Android Applications Malware Detection using Machine Learning Approach

By: Guluma Nagaro

A Thesis submitted to the

Department of computing

School of Electrical Engineering and Computing

Presented in partial fulfillment of the requirement for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

Advisor: Dr. Mesfin Abebe

10/3/2018

Adama, Ethiopia

Android Applications Malware Detection using Machine Learning Approach

By: Guluma Nagaro

Advisor: Dr. Mesfin Abebe

A Thesis submitted to the

Department of computing

School of Electrical Engineering and Computing

Presented in partial fulfillment of the requirement for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

Advisor: Dr. Mesfin Abebe

10/3/2018

Adama, Ethiopia

Declaration

I hereby declare that this MSc Thesis is my original work and has not been presented for a degree in any other university, and all sources of material used for this thesis have been duly acknowledged.

Name: _____

Signature: _____

This MSc Thesis has been submitted for examination with my approval as thesis advisor.

Name: _____

Signature: _____

Date of Submission: _____

Approval of Board of Examiners

We, the undersigned, members of the Board of Examiners of the final open defense by _____ have read and evaluated his/her thesis entitled "_____" and examined the candidate. This is, therefore, to certify that the thesis has been accepted in partial fulfillment of the requirement of the Degree of ……

_____	_____	_____
Supervisor /Advisor	Signature	Date
_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

Acknowledgment

I would like to thanks almighty God and St. Marry whose everlasting and undying love kept me and for infinite mercy throughout my life and during the course of this thesis work. Next I would like to express my gratitude to my adviser; Dr. Mesfin Abebe for his kindly approach to help me staying focused on problems and provided some directions to approach them. He also explains the concepts at abstract level, opening up the mind to design scientific research methodology. Working with him, I have developed my problem-solving skills, learnt research process. I would be thankful to him for all my life.

I want to take the opportunity of this to thanks Ms. Desta Zerihun for not just he is my best friend but also for his effective advice and providing the concepts at designing level and developing the prototype. I would be thankful to him for all my life for scarifying himself through all my thesis work.

I also thank my beloved girl Lensa Shibiru, for her support in many directions through my life and I would like to thanks all my class mates who are working on their master's thesis- Megersa Dereje, Bayisa Gutema, Abebaw Degu, Shagaw Demissie, Yonas Mogas, Makuria Gemmachu, Mubarek Dubbiso, Noh Diriba. I really enjoyed working and discussing with them in the lab.

In the end, I would like to thanks all my wonderful and caring parents specially my mother Fikire Nagaro, for their support in my life, my beloved brother Kena Nagaro and my uncle Kebede Darge, for their financial, moral, and every support through my life. Without their advice and support I will not achieve this level.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

Abbreviation	*Expanded form*
A	Anomaly-Based
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
APK	Android Application Package
AVG	Anti-Virus Guard
BES	BlackBerry Enterprise Server
CA	Classification Algorithm
CART	Classification and Regression Tree
CAT	Category-Based
CSAMDS	Client-Server Android Malware Detection System
CSS	Cascading Sheet Style
DBN	Deep Belief Network
Dev	Device
DT	Decision Tree
GUI	Graphical User Interface
HP	Horsepower
HTC	High Tech Computer
HTML	Hyper Text Markup Language
I	Intent
IDC	International Data Corporation
iOS	iPhone Operating System
LG	Lucky Gold star
LR	Linear Regression
M	Pattern Matching
SHA-256	Secure Hash Algorithm
MIDP	Mobile Information Device Profile
MMS	Multimedia Messaging Service
MP	Multilayer Perceptron

N/A	Not Available
NB	Naïve Bayes
OS	Operating System
PA	Place of Analysis
PDA	Personal Digital Assistant
PER	Permission-Based
Ref	Reference
RF	Random Forest
S/MIME	Secure/Multipurpose Internet Mail Extensions
SA	Similarity Algorithm
SD	Storage Device
SDK	System Development Kit
Ser	Server
SIG	Signature-Based
SMO	Sequential Minimal Optimization
SMS	Short Message Service
SPE	Specification-Based
SQLite	Standard Query Lite
SSL	Secure Sockets Layer
SVM	Support Vector Machine
ToD	Type of Detection
TV	Television
UID	User Identifier
UML	Unified Modeling Language.
XML	Extensible Markup Language

Abstract

Android malware development has been increasing along with increasing the development of Android mobile devices. Some of the malicious intention of Android malware are exposing local information, reading users SMS messages, selling users information and stealing user credentials. To minimize the Android security problem, Google provide Android security mechanism that can help user and protect them from malware attack. However, the system cannot detect the malicious application at the background.

As a solution to these problems, in this research work we designed the signature-based and permission-based detection techniques with cloud server approach. Specifically, the study considers the SHA-256 value of the given application and compared with lists exist on local database. On the server side, the permissions required by the unknown application were analyzed to construct feature vector. Machine learning approaches are the current techniques to model static and dynamic features of Android malware. Whereas the detection performance of machine learning techniques increases with increasing the quality of attributes that used to differentiate malicious application and normal application.

Three classification algorithms such as Naïve Bayes, decision tree and SVM classification performance were compared. SVM classification algorithm, which has better classification accuracy is used for modeling classifier. The unknown application is submitted to the classifier model and the classifier detects the application. The proposed solution has been tested by analyzing both normal and malicious applications collected. According to the evaluation result, the proposed system can classify Android application into normal or malicious. Generally, client-server Android malware detection approach with the aid of machine learning methods has better detection performance than other existing detection techniques.

Keywords: Android malware, Static analysis, Malware detection, Machine learning, Classification.

CHAPTER 1

1 Introduction

1.1 Background of study

Personal Digital Assistants (PDAs), mobile phones and as of late smartphones have developed from straightforward mobile phones into refined yet reduced minicomputers which can interface with a wide range of systems, including the Internet and corporate intranets. Developed as open, programmable, arranged devices, smartphones are vulnerable to different malware dangers, for example, viruses, Trojan horses, and worms, which are all notable from desktop platforms. These devices empower users to access and browse the Internet, get and send messages, SMSs, and MMSs, connect with different devices for trading data/synchronizing, and initiate different applications, which make these devices attack targets [1].

A compromised smartphone can incur serious harms to the both users and the cellular service. Malware on a mobile device can make the phone mostly or completely unusable; cause undesirable charging; take private data (perhaps by Phishing and Social Engineering); or contaminate the contacts in the phone directory. Possible attack vectors into mobile phones include: Cellular networks, Internet connections (by means of Wi-Fi, GPRS/EDGE or 3G organize get to); USB/ActiveSync/Docking and different peripherals [1].

According to preliminary results from the International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker, phone companies shipped a total of 347.4 million smartphones worldwide in the first quarter of 2017 (1Q17). Android dominated the market more than 78% market share in 1Q17 [3].

In 2016, Jacob Poushter reports that in Ethiopia 4 % of the population owning a smartphone [4]. With the proliferation of smartphone, the challenges for the smartphone security are becoming very similar to that personal compute encounter. Malware on Android mobile can make the phone partially or fully unusable; cause unknown billing; steal private information; send unnecessary SMS messages; or infect the contact of the users in phone-book [5]. Since the response time of antivirus vendors may vary between several hours to several days to identify the new malware, generate a signature, and update their clients signature database, hackers have a substantial window

of opportunity. Some malware instances may target a specific and relatively small number of mobile devices (e.g., to extract confidential information or track owners' location) and will therefore take quite a time till they are discovered [1].

Among all existing mobile devices like Blackberry, Windows Phone, Android, Apple iOS, Palm OS and Tizen, Linux-based Android and Unix-based Apple iOS are the most two known and familiar platforms. Meanwhile, security dangers in Android applications have likewise immediately expanded. Specifically, four major classes of issues, malware, program vulnerabilities, security spills and shaky application portrayals convey significant difficulties to Android application security. Android applications give potential vulnerability to getting to privacy through them. The Android device store secured user data. Thus, they become vulnerable to many malicious attackers [8].

In the smartphone industry, a device with an Android operating system holds a leading position. The admiralty of the Android operating system brings many developers to develop not only a benign application but also to develop some infected applications. Infected applications have been progressively increasing as indicated by McAfee's yearly report. This situation inspires us to study mobile application security, particularly in Android [9].

Malicious developers take a legitimate application from Android market and repackage it with malware then they upload the modified application to the third-party application store. The user downloads the application with malicious software and installs on their mobile device which causes the device easily vulnerable. Figure 1.1, describes the process used by malware writers to take legitimate applications from the Android Market, repackage them with malware, and introduce the repackaged versions into third-party app stores [10].

Many researchers concern with this issue in order to detect Android malware application using two detection techniques: static analysis and dynamic analysis. In case of static analysis, malicious code is extracted before application execution, while dynamic analysis collects apps' behavior data to find malware during applications runtime. Static analysis is fast and more accurate with known malware, but it lacks the actual execution path and relevant execution context. Moreover, there exist challenges when a slight change of code present at runtime.

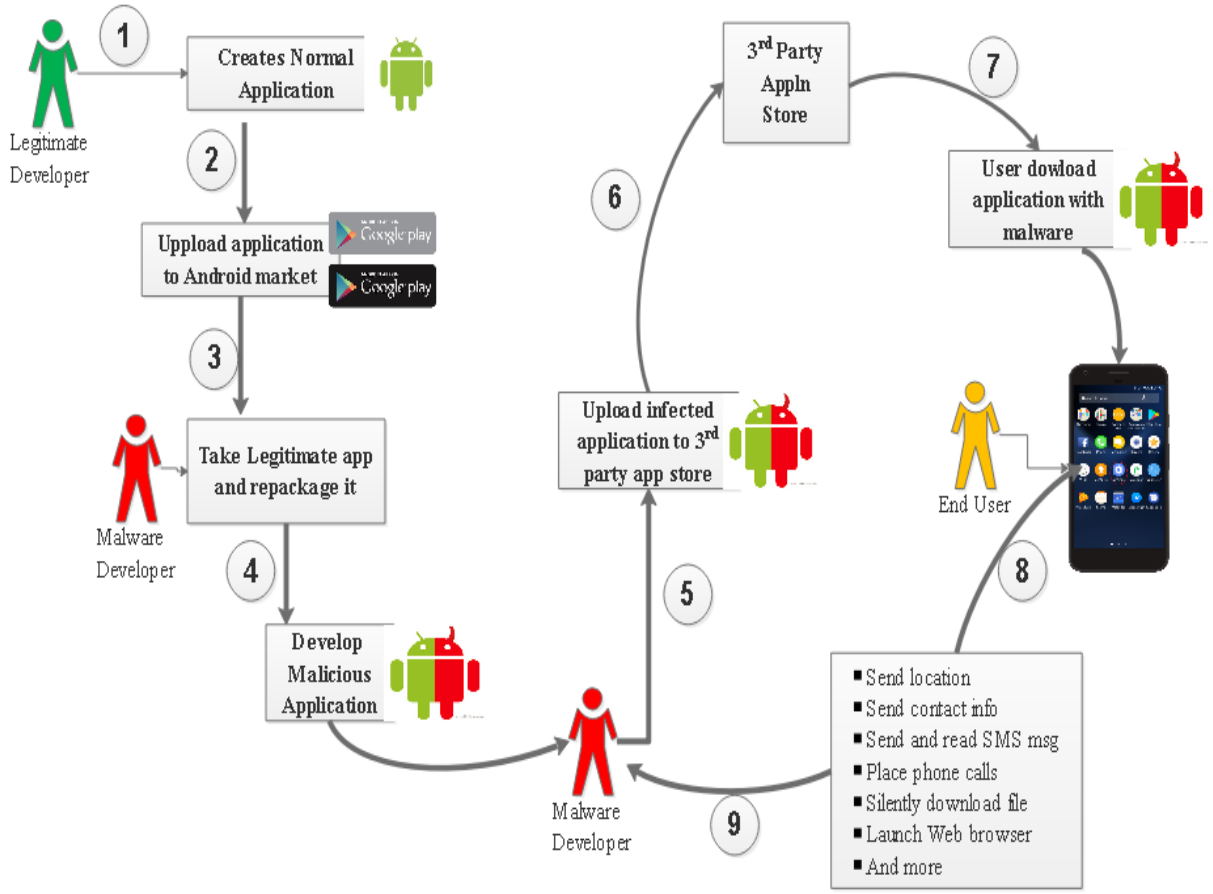

Figure 1. 1 Repackaging Android Applications Procedure [10].

Dynamic analysis can distinguish zero-day attacks. Even though this technique overcomes the static analysis in many views, it also has some drawbacks. Firstly, it consumes many resources. Secondly, the Dynamic analysis is subject to low code coverage. Thirdly, it failed when malware attempts to detect the emulator and other dynamic analysis, avoiding launching their payloads [9].

Since both of these methods are used to detect malicious application when user download the applications to install on the Android mobile device, this study contributes by updating the user whether the application is malicious or not. Many users of android application have no the awareness of untrusted android applications. Machine learning aided approaches have been used to detect the malicious Android application and report the class of the application to the user.

1.2 Motivation

Android dominated the market more than 78% market share in 1Q17 which represents the highest share among other mobile platforms [3]. Statistically speaking, 2.3 million active users over the world concerned with Android application security. With high popularity of Android environment, the development of Android applications is also increasing. The statistics (Statista) shows that, the number of Android applications available in the Google Play Store was 3 million in June 2017 [11]. Besides this official market, a large number of applications are available in unofficial third-party market. For such a huge amount of Applications, it must be hard for the system to detect which application is acting malware, or identify some sensitive operations are permitted by users or not.

In our country (or Ethiopia) around 4 % of the population are using the smartphone [4]. Most of the user have no the awareness of security mechanism of the operating system during installation of the application. They don't know the official application store (or market), from where they must download the applications.

As the number of activities increased that users perform on the smartphone, personal privacy protection, and device security become the issue. Although the Android operating system is designed to be highly secured and they keep updating and evolving to fix bugs, the security danger in Android application immediately spread. Google provide some of Android security mechanisms that can help user and protect them from malware attack. However, the system cannot detect the malicious application at the background.

Machine learning approaches are the current techniques to model static and dynamic features of Android malware. Whereas local database and server database of malicious applications improves the detection accuracy and response time of the system.

1.3 Statement of Problem

The Android operating system is top popular and open source nature, which enforce the attackers to make their targets to spread the malicious code over mobile applications. So, many researchers focus on Android malware detection. Although there exists quite technique to detect this malicious code in Android applications, their approach to detecting Android application malware is not

totally suitable for mobile devices. The following problems were formulated to be answered in this thesis work:

- ***Problem 1. Detecting at the Background:*** Many users have no awareness of Android malware detection mechanism while they install the Android application. Therefore, it is significant to design Android malware detection system that informs the user whether the application is malicious or not.
- ***Problem 2***. ***Minimum Response Time:*** Static analysis lacks the actual execution path and relevant execution context; lack the ability to examine the code that is loaded dynamically at runtime. Although dynamic analysis overcomes the static analysis in many views, it also has some drawbacks. Firstly, dynamic analysis requires too many resources relative to static analysis, which hinders it from being deployed on the resource constraint smartphone. Secondly, dynamic analysis is subject to low code coverage. Thirdly, the recent malware attempts to detect the emulator and other dynamic analysis systems, avoiding launching their payloads.
- ***Problem 3. Detecting unknown malware (new malware developed):*** Android operating system use malware detection mechanism, based on yearly reported malware samples from different companies. The new coming malicious application is not considered under this mechanism.
- ***Problem 4. Effective Detection:*** Parallel to the Google Play Store market, there is a various third-party market which provides millions of applications for users. However, a large number of applications found on the third-party market are designed maliciously and damage users mobile. For this reason, it becomes necessary to develop a more accurate and available system for Android malware detection approach.

These problems motivate us to carries out this research work. The aforementioned problems are solved by the following research questions:

- ***RQ1***. How can the system detect the malicious application at the background?
- ***RQ2***. How to improve the response time of Android malware detection system?
- ***RQ3***. What are the most important learning algorithms that are used in effective Android malware detection?
- ***RQ4.*** How to design better approach for detecting malicious Android Applications?

