

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**MINING AND DETECTION OF ANDROID MALWARE BASED ON
PERMISSIONS**

ABDIRASHID AHMED SAHAL

**A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING**

GEBZE

2019

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**MINING AND DETECTION OF ANDROID
MALWARE BASED ON PERMISSIONS**

ABDIRASHID AHMED SAHAL
**A THESIS SUBMITTED FOR THE DEGREE OF MASTER OF
SCIENCE**
DEPARTMENT OF COMPUTER ENGINEERING

THESIS SUPERVISOR
PROF. DR. İBRAHİM SOĞUKPINAR
THESIS CO-SUPERVISOR
DR. SHAHID ALAM

GEBZE
2019

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

İZİNLERE DAYALI ANDROID MALWARE'İN
MADENCİLİĞİ VE TESPİTİ

ABDIRASHID AHMED SAHAL
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI
PROF. DR. İBRAHİM SOĞUKPINAR
EŞ DANIŞMANI
Dr. SHAHID ALAM

GEBZE

2019

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 03/07/2019 tarih ve 2019/30 sayılı kararıyla oluşturulan jüri tarafından 23/07/2019 tarihinde tez savunma sınavı yapılan ABDİRASHİD AHMED SAHAL'ın tez çalışması Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE

(TEZ DANIŞMANI) : Prof. Dr. İbrahim SOĞUKPINAR



ÜYE

: Doç. Dr. Mehmet GÖKTÜRK



ÜYE

: Prof. Dr. Erkay Savaş



ONAY

Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun

...../...../..... tarih ve/..... sayılı kararı.

SUMMARY

Due to the open app distribution and more than two billion active users, Android platform continues to serve as low-hanging fruit for malware developers. According to the McAfee threat report, the number of malware families found in the Google Play increased by 30% in 2017. Permission-based access control model is one of the most important mechanisms to protect Android apps against malware.

In this work, we propose a new permission-based model that enhances the efficiency and accuracy of Android malware analysis and detection and has the capability of potentially detecting previously unknown malware. In this new model, we improve the feature selection by introducing a new weighting method, named TF-IDFCF, based on the class frequency (CF) of the feature. The results of our experiments show that our proposed method has a detection rate of greater than 95.3% with a low false positive rate, when tested with different classifiers.

Key Words: Android, Permissions, Malware Analysis and Detection, TF-IDF, Machine Learning.

ÖZET

Açık uygulama dağıtımı ve iki milyardan fazla aktif kullanıcısı nedeniyle, Android platformu kötü amaçlı yazılım geliştiricileri için düşük maliyetli meyve olarak kullanılmaya devam ediyor. McAfee tehdit raporuna göre, Google Play'de bulunan kötü amaçlı yazılım ailelerinin sayısı 2017'de% 30 arttı. İzin tabanlı erişim denetimi modeli, Android uygulamalarını kötü amaçlı yazılımlara karşı korumak için en önemli mekanizmalardan biridir.

Bu çalışmada, Android kötü amaçlı yazılım analizi ve algılamasının etkinliğini ve doğruluğunu artıran ve önceden bilinmeyen kötü amaçlı yazılımları potansiyel olarak tespit etme yeteneğine sahip yeni bir izin tabanlı model öneriyoruz. Bu yeni modelde, özelliğin sınıf frekansına (CF) bağlı olarak TF-IDFCF adlı yeni bir ağırlıklandırma metodu ekleyerek özellik seçimini iyileştiriyoruz. Deneylerimizin sonuçları, önerilen yöntemin, farklı sınıflandırıcılarla test edildiğinde düşük bir pozitif pozitif oranla% 95,3'ten daha yüksek bir tespit oranına sahip olduğunu göstermektedir.

Anahtar Kelimeler: Android, İzinler, Kötü Amaçlı Yazılım Analizi ve Algılama, TF-IDF, Makine Öğrenmesi.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor Prof. Dr. İbrahim SOĞUKPINAR for his valuable advices, guidance and continuous support during this work.

I would also like to express my deep and sincere gratitude to Dr. Shahid ALAM. His support, suggestions and encouragement gave me the drive and will to complete this work.

I also wish to express my thanks to all my friends and laboratory colleagues for their support during this study.

Most importantly I would like to express my deepest thanks to my Mother, Father, brothers and sisters for their encouragement and support during this research and my whole life.

I am grateful to the public of Turkey particularly Ministry of higher Education especially Turkish scholarship team for awarding me scholarship to earn a master's degree of computer engineering.

Above all, I owe it all to Almighty Allah who made all things possible.

TABLE of CONTENTS

	<u>Page</u>
SUMMARY	v
ÖZET	vi
ACKNOWLEDGEMENTS	vii
TABLE of CONTENTS	viii
LIST of ABBREVIATIONS and ACRONYMS	xi
LIST of FIGURES	xii
LIST of TABLES	xiii
1. INTRODUCTION	1
1.1. Motivation	1
1.2. Android Malware Threat	2
1.3. Permissions Based Models	3
1.4. Our Framework	5
1.5. Contribution	6
1.6. Research Limitations	6
1.7. Thesis Structure	6
2. BACKGROUND AND RELATED WORK	8
2.1. Background	8
2.1.1. Android System	8
2.1.1.1. Linux kernel	8
2.1.1.2. Native C/C++ Libraries	9
2.1.1.3. Android Runtime	9
2.1.1.4. Java API Framework	9
2.1.1.5. System apps	9
2.1.1.6. Hardware Abstraction Layer (HAL)	11
2.1.2. Applications	11
2.1.3. Reverse Engineering	12

2.1.4. Android Permission and Security	13
2.1.5. Common Detection Methods	14
2.1.5.1. Anomaly detection	14
2.1.5.2. Misuse detection	15
2.1.6. Malware Analysis	15
2.1.6.1. Static Analysis	15
2.1.6.2. Dynamic Analysis	16
2.1.6.3. Hybrid Analysis	16
2.2. Data Mining	16
2.3. Common Data Mining Methods	17
2.3.1. Association	17
2.3.2. Classification	17
2.3.3. Clustering	18
2.3.4. Prediction	18
2.3.5. Outlier detection	18
2.4. Data Mining Algorithms	18
2.4.1. Decision Tree	19
2.4.2. K-Nearest Neighbors	19
2.4.3. Support Vector Machines	19
2.4.4. Naïve Bayes	20
2.5. Android malware detection steps	20
2.5.1. Preprocessing Apps	20
2.5.2. Representation	20
2.5.3. Dimension Reduction	21
2.5.4. Model Training	21
2.5.5. Testing and Evaluation	21
2.6. Related Work	22
3. SYSTEM DESIGN	25
3.1. Design	25
3.2. Preprocessing Model	25
3.3. Feature selection	27

3.3.1. Traditional TF-IDF	27
3.3.2. Improved TF-IDF-CF	28
3.4. Model Learning and Generation	29
4. EVALUATION, ANALYSIS AND COMPARISON	30
4.1. Dataset	30
4.2. Classification Models	32
4.3. Learning Environment	33
4.4. Classifiers	34
4.5. Evaluation	35
5. CONCLUSIONS	39
5.1. Summary	39
5.2. Contributions and Results	39
5.3. Future Work	40
REFERENCES	41
BIOGRAPHY	44
APPENDICES	45

LIST of ABBREVIATIONS and ACRONYMS

<u>Abbreviations</u> <u>and Acronyms</u>	<u>Explanations</u>
APK	: Android Package Kit
App	: Application
TF – IDF	: Term Frequency – Inverse Document Frequency
<i>CF</i>	: Class Frequency
KNN	: K-Nearest Neighbors
SVM	: Support Vector Machines
API	: Application Program Interface
PCA	: Principle Component Analysis
OS	: Operating System
IG	: Information Gain
GPS	: Global Positioning System
SMS	: Short Message Service
XDA	: Xerox Device Agent
GPL	: General Public License

LIST of FIGURES

<u>Figure No:</u>	<u>Page</u>
1. 1 Number of Available Android Apps (2018)	1
1. 2 Android Malware Samples (2018)	3
2. 1 Android Platform Architecture	10
2. 2 APK File Structure	11
2. 3 Example of Android Manifest.xml File	13
3. 1 Design of the system	26
4.1: Frequency of dangerous permissions (15 shown here) in 500 malwares and 500 benign APKs.	30
4.2: WEKA GUI.	34
4.3: Overall Accuracy Rate for TF-IDF-CF, PCA and IG.	36

LIST of TABLES

<u>Table No:</u>	<u>Page</u>
4.1: Datasets.	31
4.2: Experimental Results.	33
4. 3 Compared Results of TF-IDF-CF, PCA and IG	37
4.4: Datasets with different number of samples.	38
4.5: Experimental results by using Dataset1 with 10-Fold Cross Validation.	38
4.6: Experimental results by using Dataset2 with 10-Fold Cross Validation.	38

1. INTRODUCTION

1.1. Motivation

Android is one of the most popular operating systems for mobile devices. Android has lately become the most purchasing platform on portable devices (The Statistics Portal, 2018). It attracts hundreds of thousands of Android app development vendors and developers because of its openness and ease of use. The monthly active users of Android platform exceeds two billion (Popper, 2017). As CEO Sundar Pichai announced at Google's I/O developer conference 2017 that the company's Android operating system has recently crossed a major milestone. It now has more than 2 billion active monthly devices. Cybercriminals are developing several distinct malicious software projects to target Android software actively and continually. As stated in the McAfee threat study, the discovered malicious software categories in Google Play became greater 30% in 2017 (Abawajy, Huda, Sharmeen, Hassan, and Almogren, 2018).

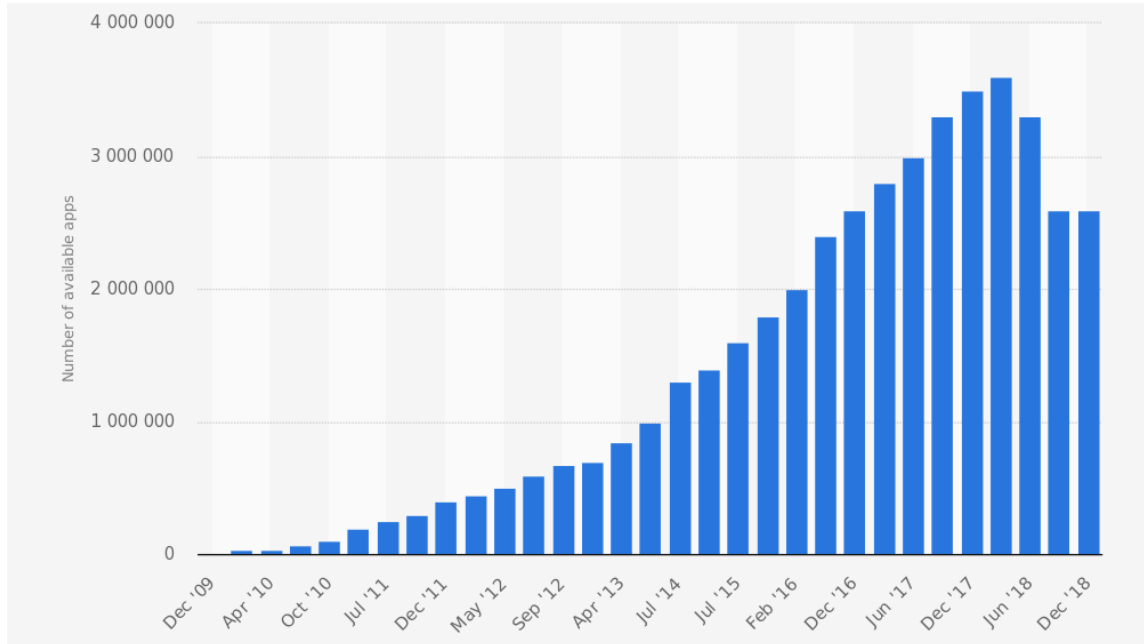


Figure 1. 1: Number of Available Android Apps (2018).

