PERMISSION-BASED
ANOMALOUS APPLICATION DETECTION
ON ANDROID SMART PHONE

HTET HTET WIN

M.C.Sc. **JANUARY 2019**

PERMISSION-BASED
ANOMOLOUS APPLICATION DETECTION
ON ANDROID SMART PHONE

By

HTET HTET WIN

B.C.Sc.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Computer Science

(M.C.Sc.)

University of Computer Studies, Yangon

JANUARY 2019

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere thanks to those who helped me with various aspects of conducting research and writing this thesis.

First and foremost, I would like to express my gratitude and my sincere thanks to Professor **Dr. Mie Mie Thet Thwin**, the Rector of the University of Computer Studies, Yangon, for allowing me to develop this thesis.

My sincere gratitude goes to **Dr. Zon Nyein Nway**, my supervisor, Lecturer of Information Science Department, the University of Computer Studies, Yangon, for giving a great deal of her time, always finding enough to advise every step of the thesis process, to read and correct the thesis presentations as well as system implementation, and make constant assessment on my work done. Honestly, because of her continuous pilot, I admit that my thinking and writing perspective of the thesis are so much stronger than in my initial stage.

I would like to express my special appreciation to **Dr. Thi Thi Soe Nyunt**, Professor, and Head of Faculty of Computer Science, for her administrative supports and encouragements in development of the thesis.

I also wish to express my gratitude to **Daw Khin Mar Kyu,** Lecturer, Department of English, the University of Computer Studies, Yangon, for editing this thesis from the language point of view.

Moreover, I would like to extend my thanks to all my teachers for their support not only for the fulfillment of the degree of M.C.Sc. but also for my life.

I also thank my friends and colleagues for supporting in various ways to complete this thesis.

Last but not least, my family and my special friend: Phyoe Paing Paing Minn deserve special thanks for their love, care and immense support throughout my whole life as well as during my thesis.

STATEMENT OF ORIGINALITY

I hereby certify that the work embodies in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

---------- ----------

Date Htet Htet Win

ABSTRACT

Information applications are widely used by millions of users to perform many different activities. Android-based smart phone users can get free applications from Android Application Market. But, these applications were not certified by legitimate organizations and they may contain malware applications that can steal private information from users.

The proposed system develops a permission-based malware detection to protect the privacy of android smart phone users. This system monitors various permissions obtained from android applications and analyses them by using a statistical technique called **Singular Value Decomposition** (SVD) to estimate the correlations of permissions. The dataset including approximately 4000 malware JSON files are downloaded from https://www.kaggle.com/goorax/static-analysis-of-android-malware-of-2017. The training phase emphasizes on the malware samples (approximately 300) which includes the most significant patterns of the current malware environment according to the analysis results. The testing phase is conducted on 120 malware and goodware apps.

The proposed system evaluates the risk level (High, Medium, and Low) of Android applications based on the correlation patterns of permissions. The overall accuracy of the system is 85% for malware applications and goodware applications as the test results.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER 1
INTRODUCTION

Android is the powerful operating system supporting a large number of applications in smart phones. These applications make life more comfortable. With the repaid growing of Android application every day, there are growing threats for the mobile users by installing more malwares without ability to detect them before installing the applications to the user device. Malware name came from “Malicious Software”, its software was designed to secretly access a system without the owner’s device knowledge. A key challenge is to identify a suspected application as anomalous (malware). Therefore, the system that can detect whether the particular app is malicious or not is proposed and the installation can be canceled if the permissions are unacceptable [5].

According to the future of mobile, there were 4.1 billions of Internet users at 2018. And there will be 5.4 billions at 2025. As the Internet users are increasing around the world. On the other hand, there also increases the people who connect the Internet via mobile. So, there will be 80% of Internet users who make Internet connections via mobile at 2025. Moreover, 50% of transactions will be made by phone at 2050.

There are a lot of attacks such as device attack, network attack or datacenter attack, etc. Moreover, there are many different ways to attack. For example, in device attack, attackers can attack our mobile phone through browser, phone, sms or applications, etc. Among them, the system is intended to detect applications which use unintended permissions (Misconfigured apps can open doors to attackers by providing unintended permissions).

1.1 Overview of the Proposed System

To implement the proposed system, the first thing is to collect the information about the risky apps as much as we can and then need to analyze the nature of risky apps. According to the literature, there are so many ways to analyze different kinds of apps such as by analyzing signature features, behavior features or anomaly features and so on. Among them, we choose to analyze the apps based on permissions.

Because permission is the main gate to allow the application (which operations must be done). So, the permissions of android application are need to learn.

There are a lot of permissions that are declared by Google. Moreover, there are also customized permissions. The specific permission has its own task such as reading contacts, or sending sms or getting GPS, etc. Some of them are dangerous. Some of them are normal. Some of them are nothing meaning etc. But when analyzing permissions, it isn't enough to know which permissions are dangerous and which permissions are normal. One application can use as much as permissions according to the developer. And, we cannot conclude that an application has high risk by seeing one of dangerous permissions.

So, the correlation patterns of permissions are usually involved in high risk application. To get the correlation patterns of permissions, Singular Value Decomposition (SVD) technique is chosen. To apply SVD technique, the original matrix (permission-app matrix) is needed. For choosing the training dataset, malware dataset is needed to train since the system gives the knowledge that how much risk level has an incoming application. That kind of dataset didn’t download easily as malware based dataset are very restricted.

The required dataset is obtained from https://www.kaggle.com/goorax/static-analysis-of-android-malware-of-2017. Kaggle website describes the specific analysis results of malware applications by separating into four folders. These folders are apkMetaReport, byteCodeReport, virusTotalReport, and assestReport. apkMetaReport folder contains the contents of Manifest.xml files. byteCodeReport folder contains the contents of classes.dex. virusTotalReport folder contains the reports of virusTotal service. assestReport folders contains names of assests and lib contents. So, we choose to download apkMetaReport. That dataset contains over 4000 json files (one Json file for one malware application). An android app name is changed via its sha256 hash value to be used as its file name.

1.2 Problem Definition

Mobile devices are replacing desktops and laptops, as they enable the users to access email, Internet, GPS navigation, and the storage of critical data such as contact lists, passwords, calendars, and login credentials. Also, recent developments in mobile commerce have enabled users to perform transactions such as purchasing goods and applications over wireless networks, and even banking from their smart phones [2]. Believing that surfing the internet on mobile devices is safe, many users fail to enable existing security software. And applications use a lot of permissions to access the SD card, use the Internet and so on. The number of users is ignoring that permissions as they don't understand the permission information, but this harms to our mobile devices. This causes unwanted things like break the security of our mobile phone or else this can effect on our sensitive information. Therefore, if the specific-app permissions risks are known, the installation of that app can be eliminated [10].

1.3 Objectives of the Thesis

The objectives of the thesis are as follows:

- To develop a malware application detection system for android smart phone.
- To support the user about the risk level information of application before installing it.
- To know the statistical correlations of permissions using Singular Value Decomposition (SVD).

1.4 Organization of the Thesis

This thesis is organized into five chapters.

Chapter 1 includes introduction, overview of the proposed system, problem definition and objectives of the thesis.

Chapter 2 describes the detail information about android mobile operating system and android malware including types of android malware and kinds of malware detection and analysis methods.

Chapter 3 explains the proposed system methodology.

Chapter 4 presents design and implementation of proposed system which includes system flow diagram, database design, data dictionary, screen designs of the proposed system, empirical results.

The last chapter, Chapter 5 includes the conclusion of the system, advantages and limitations of the system and future extension.

CHAPTER 2
ANDROID OPERATING SYSTEM AND ANDROID MALWARE

Nowadays, android is the most popular mobile operating system, based on the Linux kernel, primarily designed for touchscreen mobile devices. Google became involved with the financial backing of Android Inc. in 2005, with smartphones using the operating system, which debuted in 2008 (HTC Dream). The operating system is open source, distributed under the Apache License, leading to rapid development by many globally. According to AppBrain, over 1.1 million Android apps exist in the market as of February 13, 2014, with 22 percent identified as low-quality apps.

The architecture of the Android operating system is well published, involving the Linux kernel, libraries, an application framework, applications, and the Dalvik Virtual Machine (DVM) environment. To gain "root" on a device one must gain access to the core Linux kernel running an Android device. Most Android malware do not attempt to perform exploits to get to root, as that is not required for nefarious motives. Rather, apps are commonly modified to add in a hidden Trojan component so that the Trojan is also installed when a user installs an app. Once installed and run, Android malware may employ a wide variety of permissions enabled for the app to then send text messages, and phone and geolocation information to manage and intercept all types of communications and more [8].

2.1 Introduction to Android Operating System

Android operating system versions are named after consumables starting with version 1.5. The version where each platform name was first provided is in parenthesis: Cupcake (1.5), Donut (1.6), Eclair (2.0), Froyo (2.2), Gingerbread (2.3), Honeycomb (3.0), Ice Cream Sandwich (4.0), Jelly Bean (4.1), and KitKat (4.4), with Key Lime Pie (5.0) expected in the future. There is a pattern in the naming of each version, can you spot it? Each version introduces new functionality and requirements. For example, KitKat, the most recent release, is designed to streamline memory usage for maximum compatibility with all devices in party by introducing new application programming interface (API) solutions, such as

"ActivityManager.isLowRamDevice()", tools like meminfo for developers. Back to the teaser above each version of Android is named after a sequential letter in the English alphabet, with versions Cupcake through KitKat representing versions C, D, E, F, G, H, I, J, and K. The next major version following Key Lime Pie should start with the letter L and be a dessert item such as Ladyfingers, Lemon Meringue Pie, or Licorice [11].

2.1.1 Android Architecture

Android is a software stack meaning that it features four main software layers (from top to bottom): the application layer, the framework layer, the runtime and native libraries layer and the kernel layer.

The top layer features Android applications. Typical Android applications are: the Home application which is the first running application that displays icons to start other applications; the Contact application to manage the list of contact; the Phone application to give phone calls; and the Browser application to visit web resources. Users of devices running Android can install more applications on their device, usually by downloading them from a repository such as F-Droid1 or the official Google market named Play Store2. Applications are mainly written in the Java programming language but can also contain native code. Applications rely on the framework layer to communicate with the system [16].

The framework layer is an interface written in Java between applications and the rest of the system. It provides facilities to retrieve information from a system resource (e.g. the application can retrieve GPS coordinates through the Location Manager) or to ask the system to call them back when there is a new event (e.g. ask the TelephonyManager to notify the application when there is a phone call).

The third layer features two distinct entities: the Android runtime and the native libraries.

- The Android runtime consists of the Dalvik virtual machine, which executes Android applications' Dalvik bytecode3, and Android core libraries, basically Java classes, which applications can leverage (e.g. application can use the Https URL Connection class to open a secure connection to a website). Some libraries contain wrappers around native libraries. For instance, Java classes for the core library handling secure connections to websites such as Https

URL Connection may use the Open SSL native library depending on the environment's configuration.

- The native libraries4 provide basic building blocks that can be used by applications, the framework layer or core libraries. Applications can have native code that directly uses the native OpenGL library for fast graphic processing. The framework layer can use the native SQLite library to store data.

The lowest layer is the Linux kernel. From upper software layers it can be seen as an interface to the hardware (CPU, memory,). Indeed, it is responsible for running programs on the CPU5 and it has a number of drivers to handle different hardware such as the display, the audio, and drivers to manager network communication. It also features a special driver for efficient Inter-Process Communication called the Binder driver.

An Android application can use elements from the framework layer, core and native libraries as well as directly communicate with the kernel. The Android system implements security features to prevent applications from having access to every part of the system. In short, developers give a list of permissions to every application they write. This list specifies what the application is allowed to do on the system and has to be validated by the user at installation time. When an application is installed, it is given a User ID (UID). Every Android application can be seen as a Linux user. Moreover, the Android system has a list of mapping for each permission to a Group ID (GID). For every permission the application declared, the system adds the application (or more precisely the corresponding Linux user) to the corresponding GID. So, if an application does not have the GPS permission and wants to retrieve the GPS coordinates through the LocationManager or the Linux driver for the GPS, the Android system detects that the application is not in the GPS group and prevents it from accessing GPS data [11].

2.1.2 Structure of Android Application

An Android application is a compressed zip file signed with the private key K of the developer. It contains the Dalvikbytecode of the application (compiled from the Java source code), data the application needs (pictures, sound,) and a manifest file

describing the application's structure and permissions the application requires. In short,

Application = Sign(Zip(DalvikBytecode;Manifest;Data);K):

The fact that Android applications are signed with the private key of the developer ensures that applications can only be updated by code signed by the same developer and that applications signed with the same key have the possibility to share permissions and UID. However, it does not guarantee the authenticity of the author of the application since certificates can be self-signed (e.g., anyone could claim to be John Doe).

Components. Android applications are made of components. There exist four kinds of components: activity, service, content provider and broadcast receiver. Activity components are used for the GraphicalUser Interface (GUI). They display graphical elements such as buttons, lists or pictures. Service components are used for computational intensive tasks or tasks that take a long time such as playing an audio file. Content providers are used to share data between applications. For instance, the list of contact is implemented as a content provider so that any application can have access to it (if it has the proper permission). Finally, broadcast receiver components receive messages from the system or other applications (e.g. an SMS has been received by the system). Concretely, every component is a Java class which inherits from a specific super class such as Activity, Service, etc.

Communication with Intent and URI. Components of an Android application usually communicate using special system methods called Inter-Component Communication (ICC) methods. There are about forty ICC methods which a component can use to communicate with another component. The most used ICC method is startActivity(Intent). This method is used to tell the system to start a new activity component described by the method's parameter.

Intent. Components can communicate with one another using an abstract object called Intent. Communications can take place between components of a single application or between components of multiple applications. When component A wants to communicate with component B, it initializes an Intent and sets component B as the destination. This kind of communication is said to be explicit because the target component is explicitly specified. A communication can also be implicit in which case the source component initializes the Intent with the action it would like to perform (e.g. view a pdf document). When the component sends the Intent, the system

checks for components having the action in their intent filter. The selection of the target component can be done automatically by the system or may require user intervention if multiple components can handle the action. For instance, if Activity3 sends an Intent with action "view txt" the system starts Activity2 since it is the only component having the "view txt" intent filter. Intents can encapsulate data in form of key/value pairs in objects called Bundles. Intents are used for communications between activities, service and broadcast receivers.

