MASTER THESIS

Android Malware Detection Using Machine Learning with User Feedback and Static Features

Author:
Mohammad Modallal

Supervisor:
Dr. Ahmad ALSADEH

A thesis submitted in partial fulfillment for the degree of Master of Computing

June 22, 2020

نموذج إتمام تعديلات رسالة الماجستير

اسم الطالب: ـ محمد مفيد محمود مدلل ـ الرقم الجامعي: ـ 1165407
التخصص: ـــ ماجستير الحوسبة

تاريخ المناقشة: ـــ 2020-05-21
عنوان الرسالة باللغة العربية:
الكشف عن البرامج الضارة في الأندرويد باستخدام التعلم الآلي مع ملاحظات المستخدم والميزات الثابتة

عنوان الرسالة باللغة الإنجليزية: ـــ
Android Malware Detection Using Machine Learning with User Feedback and Static Features

أتم الطالب التعديلات التي طلبت منه في جلسة مناقشة الرسالة
[✓ نعم] ☐ لا

رئيس اللجنة (المشرف): ـ أحمد السعده ـ التوقيع: [illegible] التاريخ: 23-06-2020

عضو اللجنة : أبو السعود حنني التوقيع: [illegible] التاريخ: ٢٣/٦/٢٠٢٠م

عضو اللجنة : أحمد أبو سنينة التوقيع: [illegible] التاريخ: **2020/6/22**

نسخ / دائرة التسجيل والقبول، العميد، البرنامج، المشرف، أعضاء اللجنة، الطالب/ة

Declaration of Authorship

Abstract

According to International Data Corporation (IDC), the Android operating system has occupied most of the market share and it will increase to 87.4% by the end of 2023. Unfortunately, there is a large number of malicious applications that attack Android OS in different ways. Therefore, much research has been done in this area to detect these malware, but until now there is no complete solution that can detect all malware, especially new malware that increases from year to year. In this research, we proposed *MLSecAndroid*: an Android anti-malware approach that automatically detects malicious applications in Android applications' marketplaces using machine-learning techniques. MLSecAndroid uses 14984 benign applications and 2116 malicious applications from the Google Play and Aptoide stores with their users' feedbacks. The collected dataset need for cleaning to be ready for processing. Therefore, the pre-processing stage is applied to clean the data using many steps such as data cleaning, data integration, and data transformation. MLSecAndroid uses static features, such as permissions, system API calls, users' feedbacks, and other features. The feature extraction stage produces thousands of features. Therefore, we applied the feature selection stage to select the best n-features. We evaluate the use of the features with different machine learning classifiers such as support vector machines (SVM), Random Forest, Decision Tree(DT), K-Nearest Neighbors algorithm (KNN), AdaBoost, and Naive Bayes classifiers in order to classify unknown Android application as either malware or benign. The experiment result shows that the MLSecAndroid has a very good performance in comparison with related works. It achieves an accuracy of 98.21% and a recall of 93.52% when using the SVM classifier.

مستخلص

وفقًا لشركةِ البيانات الدولية (International Data Corporation)، يشكل نظام تشغيل أندرويد (Android) معظم حصة السوق ومن المتوقع أن يزيد إلى أكثر من 87.4% بحلول نهاية عام 2023. للأسف، هناك عدد كبير من التطبيقات الضارة التي تهاجم نظام التشغيل أندرويد بطرق مختلفة. لذلك، تم إجراء العديد من الأبحاث في هذا المجال للكشف عن هذه البرامج الضارة، ولكن حتى الآن لا يوجد حل كامل يمكنه الكشف عن جميع هذه البرامج الضارة؛ خاصة البرامج الضارة الجديدة التي تزداد من عامٍ إلى آخر.

في هذا البحث، اقترحنا منهج لمكافحة البرامج الضارة الذي يكتشف تلقائيًا تطبيقات أندرويد الضارة باستخدام تقنيات التعلم الآلي (machine-learning). يستخدم هذا الأسلوب 14984 تطبيقًا حميدًا و 2116 تطبيقًا ضاراً من متاجرِ Google Play و Aptoide مع آراء المستخدمين.

تحتاج مجموعة البيانات المجمعة إلى التنظيف لتكون جاهزة للمعالجة. لذلك، يتم تطبيق مرحلة المعالجة المسبقة لتنظيف البيانات باستخدام العديد من الخطوات مثل تنظيف البيانات وتكامل البيانات وتحويل البيانات. يستخدم المنهج المقترح ميزات ثابتة مثل أذونات استخدام مصادر الأجهزة المشغلة لتطبيقات الأندرويد وتعليقات و أراء المستخدمين وميزات أخرى.

تنتج مرحلة استخراج الميزات آلاف الميزات، لذلك قمنا بتطبيق مرحلة اختيار الميزة لتحديد أفضل عدد من الميزات (n-features) عن طريق تقييم استخدام الميزات مع التصنيفات المختلفة لتعلم الآلة مثل خوارزمية (support vector machines)، خوارزمية (Random Forest)، خوارزمية (Decision Tree)، خوارزمية (K-Nearest Neighbours)، خوارزمية (AdaBoost)، خوارزمية (Naive Bayes) من أجل تصنيف تطبيقات أندرويد غير المعرفة والتي إما أن تكون تطبيقات ضارة أو حميدة.

تظهر نتيجة التجربة أن نهجنا يتمتع بأداء جيد جداً مقارنة بالأعمال ذات الصلة. حيث يحقق دقة (Accuracy) بقيمة 98.21% واستدعاء (Recall) بقيمة 93.52% عند استخدام خوارزمية (SVM).

Contents

Declaration of Authorship **iii**

Abstract **v**

List of Figures **ix**

List of Tables **xi**

List of Abbreviations **xiii**

1 Introduction **1**
1.1 Research Problem and Motivation 2
1.2 Research Questions 2
1.3 Objectives 3
1.4 Contributions 3
1.5 Thesis Organization 4

2 Background **5**
2.1 Android System Architecture 5
2.1.1 System Applications 5
2.1.2 Java API Framework 5
2.1.3 Native C/C++ Libraries 7
2.1.4 Android Runtime 7
2.1.5 Hardware Abstraction Layer (HAL) 8
2.1.6 The Linux Kernel 8
2.1.7 Power Management 8
2.2 Android Permissions 8
2.2.1 Privacy or Read Related Permissions 9
2.2.2 Write or Modify Related Permissions 9
2.3 Android System API Calls 10
2.4 Android Users Feedback 10
2.5 Android System Malware 11
2.6 Malware Detection Approaches and Tools 13
2.6.1 Reverse Engineering 13
2.6.2 Androguard 13
2.6.3 Aptoide 14
2.6.4 VirusTotal 14
2.6.5 Sentiment Analysis 14
2.6.6 Selenium 14

3 Related Work 17
3.1 Non-Machine Learning Techniques 17
3.1.1 Static Analysis Techniques 17
3.1.2 Dynamic Analysis Techniques 18
3.1.3 Hybrid Analysis Techniques 18
3.2 Machine Learning Techniques 19
3.2.1 Classical Machine Learning Techniques 19
3.2.2 Deep Learning Techniques 21
3.3 Summary 22

4 MLSecAndroid Approach 23
4.1 MLSecAndroid Approach Overview 23
4.2 Data Collection Stage 24
4.3 Pre-processing Stage 27
4.4 Feature Extraction Stage 28
4.4.1 Android Application Permissions 28
4.4.2 System API Calls 30
4.4.3 Users Feedback 30
4.4.4 Other Features 32
4.5 Feature Selection Stage 32
4.6 Classification Stage 33
4.7 Training and Testing Models Stage 35

5 Experiments and Results 37
5.1 System Environment 37
5.2 Evaluation Criteria 37
5.3 Experiments and Results 39
5.3.1 Machine Learning Classifiers Evaluation Experiments 39
5.3.2 Features Selection Evaluation Experiments 40
5.3.3 Dataset Size Evaluation Experiments 41
5.4 Discussion 44

6 Conclusion and Future Work 47
6.1 Conclusion 47
6.2 Future Work 47

Bibliography 49

7 Appendix 55
7.1 Androguard - Get Permissions 55
7.2 Androguard - Get System API Calls 55
7.3 Aptoide - Get Android Applications Rating and Comments 56
7.4 Aptoide - Download Applications 56
7.5 VirusTotal - Testing Android Applications 56

List of Figures

1.1 New Android mobile malware from 2016 to 2018 [2] 1

2.1 Android System Architecture [7] 6
2.2 Android Activity Life-Cycle 7
2.3 Runtime permission request dialog. 10
2.4 Install time permissions dialog 11
2.5 Example of Rating and writing comment in Google Play store. 12

3.1 Android Malware Detection Techniques 17

4.1 General classical machine learning system overview. 23
4.2 MLSecAndroid Approach Overview 24
4.3 Collected Dataset Applications Types 25
4.4 Collected Dataset Malware Types 26
4.5 MySQL Database UML diagram 26
4.6 Permissions Analysis Result. 29
4.7 Malware applications terms .Vs frequency. 32
4.8 Support Vector Machine (SVM) Classifier. 34
4.9 k-Nearest Neighborss (KNN) Classifier. 34
4.10 Random Forest Classifier. 35

5.1 Number of Features .Vs Recall. 40
5.2 Size of the Dataset .Vs Recall. 42

List of Tables

2.1 Android Activity Life-cycle Methods. 7
2.2 Privacy/Read related permissions 9
2.3 Write/Modify related permissions 9

3.1 Summary of related-work based on classical machine learning techniques. 21

4.1 Dataset Statistics. Number of Samples of Each Class Over published Years in The Google Play Store. 24
4.2 File size binning 28
4.3 Number of downloads 28
4.4 Most Frequent Permissions 30
4.5 Most Frequent API Calls 30

5.1 Confusion matrix. 38
5.2 Classifiers Evaluation Results. 39
5.3 Best Number of Features Experiment Results. 41
5.4 Features categories Experiment Results. 41
5.5 Best Number of Applications Experiment Results. 43
5.6 Dataset Evaluation Based on Published Date Years Results 43

List of Abbreviations

APK	**A**ndroid **P**ac**K**age
OS	**O**perating **S**ystem
SAT	**S**tatic **A**nalysis **T**echniques
DAT	**D**ynamic **A**nalysis **T**echniques
HAT	**H**ybrid **A**nalysis **T**echniques
SVM	**S**upport **V**ector **M**achines
KNN	**K-N**earest **N**eighbors
LDA	**L**inear **D**iscriminant **A**nalysis
NN	**N**eural **N**etworks
SMS	**S**hort **M**essage **S**ervice
API	**A**pplication **P**rogramming **I**nterface
UI	**U**ser **I**nterface
ART	**A**ndroid **R**un-**T**ime
HAL	**H**ardware **A**bstraction **L**ayer
VM	**V**irtual **M**achine
PRS	**P**remium **R**ate **S**ervices
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
TP	**T**rue **P**ositive
TN	**T**rue **N**egative
CFG	**C**ontrol **F**low **G**raph
RNN	**R**ecurrent **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
ESN	**E**cho **S**tate **N**etwork
DBN	**D**eep **B**elief **N**etwork
NLP	**N**atural **L**anguage **P**rocessing

1 Introduction

The Personal Portable Devices (PPD) have become an inseparable component of most people's lives. For example, The smartphones provide many functions such as making calls, making payments, take a photo, and many other daily functions. These functions are nearly used by every person all over the world.

The smartphones are operate using different operating systems such as Android OS which is the most popular platform with more than 70% of market share. It has attracted a lot of attention because it applied in a large number of terminal platforms including mobile phones, computers, and panel computers. The Android market share will increase to 87.4% by the end of 2023 according to International Data Corporation (IDC) due to 5G launches [1].

The rapid growth of Android system usage every day leads to the appearance of new Android applications developed by attackers to attack the users of the Android operating system. These applications are called malware. The malware applications are uploaded to Android application stores such as Google Play store due to the weak policy of submitting application on these stores. Therefore, there are growing threats for mobile users by installing more malware applications without the ability to detect them before installing the applications to the user device. Figure 1.1 shows new Android malware appear from 2016 to 2018.

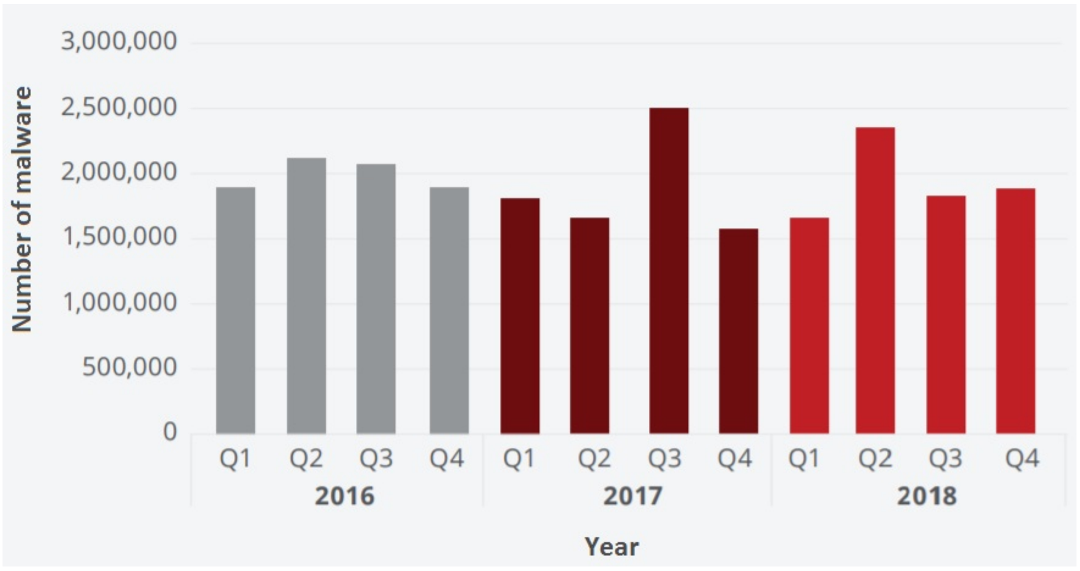

FIGURE 1.1: New Android mobile malware from 2016 to 2018 [2]

1.1 Research Problem and Motivation

Most users upload applications from Google Play Store which is supported by Google company. Google Play Store checks every uploaded application. Unfortunately, some malicious applications pass Google's test and get accepted.

In general, most Android devices contain anti-virus software, but around 60% of users do not use it [3]. The anti-virus software's offered solutions against malware, these solutions depend on scanning every app, monitoring traffic, tracking applications remotely, and so on. Also, it may be offered a data wipe if the phone has been stolen or lost. These software's are consumed the device's power, memory, and affect the device's performance. Moreover, it generally annoying the users since it appears as notifications and pop-ups. Therefore, many users do not use them and they do not have any protection methods against malicious applications.

The effect of malware is changing dramatically when the user downloads APK files from the internet or any source other than the Google Play store. There are many unofficial websites that role as third-party applications repositories, these websites provide applications to users. For example, Aptoide [4] which is a marketplace for Android applications, has more than 200 million users and it contains more than 900,000 Android applications.

The malware authors added malicious code to normal applications, Then repackaged them and uploaded them to the unofficial website. Therefore, when users downloading applications from these unofficial websites, they become victims of these malicious applications. Moreover, most users do not read the permissions when they download applications, only 17% of users read the permissions during installation [5]. When the user downloading APK files and tap the installation button, the application requests a list of permissions from users which might be sensitive such as access contacts information, camera, and microphone permissions. The studies show that most users do not pay attention and understand the requested permissions [5].

In summary, there is a potential risk when the users install the malware applications on their devices. This creates a need to find malware applications and avoid their risk. Therefore, the researchers and manufacturers make a great effort to build anti-malware detection systems, which have robust detection techniques.

1.2 Research Questions

The study gives answers to the following research questions:

- How much data is needed to build an effective Android malware detection system?
- What are the representative features to detect the malware in Android systems?
- What is the best machine learning technique for building anti-malware in Android system?

- What are the most permissions, and API calls used by benign/malware Android applications?