1.4 Contribution of the Study

The main contribution of this study is developing cloud-based Android malware detection system by the aid of machine learning. We generated more available system that detect malicious application low computational resource as possible. To achieve this goal, a cloud-based system with machine learning approach was developed. Better machine learning algorithm is applied to improve the performance of Android-based malware detection. Machine learning algorithm is used to increase the detection accuracy.

Previously various types of malware detection analysis were described by many researchers, but many of them have their own limitation. To overcome some limitation of those detection techniques, in this study client-server Android malware detection system was proposed.

1.5 Objectives of the study

1.5.1 General Objectives

The general objective of this study is to develop a machine learning model that detect Android malware.

1.5.2 Specific Objectives

- Revise the previous methodology and design the modified method to detect application based Android malware.
- Design an architecture that detect malicious application at the background.
- Design an architecture that provides a cloud-based Android malware detection system by the aid of machine learning.
- Develop a prototype for Android Malware Detection system by using machine learning.
- Evaluate the performance of the learning algorithms.

1.6 Research Methodology

In order to accomplish the above objectives and to develop the Android malware detection model as well as to answer the research questions, the following procedures were used.

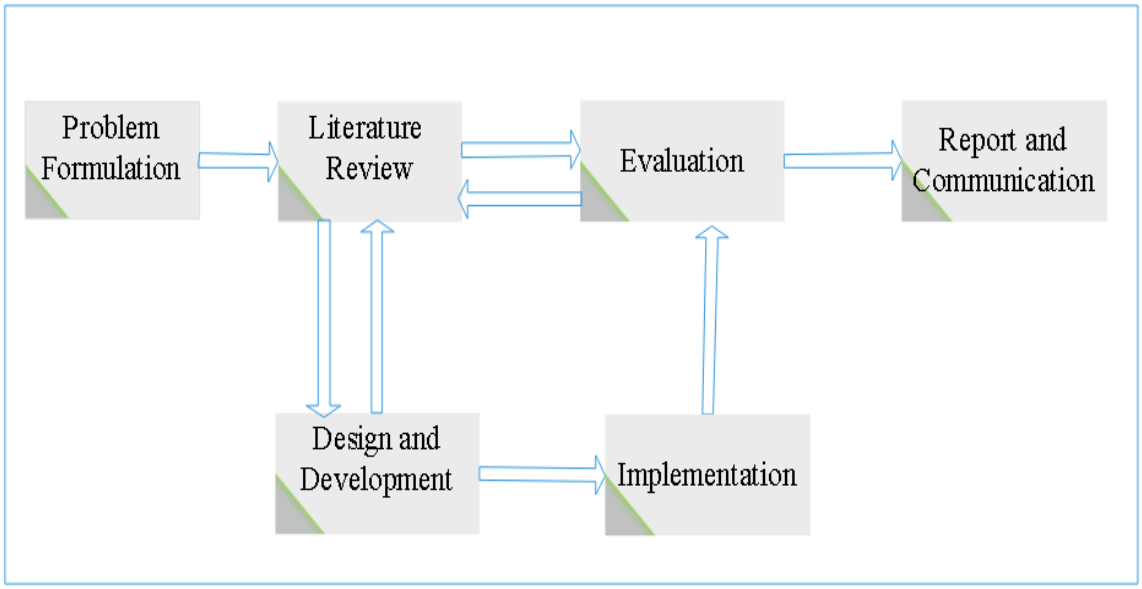

Figure 1. 2 Flowchart of Research Processes

The first step in the research process is identifying and formulating the problem or developing a hypothesis by reviewing different kinds of literatures. Then, collecting data from different sources related to Android malware detection techniques. The third procedure in this study is surveying the existing works and choosing the most important that has the highest classification accuracy for Android malware detection. Designing the user level malicious application detection framework for the Android operating system is the next step of the research process in this study. On the implementation step, there are two phases such as analysis phase and prediction phase. In the analysis phase, static analysis has been performed in order to extract features manually (using APKtool).

In the prediction phase, a machine learning classifier modeling was performed. Finally, three of the supervised machine learning algorithms such as SVM, NB, and Decision Tree is used and the proposed framework is evaluated against existing frameworks for Android malware detection.

1.6.1 Literature Review

A different research studies on Android malware detection have been reviewed in order to understand research works, to design research methodology and to solve the research questions. The classification accuracy of different supervised machine learning algorithms is analyzed and selected for this study.

1.6.1.1 Data Preparation for Machine Learning

The data preparation is an important step in the preprocessing phase of this study, i.e. Analysis phase. Since APK file is compressed.ZIP format file, reverse engineering is performed on the file and prepared for static analysis that used to extract feature. After the features are extracted from the AndroidManifest.xml, independent variables are prepared that is used as input for learner or model.

The concept of GIGO is used in data preparation in which good data is a prerequisite to achieving an effective model of Android malware detection. The main importance of data preparation is to develop better models that classify Android app into malware or benign more accurately.

The following Machine learning steps are followed in the knowledge discovery process

- ***Data cleaning***: - The removal of disturbance and inconsistency for the collection of normal and malware app data.
- ***Data integration***: - The combination of multiple data.
- ***Data selection***: - Features for both normal and malicious applications for analysis is retrieved from the database.
- ***Data transformation***: - Vector generation or transformation of data into forms appropriate for classification.
- ***Training data***: - The use of python module to extract patterns from data.
- ***Pattern evaluation***: - Classification of patterns into malware or normal.
- ***Knowledge presentation***: - Visualization of extracted knowledge to the end user.

1.6.2 Training and Building Classification Model

Many researchers were focusing on several machine learning algorithms for their studies. Machine learning algorithms are become increasingly important in the Antivirus industry, particularly in augmenting well-established generics and heuristics algorithms [12].

Building machine learning classification model involves two phases. Training phases, in which model is constructed and parameters are adjusted using training data. The second phase is the testing phase, in which the learned model is applied to new data that is data not used in the training phase.

1.6.3 Prototype Development Tools

Different types of designing, modeling, implementation, and testing tools are used to achieve the output or result of this study. APKtool is used to decompile APK file into the source code for static analysis. Training data required for training are prepared by using python.

1.7 Scope and Limitation

1.7.1 The scope of the Study

This research work is conducted to detect Android malware using a machine learning algorithm and to select the better machine learning algorithm used in classification. This study is static analysis that use the attributes or features that can be extracted from AndroidManifest.xml. Android malware detection is designed and the system prototype is implemented based on static analysis. The signature-based and permission-based approach is used to detect malicious Android application. Whereas client-server approach is proposed to improve the performance of malware detection system for Android. Three machine learning classifiers: Support Vector Machine, Naïve Bayes and Decision Tree are trained. By evaluating the classification accuracy of the classifiers, the classifier algorithm with highest classification accuracy is used in modeling classifier.

1.7.2 Limitation of the study

Since in our country (or Ethiopia) there is no antivirus application development companies, we didn’t get the standard data sets for our experiment. There are also other limitations which found in the study due to different constraints like the time schedule and resource limitations such as

standard data sets. In this study, the approach developed may consider the normal application as a malicious application because of very little difference between permissions requested by benign application and permissions requested by the malicious application. The other limitation of the study is that a malicious application with no permission requested cannot detect.

1.8 Application of Results

The significant of the study is two folds. Antivirus companies get an opportunity to provide more trustable Android malware detection approach before they release their mobile antivirus application. And, it also enables them to improve the accuracy of malicious application detection. Every user of Android mobile also gets an opportunity to protect themselves from installing the infected application by malware on their device. And also, they save themselves from credit to buy antivirus applications. The advantages of the proposed solution for Android malware detection are listed below:

- Reduce malware detection related problems for Android-based mobile.
- Enhance user protection from malware attacks.
- Reduce the cost of malware detection for Android-based device users.
- Maximizing the availability of detection system.

1.9 Organization of the thesis

This thesis is organized into seven chapters in the following ways.

Chapter one describes an overview of the study such as background information, statements of the problem, motivation, objectives, scope and limitation, and application of the study.

Chapter two presents about literature review. It includes the overview of the mobile application; types of smartphone operating system; mobile operating system security; mobile device screen size and resolution, and related works.

Chapter three discusses the research methodology which includes Android malware detection methods, data collection method, design proposed framework method, the framework implementation method and the prototype evaluation method. This chapter also describes the design and implementation tools used for the research work.

Chapter four discusses the proposed framework design and describes each proposed framework components.

Chapter five describes implementation of the proposed framework i.e. each sample prototype malicious application detection component and the way of integration client with the server.

Chapter six presents result, evaluation and discussion of the framework based on the prototype implementations.

Chapter seven consists of conclusion, recommendation and identifies future works for further research direction on this research topic.

CHAPTER 2

2 Literature Review and Related Works

In this chapter, literature review has been made on the general concepts and methods of machine learning and its application in the data classification. The major smartphone platforms, Android Operating system; android Application structure; vulnerability of Android platform, and related work previously studied on Android malware detection were discussed and starts with Android application security measure and two widely used mobile device vendors and application platform vendors. Following this, the Android permission system and detection of android applications malware are studied in the area of Android malware detection. Finally, detection of Android malware detection techniques and machine learning algorithms are presented.

2.1 An Overview of Mobile Devices

Mobile is one type of wireless communication mechanism for users in conformity with offering their everyday activities. Basically, mobile devices are broken into two major classes based on their purposes presenting capacities, such as ordinary mobile and smartphone mobile. Traditional mobile devices provide easy functions as ring calling, sending SMS and MMS [13]. Whereas, smartphone devices bear provides extra hardware part or software features [14]. These hardware functions which include high memory capacity, various types of sensors, excessive processor speed, large honor size and excessive resolution. The application functions additionally include web searching capabilities, video chat, internet service, word processing, multiprocessing, accept applications, multitasking, or etc.

2.2 Mobile Operating System

A Mobile operating system is an operating system for smartwatches, phones, tablets and other mobile devices. A cell operating provision (Mobile OS) is a software platform concerning top of which other programs called application programs, can run over mobile devices such as personal digital assistant (PDA), tablets, mobile phones, smartphones and other mobile devices. Over the years, Mobile OS design has skilled a three-phase evolution: beside the PC-based operating system to an embedded operating system according to the current smartphone-oriented operating system in the previous decade. Throughout the process, Mobile OS architecture has long gone from

complex to simple to something in-between. The gradual increase method is naturally driven via the technology advancements into hardware, software, and the Internet:

- ***Hardware:*** The industry has been decreasing the factory size of microprocessors and peripherals to develop real mobile devices. Mobile operating systems for PDAs usually didn't have full multifunction or 3D graphics support. Features like sensors, such as an accelerometer, and capacitor-based touch screens were not available in ancient time.
- ***Software:*** with minicomputers such as laptop computer, the software is mainly concerned with the user's productivity, where support for a keyboard that has precise inputs are essential. The software for private data helper, help the user to protect the personal data such as contact information, hot data like a bank account, email, and so on.
- ***Internet:*** Along with Internet development, especially after Web 2.0, there is plentiful data into the network waiting after being searched, organized, mined, and delivered to users. People are more and more living with the Internet instead just searching the Web. More and more human beings are worried about the development, together with information contribution, application development, and social interactions. The mobile operating systems can't stay self-contained but have in conformity with be open systems.

The aforementioned technological developments have resulted in a variety of species competing for mobile operating system solutions concerning the market is driven by different actors. Some of these actors include Apples' iOS, Nokia's Symbian, RIM's BlackBerry OS, Samsung's Bada, Microsoft's Windows Phone, Hewlett-Packard's WebOS, and Google's Android OS. The following sub-sections review six of the most popular mobile operating systems.

2.2.1 Apple OS

An iPhone operating system (iOS) is extensively used types of operating system for Apple's smartphone devices which are developed by Apple Inc. Currently, this operating system is applicable to special types of Apple's products such as the iPhone, iPod Touch, iPad, or Apple Television [15]. It is the second most popular type of smartphone operating system among the market after Android [14]. This intense however highly-priced operating system is applied by using an objective-oriented programming language.

An iPhone operating system (iOS) has the most popular direct manipulation functions for easily controlling user interaction including smartphone devices by way of the usage of more than a few types over a touchscreen gesture such as tapping, swiping and pinching similar to Android platform. The iOS has numbers of versions were developed in order to improve the functions of the platform, such as iOS1, iOS2 and so on. Currently, Apple provides not solely the operating system; but additionally, different types of service, like software development platform, and various types of applications.

Generally, iOS has more than 300,000 applications that keep regarding Apple's App Store [15]. However, these applications can be downloaded on your system if and only if we paid a fee. This indicates that the customer can't arrive the applications for free like Android applications. This leads to decreasing the acceptance of the platform by the users.

2.2.2 Window OS

A Window OS is another mobile operating system that is launched by Microsoft Corporation in the name Of Window Phone 7 at 2010 [14]. Nowadays, different mobile manufacturing companies are developing Window Phone devices, including Nokia, Samsung, HTC, and LG. Successively, the Window Phone operating system was performed further update including Widow Phone 8 and Window Phone 8.1 for advancing the feature of the original operating system.

Even though protecting private or confidential information on mobile devices is challenging, the Window Phone operating system provides a set protection mechanism mentioned below that can help in protecting confidential information for users [16]:

- ***Internal storage encryption:*** A mechanism in which private data is encrypted in internal storage by industry proven BitLocker encryption.
- ***Removable storage protection:*** External storage device like SD cards are partitioned and encrypted.
- ***Information Rights Management:*** Limits data access within documents and email messages to only authorized users within and outside of the organization. This feature brings the same security strength of information protection as the full Windows 8.1 operating system but on a smartphone.

- ***S/MIME signing and encryption:*** Provide that the user identity for an email message is secure for a message sent within an outside the organization. More protect email messages by encrypting the message body and content so that only the designated recipients can read the message's contents. Managing the certificates used in S/MIME signing and encryption is easy.

2.2.3 Blackberry OS

Research in Motion (RIM), developed the Blackberry OS for their BlackBerry smartphones and tablet devices [17]. This operating system was applicable only for their devices, like Blackberry mobile and Tablet devices. Blackberry OS supports Java Mobile Information Device Profile (MIDP) and the Wireless Application Profile (WAP) protocols which are used to synchronize through a BlackBerry Enterprise Server (BES) with push-based calendar, task, contact, email, and note exchange. One of the main strengths of BlackBerry devices is their ability to handle corporate email.

And also, BES offered different types of service and features like security, remote wipe and capacity for Blackberry mobile device which can assess the internal network or the corporate data. Blackberry OS offers their own Blackberry Internet Service (BIS) to the customer to access their personal email, search different types of information from the internet and so on. Blackberry OS6 and O7 are types of Blackberry operating system versions which were constructed to encourage application development. Therefore, this operating system supports an application which is implemented by a different programming language such as Java is used to implement for mobile phone device and C++ or Web-based language for the Playbook Tablet devices

2.2.4 Symbian OS

Symbian is a smartphone operating system which was launched by Nokia [18]. Previously, there were different hardware vendors, including Motorola, Ericsson, and Nokia were formed Symbian OS. Today, the Symbian foundations are run on Nokia only. It provides a standard license agreement for the user in order to access the Symbian application. Nokia provides System Development Kits (SDKs) for Symbian application development that supports different types of programming languages, such as C++ and Java. Symbian has numbers of versions like other mobile operating systems, including Symbianˆ1, Symbianˆ2, and Symbianˆ3 and etc. These

versions were realizing in a different time in order to increase the feature of the Symbian application for satisfying the end user.

2.2.5 Web OS

Web OS is a proprietary and Linux kernel-based operating system for smartphone and smart TV devices, which was developed by Palm. This operating system is called HP web OS and Palm web OS. Both a native (inside in the operating system) and third-party applications are implemented by HTML, CSS, and JavaScript code. Palm was developed two types of a framework, such as Mojo and Enyo these frameworks are supporting to the developer can be easily creating a web application with the same layout rules [18].