URI. A URI, or Uniform Resource Identifier, identifies an abstract or physical resource. In short a URI is used to communicate with content providers. They may also be used to initialize Intents to target specific resource. As an example link, URI:content://com.android.calendar/events, it can be cut into three parts. The first one, content, identifies how to access the resource. The reader may already know the http scheme for accessing web pages through the HTTP protocol. Content means that access to the resource is done through a content provider. The second part, com.android.calendar, called the authority identifies the holder of the resource. The reader may be familiar with authorities such as mywebsite.com which identify a registered host on the Internet. In our example, the authority identifies the content provider called com.android.calendar which has been registered to the Android system. Finally, events, called the path, is the part identifying the target resource. The reader may be familiar with paths such as index.html identifying web page resources. In example, this is the database table events of the content provider.

```
<manifest package="com.android.providers.calendar">
        <application android:process="com.android.calendar">
                <provider android:name="CalendarProvider" />
                <service android:name="CalendarSyncAdapterService" >
                        <intent-filter>
                                <action android:name="SyncAdapter" />
                        </intent-filter>
                </service>
                <activity android:name="CalendarContentProvider" >
                        <intent-filter>
                                <action android:name="MAIN" />
                                <category android:name="UNIT_TEST" />
                        </intent-filter>
```

```
            </activity>
            <receiver android:name="CalendarReceiver">
                <intent-filter>
                    <action android:name="BOOT_COMPLETED" />
                </intent-filter>
            </receiver>
        </application>
        <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Figure 2.1 Example of Android Manifest

The Manifest File. The manifest describes the application's structure in terms of components. A component can be exported so that other applications can use it. It can also declare intent filters to specify to the system what kind of action or data it handles. The manifest also lists all the permissions that the application requests (e.g. INTERNET, GPS). An example of manifest is presented in Figure 2.1. It declares an application with one content provider, one service, one activity and one broadcast receiver. The service only accepts intent with action SyncAdapter, the activity intents with action MAIN and category UNIT_TEST and the broadcast receiver intents with action BOOT_COMPLETED [2].

2.1.3 Android Permission

Application vendors define a set of permissions for each application. For installing an application, the user has to approve as a whole all the permissions the application's developer has declared in the application manifest. If all permissions are approved, the application is installed and receives group memberships. The group memberships are used to check the permissions at runtime. For instance, an application *Foo* is given two group memberships net_bt and inet when installed with permissions BLUETOOTH and INTERNET, respectively. In other terms, the standard Unix ACL is used as an implementation means for checking permissions.

Android 2.2 defines 134 permissions in the android.Manifest$permission system inner class, whereas Android 4.0.1 defines 166 permissions. This gives us an upper-bound on the number of permissions which can be checked in the Android framework.

Android has two kinds of permissions: high level and low level permissions. High-level permissions are only checked at the framework level (that is, in the Java code of the Android SDK). Android 2.2 declares eight low-level permissions which are either checked in C/C++ native services (RECORD AUDIO for instance) or in the kernel (e.g., when creating a socket).

In this chapter, we focus on the high-level permissions that are only checked in the Android Java framework [4].

2.2 Android Malware

One of the biggest problems that Internet surfers face today on the World Wide Web is malware. Malware is short hand for malicious software. It is software developed by cyber attackers with the intention of gaining access or causing damage to an electronic device's normal operation. Malware can infect personal computers, smartphones, tablets, servers and even equipment – basically any device with computing capabilities. As technology, computing and software have advanced during the last two decades, so has the sophistication and prevalence of malicious software. Malware is installed on your electronic device usually without your knowledge and it can enter your electronic device as a result of surfing the Internet and in a variety of different ways. Once it sneaks into your device, malware is capable of spying on your surfing habits, logging your passwords by observing your keystrokes, stealing your identity, reading your email, and variety of other invasive tactics [12].

2.2.1 Types of Android Malware

Mobile malware is malicious software that is specifically built to attack mobile phone or smartphone systems. These types of malware either install themselves or are installed on the device by unwitting mobile users, and then perform functions without user knowledge or permission. Malicious mobile apps are often disguised as legitimate applications. They can be distributed through the internet via mobile browsers, downloaded from app stores or even installed via device messaging functions. The insidious objectives of mobile malware range from spying to keylogging, from text messaging to phishing, from unwanted marketing to outright fraud. There is malware out there targeting every mobile platform – from Apple iOS to WinMobile to Blackberry – yet the vast majority of mobile malware programs

today target Google Android users. Some researchers report a rate of infection as high as 90 percent, due to Google's open app development and distribution model.

(a) Spyware

Figure 2.2 Spyware

Spyware on your Android will monitor record and send all your information to the attackers. It will steal all the information you enter on your Android device. Spyware will come attached with some application and it will go unnoticed until some security software is installed on your device. Most of the applications that you directly download from the Internet contain spyware.

(b) Adware

Figure 2.3 Adware

This is the most common and all time popular android malware that a smartphone phone gets infected with. Having adware on the device can be a very frustrating thing, as you will receive continuous popups and ads on your screen. Also, if any of the ads is clicked then another malicious program will be downloaded or some unwanted application will be installed on your device.

(c) Trojans

Figure 2.4 Trojans

Mobile Trojans infect user devices by attaching themselves to seemingly harmless or legitimate programs, are installed with the app and then carry out malicious actions. Such programs have been known to hijack the browser, cause the device to automatically send unauthorized premium rate texts, or capture user login information from other apps such as mobile banking.

(d) Viruses

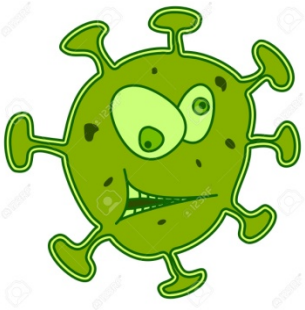

Figure 2.5 Viruses

Mobile viruses can be installed on the device any number of ways and cause effects that range from simply annoying to highly-destructive and irreparable. Malicious parties can potentially use mobile viruses to root the device and gain access to files and flash memory.

(e) Phishing Apps

Mobile browsing of the internet is growing with smartphone and tablet penetration. Just as with desktop computing, fraudsters are creating mobile phishing sites that may look like a legitimate service but may steal user credentials or worse. The smaller screen of mobile devices is making malicious phishing techniques easier

to hide from users less sophisticated on mobile devices than PCs. Some phishing schemes use rogue mobile apps, programs which can be considered “trojanized”, disguising their true intent as a system update, marketing offer or game. Others infect legitimate apps with malicious code that’s only discovered by the user after installing.

(f) Malware that can make Calls and Send SMS

Another type of malware that users encounter is the malware which will make fake calls and send SMS to the contacts. The messages that are being sent contains malicious links, and which the receiver taps on the links they will also get infected by the malware.

2.2.2 Ways of Malware Infection

Cybercriminals looking to have a greater return focus their efforts on organizations and use a variety of tactics to infect the maximum number of corporate device with their malware variants.

- **Infected application:** The most common way for a smartphone to get infected is by downloading an app that has a virus or malware embedded inside the app code. Malware operators will usually choose popular apps to repackage or infect, increasing the likelihood that victims will download their rogue version. Sometimes, however, they will come up with brand new applications. Infected applications are usually found on third-party app stores. When the app is installed, the virus or malware infects the smartphone operating system.
- **Malvertising:** Malvertising is the practice of inserting malware into legitimate online ad networks to target a broad spectrum of end users. The ads appear to be perfectly normal and appear on a wide range of apps and web pages. Once the user clicks on the ad, his or her device is immediately infected with the malware. Some more aggressive malvertisements for example, take up the entire screen of the device while the user is browsing the web. Faced with this situation, many users’ first response will be to touch the screen, triggering the malicious download.
- **Scams:** Scams are common tools used by hackers to infect mobile devices with malware. They rely on a user being redirected to a malicious web page,

either through a web redirect or pop-up screen. In more targeted cases, a link to the infected page is sent directly to an individual in an email or text message. Once the user is taken to the infected site, the code within the page automatically triggers the malicious software download. The website is usually disguised to look legitimate in order to get users to accept the file onto their devices.

- **E-mail attachments:** It may also be possible for an e-mail to infect a smartphone if the user attempts to open an attachment on their smartphone and that attachment has a virus or malware. For example, an infected PDF attachment can infect a smartphone.
- **SMS or bad website:** Another common tactic to infect smartphones is done through an SMS. For example, an unknown contact could send you a link to visit that sends you an infected attachment, attempts remote control, or attempts to phish private information from you.