1.3 Objectives

The primary goal of this research is to develop an anti-malware approach that can detect malware applications in Android applications‘ marketplaces. The proposed approach use combination of machine learning techniques and a number of extracted features from the Android APK file and user feedback. It presents a malware detection method for any unknown Android application. While most of the previous research extracted features that are based on static features extracted from Android APK files such as permissions, and API calls only. In this research, we introduced MLSecAndroid which is an approach that combines features extracted from Android APK and user feedback with machine learning techniques that can be used to detect potential Android malware without the need for the application source code. A set of sub-goal for this research had been set as followed:

1. Design and implement a framework for data collection, data pre-processing, and feature extraction. The key elements of this framework are:
    - Implement the data collection service that downloads applications and extracts users‘ feedbacks from Android applications stores such as the Google Play Store and Aptoide Store.
    - Implement data cleaning service that clean collected data throw many steps that include data cleaning, data integration, and data transformation.
    - Implement a feature extraction service that extracts permissions, API calls, and other features from Android APK files, and extract user feedback features from user feedback texts using sentiment analysis tools and NLP techniques.
2. Evaluate the use of machine learning classifiers to identify potential Android malware based upon the features extracted in goal number 1.
3. Evaluate the features extracted in goal number 1.
4. Evaluate the dataset collected in goal number 1.

1.4 Contributions

Our contribution can be summarized in the following areas:

- **Features** :While most of the previous studies do not take into consideration the user feedback features, in this research, we do not depend only on static features such as permissions and API calls, but we also depend on Android application users rating and analysis user feedback comments in most common Android stores such as Google Play and Aptoide stores (see section 2.4).
- **Android Malware dataset [6]** : We generate a representative dataset of Android applications that contains more than 10 different categories with a total of 17100 benign and malware applications. The dataset is labeled using Virustotal which used more than 60 tools (see section 2.6.4).

- **New malware detection approach**: We propose MLSecAndroid, a new malware detection approach that is more efficient and accurate than available solutions (see section 4.1).

1.5 Thesis Organization

This thesis is further outlined as follows. In Chapter 2, we provide background about system architecture, permissions, system API calls, users feedback types, and malware categories. The related work discusses in Chapter 3. In Chapter 4, we discuss the Android malware detection proposed approach. In chapter 5 we discuss the experiments and results. Finally, we conclude our work and talk about future work in Chapter 6.

2 Background

In this chapter, we discuss the Android system. Section 2.1 is a brief explanation of the Android system architecture and components. Section 2.2 talks about Android typical permissions. Section 2.3 discusses Android system API calls. Section 2.4 introduces Android users feedback. Finally, Section 2.5 summarizes Android malware types.

2.1 Android System Architecture

The Android operating system is designed for Android devices, its a Linux-based, open-source, and run on a wide number of devices (mobiles, tablets, TVs, ...etc). Figure 2.1 shows the Android platform components [8].

2.1.1 System Applications

Android system has sets of core applications, such as Calendar, Email, SMS, Browsers. These applications come with Android and the Android users cannot delete them. The Android users can download applications, such as Facebook, Twitter, Linked-In from the official website (Google Play store) or any third party applications repositories.

2.1.2 Java API Framework

The Android OS provides a set of entire features through APIs. These APIs are written using JAVA language and it is the essential building blocks that the developer needs to build Android applications.
The Java API framework includes the following:

- **Activities**: The activities display one screen of the Android app's user interface, it's much like a form for a web page. The Android application may contain more than one activity. An example of Android activity is the Activity class which displays a Login Screen to the user. Figure 2.2 shows Android Activity life-cycle and Table 2.1 describes the methods that represent the states that the Android activity goes during its life cycle.

- **Content Provider**: The Content Provider shares application data that is saved in the SQLite database or other storage locations. It works in two scenarios which are access content providers in other applications or create new content providers in your application and allow other applications to access it [7].

- **Services**: The service is run in the background without user interaction to perform long-running tasks. For example, checking for new data, Internet downloads and updating content providers.

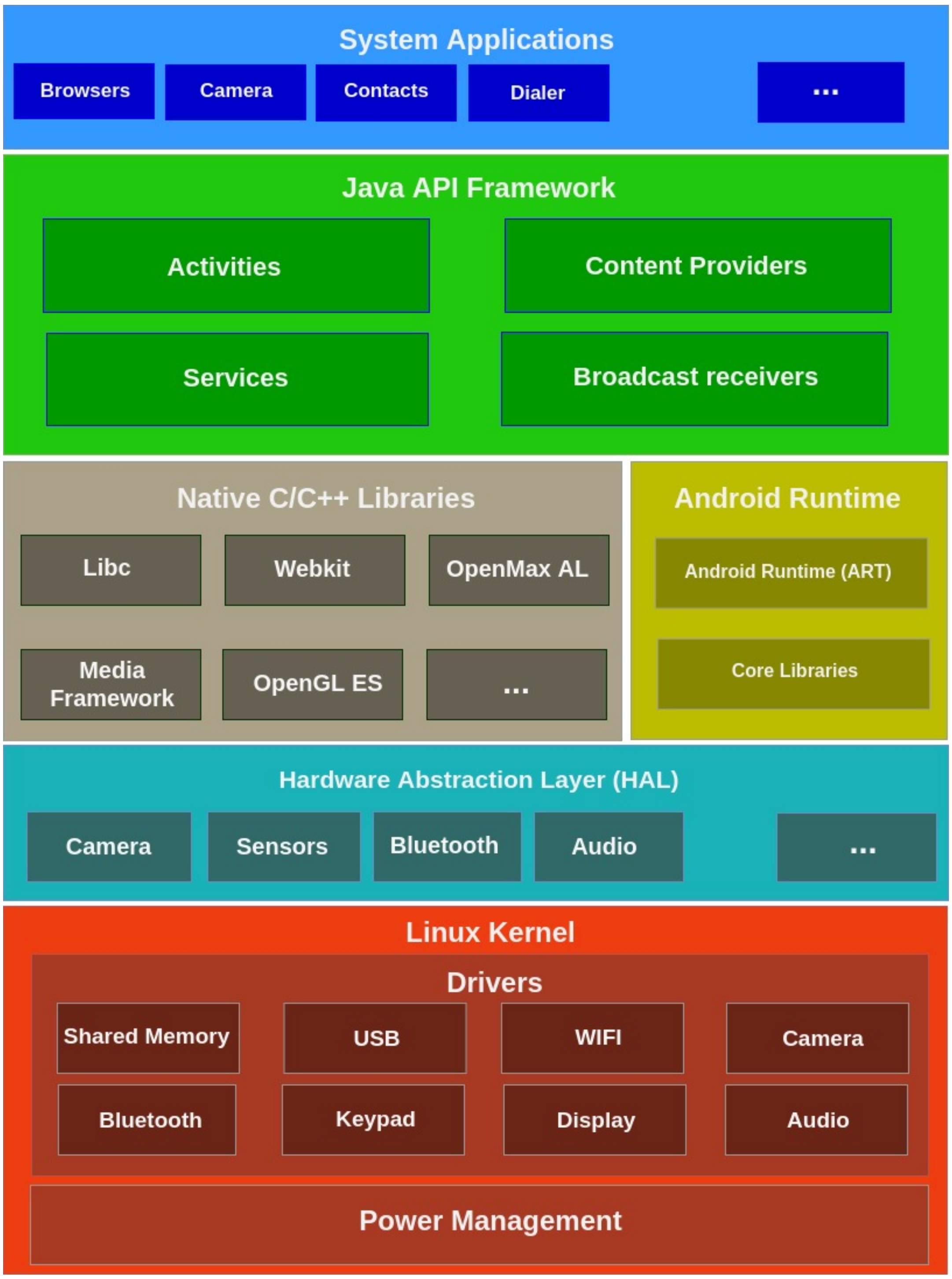

FIGURE 2.1: Android System Architecture [7]

- **Broadcast receivers**: This type contains components to receive and react to broadcast announcements. These broadcast announcements initiated by application code or other applications. For example, The WiFi connection establishes signals and a broadcast receiver for an email app.

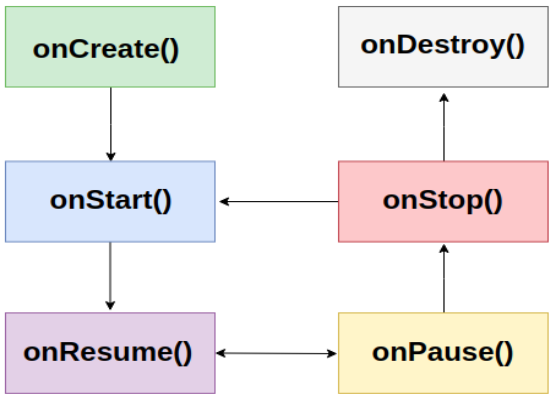

FIGURE 2.2: Android Activity Life-Cycle

TABLE 2.1: Android Activity Life-cycle Methods.

Method	**Description**
`onCreate()`	The method `onCreate()` is the first method that the Android OS called when an activity is created. it followed by onStart() method.
`onStart()`	The method `onStart()` called when the activity is ready and becoming visible to the user. It may be followed by `onResume()` method or `onStop()` method.
`onResume()`	The method `onResume()` called when the activity starts interacting with the device users. It followed by `onPause()`.
`onStop()`	The method `onStop()` called when the activity is no longer visible to the Android device user since there is another activity that has been resumed.
`onRestart()`	The method `onRestart()` called when the activity has been stopped and start again. It followed by `onStart()` method.
`onPause()`	The `onPause()` method called when the system is started resuming a previous activity. The implementations of this method should be very quick since the next activity will not be resumed until this method is finished.
`onDestroy()`	The `onDestroy()` method called before the activity is destroyed.

2.1.3 Native C/C++ Libraries

The Android system contains many components and services, such as Android Runtime (ART) that are built using native code which needs native libraries that written in `C` and `C++` programming languages. Examples of these libraries are Libc, Open-MAX AL, and OpenGL ES libraries.

2.1.4 Android Runtime

The Android Run-time (ART) is used to run multiple virtual machines on low-memory Android devices. The Android devices with Android version 5.0 or higher run each application in its instance of ART. The ART includes many features, such as it optimize garbage collection and its provide better-debugging support.

2.1.5 Hardware Abstraction Layer (HAL)

The HAL is provided interfaces for device hardware to Java API framework. It consists of many modules, each of them is interfaced with a specific type of hardware such as Bluetooth, Headphone, and Camera.

2.1.6 The Linux Kernel

The Android platform built based on the Linux kernel. This gives it the security features existing in a Linux system and allows the device manufactures to make hardware drivers depend on the well-known kernel.

2.1.7 Power Management

The Android devices with version 9.0 or higher have an improvement in device power management throw limiting applications access to device resources depend on the user usage pattern and restrict all applications when the user turned on the battery save mode [9].

2.2 Android Permissions

Android has a permission model that aims to protect Android users. This permission model prevents the Android application from access to sensitive information or uses device hardware without taking permission from the device users. All Android applications contain `AndroidManifest.xml` file that contains the list of permission that the application request. When the user downloads the application and tab install button, the Android system reads the permissions list from `AndroidManifest.xml` and display them to the user. Then it asks the user to allow or deny this application from getting this permission [10].

There are two types of permissions which are normal permissions and dangerous permissions. The normal permissions are granted to applications automatically at install time by the Android system and without user involvement, such as setting the time zone permission. The dangerous permissions require user agreement. The way that the Android system asks the user to grant this permission is depending on the Android OS version that is running on an Android device.

There are two ways to request dangerous permission which are: run-time and install-time. The Android system asks for real-time permissions in Android 6.0 (API 23) or higher. The user is not asked to grant the permissions in install-time, instead of that, the system asks users to grant dangerous permission when they operate the functions (Figure 2.3). In Android 5.1.1 (API 22) or lower, the Android system automatically displays the list of permissions to the user (Figure 2.4) and asks the user to grant all permissions at install-time [11]. There are many dangerous permissions some of them are more dangerous than others. We can divide them according to effects to the following [12]:

2.2.1 Privacy or Read Related Permissions

Many Android malware applications ask users to access private information that they may do not want to share with others. These permissions mostly read permissions. For example, The Camera application does not need to access the user's contact information. Table 2.2 display list of privacy/read related permissions.

TABLE 2.2: Privacy/Read related permissions

Method	Description
ACCESS_FINE_LOCATION	Access user exact location.
READ_CALL_LOG	Read user call log.
READ_CALENDAR	Read calendar information.
RECORD_AUDIO	Allow the application to record audio from the user device.
READ_SMS	Read user SMS messages.
RECEIVE_SMS	Receive SMS messages
CALL_PHONE	Allow the application to initiate a phone call without run the Dialer user interface.
CAMERA	Access all camera features.
GET_ACCOUNTS	Allows application to access the list of accounts in the Accounts Service.
READ_PHONE_STATE	Allow the application to read the device state and the phone number.
READ_CONTACTS	Read the user's contacts information.

2.2.2 Write or Modify Related Permissions

The write related permissions allow the Android application to write to the user device. For example, The malware application may write on the user device calendar. The modify related permissions allow the Android application to modify user device data. For example, the malware application may modify a contact number to a misleading fraud number. The Table 2.3 display list of Write or Modify related permissions.

TABLE 2.3: Write/Modify related permissions

Method	Description
WRITE_CALENDAR	Write data to the user's calendar.
WRITE_CONTACTS	Write data to the user's contacts.
WRITE_CALL_LOG	Write data to the user's call log.
SEND_SMS	Allow the application to send SMS message from user device.
WRITE_SETTINGS	Allows the application to read/write the system settings.
ADD_VOICEMAIL	Allows the application to add voicemails.

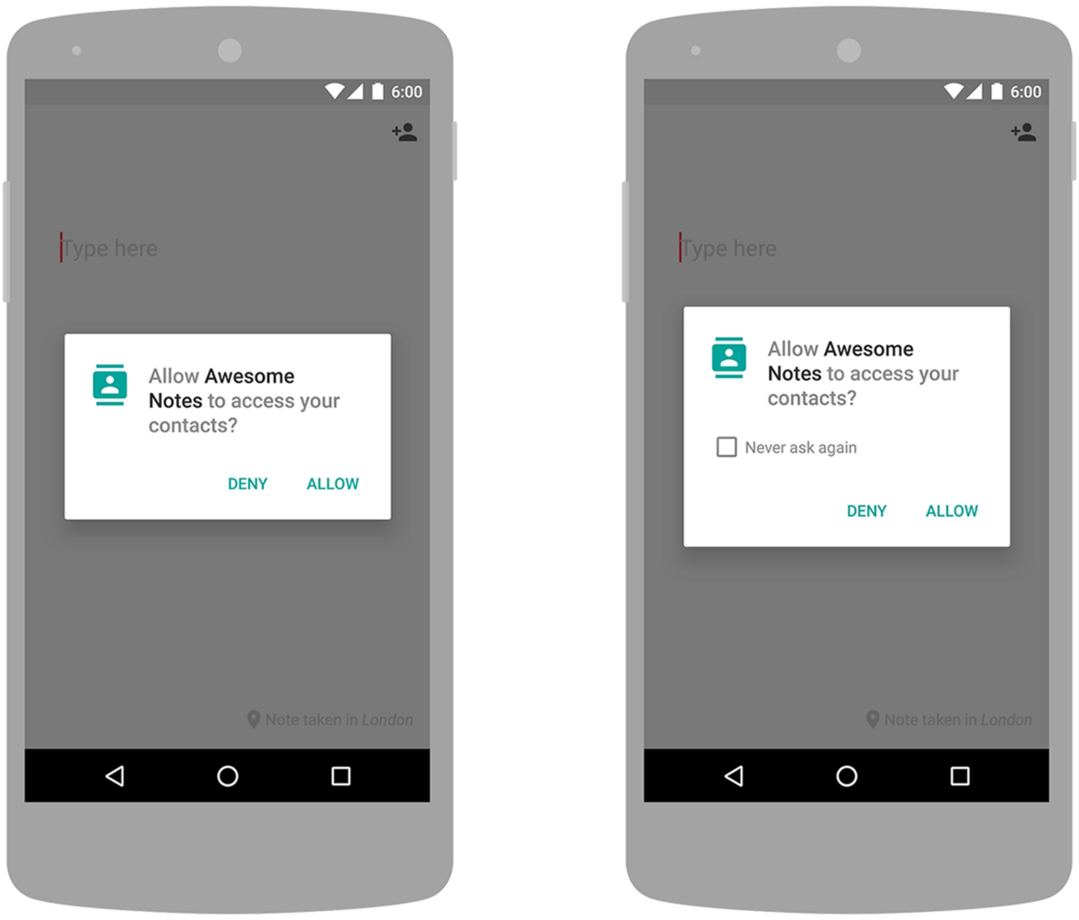

FIGURE 2.3: Runtime permission request dialog.

2.3 Android System API Calls

The system API calls feature represents all the functions used by Android applications to interact with Android OS. There is a large number of Android API calls. In this research, we analyze the malware applications and specified the list of API calls that frequently used by Android malware, such as `getSubscriberId()` and `getDeviceId()`. [13].