This rapid growth and malicious software variety present a significant obstacle to Mobile apps. Figure 1.1 shows how many applications are available from December 2009

to March 2018 in the Google Play Store. After surpassing 1 million apps in July 2013, in Google Play Store the total of available applications has recently been placed at 2.6 million apps. With most Google Play Store apps available free of charge, the company needs to use efficient business models to secure healthy revenue. Popular gaming apps such as Pokemon Go, and the ever-popular Candy Crush Saga were among the top grossing Android apps in the world as of July 2018. Despite the robust revenue from the gaming app, most gaming apps are free to download and rely on monetization through buying in-game items such as boosters and accessories. Google Play was launched under the name of Android Market in October 2008. As the official app store for Google, it provides its customers with a variety of apps and digital media including music, magazines, books, movies, and TV. Hundreds of millions of apps provide Android users with a variety of functions, including shopping online, social networking, gaming, and map service.

1.2. Android Malware Threat

Android malware is uploaded every day by cyber criminals to either Google's official market or unofficial markets, which puts users at high risk. As the capabilities of such phones increase, more and more complex malware has emerged to target these devices. The popularity of the Android operating system has brought malicious developers to its platform and has led to an alarming increase in the number of malicious software targeting this platform. Until the end of the third quarter of 2018, G DATA analysts counted around 3.2 million new malware apps as shown in figure 1.2. This represents an increase of more than 40% compared to last year's same period (Q3 2017: 2,258,387 samples of malware). Cyber criminals, especially those with Android operating systems, are increasingly focusing on mobile devices. The reason: eight out of ten people around the world use a popular OS smartphone because they are often cheap to buy. This makes the use of a security app more important.

Due to the fast growth of Android Malware, we need to develop sustainable solutions. The smartphone carries many personal privacies such as account information, it becomes a target for malware; due to its openness, the Android system is attacked on all sides. In recent years, Android malicious software has emerged continually and different

attack techniques. Various solutions to defend smartphone users against the growing threat of Android malware have been proposed.

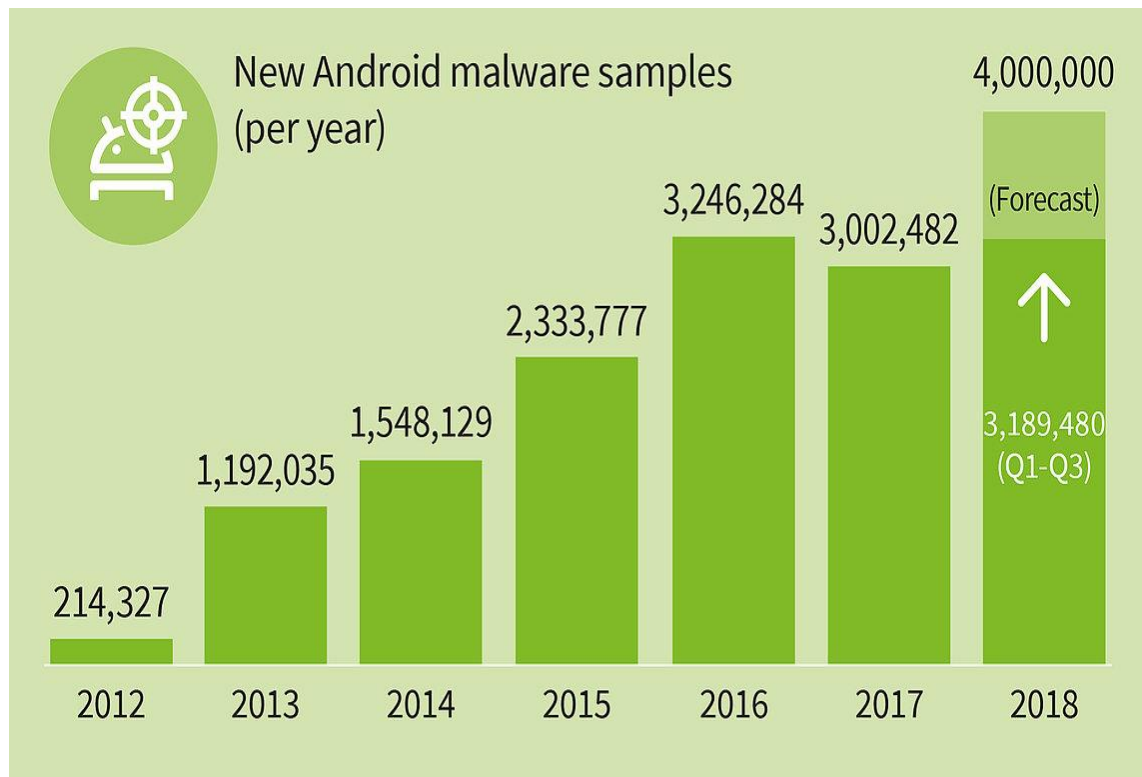


Figure 1. 2: Android Malware Samples (2018).

1.3. Permissions Based Models

One of the main defense mechanisms for Android against malicious apps is the model based on permission, which enforces restrictions on the specific operations that a particular process can perform. The permission system is focused on a lot of researchers. They studied how Android apps use the permissions. A measurement study quantifying the use of 1000 Google Chrome extensions and 956 Android applications was conducted (Felt, Greenwood, and Wagner, 2011). Both platforms used permissions for installation time. They found that while installing nearly every extension and application, Google Chrome and Android users are presented with at least one dangerous request for permission. They also built Stowaway, a tool that detects over-privilege in compiled Android apps and found that many of the apps they investigated were over-privileged.

Android security model relies heavily on mechanism based on permission. There are approximately 130 permissions governing access to various resources (Barrera, Kayacik, Van Oorschot, and Somayaji, 2010). Whenever the user installs a new app, all permissions requested by the application would be approved or rejected promptly. Users, however, have rare knowledge to determine whether permissions may or may not be harmful. To help them make the right choice, they need additional information.

Enck, Ongtang, and McDaniel (2009) propose a security service system, named Kirin, which certificates an application at the moment of installing, using a set of predefined security rules. They analyze the type of malicious behaviors and defined the rules configured with permission and intent information. Peng et al. (2012) and Sarma et al. (2012) both studies propose a risk-scoring method for applications using their own algorithms. compare distribution of permissions between benign application and malware and then determine critical permissions to detect malware. AdDroid system separates privilege from advertising framework to Android platform (Pearce, Felt, Nunez, and Wagner, 2012). It prevents advertising library to access sensitive information allowed for other permissions of the application.

Permission based malware detection model's major weakness is low accuracy. The problem arises from the laxity of Android permission architecture. Application developers can request permissions that are not necessary; the distribution of permissions becomes incorrect. However, using the permissions, access to information from other device subsystems, such as GPS system and database information such as SMS data and contact entries can be granted to an application. So, it is still possible for a malware application to operate in isolation and still perform many different types of attacks and violations, including attacks on resource and data loss. The Android permission policy may seem to be robust, but the problem is that this approach depends on the user deciding whether or not an application's combination of permissions is safe. Many users may use permissions while they do not have the necessary technical knowledge. In this study, we build and train models to classify Android malware applications by using Android permission datasets with numerous machine learning techniques. Malicious users and cybercriminals take full advantage of both the limited smartphone capabilities and the absence of enhanced security mechanisms to design mobile device malware that access sensitive

information, steals mobile phone credit from users or restrict access to certain device functionalities. It is finally concluded that Android still faces several security issues and obstacles in the overwhelming number of malicious software.

1.4. Our Framework

In this thesis, we introduce a scalable static analysis model for Android applications that detects suspicious and potentially malicious apps. Our framework tends to focus on using as little info and low computing resources as possible to generate appropriate precision without installing malicious apps. To achieve this goal, we introduce new feature weighting method that enhances the detection precision by improving the traditional weighting TF-IDF method to TF-IDF-CF that gives higher detection accuracies. In our framework, we follow a general data mining approach to create an Android app classifier. The classifier automatically identifies and make intelligent decisions based on complex malware patterns.

This framework uses static analysis techniques and android permission datasets in the classification process of android apps into malicious software and benign. This framework has the capacity to easily identify previously unidentified malicious software. Introducing a new weighting method strengthens our framework strengthen upon previous static analysis-based studies. TF-IDF-CF maximizes the effectiveness and precision of Android malicious software analysis and detection. The findings in this research show that when tested with separate classifiers such as SVM, J48, Naive Bayes and KNN which are discussed in section 4, our suggested technique gives detection accuracy which is greater than 95.3 percent with a small false positive rate.

The classification and evaluation of Android malicious software involve static or dynamic program analysis. In dynamic analysis, it's difficult to trace the code vulnerabilities, and fixing the problems take longer. Dynamic analysis consumes a lot of resources, often unavailable in smartphone. This thesis attempts to approach the security problem of Android by introducing a new and creative method for statically analyzing the code and measuring correlation to known malicious apps. Static analysis can quickly and efficiently examine all the source code in the application. In other words, static analysis

has the advantage that it is not connected to binary execution. It only depends on the binaries themselves, which are statically examined.

1.5. Contribution

Our contribution of this thesis is an improvement of TF-IDF weighting in vector space model. The TF-IDF method considers both TF and IDF. If only in certain applications a term appears, the TF is high, then this term has a very good ability to differentiate the applications. A feature occurring frequently in the applications within same class represents more characteristics of the class. Therefore, we use total feature occurrence as a new parameter and enhance the TF-IDF to improve the efficiency of our classifier. The results of our experiments show that our proposed method has a detection rate of greater than 95.3% with a low false positive rate, when tested with different classifiers. In short, we explain the process of extracting features from Android APK files to create an Android malware detection framework, we generate a data set from the extracted Android app features and empirically validate our approach and reveal that high precision can be achieved.