- **Direct to Device:** Possibly the most James Bond-esque infection method, direct to device, dictates that the hacker must actually touch the phone in order to install the malware. Usually, this involves plugging the device into a computer and directly downloading the malicious software onto it (also known as side loading) [14].

2.2.3 Android Malware Preventions

The best way to protect android smartphones is to only download apps from a verified, reputable source. Google Play is the best place to download apps. Apps in online stores are checked for viruses and malware and much less likely to cause problems for android smartphones.

You can also download and install antivirus and anti-malware apps for your android smartphone. For example, AVG AntiVirus is available for Android phones, and Kaspersky Safe Browser are examples of apps that help protect android smartphones from malware.

Android malware is increasingly common, and that means mobile device-users need to be on guard when it comes to what types of apps they choose to download. Through malicious malware — in the form of apps — hackers can easily take hold of

your personal data. Users who don't take security seriously will be at a greater risk for downloading these dangerous apps. To prevent android malwares from invading the mobile devices, some important guidelines are described.

- Guard your privacy by taking time to read the permissions the app requires. Think about whether they match the purpose of the app; granting the wrong permissions can send your sensitive data off to third parties.
- Read the app's reviews. Check to see if there are any strange concerns or experiences with the app.
- Avoid downloading apps from third-party marketplaces. That's exactly where hackers plant their malware-ridden apps.
- Stay away from dodgy websites and always check if the developers are legitimate. If you've never heard of them, see if there have been any concerns about them published online.
- Be wary of a free antivirus trial, because it could be malware in disguise that attacks your mobile device. Affordable Android security software is available from trusted vendors, and it effectively does the job of blocking malicious apps [14].

2.2.4 Android Malware Detection

The popularity of Android mobile devices has gone up in our lives and are being used for handling a lot of our personal and confidential information. Hence they are now an ideal target for attackers. Android based smart-phone users can download a lot of free applications from Android Application Market/Play Store. At the same time, the increasing number of security threats that target mobile devices has emerged. In fact, malicious users and hackers are taking advantage of both the limited capabilities of mobile devices and the lack of standard security mechanisms to design mobile-specific malware that access sensitive data, steal the user’s phone credit, or deny access to some device functionalities. To mitigate these security threats, various mobile specific Intrusion Detection Systems (IDSes) have been recently proposed. Most of these IDSes are behavior-based, i.e. they don’t rely on a database of malicious code patterns, as in the case of signature-based IDSes [15].

2.2.5 Android Malware Analysis

Malware analysis is the process of determining the purpose and functionality of a given malware sample such as a virus, worm, or Trojan horse. This process is a necessary step to be able to develop effective detection techniques for malicious code. In addition, it is an important prerequisite for the development of removal tools that can thoroughly delete malware from an infected machine. Traditionally, malware analysis has been a manual process that is tedious and time-intensive. Unfortunately, the number of samples that needs to be analyzed by security vendors on a daily basis is constantly increasing**.** The process of analyzing a given program during execution is called dynamic analysis; while static analysis refers to all techniques that analyze a program by inspecting it.

(a) Static Malware Analysis

Analyzing software without executing, it is called static analysis. Static analysis techniques can be applied on different representations of a program. If the source code is available, static analysis tools can help finding memory corruption flaws and prove the correctness of models for a given system. Static analysis tools can also be used on the binary representation of a program. When compiling the source code of a program into a binary executable, some information gets lost. This loss of information further complicates the task of analyzing the code.

The process of inspecting a given binary without executing is mostly conducted manually. For example, if the source code is available, several interesting information, such as data structures, used functions and call graphs can be extracted. This information gets lost once the source code has been compiled into a binary executable and it will impede further analysis. Within the malware domain typically the latter is the case, since the source code of a current malware binary is typically not available [11].

Various techniques are used for static malware analysis. Some of those are described below.

•**File fingerprinting**: Beside examining obvious external features of the binary this includes operations on the file level such as computation of a cryptographic hash (e.g., md5) of the binary in order to distinguish it from others and to verify that it has not been modified.

•**Extraction of hard coded strings:** Software typically prints output (e.g., status-or error-messages), which ends up embedded in the compiled binary as readable text. Examining these embedded strings often allows conclusions to be drawn about internals of the inspected binary.

•**File format:** By leveraging metadata of a given file format additional, useful information can be gathered. This includes the magic number on UNIX systems to determine the file type as well as dissecting information of the file format itself. For example from a Windows binary, which is typically in PE format (portable executable) a lot of information can be extracted, such as compilation time, imported and exported functions as well as strings, menus and icons.

•**AV scanning:** If the examined binary is well-known malware, it is highly likely to be detected by one or more AV scanners. To use one or more AV scanner is time consuming but it becomes necessity sometimes.

•**Packer detection:** Nowadays malware is mostly distributed in an obfuscated form e.g., encrypted or compressed. This is achieved using a packer, whereas arbitrary algorithms can be used for modification. After packing, the program looks much different from a static analysis perspective and its logic as well as other metadata is thus hard to recover. While there are certain unpackers such as PEiD2, there is accordingly no generic unpacker. This makes a major challenge of static malware analysis.

•**Disassembly**: The major part of static analysis is typically the disassembly of a given binary. This is conducted utilizing tools, which are capable of reversing the machine code to assembly language, such as IDA Pro. Based on the reconstructed assembly code an analyst can then inspect the program logic and thus examine its intention. Usually this process is supported by debugging tools such as OllyDbg. The main advantage of static malware analysis is that it allows a comprehensive analysis of a given binary. That is, it can cover all possible execution paths of a malware sample. Additionally, static analysis is generally safer than dynamic analysis as the source code is not actually executed. However, it can be extremely time-consuming, cumbersome and thus requires expertise.

Limitation of Static Malware Analysis: Generally, the source code of malware samples is not readily available. That reduces the applicable static analysis techniques for malware analysis to those that retrieve the information from the binary

representation of the malware. Analyzing binaries brings along intricate challenges. Consider, for example, that most malware attacks host executing instructions in the IA32 instruction set. The disassembly of such programs might result in ambiguous results if the binary employs self modifying code techniques. Additionally, malware relying on values that cannot be statically determined (e.g., current system date, indirect jump instructions) exacerbate the application of static analysis techniques. The other is that malware authors know of the limitations of static analysis methods and thus, will likely create malware instances that employ these techniques to thwart static analysis. Therefore, it is necessary to develop analysis techniques that are resilient to such modifications, and are able to reliably analyze malicious software [5].

(b) Dynamic Malware Analysis

Executing a given malware sample within a controlled environment and monitoring its actions in order to analyze the malicious behavior is called dynamic malware analysis. Since Dynamic Malware Analysis is performed during runtime and malware unpacks itself, dynamic malware analysis evades the restrictions of static analysis (i.e., unpacking and obfuscation issues). Thereby it is easy to see the actual behavior of a program. Another major advantage is that it can be automated thus enabling analysis at a large scale basis. However, the main drawback is so-called dormant code: That is, unlike static analysis, dynamic analysis usually monitors only one execution path and thus suffers from incomplete code coverage. In addition, there is the danger of harming third party systems if the analysis environment is not properly isolated or restricted respectively. Furthermore, malware samples may alter their behavior or stop executing at all once they detect to be executed within a controlled analysis environment [11].

Mainly two basic approaches for dynamic malware analysis can be distinguished:

•**Analyzing the difference between defined points**: A given malware sample is executed for a certain period of time and afterwards the modifications made to the system are analyzed by comparison to the initial system state. In this approach, Comparison report states behavior of malware.

•**Observing runtime-behavior**: In this approach, malicious activities launched by the malicious application are monitored during runtime using a specialized tool.

An example for the first approach is Truman (The Reusable Unknown Malware Analysis Net). Thereby malware is executed on a real Windows environment rather than within a Virtual Machine. During runtime Truman provides a virtual Internet for the malware to interact with. After execution the host is restarted and boots a Linux image, which then mounts the previously used Windows image in order to extract the relevant data, such as the Windows registry and a complete file list. Finally, the Windows environment is reset to its initial clean state. By using a native environment Truman is able to circumvent possible anti-debugging measures of malware. However, since the result is only a snapshot of the infected system, information related to dynamic activities such as spawned processes and temporarily created files are lost. Hence observing the runtime-behavior of an application is currently the most promising approach. It is mostly conducted utilizing sandboxing. A sandbox hereby refers to a controlled runtime environment which is partitioned from the rest of the system in order to isolate the malicious process. This partitioning is typically achieved using virtualization mechanisms on a certain level. While in principle existing tools, such as chroot could be used to deploy such a controlled environment several sandbox environments dedicated to malware analysis exist implementing specialized techniques [12].