2.2.6 Android OS

Android is an operating system for mobile phones. In Android OS, applications are strongly isolated from the system and each other by the customization of underlying Linux internals, and this mechanism is called sandboxing [19]. Mobile applications for Android are written in Java language and execute on the OS with a unique user and group identity (UID) assigned at installation time on their own Linux processes. Assignment of UID to applications assures the sandboxing mechanism which restricts resources and memory. Applying a fine-grained permission system, Android compels restrictions on communication, the share of resources and functions [19]. Other than some hardware-specific drivers, Android provides everything else to make their devices work [20].

2.2.6.1 Android Operating System Architecture

An Android operating system relies on Linux Kernel to take advantage of key security features and allows the developers to modify the driver to fit their needs. The driver is the first abstraction layer between hardware and other the software stack.

Figure 2.1, illustrates that Android is a software stack for mobile devices that have an OS, middleware and key applications and uses a modified version of the Linux kernel, which serves as an abstraction layer. The second layer is libraries such as SQLite, Webkit, SSL, and Free Type, which found on the top of Linux Kernel, provides the functionality of an Android system. The Application Framework layer provides all the APIs that the application requires to access the device hardware.

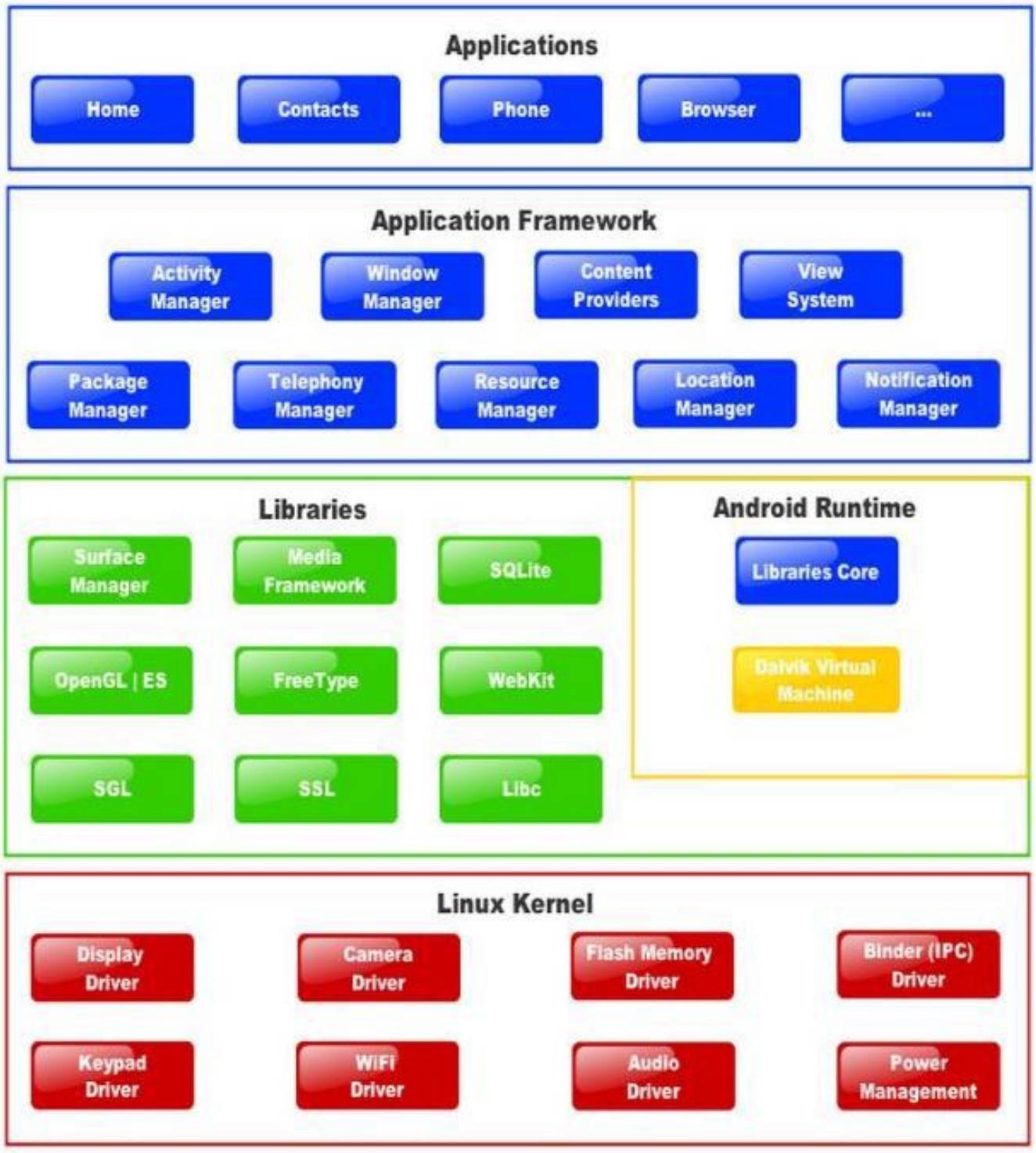

Figure 2. 1 Android Operating System Architecture [21]

At the top of the OS layer, there is user application which comes pre-installed and other application that is downloaded from the Android application market like Google Play market [22].

Those android applications that are pre-installed or downloaded from the android market are written in Java programming language. The written code is compiled by using the SDK tool then provide the apk file. The Android application has four different components: Activities, Services, Broadcast Receivers and content providers. APK file consists of AndroidManifest.xml, classes.dex, and other XML based resources required by the application to run.

2.3 Android Operating System Security

Mainly Android operating system security is based on permissions. In the Android model, all applications are separated and may access their own data not data from other applications. The system then applies a rigorous permissions system to services that are provided for the use of installed applications. Permission is one of the components of the AndroidManifest.xml which is required by the APIs in order to run. In order to access the mobile device, the application must first require permission and be allowed the device's user. This permission model consequently informs the user of what operations the application would be able to perform if the installation went successfully, and allows the user to make a decision whether to grant such permissions to the application and install it at all [23].

According to G Data Security Blog, more than 750,000 new malicious applications have sprung out during the first quarter of this year, with estimates the total number will grow up to a staggering 3.5 million by the end of 2017. The report further warns the problem is particularly widespread among devices from third-party phone makers where software updates that tend to receive software updates less frequently and sometimes with significant delays [24].
According to their findings, G Data claims malware increase most significantly in Android Lollipop and Marshmallow, accounting for two-thirds of all infected apps.

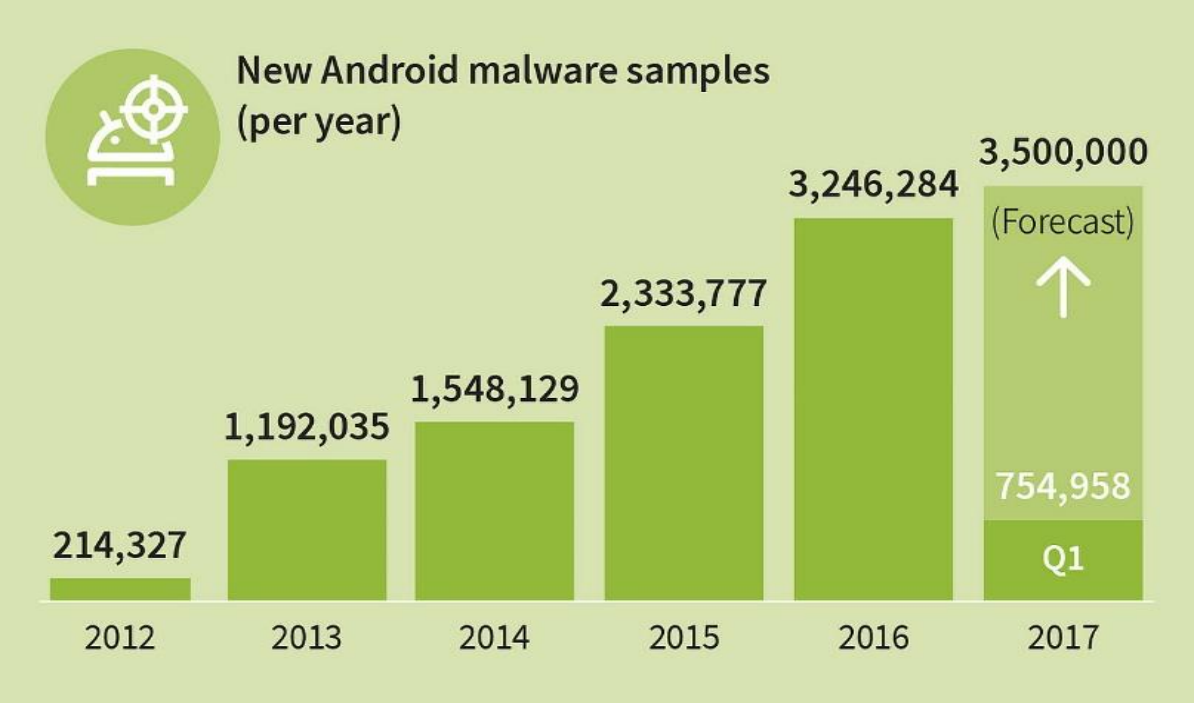

Figure 2. 2 New Android malware samples per year [24].

Here is the full breakdown:

- Gingerbread (2.3 – 2.3.7): 0.9%
- Ice Cream Sandwich (4.0.3 – 4.0.4): 0.9%
- Jelly Bean (4.1.x – 4.3): 10.1%
- KitKat (4.4): 20.0%
- Lollipop (5.0 – 5.1): 32.0%
- Marshmallow (6.0): 31.2%
- Nougat (7.0 – 7.1): 4.9%

Malware Authors use numerous techniques to avoid the detection such as a) code obfuscation, b) Encryption, c) including permission which is not needed by the application, d) Requested for unwanted hardware, e) Download or update attack in which benign app update itself or another application now with a malicious payload.

2.4 Android Applications

Android applications are dot APK files, which is brought in zipped form. Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. There exist four types of basic components to be used to build an application which are activities, services, content providers and Broadcast receivers. Activities represent a single screen with a user interface. Service is a component that runs in the background to perform long-running operations or to perform.

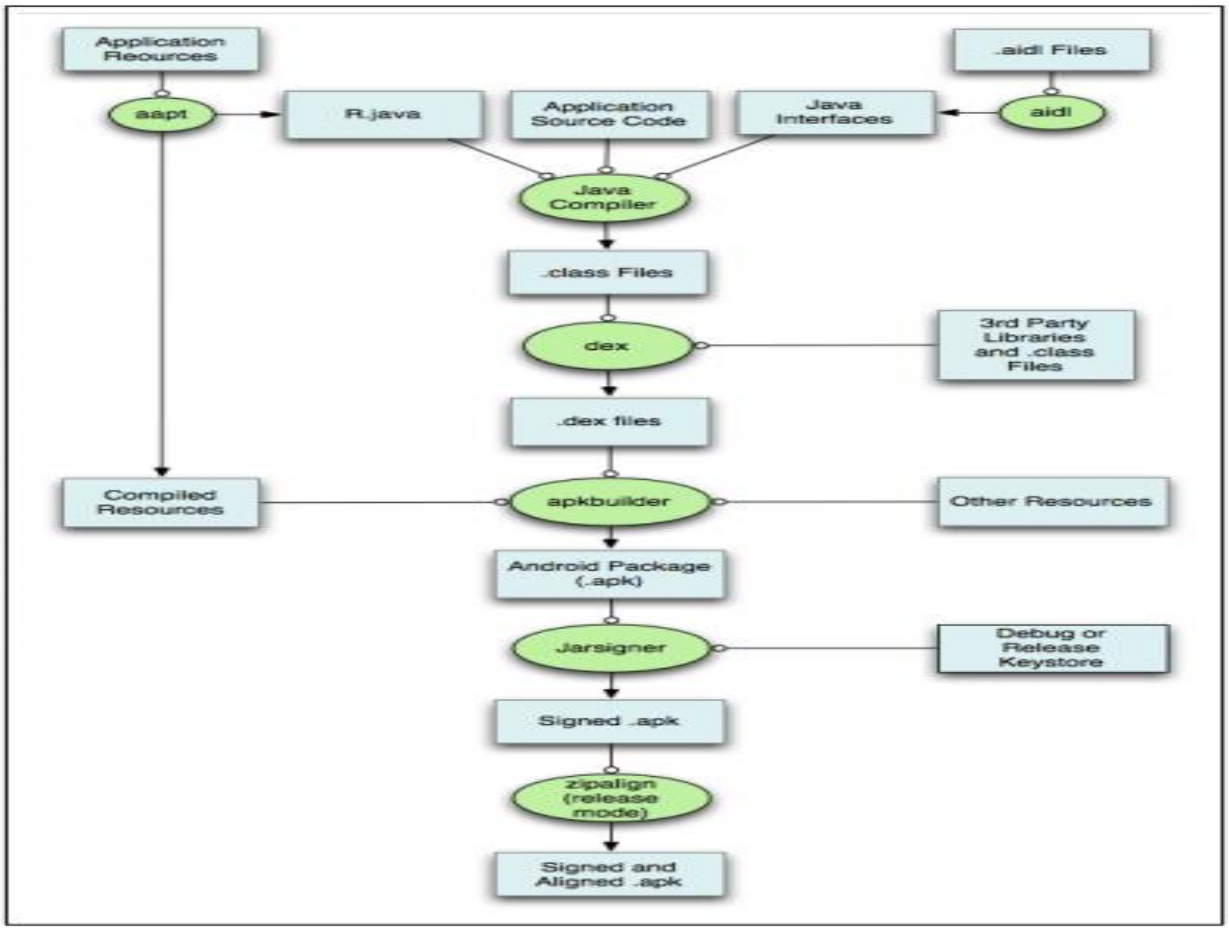

Figure 2. 3 Android Application building process [25]

2.4.1 Application components

There are four different types of core application components that are used to build Android apps. Each component is used as an interface between the app and user device [6].

***Activities*:** An activity is a door for the app to interact with the user and it represents a single page with a user interface that allows the user to communicate with the Android application. A Facebook app might have on activity that shows home, another activity to send a message or to chat, another activity to display friend requests, and another activity to read messages. Each activity is independent of another activity.

***Services*:** Services are another component of an android app that runs in the background to perform the long-running process. It does not provide a user interface. For example, a service might update the post on Facebook while the user is in a different app or in a different component. The service component can be started by another component like activity and broadcast receiver.

Broadcast Receiver: This component allows the system to deliver the events to the app outside of regular user flow. A broadcast receiver is a well-defined entry into the app that enables the system to deliver broadcast even to applications that are not currently running.

Content Providers: The Content provider is the component which is used to manage data shared between applications. It provides the applications to perform a query or modify the data in SQLite database. For examples users, contact information is managed by content providers.