2.4 Android Users Feedback

The Android application stores, such as Google Play and Aptoide allow Android users to provide their feedback about the applications in two ways by rating application and writing comments.

- **Rating Application**: The user can rate an application by choosing 0 to 5 stars. Choosing 0 stars is the worst rating and it means that the user not satisfied with the app, whilst choosing 5 stars is the best rating and means that the user is very satisfied with the app.
- **Writing comments**: Besides the application rating, the users can write comments to ask a question about the application or provide feedback about their experience with the app. The feedback may be positive and will encourage

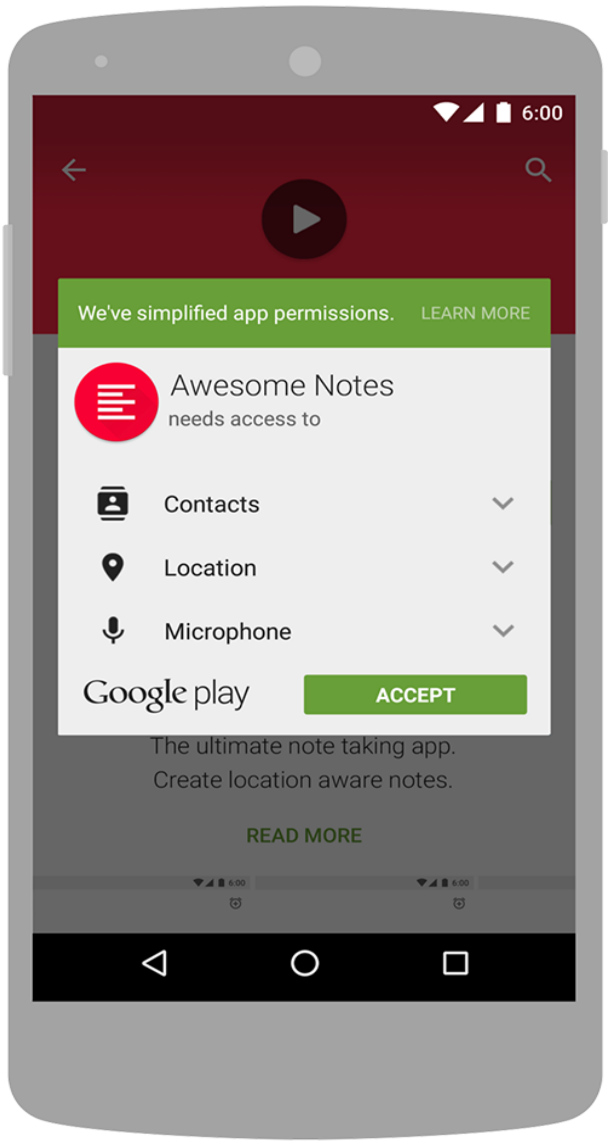

FIGURE 2.4: Install time permissions dialog

other users to use the app, or it may be negative and warned other users from using this app. Also, it may be neutral.

The bad rating and negative comments mean that the application has something wrong and may it has malicious behaviors. Figure 2.5 shows an example of how the user rate and writes the comment in the Google Play store.

2.5 Android System Malware

Android malware is developing rapidly. The first Trojan malware was discovered in 2010 [14] and since this time the Android malware becomes more complex and appears in different types. Android malware can be classified into five categories according to functionality which are phishing, financial charges, extortion, privilege escalation, and information collection [15, 14].

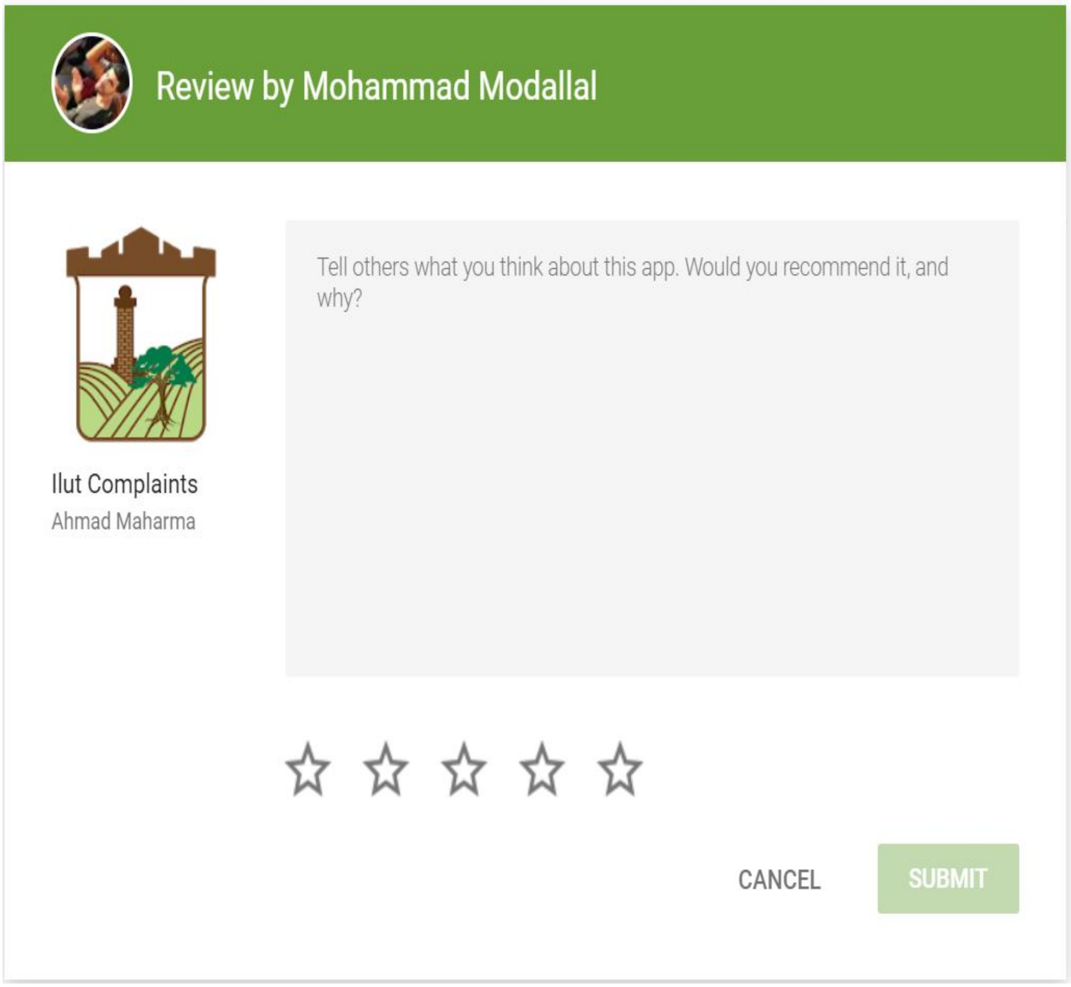
Review by Mohammad Modallal
Tell others what you think about this app. Would you recommend it, and why?
Ilut Complaints
Ahmad Maharma
CANCEL
SUBMIT

FIGURE 2.5: Example of Rating and writing comment in Google Play store.

- **Phishing**:
  The Phishing attack depends on social engineering and disguises the Android malware application to be as a normal app. This attack becomes more dangerous when it targets financial applications. For example, The SmiShing phishing attack that sends fake SMS to users, these SMS contains a link to the crafted webpage that can steal user credentials information.

- **Financial charges**:
  The Android malware that related to financial charges categories causes financial charges to users without their awareness. For example, the Premium Rate Services (PRS) are paid services that the Android users can buy by sending an SMS message to a telecom provider. The Android malware applications can send SMS from infected users devices that cause additional fees.

- **Extortion**:
  In this type of attack, the attacker accesses the important files in device storage and encrypt them. Then, the attacker asks the infected user to ransom for decrypting important files. The attacker may delete the files after receiving the ransom from the user.

- **Privilege escalation**: Although Android applications cannot access the user device resources until the user grant it the required permissions, but some Android Malware exceeds its privileges and circumvents the permission techniques throw indirect tactics to access. This can be done by malware that can access the normal applications that have privileges to access the device resources.
- **Information collection**: In addition to the above categories, There is a type of malware that collects different information from infected user devices, this includes contact information, SMS messages, and user accounts. For example, The SndApps application sends device information such as email addresses and phone numbers to a remote server.

2.6 Malware Detection Approaches and Tools

In this section, we discuss research tools and approaches that we use in our proposed approach. Section 2.6.1 discusses the Reverse Engineering approaches. Section 2.6.2 introduces the `Androguard` tool. Section 2.6.3 talks about Aptoide application store. Section 2.6.4 introduces VirusTotal tools. Section 2.6.5 discusses sentiment analysis approaches. Finally, Section 2.6.6 discusses Selenium tool.

The importance of detecting malware in Android devices encourages researchers and manufacturers to create new tools and techniques to detect this malware. Many anti-malware tools used static analysis, dynamic analysis, or hybrid analysis. In this chapter, we introduce the techniques and tools used for static code analysis.

2.6.1 Reverse Engineering

In general, Reverse engineering is a process of divide an object into parts to understand how it works. In this section, we introduce reverse engineering for Android applications which is the process of generating the source code from executable code. Reverse engineering allows the reverse engineer to understand the functioning of Android applications. Also, it allows him to modify the Android applications [16].

Dalvik bytecode is designed for the Android platform as follows: it was written in Java using Android API, then it compiled to Java bytecode, then it converted to Dalvik instructions. It is executed by the Dalvik VM. There are many tools to reverse Dalvik bytecode, such as undX [17] and Dex2Jar [18] that generate the `.jar` file from a `.apk` file. These tools work fine for simple applications, but they have difficulties when dealing with complex applications that have complex Dalvik bytecode.
In this research, we used the Androguard program which can reverse different Android applications by converting them from bytecode to readable code.

2.6.2 Androguard

Androguard is a tool written in python programing language to deal with Android `.apk` files. it can reverse different Android applications by converting them from bytecode to readable code [19].
It can play with the following files types:

- `.dex`, `.odex` files.
- Android application .apk files
- Android's binary XML files.
- Android resources files.

Androguard tool has many features such as it can reverse engineering of Android applications, disassemble of Android `.dex/.apk` files format, decompilation from Dalvik bytecode to Java code, and static analysis of the code, permissions, and system API calls.

2.6.3 Aptoide

Aptoide is a marketplace for Android applications. It allows users to own and manage their stores which is different than Google play which has a centralized store. It has more than 200 million users around the world. It is available in different languages. It contains more than 900,000 Android applications and it has over 7 billion applications downloads [4].

The Aptoide provides many APIs that allow developers to get Android application information such as application rating, number of application downloads, and user feedback comments. Appendix 7.2 shows Python programming language source code for getting applications rating.

2.6.4 VirusTotal

VirusTotal is a free online tool that allows users to check if the applications have malicious behaviors or not. It also provides users with domain/URL blacklisting services. Moreover, it provides developers with ready APIs that allow them to scan applications using any programming languages. VirusTotal consists of more than 70 anti-malware scanners [20].

In this research, we use VirusTotal in generating our Android malware dataset. We consider the majority result in classifying the Android applications into malware or normal applications.

2.6.5 Sentiment Analysis

The Sentiment Analysis is a process to analyze what people feel about a specific topic. It based on text analysis and it classifies the text to one of these classes: positive, neutral, and negative [21].

In this research, we use the open-source sentiment analysis tool called sentistrength to analyze the user feedback comments [22].

2.6.6 Selenium

Selenium is an open-source project that consists of a set of tools that automating web browsers for web applications for testing purposes. It supports many web browsers, such as Google Chrome, Mozilla Firefox, Safari and Internet Explorer. It also supports different programming languages such as `Python, JAVA, PHP, C#, PHP,`

and `Ruby`. It can be run in Windows, Linux or Mac operating systems [23].
In this research, we used Selenium to collect user's ratings and comments from the Google Play store website, because the Google Play store does not have ready APIs that can be used to get user's ratings and comments.

3 Related Work

In this chapter, we discuss the most recent researches related to the Android malware detection field. Section 3.1 discusses the non-machine learning available techniques. Section 3.2 discusses machine learning available techniques. Finally, Section 3.3 summarizes the related work.

Malware detection techniques can be categorized into two categories: non-machine learning and machine learning-based techniques (see Figure 3.1). Non-machine learning techniques include static, dynamic, and hybrid analysis. Machine learning-based techniques include classical machine learning and deep learning techniques.

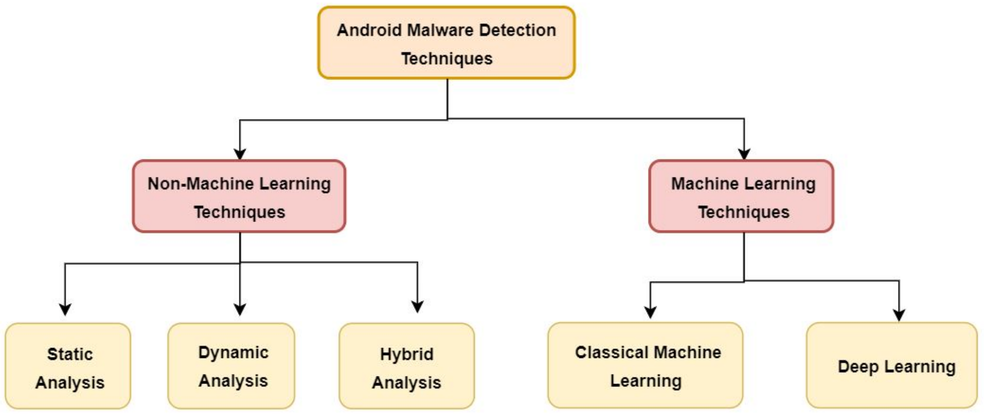
Android Malware Detection Techniques
Non-Machine Learning Techniques
Machine Learning Techniques
Static Analysis
Dynamic Analysis
Hybrid Analysis
Classical Machine Learning
Deep Learning

FIGURE 3.1: Android Malware Detection Techniques

3.1 Non-Machine Learning Techniques

In the past, non-machine learning techniques were used to detect malware. These techniques include static, dynamic, and hybrid analysis techniques. The static analysis technique detects malware without the need to execute the application in an emulator or device. It extracts static features from the disassembled code decompiled by specific tools. Dynamic analysis techniques detect malware by executing the application and monitoring the behavior during the execution. The hybrid analysis technique can be thought of like a mix of both static and dynamic techniques [24].

3.1.1 Static Analysis Techniques

There are many techniques that perform static analysis. Some of these techniques depend on binary file characteristics such as extracted bytecode sequence from the binary file and extracted opcode sequences after disassembling the binary file. Other techniques depend on assembly file characteristics such as extracting the control flow graph (CFG). In addition, some techniques based on the Android application signature which is a sequence of bytes that is unique for each Android application, these techniques depend on the malware signatures database and it required an up-to-date signatures database.

Christodorescu and Jha [25] proposed a technique that can detect malicious behavior of executable files by extracting the control flow graph (CFG) of executable files. The technique focus on obfuscation patterns of the malware.

Kang et al. [26] developed a fast malware detection system that uses static analysis to extract information from Android applications, such as the creator information (ex. the serial number of the certificate) and permissions. Additionally, it also uses a similarity score to classify the applications.

The static analysis techniques have many drawbacks. They cannot detect newly generated Android malware because the signature of new malware is unknown and the binary file structure is become more complex with increasing the number of features that Android devices support.

3.1.2 Dynamic Analysis Techniques

The dynamic analysis tracking Android applications behaviors by executing code in virtual machines (emulators). In this section, we review the most common Android malware detection researches based on dynamic analysis.

Enck et al. [27] built a malware detection system called `TaintDroid` which uses dynamic analysis techniques such as taint tracking. The system provides real-time analysis of Android applications by executing the applications in the virtual environment.

Kolbitsch et al. [28] presented a malware detection technique that uses dynamic analysis. They generate fine-grained models that capture the character of the application using system API calls and use a scanner to find similarity between unknown applications activity and these models.

The dynamic analysis techniques may fail in determining the amount of code that the application executed. Therefore, The malicious code part may not be executed which adversely affects the detection result. Also, The dynamic analysis techniques are hard to implement and need a lot of time for training.

3.1.3 Hybrid Analysis Techniques

We can collect the features from Android applications using static or dynamic analysis, or a combination of them. In this section, we discuss the most common research that used hybrid analysis in the detection of Android malware.