2.2.6 Symptoms of Malware Compromised Devices

Smartphone users are beginning to understand how important it is to protect their devices so malware can't be installed on them. However, many users are unaware of **what measures they can take to identify malicious activity** on their devices. Nonetheless, eventually the malicious activity will have to kick into action, and that's when you can pay attention to certain signs to detect illegitimate activity.

- **Battery Life is Much Shorter:** A hacked smartphone will have a much shorter battery life. If your phone is suddenly dying after a few short hours, it could be because spyware or another type of malware is running in the background.
- **Android is Performing Poorly:** Your phone may be performing poorly due to a lack of memory, but it could also be due to malware running in the background of your phone. If your phone is suddenly lagging behind, freezing,

refusing to load certain apps or web pages, or overheating, there may be malware on your phone.

- **Data Usage Has Increased:** If your phone bill shows a serious spike in data usage and other unusual charges — such as calls and texts to international numbers — then a hacker has gained access to your device. Although you may not notice this until your phone bill arrives, you can also check the data usage for each app on your phone. If one app — particularly an app you recently downloaded — is using much more data than it should be, then the app is likely malicious.
- **Adware and Pop-Ups Have Appeared:** This is a more obvious sign of a hacked smartphone. If pop-ups and advertisements are now appearing on your device, then your phone is surely infected.
- **Android is Sending Unusual Messages**: If friends, family, or acquaintances say that they receive a strange text, email, or Facebook message from you, then your phone and accounts have likely been hacked. If you've started to receive a lot of strange phone messages recently, this could also be a sign that a hacker has compromised your phone.
- **Websites appear somewhat different than before:** If someone has installed malware that is "proxying" on your device--that is, sitting between your browser and the internet and relaying the communications between them (while reading all of the contents of the communications and, perhaps, inserting various instructions of its own)--it might affect how some sites display.
- **Some apps stop working properly:** If apps that used to work properly suddenly stop working, that may also be a sign of proxying or other malware interfering with the apps' functionality.
- **Cell-phone bill shows unexpected charges:** Criminals can exploit an infected device to make expensive overseas phone calls on behalf of a remote party proxying through your device, can send SMS messages to international numbers, or ring up charges in other ways.

- **Data breaches and/or leaks:** Of course, if you have experienced some data leak you should always check to determine the source of the problem--and the process of checking obviously includes examining your smartphone.
- **The Internet connects on its own:** Viruses and other malware use your phone's data to spread its message. If you see that your phone is mysteriously switching your Wi-Fi and data connections on without your intervention, it could be due to malware. These programs can override your preferences and connect to the internet on their own. If you see unusual internet activity, scan your phone for viruses and clean using an anti-virus program [8].

CHAPTER 3
THE PROPOSED SYSTEM METHODOLOGY

This chapter provides a description of, the much required, theoretical foundation for our work and the general framework that we developed to carry out our experiments.

3.1 Malware Detection Systems

The popularity of Android mobile devices has gone up in our lives and is being used for handling a lot of our personal and confidential information. Hence they are now an ideal target for attackers. Android based smart-phone users can download a lot of free applications from Android Application Market/Play Store. At the same time, the increasing number of security threats that target mobile devices has emerged. In fact, malicious users and hackers are taking advantage of both the limited capabilities of mobile devices and the lack of standard security mechanisms to design mobile-specific malware that access sensitive data, steal the user's phone credit, or deny access to some device functionalities. To mitigate these security threats, various mobile specific Intrusion Detection Systems (IDSes) have been recently proposed. Most of these IDSes are behavior-based, i.e. they don't rely on a database of malicious code patterns, as in the case of signature-based IDSes [13].

Malware Detection: Malware detection is a field of study that deals with the analysis, detection and containment of malware. The greatest challenges in security tasks that are still battling the exploration of mobile communication devices, computer and network infrastructures, and web technology are Malware attacks, Malware detection and Malware analysis [9].

There are three different types of malware detection techniques.

1. **Attack or Invasion Detection:** Tries to detect unauthorized access by outsiders.
2. **Signature-based Detection (Misuse Detection):** Tries to detect misuse by insiders.
3. **Behavior-based Detection (Anomaly Detection):** Detects the patterns in a given dataset that do not conform to an established normal behavior.

3.2 Statistical Technique: Singular Value Decomposition (SVD)

Singular value decomposition (SVD) can be looked at from three mutually compatible points of view. On the one hand, we can see it as a method for transforming correlated variables into a set of uncorrelated ones that better expose the various relationships among the original data items. At the same time, SVD is a method for identifying and ordering the dimensions along which data points exhibit the most variation. This ties in to the third way of viewing SVD, which is that once we have identified where the most variation is, it is possible to find the best approximation of the original data points using fewer dimensions. Hence, SVD can be seen as a method for data reduction [6].

As an illustration of these ideas, consider the 2-dimensional data points. The regression line running through them shows the best approximation of the original data with a 1-dimensional object (a line). It is the best approximation in the sense that it is the line that minimizes the distance between each original point and the line. If we drew a perpendicular line from each point to the regression line, and took the intersection of those lines as the approximation of the original data point, we would have a reduced representation of the original data that captures as much of the original variation as possible. Notice that there is a second regression line, perpendicular to the first.

This line captures as much of the variation as possible along the second dimension of the original data set. It does a poorer job of approximating the original data because it corresponds to a dimension exhibiting less variation to begin with. It is possible to use these regression lines to generate a set of uncorrelated data points that will show sub groupings in the original data not necessarily visible at first glance.

These are the basic ideas behind SVD: taking a high dimensional, highly variable set of data points and reducing it to a lower dimensional space that exposes the substructure of the original data more clearly and orders it from most variation to the least. What makes SVD practical for NLP applications is that you can simply ignore variation below a particular threshold to massively reduce your data but be assured that the main relationships of interest have been preserved [2].

Full Singular Value Decomposition: SVD is based on a theorem from linear algebra which says that a rectangular matrix A can be broken down into the product of three matrices - an orthogonal matrix U, a diagonal matrix S, and the transpose of an orthogonal matrix V.

Reduced Singular Value Decomposition: Reduced singular value decomposition is the mathematical technique underlying a type of document retrieval and word similarity method variously called Latent Semantic Indexing or Latent Semantic Analysis. The insight underlying the use of SVD for these tasks is that it takes the original data, usually consisting of some variant of a word x document matrix, and breaks it down into linearly independent components. These components are in some sense an abstraction away from the noisy correlations found in the original data to sets of values that best approximate the underlying structure of the dataset along each dimension independently. Because the majority of those components is very small, they can be ignored, resulting in an approximation of the data that contains substantially fewer dimensions than the original. SVD has the added benefit that in the process of dimensionality reduction, the representation of items that share substructure become more similar to each other, and items that were dissimilar to begin with may become more dissimilar as well. In practical terms, this means that documents about a particular topic become more similar even if the exact same words don't appear in all of them. As we have already seen, SVD starts with a matrix, so we will take word x document matrix as the starting point [10].

3.2.1 Vector Terminology

The proposed methodology is based on statistical and matric method. So, some of the vector terminology is mentioned as follows.

(a) Vector Length

The length of a vector is found by squaring each component, adding them all together, and taking the square root of the sum. If $\vec{v}$ is a vector, its length is denoted by $|\vec{v}|$. More concisely,

$$|\vec{v}| = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{3.1}$$

For example, if $\vec{v} = [4, 11, 8, 10]$, then

$$|\vec{v}| = \sqrt{4^2 + 11^2 + 8^2 + 10^2} = \sqrt{301} = 17.35$$

(b) Vector Addition

Adding two vectors means adding each component in ~v1 to the component in the corresponding position in ~v2 to get a new vector. For example

$$[3, 2, 1, -2] + [2, -1, 4, 1] = [(3 + 2), (2 - 1), (1 + 4), (-2 + 1)] = [5, 1, 5, -1]$$

More generally, if $A = [a_1, a_2, \ldots a_n]$ and $B = [b_1, b_2, \ldots b_n]$, then $A + B = [a_1 + b_1, a_2 + b_2, \ldots . a_n + b_n]$.

(c) Scalar Multiplication

Multiplying a scalar (real number) times a vector means multiplying every component by that real number to yield a new vector. For instance, if $\vec{v} = [3, 6, 8, 4]$, then $1.5 * [3, 6, 8, 4] = [4.5, 9, 12, 6]$. More generally, *scalar multiplication* means if d is a real number and $|\vec{v}|$ is a vector $[v_1, v_2, \ldots v_3]$, then