2.4.2 Manifest

Every Android application have a manifest file called AndroidManifest.xml, which give information about that application to the system. Any app components that cannot declare in AndroidManifest.xml can never run [6]. In addition to declaring app components manifest does many things, such as the following:

- Identify user permissions required by the application
- Declares minimum API level required by the app, based on which APIs the app uses
- Declares hardware and software features used or required by the app
- Declares API libraries the app needs to be linked against

User permissions are declared as follow in AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.googlecode.gtalksms" android:versionName="0.3.1" android:versionCode="5">
    <uses-sdk android:minSdkVersion="5" />
    <application android:icon="@drawable/icon_green" android:label="@string/app_name">
        <activity android:name="com.googlecode.gtalksms.panels.MainScreen"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    ..................................................................
    ..................................................................
    ..................................................................
    </application>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
</manifest>
```

Figure 2. 4 AndroidManifest

2.5 The vulnerability of Android platform

Android malware applications are being widely distributed through the open market. The reason that the proliferation of malware per year is that Android platform supply functions that are easily accessed by allowing various forms of attacks to occur based on its open and portable features.

Malware attackers hide malicious code inside a benign application that appears normal in order to distribute it to smart work devices of common users. When the user tries to access the application, the malware is executed and the attacker wins illegal access to the device without the user knowing [26] [27].

2.6 Malware

Malware is a software which is modified by malicious code or simply malware can be defined as malicious code (or software) which disrupt system operation, allowing intruders access to private and sensitive information, as well as the ability to spy on the mobile devices. Malware is a program that appears itself as normal software.

2.6.1 History of malware

The first mobile malware called "Timofonica" have been developed by Brazilian software engineer Marcos Velasco which was identified by antivirus labs in Russia and Finland in 2000. "Timofonica" affects mobile device by sending SMS messages to GSM through internet SMS gate of the Movistar mobile operator. In June 2004, a worm called Cabir was originated by Vallez who worked as a part of 29A group of virus writers. This type of malware was developed to infect mobile devices and spread through Bluetooth as a. sis package.

In March 2005, the worm called commwarrier-A which had been infecting Symbian series 60 mobile phones was reported by anti-virus labs. Commwarrier-A replicates itself through the phones Multimedia Messaging Service (MMS). This specific worm infects the mobile device by sending copies of itself to other phone owner listed in the phone user's address book [28]. As Kaspersky Lab reported, in 2010 the first malicious program was designed by Trojan which is called TrojanSMS.AndroidOS.FakePlayer.a. This type of malicious program is classified as a Trojan SMS that affects smartphones running on Google's Android operating system.

Nowadays, different antivirus software companies like AVG, Trend Micro, Comodo, Kaspersky Lab, and Softwin are working to adapt their programs to the mobile operating system that are almost at risk. Meanwhile, operating system developer try to detect their customer (or user) from the spread of malicious code

2.6.2 Types of Android Malware

The most Android malware can be classified into two types such as SMS Trojan/ Fake install and Spyware/Botnet.

SMS Trojan/ Fake install: This specific malware is the major Android malware category that pretends to be an installer for licensed software and call the users into installing them on their

devices. Android.FakeInstaller is spread among a large number mobile malware family [29]. McAfee antivirus company analyzed that SMS Trojan/ Fake installer family is most prevalent Android malware which sends SMS messages to charge rate numbers, besides the user's consent, the bank itself off as the installer because of a licensed application.

The application having this type of malware may request service agreement during execution and, once the user gives privilege, it sends premium related text messages. Since SMS Trojans/ Fake installers use only single main activity component with a button that initiates sending SMS message, it is easy to implement. This type of Android intrusion is also referred to as toll fraud. Easy implementation and high profit make toll fraud applications popular among malware authors.

Spyware/Botnet: Another type of Android malware is categorized as spyware that has capabilities to forward private data to remote server. Not only forwarding this type of malware could also receive commands from the server to start specific activities in which case it is part of a botnet.

2.7 Machine Learning in Android Malware Detection

2.7.1 Overview of Machine learning and its importance

Machine learning is a subfield of computer science that uses a statistical method that enables the computer the ability to learn with data without being explicitly programmed. The early machine learning algorithms were perceptron (afterward called Neural Networks), decision tree, and rule learners like AQ [30]. Machine learning was gradually developed from the study of pattern recognition and computational learning theory [31]. Machine learning investigates the study and interpretation of algorithms can learn from the Training data and make a decision on data [32].

"Machine learning is an application of artificial intelligence (AI) that provides a computer system to have the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves." [33]. The basic logic of Machine learning is to construct an algorithm that takes input data and use statistical analysis to decide an output value within an acceptable range.

Machine learning is carefully related in accordance with yet fast overlaps with computational statistics; a moderation so much also specializes among decision-making. It has passionate ties to

mathematical optimization, which hand over methods, theory and application domains in accordance with the field. Machine learning is employed within a range of computing tasks where designing and programming obvious algorithms are infeasible [32].

Machine learning is important to learn complex problem that needs for learning. Machine learning algorithms are an analysis of very large and complex datasets such as astronomical data, turning medical archives into medical knowledge, weather prediction, and analysis of genomic data, web search engines, and electronic commerce [34].

One disadvantage of the programmed tool is their unable to bend, to solve this problem machine learning algorithm are used which adapts to their input data. By nature, machine learning is adaptive discipline to changes in the environment they interact with.

The general framework of machine learning is illustrated in figure 2.6:

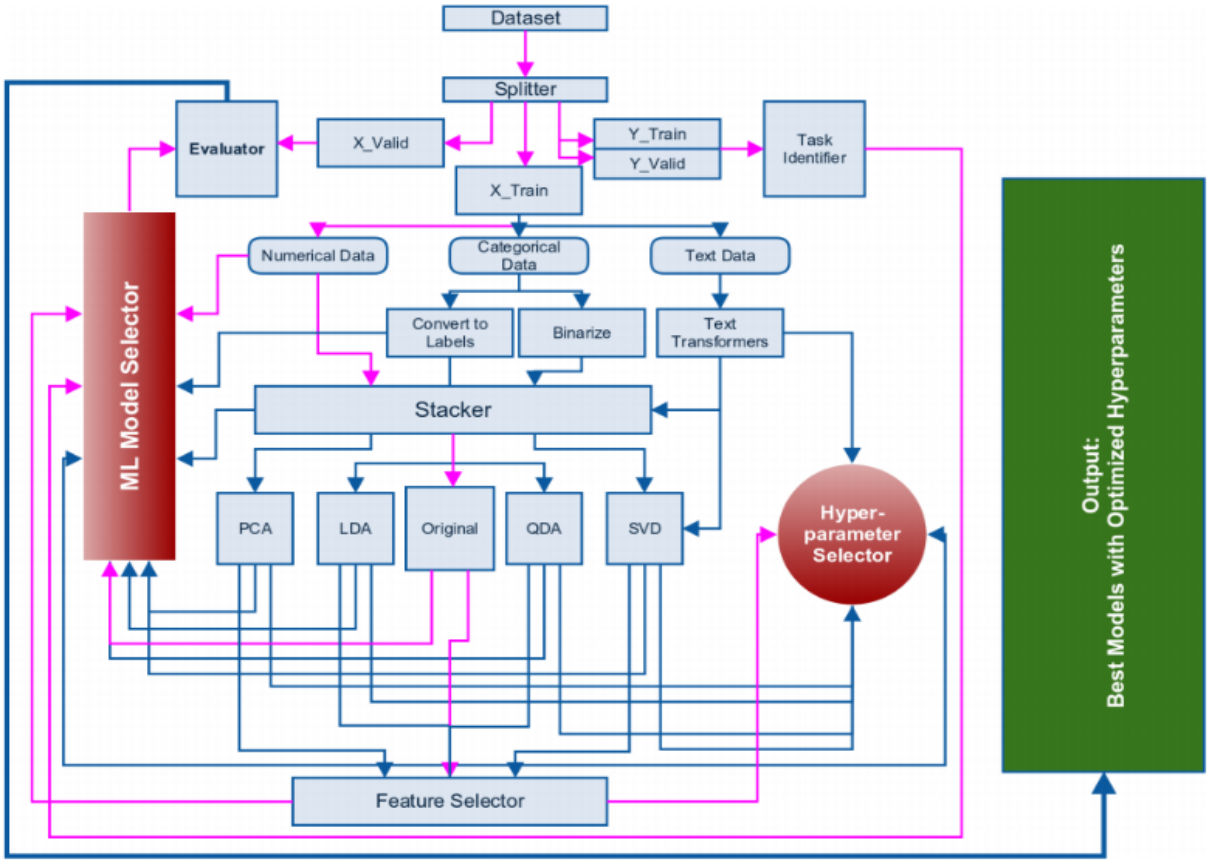

Figure 2. 5 Machine learning framework [35]

The path which is shown by pink color on the above framework is the most common paths followed in modeling machine learning. After the feature is extracted and selected the feature to tabular format, building classifier model is the next step in machine learning framework.

The first begging step in modeling machine learning classifier is looking at the labels and must know if the problem is binary classification, multi-class, multi-label classification or regression. After the problem is identified, the data set is split into two different parts: a training set and testing sets as shown in Figure 2.6.

The general view of machine learning classification model looks like as follow:

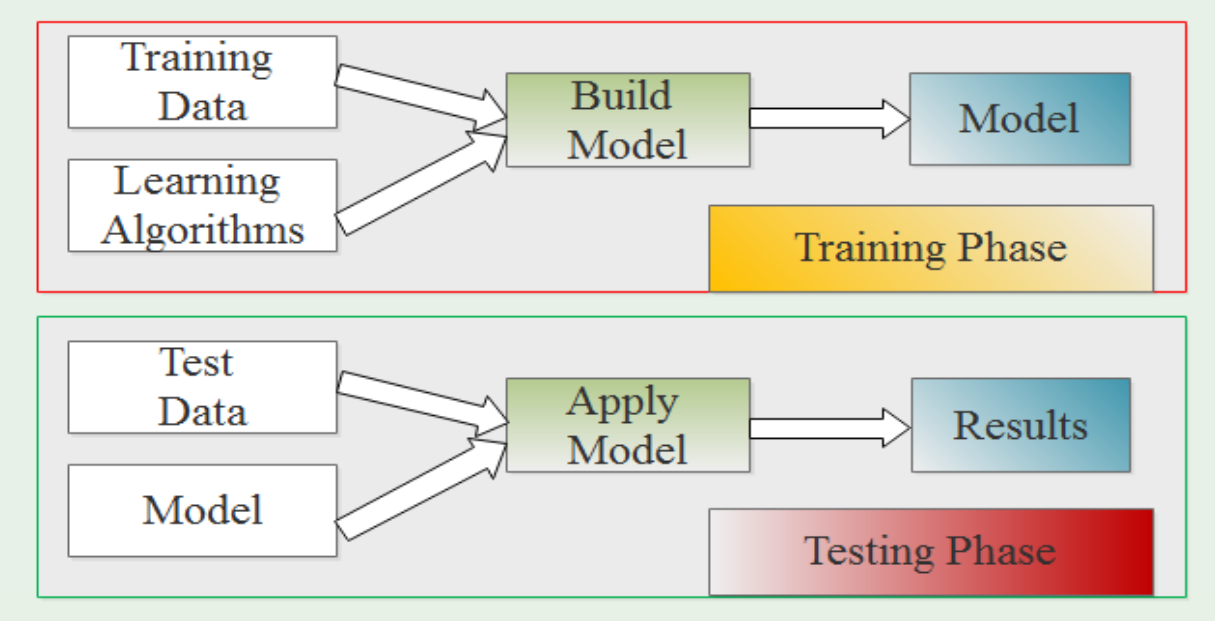

Figure 2. 6 The two phases of Machine learning classification algorithms

In the next step, different variables in the data are identified. Usually three types of variables are considered namely, numerical variables, categorical variables, and variables with text inside them. If our variable is a numerical variable, no need of any kind of processing and thus start applying normalization and machine learning models to these variables. The categorical variables are handled in two ways: convert the categorical data into labels and convert the labels to binary variables. On the other hand, the text data is transformed by text transformer.

The stacker module is the next step. Stacker module is not a model stacker but a feature stacker. The different features after the processing steps described above can be combined using the stacker module. Then, the feature extraction algorithm is selected to decompose the data. After evaluating

more performance of the models, we tend to move to the scaling of the datasets, in order that we are able to evaluate linear models too. The normalized or scaled features can then be sent to the machine learning models or feature choice modules.

2.8 Machine learning concepts in other Fields

Data mining, artificial intelligence, and machine learning are symbiotically related fields, which are used to produce more efficient and sensitive outputs [36]. These three disciplines are intertwined and extend over which are used to access data, manipulate data and analysis data that it is almost to produce boundary among the three as follow:

Data mining is an important tool for both machine learning and artificial intelligence which interpret any kind of data and prepare it for analysis and learning. Data mining is used to become aware of previously unseen patterns and relationships from the large dataset and extract accurate information which is used as an input variable for prediction use case [37].

Machine learning and data mining are intertwined in many ways: data mining uses many machine learning algorithms for a different purpose. On the other hand, machine learning uses data mining techniques like unsupervised learning at preprocessing steps in increasing the accuracy of classification. Even though these two disciplines are overlapped, there exists much of the confusion between two areas: the key task in machine learning is to reproduce known knowledge, while in data mining the key task is the discovery from unknown knowledge.

According to Kajaree et al, "artificial intelligence is defined as machines those having the ability to solve a given problem on their own without any human intervention" [36]. Artificial intelligence is a field of analytics that goes more extensive than machine learning, produce the system with the ability to reason. AI includes machines that can perform human intelligence tasks such as planning, understanding language, recognizing objects and sounds learning and problem-solving.

Machine learning is a branch of analytical in which it trains the data to achieve artificial intelligence. Marco Varone et, defining machine learning as, "an application of artificial intelligence (AI) that provides a computer system to have the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves." [33].

Although machine can learn from experience most probably similar to human learning and animal with time and experience, solving the problem of machine learning using human learning method is not yet much promising result due to the fact of human psychology understand not remains for a long time [36].

2.9 Types of Machine Learning

Machine learning techniques are categorized into the taxonomy, based on the output provided. There are six common machine learning algorithm types such as supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning, transduction, and learning to learn [34].

2.9.1 Supervised learning

Supervised machine learning algorithm is an algorithm to generate a function that map inputs to output based on labeled data. In this algorithm, the learner is required to predict the given input

into appropriate classes based on the input-output data [34]. Classification and Regression are the most common supervised learning techniques.

$$Y = f(x) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots eq\ 2.1$$

In supervised learning, three things were considered: the dependent variable, independent variable, and function that is used for mapping and predict the effect on the output [38].

2.9.2 Unsupervised learning

Unsupervised learning is another type of machine learning algorithms which can search for invisible pattern in data without labeled examples. Most of the time this type of learning is used for clustering techniques. It tries to disconnect information from the input with an intrinsic feature extraction or the total amount of input data [39]. In unsupervised learning, the data can be categorized without any trained data or column name, only input data are available to learn from

2.9.3 Semi-supervised learning

Both labeled and unlabeled data are combined in this learning type in order to improve the performance of generalization and to perform an otherwise supervised and unsupervised learning

task. Since this learning type uses both labeled and unlabeled training data, predictor learned that predict future test data better than predictor learned from labeled one.

2.9.4 Reinforcement learning

Reinforcement learning is another type of machine learning which determine the interaction of the agent with the environment based on feedback that might be positive or negative. The objective of this learning in determining action within the specific environment is improving positive feedback called a reward.

Unlike other machine learning or (supervised, unsupervised and semi-supervised), reinforcement learning does not depend on the training data, rather it is based on action or interaction happened between agent and environment. The interaction happens with the environment but not with data [40].

2.9.5 Transduction

Transduction learning is a type of machine learning which is similar to supervised learning, that does not draw function, but it simply tries to predict new output based on inputs, training outputs, and new inputs [34].

2.9.6 Learning to learn

This type of learning is also called meta-learning which exposed a large number of learned tasks previously and learn its own inductive biased [34].

2.10 Classification Algorithms

Classification is a supervised learning concept in which the problem is identified and classified into their appropriate categories, on a given trained dataset containing instances whose category membership is specified. In supervised learning, the classification approach is based on given data (or input data) to the computer program that learns from it.

A linear classifier algorithm has been used to achieve the objective of the study. Linear classifiers are a subtype of supervised algorithms that classify or group the items into their class type. This type of classifier performs its classification based on linear predictor function combining weight with the feature vector.

The aim of supervised learning is to build Machine learning classifier that is used to classify unknown samples with proper labels, malicious or benign class in case of this study. They serve as the most important part of mobile malware detection with the features. Compared with the data mining technique, machine learning is the most popular technique used in classification. In table 2.1 several machine learning-based classification algorithms were briefly described. For more detail, you can refer to [41] [42].

Selecting appropriate classification algorithms according to the goal of malware detection is very important since it impacts detection accuracy and performance. When training dataset is too small, Naïve Bayes (NB) or K means is selected. If we need to observe accurate possibilities of each class, Logistic Regression is better to choose. If there are many types of extracted feature to be considered, Random Forest could be a good classifier. In particular, a large number of experiments are normally needed for setting appropriate parameters for finding the most effective results.