Spreitzenbarth et al. [29] presented a system called `Mobile-Sandbox` that automatically analyzes applications by static analysis and using the result of static analysis to guide the dynamic analysis. The system also logs calls to native APIs. The writer evaluated the system on over 36,000 Android applications.

Lindorfer et al. [30] built a framework called `ANDRUBIS` which combines static analysis and dynamic analysis. The static analysis uses the bytecode and `Manifest.xml` file and the dynamic analysis includes method tracing and system-level tracing.

Choi et al. [31] proposed a framework called `GATTACA` that is a combination of static analysis and dynamic analysis. The framework extracts Mal-DNA(Malware DNA) features which are hybrid characteristics of the Android applications. It contains three components which are a static analyzer, debugging-based behavior monitor, and classifier using Mal-DNA.

3.2 Machine Learning Techniques

To address the limitation in non-machine learning techniques, the researchers started to develop more efficient techniques based on data mining and machine learning. The machine learning techniques use many feature extraction, i.e. requested permissions, called API, and data representation. It also uses many classifiers, i.e. support vector machines (SVM), Random forest, the K-Nearest Neighbors algorithm (KNN), Naive Bayes, and deep learning approach, methods to build intelligence anti-malware.

3.2.1 Classical Machine Learning Techniques

Classical machine learning techniques repeatedly demonstrated its superior performance on a wide variety of tasks such as optical character recognition, natural language processing, vision, and playing games. This encourages researchers to use classical machine learning techniques in the Android malware detection field.

Xiaoqing, Wang, and Zhu [32] proposed an Android malware detection framework that depends on actually used permissions and system API call features. They used machine learning classifiers such as Random Forest, SVM, KNN, and AdaboostM1 for classification, and CFS feature selection algorithms. They used a 10-fold cross-validation approach to training and testing the system on their dataset that consists of 1205 benign applications and 1170 malware applications. The proposed system got very good results for the used dataset, but it cannot be generalized because the dataset is very small. Also, the writer only considers the actual permissions used and ignore the other permissions, this can make some errors in results.

Yerima, Sezer, and Muttik [33] proposed an Android malware detection approach based on a parallel combination of classification decisions obtained from each classifier. They extracted 179 features that include permissions, API calls, and commands related features from Android applications McAfee internal repository dataset. The dataset contains 3,938 benign applications and 2925 malware applications. The proposed approach utilizing many classifiers such as Decision Tree, Simple Logistic, and Naïve Bayes classifiers. The proposed system only depends on a few numbers of features (179 features) that are not representative and they cannot be used to detect all malware.

Afonso et al. [34] presented a malware detection system based on dynamic analysis features and machine learning classification techniques. The system extracts Android API calls and system call traces and evaluated the system with 7,520 Android applications. This system has a drawback of it can detect malware only if the applications have certain API level.

Eskandari, Khorshidpur, and Hashemi [35] proposed a novel feature extraction method that extracts API calls sequence using dynamic analysis. They utilize the N-grams technique to keep the order of API calls sequence.

Wei et al. [36] proposed a real-time malware detection tool called `Androidetect`. The tool is based on analyzing the relationship between system functions, permissions, and API calls. It applies many machine learning classifiers, such as Naive Bayesian and J48 decision tree. The writer train and test the tool by applying the 10-fold cross-validation technique in 200 samples (100 benign applications and 100 malware applications).

Hadad et al. [37] proposed an approach to detect malware using machine learning algorithms with extracted features from reviews and domain lexicons from computer security books. This work motivated us to use reviews written by users who have already been exposed to the application's security threats and malicious behaviors serve as important features for detecting malware application. However, this approach uses powerful user reviews features which can give higher detection rates when using it with other statics features such as permissions and API calls as proven in our research experiments.
Table 3.1 shows a summary of related work based on classical machine learning techniques.

TABLE 3.1: Summary of related-work based on classical machine learning techniques.

Paper	Year	Dataset	Features	Real time detect	Classifers	Meatures	Value
Sanz et al. [38]	2012	820 applications	- Uses permission - Uses feature - Number of ratings - Size of applications - Users Rating	Yes	- Bayesian Networks - Decision Trees - KNN - SVM	- AUC	Best Values 93%
Wu et al. [39]	2012	- 1500 benign - 238 malware	- API Calls - Permissions, - Intent message passing	Yes	KNN	- Accuracy - Recall - Precision - F-measure	97.87% 87.39% 96.74% 91.83%
Ham and Choi [40]	2013	- 30 benign - 5 malware	- Network - SMS - CPU - Power - Memory -Virtual Memory	No	- SVM - Naïve Bayesian - Logistic Regression - Random Forest	- Precision - Recall - F-Measure - ROC Area	Best values 99.6% 99.0% 99.3% 99.8%
Sanz et al. [41]	2013	- 1811 benign - 249 malware	- Permissions	Yes	- Simple Logistic - NaiveBayes - J48 - Random Forest	- Accurecy	Best Values 86.37%
Arp et al. [42]	2014	- 123,453 benign - 5,560 malware	- API calls - Permissions, - Hardware features - Network addresses	Yes	SVM	- Accurecy	94%
Liu and Liu [43]	2014	- 28548 benign - 1536 malware	- Permissions	Yes	J48 classifier	- Accurecy - Precision	98.6% 89.8%
Li, Ge, and Dai [44]	2015	- 350 malware - 350 benign	- API Calls - Permission	Yes	SVM	- Accurecy	81% 86%
Xu, Li, and Deng [45]	2016	- 12026 benign - 5264 malware	- ICC-related features	Yes	- SVM, - Decision Tree - Random Forest	- Accurecy - TPR - FPR	97.4% 93.1% 0.67%
Xiaoqing, Wang, and Zhu [32]	2016	- 1205 bengin - 1170 malware	- API Calls - Permission	Yes	- J48 - RandomForest - KNN - libSVM - AdaboostM1	- Accurecy - TPR	Best Values 95.% 99.6%
Milosevic and Nikola [46]	2017	- 200 benign - 200 malware	- Permission - Source code	No	- C4.5 - Random forest - Bayesian Networks - SVM - JRip - Logistic regression	- Precision - Recall - F-Score	Best values 87.9% 87.9% 87.9%
Hadad et al. [37]	2017	- 2170 bengin - 336 malware	- 128,863 Applications reviews	No	- C4.5 - Random forest - Decision Tree - Logistic regression	- TPR - FPR - Accuracy - AUC	Best values 34.8% 5.9% 86.7% 78.2%
Kakavand and Mohsen [47]	2018	- 200 bengin - 200 malware	- Frequency of keywords in manifest file	Yes	- SVM - KNN	- Accuracy - TPR	80.50% 80.00%
Yerima and Khan [48]	2019	- 22,378 bengin - 13,805 malware	- API calls - permissions -intents - other	Yes	- NB - J48 - SVM - RF - SL	- Accuracy - TPR - FPR - F-measure	Best values 91.0% 87.1% 06.3% 90.9%
Vinod, Zemmari, and Conti [49]	2019	- 3130 benign - 2520 malware	- permissions - system calls	No	- Random forest - Rotation forest -AdaBoost	- Accuracy - FPR - AUC	Best values 92.37% 7.6% 97.3%
Mateless et al. [50]	2020	- 24,553 benign - 60,000 malware	- permissions -APIs calls - source code analysis	No	- Random Forest - naïve Bayes - SVM	- Accuracy - TPR - AUC -F1-measure	Best values 98% 97% 99% 97%

3.2.2 Deep Learning Techniques

In recent years, the researchers used deep learning techniques in the Android malware detection area. These techniques can analyze Android applications automatically and they work well in detecting unknown applications. In this section, we review the most recent techniques.

Yuan, Lu, and Xue [51] implemented a malware detection engine called `DroidDetector` which based on deep learning techniques. The `DroidDetector` automatically checked Android applications depending on the static analysis and dynamic analysis features, such as required permissions, sensitive APIs, and dynamic behaviors. It used 1760 malware applications and 20,000 benign applications. The benign applications collected randomly from Google play store and maybe include malware applications.

Yuan et al. [52] proposed a deep learning-based method that uses over 200 features extracted from dynamic analysis and static analysis. It used a small dataset (250 malware and 250 benign applications) and achieve high accuracy of 96%.

Su et al. [53] utilize a deep learning approach to detect Android malware. They used 3,986 malware and 3,986 benign applications for training and testing. They extracted over 32,000 binary features that include five types of features: Requested permission, used permission, action, sensitive API calls, and application components.

Hou et al. [54] developed a malware detection tool called `Deep4MalDroid`. The tool based on dynamic analysis and it used the Component Traversal method that can execute the code automatically and extract system API calls. It applies deep learning techniques to weighted API call graphs. They tool trained and tested using real Android applications dataset collection from Comodo Cloud Security Center. The dataset consists of 1,500 malware and 1,500 benign applications.

Zhu et al. [55] built a `DeepFlow` malware detection tool. The `DeepFlow` detects malware based on the data flows in Android applications. It used the SUSI technique to transform the data flow features from method level to category level and it used the Deep Belief Networks (DBN) technique for classification. The DeepFlow used 8000 malware and 3000 benign applications in building the model and it achieved a high detection F-score of 95%.

Unlike classical machine learning techniques, the deep learning techniques have high computational calculations and need complex hardware such as GPU. Also, they need a lot of time for training and ruining the models. Moreover, they need a very large dataset to achieve high performance.

3.3 Summary

After we discussed the related work of the Android malware detection field, we can conclude that the available malware detection systems are not totally completed and every system has its drawbacks. Some available systems training the models using small dataset size, some of them extract a small number of features that cannot include all malware applications, some of them need a lot of time for training and testing, some of them need a lot of hardware to be implemented, and the other systems have a bad performance.

In this research, we created a new dataset that contains many malware types(Virus, Spyware, Trojan, Riskware, Adware, and others) and used them to build MLSecAndroid that able to identify Android malware using a combination of machine learning classification techniques and the number of extracted features (permissions, API calls, user feedback, and others).

4 MLSecAndroid Approach

In this chapter, we discuss the MLSecAndroid approach in details. Section 4.1 shows the proposed approach overview. Section 4.2 discusses the Android application data collection stage. Section 4.3 discusses pre-processing stage. Section 4.4 discusses the Android static feature extraction stage. Section 4.5 discusses feature selection stage. Finally, Section 4.6 discusses the classification techniques used.

4.1 MLSecAndroid Approach Overview

In general, the classical machine learning approaches consist of five stages (see Figure 4.1) which are data collection, data preprocessing, feature extraction, feature selec-tion, and classification [56].

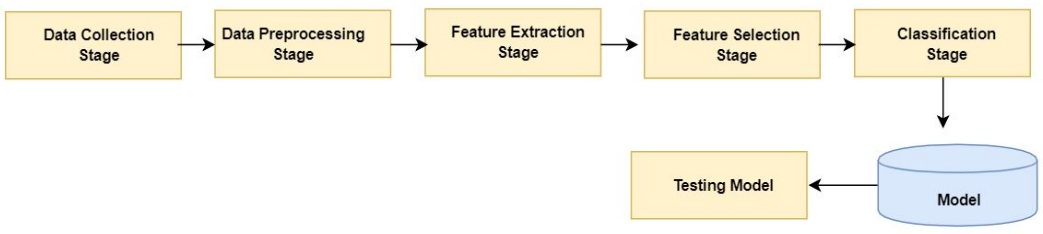

FIGURE 4.1: General classical machine learning system overview.

We use these stages to design MLSecAndroid approach that goes through a sequence of stages (Figure 4.2). The first stage is the Android application data acquisition where the Android APK files are collected from Android application stores. The second stage is the Android application pre-processing where the applications are filtrated using well-know Android malware detection tools. The third stage features extraction where static features, such as permissions, system API calls, user feedback are extracted from APK files and application stores. The fourth stage is the feature selection stage where the features result from the feature extraction stage are reduced to fewer relevant features set. The fifth stage is the classification stage where the machine learning model is trained. The last stage is evaluating the machine learning model using confusion matrics methods.

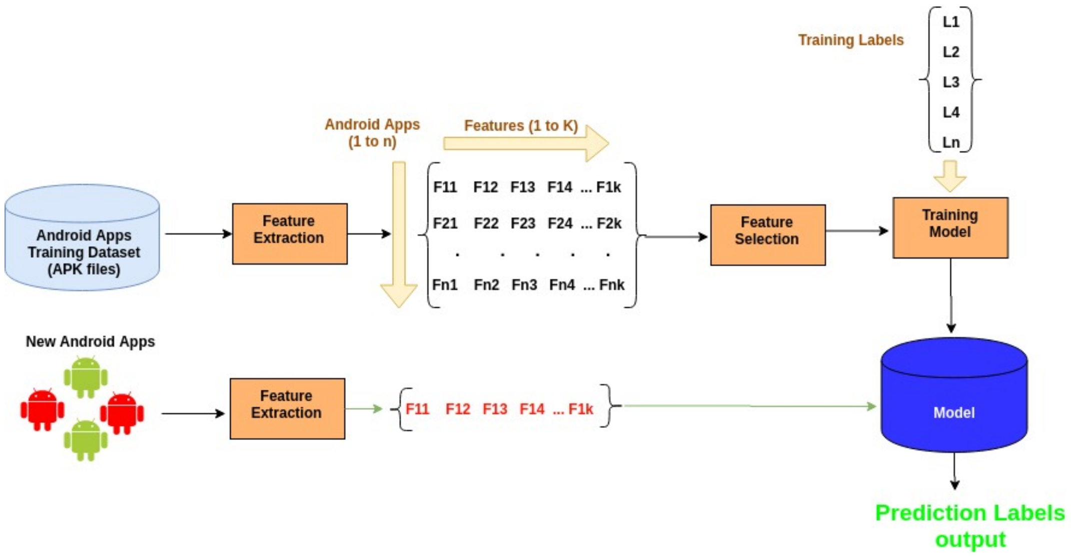

FIGURE 4.2: MLSecAndroid Approach Overview

4.2 Data Collection Stage

This stage aims to collect a representative dataset that will be used to build an anti-malware system. To the best of our knowledge, there is no publicly available dataset for Android applications along with their user's feedback reviews. The Malicious applications that officially reported are immediately removed from Android play stores. Therefore, we cannot obtain their users' feedback for analysis.

In order to collect available Android applications, a single crawler session was performed to extract the latest version for each application from the Aptoide Android application store for three months period (September to November 2019). The crawler uses randomly generated text to search for applications using Aptoide search and download APIs. The number of available applications is too large. Therefore, we selected a subset of applications' randomly. In total, we collected 17100 applications' APK files as shown in the table 4.1. Figure 4.3 shows the collected application types of distribution [6].

TABLE 4.1: Dataset Statistics. Number of Samples of Each Class Over published Years in The Google Play Store.

