Y. Zhang, M. Zhou, C. Lv, RepassDroid: Automatic detection of Android malware based on essential permissions and semantic features of sensitive APIs, in 2018 International Symposium on Theoretical Aspects of Software Engineering, TASE, IEEE, 2018, pp. 52–59 https://ieeexplore.ieee.org/document/8560733.

[20] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, XManDRoid: A New Android Evolution to Mitigate Privilege Escalation Attacks, Tech. Rep. TR-2011-04, Technische Universität Darmstadt, 2011.

[21] Asma Razgallah Raphaël Khoury Sylvain Hallé Kobra Khanmohammadi: A survey of malware detection in Android apps: Recommendations and perspectives for future research (Table 05).

[22] Choi, Y.H. et al.: Toward extracting malware features for classification using static and dynamic analysis. Computing and Networking Technology (ICCNT), South Korea,

[23] Eskandari, M., Khorshidpour, Z., Hashemi, S.: HDM-Analyser: A hybrid analysis approach based on data mining techniques for malware detection. J. Comput. Virol. Hack. Techn.

[24] Feature Selection on Permissions, Intents and APIs for Android Malware Detection by Fred Guyton/04-21-2021/NSU Florida.

[25] Android-Apktool, A tool for reverse engineering Android apk files https://code.google.com/p/android-apktool/ .

[26] Yajin Zhou , Xuxian Jiang, Dissecting Android Malware: Characterization and Evolution.

[27] Detecting Malicious Android Applications Based on API calls and Permissions Using Machine learning Algorithms in May 2021/DOI: 10.1109/MIUCC52538.2021.9447594.

[28] Sotiroudis, S.P.; Goudos, S.K.; Siakavara, K. Feature Importances: A Tool to Explain Radio Propagation and Reduce Model Complexity. Telecom 2020, 1, 114–125.

[29] Nasir, M.; Javed, A.R.; Tariq, M.A.; Asim, M.; Baker, T. Feature engineering and deep learning-based intrusion detection Framework for securing edge IoT. J. Supercomput. 2022, 78, 1–15.

[30] For Mobile & Edge, https://www.tensorflow.org/lite/

[31] Eslam Amer, Seif ElDein Mohamed,Using Machine Learning to Identify Android Malware Relying on API calling sequences and Permissions /Journal of Computing and Communication Vol.1 , No.1 , PP. 38-47 , 2022

[32] CICMalDroid 2020. https://www.unb.ca/cic/datasets/maldroid-2020.html/, 2020.

[33] V. Vapnik, The nature of statistical learning theory. Springer, 2000.

[34] «Java,» CommentCaMarche,. [En ligne]. http://www.commentcamarche.net/contents/557java .

[35] «Learn about Java technology,» Oracle, [En ligne]. https://java.com/en/about/.

[36] "XML 1.0 Specification". World Wide Web Consortium. Retrieved 22 August 2010.

[37] "Extensible Markup Language (XML) 1.0". www.w3.org.

[38] "W3C DOCUMENT LICENSE". W3.org. Retrieved 24 July 2020.

[39] Fennell, Philip (June 2013). "Extremes of XML". XML London 2013: 80–86. doi:10.14337/XMLLondon13.Fennell01. ISBN 978-0-9926471-0-0.

[40] Ducrohet, Xavier; Norbye, Tor; Chou, Katherine (May 15, 2013). "Android Studio: An IDE built for Android". Android Developers Blog. Retrieved May 16, 2013.

[41] "Getting Started with Android Studio". Android Developers. Retrieved May 14, 2013.

[42] "Download Android Studio". Android Developers. Retrieved June 13, 2015.

[43] Google Go language IDE built using the IntelliJ Platform: go-lang-plugin-org/go-lang-idea-plugin, Go Language support for IDEA based IDEs, February 23, 2019, retrieved February 23, 2019, Supported IDEs [...] Android Studio 1.2.1+ .

[44] Get Started with Kotlin on Android | Android Developers". developer.android.com. Retrieved October 25, 2017.

[45] tensor flow official website: https://www.tensorflow.org/ .

[46] Snijders, C.; Matzat, U.; Reips, U.-D. (2012). "'Big Data': Big gaps of knowledge in the field of Internet".

[47] Android Malware Samples https://github.com/ashishb/android-malware.