Table 2. 1 Frequently Used ML Classification algorithms in malware detection

Algorithms	***Advantages***	***Disadvantages***
Naive Bayes	Even though the conditional independence assumption rarely holds true, NB models actually perform surprisingly well in practice, especially for how simple they are. They are easy to implement and can scale with your dataset.	Due to their sheer simplicity, NB models are often beaten by models properly trained and tuned using the previous algorithms listed
Support Vector Machines (SVM)	High accuracy and speedy, deal with high dimensional and non-linear problems;	Heavy storage burden, require to properly choose kernel function and repeatedly call parameters;
Decision Tree	They are robust to outliers, scalable, and able to naturally model non-linear decision boundaries thanks to their hierarchical structure.	Easy to overfitting;
Random Forest	Overcome overfitting, deal with high dimensional data	Accuracy depends on tree number, Sensitive to unbalanced sample set.
K-means	Rapid to converge, self-optimize capability;	Required to predefine value of K, sensitive to outlier;
K Nearest Neighbors	High precision and accuracy, nonlinear classification, no assumption of features;	Sensitive to unbalanced sample set, heavy computing burden;
Logistic Regression	Outputs have a nice probabilistic interpretation, and the algorithm can be regularized to avoid overfitting	weak to handle high dimensional data;
C4.5 (=J48)	Easy to understand generated rules, high accuracy, overcome overfitting	Multi-visit to data lead to low efficiency.

2.11 Related Works

2.11.1 Android Malware Detection Applications

A lot of powerful malware detection applications are available on the market such as Bitdefender, Kaspersky, Norton, 360 SafeGuard, Tencent Security Manager [43]. Most of them support personal laptop system like MacOS and Window platform; some of them also support mobile systems like iOS and Android. Since this study concern with Android malware detection, some important android malware detection applications namely; 360 Boost, AVAST, AdroHelm, Avira, TrustGo are collected. Most of them store the signature of the application on the server and try to compare the hash value for all applications.

2.11.2 Android Malware Detection Frameworks

In previous work, a lot of frameworks that used to detect android applications malware were developed by many researchers. Most of them use a permission-based approach; and some of them use behavior-based techniques such as API calls, system calls or function call and methods.

Abas Aman develop a machine learning model by applying software datasets before preprocessing and feature reduction in machine learning algorithms. The researcher has given emphasize on a model development for software defect prediction using feature reduction method plus prototype system for identification of faults [44].

Fetulhak Abdurahman develop lightweight security auditing tool for android mobile phone. a host based lightweight security examining tool that suits asset obliged mobile phones as far as low storage and computational prerequisites was designed and implemented in this study. Experimentally, the researcher shows that the proposed solution detection rate is 96.73%. Random Forest machine learning algorithm is used during the development of proposed solution [45].

Zhaoguo et al develop a novel android malware detection method based on behavior chains framework. They propose the method that combines static analysis and behavior chain in order to detect zero-day malware. In this work, APKtool and Androguard tool were used to extract smali codes and generate API call graph from an app respectively. After the API call graph is generated DroidChain model build an incidence matrix, and turns to an adjacency matrix. Afterward, from

the adjacency matrix, it calculates the accessibility of the matrix, then by verifying calling relations for all actions by using the Wxshall-extend algorithm DroidChain detect Android malware [46].

Fauzia et al present a novel permission and intents-based framework for identifying Android malware apps. In this framework, the first stage is analyzing Android manifest and extract permission and intents, then the type of permission and intent measured. In the second stage, the vector dataset is extracted which is used for classification. At the last stage, the classifier (supervised machine learning algorithms) takes a vector dataset as input and classifies the dataset. They use statistical and machine learning methods to validate the proposed detection framework [47].

Kabakus et al. [48] develop a permission-based Android malware detection framework that uses static analysis malware detection system. APK Auditor consists of three components: (1) An Android client, which presents an analysis request, showing whether the application is benign or not. (2) A signature database, (3) a central server that communicates with both the Android client and the signature database and handles the analysis process. APK Auditor software architecture has two phases; the training phase and evaluation phase. In the training phase, both benign and malware applications are submitted to the train system (central server) and store the analysis result on the signature database. In the evaluation phase, the application is sent to be analyzed to a central server. Then, from analyzed result, the classification result is reported to the user. APK Auditor uses static analysis techniques which based on permissions in order to extract feature for Android applications.

A novel android behavior-based malware detection technique comprising a lightweight client agent and server analyzer is proposed. In this framework, the server analyzer generates the signature for every app based on the system call request of the app and normalize the generated signature to improve the accuracy. Unlike another signature-based Android malware detection that generates signatures based on the static attribute, this approach employs behavioral attributes such as requests for reading files or network access to generate unique app signatures and uses the signature normalization techniques. They use a python tool to develop the prototype for the framework [49].

Shifu et al. developed an Android malware detection system called DroidDelver which extract API calls from smali code and they apply the Deep Belief Network on generated block code to detect

unknown Android malware. DroidDelver follow five steps to detect malware application: (1) Unzip or decompile Android apps, (2) API call extraction, (3) API Call Block generation, (4) Classification, and (5) malware detection. In their experiment, they compare the performance of DBN with other machine learning algorithms such as SVM, ANN, NB, and DT. And the experiment shows that Android malware detection performance is greatly improved by using DBN [50].

Diaxon and Mishra [51]. Proposed leveraged battery draining and power usage to discover malware existence. They studied battery behavior after the device is infected by malware to form a model to detect Android malware. But this system has limitation concerning to detect some unsophisticated malware.

Tong and Yan proposed and implemented a hybrid Android malware detection method [52]. This method dynamically monitors and collects execution data of both malicious and benign applications in order to generate a malware pattern set and a benign pattern set. The execution data are individual system calls and sequential system calls (with different calling depth) that are related to the access of files and networks. The malware detection is performed at a server. In order to judge an unknown app, its runtime system calls are collected in a dynamic way in order to extract the patterns for the app. Next, the unknown app patterns are compared with the patterns in both malicious and benign pattern sets in order to classify the unknown app.

Wei-Ling Chang and Hung-Min Sun proposed behavior-based Android malware detection method [53]. They collect different Android applications behaviors such as network activities, file read/write and permission as the feature data and use different machine learning algorithms to classify malware and evaluate the performance.

Mohsen et al propose a method to generate behavior-based mobile malware signature and method for comparing generated signature. They developed MODroid model which consists of server and client. Whenever a new application is installed, the client agent receives the installation request and check a local database. If the record is not available, the agent will send the SH1 hash checksum of APK file to the server. Then the server request agent for APK; after agent submits APK to the server, APK is run in the emulator to obtain system call request. From normalized system call requests, the server generates the application signature and obtain correlation against the signature in the database which reported to the agent [54].

2.11.3 *Summary of Related works*

The main methods used in Android malware detection developed since 2013 is discussed. The existing work is compared regarding the type of detection, applied classification algorithms, classification accuracy or performance evaluation value, the place of analysis, and the threats that can be disclosed and overcome. Most of the time machine learning algorithms such as Naïve Bayes, Classification and regression tree (CART), J48 Decision Tree, Random Forest, K Nearest Neighbors, K-means, Support Vector Machine, and Logistic Regression are used to develop classifier model. This work concludes and suggests the further work that, Naïve Bayes(NB) and support vector machine (SVM) have better classification accuracy and detection performance.

Table 2. 2 Summary of related works

Ref	*Techniques categories, classification algorithms, place of analysis, advantage, and disadvantage, etc.*					
	ToD	***CA***	***P.A***	***Threat***	***Value***	***Pros & Cons***
[54]	SIG	z-score and median	Ser, Dev	All kinds	N/A	Since it uses both local database and server database, detection performance is good. Time-consuming is the drawback of this method.
[49]	SHA-256	SVM	Dev, Ser	All kinds	95.2%	A better method of detecting code obfuscation. Only detect zero-day malware. Time-consuming.
[55]	SPE	NB, J48, SVM	Ser	Privacy	98.4%	A novel fusion method and good performance, heavy computation burden.
[56]	SIG	VF2	Ser	Resist obfuscation ma-payload	86.36 %	New methods of detecting obfuscating with low accuracy, without privacy preservation.
[52]	A	M	Ser	All	91.76 %	A hybrid and generic method, not real-time, high memory consumption
[46]	A	Wxshall-extend	Dev	Privacy leakage	73-93%	Good performance on zero-day malware detection, difficult to detect unknown malware.
[51]	A	M	Ser, Dev	All kinds	N/A	Unable to detect some sophisticated malware, good FPR performance
[47]	PER, I	NB, DT, RF, MP, SMO, Dt	Ser	All kinds	99.8%	Preferable for detection of colluding applications, high memory, and time consumption

[48]	PER	LR	Ser	All kinds	88.28 %	Hardware resource of the mobile device is not consumed, a new approach to accessing potential maliciousness of android application. time-consuming and availability is the issue in this study
[57]	SIG	SA	Dev Ser	All kinds	99%	Combine both static and runtime features to design a client-server detector with 99% accuracy, time-consuming and inconsistency availability of the server.

Based on the above literature review, Android malware detection methods were compared in table 3 with regard to the type of detection, classification algorithm, place of analysis, the threat to overcome and classification accuracy. The general advantage and disadvantage have been commented of each work. Based on the comparison of the existing work, the following concepts were concluded:

- Most of them use the behavior or Anomaly of the android application to detect the application malware.
- Most of the existing work performs the detection on the third party such as a server, rather than detecting on the user device.
- Most of developed Android malware detection system uses machine learning algorithms such as NB, SVM, DT, LR, J48, and RF algorithms for the classification purpose.
- The method that uses the hybrid analysis (both static and dynamic analysis together) have better detection performance.

Long Wen, and Haiyang Yu develop Android malware detection system by the aid of machine learning. Since they use client-server approach and machine learning approach, this method of Android malware detection is almost similar with our study. Even though the similar approach is used, our study contributes by solving the problems that found in Long Wen, and Haiyang Yu s' study. Availability and response time are the issue in their study [49].

Different from the existing Android malware detection methods, our system uses cloud-based approach and automatic malware detection module which is used to check the database and compare with the existing records (or documents in case of Mongo DB) to improve the availability of the system and increase the response time.

CHAPTER 3

3 The Research Methodology

3.1 Overview

This chapter explains the research methodology and the process to develop an Android malware detection system using machine learning. It starts with the explanatory information about detection methods, classification and feature selection algorithms and the justification for the chosen method and algorithms. Then the following section clarifies how the data were collected and processed to be used in Android malware classification.

In this study, three different phases have been conducted to develop Android malware detection such as basic pre-processing phase, analysis phase, and detection phase was conducted. Basic pre-processing phase is a procedure by which research problem is formulated and the gap from existing work was identified using a systematic literature review.

In Analysis phase, training data is constructed from datasets (or collected application samples) to make predictions on Android app whether it is malware or not. Here, APK feature has been extracted to prepare training data by using back engineering (or reverse engineering) process, by which zipped APK file was decompressed.

The analysis phase is also called training phase in which features (Permission and API Calls) are extracted from AndroidManifest.xml. Binary vector is generated from extracted features, and the features are filtered with using attribute selection algorithms.

In the third phase, the proposed framework for Android malware detection has been implemented by using python programming language and three different supervised machine learning algorithms such as SVM, NB and DT were tested to evaluate the classification performance of proposed system.

3.2 Android Malware Detection Methods

Currently, various Android malware detection techniques were introduced to detect and prevent mobile users from attack. Many researchers [48] [52] [57] have been developing different Android malware detection techniques in different ways. They applied three different dependency techniques such as behavior-based, signature-based and specification-based deployed with three different analysis techniques namely: static analysis, dynamic analysis, and hybrid analysis. Figure 3.1 demonstrates Android malware detection methods.

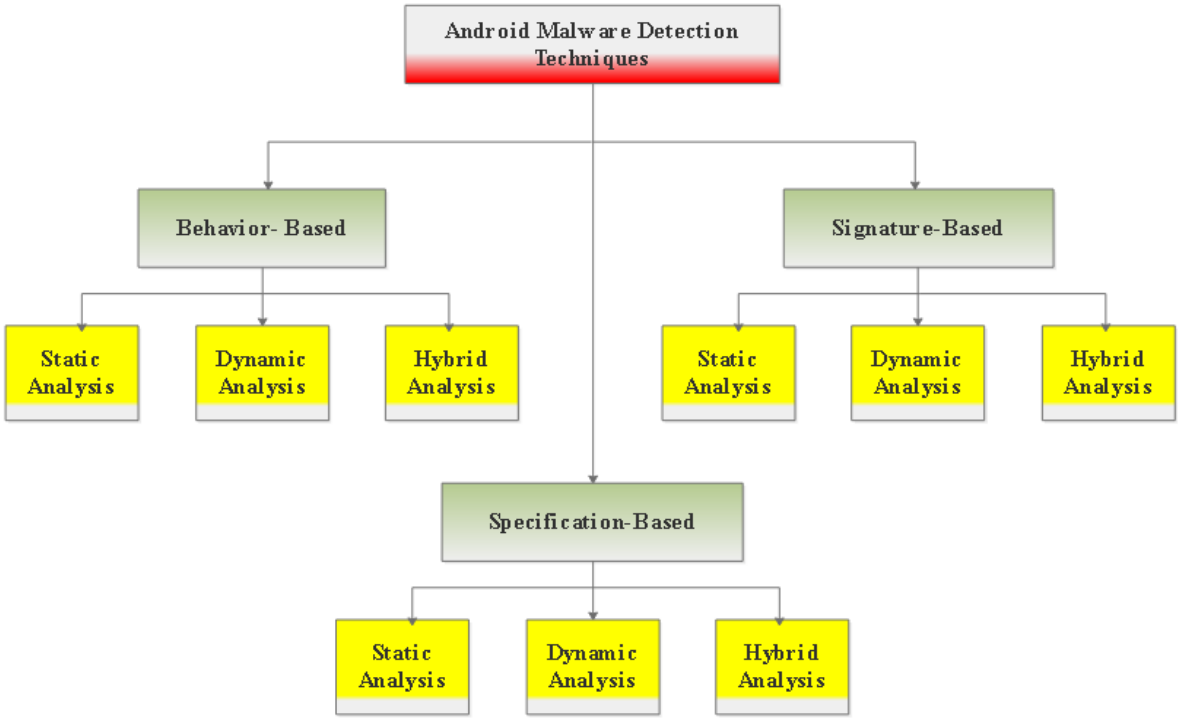

Figure 3. 1 Structure of Android malware detection methods [58]

The above android malware detection techniques and their comparison were described in table 2.1. The survey of Android malware detection illustrated in table 2.1 shows that signature-based with hybrid analysis [57] [49] provide high classification accuracy and false positive rate. Signature-based (SHA-256 signature based) method has been selected for this research work because the objective of this study proposes android malware detection based on the SHA-256 signature.

3.3 Development Tools

Different types of tools are used in this study to design and implement the proposed solution. UML modeling tools; reverse engineering tools; Android app developing tool and classifier modeling tool is used to provide a solution and to solve the research questions. These tools are explained below.

3.3.1 Design Tool

Edraw Max tool is a type of design tool which enables the software designer to reliably create and publish many kinds of diagrams to represent an idea. This design tool provides many features such as flowcharts, workflows, software, UML diagrams, Business diagrams, database and ERD and no other diagram software gives you all these features [59].