1.6. Research Limitations

In this work, our weight assigning method relies on the labeled (into classes) data. Before training, the data needs to be divided into classes, which enables us to calculate the class frequency (c_i) and allows us to use class-based instead of corpus-based TF-IDF. Currently the only classes data is divided into, before training, are malware and benign.

1.7. Thesis Structure

The overall objective of this thesis is to explore a new Android malware detection framework based on the permission feature. We focus on Android permissions to develop an effective Android malware detection system. The remaining sections of the thesis are planned as follows:

In chapter 2, we describe the Android system background, Android apps, reverse engineering, permissions and security, common malware detection and malware analysis methods. We also highlight some background about data mining, the common data mining methods and algorithms, and briefly describe the proposed Android malware detection steps. In addition to the above, we also discuss the related works about permission-based Android malware detection approaches.

Chapter 3 describes the proposed model, gives overview of the system design, the procedure to retrieve and pre-process the raw data, and then introduce feature selection and the learning methods used.

Chapter 4 first discusses the datasets, classification models, system evaluation, compares the approach to multiple detection approaches with different feature selection methods, and presents and analyzes the experimental results.

In chapter 5, we conclude the thesis works and highlight some potential future research directions.

2. BACKGROUND AND RELATED WORK

2.1. Background

In this section, we give some background about the Android system, Android applications, reverse engineering, permissions and security, common malware detection and malware analysis methods. Also, we give some background about data mining, the common data mining methods, data mining algorithms and briefly describe the steps that we carried out to detect Android malware.

2.1.1. Android System

Android is today Google's most widely used mobile operating system. Android OS is built on Linux and some additional open source software and is mainly designed for smart and touchscreen mobile devices. As well as, Google generated different versions of Android operating systems like Android TV, Android Auto and Wear OS, each with the appropriate user interface. Also, some electronics such as the PCs, digital cameras and game consoles use variants of Android OS.

We give a summary of android platform architecture; existing studies (Brahler, 2010; Ehringer, 2010) explain in detail. As illustrated in figure 2.2 the Android OS can be subdivided into: The Linux Kernel, Hardware Abstraction Layer (HAL), Android Runtime, Native C/C++ Libraries, Java API Framework and System Apps.

2.1.1.1. Linux kernel

The Android platform uses a modified Linux kernel version to provide essential services such as hardware drivers, memory and power management, network, etc. Like any other operating system, the Linux kernel is the basis of the Android OS. The Linux kernel acts mainly as an abstraction of hardware in Android platform.

2.1.1.2. Native C/C++ Libraries

Android is designed to run on mobile devices with much reduced powered CPUs and memory. As most essential parts and services of Android platform are built from native code, the native C/C++ libraries are required. The Framework for applications accesses these essential C/C++ libraries through Dalvik Virtual Machine DVM. Java APIs assess these native libraries through Java Native Interface.

2.1.1.3. Android Runtime

Android Runtime (ART) is a middleware component that optimizes and execute efficiently the DEX files on Android devices with less-memory and less-power. Android Runtime includes a set of basic libraries and the virtual Dalvik machine (Dalvik VM or DVM). The libraries are java written implementation of general APIs that provide most of the functionality of Java. The Dalvik VM is responsible for interpreting the Java byte code into the Dalvik byte code.

2.1.1.4. Java API Framework

The Java API Framework layer provides Java written APIs that form building blocks to create applications, by allowing the reusability of system components, core libraries and services. Most components in this layer are just like applications in their implementation but operate or run in background with no user interface. The Java API Framework layer consists of different components such as services, broadcast receivers, activities and content providers that runs in their own Dalvik VM.

2.1.1.5. System apps

The system apps or applications are used to provide interaction between the users and the devices. The system apps provide key tasks such as phone call, browsing, sending or receiving e-mail, etc. These apps give users the capabilities to increase the functionality of their own apps. For example, to process (send/receive) an SMS message a user app will

call the preinstalled system SMS app. We discuss the Android apps in more detail in section 2.1.2.

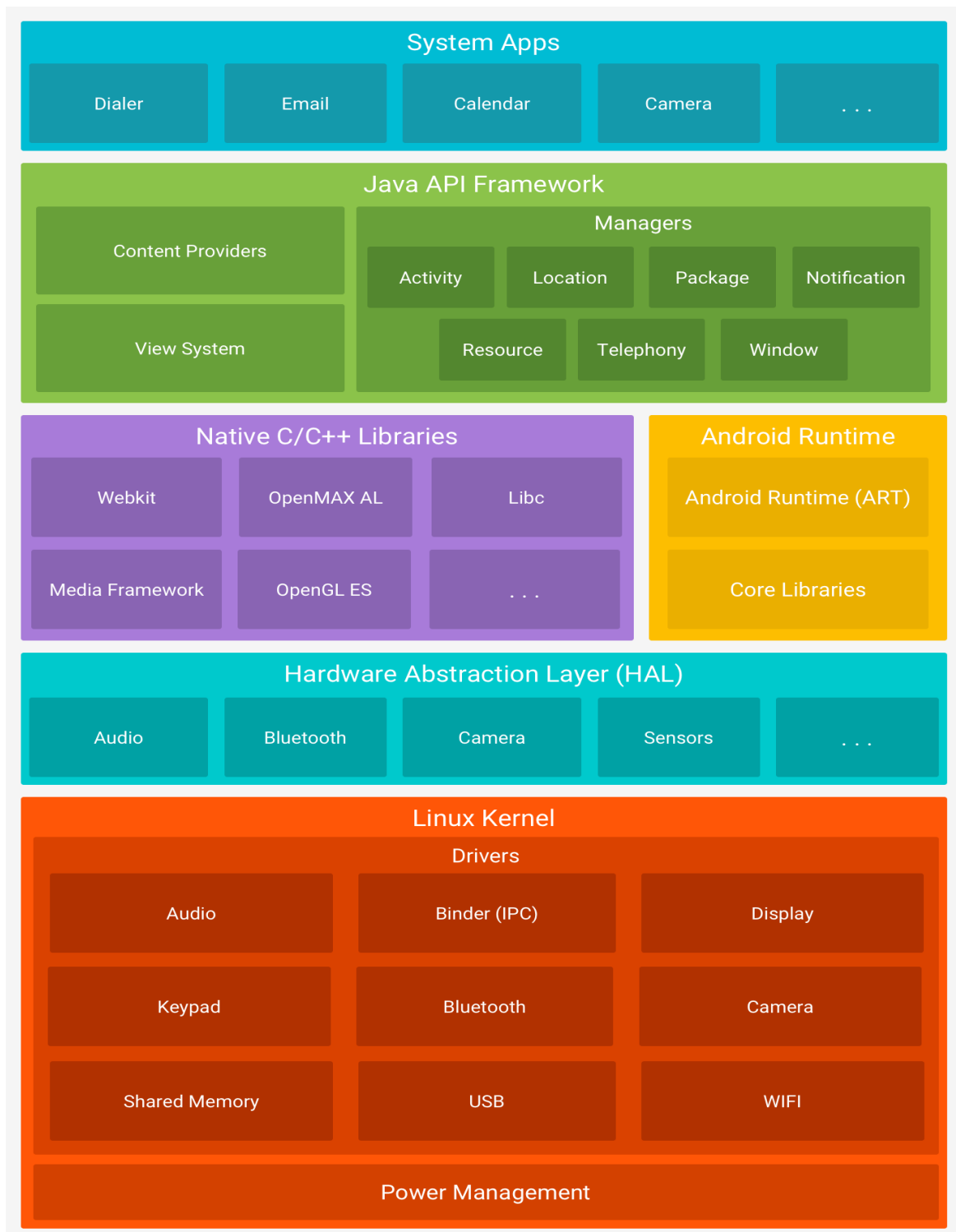


Figure 2.1: Android Platform Architecture.

2.1.1.6. Hardware Abstraction Layer (HAL)

The hardware abstraction layer (HAL) consists of multiple library components that provide standard interfaces with the higher Java API frame for mobile device hardware. Each of these library module implements an interface for a relevant hardware component, such as the camera or the pen.

2.1.2. Applications

The Android platform uses APKs (Android application packages) for mobile applications and middleware distribution and installation. An APK file contains the code of the program (such as .dex files), manifest file features, certificates, resources and assets. An APK file can be provided with any name that ends with *.apk*.

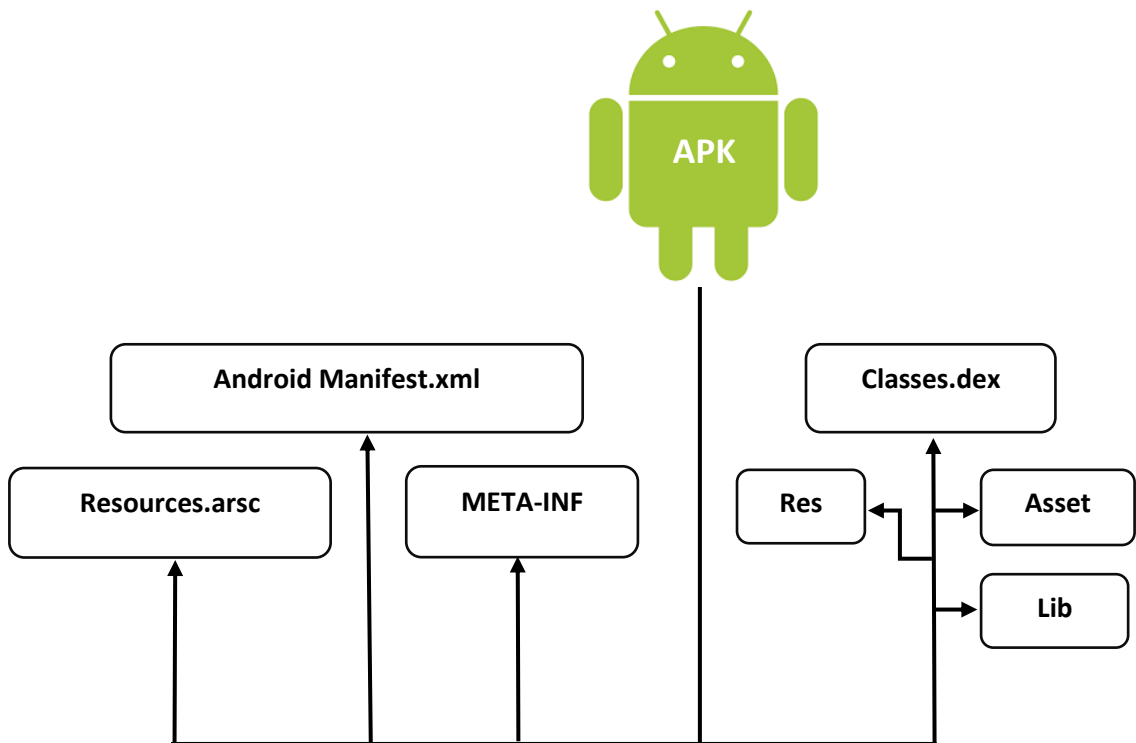


Figure 2.2: APK File Structure.