$$d * \vec{v} = \sqrt{dv_1, dv_2, \ldots dv_n} \tag{3.2}$$

(d) Inner Product

The inner product of two vectors also called the dot product or scalar product denies multiplication of vectors. It is found by multiplying each component in $\overrightarrow{v_1}$ by the component in $\overrightarrow{v_2}$in the same position and adding them all together to yield a scalar value. The inner product is only defined for vectors of the same dimension. The inner product of two vectors is denoted $(\overrightarrow{v_1}, \overrightarrow{v_2})$ or $\overrightarrow{v_2} . \overrightarrow{v_2}$ (the dot product). Thus,

$$(\vec{x}, \vec{y}) = \vec{x} * \vec{y} = \sum_{i=1}^{n} x_i y_i \tag{3.3}$$

For example, if $\vec{x} = [1, 6, 7, 4]$ and $\vec{y} = [3, 2, 8, 3]$, then

$$\vec{x} * \vec{y} = 1(3) + 6(2) + 7(8) + 4(3) = 83$$

(e) Orthogonality

Two vectors are orthogonal to each other if their inner product equals zero. In two dimensional space this is equivalent to saying that the vectors are perpendicular, or that the only angle between them is a 90 degree angle. For example, the vectors [2, 1, -2, 4] and [3, -6, 4, 2] are orthogonal because

[2, 1, -2, 4] * [3, -6, 4, 2] = 2(3) + 1(-6) - 2(4) + 4(2) = 0

(f) Normal Vector

A normal vector (or unit vector) is a vector of length 1. Any vector with an initial length >0 can be normalized by dividing each component in it by the vector's length. For example, if $\vec{v} = [2, 4, 1, 2]$, then

$$|\vec{v}| = \sqrt{2^2 + 4^2 + 1^2 + 2^2} = \sqrt{25} = 5$$

Then $\vec{v}$ = [2/5, 4/5, 1/5, 1/5] is a normal vector because

$$|\vec{v}| = \sqrt{(\frac{2}{5})^2 + (\frac{4}{5})^2 + (\frac{1}{5})^2 + (\frac{1}{5})^2} = \sqrt{25/25} = 1$$

3.2.2 Matrix Terminology

Matrix Terminology is mentioned as follows.

(a) Square Matrix

A matrix is said to be square if it has the same number of rows as columns. To designate the size of a square matrix with n rows and columns, it is called n-square. For example, thematrix below is 3-square.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

(b) Transpose Matrix

The transpose of a matrix is created by converting its rows into columns; that is, row 1 becomes column 1, row 2 becomes column 2, etc. The transpose of a matrix is indicated with a superscripted^T, e.g. the transpose of matrix A is A^T. For example, if

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Then its transpose is

$$A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

(c) Matrix Multiplication

It is possible to multiply two matrices only when the second matrix has the same number of rows as the first matrix has columns. The resulting matrix has as many rows as the first matrix and as many columns as the second matrix. In other

words, if A is a m x n matrix and B is a n x s matrix, then the product AB is an m x s matrix.

The coordinates of AB are determined by taking the inner product of each row of A and each column in B. That is, if A_1, … ,A_m are the row vectors of matrix A, and B^1, … , B^s are the column vectors of B, then ab_{ik} of AB equals A_i x B^k. The sample calculation of matrix multiplication is described as follows.

$$A = \begin{bmatrix} 2 & 1 & 4 \\ 1 & 5 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 2 \\ -1 & 4 \\ 1 & 2 \end{bmatrix} \quad AB = \begin{bmatrix} 2 & 1 & 4 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ -1 & 4 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 9 & 16 \\ 0 & 26 \end{bmatrix}$$

$$ab_{11} = \begin{bmatrix} 2 & 1 & 4 \end{bmatrix} \begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix} = 2(3) + 1(-1) + 4(1) = 9$$

$$ab_{12} = \begin{bmatrix} 2 & 1 & 4 \end{bmatrix} \begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix} = 2(4) + 1(4) + 4(2) = 16$$

$$ab_{21} = \begin{bmatrix} 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 2 \end{bmatrix} = 1(3) + 5(-1) + 2(1) = 0$$

$$ab_{22} = \begin{bmatrix} 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 2 \end{bmatrix} = 1(2) + 5(4) + 2(2) = 26$$

(d) Identity Matrix

The identity matrix is a square matrix with entries on the diagonal equal to 1 and all other entries equal zero. The diagonal is all the entries a_{ij} where $_i = _j$, i.e., a_{11}, a_{12}, …, a_{mm}. The n-square identity matrix is denoted variously as I_{n*n}, I_n, or simply I. The identity matrix behaves like the number 1 in ordinary multiplication, which mean AI = A, as the example below shows.

$$A = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix} \quad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad AI = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$ai_{11} = \begin{bmatrix} 2 & 4 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 2(1) + 0(4) + 0(6) = 2$$

$$ai_{12} = \begin{bmatrix} 2 & 4 & 6 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = 2(0) + 4(1) + 6(0) = 4$$

$$ai_{13} = \begin{bmatrix} 2 & 4 & 6 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 2(0) + 4(0) + 6(1) = 6$$

$$ai_{21} = \begin{bmatrix} 1 & 3 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 1(1) + 3(0) + 5(0) = 1$$

$$ai_{22} = [1 \quad 3 \quad 5]\begin{bmatrix}0\\1\\0\end{bmatrix} = 1(0) + 3(1) + 5(0) = 3$$

$$ai_{23} = [1 \quad 3 \quad 5]\begin{bmatrix}0\\0\\1\end{bmatrix} = 1(0) + 3(0) + 5(1) = 5$$

$$= \begin{bmatrix}2 & 4 & 6\\1 & 3 & 5\end{bmatrix}$$

(e) Orthogonal Matrix

A matrix A is orthogonal if $AA^T = A^TA = I$. For example,

$$A = \begin{bmatrix}1 & 0 & 0\\0 & 3/5 & -4/5\\0 & 4/5 & 3/5\end{bmatrix}$$

is orthogonal because

$$A^TA = \begin{bmatrix}1 & 0 & 0\\0 & 3/5 & -4/5\\0 & 4/5 & 3/5\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\0 & 3/5 & -4/5\\0 & 4/5 & 3/5\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$$

(f) Eigenvectors and Eigenvalues

An eigenvector is a nonzero vector that satisfies the equation

$$A\vec{v} = \lambda\vec{v} \tag{3.4}$$

where A is a square matrix, ¸ λ is a scalar, and $\vec{v}$ is the eigenvector. λ is called an eigenvalue. Eigenvalues and eigenvectors are also known as, respectively, characteristic roots and characteristic vectors, or latent roots and latent vectors.

You can find eigenvalues and eigenvectors by treating a matrix as a system of linear equations and solving for the values of the variables that make up the components of the eigenvector. For example, finding the eigenvalues and corresponding eigenvectors of the matrix

$$A = \begin{bmatrix}2 & 1\\1 & 2\end{bmatrix}$$

means applying the above formula to get

$$A\vec{v} = \lambda\vec{v} = \begin{bmatrix}2 & 1\\1 & 2\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \lambda\begin{bmatrix}x_1\\x_2\end{bmatrix}$$

in order to solve for λ, x_1 and x_2. This statement is equivalent to the system of equations

$$2x_1 + x_2 = \lambda x_1$$

$$x_1 + 2x_2 = \lambda x_2$$

which can be rearranged as

$$(2 - \lambda)\, x_1 + x_2 = 0$$

$$x_1 + (2 - \lambda)\, 2x_2 = 0$$

A necessary an sufficient condition for this system to have a nonzero vector $[x_1, x_2]$ is that the determinant of the coefficient matrix

$$\begin{bmatrix} (2-\lambda) & 1 \\ 1 & (2-\lambda) \end{bmatrix}$$

be equal to zero. Accordingly,

$$\begin{vmatrix} (2-\lambda) & 1 \\ 1 & (2-\lambda) \end{vmatrix} = 0$$

$$(2 - \lambda)\,(2-\lambda) - 1*1 = 0$$

$$\lambda^2 - 4\lambda + 3 = 0$$

$$(\lambda - 3)\,(\lambda - 1) = 0$$

There are two values of λ that satisfy the last equation; thus there are two eigenvalues of the original matrix A and there are $\lambda_1 = 3$, $\lambda_2 = 1$.

We can find eigenvalues which correspond to these eigenvalues by plugging λ back in to the equations above and solving for x_1 andx_2. To find an eigenvetor corresponding to $\lambda = 3$, start with

$$(2 - \lambda)\, x_1 + x_2 = 0$$

and substitute to get

$$(2 - 3)\, x_1 + x_2 = 0$$

Which reduces and rearranges to

$$x_1 = x_2$$

There is an infinite number of values for x_1 which satisfy this equation; the only restriction is that not all the components in an eigenvector can equal zero. So if $x_1 = 1$, then $x_2 = 1$ and an eigenvector corresponding to $\lambda = 3$ is [1, 1].

$$(2 - 1)\, x_1 + x_2 = 0$$

$$x_1 = -x_2$$

So an eigenvector for $\lambda = 1$ is [1, -1].

3.3 Similarity Measure

Another key factor in the success of the proposed system is the similarity measure between testApp and malware apps. There are three simple and well known similarity measures to calculate the similarity. They are the Dice, Jaccard and Cosine Coefficients. Among these three similarity measures, the system is used Jaccard similarity to measure the related permissions patterns in testApp and set of trained malware apps.

Jaccard Similarity Coefficient: The Jaccard index, also known as the Jaccard similarity coefficient (by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

Let D = {D1, D2,……Dn} be the collection of N malware apps. Each malware app Di can be represented by a corresponding set Si such that Si is a set of all the permissions contained in Di. Let us denote that set by Di such that Di={S1,S2,…….. Sn} [1].

Some attributes are present in just a few objects of a data set. As they assume zero values in most of the cases, they are called asymmetric. Jaccard Similarity Coefficient measure is used to handle asymmetric binary attributes as only non-zero values are relevant for the calculation [7].

$$\frac{\sum_{i=1}^{|v|} \mathrm{Per_{iApp}} * \mathrm{Per_{iqueryApp}}}{\sum_{i=1}^{|v|} (\mathrm{Per_{iApp}})^2 + \sum_{i=1}^{|v|} (\mathrm{Per_{iqueryApp}})^2 - \sum_{i=1}^{|v|} \mathrm{Per_{iApp}} * \mathrm{Per_{iqueryApp}}} \qquad (3.5)$$

Where: Per_{App} = permissions of trained application

$Per_{queryApp}$ = permissions of user chosen application

3.4 Measuring System Effectiveness

The proposed system uses the statistical method SVD and Jaccard Coefficient similarity. The system can be used to detect whether the incoming app should be installed or not on Android Smartphone.

Firstly, the user wants to install a specific app. That is passed to the pre-processing stages. In database, all significant malware apps are already pre-processing and calculated the malware related patterns. The input app is compared to this malware pattern. Finally, the system displays risky level according to the similarity values.

Accuracy: Accuracy of a system is evaluated on how well the system is able to distinguish anomalous app or not.

$$\text{Accuracy} = \frac{tp+tn}{tp+fp+tn+fn} \tag{3.6}$$

Where:

tp (true positive) is an outcome where the system correctly predicts the malware app.

tn (true negative) is an outcome where the system correctly predicts the godware app.

fp (false positive) is an outcome where the system incorrectly predicts the malware app.

fn (false negative) is an outcome where the system incorrectly predicts the goodware app.

CHAPTER 4

THE PROPOSED SYSTEM IMPLEMENTATION

Since the proposed system is implemented by using vector space model, preprocessing stages and indexing are needed. Therefore, query and documents can easily and quickly be compared. The result documents are shown in decreasing order of similarity to the query term. The system uses vector space information retrieval model and Jaccard Coefficient for similarity ranking. As the non-function requirements, a computer which has at least Intel@ Core i5-2410M, 640 GB HDD and 8 GB DDR3 Memory are required to implement our proposed system. As the functional requirements, we need to install Android Studio, an oreo-versioned mobile phone to run and test our proposed system.