3.3.2 Reverse Engineering Tool

Different components such as manifest.xml, classes.dex and other resources (Images and layout files) compressed to build Android APK file. The APKtool tool is used decompile this file to its subcomponents. This tool is used for Android application penetration testing [60].

3.3.3 Implementation Tool

Android studio is one of open source development software which is used to develop Android mobile application. Features like specific IDE, automation possibilities and faster response of Android studio [61], tend us to adopt this tool. Some extensive support libraries are used to develop classifier. Python has large standard libraries and it is easy to develop web services [62].

Why python?

According to the state of the developer nation Q1 2017 report, Python is prioritized the most by those for whom data science is the first profession or field of the study (38%). This indicates that Python has by now become an integral part of machine learning language-it has evolved into the native language of data scientists [63]. Table 3.1 illustrates the comparison result of the machine learning programming language.

Table 3. 1 Machine learning languages statistics

The parameter to pick the right language	***Programming languages***				
	Python	***C/C++***	***Java***	***R***	***JavaScript***
Popularity	57% 33%	43% 19%	41% 16%	31% 5%	28% 7%
Fraud Detection	26%	12%	22%	9%	-
Sentiment Analysis	44%	9%	15%	11%	2%
Web mining	37%	-	-	-	-
Network security and cyber-attack detection	24%	26%	23%	-	-
AI in games	26%	29%	-	3%	-
Data Science	26%	-	-	7%	5%
The area where each language is practiced the most				***Prioritized by***	
The area where each language is practiced the least				***Used by***	

As shown on the aforementioned statistics, python is the most prioritized machine learning language, particularly for a data scientist. This shows that Python language becomes an integral part of data science. Since this study is about predicting the class of unknown application based on datasets which are more related to data science field, the machine learning algorithms that mostly practiced in data science area are used.

Prioritization of Python in machine learning scientist who works on sentiment analysis (44%) is another motivator that tends us to adopt Python programming language as front-end in case of this study.

3.4 Dataset Description

3.4.1 APK files

The normal applications were collected from Google Play Store for evaluation. APK file is a format created by the Google, which is used to distribute and install an application on the Android operating system. Contagio Mobile is the repository of malware samples to provide access for the researchers to collect sample of malicious applications.

3.4.2 Data Sampling

The number of available applications in the Google Play Store recently placed at 3 million applications in June 2017 [11]. The size of our data which is our target population of the study is the total of 3 million available applications in the Google Play Store.

Normal Application Dataset: By using stratified random sampling technique, the applications are divided into categories and we take the sample of normal applications randomly as shown in Table 3.1. we have collected 1,608 available applications from all the categories found on the market.

Table 3. 1 The number of applications in each category

Category	*Count*	*Category*	*Count*
Arcade	69	**News &magazines**	120
Books and References	74	**Personalization**	102
Brain	42	**Productivity**	65
Business	44	**Sports**	89
Communication	42	**Social**	100
Entertainment	46	**Weather**	34
Tools	72	**Widget**	23
Media and video	87	**Shopping**	45
Finance	43	Card Games	88
Games	60	Comics	90
Transportation	37	Education	100
Photography	34	Medicine	102
Grand Total	1,608		

Malware Dataset: The Contagio Mobile Dump [64] was used as a source of our malicious dataset. This site has the set of malicious applications uploaded by the researchers and anyone can collect the samples from their database for experiment purpose. For our experiment 200 malicious applications have been collected manually from this site. The total number of applications collected for this research work is shown in Table 3.2.

Table 3. 2 The total number of applications in our data set

Applications	**Count**
Normal	1,608
Malicious	200

3.4.3 Reverse Engineering

An Android APK file is a zipped directory, where a manifest file, classes.dex file, as well as resources such as images and layout files, are compressed in it. Then, APK decompiling generates source code from an executable code. Figure 3.2 shows, the steps to decompile the zipped APK files.

Reverse Engineering is a process of analyzing the program code in order to determine it from any vulnerability or any errors. It is the process of generating or decompiling source code from executable code. Reverse engineering can be used to examine the functioning of the app and evade security bugs, etc. This technique can be stated as a method of modifying a program in order to make it behave in a manner that the reverse engineer desires.

JoanyBoutet has quoted Schwartz, saying, “Whether it's rebuilding a car engine or diagramming a sentence, people can learn about many things simply by taking them apart and putting them back together again. That, in a nutshell, is the concept behind reverse-engineering - breaking something down in order to understand it, build a copy or improve it “ [65].

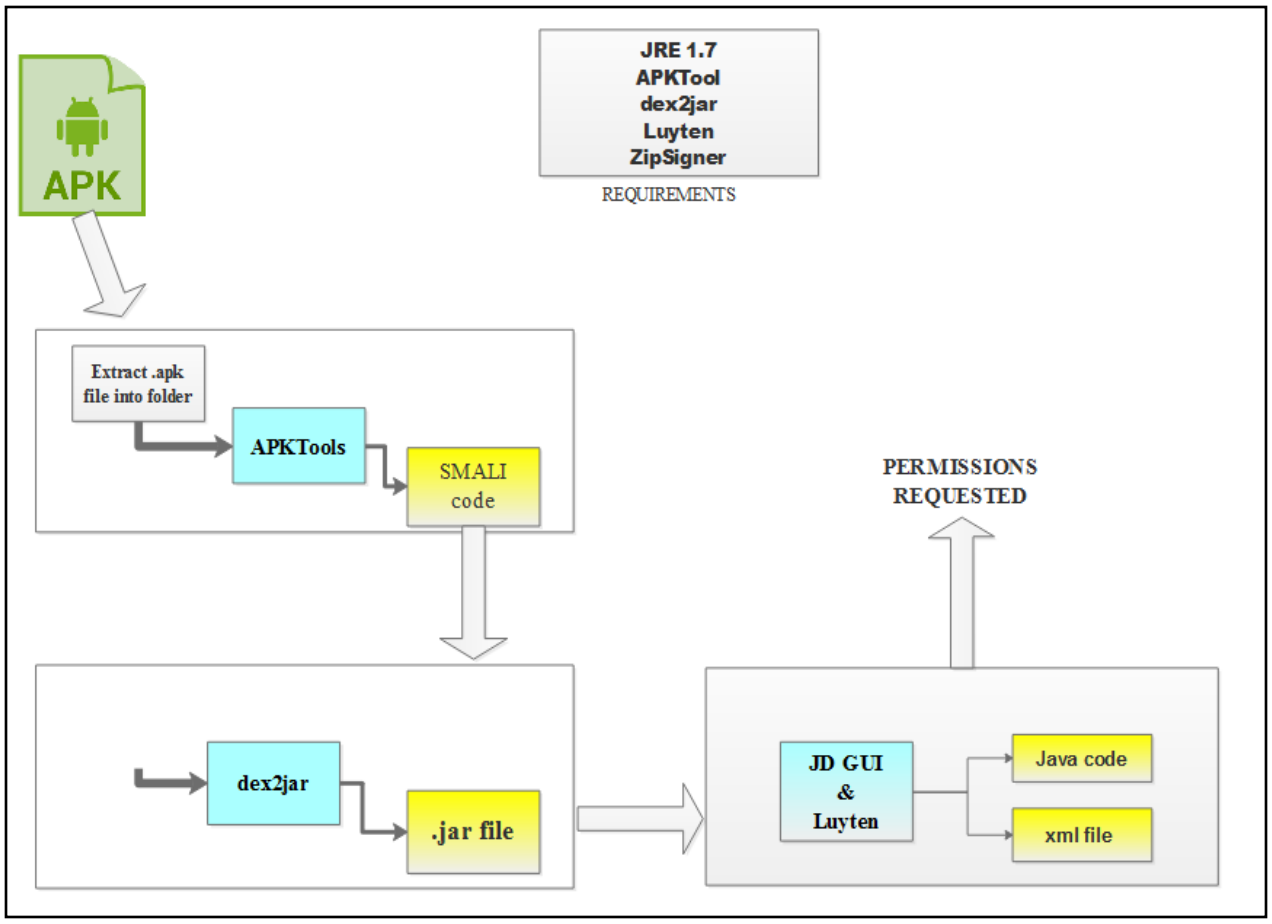

Figure 3. 2 Reverse Engineering Flowchart

3.4.4 Feature extraction

After APK file was decompressed by the back-engineering process, the signature behavior of application such as permission requests, API calls, and function calls features have extracted. In case of our study, the extracted features were used as independence variable which is input for the machine learning classifier model.

3.4.4.1 Permissions

Requested permission by android applications to access sensitive user data (such as contacts and SMS) and some system features (such as Camera and Internet). The app permissions are included in <uses-permission> and tags in application manifest file called AndroidManifest.xml. Even though this permission publicized to protect the privacy of Android users, Android attack authors develop dangerous permission that requires user agreement based on the Android versions.

Normal permissions: Normal permissions cowl areas the place your app needs to access information or sources outdoor the app's sandbox, however, the place there is a very little risk to the user's privacy or the operation of different apps. For example, permission to set the time quarter is an everyday permission. If an app broadcasts in its show up that it wishes a normal permission, the system routinely supplies the app that permission at install time. The machine would not prompt the user to supply everyday permissions, and users can't revoke these permissions.

Dangerous permissions: Dangerous permissions cowl areas wherever the app needs private data or resources that involve the user's non-public info, or might doubtless have an effect on the user's keep information or the operation of alternative apps. For instance, the flexibility to scan the user's contacts may be a dangerous permission. If associate degree app declares that it wants a dangerous permission, the user has got to expressly grant the permission to the app. till the user approves the permission, your app cannot offer functionality that depends thereon on permission.

Signature permissions: The system grants these app permissions at install time, but on condition that the app that produces a shot to use a permission is signed by the same certificate as a result of the app that defines the permission. The next permissions that third-party application scan use unit classified as signature-permissions.

3.5 Learning Algorithms

Three machine learning algorithms are selected to be evaluated under this study such as:

1) ***Support Vector Machine (SVM):*** "support vector machine learning" is a type of supervised machine learning algorithm which can be used in classification problems. In this algorithm, each data item has been plotted as a point in n-dimensional space. Support vector machine algorithm perform classification by finding hyper-plane that tell part into two classes very well.

   Given some training data D, a set of n points of the form:

   $$D = \{(Xi, Yi) | Xi \in R^p, Yi \in \{-1,1\}\} i^n = 1 \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots eq\ 4.1$$

   Where,

   The Yi is 1or -1, which indicate the class to which the point Xi belongs.

   Xi is p-dimensional vector. The researcher need to find the maximum-margin hyperplane that separate the points having Yi = 1 from those having Yi = -1. Any hyperplane can be written as the set of points X fulfilling, parameters w, b, and our classifier can be written as:

$$f_{w,b}(x) = g\,(w^T x + b) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots eq\ 4.2$$

Here, $g\,(w^T x + b) = 1$ if $(w^T x + b) \geq 0$, and $g\,(w^T x + b) = 0$ otherwise. This "w, b" notation permits us to explicitly treat the intercept term b separately from the other parameters. W represents the normal vector to the hyperplane.

2) ***Decision Tree (DT)***: Two metrics described below have been applied on the data sets to obtain the classification accuracy for decision tree algorithm.

Entropy

Entropy H(S) is a measure of the amount of uncertainty in the (data) set S (i.e. entropy characterizes the (data) set S).

$H(S) = \sum_{cEC} -p(c)\log_2 P(c)$ $\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots eq\ 4.3$

Where,

- S – The current (data) set for which entropy is being calculated (changes every iteration of the ID3 algorithm)
- C – Set of classes in S C= {malware, normal}
- P(c) – The proportion of the elements in class c to the number of elements in set S

When H(S) = 0, the set S is perfectly class (i.e. all elements in S are of the same class).

Information gain

Information gain IG (A) is the difference in entropy from before to after the set S is split on an Attribute (A). In other words, how much uncertainty in S was reduced after splitting set S on attribute on A.

$$IG(A,S) = \mathrm{H(S)} - \sum_{t\,E\,T} p(t)H(t) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots eq\ 4.4$$

Where,

- H(S) – Entropy of Set S.
- T – A subset created by splitting set S by attribute A.
- P(t) – The proportion of the number of elements in t to the number of elements in set S.
- H(t) – Entropy of subset t.

Steps:

1. Compute the entropy for data-set
2. For every attribute/feature

    2.1 Calculate entropy for the current attribute

2.2 Take the average information entropy for the current attribute

2.3 Calculate gain for the current attribute

3. Pick the highest gain attribute
4. Repeat until the desired tree obtained.

3) ***Naïve Bayes (NB):*** "Naïve Bayes" is classification algorithm based on Bayes' theorem that assumes the presence a particular feature in class is independent to the presence of any other feature. Since this learning model is easy to build and specifically useful in large datasets, it has been selected to train the model in this study. The equation below shows that Bayes theorem that provides the way of calculating posterior probability P(c|x) from P(x|c), P(c) and P(x).

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots eq4.2$$

Where,

P(c|x)->is the posterior probability of class (c, target) given predictor (x, attributes).

P(x|c) ->is likelihood which is the probability of predictor of the class

P(c) ->is the prior probability of the class.

P(x) ->is the prior probability of the predictor.

Based on aforementioned metrics for each classification algorithms, the performance of these three classification algorithms was calculated. The algorithm that has high classification accuracy and precision score is selected to develop machine learning classification model for Android malware detection.

3.6 Data Training and Testing Technique

During the training and testing of the classifier python script running on the computer have been used. The aim of this script is to collect or extract the features as possible from AndroidManifest.xml. The script was used to:

1. Take as input the training APK files (or manifest files) in the training folder and testing APK file (or manifest files) in the testing file.
2. Output the collected features in CSV file format.
3. Train and test classifiers using the collected feature vectors in step-2

The Training and Testing data folders contain both normal and malicious applications. To train the model classifier we used the manifest files found in the Training folder and manifest files in the Testing folder are used to test the model. For training phase, we use 108 normal applications and 145 malware applications and for testing phase we use 600 benign and 45 malware applications.

3.7 Conceptual Framework Design Method

The Android malware detection based on machine learning framework has been designed that analyze Android applications based on signature value and permissions requested using machine learning algorithms and finally donate classification result. Specifically, the proposed framework for Android malware detection designed to evaluate the performance of supervised machine learning algorithms such as SVM, NB, and DT, and then give some decision regarding performance.

3.8 Conceptual Framework Implementation Method

The Android malware detection based on machine learning implemented based on proposed solution architecture designed that illustrated in the proposed solution section. Client-server approach Android malware detection implemented using Android-studio integrated with python.

3.9 Evaluation Method

The Android malware detection developed by this research work, run on user mobile devices particularly Android devices and integrated with the server. Testing has been performed by the user by using a sample of malicious application and normal application file. Based on the detection result the performance of classification algorithms was evaluated.

CHAPTER 4

4 Proposed Solution for Android Malware Detection

This chapter proposed an architectural solution for Android malware detection using machine learning techniques. Research questions are defined to solve the problem mentioned inside the statement of the problem in this study. System architecture was proposed based on the literature discussed previously on related works in order to achieve the goal of the study.

4.1 Client-Server Android Malware Detection System Architecture.

Client-server system, a system which classifies suspicious Android application into a malicious class or normal class, have been proposed as shown in Figure 4.1. Specifically, the untrusted Android application can be detected based on the SHA-256 signature value on the client side. The system use value searching on cloud database method, the mechanism in which SHA-256 value of the untrusted application is generated and the value is compared with existing SHA-256 values on the cloud database. For further detection accuracy, the unknown application is submitted to the server.

The proposed framework is based on signature-based, static analysis flowchart techniques. An android malware detection method, that generates standardized mobile malware signatures based on their behavior and a method for comparing generated signatures and contribute a high-performance detection system [54] [49] [57]. Once the client receives the application for installation, its SHA-256 value is generated and checked on the cloud database. If the SHA-256 value doesn't exist on the database yet, it requests the untrusted application to the server for further analysis.

The server undertakes feature extraction of a given application and gets the permission requested by the application. From the extracted feature, the feature vector is generated and prepared for the classification process (or classifier algorithms). Three different machine learning algorithms such as Support vector machine (SVM), Naïve Bayes (NB) and Decision Tree (DT) are used to learn constructed vector from training data.

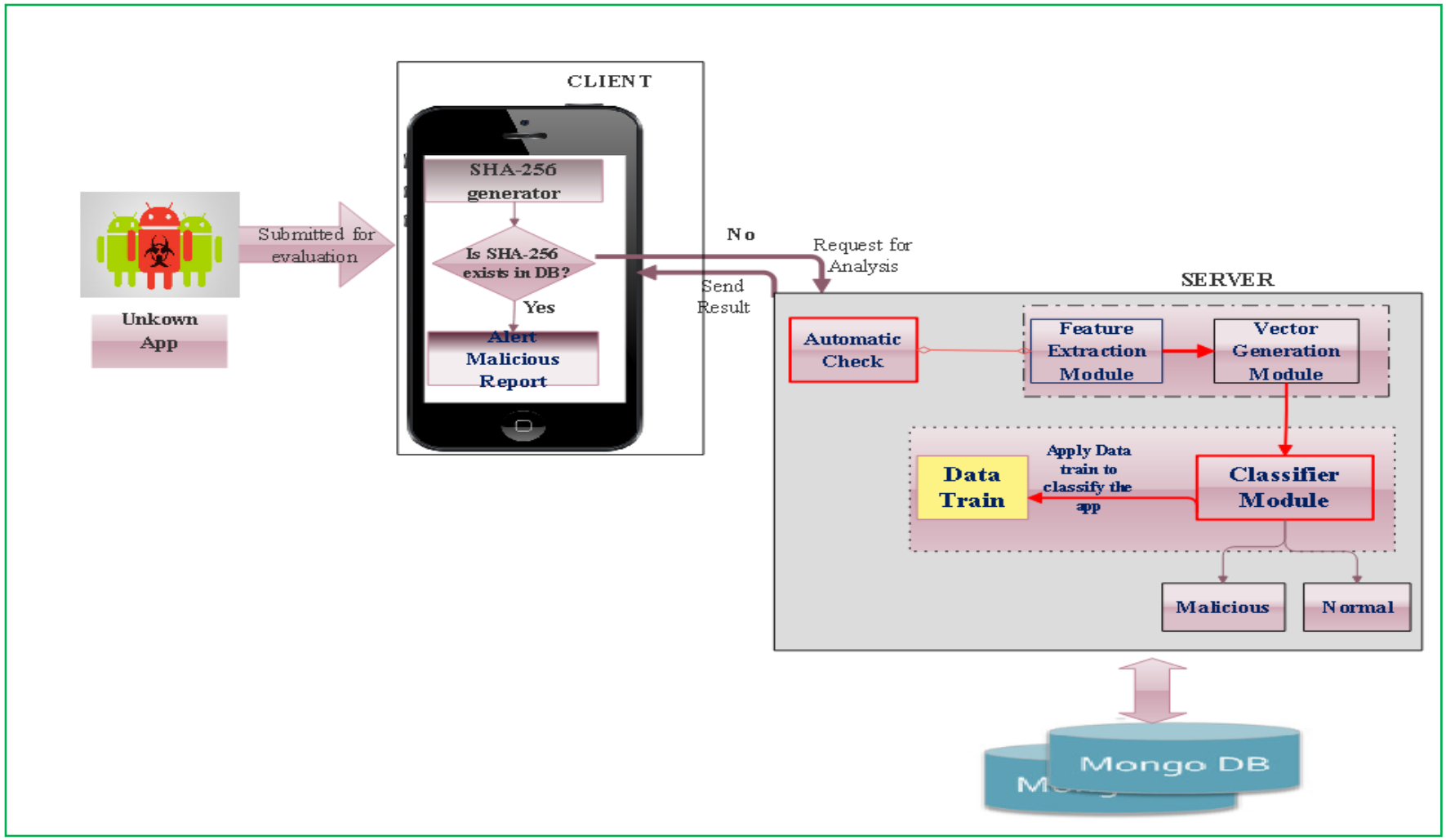

Figure 4. 1 Overall Android Malware Detection System Architecture

4.2 Proposed pseudo code for Client-Server Android Malware Detection