The APKs are container files or archives that normally contains *Classes.dex*, *Asset*, *Res*, *Lib*, *META-INF*, *Resources.arsc* and *Android Mainfest.xml* file.

- `Classes.dex`: the dex format compiled application's code.
- `AndroidManifest.xml`: the manifest file in binary XML format. Example of `AndroidManifest.xml` is illustrated in figure 2.4.
- `META-INF`: this folder contains a `MANIFEST.MF` File stored in the `MANIFEST.MF` file with meta content data, application certificate and resource list and SHA-1 digest of the corresponding lines.
- `Resources.arsc`: this file contains precompiled resources of the application, in binary XML.
- `Lib`: this is an optional folder that contains compiled code - i.e. Native code libraries.
- `Res`: this folder contains not compiled resources into `resources.arsc`.

2.1.3. Reverse Engineering

Reverse engineering, also named back engineering, is the mechanism of disassembling the man-made object and examination of its construction. Reverse engineering can be seen as the process of identifying the software's composition and analyze in detail to exploit its weaknesses. There are lots of tools, such as APKTool, dex2jar, diStorm3, edb-debugger, Jad Debugger and Javasnop to reverse engineer or extract important features from an APK file.

We use APKTool to decompile the APK files during the preprocessing model. APKTool is one of the most popular reverse engineering tools, it is a powerful tool for decompiling Android apps that developed by XDA Senior Member ibotpeaches (APKTool, 2019). According the other reverse engineering tools APKTool has ease of use advantage. APKTool enables reverse engineering APK files, allowing to decode the APK files to their original form, so it can be easily modified and recompiled. In this thesis, to extract permissions from an APK file, we use APKTool to reverse engineer the APK file. It allows us to extract the source code near to the original form of the app with the name of the decompiled APK. It extracts the android manifest XML file which contains vital information about the app including the permissions. An example manifest file is

illustrated in figure 2.4. It lists the system permissions that the application requires in order for the app to operate correctly.

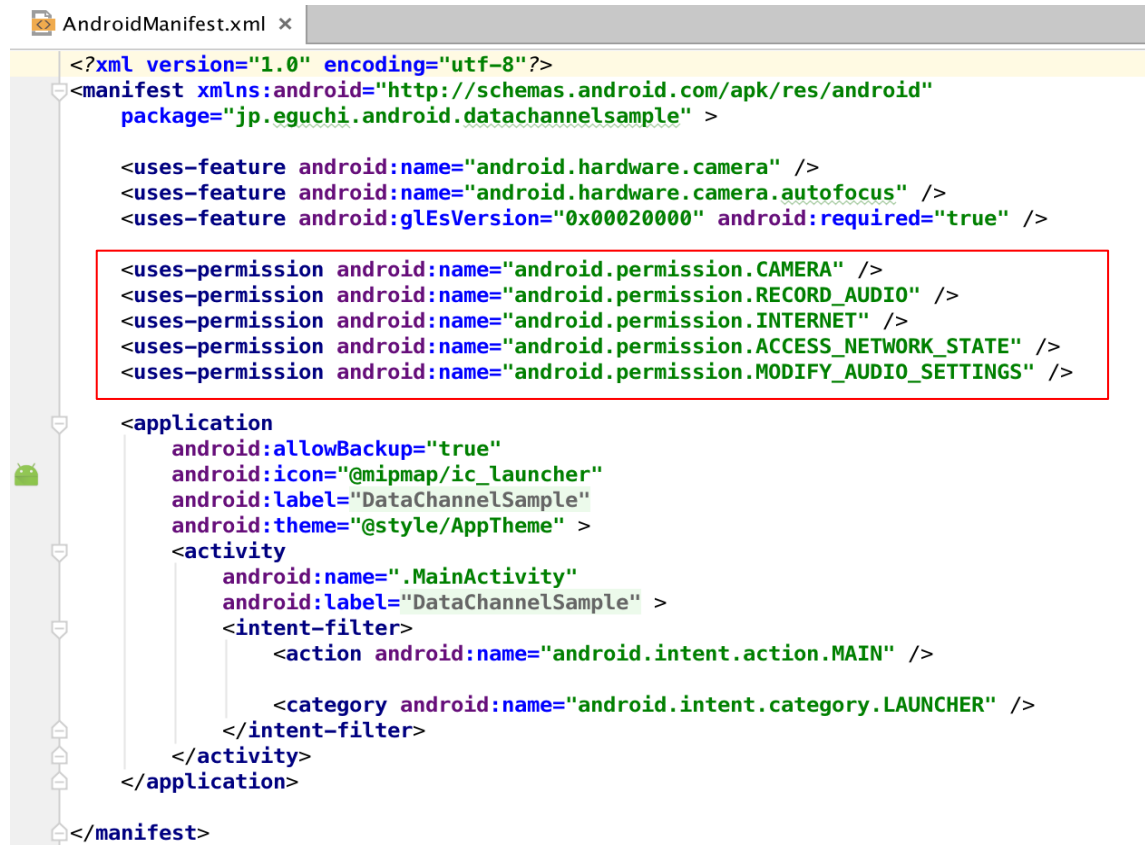


Figure 2.3: Example of Android Manifest.xml File.

2.1.4. Android Permission and Security

Permission is a special privilege that allows applications to access sensitive media or hardware features in a smartphone. Certain permissions (such as, Camera and Location, etc.) should be given with greater care. The permissions required by an Android application is declared in its Android manifest file, and it can also define some additional permissions to use for restricting some components. During the runtime an Android application asked from the user to allow (permit) certain actions. The importance of the permissions in the Android operating systems is to keep the privacy of the user. Permission is required to access sensitive user data, such as contacts and Android SMS apps. In addition to certain system features, such as camera and internet. Depending on

the feature, the Android OS might grant permissions by default or might ask the user to approve the request. For security and privacy purposes, during installation Android System gives each application a unique user and group ID. In this way the data for each application remains private, and the other applications have no access to this data. The only exception is where a user gives permissions to other applications to access this data, such as contacts, media files etc. Below we discuss some important permissions that can be used to perform malicious actions:

- **WRITE_SMS, SEND_SMS and READ_SMS:** These permissions allow an application to gain access to harmful API calls, and hence allowing the application to read, write, and send SMS (such as contacts, banking information etc.) Without user notification.
- **CALL_PHONE:** This permission allows an application to make calls on behalf of a user without confirming from the user, and hence is dangerous to the privacy of a user.
- **READ_HISTORY_BOOKMARKS:** This permission allows an application to read a user's browsing history and bookmarks, and hence is a risk to the privacy of a user.
- **ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION:** These permissions allow access to the coarse (e.g. cell ID, Wi-Fi) and fine (e.g. GPS) for Android apps. locations (sarma2012android) of a smartphone, and hence pose a risk to the privacy of a user.

2.1.5. Common Detection Methods

The most well-known detection methods of android malware are misuse and anomaly detection. Misuse is limited and restricts to the detection of specific malicious attacks, but Anomaly is adaptive and can detect even early attacks. In Misuse detection method anything not known is normal, while anything known is abnormal in anomaly detection method.

2.1.5.1. Anomaly detection

Anomaly detection is a method of behavioral attack detection that refers to the detection of patterns in each dataset that do not conform to normal behavior. it detects the

malware by learning familiar malware behavior using machine learning algorithms. Anomaly detection method also has the advantage of identifying unfamiliar malware. A high false prediction may be problem for this method.

2.1.5.2. Misuse detection

Misuse detection is a signature-based detection method that attempts to detect insiders' misuse and gives high accuracy detection results for certain well-known attacks. Misuse can accurately identify malware if matching any of signatures according to sets of rules and policies, and it is also the primary method for most of the anti-virus Apps. Therefore, signature-based detectors can only provide very limited zero-day protection, which means it is almost unable to detect malware unless this file has already been captured with the signature generated and stored in its database. The advantage of misuse detection method is ease of adding familiar malware attacks to the model and its disadvantage is inability of detecting unfamiliar malware attacks.

2.1.6. Malware Analysis

Malware is any application designed to damage the host android operating system or to rob the user's sensitive data. The process of assessing the potential impact, origin and functionality of a malicious software sample is called malware analysis. Malware may contain software which collects user information without consent. The method used to analyze malware usually falls within one of the following types:

2.1.6.1. Static Analysis

Static analysis is one of the most common methods for analyzing Android malware. Static malware analysis generally uses the source code of the program to achieve the goals. In static analysis, the features are easily extracted from the source code without running the app. (Szalay and Chandraiah, 2014). Static features such as permissions are extracted easily from android APK contents. The static malware analysis has the advantage of low cost and accurate performance. This technique is relatively easy and quick. It uses multiple

tools to determine whether the file is malware or benign, giving details about how it works and gather technical details to generate signatures.

2.1.6.2. Dynamic Analysis

Dynamic analysis examines the behavior of the application during its runtime instead of examining the source code. Dynamic analysis executes the doubtful application using an environment called sandbox. Although the usage of dynamic analysis gives wide-ranging view, it is still difficult to trace the vulnerabilities, and fixing the problems take longer. Dynamic analysis has the disadvantage of consuming a lot of resources, often unavailable in smartphone.

2.1.6.3. Hybrid Analysis

Hybrid Analysis techniques can improve reliability, code coverage, monitoring and detecting vulnerabilities of Android apps through a mixture of static and dynamic analysis. Hybrid analysis is a method capable of integrating runtime data extracted from dynamic analysis into a static analysis algorithm to detect behavior or malicious application functionality.

2.2. Data Mining

Data mining extracts and identifies potentially important and fundamentally understandable patterns from a set of data. Different tools and algorithms are then used to analyze these data patterns (Hand, 2007). Data mining in computer science combines artificial intelligence and statistical tools (such as machine learning) with database management and use it to explore useful and informative patterns and relationships. The process of data mining includes several major steps, from comprehending the objectives of a project and what data is collected to implementing process changes based on the final analysis. The primary steps are model learning, model assessment and model use. Logically, you might think that data mining refers to new data extraction, but it does not. Instead, data mining is an examination of the data you have obtained to generate new information.

2.3. Common Data Mining Methods

Recently, several important data mining methods have been created and used. Classification, clustering, prediction, association, decision tree, sequential pattern and regression analysis are the most commonly used techniques. (Joseph and Kesavaraj, n.d.). Here we briefly describe some of the most common data mining methods and how they work.

2.3.1. Association

Association is one of the best-known methods for data mining and it is used to analyze the things that tend to happen in pairs or larger groups together. The association is referred to tracking patterns but is much more specific to variables related dependently. In Association, a pattern is discovered based on a relationship between specific events or attributes that are closely related to each other. Therefore, the association method is also known as the relation method. Retailers use the association technique to investigate the purchasing habits of customers. The retailer finds things that customers mostly buy together, such as buying beers with snacks from previous sales data, and combines them to save time and increase customer sales.