	2011	2012	2013	2014	2015	2016	2017	2018	2019	Total
Benign	567 (86.96%)	1997 (86.53%)	2567 (89.22%)	2937 (89.35%)	1398 (89.39%)	1179 (86.56%)	1982 (89.60%)	1245 (86.04%)	1109 (79.73%)	14981 (87.61%)
Malware	85 (13.04%)	311 (13.47%)	310 (10.78%)	350 (10.65%)	166 (10.61%)	183 (13.44%)	230 (10.40%)	202 (13.96%)	282 (20.27%)	2119 (12.39%)
Total	**652**	**2308**	**2877**	**3287**	**1564**	**1362**	**2212**	**1447**	**1391**	**17100**

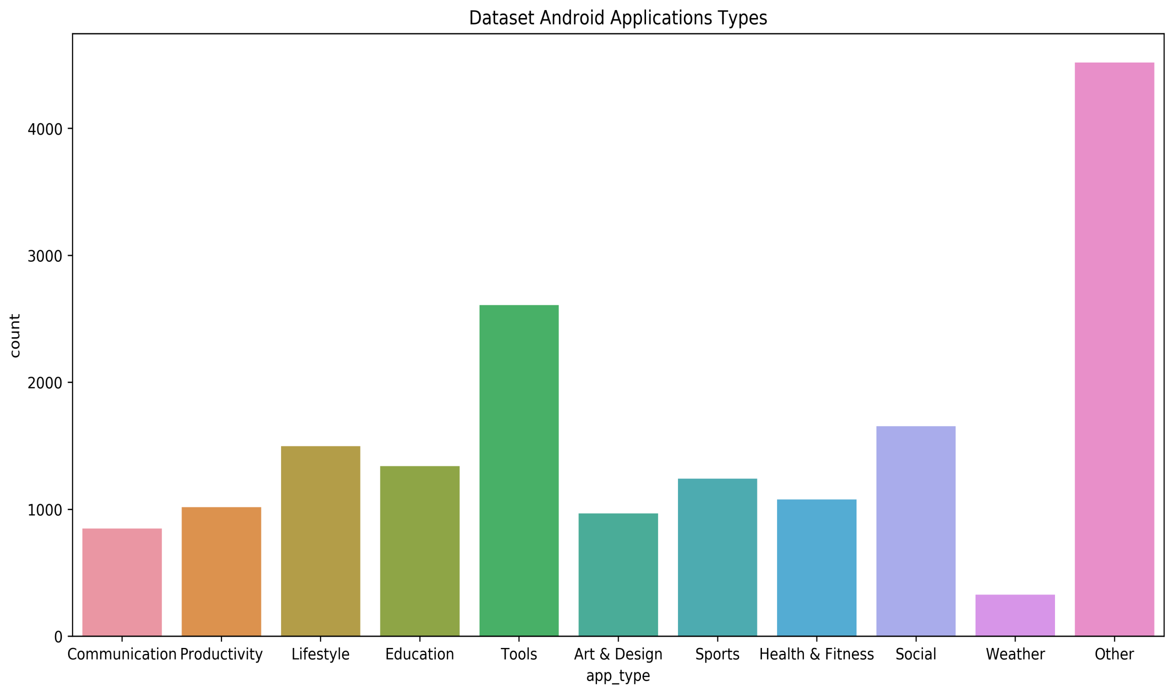

FIGURE 4.3: Collected Dataset Applications Types

For each collected application, two crawler sessions were performed to collect user feedback (reviews and comments). One of them collects the user feedback from the Aptoide store using Aptoide comments; API and The other crawler uses Selenium software to search for the application in Google play. Both of the two crawlers depend on the application package name. In total, we collected 62,329 users' feedbacks which includes reviews and comments.

The collected user feedback's comments were checked by Sentistrength [22], which is an online sentiment analysis tool that analyzes users' comments and returns the results of the analysis which is positive, neutral, or negative.

The collected applications' APK files were scanned by VirusTotal tools, which is an online scan engine that provides comprehensive reports regarding whether a given application APK file is malicious or benign with a malicious threat using different antivirus tools. There is a diversity in VirusTotal tools. Therefore, single APK may associate with different malicious families. Thus, the collected dataset contains many malicious families such as Virus, Spyware, Trojan, Riskware, Adware, and others less familiar malicious threats. Figure 4.4 shows the malware family distribution available in the collected dataset.

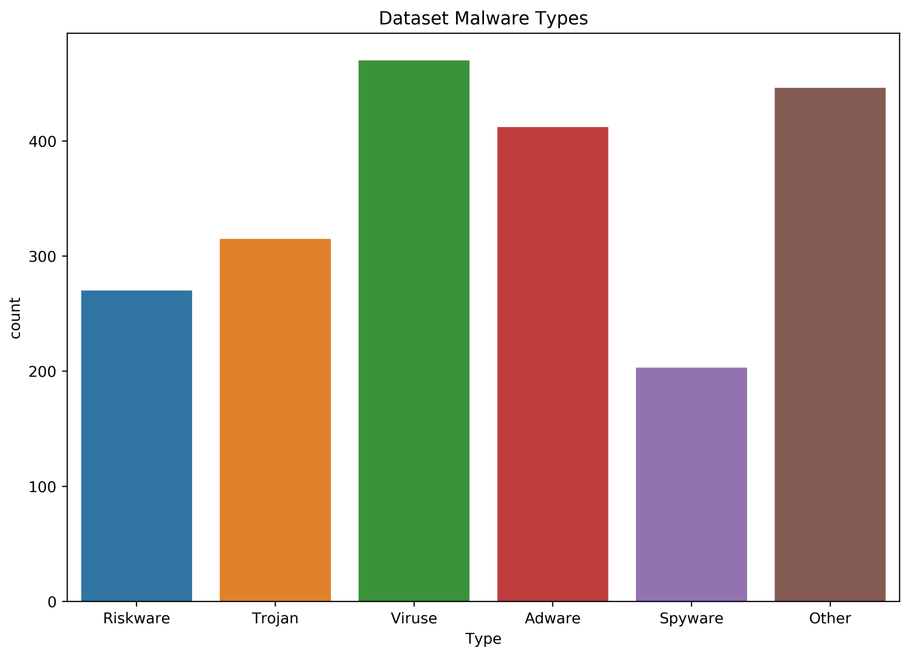

FIGURE 4.4: Collected Dataset Malware Types

The final collected data includes Android application basic information(application name, package name, size, published date, ...etc), permissions, API calls, user feedback(reviews and comments), and virusTotal analysis results. We store them in the MySQL database to make it easy to process them. Figure 4.5 shows the MySQL database schema UML diagram.

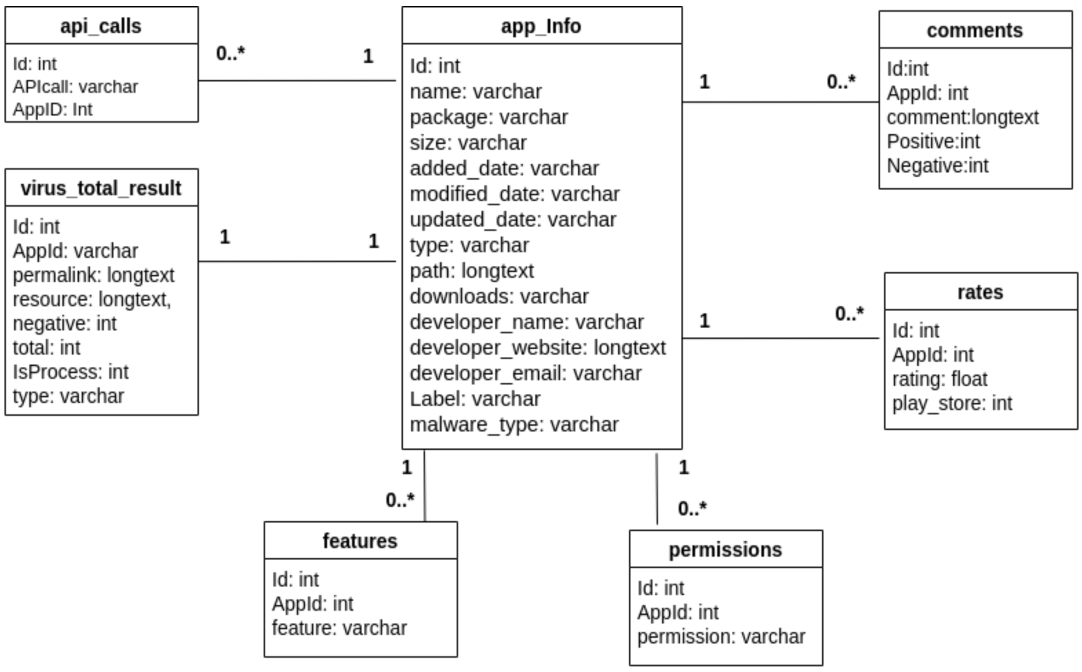

FIGURE 4.5: MySQL Database UML diagram

The following steps summarize the data collection stage:

- **Step 1**: Generate search keywords randomly.
- **Step 2**: Search for the keywords in Google play and Aptoide stores.
- **Step 3**: Download Android applications using Aptoide store ready Restful APIs (Appendix 7.4).
- **Step 4**: Get user‘s comments, application rates, and other information from Google play and Aptoide stores.
- **Step 5**: Analyze user comments using Sentistrength [22] sentiment analysis tool.
- **Step 6**: Extract permissions, features, and API calls from Android .APK files using Androguard tool (Appendix 7.1, and 7.2).
- **Step 7**: Get applications labels (malware or benign) using VirusTotal tools (Appendix 7.5).
- **Step 8**: Store all applications information in MySQL database tables.

4.3 Pre-processing Stage

After the data collection stage, the dataset might not be clean or not ready for processing. Therefore, we need a pre-processing stage to clean the data and prepare it for the feature extraction stage. In this research, the data pre-processing stage represented in the following steps:

- Data cleaning: The data cleaning can include many steps such as filling the missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies. We remove all records that have more than 50% missing values.
- Data integration: The data integration step combines data from multiple sources. We use the sentiment analysis tool called Sentistrength [22] to analyze user comments and store the result in the MySQL database. also, we use VirusTotal [20] tools to label the applications (benign or malware).
- Data Transformation: The data transformation maps the entire set of values of a given feature to a new set of replacement values. We use data transformation to bin the features that have a large range of values into discrete values. Table 4.2, 4.3 shows file size and number of binned downloaded features.

TABLE 4.2: File size binning

#	Value	Bin
1.	1-10000	1
2.	10001-50000	2
3.	50001-100000	3
4.	100001-500000	4
5.	500001-1000000	5
6.	1000001-5000000	6
7.	5000001-10000000	7
8.	10000001-100000000	8
9.	>100000001	9

TABLE 4.3: Number of downloads

#	Value	Bin
1.	0-20	1
2.	21-100	2
3.	101-500	3
4.	501-1000	4
5.	1001-5000	5
6.	5001-10000	6
7.	10001-50000	7
8.	50001-100000	8
9.	>100001	9

4.4 Feature Extraction Stage

To build machine learning models, the representative features need to be extracted from the collected dataset. The features type and feature extraction methods depend on the system type and they may vary from one system to another. For example, Natural Language Processing (NLP) systems may extract word n-grams and characters n-grams features, Optical Character Recognition (OCR) systems may extract the number of holes and pixels distribution in the image features, and speech recognition system may extract MFCC features.

In this research, four types of static features extracted from Android applications APK files and application stores which are permissions, system API calls, user feedback, and others.

4.4.1 Android Application Permissions

Android permissions aim to protect Android by preventing Android applications from access to sensitive information or use the device‘s hardware without taking permission from the device users. We discuss Android permissions in section 2.2. We used Androguard tool (section 2.6.2) to extract Android permissions.

The analysis of the collected dataset (Figure 4.6) shows that malware applications usually require more permissions than benign applications. More than 90% of the benign applications require less than 10 permissions, while about 66.4% of malware applications require more than 10 permissions.

There are many permissions requested by Android applications. Therefore, we select the most 150 frequent permissions requested by malware and the most 150 frequent permissions requested benign applications, 300 features in total. Appendix 7.1 shows the Python code that we used to extract Android Permissions.

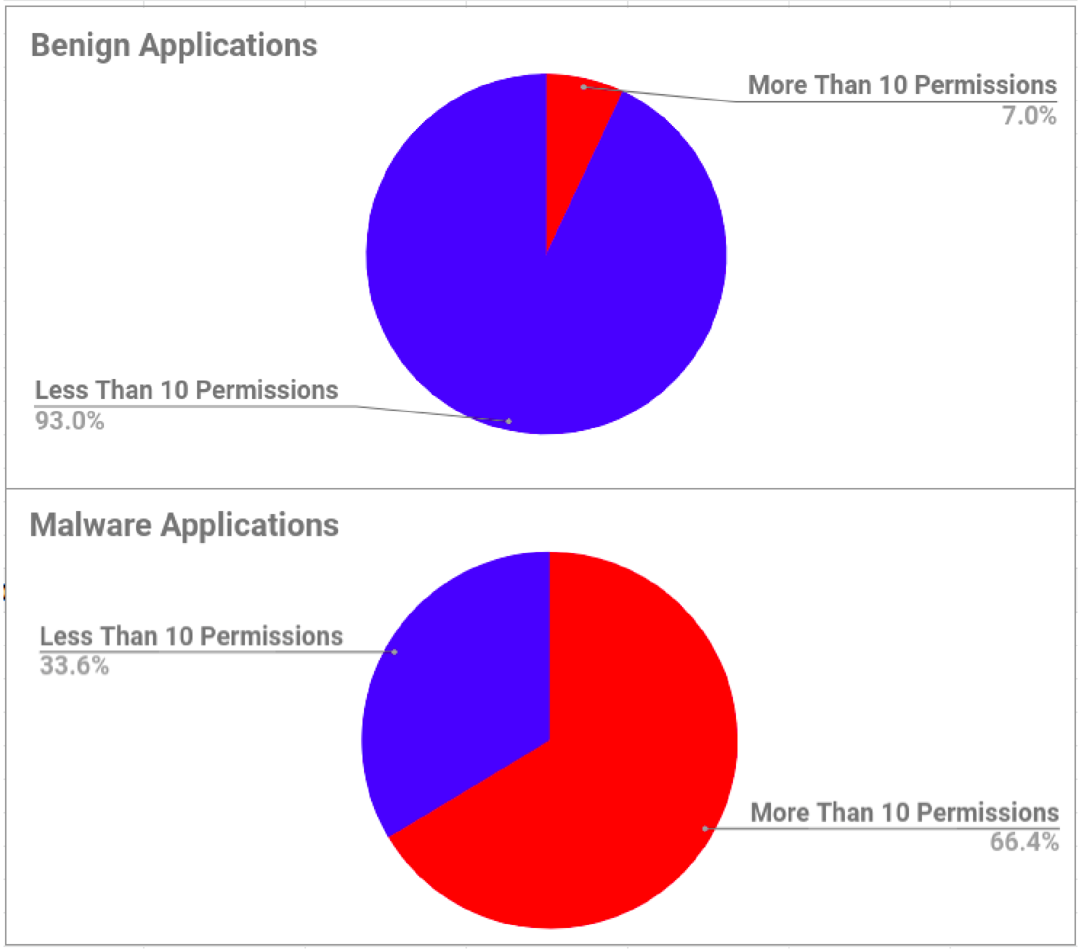

FIGURE 4.6: Permissions Analysis Result.

TABLE 4.4: Most Frequent Permissions

Permissions	Frequency
'android.permission.INTERNET'	10393
'android.permission.ACCESS_NETWORK_STATE'	7513
'android.permission.WRITE_EXTERNAL_STORAGE'	4481
'android.permission.ACCESS_WIFI_STATE'	2749
'android.permission.WAKE_LOCK'	2519
'android.permission.READ_PHONE_STATE'	2484
'android.permission.VIBRATE'	2213
'android.permission.ACCESS_FINE_LOCATION'	1951
'android.permission.RECEIVE_BOOT_COMPLETED'	1857
'android.permission.ACCESS_COARSE_LOCATION'	1620

4.4.2 System API Calls

We also extract system API call features which we discuss in section 2.3. There are many system API calls. Therefore, we select the most 1000 frequent system API calls using in malware and benign applications. Table 4.5 shows the most 10 frequent API calls used by Android applications in the collected dataset.
Appendix 7.2 shows the Python code that we used to extract Android system API calls.

TABLE 4.5: Most Frequent API Calls

API Call	Frequency
Ljava/lang/Object;-&gt;()V	16775
Landroid/app/Activity;->()V	15272
Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V	15081
Ljava/lang/StringBuilder;->append(Ljava/lang/StringBuilder;	14431
Ljava/lang/StringBuilder;->toString()Ljava/lang/String;	14536
Ljava/lang/String;-&gt;equals(Ljava/lang/Object;)Z	13279
Landroid/net/Uri;->parse(Ljava/lang/String;)Landroid/net/Uri;	12489
Ljava/lang/StringBuilder;-&gt;append(I)Ljava/lang/StringBuilder;	12469
Ljava/lang/StringBuilder;-&gt;()V	12411
Ljava/lang/String;-&gt;length()I	12013

4.4.3 Users Feedback

As we discussed in section 2.4, the user feedback is a powerful feature that can use with other features in detection Android malware applications. We get user app-rating and comments from Android applications stores like Aptoide and Google Play store. The Aptoide store provides developers with ready RESTFUL APIs that can be used to get user ratings and comments, whilst Google play doesn't support this type of APIs. To get user ratings and comments from Google play we use Selenium (section 2.6.6). Appendix 7.3 shows the Python code that we used to get user application ratings and comments.

We extracted three features from the user's feedbacks which are the user rate feature that depends on the user's rates, users comments analysis using sentiment analysis tool feature, and users comments analysis using the keywords list feature.

- **User rate feature**: The user can rate an application by choosing 0 to 5 stars and each application can have many user rates. The following algorithm is used to find the final rate feature value.

  Algorithm 1: Find users rates feature algorithm

  ```
  Result: users rates feature (FinalRate)
  if Number of users rates < 10 then
      FinalRate = MIN(rates);
  else
      FinalRate = FLOOR(AVG(rates));
  end
  ```

  The new applications may have a few numbers of rates and we cannot take the average value of these rates because the malware writer may add some rates with 5 stars for his malice application. Instead of that, we take the minimum rate as the final rate.