4.1 Brief Overview of the Proposed System

The proposed system is a malicious application detection system based on permission information from Manifest file. The system uses static analysis for malware detection which means that applications are not executed or analyzed at run time. The proposed system comprises into two groups as follows.

For training phase,

1. Extract the permissions from JSON files of ApkMetaReport folder for static analysis of android malware 2017.(https://www.kaggle.com/goorax/static-analysis-of-android-malware-of-2017)
2. Preprocess the extracted permissions such as redundant permissions removal.
3. Perform statistical SVD approach.

For testing phase,

1. Extract the permissions from the tested App.
2. Find the permission-correlation pattern (T) of it.
3. Find the similarity value between trained-permission patterns and T.

4.2 Training Phase of the Proposed System

4.2.1 Data Collection

To implement the proposed system, the first thing is to collect the information about the risky apps as much as. According to the literature, there are so many ways to analyze different kinds of apps such as by analyzing signature features, behavior features or anomaly features and so on. Among them, the proposed system analyzes the apps based on permissions. Because permission is the main gate to allow the application (which operations must be done). So, this is the fact to learn about permissions of android application.

There are a lot of permissions that are declared by Google. Moreover, there are also customized permissions. The specific permission has its own task such as reading contacts, or sending sms or getting GPS, etc. Some of them are dangerous. Some of them are normal. Some of them are nothing meaning etc. But when analyzing permissions, it isn't enough to know which permissions are dangerous and which permissions are normal. One application can use as much as permissions according to the developer. And, it cannot be concluded that an application has high risk by seeing one of dangerous permissions.

So, it is needed to analyze which correlation patterns of permissions are usually involved in high risk application. Singular Value Decomposition (SVD) technique is applied to get the correlation patterns of permissions. To apply SVD technique, the original matrix (permission-app matrix) is needed to get. For choosing the training dataset, malware dataset is needed to train since the propose of the system is to give the knowledge that how much risk level has an incoming application. Malware dataset didn't download easily as malware based dataset are very restricted.

The required dataset is obtained from https://www.kaggle.com/goorax/static-analysis-of-android-malware-of-2017. Kaggle website describes the specific analysis results of malware applications by separating into four folders. These folders are apkMetaReport, byteCodeReport, virusTotalReport, and assestReport. apkMetaReport folder contains the contents of Manifest.xml files. byteCodeReport folder contains the contents of classes.dex. virusTotalReport folder contains the reports of virusTotal service. assestReport folders contains names of assests and lib contents. So, apkMetaReport folder is downloaded. That dataset contains over 4000 json files (one

Json file for one malware application). An android app name is identified by its sha256 hash sum, which is used by file name.

4.2.2 Dataset Description

The dataset is obtained from https://www.kaggle.com/goorax/static-analysis-of-android-malware-of-2017. For static analysis of android malware 2017, this dataset contains 4000 JSON files. The JSON files contains the method names. An Android app name is uniquely identified by its sha256 hash sum, which is used as the file name. The following folders store specific analysis results:

1. ApkMetaReport: Contents of the AndroidManifest.xml.
2. ByteCodeReport: Contents of the classes.dex.
3. VirusTotalReport: Report of the Virustotal service.
4. AssetReport: Names of assets and lib contents.

Among them, the proposed system uses ApkMetaReport file. The analyzed malware was originated at Technical University Berlin. It is a part of the Virusshare repository. The static analysis extracted information from the AndroidManifest.xml.

4.2.3 Preprocessing

We need to preprocess the downloaded dataset to be ready to use as the trained dataset in our proposed system. There are two steps for preprocessing phase: tokenization and removing Duplicate Permissions.

Tokenization: Computers do not understand the structure of a natural language document and cannot automatically recognize words and sentences. So, humans must program the computer to identify what constitutes and individual or distinct word referred to as a token. Such a program is commonly called a tokenizer or parser or lexer. Tokenizing is the process of breaking of stream of text up into words, phrases, symbols or other meaningful elements.

To store the permissions for each application, we extract the required permissions from the json dataset by tokenization. Then we build original matrix (permission-app matrix).

Removing Duplicate Permissions: Some permission is frequently occurring and that do not represent any content of the application. Duplicate permissions are list of permissions that the developer includes them unintentionally. So, in this phase, duplicate permissions are removed before building the original malware vector.

4.2.4 Implementation Steps for the Training Phase

The implementation steps for the training phase are as follows.

Step 1 Place JSON files under 'download' folder of emulator's internal storage.

Step 2 For each JSON file,

Extract Permissions and do Preprocessing phase

Create a Permission_App relation (perID and appID)

Step 3 Generate the Boolean permission_app matrix and save it to database

Step 4 Compute S, V, U metrics (using Singular Value Decomposition) and reduce the metrics with k dimension (suppose: k=4).

Step 5 Save S^{-1} and U metrics to the corresponding data files

Step 6 Transpose V (malApp vector) and save it to the corresponding data file

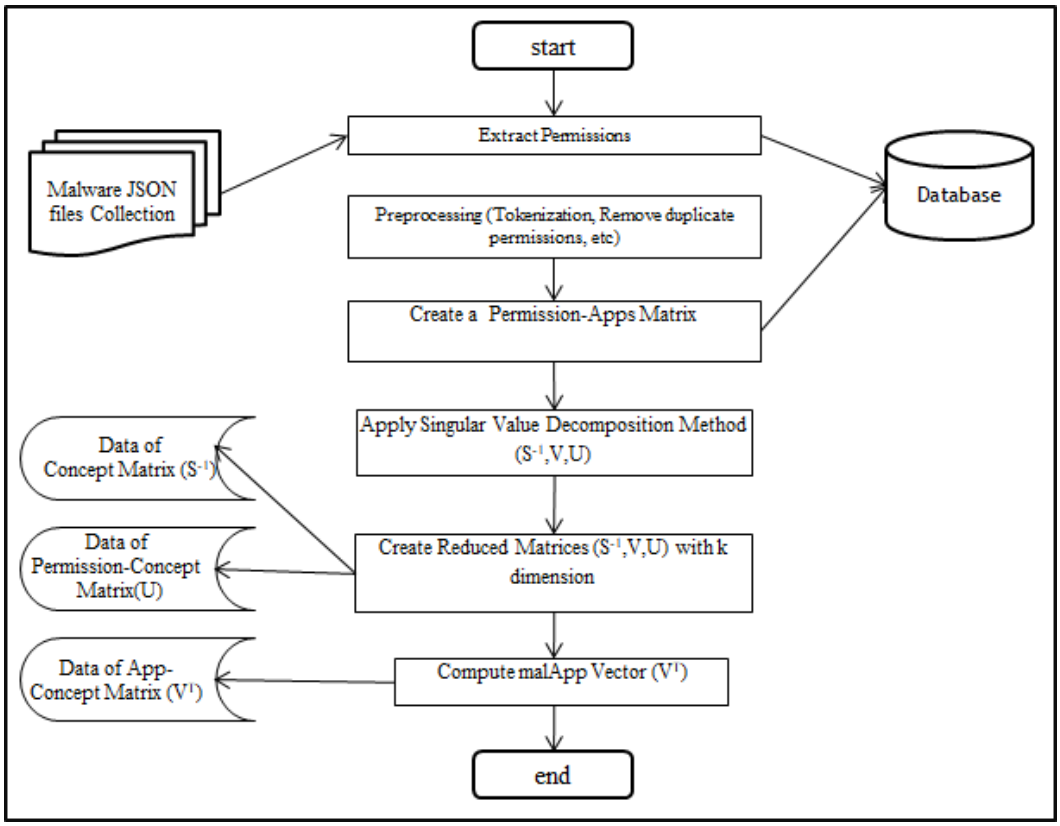

Figure 4.1 Process Flow Diagram for Training Phase

4.3 Testing Phase of the Proposed System

Testing phase contains two sub-phases. Figure 4.2 (a) is about finding the permission-correlation pattern of the user chosen application and the steps are as follow.

Step 1 Accept a test app.

Get Permission Values of <uses-permission> ElementfromAndroid's Manifest.xml (by getRequestedPermissionList() method)

Step 2 Generate testApp vector (q).

Step 3 Calculate queryApp vector by computing q^T U S^{-1}

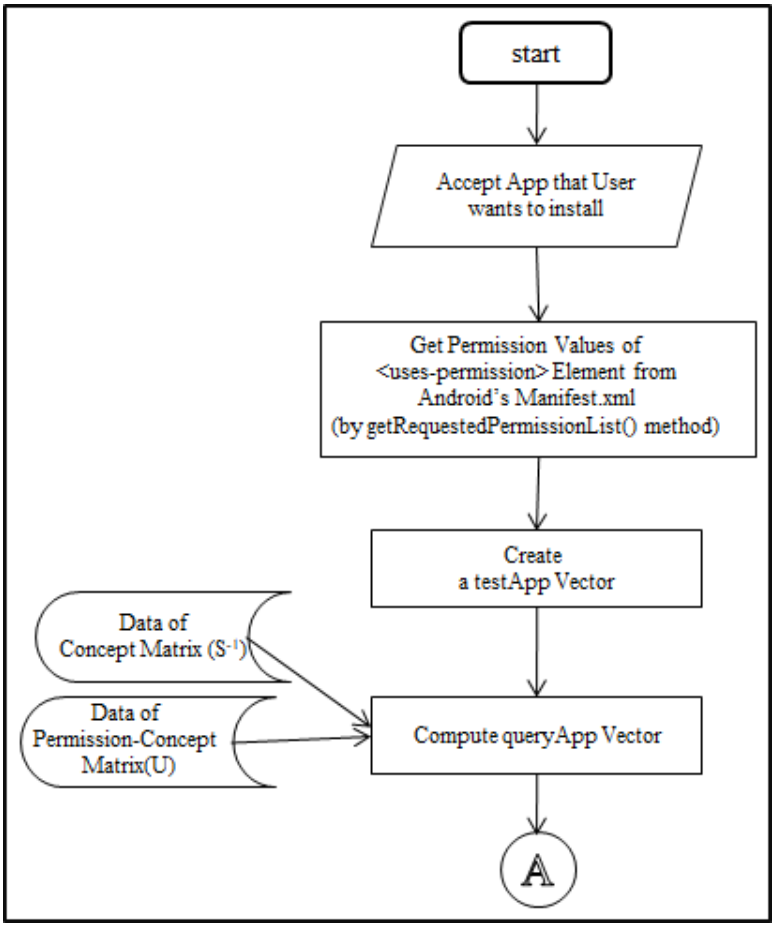

Figure 4.2 (a) Process Flow Diagram for creating queryApp Vector

The figure 4.2 (b) is about giving information to the user for the user chosen application's risk level. The steps of creating queryApp vector are as follows.

Step 1 Fetch V^T from corresponding data file.

Step 2 Compute the similarity values between queryApp vector and malApp vectors (V^T) and save them to temporary similarity result array.

Step 3 Choose the highest similarity value (h) from the similarity result array.

Step 4 If h is greater than or equal to maximum threshold value (0.8000), show the message “The application has high risk permissions”.

Step 5 Else if h is greater than or equal to minimum threshold value (0.5847), show the message “The application has medium risky permissions”. Otherwise, show the message “The application has low risk permissions”.

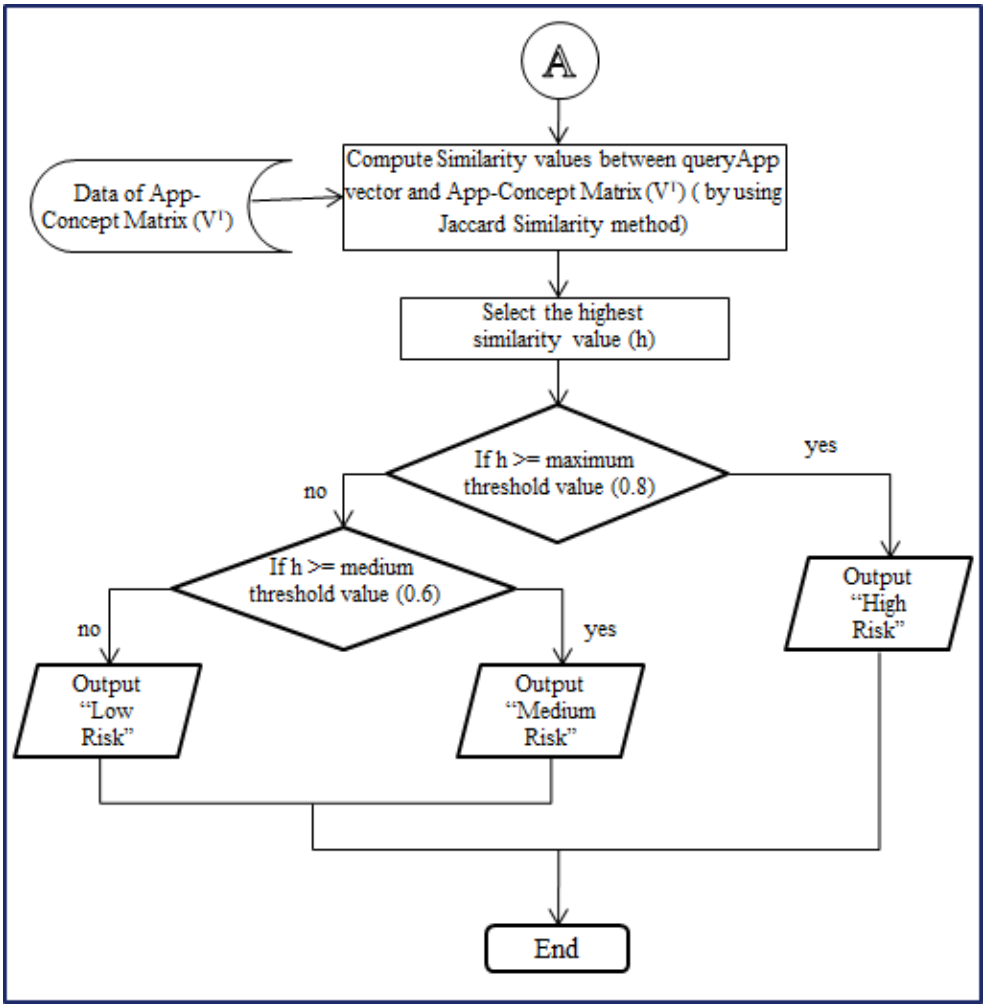

Figure 4.2 (b) Process Flow Diagram for Finding Risk Level

4.4 Database Design of the Proposed System

Our proposed system needs to use the following three tables.

1. PermissionAppMatrix table,
2. ApplicationData table and
3. PermissionData table

PermissionAppMatrix table stores the relationship of the permissions and applications as element field (1 means exist, and 0 means not exist). That table has composite primary key to join with both of PermissionData table and ApplicationData table. ApplicationData table is used to store the general information of the applications with appName for application's name and appIcon for application's icon. PermissionData table is used to store the permission name (such as android.permission.INTERNET).

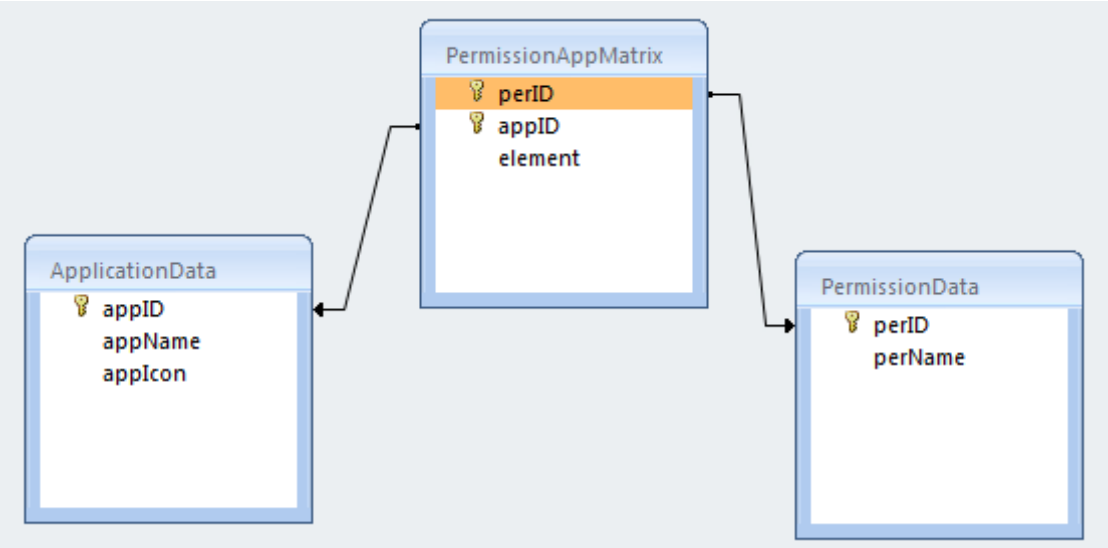

Figure 4.3 Database Design of the Proposed System

And the resulted matric are needed to save as files after applying Singular Value Decomposition (SVD) method to use later in calculation of testApp vector.

4.5 Analysis and Empirical Result

There is over 4000 malware dataset as described in Section 4.2. But the system cannot be trained with all of that according to phone storage and emulator performance. Firstly, the proposed system is trained with data beginning from 50 dataset by adding 50 JSON files again and again to the existing dataset. The emulator was hung at trained dataset 200 on 4G RAM. So, the proposed system is trained dataset beginning from 250 dataset on a laptop which has 8G RAM. It took a lot of time to train that amount of dataset. According to emulator's performance and mobile phone storage, 300 dataset is more suitable on the current situation.

At that time, there was another problem that is which 300 dataset will be trained among these 4000 malware apps. So, 4000 dataset was separated by 300. And

the proposed system is trained with different 300 datasets to know which dataset has the more features of current environment malware apps by testing 95 malware apps. According to the following Figure 4.5, D4 dataset includes the more features of the current environment malware apps. So, D4 dataset is chosen to train on the proposed system.

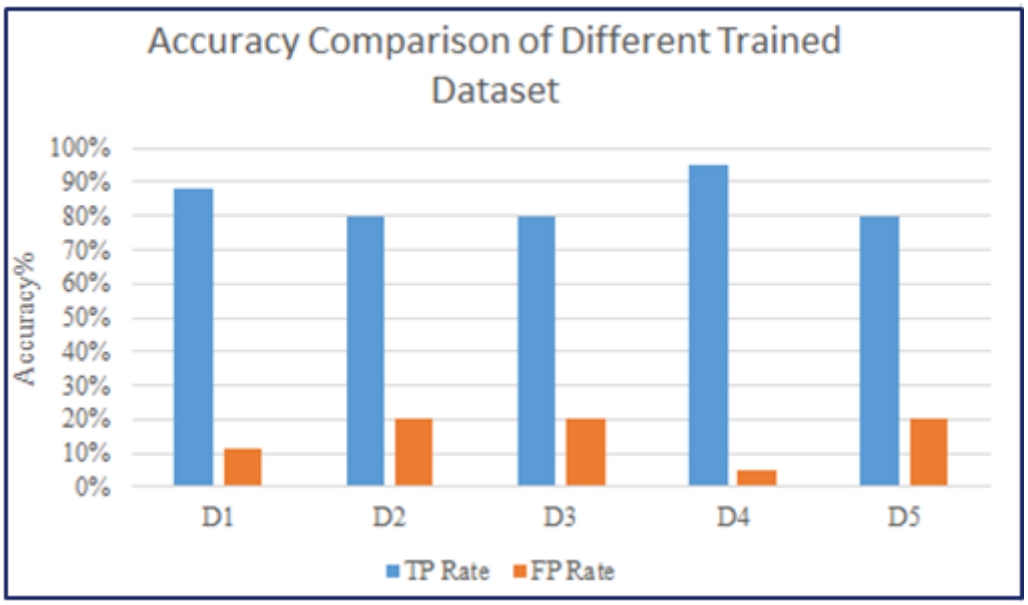

Figure 4.4 Accuracy Comparison of Different Trained Dataset

In figure 4.4, D1, D2, D3, D4 and D5 are different Malware Datasets contained 300 JSON files from ApkMetaReport Malware dataset. The system is trained with all of 300 separated dataset of 4000 malware apps. But at that figure 4.4, only the appearance datasets are highlighted.

After getting the best trained dataset, we have to found out which k value will be the best on our trained data according to SVD method. So, we analyze different k value on our trained dataset.

According to the figure 4.5, k=4 is the best value of the others. This figure shows the overall accuracy of the proposed system at different k value. But only k=2 to k=5 are highlighted among a lot of different k value.

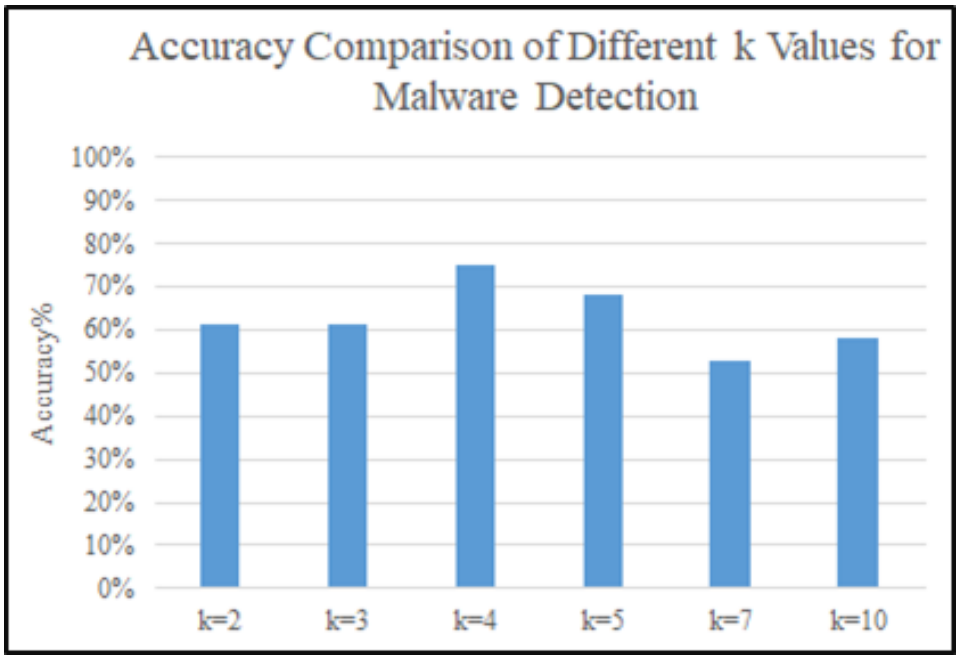

Figure 4.5 Accuracy Comparison of Different k value for Malware Detection

In figure 4.6, the correctness of malware is 100% at k=2. But the correctness of goodware is too low. At k=3, also like k=2. At k=4, the correctness of malware decreases a little, but the correctness of goodware is significantly high. So, the accuracy of the system also increases significantly than others. At later k value, the correctness of malware is lower.

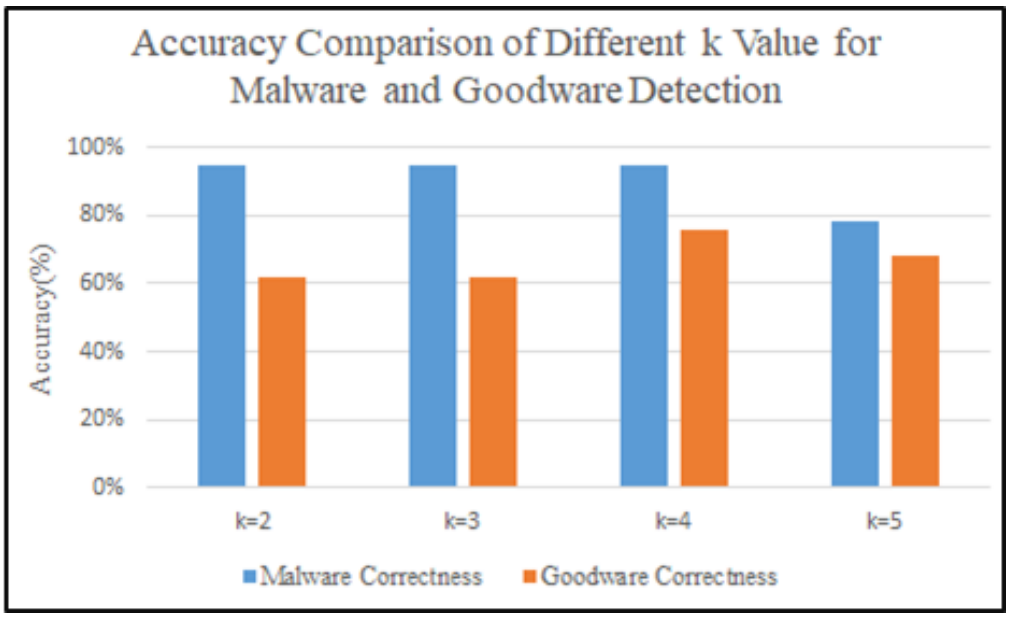