2.3.2. Classification

Classification is defined as the assignment process for items in one of several pre-determined classes or groups in a dataset. It is a traditional machine learning based data mining method. Classification covers things that are already labelled. This labelled data is called the training data, i.e., when information exists and can be trained or classified with an algorithm easily. Spam filters, for example, identify differences between content in valid messages and spam messages. This can be achieved by identifying large sets of e- mails as spam. Classification is a more complex method for data mining that enables us to collect different attributes into distinguishable categories, which we can then use to draw further conclusions or functions.

2.3.3. Clustering

Clustering is a data mining technique which automates the grouping of meaningful and useful object clusters with similar characteristics. In short, clustering is the process through which a set of objects is divided into homogeneous groups or groups. This method is very similar to classification, but it includes grouping data pieces according to their similarities. The clustering method describes, defines, and places the objects in each class, while objects are assigned to predetermined classes in the classification methods. Cluster detection is called the recognition of different groups or subcategories in the data.

2.3.4. Prediction

Prediction is a supervised learning method used to predict data values that are missing or not available. Regression analysis is the commonly used numerical prediction methods. In many cases, it is enough to recognize and understand historical trends to determine a rather accurate forecast of what will happen in the future. For example, you could review the credit history of consumers and past purchases to predict whether they will pose a future credit risk.

2.3.5. Outlier detection

Outlier or anomaly detection in data mining can be used to determine if something differs significantly from the regular pattern. The outlier differs greatly from other observations, the suspicions created by several methods (Hawkins, 1980). The detection of anomalies (also outlier detection) in data mining is the identification of rare items, events or observations that cause suspicions by differing considerably from most data. Anomalies are also called outliers, novelties, noises, discrepancies and exceptions.

2.4. Data Mining Algorithms

Data mining is generally known as an interdisciplinary computer science field and is essentially a computer pattern process in large data sets. It is considered a key process for the extraction of data patterns by intelligent methods. Algorithms for data mining contain

a set of calculations used to create a model from large data sets. First, the algorithms analyze the data input and then generate a model. Here we focus on top and most common data mining algorithms such as Decision Tree, K-NN, SVM and Naive Bayes.

2.4.1. Decision Tree

We used the J48 Ross Quinlan's implementation of C4.5 Decision Tree-based learning algorithm. J48 decision tree is an important type of algorithm that builds classification model in the form of a tree. J48 is a machine learning model based on the prediction that creates a tree which is based on the attribute values of the training data for classifying a new item (Bhargava, Sharma, Bhargava, and Mathuria, 2013).

2.4.2. K-Nearest Neighbors

The K-Nearest Neighbors (KNN) is a very effective and simple classification algorithm which is fundamentally different from the other learning algorithms. KNN is mostly referred as a lazy learner because of memorizing from the training dataset (Zhang and Zhou, 2007). KNN is a simple classification algorithm. It is used both for classification and regression but primarily for classification purposes. Each tuple represents a point in a n-dimensional array in this method. A tuple is an ordered set of values which is often separated by commas according to the language rules for each value. This algorithm compares the similarities between the tuples of training and the tuples given. KNN is a lazy learner algorithm that works based on a minimum distance to the training samples from the query instance to determine the nearest K neighbors. If $K=1$, the case is simply assigned to the class of its closest neighbor.

2.4.3. Support Vector Machines

Support Vector Machines (SVM) is data mining algorithm, which is widely used for classification. In data mining and machine learning SVMs are one of the most popular algorithms. SVMs can efficiently perform linear and non-linear classifications (Smola and Schölkopf, 2004). Many prefer the support vector machine because it gives significant precision with less computational power.

2.4.4. Naïve Bayes

Naive Bayes is one of the supervised classification methods and a powerful algorithm for predictive modeling. Naive Bayes classification techniques are probabilistic algorithms based on Bayes theorem, which assumes independence between predictors. Naive Bayes is an easy model to build and works well in large datasets. It observes the whole attributes separately and makes their usage in the data simpler (Patil and Sherekar, 2013). Naive Bayes classifier uses the distribution of probabilities to classify the given input over a class set.

2.5. Android malware detection steps

The permission system is one of the most important security mechanisms in Android. We developed a novel permission-based malware detecting scheme using data mining and machine learning technologies based on the features extracted. In general, our proposed malware detection system involves five main steps: *preprocessing apps*, *representation*, *dimension reduction*, *model training*, *testing* and *evaluation*. Here we give a brief description about each of these steps:

2.5.1. Preprocessing Apps

This step is about decompressing every Android application package file, getting AndroidManifest.xml from the extracted content, and then decompiling it. The APK file is a compressed file, and we use the APKTool to decompress it. After decompressing the AndroidManifest.xml file, we get the permission list of each APK.

2.5.2. Representation

Before classifying, we need to convert the APK files to a format that can be recognized by the computer. The vector space model (VSM) is the most common approach. With the extracted requested permissions, we convert the permissions into a Boolean vector. Suppose the 385 permissions we selected are labeled from 1 to 385 (F1, F2, · · · , F385), an application that requires F2, F4 and F5 would be represented as a

vector of Boolean value (0, 1, 0, 1, 1,..., 0). In addition to the above features, an *App- Type* label is attached at the end of a feature vector to determine whether the vector is a benign or malicious application. The App-Type value contains only Benign and Malware. If an APK file comes from a benign dataset, it is labeled "*Benign*". If an APK file is from malware, it is labeled "*Malware*".

2.5.3. Dimension Reduction

In Android, there are hundreds of permission features, by selecting all Android permissions as features of same weights, classification becomes difficult. The most commonly used selection methods are information gain, document frequency, principal components analysis PCA and mutual information. In this step, the original feature set from the previous steps is reduced, by applying the improved TF-IDF-CF, to give more weights to the most representative features for each class.

2.5.4. Model Training

This is the main step of the detection system. It includes selecting some apps from corpus to include in the training set, learning on the training set, and then generating the model. During the training phase, the classifier receives a training set consisting of malware sample vectors and benign software samples. We use machine learning methods to automatically learn the separation between malicious and benign applications. The use of machine learning prevents us from manually establishing detection rules for the extracted functions. While several methods of learning can be used to separate two classes, we used J48, Naive Bayes, SVM and KNN.

2.5.5. Testing and Evaluation

In this step, the generated models in the previous step are used to classify the apps in the test set. The standard metrics used to evaluate the performance of the system are as follows: TP (*True Positive*): Correctly detected number of malware apps. FP (*False Positive*): Number of benign applications wrongly identified as malware. TN (*True*

Negative): Correctly detected number of benign apps. FN (*False Negative*): Number of malware applications wrongly identified as benign.

2.6. Related Work

Numerous researches have been performed on Android malware classification, using permission related features. In this section we highlight some of the related works of permission-based Android malware detection.

Liu and Liu (2014) proposed a framework that considers both requested and used permissions in the Android applications. This framework is two layered malware detections and uses machine learning techniques to get high detection accuracy with the potential of detecting Android malware applications based on permissions. The first layer involves two steps, in step one extracts the requested permissions and uses j48 classifier and in step two uses requested permission pairs and j48 Classifier to identify the malicious apps. The second layer involves the third step which is used to detect the apps that cannot be detected by the first layer. This work concludes a permission-based mechanism for detecting malware that can be used in the first step as a filter to identify malware apps.

Rovelli and Vigfusson (2014) proposed a simple, client-server architecture malware detection system based on permissions which views requested permissions as behavioral markers. PMD is a client-server architecture permission-based Android malware detection method. The system has a permission checker part in each server-side and client-side. The client-side part extracts the permissions from the Android apps and forwards the extracted permissions to the server-side part. The server-side permission checker is responsible for classifying apps as benign or malicious.

Aung and Zaw (2013) proposed machine learning based framework for malicious apps detection and security enhancement of Android based mobile device users. This system uses machine learning classifiers to monitor various permission features extracted from Android apps and labels the application as benign or malicious. The system uses information gain as feature selection method. They extracted more permissions from different downloaded applications from Android markets to generate the model.

Xiaoyan, Juan, and Xiujuan (2014) proposed a malware detection system that analyzes the application activity of the mobile phone. This approach firstly decompresses

all android apk files and the end extracts apk permission list from AndroidManifest.xml file in the decompressed apk content folder. This work uses PCA algorithm as feature selection method and then applies support vector machine method to train and build classifier for malware detection. With the contributions of the android based mobile device users, this framework can classify the android apps as benign and malicious.

Arp et al. (2014) proposed a lightweight Android malware detection approach that enables detecting malware applications in the smartphones directly. The approach performs broad static analysis by collecting all possible features of Android APK. It embeds in a joint vector space the whole features and automatically detects the malicious apps. DREBIN examines the application in a static way and extracts the features from the given application. The extracted features analyzed geometrically using joint vector space. This approach detects malicious apps using machine learning algorithms such as a linear support vector. The approach gives the advantage of time efficiency in analyzing unknown apps. The approach can be used on the mobile phone directly and enables protection of installation from untrusted sources.

Aafer, Du, and Yin (2013) have performed a study thorough malware analysis for extracting malware relevant features and behavior captured at API level, they trained multiple robust and lightweight machine learning classifiers using the generated dataset from the extracted feature set. In their work, they built classifiers using a basic data mining method with machine learning algorithms. The classifier potentially identifies the malicious apps and predicts the maliciousness of the application in smart way.

Moser, Kruegel, and Kirda (2007) present a binary obfuscation scheme that relies on the idea of a primitive constant loading into a register. An analysis tool cannot be used to determine the value of the analysis tool so that program control flow, access to local or world variables and the tracking of values contained in processor registers cannot be obscured. Using their proposed sham approach, they demonstrate that advanced semantics-based malware detectors cannot be avoided, and the techniques of static analysis are not the only way to find malware; therefore, dynamic analysis must complement them.

In this section we have discussed several researches of analyzing and detecting android malware based on permissions. After analyzing both threats and solutions

discussed and proposed in these researches, we saw the need to develop more efficient permission-based systems. We have designed a new system that increases the efficiency and accuracy of Android malware detection and the ability to identify previously unknown malicious apps. Our research differs from the previous works by introducing a new method of assigning weights to each feature extracted from the APK. This new method reinforces the traditional TF-IDF technique to increase the efficiency of the system. According to the results obtained from the performance tests carried out in this thesis, our proposed method has higher accuracy rate than the other such techniques. We achieved a detection rate higher than 95.3% using different classifier algorithms.

3. SYSTEM DESIGN

The ultimate goal of our study is to comprehend if a group of permissions required by an application corresponds to its ultimate behavior as either benign or malicious. If a correlation exists, how it can be used to automatically detect potentially harmful behavior of previously unknown applications? In this Section, we give an overview of the design of our permission-based detection system, the preprocessing model, feature selection and the learning methods used.