- **User comments analysis using sentiment analysis tool**: We analyze the user comments using the Sentistrength [22] tool. The tool takes the user's comments as input and returns the results of the analysis which is positive, neutral, or negative. Then, we use the analysis results to find the feature numeric values based on the majority. The applications with majority positive comments analysis take "0" value, the applications with majority neutral comments analysis take "1" value, and the applications with majority negative comments analysis take "2" value.

- **User comments analysis using keywords list**: The user can add comments to applications and each application can have many user's comments. also, the comments may be written in a different language and may have emojis and special characters. Therefore we should process the user comments and extract them as numeric values.

  We applied the following steps to find the user's comments feature:
  Step 1: Remove Non-English characters from the comments texts
  We chose the English language because it is the most common language that users use in their comments. The comments may contain non-English characters such as special characters, emojis, and other language characters. Therefore, we keep only English characters.

  Step 2: Remove duplicated characters
  The comments texts may contain words with duplicate characters. For example, the may write the word "virus" as "virussss". Therefore, we removed duplicate characters.

  Step 3: Extract the keywords List that indicates that applications have malicious behavior
  We extract the most frequent N-Grams from the user's comments. Then, we manually choose the terms that indicate that applications have malicious behavior. Figure 4.7 shows the most common terms.

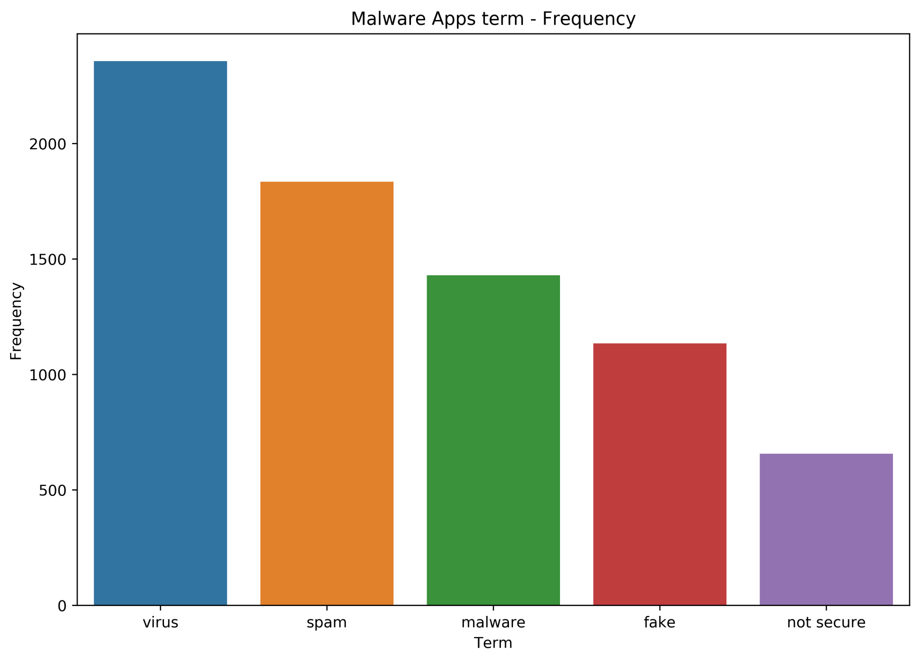

FIGURE 4.7: Malware applications terms .Vs frequency.

Step 4: Check the comments of the applications
We check the application comments, the applications that have comments contain one of the keywords List values take "1" value and the other applications take "0" value.

4.4.4 Other Features

In addition to previous features, we also extract another feature such as application type, the number of users rating the application, the number of downloads, publish date, and developer information.

4.5 Feature Selection Stage

Feature selection is a very critical component in the machine learning workflow. It is the process of selecting a subset of relevant features to transform high dimension dataset into low dimension dataset without loss of the total information. We need the feature selection stage for the following reasons [57]:

- The feature selection stage reduces training models time which increases exponentially with a number of features extracted from the dataset.
- In the feature extraction stage, the machine learning system may extract irrelevant features that increase the risk of overfitting especially when the number of features is too high. Therefore, feature selection can improve the accuracy of the system.

The feature selection techniques can be classified into three methods which are filter methods, wrapper methods, and embedded methods [58, 59].

In this research, the ANOVA F-value filter method is used to select the best N-features. It picks up the intrinsic properties of the features measured via univariate statistics test that can be used to select those features that have the strongest relationship with the output variable. It is appropriate for numerical inputs and categorical data, as our dataset. We also used a feature selection stage by selecting the most 1000 frequent system API calls and the most 300 frequent permissions from thousands of features.

4.6 Classification Stage

Machine learning problems required an automated decision. Is an email have spam content or not? Is a text written in the Arabic language or other languages? Is the customer able to pay the fees? All of these problems are classification problems.

Classification aims to predict which classes the new data samples belong to. For example, the binary classifier can predict whether the email is 0 or 1(spam/Ham). In some cases, we need a classifier with multi-classes, such as filtering Gmail inbox into "social", "primary", or "promotion".

There are many types of classifiers, such as SVM, Random Forest, DT, KNN, and AdaBoost classifiers. We use them to predict if Android applications have malicious behaviors or not.

We used six classifiers which work with labeled Android applications dataset to find a pattern to build a proper Android malware detection models. The classifiers are as follows:

- **Support Vector Machine (SVM) Classifier**: SVM is a classification method used by finding the hyperplane that maximizes the margin between the two classes. The hyperplane is defined by support vectors as shown in Figure 4.8. To perform the SVM algorithm, we define an optimal hyperplane by maximization of the width of the margin between classes and the minimization of the misclassifications [60].

- **K-Nearest Neighbor Classifier**: KNN classifier store all training data samples and classify new samples based on the distance measure. Figure 4.9 shows an example of the K-Nearest Neighbors classifier.
  In this experiment, 5 neighbors were chosen to perform this classifier.

- **Decision Tree Classifier**: The Decision Tree classifier work by creating a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

  In this experiment, we used a decision tree with a max depth equal 5, this is the default value.

- **Random Forest Classifier**: Random forests generate many decision trees depend on the random selection of the data and variables. The results of the Random Forest classification are based on the generated decision tree's majority results. It has many features, such as it can handle large datasets with high dimensions. Figure 4.10 shows an example of Random Forest classifier [61].

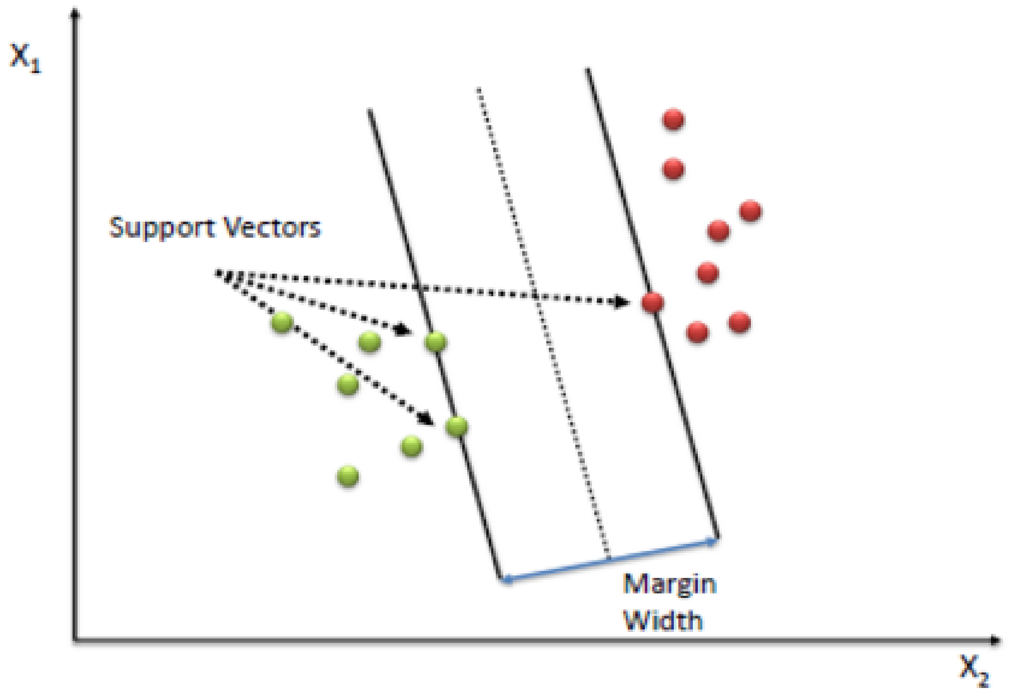

FIGURE 4.8: Support Vector Machine (SVM) Classifier.

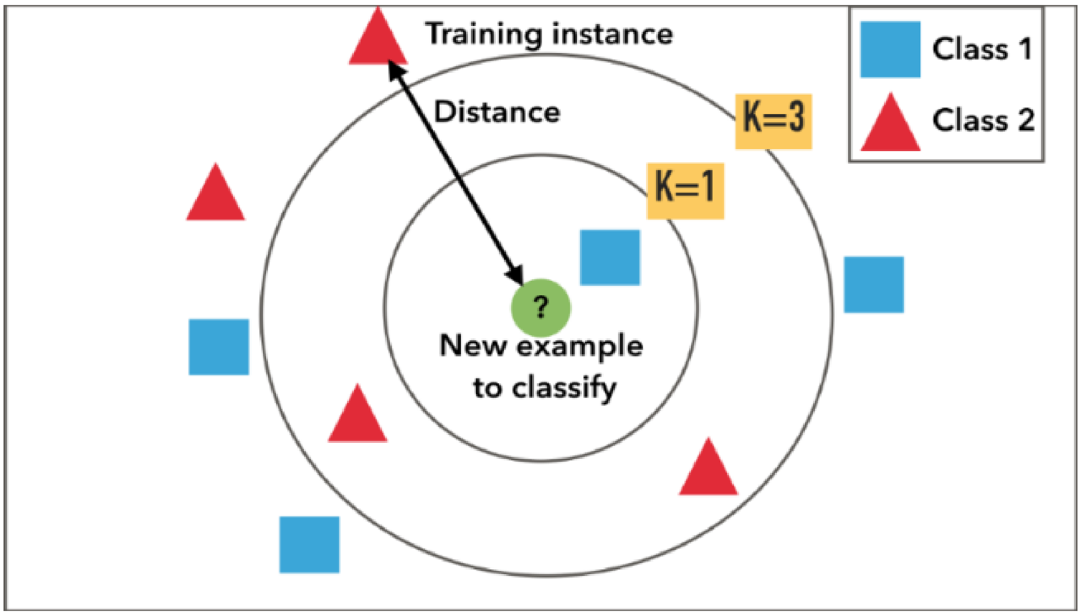

FIGURE 4.9: k-Nearest Neighborss (KNN) Classifier.

In this experiment, we use Random Forest with a number of trees in the forest (n_estimators) equals 10 and the maximum depth of the tree (max_depth) equals 5.

- **AdaBoost Classifier**: The Ada-boost classifier fitting weak classifier algorithm to form a strong classifier. The classifier combines multiple classifiers with a selection of training set at every iteration and assigning the right amount of weight in the final voting.

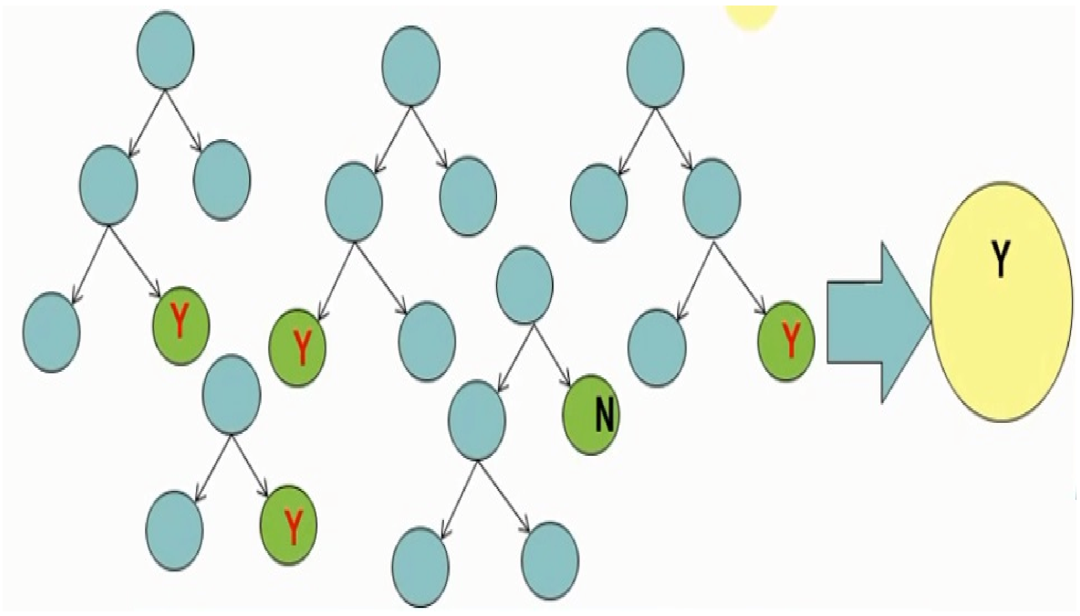

FIGURE 4.10: Random Forest Classifier.

In this experiment, we use the AdaBoost classifier with a maximum number of estimators (n_estimators) equal 50 and the learning rate (learning_rate) equals 1 which are the default values.

- **Naive Bayes Classifier**:The Naive Bayes classifier is one of the most successful learning algorithms which is based on the Bayes' theorem that assuming conditional independence between classes. Based on the rule, using the joint probabilities of sample observations and classes, the algorithm attempts to estimate the conditional probabilities of classes given an observation [62].

4.7 Training and Testing Models Stage

In this stage, we evaluate classifiers using two validation methods known as k-fold cross-validation and 70% split. The K-fold cross-validation is the most common way of dividing the dataset which divides the dataset into k disjoint subsets. Then, (K-1) of the subsets used for training the model and the remaining subset used for testing the model. The training process is repeated K times and in each time a new subset of the dataset used for testing and other subsets for training. Finally, the average of the training results considers the final result [63]. In this research, we use 10-fold cross-validation.

The second method is 70% split. In this method the 70% of the dataset used for training the model and 30% of the dataset used for testing. In this research, We trained the classifiers using 70% of each benign and malicious Android application vectors. The training set was randomly selected from the pool of benign and malicious. The data were randomly arranged in the dataset. We used the remaining of the applications 30% of the dataset for testing.
We found the Confusion matrix and use it to calculate the Accuracy, Recall, Precision, and F-measure(see section 5.2).

5 Experiments and Results

This chapter discusses the experiment results. The aims of experiments are to evaluate the Android malware detection model performance based on static features of the Android applications. The experiments use 1403 features that include permissions, API calls, users' feedbacks, and other features. Also, six machine learning classifiers applied in the experiments in order to determine which classifier can give the best classification accuracy. Moreover, we applied some experiments to evaluate our dataset.

This chapter organizes as follows, Section 5.1 shows the system environment specification that we use in this research. Section 5.2 display the evaluation criteria that we use to evaluate the machine learning models. In Section 5.3, the experiments and results are discussed in detail. Finally, Section 5.4 concludes the experiments.

5.1 System Environment

This experiment was thoroughly performed on a single Linux server. The server specifications are:

- Processor: 8 Core, 2.8-3.0 GHz each.
- Memory: 64 GB of RAM
- Operating System: Ubuntu 18.04.2 LTS.
- Machine Learning: sklearn.
- Python: Version 2.7.15+

5.2 Evaluation Criteria

After training the model, it can be tested using the testing dataset. The testing result shows the model performance. There are many metrics to evaluate machine learning models. In this research, we discussed and used the Confusion matrix [64] method which is valid to use in binary classification.
The Confusion matrix (see Table 5.1) describes the complete performance of the model. It contains four terms which are True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

- **True Positive (TP)**: The True Positive means that the predicted output is Yes and the actual output is Yes.
- **True Negative (TN)**: The True Negative means that the predicted output is No and the actual output is No.

- **False Positive (FP)**: The False Positive means that the predicted output is Yes and the actual output is No.
- **False Negative (FN)**: The False Negative means that the predicted output is No and the actual output is Yes.

TABLE 5.1: Confusion matrix.