Figure 4.6 Accuracy Comparison of Different k value for Malware and Goodware Detection

Finally, the proposed system is implemented on trained dataset (300 JSON files) which has the most suitable malware apps of the current environment with k=4 according to our analysis of figure (4.7).

No: of Test Data	TP	TN	FP	FN	Acccuracy
120	83	19	5	13	85%

Table 4.1 Accuracy for the Proposed System

And the final accuracy is shown in table 4.1. The accuracy of proposed system is 85% on the tested data 120 including malware and goodware.

4.4 Screen Transactions of the Proposed System

Figure 4.7 shows the screen for retrieving original permission-app matrix. On the other hand, that screen design shows the relationships between trained malware applications and permissions.

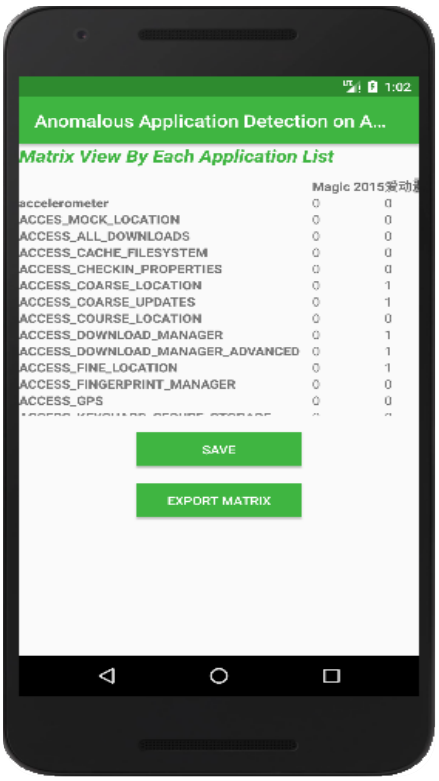

Figure 4.7 Screen Design for Retrieving Original permission-app Matrix

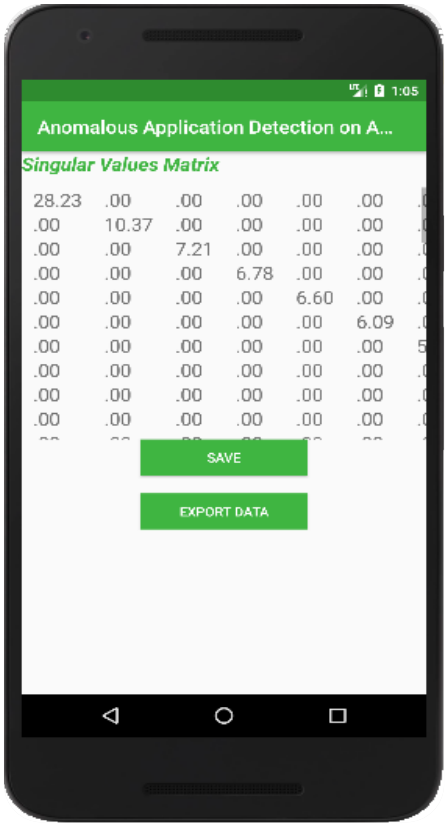

Figure 4.8 Screen Design for Retrieving Singular Value Matrix

Figure 4.8 describes the screen for retrieving singular value matrix. That matrix is the concept matrix of eigenvalue matrix and U matrix which are gotten after applying Singular Value Decomposition (SVD) method.

Figure 4.9 mentions the screen for retrieving eigenvalue matrix. That matrix is the relationship of trained malware applications and the concept matrix (singular value matrix) that are gotten after applying Singular Value Decomposition (SVD).

Figure 4.10 shows the screen for retrieving U matrix. That matrix is the relationship of permission of each trained malware application and the concept matrix (singular value matrix) that are gotten after applying Singular Value Decomposition (SVD).

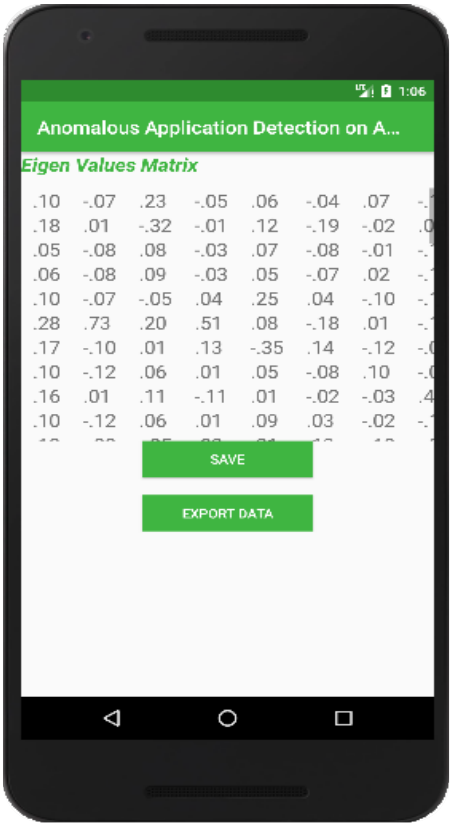

Figure 4.9 Screen Design for Retrieving Eigen Value Matrix

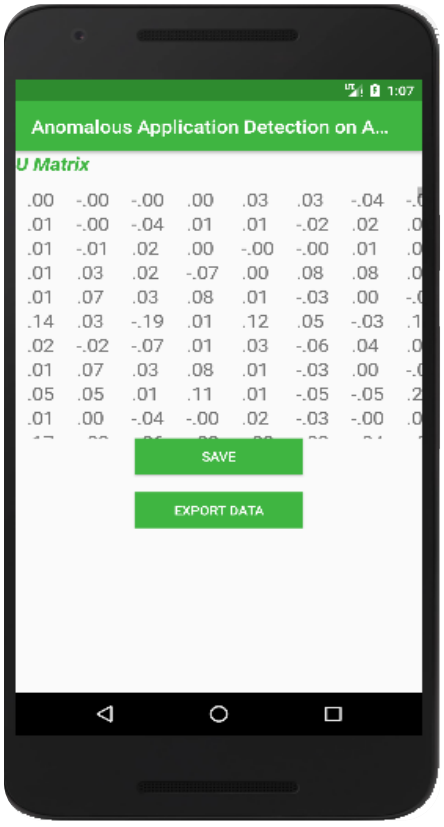

Figure 4.10 Screen Design for Retrieving U Matrix

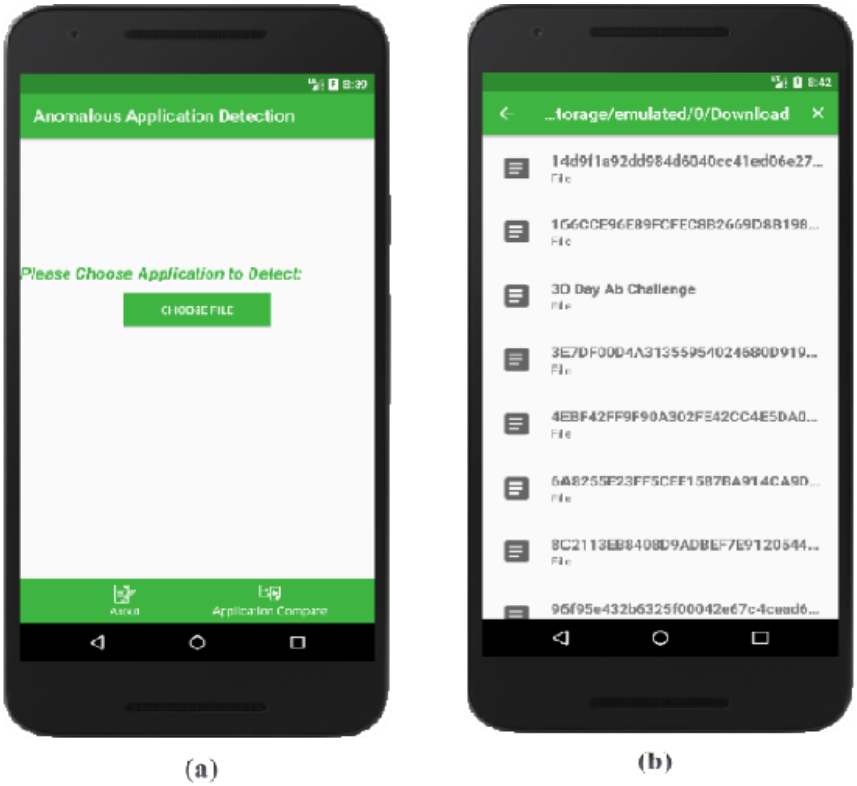

Figure 4.11 (a+b) Screen Design for Choosing Application to detect

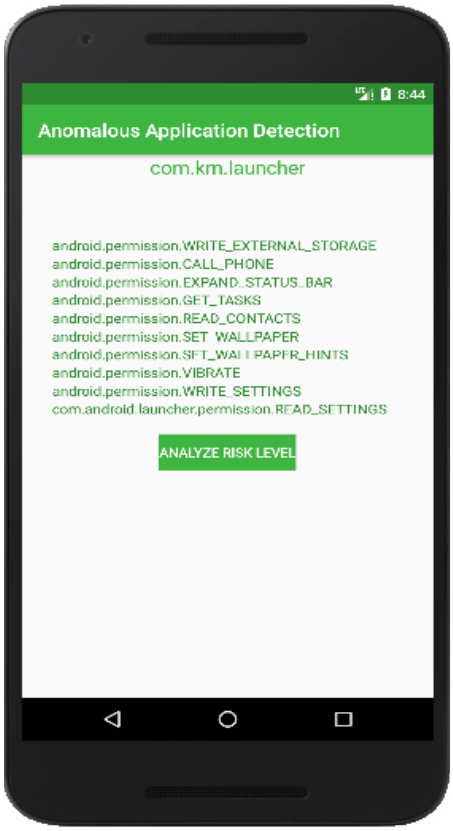

Figure 4.12 Screen Design for showing permission of chosen app

Figure 4.11 explains the testing screen design for choosing application to detect. Users need to browse an apk file under Download folder before installing it.

Figure 4.12 shows the screen after browsing the user chosen application. That screen shows the permissions which are used in the user chosen application.

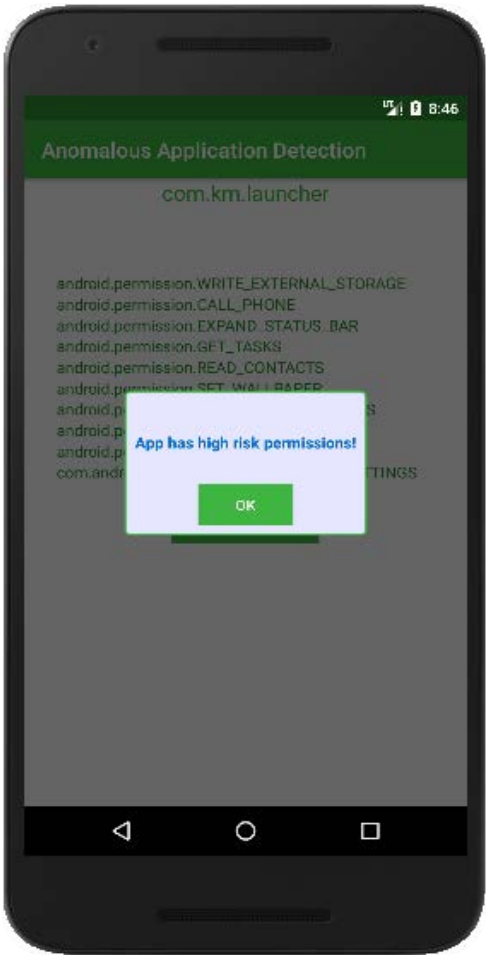

Figure 4.13 Screen Design for showing the similarity result when pressing the analyze risk level button

Figure 4.13 mentions the screen for retrieving the message about the risk level of user chosen application. User can know the risk level of the chosen application before installing it by using our proposed system.

CHAPTER 5
CONCLUSION

Focus of attackers and malware writers has changed to mobile devices due to the increased adoption of mobile devices for business and personal purposes and comparatively lesser security controls. Therefore, App stores are common targets for attackers to distribute malware and malicious apps. The system proposes to detect the risk level for anomalous Android applications. The malware dataset is identified using Singular Value Decomposition (SVD) based approach where a permission-malapp matrix needs to be developed and then query-app can be detected from the set of risky permissions. However, the growing amount and diversity of Android malware have significantly weakened the effectiveness of the conventional defense mechanisms, and thus, Android platform often remains unprotected from new and unknown malware [9].

The proposed system suggests that the implementation is well suited by finding Jaccard Similarity Values between existing malwares and the user query apps. As the conclusion, the Jaccard Similarity measures are well suited for mediate amount of data set and can effectively be helpful in finding similar values between user query apps and malwares. So, the system enables users to search similar risky apps as efficiently and as fast as possible. Therefore, the system can save time in finding the risky apps even the users didn't know which apps are closely related and can access the system effectively without an internet connection.

5.1 Advantages of the System

In the malware detection system, the nature of malware permissions' signatures is important. The statistical method, Singular Value Decomposition (SVD) can find the correlation patterns of malware permissions' signatures involved in most malware applications. Therefore, the proposed system methodology is effective to detect android malwares. Jaccard Coefficient is more effective to calculate the similarity on the data objects that have binary attributes. And the proposed system's trained dataset uses binary attributes for the relation of permission and application. So, finding the similarity value between the permissions of user chosen application and each trained application by using Jaccard Coefficient makes the system more effective.

And the system leads to know the permission risk level of user chosen application even if user doesn't know the permissions in details. The system is very effective for permission only based detection as the used method is computationally efficient in finding the correlation patterns of malware's permission nature.

5.2 Limitations of the System

There are some limitations in the proposed system. The system cannot grantee advanced obfuscation techniques such as polymorphic and metamorphic malware. Alternatively, it is not significant at detecting the disguised malware as the goodware since the disguised malware may use many goodware permissions as much as they can.

5.3 Further Extension

There are a number of directions for further extension. The importance of mobile phones in our everyday life and many activities is undeniably unending. Therefore, this system could also be implemented on not only android OS but also IOS. Moreover, the proposed system should be implemented by considering additional features (signatures) to improve the capabilities and efficiency. On the other hand, malware classification system can be extended in addition to malware detection system by using specific classification method.

AUTHOR'S PUBLICATION

[1] Htet Htet Win, Zon Nyein Nway, "*Permission-Based Anomalous Application Detection on Android Smart Phone*", the Proceedings of the 9th Conference on Parallel and Soft Computing (PSC 2018), Yangon, Myanmar, 2018.

REFERENCES

[1] A. Marcus and J. Maletic, “Using Latent Semantic Analysis to Identify Similarities in Source Code to Support Program Understanding”, Proc. of 12th IEEE International Conference on Tools with Artificial Intelligence, November 2000, pp.46-53.

[2] C. Hein, “Manifest Files Classification of Android Malware”, the 8th International Conference on Application of Information and Communication Technologies, Mandalay, Myanmar, December 2014, pp.119-133.

[3] D. Kalman, “A Singular Value Decomposition: The SVD of a Matrix”, The American University, Washington, DC 20016, February13, 2002.

[4] F. Tchakounte, “Permission-based Malware Detection Mechanisms on Android: Analysis and Perspectives”, Journal of Computer Science and Software Application, December 2014, pp.63-77.

[5] G.N. Bharathi, T. Anusha, R.S. MeenaKumari, P. Aparna, “Effective Permission Analysis and Complete Security for Android Application”, SSRG International Journal of Computer Science and Engineering, March 2017, pp.97-104.

[6] H. Shahriar, V. Vlincy, “Anomalous Android Application Detection with Latent Semantic Indexing”, Conference of 40th Annual Computer Software and Applications, 2016, pp.624-625.

[7] I. Toure, A. Gangopadjyay, “Analyzing Terror Attacks using Latent Semantic Indexing”, ISBN-978-1-4799-1535-4, 2013, pp.334-337.

[8] K. Baker, “Singular Value Decomposition Tutorial”, March 29, 2005.

[9] K. Dunham, S. Hartman, J. Morales, M. Quintans and T. Strazzere, “Android_Malware_and_Analysis”, ISBN-13: 978-1-4822-5220-0, 2015.

[10] L. Jin, S. Lichao, Y. Qiben, L. Zhiqiang, S.A. Witawas, Y. Heng, “Significant Permission Identification for Machine Learning Based Android Malware Detection”, IEEE Transactions on Industrial Informatics, 2017.

[11] M.A. Siddiqui, “Data Mining Methods for Malware Detection”, Modeling and Simulation in the College of Sciences, Orlando, Florida, 2008.

[12] N.V. Duc, P.T. Giang, P.M. Vi, “Permission Analysis for Android Malware Detection”, Proc. of the 7th Vast – AIST Workshop “Research Collaboration: Preivew and Perspective”, At Hanoi, Vietnam, November 2015, pp.207-216.

[13] R.K. Jidigam, “Metamorphic Detection Using Singular Value Decomposition”, Master's Projects, San Jose State University, December 2013.

[14] T.S. Barhoom, M.I. Nasam, “Malware Detection Based on Permissions on Android Platform Using Data Mining”, Journal of Engineering Research and Technology, Volume 3, Issue 3, September 2016, pp.51-57.

[15] V.N. Cooper, H. Shahriar, H.M. Haddad, “A Survey of Android Malware Characteristics and Mitigation Techniques”, Proc. of the 11th International Conference on Information Technology: New Generations, IEEE CPS, Las Vegas, USA, April 2014, pp.327-332.

[16] Z. Aung, W. Zaw, “Permission-Based Android Malware Detection”, Internal Journal of Scientific and Technology Research Volume 2, Issue 3, March 2013, pp.228-234.

APPENDIX

Sample Calculation for the Proposed System

	App1	App2	App3	testApp
Access_Network_State	1	1	1	1
Access_Wifi_State	1	1	1	1
Internet	1	1	1	1
Read_Phone_State	1	1	1	1
Write_External_Storage	1	1	1	1
Mount_Unmount_Filesystems	1	1	0	0
Read_External_Storage	1	0	0	1
Install_Shortcut	1	1	1	0
Uninstall_Shortcut	1	0	0	0
Read_Settings	1	1	0	0
Receive_Boot_Completed	1	0	0	1
Get_Tasks	1	0	1	0
System_Alert_Window	1	0	1	0
Wake_Lock	1	0	1	1
Get_Accounts	1	0	0	0
Raised_Thread_Priority	0	1	0	0
Write_Secure_Settings	0	1	0	0
Write_Settings	0	1	0	0
Change_Network_State	0	1	0	1
Receive_MMS	0	1	0	0
Receive_Wap_Push	0	1	0	0

Read_SMS	0	1	0	1
Send_SMS	0	1	0	1
Receive_SMS	0	1	0	1

$$A^TA = \begin{bmatrix} 15 & 8 & 9 \\ 8 & 17 & 6 \\ 9 & 6 & 9 \end{bmatrix}$$

By $A^TA - \lambda I = 0$

$$\begin{bmatrix} 15 & 8 & 9 \\ 8 & 17 & 6 \\ 9 & 6 & 9 \end{bmatrix} - \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} = 0$$

$$\begin{bmatrix} 15-\lambda & 8 & 9 \\ 8 & 17-\lambda & 6 \\ 9 & 6 & 9-\lambda \end{bmatrix} = 0$$

$$(-1)^{1+1}(15-\lambda)\begin{bmatrix} 17-\lambda & 6 \\ 6 & 9-\lambda \end{bmatrix}+(-1)^{1+2}(8)\begin{bmatrix} 8 & 6 \\ 9 & 9-\lambda \end{bmatrix}+(-1)^{1+3}(9)\begin{bmatrix} 8 & 17-\lambda \\ 9 & 6 \end{bmatrix}=0$$

$$(15-\lambda)\left[(17-\lambda)(9-\lambda)-36\right]-8\left[8(9-\lambda)-54\right]+9\left[48-9(17-\lambda)\right]=0$$

$$(15-\lambda)[153-17\lambda-9\lambda+\lambda^2-36]-8[72-8\lambda-54]+9[48-153+9\lambda]=0$$

$$(15-\lambda)(\lambda^2-26\lambda+117)-8(18-8\lambda)+9(9\lambda-105)=0$$

$$-\lambda^3+41\lambda^2-362\lambda+666=0$$

‘λ_1=29.4907

‘λ_2=9

‘λ_3=2.5093

$S_1=\sqrt{\lambda_1}$=5.4305

$S_2=\sqrt{\lambda_2}$=3

$S_3=\sqrt{\lambda_3}$=1.5841

$$S=\begin{bmatrix} 5.4305 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1.5841 \end{bmatrix}$$

$$S^{-1}=\begin{bmatrix} 0.1842 & 0 & 0 \\ 0 & 0.3333 & 0 \\ 0 & 0 & 0.6313 \end{bmatrix}$$

By $(A^TA - \lambda I)\,\vec{v}=\vec{0}$