3.1. Design

An Android malware detection system can be built in a variety of ways. In this section, we describe design decisions and the implementation of a framework detection architecture that exploits a classifier based on application permissions. As illustrated in figure 3.1 Our system is made up of three major parts: The Pre-processing model, Feature selection and Model learning and generation. This framework takes an APK, decompile it using the APKTool, and parse the manifest file to extract the permission features. It then generates the feature sets and the datasets by applying the proposed feature selection method. Then these feature sets are used to build (using WEKA) machine learning classifiers to identify benign and malicious apps. Weka is a collection of algorithms for the learning of data mining. The algorithms can be applied to a data set directly or called from your own Java code (Thornton, Hutter, Hoos, and Leyton-Brown, 2013). Weka includes data pre- processing, classification, regression, clustering, rules of association and visualization tools.

3.2. Preprocessing Model

In this section, we describe the process used to obtain data from the Android apps. Firstly, we collect the malware and benign APK samples and then we decompile the Android APK sample files using the APKTool to get the resources. The primary task of this module is to decompile samples of the software and to create initial features. Since APK (Android application package) is essentially a zip format file, APK should first be

unzipped using APKTool and then decompiled. After decompression, the overall configuration file for AndroidManifest.xml is viewed in the extracted content. AndroidManifest.xml contains the application name of the Android application, a Linux user identity, permissions, minimum API version needed for this application. We extract the permissions requested by the Android applications from the AndroidManifest.xml file. After extracting the permissions, we identify the unique permissions in malware and benign, then we build a binary matrix storing (0,1) binary values for the features. We apply the improved TF-IDF-CF weighting method on this binary matrix for feature selection to build our datasets.

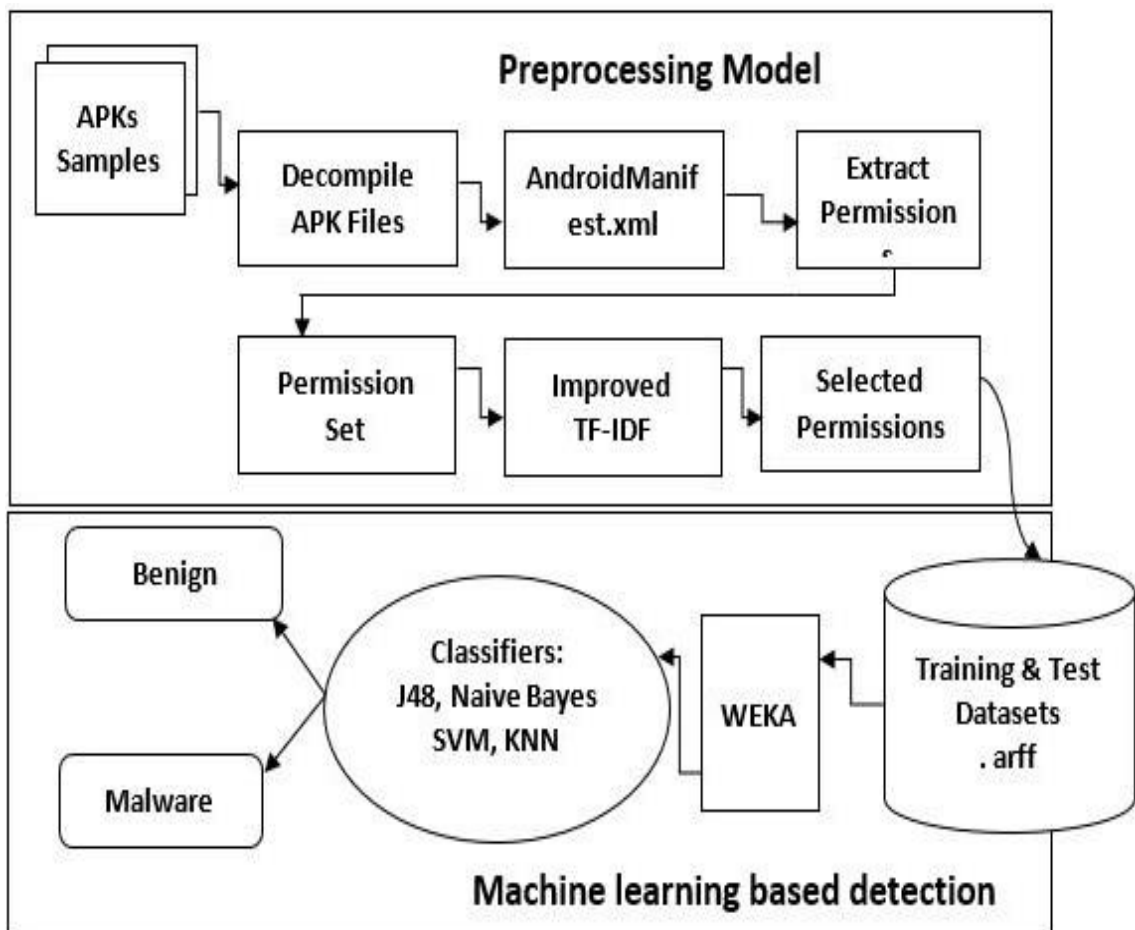


Figure 3.1: Design of the system.

3.3. Feature selection

Each Android app must contain a manifest file called `AndroidManifest.xml` that provides data supporting the application's installation and later execution. The Malware tends to ask for certain permissions much more than benign applications. For example, a large percentage of present malware wants to send premium SMS messages and requests SEND SMS permission. Therefore, all permissions in the Manifesto are collected in a set of features. Android permissions are stored in this file and can be extracted easily and efficiently. The permission system is among the most significant security frameworks proposed in Android. The user actively grants permissions during installation of an app and allows access to security-related resources for the app.

If an app is detected by the system, the calculation quantity of the classifier increases if the selected feature dimension is high. Since the power of computing, storage space, power and other resources of smartphones are limited, this kind of defect is more prominent. In this stage, the original permission set from the Pre-processing model will be processed by the improved TF-IDF-CF method to extract the significant permissions. The TF-IDF weighting method in vector space model is often used in information retrieval and categorization, it gives more weight only to the term that appears rarely in an application.

3.3.1. Traditional TF-IDF

The TF-IDF is a scoring measure commonly used for information retrieval (IR) or summary, which is the term frequency—the inverse document frequency. The purpose of TF-IDF is to reflect the relevance of a term for a particular record. In other words, TF-IDF is a technique of information collection that weighs the frequency of a term (TF) as well as its inverse document frequency (IDF). For each word or term, the product of its TF and IDF scores is referred as the TF-IDF score of the term. Simply put, the higher the TF-IDF weight, the less frequently used, and vice versa.

$$W_{ij} = tf_{ij} * \log \left(\frac{N}{n_i} \right) \quad (3.3.1.1)$$

In Equation (3.1), tf_{ij} represents the raw frequency of the term i in j , N is the number of the total applications in the corpus (corpus means the complete dataset), and n_i is the number of applications that feature i appears in. We know some features can have different weights in different classes, so we improved the TF-IDF method to represent more class's characteristics, by considering the feature frequently appearing in the applications within the same class.

3.3.2. Improved TF-IDF-CF

As far as TF- IDF weaknesses are concerned, we introduce a new parameter to show the in-class characteristics and we refer to this class frequency that calculates the term frequency of one class apps.

$$W_{ij} = \text{tf}_{ij} * \log\left(\frac{n}{n_i}\right) * c_i \quad (3.3.2.1)$$

In equation (3.2), we use a class-based TF-IDF, instead of corpus-based method, tf_{ij} is the raw frequency of the feature i in j App, n is the total number of applications in the class, and n_i is the number of applications in the class that feature i appears in. To give priority to the feature that appears more frequently in the same class, we introduced another term c_i which is the class frequency of feature i and hence this new equation/model also includes class's characteristics.

In this stage, we collect all possible features extracted from the Android APK files. We use the new TF-IDF based method, named TF-IDF-CF shown in Equation (3.2), to select best features and give them weights to enhance the detection accuracy of our proposed framework, and to make the learning easy. For each Android application, we compare the permission list obtained from the corresponding application package with the total set of permissions of the Android system and store the selected features as a binary number (0 or 1) represented as a sequence of separate commas. Following is an example of the benign and malware feature vector extracted from an Android APK file:

1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

(a) Benign

```

1 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0

```

(b) Malware

3.4. Model Learning and Generation

In this stage, to detect automatically potential malicious applications. We trained multiple classifiers using different machine learning algorithms. We use four most commonly used machine learning algorithms including: J48 decision tree, Naive Bayes, SVM and KNN. The main objective of our study is to build a predictive model by integrating multiple models, thereby improving the performance of each model. We generate first a group of structured data to implement machine learning techniques. The entire process of learning a model from data collected is literally called "learning" or "training" and is carried out using an algorithm for learning. During the training phase, the classifiers receive a training set consisting of the permission vectors of malware and benign samples. Then the trained classifiers are used classify the input permission vectors of an unknown APK during the detection phase. In short, this system has high detection rates for unknown malware through the sample library. In practice, it is equivalent to the system detecting unknown good software in real time without updating the sample library.

4. EVALUATION, ANALYSIS AND COMPARISON

In this Chapter, we describe the dataset, classification models used for evaluating the system, the learning environment and the classifiers used. We also discuss the experimental results for the system and compare the framework to multiple detection frameworks that uses different feature selection methods such as principal component analysis (PCA) and information gain (IG).

4.1. Dataset

Each Android system for analysis, detection and classification should be evaluated on a data set of benign and/or malicious Android app samples. In this Section, we describe the data sets we used in our study and explore the statistical property of the requested permissions. However, based on the rapidly evolving nature of Android malware, it is important to update the dataset with newer samples in order to efficiently test systems.