	Predicted: No	**Predicted: Yes**
Actual: No	TN	FP
Actual: Yes	FN	TP

The Confusion matrix is used to find five metrics which are error rate, accuracy, recall, precision, and F-measure.

- **Error rate**: Error rate metric is defined as the number of all incorrect predictions divided by the total number of the samples. The best error rate result is 0 and the worst error rate result is 1. Equation 5.1 shows the error rate metric In terms of confusion matrix items. It also can be calculated by (1 - Accuracy).

$$ErrorRate = \frac{Number of incorrect prediction}{Total number of samples} = \frac{FP + FN}{TP + FP + TN + FN} \tag{5.1}$$

- **Accuracy**: The accuracy metric is used for evaluating classification models. It is the fraction of the samples that are successfully classified by the models. The best accuracy result is 1 and the worst accuracy result is 0. Equation 5.2 shows accuracy metric in terms of Confusion matrix items. It also can be calculated by (1 - Error Rate).

$$Accuracy = \frac{Number of correct prediction}{Total number of samples} = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.2}$$

- **Recall**: The recall metric is defined as the number of TP over the number of TP plus the number of FN. The best recall result is 1 and the worst recall result is 0. Equation 5.3 shows Recall metric in terms of Confusion matrix items.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

- **Precision**: Precision metric is defined as the number of TP over the number of TP plus the number of FP. The best precision result is 1 and the worst precision result is 0. Equation 5.4 shows precision metric in terms of Confusion matrix items.

$$Precision = \frac{TP}{TP + FP} \tag{5.4}$$

- **F-measure**: F-score is a harmonic mean of recall and precision. The best recall result is 1 and the worst recall result is 0. Equation 5.5 shows F-score metric In terms of Confusion matrix items.

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5.5}$$

5.3 Experiments and Results

The aim of the experiment is to evaluate the performance of classifiers using the features from the Android APK files such as API calls, permissions, and user feedback to build an accurate malware detection technique. We used six different machine learning classifiers included support vector machines SVM, RF, DT, KNN, AdaBoost, and Naive Bayes to evaluate the performance in this experiment. The result is shown in terms of accuracy, recall, precision, and F1-score. In general, the higher the number of measurements, the better the result is. There is a tradeoff between precision and recall, but we interested in recall more than precision. because of the effect of classifying the benign applications as malware is less than the effect of classifying the malware applications as benign applications.

We did different experiments to evaluate the six classifiers and to answer the research questions. These experiments use the 10-fold cross-validation and the 70% split evaluation methods.
This research includes the following experiments:

- Experiment 1: Machine Learning Classifiers Evaluation Experiments.
- Experiment 2: Features Selection Evaluation Experiments.
- Experiment 3: Dataset Size Evaluation Experiments.

5.3.1 Machine Learning Classifiers Evaluation Experiments

The aim of these experiments is to evaluate the classifiers performance and find the best classifier. The experiments evaluate six classifiers which are SVM, RF, DT, KNN, AdaBoost, and Naive Bayes classifiers. The experiments applied all dataset records (14981 benign and 2119 malware) with all features that include permissions, API calls, users' feedbacks, and other features (1403 features). The classifiers are evaluating using the 10-fold cross-validation and the 70% split evaluation methods. Table 5.2 shows the results.

TABLE 5.2: Classifiers Evaluation Results.

	10-fold validation				**2/3, 1/3 split validation**			
	Accuracy	**F-Score**	**Precision**	**Recall**	**Accuracy**	**F-Score**	**Precision**	**Recall**
Decision Tree	96.01	83.43	81.52	85.45	84.82	83.67	82.95	84.42
Random Forest	95.46	89.48	91.61	87.46	91.33	87.29	88.73	85.91
Nearest Neighbors	79.44	66.83	81.34	56.72	67.58	73.39	82.32	66.22
Linear SVM	**98.19**	**95.48**	**97.53**	**91.52**	**97.3**	**95.63**	**98.41**	**91.01**
AdaBoost	92.35	83.86	84.62	83.13	90.02	84.69	87.45	82.11
Naive Bayes	91.26	79.20	81.03	77.45	89.36	82.01	83.99	80.12

The classifiers are evaluated using four metrics which are accuracy, F-Score, precision, and recall. In the malware detection field, we interested in Recall more than other metrics, because of the effect of classifying the benign applications as malware is less than the effect of classifying the malware applications as benign applications.

The classifier with the best overall performance is SVM, Random Forest, Decision Tree, AdaBoost, Naive Bayes, and Nearest Neighbors respectively.

We also applied an experiment to evaluate the ability to use more than one classifier at the same time for detecting the malware. We used the SVM, Random Forest, and Decision tree models that result from the previous experiment as one model that its output is the majority results from the three models. The result of multi-models is Recall of 86.69% which is less than the result from the SVM classifier alone.

After evaluating the classifiers and finding the best one which is the SVM classifier, then we will use it in the next experiments.

5.3.2 Features Selection Evaluation Experiments

The aim of these experiments is to evaluate the features and find the best features vector. The experiments evaluate the features as follow:
1) The ANOVA F-value filter method is used to extract the best N-features.
2) The SVM classifier is training/testing using the selected N-features.
3) The 70% split evaluation method is used.
4) The Recall value is calculated and used as an evaluation metric.

We start with 100 features and increase them by 100 in each iteration until the total features 20000 features.
Figure 5.1 shows the features selection evaluation results. From the figure, we can conclude that using all features (1403 features) gives the best performance. Table 5.3 shows the details of the result.

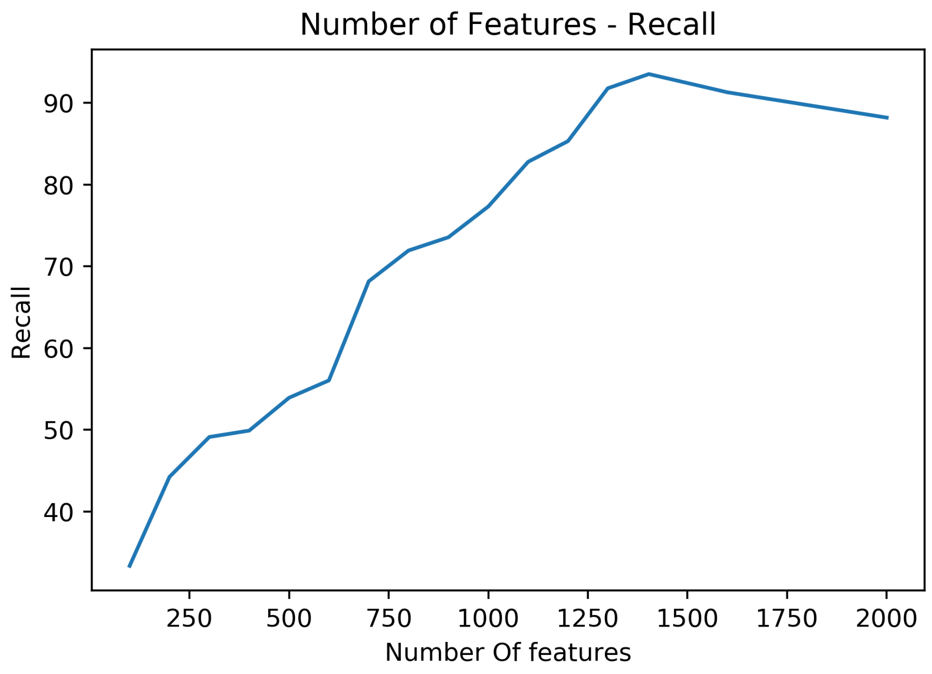

FIGURE 5.1: Number of Features .Vs Recall.

TABLE 5.3: Best Number of Features Experiment Results.

Experiment #	# of Features	Recall
1	100	32.34
2	200	44.23
3	300	49.11
4	400	49.89
5	500	53.91
6	600	56.03
7	700	68.16
8	800	71.93
9	900	73.55
10	1000	77.31
11	1100	82.78
12	1200	85.31
13	1300	91.78
14	1403	93.52
15	1600	91.29
16	2000	88.19

We also applied feature categories selection to evaluate the features categories that include permissions, API calls, users' feedbacks, and other features. We evaluate the features categories by training/testing the SVM classifiers using each features' category alone and then we use the combination of them. Table 5.4 shows the features categories evaluation, we can note that the users' feedbacks features have improved the overall performance.

TABLE 5.4: Features categories Experiment Results.

#	Features categories	# Of Features	Recall
1	Permissions only	300	44.06
2	API calls only	1000	55.84
3	Users' Feedbacks only	3	52.73
4	Permissions & API calls & other features	1400	85.89
5	Permissions & API calls & users' Feedbacks & other features	1403	93.52

5.3.3 Dataset Size Evaluation Experiments

The aim of these experiments is to evaluate the dataset size and find the best dataset size. The machine learning techniques work best when the number of samples in each class are about equal because they are designed to maximize accuracy and minimize error. The Android malware detection domain has an imbalanced class distribution. There are many more benign applications than malicious, because of the nature of Android application stores that contain a large percentage of benign applications. This causes a problem and affects the model's performance. There are many methods used to deal with the imbalanced dataset such as resampling method that contains two techniques over-sampling or under-sampling. The over-sampling define as adding more copies of the minority class and the under-sampling defined as removing some records from the majority class. We use a dataset that contains 14981 benign and 2119 malware which means that we have imbalance data problems. Therefore, we applied both the under-sampling and over-sampling techniques. The under-sampling technique reduces the benign class(majority class) records to become equal to the number of records in malicious class (minority class). In contrast, The over-sampling technique adds more copies of the malicious class (minority class).

We start training/testing the SVM classifier with 500 records(250 benign and 250 malware) and increase them in each iteration until the total dataset records 29962 records. All experiments use an equal number of benign and malware class records. Figure 5.2 shows the dataset size evaluation results. We find that the dataset that results from the under-sampling technique gives the best Recall value. Table 5.5 shows the details of the result.

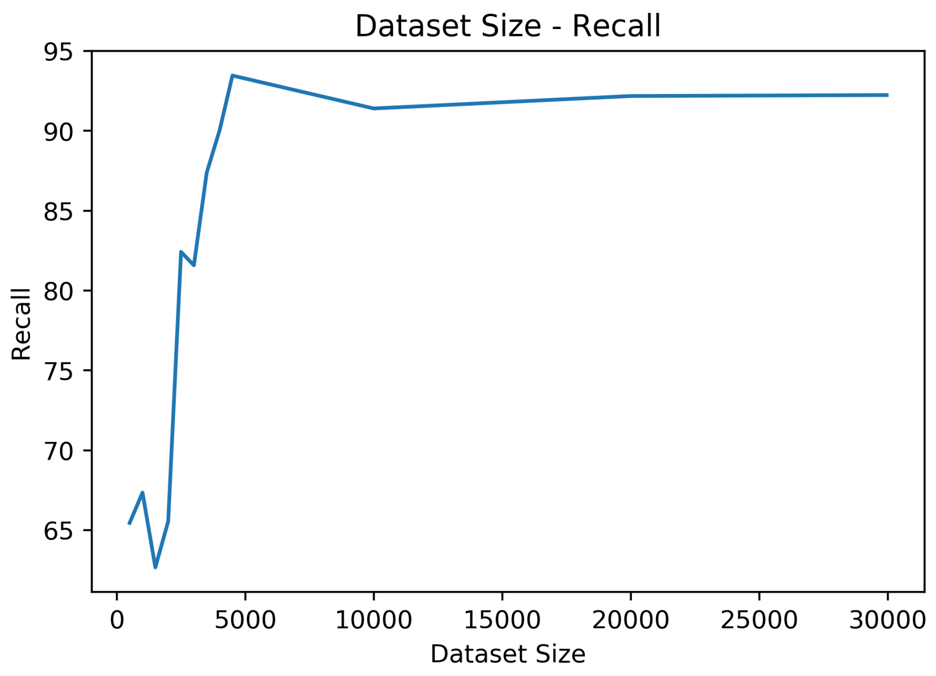

FIGURE 5.2: Size of the Dataset .Vs Recall.

TABLE 5.5: Best Number of Applications Experiment Results.

Experiment #	# of Applications	Recall
1	500	65.45
2	1000	67.35
3	1500	62.67
4	2000	65.55
5	2500	82.42
6	3000	81.58
7	3500	87.39
8	4000	90.03
9	4500	93.46
10	10000	91.40
11	20000	92.18
12	29962	92.24

We also evaluate the dataset based on published date years. The dataset contains Android applications published between the year 2011 and the year 2019. We use the one-year dataset for training the SVM classifier and the other years' datasets for testing. Table 5.6 shows the dataset evaluation based on published date years Results, we can note that the models that training using the last year's datasets have best results than the years near the year 2011. This may because the old applications haven't all-new features that exist on only new applications.

TABLE 5.6: Dataset Evaluation Based on Published Date Years Results

Training Data	**Validation**	**Recall/ Testing Data by Year**								
		2011	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**
2011	**10-fold**	-	62.84	45.58	54.53	61.57	62.42	50.81	66.91	66.79
	2/3, 1/3 split		59.91	51.88	53.73	69.22	55.25	60.15	48.76	70.63
2012	**10-fold**	68.88	-	49.88	46.21	61.55	65.56	55.46	47.28	58.00
	2/3, 1/3 split	52.38		66.85	70.75	70.01	59.10	66.86	65.75	56.13
2013	**10-fold**	70.28	50.69	-	49.63	44.30	57.28	51.12	52.57	46.30
	2/3, 1/3 split	47.02	53.81		60.41	67.20	69.63	52.79	42.83	45.38
2014	**10-fold**	62.19	66.82	69.31	-	72.43	70.21	73.82	63.83	63.85
	2/3, 1/3 split	62.29	66.78	74.59		61.35	60.42	74.51	66.91	65.24
2015	**10-fold**	73.97	66.35	66.88	74.54	-	63.52	68.67	72.60	65.30
	2/3, 1/3 split	60.76	60.80	74.44	74.84		65.69	71.72	74.87	62.99
2016	**10-fold**	71.42	64.61	62.13	75.59	61.24	-	71.50	65.15	67.25
	2/3, 1/3 split	61.13	70.45	73.32	72.76	69.94		66.41	60.51	66.53
2017	**10-fold**	60.55	64.05	65.99	74.73	73.10	73.75	-	74.51	63.64
	2/3, 1/3 split	69.00	64.18	64.79	70.26	67.24	75.62		71.85	63.94
2018	**10-fold**	70.25	62.56	72.81	65.73	75.57	61.08	68.64	-	69.55
	2/3, 1/3 split	67.27	61.87	72.82	67.33	66.81	65.23	68.37		64.69
2019	**10-fold**	79.15	79.99	73.25	78.17	83.12	81.50	76.89	72.20	-
	2/3, 1/3 split	83.24	80.72	74.43	79.76	70.76	73.87	79.79	70.06	

5.4 Discussion

The aim of this research is to propose anti-malware approach that can detect malware applications on the Android marketplaces. To achieve this goal, we applied many steps starting from data collection until machine learning models training, testing, and evaluation.

In order to collect representive dataset, we run python jobs on the Linux server for three months to collect data from Google Play and Aptiod stores. The dataset contains applications that published between year 2011 and year 2019. It has different application types such as educations, communications, and sports applications. Also, It contains many malicious families such as Virus, Spyware, Trojan, Riskware, and Adware. The collected dataset is good enough to build anti-malware models, but it has an imbalanced class distribution problem. This problem related to the Android malware detection domain where there are many more benign applications than malicious, because of the nature of Android application stores that contain a large percentage of benign applications. The dataset contains 14981 benign and 2119 malware. This causes a problem and affects the model's performance.

There are many methods used to deal with the imbalanced dataset such as resampling method that contains two techniques over-sampling or under-sampling. We try both of them and the under-sampling gives the best results.

In total, we only used 4232 applications for building the models. Using more samples should yield higher accuracy, but the processes of downloading applications, collecting user's feedbacks(application comments and rates), and extracting features from applications take a lot of time.

The collected dataset is preprocessed throw many steps that include data cleaning, data integration, and data transformation. Then, the processed dataset is used for extra representative features. We extract four types of features which are permissions, API calls, users' feedbacks, and other features. The permissions and API calls are strong features for detecting malware in Android systems and they achieve good performance when they applied with machine learning techniques. but, we can achieve higher performance when we use them with user feedback features. Thus, using a combination of permissions, API calls, and user feedback features with machine learning techniques can build an anti-malware for Android system with very good performance.