Let $\lambda_1=29.4907$

$$\begin{bmatrix} 15 & 8 & 9 \\ 8 & 17 & 6 \\ 9 & 6 & 9 \end{bmatrix} - \begin{bmatrix} 29.4307 & 0 & 0 \\ 0 & 29.4307 & 0 \\ 0 & 0 & 29.4307 \end{bmatrix} \vec{v}=\vec{0}$$

$$\begin{bmatrix} -14.4907 & 8 & 9 \\ 8 & -12.4907 & 6 \\ 9 & 6 & -20.4907 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$-14.4907x_1+8x_2+9x_3 = 0$ …….... Eq:1

$8x_1-12.4907x_2+6x_3 = 0$ …….... Eq:2

$9x_1+6x_2-20.4907x_3 = 0$ …….... Eq:3

By Eq:1/9-Eq:2/6,

$-1.6101x_1 + 0.8889x_2 + x_3 = 0$

$1.3333x_1 - 2.0818x_2 + x_3 = 0$

‘+ ‘+ ‘-

$-2.9434x_1 + 2.9707x_2 = 0$ …….. Eq:4

By Eq:2/8-Eq:3/9,

x_1 $-1.5613x_2$ $+0.75x_3$ = 0

x_1 $+0.6667x_2$ $-2.2767x_3$ = 0

‘- ‘- ‘+

$-2.228x_2$ $+3.0267x_3$ = 0 …….. Eq:5

By Eq:4/2.9707+Eq:5/2.228,

$0.9908x_1$ $+x_2$ = 0

$-x_2$ $+1.3585x_3$ = 0

$0.9908x_1$ $+1.3585x_3$ = 0

Let x_1=1,

$1.3585x_3$ =-0.9908

‘x_3=-0.7293

In Eq:4, $2.9707x_2$ =2.9434

x_2 =0.9908

‘v= $\begin{bmatrix} 1 \\ 0.9908 \\ -0.7293 \end{bmatrix}$

Length, L = $\sqrt{(1)^2 + (0.9908)^2 + (-0.7293)^2}$ =1.5854

Normalized vector, v_1= $\begin{bmatrix} 0.6308 \\ 0.625 \\ -0.46 \end{bmatrix}$

Let $\lambda_2 = 9$

$$\begin{bmatrix} 6 & 8 & 9 \\ 8 & 8 & 6 \\ 9 & 6 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$6x_1+8x_2+9x_3 = 0$ …….. Eq:1

$8x_1+8x_2+6x_3 = 0$ …….. Eq:2

$9x_1+6x_2 = 0$ …….. Eq:3

By Eq:1-Eq:2,

$6x_1 +8x_2 +9x_3 = 0$

$8x_1 +8x_2 +6x_3 = 0$

‘- ‘- ‘-

$-2x_1 +3x_2 = 0$ …….. Eq:4

Let $x_1 =1$,

$3x_3 =2$

$x_3 =0.6667$

In Eq:3,

$6x_2 =-9$

‘$x_2 =-1.5$

$$\text{‘v} = \begin{bmatrix} 1 \\ -1.5 \\ 0.6667 \end{bmatrix}$$

Length, $L = \sqrt{(1)^2 + (-1.5)^2 + (0.6667)^2} = 1.9221$

Normalized vector, $v_2 = \begin{bmatrix} 0.5203 \\ -0.7804 \\ 0.3469 \end{bmatrix}$

Let λ_3=2.5093,

$$\begin{bmatrix} 12.4907 & 8 & 9 \\ 8 & 14.4907 & 6 \\ 9 & 6 & 6.4907 \end{bmatrix} \begin{bmatrix} \chi_1 \\ \chi_2 \\ \chi_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$12.4907x_1+8x_2+9x_3 = 0$ …….. Eq:1

$8x_1+14.4907x_2+6x_3 = 0$ …….. Eq:2

$9x_1+6x_2+6.4907x_3 = 0$ …….. Eq:3

By Eq:1/9-Eq:2/6,

$1.3879x_1 + 0.8889x_2 + x_3 = 0$

$1.3333x_1 + 2.4151x_2 + x_3 = 0$

‘- ‘- ‘-

$0.0546x_1 - 1.5262x_2 = 0$ …… Eq:4

By Eq:2/8-Eq:3/9,

$x_1 + 1.8113x_2 + 0.75x_3 = 0$

$x_1 + 0.6667x_2 + 0.7212x_3 = 0$

‘- ‘- ‘-

$1.1446x_2 + 0.0228x_3 = 0$ ……. Eq:5

By Eq:4/1.5262+Eq:5/1.1446,

$0.0358x_1 - x_2 = 0$

$x_2 + 0.0252x_3 = 0$

$0.0358x_1 + 0.025x_3 = 0$

Let x1=1, $0.025x_3$ = -0.0358

x_3 =-1.4206

In Eq:4, $-1.5262x_2$ =-0.0546

x_2 =0.0358

‘v=$\begin{bmatrix} 1 \\ 0.0358 \\ -1.4206 \end{bmatrix}$

Length, L = $\sqrt{(1)^2 + (0.0358)^2 + (-1.4206)^2}$ =1.7376

Normalized vector, v_3= $\begin{bmatrix} 0.5755 \\ 0.0206 \\ -0.8176 \end{bmatrix}$

V=$\begin{bmatrix} 0.6308 & 0.5203 & 0.5755 \\ 0.625 & -0.7804 & 0.0206 \\ -0.46 & 0.3469 & -0.8176 \end{bmatrix}$

By U =AVS^{-1}

=A$\begin{bmatrix} 0.1162 & 0.1734 & 0.3622 \\ 0.1151 & -0.2601 & 0.013 \\ -0.0847 & 0.1156 & -0.5162 \end{bmatrix}$

$$= \begin{bmatrix} 0.1466 & 0.0289 & -0.1399 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.2313 & -0.0867 & 0.3763 \\ 0.1162 & 0.1734 & 0.3633 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.1162 & 0.1734 & 0.3633 \\ 0.2313 & -0.0867 & 0.3763 \\ 0.1162 & 0.1734 & 0.3633 \\ 0.0315 & 0.289 & -0.1529 \\ 0.0315 & 0.289 & -0.1529 \\ 0.0315 & 0.289 & -0.1529 \\ 0.1162 & 0.1734 & 0.3633 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \end{bmatrix}$$

$$V_k^T = \begin{bmatrix} 0.6308 & 0.625 & -0.46 \\ 0.5203 & -0.7804 & 0.3469 \end{bmatrix}$$

$$S_k^{-1} = \begin{bmatrix} 0.1842 & 0 \\ 0 & 0.3333 \end{bmatrix}$$

New App,

App1(0.6308,0.5203)

App2(0.625,-0.7804)

App3(-0.46,0.3469)

New QueryApp,

‘queryApp $= q^T U_k S_k^{-1} = \begin{bmatrix} 1.4573 & -0.2601 \end{bmatrix} S_k^{-1}$

$= \begin{bmatrix} 0.2684 & -0.0867 \end{bmatrix}$

By Jaccard Similarities,

$$\text{Sim(App1,queryApp)} = \frac{.6308 * 0.2684 + 0.5203 * (-0.0867)}{(0.6308)^2 + (0.5203)^2 + (0.2684)^2 + (-0.0867)^2 - 0.1242}$$

$$= \frac{0.12420}{0.3979 + 0.2707 + 0.072 + 0.0075 - 0.1242}$$

$$= 0.1991$$

$$\text{Sim(App2,queryApp)} = \frac{0.625 * 0.2684 + (-0.7804) * (-0.0867)}{(0.625)^2 + (-0.0784)^2 + (0.2684)^2 + (-0.0867)^2 - 0.2355}$$

$$= \frac{0.2355}{0.3906 + 0.006 + 0.072 + 0.0075 - 0.2355}$$

$$= 0.9788$$

$$\text{Sim(App3,queryApp)} = \frac{-0.46 * 0.2684 + 0.3469 * (-0.0867)}{(-0.46)^2 + (0.3469)^2 + (0.2684)^2 + (-0.0867)^2 + 0.1536}$$

$$= \frac{-0.1536}{0.2116 + 0.1203 + 0.072 + 0.0075 + 0.1536}$$

$$= -0.2719$$

	App1	App2	App3	testApp
Access_Network_State	1	1	1	1
Access_Wifi_State	1	1	1	0
Internet	1	1	1	1
Read_Phone_State	1	1	1	0
Write_External_Storage	1	1	1	1
Mount_Unmount_Filesystems	1	1	0	0
Read_External_Storage	1	0	0	1
Install_Shortcut	1	1	1	0
Uninstall_Shortcut	1	0	0	0
Read_Settings	1	1	0	0
Receive_Boot_Completed	1	0	0	0
Get_Tasks	1	0	1	0
System_Alert_Window	1	0	1	0
Wake_Lock	1	0	1	1
Get_Accounts	1	0	0	0
Raised_Thread_Priority	0	1	0	0
Write_Secure_Settings	0	1	0	0
Write_Settings	0	1	0	0
Change_Network_State	0	1	0	0
Receive_MMS	0	1	0	0
Receive_Wap_Push	0	1	0	0
Read_SMS	0	1	0	0
Send_SMS	0	1	0	0
Receive_SMS	0	1	0	0

$$A^TA = \begin{bmatrix} 15 & 8 & 9 \\ 8 & 17 & 6 \\ 9 & 6 & 9 \end{bmatrix}$$

By $A^TA - \lambda I = 0$

$$\begin{bmatrix} 15 & 8 & 9 \\ 8 & 17 & 6 \\ 9 & 6 & 9 \end{bmatrix} - \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} = 0$$

$$\begin{bmatrix} 15-\lambda & 8 & 9 \\ 8 & 17-\lambda & 6 \\ 9 & 6 & 9-\lambda \end{bmatrix} = 0$$

$$(-1)^{1+1}(15-\lambda)\begin{bmatrix} 17-\lambda & 6 \\ 6 & 9-\lambda \end{bmatrix}+(-1)^{1+2}(8)\begin{bmatrix} 8 & 6 \\ 9 & 9-\lambda \end{bmatrix}+(-1)^{1+3}(9)\begin{bmatrix} 8 & 17-\lambda \\ 9 & 6 \end{bmatrix}=0$$

$$(15-\lambda)\left[(17-\lambda)(9-\lambda)-36\right]-8\left[8(9-\lambda)-54\right]+9\left[48-9(17-\lambda)\right]=0$$

$$(15-\lambda)[153-17\lambda-9\lambda+\lambda^2-36]-8[72-8\lambda-54]+9[48-153+9\lambda]=0$$

$$(15-\lambda)(\lambda^2-26\lambda+117)-8(18-8\lambda)+9(9\lambda-105)=0$$

$$-\lambda^3+41\lambda^2-362\lambda+666=0$$

‘λ_1=29.4907

‘λ_2=9

‘λ_3=2.5093

$S_1=\sqrt{\lambda_1}$=5.4305

$S_2=\sqrt{\lambda_2}$=3

$S_3=\sqrt{\lambda_3}$=1.5841

$$S=\begin{bmatrix} 5.4305 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1.5841 \end{bmatrix}$$

$$S^{-1}=\begin{bmatrix} 0.1842 & 0 & 0 \\ 0 & 0.3333 & 0 \\ 0 & 0 & 0.6313 \end{bmatrix}$$

By $(A^TA - \lambda I)\,\vec{v}=\vec{0}$

Let $\lambda_1=29.4907$

$$\begin{bmatrix} 15 & 8 & 9 \\ 8 & 17 & 6 \\ 9 & 6 & 9 \end{bmatrix} - \begin{bmatrix} 29.4307 & 0 & 0 \\ 0 & 29.4307 & 0 \\ 0 & 0 & 29.4307 \end{bmatrix} \vec{v}=\vec{0}$$

$$\begin{bmatrix} -14.4907 & 8 & 9 \\ 8 & -12.4907 & 6 \\ 9 & 6 & -20.4907 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$-14.4907x_1+8x_2+9x_3 = 0$ …….. Eq:1

$8x_1-12.4907x_2+6x_3 = 0$ …….. Eq:2

$9x_1+6x_2-20.4907x_3 = 0$ …….. Eq:3

By Eq:1/9-Eq:2/6,

$-1.6101x_1 \quad +0.8889x_2 \quad +x_3 = 0$

$1.3333x_1 \quad -2.0818x_2 \quad +x_3 = 0$

‘+ ‘+ ‘-

$-2.9434x_1 \quad +2.9707x_2 = 0$ …….. Eq:4

By Eq:2/8-Eq:3/9,

$x_1 \quad -1.5613x_2 \quad +0.75x_3 = 0$

$x_1 \quad +0.6667x_2 \quad -2.2767x_3 = 0$

‘- ‘- ‘+

$-2.228x_2 \quad +3.0267x_3 = 0$ …….. Eq:5

By Eq:4/2.9707+Eq:5/2.228,

$0.9908x_1 + x_2 = 0$

$-x_2 + 1.3585x_3 = 0$

$0.9908x_1 + 1.3585x_3 = 0$

Let $x_1=1$,

$1.3585x_3 = -0.9908$

$'x_3 = -0.7293$

In Eq:4, $2.9707x_2 = 2.9434$

$x_2 = 0.9908$

$$'v = \begin{bmatrix} 1 \\ 0.9908 \\ -0.7293 \end{bmatrix}$$

Length, $L = \sqrt{(1)^2 + (0.9908)^2 + (-0.7293)^2} = 1.5854$

Normalized vector, $v_1 = \begin{bmatrix} 0.6308 \\ 0.625 \\ -0.46 \end{bmatrix}$

Let $\lambda_2 = 9$

$$\begin{bmatrix} 6 & 8 & 9 \\ 8 & 8 & 6 \\ 9 & 6 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$6x_1+8x_2+9x_3 = 0$ …….. Eq:1

$8x_1+8x_2+6x_3 = 0$ …….. Eq:2

$9x_1+6x_2 = 0$ …….. Eq:3

By Eq:1-Eq:2,

$6x_1 \quad +8x_2 \quad +9x_3 \quad = \quad 0$

$8x_1 \quad +8x_2 \quad +6x_3 \quad = \quad 0$

‘- ‘- ‘-

$-2x_1 \quad +3x_2 \quad = \quad 0$ …….. Eq:4

Let $x_1=1$,

$3x_3 \quad =2$

$x_3=0.6667$

In Eq:3,

$6x_2=-9$

‘$x_2=-1.5$

‘v=$\begin{bmatrix} 1 \\ -1.5 \\ 0.6667 \end{bmatrix}$

Length, L = $\sqrt{(1)^2+(-1.5)^2+(0.6667)^2}$ =1.9221

Normalized vector, v_2= $\begin{bmatrix} 0.5203 \\ -0.7804 \\ 0.3469 \end{bmatrix}$

Let λ_3=2.5093,

$$\begin{bmatrix} 12.4907 & 8 & 9 \\ 8 & 14.4907 & 6 \\ 9 & 6 & 6.4907 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$12.4907x_1+8x_2+9x_3 \quad =0$ …….. Eq:1

$8x_1+14.4907x_2+6x_3 \quad =0$ …….. Eq:2

$9x_1+6x_2+6.4907x_3 \quad =0$ …….. Eq:3

By Eq:1/9-Eq:2/6,

$1.3879x_1 \quad +0.8889x_2 \quad +x_3 \quad = \quad 0$

$1.3333x_1 \quad +2.4151x_2 \quad +x_3 \quad = \quad 0$

‘- ‘- ‘-

$0.0546x_1 \quad -1.5262x_2 \quad = \quad 0 \quad \ldots\ldots \quad$ Eq:4

By Eq:2/8-Eq:3/9,

$x_1 \quad +1.8113x_2 \quad +0.75x_3 \quad = \quad 0$

$x_1 \quad +0.6667x_2 \quad +0.7212x_3 \quad = \quad 0$

‘- ‘- ‘-

$1.1446x_2 \quad +0.0228x_3 \quad = \quad 0 \quad \ldots\ldots\ldots$ Eq:5

By Eq:4/1.5262+Eq:5/1.1446,

$0.0358x_1 \quad -x_2 \quad = \quad 0$

$x_2 \quad +0.0252x_3 \quad = \quad 0$

$0.0358x_1 \quad +0.025x_3 \quad = \quad 0$

Let x1=1, $0.025x_3 \quad = -0.0358$

$x_3 \quad = -1.4206$

In Eq:4, $-1.5262x_2 \quad = -0.0546$

$x_2 \quad = 0.0358$

$$‘v = \begin{bmatrix} 1 \\ 0.0358 \\ -1.4206 \end{bmatrix}$$

Length, $L = \sqrt{(1)^2 + (0.0358)^2 + (-1.4206)^2} = 1.7376$

Normalized vector, $v_3 = \begin{bmatrix} 0.5755 \\ 0.0206 \\ -0.8176 \end{bmatrix}$

$$V = \begin{bmatrix} 0.6308 & 0.5203 & 0.5755 \\ 0.625 & -0.7804 & 0.0206 \\ -0.46 & 0.3469 & -0.8176 \end{bmatrix}$$

By $U = AVS^{-1}$

$$= A \begin{bmatrix} 0.1162 & 0.1734 & 0.3622 \\ 0.1151 & -0.2601 & 0.013 \\ -0.0847 & 0.1156 & -0.5162 \end{bmatrix}$$

$$= \begin{bmatrix} 0.1466 & 0.0289 & -0.1399 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.2313 & -0.0867 & 0.3763 \\ 0.1162 & 0.1734 & 0.3633 \\ 0.1466 & 0.0289 & -0.1399 \\ 0.1162 & 0.1734 & 0.3633 \\ 0.2313 & -0.0867 & 0.3763 \\ 0.1162 & 0.1734 & 0.3633 \\ 0.0315 & 0.289 & -0.1529 \\ 0.0315 & 0.289 & -0.1529 \\ 0.0315 & 0.289 & -0.1529 \\ 0.1162 & 0.1734 & 0.3633 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \\ 0.1151 & -0.2601 & 0.013 \end{bmatrix}$$

$$V_k^T = \begin{bmatrix} 0.6308 & 0.625 & -0.46 \\ 0.5203 & -0.7804 & 0.3469 \end{bmatrix}$$

$$S_k^{-1} = \begin{bmatrix} 0.1842 & 0 \\ 0 & 0.3333 \end{bmatrix}$$

New App,

App1(0.6308,0.5203)

App2(0.625,-0.7804)

App3(-0.46,0.3469)

New QueryApp,

‘queryApp $= q^T U_k S_k^{-1} = [0.7037 \quad 0.7225] S_k^{-1}$

$= [0.1296 \quad 0.2408]$

By Jaccard Similarities,

$$Sim(App1,queryApp) = \frac{0.6308 * 0.1296 + 0.5203 * 0.2408}{(0.6308)^2 + (0.5203)^2 + (0.1296)^2 + (0.2408)^2 - 0.1242}$$

$$= \frac{0.2071}{0.3979 + 0.2707 + 0.0168 + 0.058 - 0.2071}$$

$$= 0.3862$$

$$Sim(App2,queryApp) = \frac{0.625 * 0.1296 + (-0.7804) * 0.2408}{(0.625)^2 + (-0.0784)^2 + (0.1296)^2 + (0.2408)^2 + 0.1069}$$

$$= \frac{-0.1069}{0.3906 + 0.006 + 0.0168 + 0.058 + 0.1069}$$

$$= -0.1849$$

$$Sim(App3,queryApp) = \frac{-0.46 * 0.1296 + 0.3469 * 0.2408}{(-0.46)^2 + (0.3469)^2 + (0.1296)^2 + (0.2408)^2 - 0.0239}$$

$$= \frac{0.0239}{0.2116 + 0.1203 + 0.0168 + 0.058 - 0.0239}$$

$$= 0.0624$$

Similarity results for the testing of malware app and goodware app

Applications	testApp1(Malware)	testApp2(Goodware)
App1	0.1989	0.3862
App2	0.9788	-0.1849
App3	-0.2719	0.0624