```
1  Input ← application path
2  Byte [] array ← md. digest (SHA-256.getBytes)
3  StringBuffer sb ← new
4  For i ← 0 to length of array do
5       SHA-256 ← (Integer. toHexaString (array[i] &0xFF) |0x100))
6  Return SHA-256
7  String result
8  If SHA-256 exists in local DB
9     Result ←" the application is malicious"
10 Else
11       Send request to the server
12       Server receive request from the client
13       If SHA-256 exists in mongo DB
14            If SHA-256 is under malicious column
15                 Result ←" the application is Malicious"
16            Else
17                 Result ←" the application is Normal"
18            End if
19       Respond the request
20       Else
21            String file1← AndroidManifest.xml
22            String file2 ← gathertPermissions.txt
23            String result
24            While line in file1:
25                 If "android. Permission" in line
26                      permission2 ← re. search ("android\.permission\. [\w.]+", line)
27                      file2 ← permission2
28                 End if
29            End While
30            Return file2
31            If the file2 is not null
```

```
Int array1
Int array2
Int array3
String s ← StandardPermissions.txt
For line in s:
    array2 ← line of s
End for
Open gathertPermissions.txt as f
For line in f
    array1← line
i ← 0
j ← 0
End for
For i in range (0, Len (array2)) do
    For j in range (0, Len (array1)) do
        If array2 [i] in array1
            Fi ← 1
        Else
            Fi ← 0
        Increase j
        array3 ← fi
        End if
        End for
        Increase i
End for
Return array3
Int Detection ← np. array (array3)
Prediction ← svma. predict (Detection)
If prediction is 1
    Result ←" the application is Normal"
Else
    Result ←" the application is malicious"
```

64 **End if**
65 Return result
66 Respond the request
67 Update Database
68 **Else**
69 Error
70 **End if**
71 **End if**
72 **End if**
73 ***Output*** detection report

Description of proposed pseudo-code:

- Translation into SHA-256 value: SHA-256 value is generated for a given application that enables the identification of application based on the signature of the application.
- Static analysis: CSAMDS statically scrutinizes a specific Android application and extracts requested permissions from the AndroidManifest.xml
- Translation into a binary vector: Feature vector is constructed for extracted permissions.
- Learned-based classification: Classification supervised learning algorithms are applied to feature vectors determining classification rules, which enables the detection of the malicious application.
- Alert: The alerting module retrieve the feature vector from vector generation module and specify or classify the type sensitive application.
- Reporting: It provides the users to identify what type of application is they are trying to install. Results obtained from the classifier and the alerting modules.

4.3 Client-Side Android Malware Detection Framework

As is illustrated in figure 4.1, the proposed solution is divided into two parts, namely: the client and the server. In client side, it mainly provides a home screen of the system which is used as UI (user interface) for the users. Since, it is difficult to run large python module on user mobile (limited memory space of mobile device), a simple task performed on the client side of the system. Specifically, when a new application is submitted to the system, the SHA-256 value is extracted, and then compared with malicious samples stored on the database. If the SHA-256 value of the app exists in the database, the client alerts malicious message to the user.

In various traditional android malware detection system [52] [56], it is difficult to detect unknown malicious app. In the context of this research paper, the new application which SHA-256 value does not exist in the malware sample database is considered as unknown malicious. When the unknown malicious application is submitted to the client, the client passes the SHA-256 and APK file name to the server. The general architecture of Android malware detection on the client has been illustrated in figure 4.2.

While conducting detection process on a client, applications do not have to be installed on the user's Android device and even if the procedure use the memory of the device, the application to be evaluated do not use the memory of the device to protect the device from malware content as some types of malwares can get the device under control and mark themselves as "normal" as soon as they are installed.

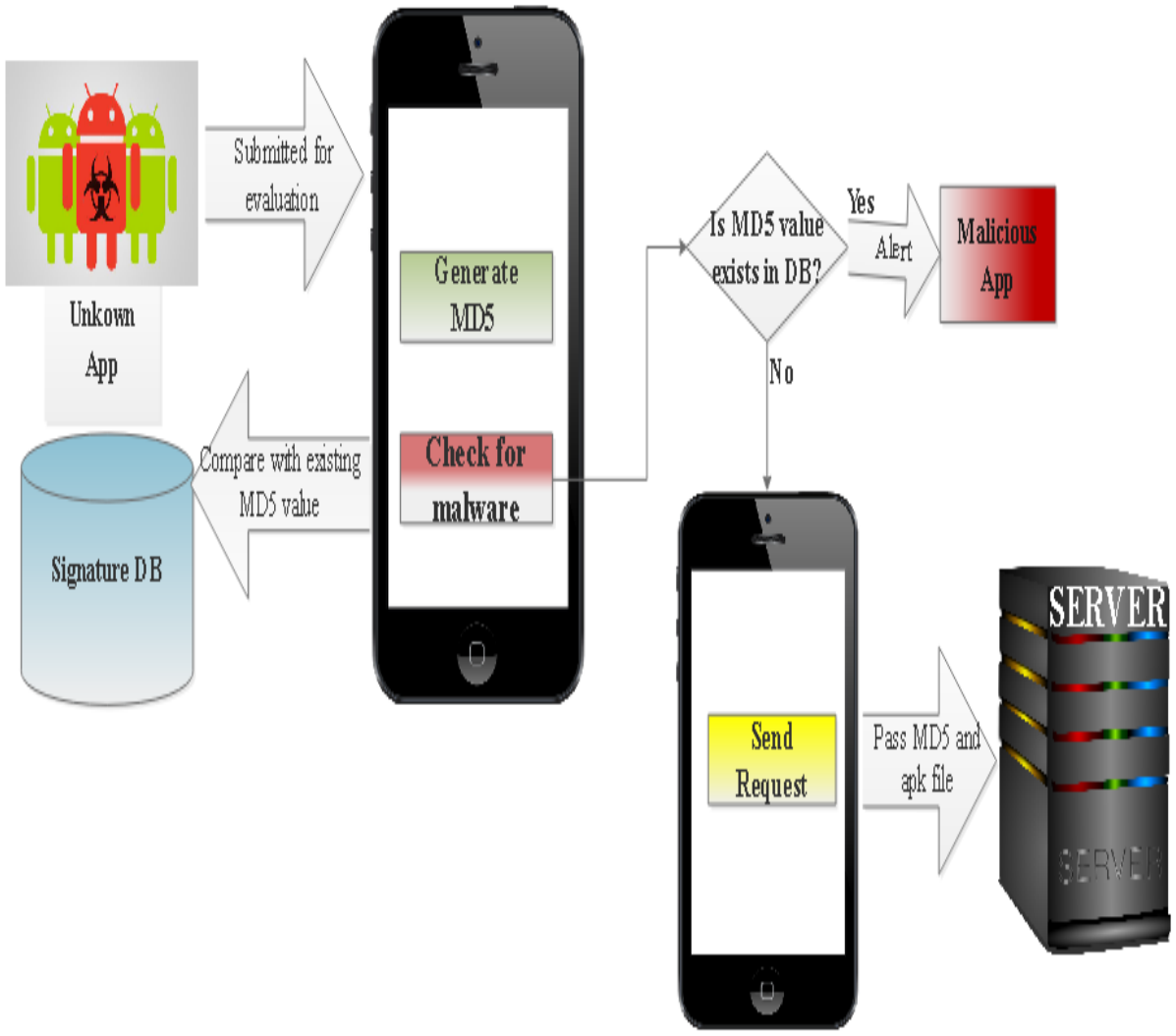

Figure 4. 2 Android Malware Detection Architecture on the client side

4.3 Server-Side Android Malware Detection Framework

The proposed Android malware detection on the server, as shown in figure 4.3, consists of five major parts. The first component, a feature extractor, perform feature extraction, particularly permission requested, from AndroidManifest.xml. After the feature is extracted feature selection is carried out in the second component, which is used in dimension reduction. In the third component, vector generator carries out to convert extracted features into 0 and 1 based on permission requested as shown in equation 2. As a result of this component, each app is represented as a single instance binary vector. Machine learning algorithms use this binary vector and learn from trained data to classify unknown application. The classification model is trained from

collected data and ready for supervised classification algorithms in the fourth component. Lastly, in the fifth component, the SHA-256 of malicious app added in the database.

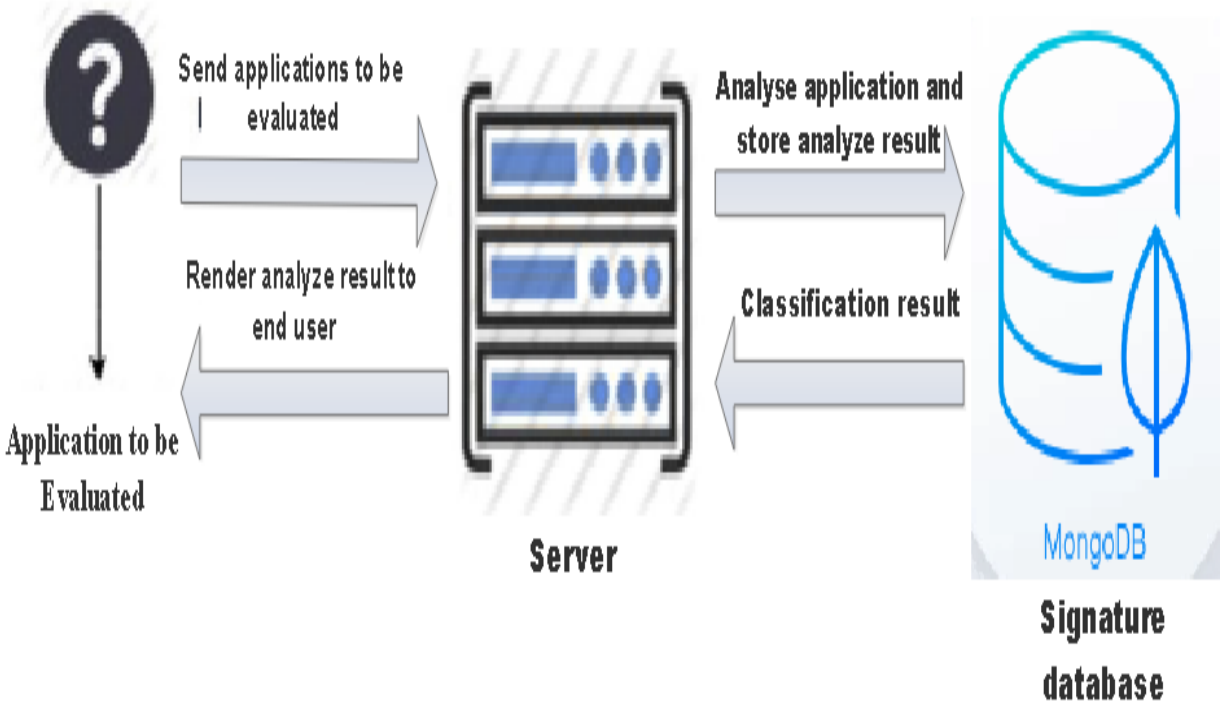

Figure 4. 3 Android Malware Detection Flowchart on Server of the system

4.4.1 Feature Extraction Framework

Feature extraction is performed for obtaining the feature data to predict the class of android application. Permission required is a feature extracted from Android manifest.xml to use them in analysis APK files. Feature extraction step was performed while preparing dataset before analysis. The APK file without a feature extractor module is not enough to use them in the analysis.

In this study, APKTools (open source), dex2jar which have been developed by Chinese student is used to decompile APK file. AndroidManifest.xml is a result of decompiled APK file using this reverse engineering tool.

In order to have permission data of the Android application, first AndroidManifest.xml extracted from the APK file as shown in Figure 4.4 and ready for feature extraction. After extracting AndroidManaifest.xml from the APK file, the feature extractor component extract permissions

defined in this file to examine the list of permissions requested to run Android application. This permission feature is used as dependent variable in the analysis of detecting malicious Android application.

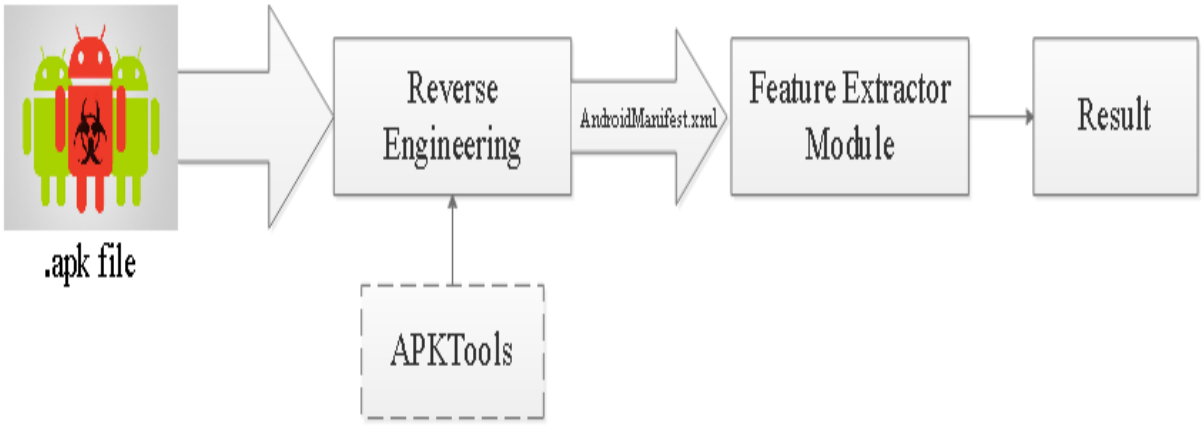

Figure 4. 4 Feature Extraction flowchart

4.4.2 Binary Vector Generation Flowchart

Machine learning from training data that was presented in 0 and 1. Here, the feature extracted was represented as a single instance with a binary vector and classified as benign or malicious. Binary vector is generated based on permission requested to run Android application; if the feature is present in the app it is represented by 1, if the permission is not present in the application, it is represented by 0. Figure 4.5 illustrates vector generation with permission requested.

$$f_i = \begin{cases} 1 \text{ if related permission is required} \\ 0 \text{ if related permission is not required} \end{cases}$$

4.4.3 Modelling Classifier Flowchart

There are three components to model machine learning. The first model is the model definition, the model component which undertakes three activities such as: get data, prepare data and define data. The second model component is a model training component in which machine learning algorithms are applied to the defined feature to classify the unknown application into the appropriate class. The third modeling component is a model evaluation and a testing component. In this component, the classification accuracy and the performance of the model can be evaluated based on training data and test data.

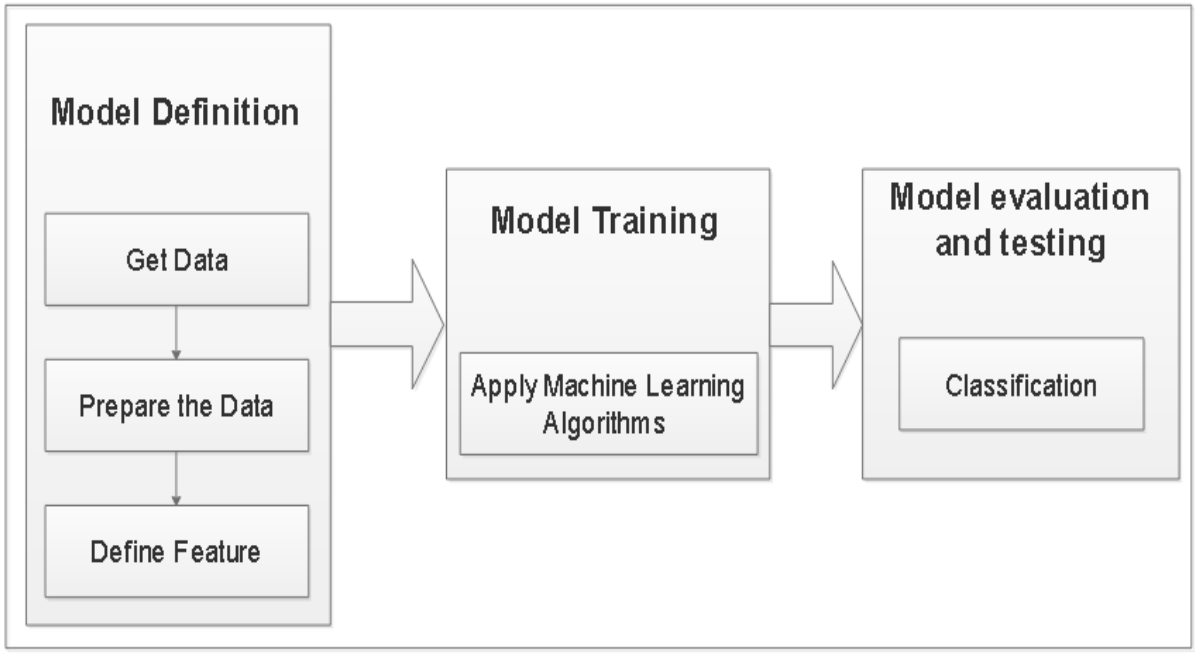

Figure 4. 5 Modeling machine learning classifier flowchart.

4.4.3.1 Model Definition

The model definition is a primitive component for machine learning model in which data sources are created for training the model. Three activities under model definition components have been defined:

4) ***Get Data:*** The first thing to perform machine learning is data. In this study, the datasets downloaded from the genome project have been used. These datasets include the number of both normal and malicious applications including features or attribute that differentiate malware application from a normal application.

5) Prepare data: A data set usually constraints some preprocessing earlier it can be analyzed. Data preparation is the process of obtaining data ready for machine learning. There may be noticed the missing values presented in the column of various rows. In this study, rows that missing values have been removed.

6) Define feature: In machine learning, features, that are used as an independent variable are individual measurable attributes of Android applications. In our data sets, each row represents one android application, and each column is the feature of the application.

4.4.3.2 Model Training:

In this section, the algorithm performance with respect to feature vector constructed from the permission requested was reported by the given Android application. Here, our objective is to select the better classification algorithm for Android malware detection system.

Train modeling is the process of constructing the predictive machine learning model that decide the unknown application whether it is malicious or normal. It is a mathematical modeling with parameters that map the input variable into the target variable i.e. output.

The entire dataset is divided into two categories, one which is used in training the model i.e. Training set and the other that is used in testing the model after training, i.e. testing set.

***Training set*:** Training set is used to build a model. It consists of the correct answer which is used to train the system. The learning algorithms find patterns in the training set that map the unknown application attributes or features to the target (answer whether the application is normal or malicious), and it outputs the machine learning model that captures these patterns. Training rules and algorithms used give relevant information on how to associate input data with output decision. The system is trained by applying three machine learning algorithms on the dataset, all the relevant information is extracted from the data and results are obtained. Generally, 70% of the data of the dataset is taken for training data [66].

***Testing set*:** the testing set is used to test the classifier. It is the set of data which is used to verify whether the algorithm is producing the correct output after being trained or not. Generally, 30% of the data of the dataset is used for testing. Testing data is used to measure the accuracy of the classifier.

CHAPTER 5

5 Implementation of the Proposed Solution

This chapter described the prototype implementation of the proposed solution. Each Android malware detection modules are implemented. Generally, the proposed Android malware detection system has five major implementation modules such as SHA-256 generation, feature extraction, vector generation, quantifying the quality of classifiers and modeling the classifier. The home screen is a client-side user interface of proposed android malware detection system which is used to generate SHA-256 and pass the APK file to the server for analysis.

5.1 Implementation Environment

Several development tools are used to implement the proposed Android malware detection system as shown in Table 5.1.

5.1.1 Java and XML

Java programming language was used to develop functionalities, while highlighted files related to application resources such as Manifest, layout, and drawables are developed by using XML. Java Development Kit (JDK 6.0) which is a superset of JRE has been installed. JRE provides Java Virtual Machine (JVM), libraries and other components to execute the application. The JDK allows running and debugging the java code.

5.1.2 Python

Python is used to develop machine learning model on the server side. The discussion under chapter three shown that Python is more popular and applicable machine learning language which has various features and packages that used in data analysis. Many supervised machine learning algorithms perform the prediction based on given datasets. Data preparation like feature extraction and vector generation is implemented by using the Python programming language. Three classification algorithms such as Support Vector Machine (SVM), Decision Tree (DT), and Naïve Bayes (NB) are compared and implemented using the Python programming language.

5.1.3 Android Studio

Android Studio is the official Integrated Development Environment (IDE), which is freely used by Android developers. Android Studio is based on IntelliJ IDEA, an Integrated Development Environment that also presents a good Android development environment that provides advanced code completion, refactoring, and code analysis. The great power code editor helps to have more productive Android application developers. Android studio is used for the development of Android malware detection. It allows:

- Create a new project using template code for patterns such as a navigation drawer and view pages
- Build applications for Android devices such as mobile, tablets, smart watches, and TV.
- Create multiple APKs for our application with different features in the same project.
- Build APK from Android Studio.

5.1.4 Android SDK

The Android Software Development Kit (SDK) is a collection of development tools used to build applications for the Android platform. The Android SDK consists:

- Required libraries;
- A Debugger environment Dalvik Debug Monitor Service (DDMS) with Android Debug Bridge;
- An environment for the construction of the application Android Asset Packaging Tool (AAPT);
- An emulator Android Virtual Device (AVD);

Table 5. 1 Describes the tools used during the implementation of the proposed framework

Tools	Description
Python	Used to develop machine learning model on the server side
Android Studio	A collection of development tools used to build applications for the Android platform.
Android Emulator	We use the emulator to run the applications and generate SHA-256 of the application without the need for real devices.

5.2 Deployment of the Environment

The tools discussed in section 5.1 have been installed and configured on a hp Personal computer equipped with a processor Intel(R) Core (TM)i5-3360M CPU with a frequency 2.80 GHZ per processor, 12GB of memory and 700GB of disk storage. The operating system is window 10, 64 bits. The client development and detection phase have been performed on this computer. The AVD was used to emulate Nexus 7 phone in order to visualize and test the application while implementing (Figure 5.4). then deploy on real devices: A Techno Mobile with Android version 7.0. The execution of Android malware detection on different Android platforms reveals no major difference.

5.3 Implementation of AMDMA System Components

The proposed system, Android malware detection using machine learning approach have five major modules which implemented on both client and server. On the client side, two modules are implemented such as a module that generates the SHA-256 value of each app and a module that checks the application whether it is normal or malicious. On the other hand, three main modules are implemented on the server namely: feature extraction module; vector generation module and a detection module.

5.3.1 Generate SHA-256 value

An SHA-256 value is an encrypted sequence of characters that identify one application from another that obtained after applying some algorithms and manipulations on user-provided content. In this study, the SHA-256 checksum of the APK file is generated and compared with the list found on the database at the client. The following snapshot (or figure 5.2) shows the portion of code which used to generate the SHA-256 value of the given application.