The feature extraction stage produces thousands of features. These features may contain irrelevant features that increase the risk of overfitting and increase model training time. Therefore, the dataset should transform from the high dimension dataset into low dimension dataset without loss of the total information. To achieve that, We applied the feature selection stage by selecting the most frequent permissions and API calls in both benign and malware applications. Then, we use ANOVA F-value filter method to select the best n-features. The results show that the feature vector with 1403 features achieves the best performance.

The feature selection stage produces feature-matrix. Then, This matrix is used for training, testing, and evaluation of the machine learning models. We evaluate six classifiers using two different types of evaluation methods, 10-fold cross-validation, and 70% split-validation and the results show that the classifier with the best Recall is Linear SVM, Random Forest, Decision Tree, AdaBoost, Naive Bayes, and Nearest Neighbors respectively. We interested in Recall metric more than other metrics, because of the effect of classifying the benign applications as malware is less than the effect of classifying the malware applications as benign applications.

In summary, the experiment results show that our detection method can identify Android malware applications with very good accuracy of 98.21% and recall of 93.52% for the best classifier. This is higher than the closest related work discuss in section 3.1. Further, the proposed approach requires 10 seconds for analysis of the Android APK file on average.

6 Conclusion and Future Work

6.1 Conclusion

In this research, we proposed a malware detection technique (MLSecAndroid) based on static features, such as Android applications requested permissions, system API calls, and user feedback. The proposed technique apply many types of machine learning techniques such as support vector machines (SVM), Random Forest (RF), Decision Tree(DT), K-Nearest Neighbors algorithm (KNN), and AdaBoost techniques. Also, we created a new Android application dataset that contains 17100 Android applications (14984 benign applications and 2116 malware applications). further, we survey the recent researches and related works in the Android malware detection field. Moreover, we analyze the collected Android applications and find the most frequent permissions and API calls used by both benign and malware applications. Experiment results show that the user feedback features have a large effect on model accuracy. We achieved an accuracy of 98.21% and a recall of 93.52% for the best classifier. This is higher than the closest related work.

6.2 Future Work

The number of Android malware increase day after day and new types of malware appears because of the increase in the number of features provided by Android devices. Android malware writers become more aware of anti-malware systems and they create new malware applications that are more complex and it's hard to detect. This will lead us to use more features such as network traffic features. Also, we will test the feasibility of integrating our proposed approach with dynamic detection techniques by extracting dynamic features such as resources' usage, network connections, and etc. In addition, we will implement the proposed approach on a large-scale level by collecting more Android applications. Moreover, we will support other languages especially the Arabic language in user feedback features.

Bibliography

[1] International Data Corporation(IDC). *Smartphone Challenges Continue in 2019, But 5G and Emerging Markets Will Bring Growth Back to the Market in 2020, According to IDC*. https://www.idc.com/getdoc.jsp?containerId=prUS45487719. Accessed on 2019-12-20.

[2] McAfee. "McAfee Mobile Threat Report 2019". In: (2019).

[3] Statista Inc. *Share of people who used anti-virus software on mobile phones in Denmark in 2016*. https://www.statista.com/statistics/649682/usage-of-mobile-anti-virus-software-in-denmark/. Accessed on 2018-11-19.

[4] Aptoide. *Android app store*. https://www.aptoide.com/. Accessed on 2018-11-19.

[5] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. "Android permissions: User attention, comprehension, and behavior". In: *Proceedings of the eighth symposium on usable privacy and security*. ACM. 2012, p. 3.

[6] Mohammad Modallal. *MLSecAndroid Dataset*. https://www.dropbox.com/sh/9nn0crhopk9m1xm/AABTkYHJiBK7JCOWmGj6FSLaa?dl=0.

[7] Android Developer. *Content provider basics*. https://developer.android.com/guide/topics/providers/content-provider-basics. Accessed on 2018-11-30.

[8] Android Developer. *Android Platform Architecture*. https://developer.android.com/guide/platform/. Accessed on 2018-11-30.

[9] Android Developer. *Power management*. https://developer.android.com/about/versions/pie/power. Accessed on 2018-11-30.

[10] Youchao Dong. "Android Malware Prediction by Permission Analysis and Data Mining". MA thesis. University of Michigan-Dearborn, 2015.

[11] android.com. *Permissions overview*. https://developer.android.com/guide/topics/permissions/overview. Accessed on 2018-11-25.

[12] Android Manifest permission. *Manifest.permission*. https://developer.android.com/reference/android/Manifest.permission. Accessed on 2018-12-1.

[13] Altyeb Altaher and Omar Mohammed Barukab. "Intelligent Hybrid Approach for Android Malware Detection based on Permissions and API Calls". In: *International Journal of Advanced Computer Science and Applications* 8.6 (2017), pp. 60–67.

[14] Guozhu Meng. "A Semantic-based Analysis of Android Malware for Detection, Generation, and Trend Analysis". PhD thesis. Doctoral dissertation, Nanyang Technological University, 2017.

[15] Xuxian Jiang and Yajin Zhou. "Dissecting android malware: Characterization and evolution". In: *2012 IEEE Symposium on Security and Privacy*. IEEE. 2012, pp. 95–109.

[16] Vibha Manjunath and Martin Colley. “Reverse engineering of malware on android”. In: *SANS Institute InfoSec Reading Room* (2011).

[17] sourceforge. *undX*. `https://sourceforge.net/projects/undx/`. Accessed on 2018-12-1.

[18] Kali tools. *dex2jar*. `https://tools.kali.org/reverse-engineering/dex2jar`. Accessed on 2018-12-1.

[19] androguard websit. *Androguard tool*. `https://androguard.readthedocs.io/en/latest/`. Accessed on 2018-12-1.

[20] VirusTotal. *Virustotal malware detection tool*. `https://www.virustotal.com`. Accessed on 2018-12-2.

[21] Yelena Mejova. “Sentiment analysis: An overview”. In: *University of Iowa, Computer Science Department* (2009).

[22] SentiStrength. *Sentiment Analysis Tool*. `http://sentistrength.wlv.ac.uk/`. Accessed on 2018-12-5.

[23] seleniumhq.org. *What is Selenium?* `https://www.seleniumhq.org/`. Accessed on 2018-12-10.

[24] Anusha Damodaran. “Combining Dynamic and Static Analysis for Malware Detection”. In: (2015).

[25] Mihai Christodorescu and Somesh Jha. *Static analysis of executables to detect malicious patterns*. Tech. rep. WISCONSIN UNIV-MADISON DEPT OF COMPUTER SCIENCES, 2006.

[26] Hyunjae Kang, Jae-wook Jang, Aziz Mohaisen, and Huy Kang Kim. “Detecting and classifying android malware using static analysis along with creator information”. In: *International Journal of Distributed Sensor Networks* 11.6 (2015), p. 479174.

[27] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. “TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones”. In: *ACM Transactions on Computer Systems (TOCS)* 32.2 (2014), p. 5.

[28] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, and XiaoFeng Wang. “Effective and Efficient Malware Detection at the End Host.” In: *USENIX security symposium*. Vol. 4. 1. 2009, pp. 351–366.

[29] Michael Spreitzenbarth, Felix Freiling, Florian Echtler, Thomas Schreck, and Johannes Hoffmann. “Mobile-sandbox: having a deeper look into android applications”. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM. 2013, pp. 1808–1815.

[30] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. v. d. Veen, and C. Platzer. “ANDRUBIS – 1,000,000 Apps Later: A View on Current Android Malware Behaviors”. In: *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. Sept. 2014, pp. 3–17. DOI: `10.1109/BADGERS.2014.7`.

[31] Young Han Choi, Byoung Jin Han, Byung Chul Bae, Hyung Geun Oh, and Ki Wook Sohn. “Toward extracting malware features for classification using static and dynamic analysis”. In: *Computing and Networking Technology (ICCNT), 2012 8th International Conference on*. IEEE. 2012, pp. 126–129.

[32] Xiaoqing, Junfeng Wang, and Xiaolan Zhu. "A Static Android Mal-ware Detection Based on Actual Used Permissions Combination and API Calls". In: *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering* 10.9 (2016), pp. 1547–1554.

[33] Suleiman Y Yerima, Sakir Sezer, and Igor Muttik. "Android malware detection using parallel machine learning classifiers". In: *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*. IEEE. 2014, pp. 37–42.

[34] Vitor Monte Afonso, Matheus Favero de Amorim, André Ricardo Abed Grégio, Glauco Barroso Junquera, and Paulo Lıcio de Geus. "Identifying Android malware using dynamically obtained features". In: *Journal of Computer Virology and Hacking Techniques* 11.1 (2015), pp. 9–17.

[35] Mojtaba Eskandari, Zeinab Khorshidpur, and Sattar Hashemi. "To incorporate sequential dynamic features in malware detection engines". In: *Intelligence and Security Informatics Conference (EISIC), 2012 European*. IEEE. 2012, pp. 46–52.

[36] Linfeng Wei, Weiqi Luo, Jian Weng, Yanjun Zhong, Xiaoqian Zhang, and Zheng Yan. "Machine learning-based malicious application detection of android". In: *IEEE Access* 5 (2017), pp. 25591–25601.

[37] Tal Hadad, Rami Puzis, Bronislav Sidik, Nir Ofek, and Lior Rokach. "Application marketplace malware detection by user feedback analysis". In: *International Conference on Information Systems Security and Privacy*. Springer. 2017, pp. 1–19.

[38] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, and Pablo Garcia Bringas. "On the automatic categorisation of android applications". In: *2012 IEEE Consumer communications and networking conference (CCNC)*. IEEE. 2012, pp. 149–153.

[39] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. "Droidmat: Android malware detection through manifest and api calls tracing". In: *2012 Seventh Asia Joint Conference on Information Security*. IEEE. 2012, pp. 62–69.

[40] Hyo-Sik Ham and Mi-Jung Choi. "Analysis of android malware detection performance using machine learning classifiers". In: *2013 international conference on ICT Convergence (ICTC)*. IEEE. 2013, pp. 490–495.

[41] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. "Puma: Permission usage to detect malware in android". In: *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*. Springer. 2013, pp. 289–298.

[42] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. "Drebin: Effective and explainable detection of android malware in your pocket." In: *Ndss*. Vol. 14. 2014, pp. 23–26.

[43] Xing Liu and Jiqiang Liu. "A two-layered permission-based android malware detection scheme". In: *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE. 2014, pp. 142–148.

[44] Wenjia Li, Jigang Ge, and Guqian Dai. "Detecting malware for android platform: An svm-based approach". In: *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*. IEEE. 2015, pp. 464–469.

[45] Ke Xu, Yingjiu Li, and Robert H Deng. “Iccdetector: Icc-based malware detection on android”. In: *IEEE Transactions on Information Forensics and Security* 11.6 (2016), pp. 1252–1264.

[46] Milosevic and Nikola. “Machine learning aided Android malware classification”. In: *Computers & Electrical Engineering* 61 (2017), pp. 266–274.

[47] Kakavand and Mohsen. “‘Application of machine learning algorithms for Android malware detection”. In: *Proc. CIIS* (2018), pp. 32–36.

[48] Suleiman Yerima and Sarmadullah Khan. “Longitudinal performance analysis of machine learning based Android malware detectors”. In: 2019.

[49] P Vinod, Akka Zemmari, and Mauro Conti. “A machine learning based approach to detect malicious android apps using discriminant system calls”. In: *Future Generation Computer Systems* 94 (2019), pp. 333–350.

[50] Roni Mateless, Daniel Rejabek, Oded Margalit, and Robert Moskovitch. “Decompiled APK based malicious code classification”. In: *Future Generation Computer Systems* (2020).

[51] Zhenlong Yuan, Yongqiang Lu, and Yibo Xue. “Droiddetector: android malware characterization and detection using deep learning”. In: *Tsinghua Science and Technology* 21.1 (2016), pp. 114–123.

[52] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. “Droid-sec: deep learning in android malware detection”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 44. 4. ACM. 2014, pp. 371–372.

[53] Xin Su, Dafang Zhang, Wenjia Li, and Kai Zhao. “A deep learning approach to android malware feature learning and detection”. In: *Trustcom BigDataSE I. SPA, 2016 IEEE*. IEEE. 2016, pp. 244–251.

[54] Shifu Hou, Aaron Saas, Lifei Chen, and Yanfang Ye. “Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs”. In: *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*. IEEE. 2016, pp. 104–111.

[55] Dali Zhu, Hao Jin, Ying Yang, Di Wu, and Weiyi Chen. “DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data”. In: *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE. 2017, pp. 438–443.

[56] P. Bruce and A. Bruce. *Practical Statistics for Data Scientists*. O’Reilly, 2017.

[57] Pat Langley et al. “Selection of relevant features in machine learning”. In: *Proceedings of the AAAI Fall symposium on relevance*. Vol. 184. 1994, pp. 245–271.

[58] Matthew Shardlow. “An analysis of feature selection techniques”. In: *The University of Manchester* (2016), pp. 1–7.

[59] Avrim L Blum and Pat Langley. “Selection of relevant features and examples in machine learning”. In: *Artificial intelligence* 97.1-2 (1997), pp. 245–271.

[60] Justin Sahs and Latifur Khan. “A machine learning approach to android malware detection”. In: *Intelligence and security informatics conference (eisic), 2012 european*. IEEE. 2012, pp. 141–147.

[61] M Alam and S Vuong. “Random Forest Classification for Android Malware”. In: *Proceedings of IEEE International Conference on Internet of Things*. 2013.

[62] Ivan Firdausi, Alva Erwin, Anto Satriyo Nugroho, et al. “Analysis of machine learning techniques used in behavior-based malware detection”. In: *2010 Second international conference on advances in computing, control, and telecommunication technologies*. IEEE. 2010, pp. 201–203.

[63] Thomas G Dietterich. “Ensemble methods in machine learning”. In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[64] M Hossin and MN Sulaiman. “A review on evaluation metrics for data classification evaluations”. In: *International Journal of Data Mining & Knowledge Management Process* 5.2 (2015), p. 1.

7 Appendix

7.1 Androguard - Get Permissions

The following code used to analyze and get Permissions from Android APK files using the Androguard program.

```
from androguard import misc
from androguard import session
import XlsxFileFunctions
# Get Android application permissions
ListOfPermissions=[]
sess = misc.get_default_session()
# Use the session
a, d, dx = misc.AnalyzeAPK(fileName, session=sess)
permissions=a.get_permissions()
for perm in permissions:
        ListOfPermissions.append(perm)
```

7.2 Androguard - Get System API Calls

The following code used to analyze and get system API calls from Android APK files using the Androguard program.

```
from androguard import misc
from androguard import session
import XlsxFileFunctions
from androguard.misc import AnalyzeAPK
# Get Android system API calls
systemAPI_list=[]
a, d, dx = AnalyzeAPK("AndroidAPK_Filename.apk")
for m in dx.find_methods():
orig_method = str(m.get_method())
        systemAPI_list.append(orig_method)
```

7.3 Aptoide - Get Android Applications Rating and Comments

The following code used to get Android applications rates and comments from Aptoide applications store. The code used ready API provided by Aptoide.

```
import json
import requests
BASE_URL="https://ws75.aptoide.com/api/7/app/getMeta/package_name="
COMMENTS_URL="https://ws75.aptoide.com/api/7/comments/get/package_name="
response = requests.get(BASE_URL+app_package_name)
app_inform_obj = json.loads(response.text)
if app_inform_obj['info']['status'] in "OK":
     #Get Application Rating
     print "Avarge Rate: "+str(app_inform_obj['data']['stats']['rating']['avg'])

     #Get user comments
     response = requests.get(COMMENTS_URL+app_package_name)
     app_comments_list = json.loads(response.text)
     if app_comments_list['info']['status'] in "OK":
         for i in range(app_comments_list['datalist']['count']):
               print app_comments_list['datalist']['list'][i]['body']
else:
     print "application not exist"
```

7.4 Aptoide - Download Applications

The following code used to download applications from the Aptoide application store.

```
import urllib2
print('Beginning file download .. ')
url = 'http://pool.apk.aptoide.com/savou/filename.apk'
filedata = urllib2.urlopen(url)
datatowrite = filedata.read()
with open('saved_file_name.apk', 'wb') as f:
    f.write(datatowrite)
```

7.5 VirusTotal - Testing Android Applications

The following code uses VirusTotal tools to check if Android applications have malicious behaviors or not.

```
import requests
import os
base_url = 'https://www.virustotal.com/vtapi/v2/file/scan'
parameters = {'apikey': '<apikey>'}
files = {'file': ('AndroidAPKFile.apk', open('myfile.exe', 'rb'))}
response = requests.post(base_url, files=files, params=parameters)
print(response.json())
```