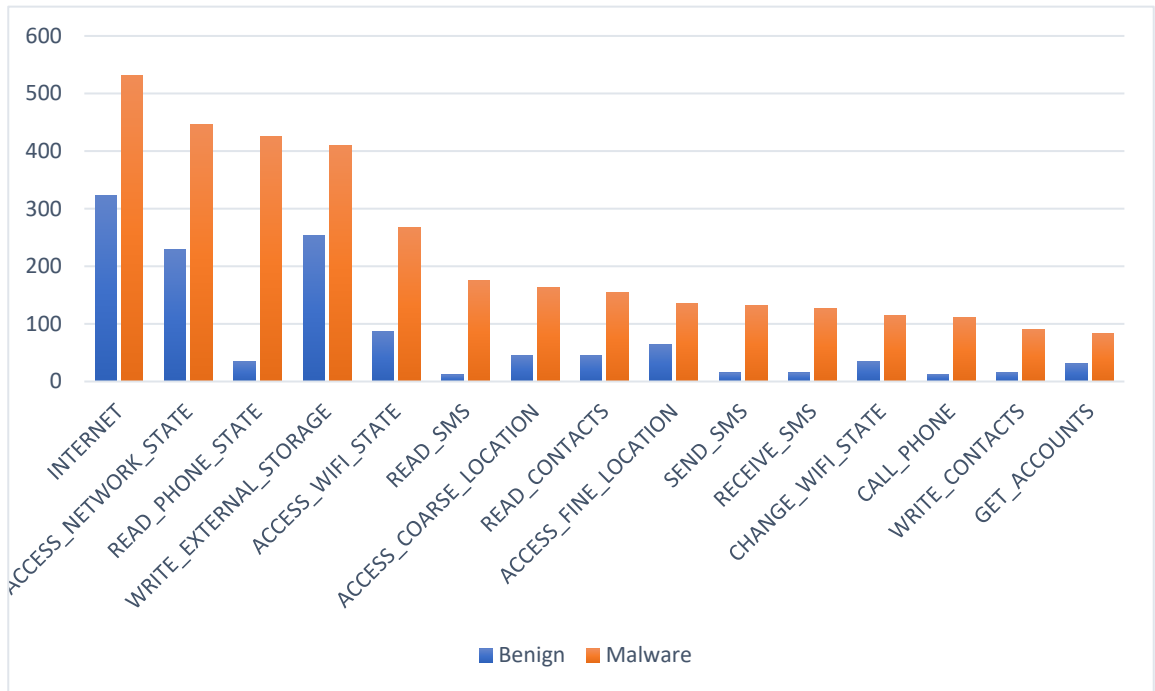


Figure 4.1: Frequency of dangerous permissions (15 shown here) in 500 malwares and 500 benign APKs.

We have 1000 APK samples (500 malware and 500 benign) collected from different sources. Figure 1 shows the most frequently requested dangerous permissions from Android apps in our dataset, ordered by decreasing popularity. The blue bars show the frequency of 15 specific permissions in benign apps, while the orange bars show the frequency of the same permissions in malicious apps. The blue bars show the number of times the specific permissions have been requested by benign apps, while the orange bars show the number of times they have been requested by malicious apps. The INTERNET privilege was the most requested permission for both benign and malicious applications. Over the years, we have witnessed several new permissions being requested. For example, in 2011, three malwares first used the dangerous permission MOUNT FORMAT FILESYSTEMS (i.e. used to format an external memory card). Other permissions that became popular due to malware include USE CREDENTIALS and AUTHENTICATE ACCOUNTS. Google classified them as dangerous as they could greatly assist in privilege escalation. In this thesis training data has been collected from different places, the benign samples are collected from the Google Play Store (Google Play Store, 2018) and the sample of malicious apps are taken from Contagio Mobile (Genome Project,” n.d.) and Android Malware Genome Project (Zhou and Jiang, 2012). These apps are different in functions, types, source and type. Some are business, games, and social applications and so on. We decompiled, extracted the features and, created training and testing datasets of 700 and 300 samples respectively. Total 385 features were selected to enhance the accuracy.

Table 4.1: Datasets.

	Samples	Malware	Benign	Attributes
Training	700	350	350	385
Testing	300	150	150	385
Total	1000	500	500	385

4.2. Classification Models

In this Section, we evaluated using Weka tool the classification capability of our framework for Android APK files as benign or malware on two datasets, a training set of 700 and a testing set of 300 samples. This framework gets the original feature set by extracting permissions from Android APK files using reverse engineering tool and forms the feature set by using the newly proposed method in this thesis, named TF-IDF-CF.

We used the techniques of machine learning to classify the Android applications. We used multiple learning algorithms such as J48, Naïve Bayes, SVM and KNN to test our model. The metrics we used are as follows:

- TP (True Positive): Correctly detected number of malware apps. In other words, the cases we forecast YES, and the actual output were YES as well.
- FP (False Positive): Number of benign applications wrongly identified as malware. In other words, the cases in which we forecast NO and the current output were NO.
- TN (True Negative): Correctly detected number of benign apps. In other words, the cases we forecast YES, and actual output were NO.
- FN (False Negative): Number of malware applications wrongly identified as benign. In other words, the cases of NO and the actual output were YES.

We use the following standard evaluation measures to assess the results of the experiments carried out from our proposed permission-based detection framework.

- True Positive Rate (TPR): Number of samples correctly classified as malware out of the total malware dataset.

$$\text{True Positive Rate (TPR)} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \quad (4.1)$$

- False Positive Rate (FPR): Number of samples wrongly classified as malware out of the total benign dataset

$$\text{False Positive Rate (FPR)} = \frac{\text{False Positive}}{\text{True Negative} + \text{False Positive}} \quad (4.2)$$

- Overall Accuracy (ACC): Proportion of Android apps, that are correctly identified as either malicious or benign app.

$$\begin{aligned} &\text{Overall Accuracy (ACC)} \\ &= \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}} \end{aligned} \quad (4.3)$$

Table 4.2: Experimental Results.

Dataset Type	Classifier Method	True Positive %	False Positive %	Overall Accuracy %
Training	J48 Decision Tree	0.99	0.01	99.14
	Naive Bayes	0.98	0.02	98
	Support vector machines	1	0	100
	k-Nearest Neighbors	1	0	100
Testing	J48 Decision Tree	0.97	0.03	97
	Naive Bayes	0.97	0.03	97
	Support vector machines	0.98	0.02	98.33
	k-Nearest Neighbors	0.95	0.05	95.33

4.3. Learning Environment

During this phase, the learning data set is trained in an adequate learning environment for classification. In the context of this work, we use WEKA which is a collection of machine learning algorithms for data mining. WEKA is a suite of Java- based machine

learning free software developed at Waikato University, New Zealand. Software issued under the General Public License (GPL).

WEKA implements algorithms for data preprocessing, classification, regression, clustering and association rules that you can directly apply to a dataset or call your Java code. WEKA uses an ARFF file format data file, the file consists mainly of attribute names, attribute types, attribute values and data. WEKA also allows import of the CSV file format.



Figure 4.2: WEKA GUI.

4.4. Classifiers

In order to select the best classifiers, we consider the True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F-Measure and AUC. There are different packages of classifiers according to the official website of WEKA. We select the classifiers, which were able to find the class of an unknown app based on the classification algorithm. We chose the following classifiers:

- Trees: Includes tree classifiers, such as decision trees. J48 Classifier was chosen.
- Bayes: It contains Bayesian classifiers; Naive Bayes was chosen.
- Functions: Support Vector Machine, Regression algorithms and Neural networks are such examples. Classifier LibSVM was chosen.
- Lazy: Learning is done here during runtime; it consists of assigning weights to neighbors as calculated by distance. The IBk algorithm was chosen.

4.5. Evaluation

We used machine learning based data mining techniques to classify the malware applications. Firstly, we built ARFF file from the extracted features and train the dataset by using multiple algorithms. The selected features are collected in the signature database and divided into training data and test data and used to detect android malware applications by standard machine learning techniques. To check the accuracy of our permission-based detection method we also used dataset containing 1000 APK samples, including 500 malwares and 500 benign samples, with 10-fold cross-validation method. This method divides the dataset into ten parts and, takes one part the 10% as testing set and the rest 90% as training set, and repeats this process 10 times, each with a different testing and training sets.

To further evaluate and make a comparison with our proposed technique, we selected two other well-known information mining techniques, Principal Component Analysis (PCA) (Jolliffe, 2002) and Information Gain (Christopher, Prabhakar, and Hinrich, 2008). Both these techniques reduce the number of attributes to achieve a high detection rate.

PCA is a statistical method used to emphasize variation and bring out strong patterns in a dataset. The comprehensive features generated are usually called the main component. Since the reduction of dimensionality process maintains the original features and keeps new features as independent as possible, the new generated features, which is the main component, have more advantages than the original feature.

IG depends on the entropy of an attribute and selects a feature that provides the foremost information gain. The information gain tells us how important a certain attribute of the vectors is. Information gain is the amount of information gained by knowing the value of the attribute, which is the entropy of the distribution before the subdivision minus the subsequent entropy of the distribution. The greatest information gain is the least entropy.

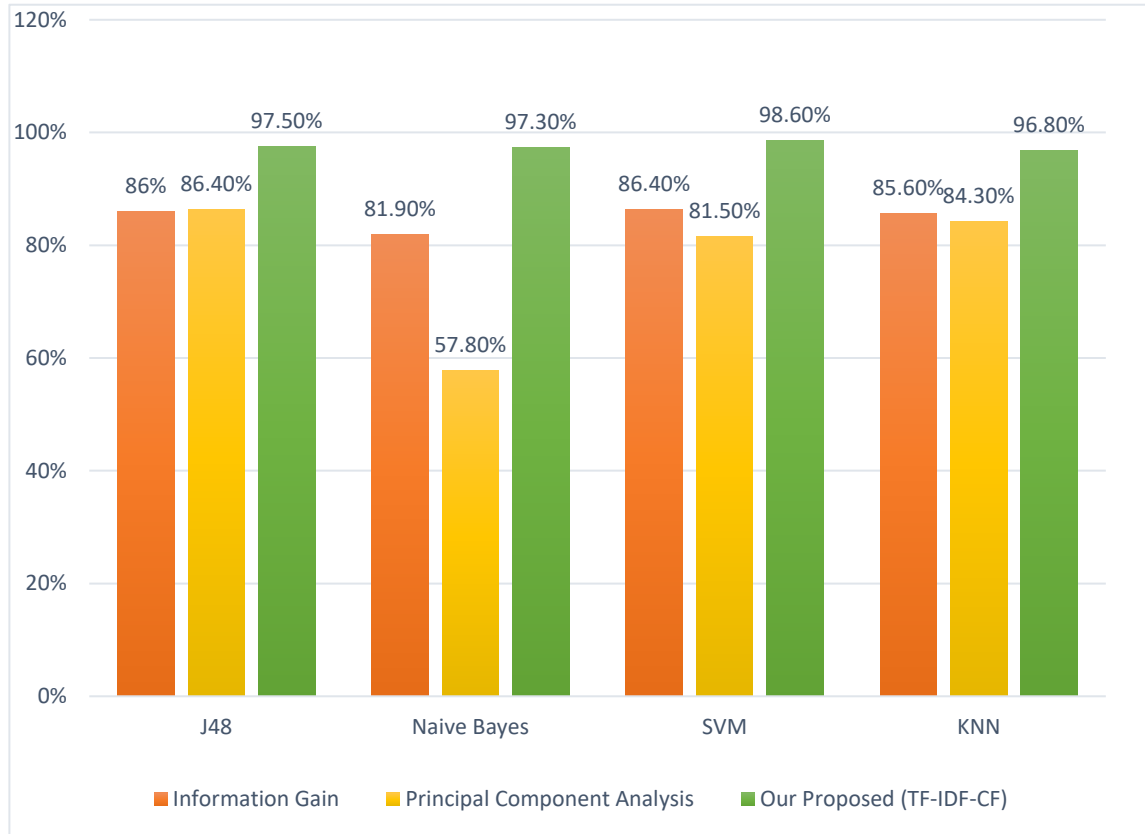


Figure 4.3: Overall Accuracy Rate for TF-IDF-CF, PCA and IG.

Using the same dataset, we compared our method to Information Gain and PCA. Results are shown in table 3. As seen in table 3, the use of our proposed method for assigning weights to each feature extracted from the APK has higher accuracy rate than the other two techniques PCA and IG.

Table 4. 3 Compared Results of TF-IDF-CF, PCA and IG

Weighting Method	K-Fold Cross Validation	Classifier Method	True Positive %	False Positive %	Overall Accuracy %
Information Gain	<i>10-fold CV</i>	J48 Decision Tree	0.86	0.14	86
	<i>10-fold CV</i>	Naive Bayes	0.819	0.181	81.9
	<i>10-fold CV</i>	Support vector machines	0.864	0.136	86.4
	<i>10-fold CV</i>	k-Nearest Neighbors	0.856	0.144	85.6
Principal Component Analysis	<i>10-fold CV</i>	J48 Decision Tree	0.864	0.136	86.4
	<i>10-fold CV</i>	Naive Bayes	0.578	0.422	57.8
	<i>10-fold CV</i>	Support vector machines	0.815	0.185	81.5
	<i>10-fold CV</i>	k-Nearest Neighbors	0.843	0.157	84.3
(TF-IDF-CF) Our Proposed Method	<i>10-fold CV</i>	J48 Decision Tree	0.975	0.025	97.5
	<i>10-fold CV</i>	Naive Bayes	0.973	0.027	97.3
	<i>10-fold CV</i>	Support vector machines	0.986	0.014	98.6
	<i>10-fold CV</i>	k-Nearest Neighbors	0.968	0.032	96.8

We used 1000 APK (500 malware and 500 benign) sample files, as shown above. We selected same number of samples to obtain more precise results. However, we also test using different number of samples at the table 4.4 to check the importance of class frequency. We use two datasets, first dataset contains 350 Malware apps and 500 benign apps, also the second dataset contains 500 Malware apps and 350 Benign apps. Our main conclusions from both datasets are almost the same and overall conclusions are sustainably robust.

The results of our experiments show that our proposed method has still a good detection accuracy, with a low false positive rate as shown in tables 4.5 and 4.6, when tested with different datasets that contains different number of samples.

Table 4.4: Datasets with different number of samples.

	Samples	Malware	Benign	Attributes
Dataset1	850	350	500	385
Dataset2	850	500	350	385

Table 4.5: Experimental results by using Dataset1 with 10-Fold Cross Validation.

Dataset1	K-Fold Cross Validation	Classifier Method	True Positive Rate	False Positive Rate	Overall Accuracy %
	<i>10-fold CV</i>	J48	0.958	0.055	95.8
	<i>10-fold CV</i>	Naive Bayes	0.884	0.127	88.4
	<i>10-fold CV</i>	SVM	0.960	0.093	92.1
	<i>10-fold CV</i>	k-NN	0.924	0.086	92.4

Table 4.6: Experimental results by using Dataset2 with 10-Fold Cross Validation.

Dataset2	K-Fold Cross Validation	Classifier Method	True Positive Rate	False Positive Rate	Overall Accuracy %
	<i>10-fold CV</i>	J48	0.946	0.057	94.6
	<i>10-fold CV</i>	Naive Bayes	0.867	0.114	86.7
	<i>10-fold CV</i>	SVM	0.954	0.048	95.4
	<i>10-fold CV</i>	k-NN	0.909	0.092	90.9

5. CONCLUSIONS

In this section, we outline our conclusions regarding static analysis of Android malware. We first propose a conclusive summary of android and our contributions in Section 5.1, followed by contributions and Results in Section 5.2 and future work in Section 5.3.

5.1. Summary

Today, life activities for all people depend on the latest technologies that provide fast and available communication and production services, smartphones are one of those technologies, they are used everywhere, by everyone, for almost any purpose. Android is the most common OS for smartphones and has the largest number of malwares. The wide use of smartphone applications results in wide growth in malware applications that threaten users. Smartphone malware is and will remain a potential security threat. The ability to extend its functionality with third - party applications is one of the most attractive features of a smartphone. Unfortunately, such a feature inevitably leads to the threat of malicious smartphone applications, otherwise known as malware for smartphones.

5.2. Contributions and Results

This work's contribution is a collection of smartphone malware detection information and methods. The approaches presented are based on static analysis and include all relevant steps from extraction of data and observables to the mechanism of detection running on. Permission is one of the most important features in Android security, and meaningful in malware detection. Our proposed permission-based framework uses machine learning algorithms to detect potentially malware apps. Also, to improve the efficiency of permission-based Android malware analysis and detection we introduced a new method based on TF-IDF to assign weight to each feature extracted from an Android APK file. We evaluated the new technique using different metrics and achieved a detection rate higher than 95.3% using different classifier algorithms. In addition to our

work relating to static analysis of Android applications, we have also provided a new method for assigning weights to each feature extracted from the APK. This new method reinforces the traditional TF - IDF technique to increase system efficiency. The result of this research shows that TF-IDF-CF seems to be an efficient method for the detection of Android malware. However, due to its high correlation with the used dataset, we conclude that in the first step to identify malicious applications, a permission-based malware detection mechanism can be used as a filter. To detect malware, it still needs a further analysis.

5.3. Future Work

Mobile devices have become ubiquitous in our lives because they provide virtually the same functionality as personal information processing systems. In recent years, in recent years, malware developers mostly are targeting the Android platform. As a future work, we plan to expand the data set by immigrating it with other existing Malwares datasets, the false detection will be reduced. To further improve malware detection, in future it will be added more features, such as, API calls etc., to our proposed framework. To improve the weight assigning method, in future, we are going to divide the data (especially malware samples) into more (than two) classes based on similarity measures.

REFERENCES

- Aafer Y., Du W., Yin H. (2013), "Droidapiminer: Mining api-level features for robust malware detection in android", In International conference on security and privacy in communication systems (pp. 86–103).
- Abawajy J., Huda S., Sharmeen S., Hassan M. M., Almogren A. (2018), "Identifying cyber threats to mobile-iot applications in edge computing paradigm", Future Generation Computer Systems, 89, 525–538.
- Arp D., Spreitzenbarth M., Hubner M., Gascon H., Rieck K., Siemens C. (2014), "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket", In Ndss (Vol. 14, pp. 23–26).
- Aung Z., Zaw W. (2013), "Permission-based android malware detection". International Journal of Scientific and Technology Research, 2(3), 228–234.
- Barrera D., Kayacik H. G., Van Oorschot P. C., Somayaji, A. (2010), "A methodology for empirical analysis of permission-based security models and its application to android", In Proceedings of the 17th ACM conference on Computer and communications security (pp. 73–84).
- Brahler S. (2010), "Analysis of the android architecture", Karlsruhe Institute for Technology, 7(8).
- Christopher D. M., Prabhakar R., Hinrich, S. (2008), "Introduction to information retrieval", New York, NY, USA: Cambridge University Press.
- Ehringer D. (2010), "The dalvik virtual machine architecture", Techn. Report (March 2010), 4(8).
- Enck W., Ongtang M., mcdaniel P. (2009), "On lightweight mobile phone application certification", In Proceedings of the 16th ACM conference on Computer and communications security (pp. 235–245).
- Felt A. P., Greenwood K., Wagner D. (2011), "The effectiveness of application permissions", In Proceedings of the 2nd USENIX conference on Web application development (p. 7).
- Hand D. J. (2007), "Principles of data mining", Drug Safety, 30(7), 621–622.
- Jolliffe I. T. (2002), "Principal Component Analysis", New York, NY, USA: Springer-Verlag.
- Joseph J., Kesavaraj G., (2018), "An empirical study of different data mining techniques and algorithms", International Journal of Technical Innovation in Modern Engineering and Science, 4(12), 109-113.

Liu X., Liu J. (2014), "A two-layered permission-based Android malware detection scheme", In Mobile cloud computing, services, and engineering (mobilecloud), 2014 2nd IEEE International Conference on (pp. 142–148).

Moser A., Kruegel C., Kirda E. (2007), "Limits of static analysis for malware detection", In Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007) (pp. 421–430).

Patil T. R., Sherekar S. S. (2013), "Performance analysis of Naive Bayes and J48 classification algorithm for data classification", International Journal of Computer Science and Applications, 6(2), 256–261.

Pearce P., Felt A. P., Nunez G., Wagner D. (2012), "Addroid: Privilege separation for applications and advertisers in android". In Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (pp. 71–72).

Peng H., Gates C., Sarma B., Li N., Qi Y., Potharaju R., ... Molloy I. (2012), "Using probabilistic generative models for ranking risks of android apps", In Proceedings of the 2012 ACM conference on Computer and communications security (pp. 241–252).

Popper B. (2017), The Verge, <https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users>, Date of access: 14 March, 2018.

Rovelli P., Vigfússon Ý. (2014), "Pmds: Permission-based malware detection system", In International Conference on Information Systems Security (pp. 338–357).

Sarma B. P., Li N., Gates C., Potharaju R., Nita-Rotaru C., Molloy I. (2012), "Android permissions: a perspective combining risks and benefits", In Proceedings of the 17th ACM symposium on Access Control Models and Technologies (pp. 13–22).

Smola A. J., Schölkopf B. (2004), "A tutorial on support vector regression", Statistics and Computing, 14(3), 199–222.

Szalay A., Chandraiah J. (2014), "evolution of android exploits from a static analysis tools perspective", In evolution of android exploits from a static analysis tools perspective (p. 9).

Thornton C., Hutter F., Hoos H. H., Leyton-Brown K. (2013), "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms", In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 847–855).

Statista, (2018), <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store>, Date of access: 12 January, 2019.

APKTool (2019), <https://ibotpeaches.github.io/Apktool>, Date of access: 12 March, 2018.

Xiaoyan Z., Juan F., Xiujuan W. (2014), "Android malware detection based on permissions", The Institute of Engineering and Technology.

Zhang M.-L., Zhou Z.-H. (2007), ML-KNN: "A lazy learning approach to multi-label learning", Pattern Recognition, 40(7), 2038–2048.

Zhou Y., Jiang X. (2012), "Android malware genome project", Disponibile a [Http://Www.Malgenomeproject. Org](http://www.malgenomeproject.org).

BIOGRAPHY

Abdirashid Ahmed Sahal was born in 1991 in Somalia. He studied his primary and secondary school in Somalia. He earned his bachelor's degree in Information Technology from SIMAD University, Somalia in 2013. He currently pursues his master's degree at Gebze Technical University Graduate School of Natural and Applied Sciences Department of Computer Engineering.

APPENDICES

Appendix A: Publications/presentations on the thesis,

A. A. Sahal, S. Alam and I. Soğukpınar, "*Mining and Detection of Android Malware Based on Permissions*," 2018 3rd International Conference on Computer Science and Engineering (UBMK), Sarajevo, 2018, pp. 264-268.