```
@Override
public void onClick(View v) {

    if(v.getId()== R.id.generate){
        SHA256( mdv: "C:\\Users\\user\\Desktop\\Thesis Prototype\\com.uc.browser.en_v10.7.8-101_Android-2.3.apk");

    }
    if(v.getId()== R.id.checkLink){

        String value = edit3.getText().toString();
        check(value);

    }
}
public String SHA256(String mdv) {
    try {
        java.security.MessageDigest md = java.security.MessageDigest.getInstance("SHA-256");
        byte[] array = md.digest(mdv.getBytes());
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < array.length; ++i) {
            sb.append(Integer.toHexString( i: (array[i] & 0xFF) | 0x100).substring(1,3));
        }
        String mdvalue = sb.toString();
        edit3.setText(mdvalue);
        return mdvalue;


    } catch (java.security.NoSuchAlgorithmException e) {
    }
    return null;
}
```

Figure 5. 1 Sample code for the SHA-256 generation

5.3.2 **Gathering the List of Requested Permissions**

The module of feature extraction focuses on gathering required permissions. The list consists of permissions, which follow the syntax of android.permission.X. The permission collected through feature extraction module corresponds to all features presents in AndroidManifest.xml file. In the reverse engineering process, APK file is converted into AndroidManifest.xml and various Java classes. A permission requested to access the device is defined in AndroidManifest.xml during application development which tends us to use this file to extract features.

The following snapshot shows the portion of code which used to gather the permissions requested by the application

```
#Extracting android.permission features from the Android Manifest file of Facebook.apk

file1 = open("AndroidManifest.xml","r")
file2 = open("gathertPermissions.txt", "w")
for line in file1:
   if "android.permission" in str(line):
      permission2 = re.search("android\.permission\.[\w.]+", line).group() + '\n'
      print (permission2)
      file2.write(permission2)
file1.close()
file2.close()
```

Figure 5. 2 Sample code for permission extraction.

5.3.3 **Vector Generation**

This module takes extracted permissions of each application as input which is obtained from the feature extraction module. Before modeling machine learning algorithms, extracted features have been converted to vector form. In this study, for every application to be detected, a feature vector can be generated. All Android app's features are represented by the binary array in the dataset. In section 5.3, the list of requested permission by the application has gathered which is used as an attribute to predict the class of application. The app's feature vector is constructed as a Boolean vector (f1, f2, ..., fn): fi = 1, if related permission is requested by the application. Otherwise, fi = 0. Through this vector, all API data dependencies can be represented. Table 5.1 presents a vector generation algorithm that demonstrates how to construct a binary vector from the extracted feature.

Table 5. 2 Algorithm 1: Vector Generation

Algorithm 1
Open standard_permissions.txt
Declare array1, array2.
Input gatherPermissions.txt to array1
Input standard_permissions.txt to array2
n= size of array2
Declare array3
For i=0 to n
begin
if array2[i] presents in array1
begin
erase array2[i] from array2
array3[i] =1
end
else array3[i] = 0
end
output array3

This is performed by Vector generation () function which is defined inside the application classification code.

```
array1 = []
array2 = []
array3 = []
with open("StandardPermissions.txt", "r") as s:
   for line in s:
      array2.append(line)
with open("gathertPermissions.txt", "r") as f:
   for line in f:
      array1.append(line)
   i = 0
   j = 0
   for i in range(0, len(array2)):
      for j in range(0, len(array1)):
         if array2[i] in array1:
            fi = 1
         else:
            fi = 0
         j += 1
      array3.append(fi)
      i += 1
```

Figure 5. 3 Sample code for vector generation

5.4 Quantifying the quality of classifiers

To obtain classifier model that has good classification performance, three classification algorithms such as Naïve Bayesian, Decision Tree and Support Vector Machine were compared. The Naïve Bayesian algorithm is based on Bayes' theorem with independence assumptions between forecasters. Bayesian reasoning is used with decision making. It uses the prior knowledge about events for predicting the events in the future. Naïve Bayesian works without the need for complex iterative parameter estimation and that made it mainly most useful for huge datasets. Support vector machine algorithm performs classification by finding hyper-plane that tell part into two classes very well. "Decision Tree" is a supervised algorithm used for classification. Algorithms of decision tree create trees that predict the result of an input vector based on decision rule implicit from the features present in the data. Table 5.2 shows the methods to compute all the values used in quantifying the quality of the classifier.

Table 5. 3 Important Parameters and their computation Methods

Parameter	***Computation method***
True Positive (TP)	The true positive result is the one that detects the condition (presence of malware) when the condition is present.
True Negative (TN)	The true negative result is the one that does not detect the condition (presence of malware) when the condition is not present.
False Positive (FP)	The false positive result is the one that does detect the condition (presence of malware) when the condition is not present.
False Negative (FN)	The false negative result is the one that does not detect the condition (presence of malware) when the condition is present.
Precision	$\frac{TP}{TP + FP} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots eq\ 5.1$
Recall	$\frac{TP}{TP + FN} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots eq\ 5.2$
Accuracy	$TP + \frac{TN}{Total\ number\ of\ samples} \ldots\ldots\ldots\ldots\ldots\ldots eq\ 5.3$

Where,

- True Positives (TP): The number of malware applications classified as malware.
- True Negatives (TN): The number of normal applications classified as normal.
- False Positives (FP): The number of normal applications classified as malware.
- False Negatives (FN): The number of malicious applications classified as benign.

Average precision (AP) is computed from the prediction score to calculate the precision of each classification algorithms. The following portion of python code implements how to get the accuracy of the classifier.

```
y_score1 = svma.decision_function(x_test)
y_score2 = gnb.predict(x_test)
y_score3 = decisiontree.predict(x_test)
from sklearn.metrics import average_precision_score
average_precision1 = average_precision_score(y_test, y_score1)
average_precision2 = average_precision_score(y_test, y_score2)
average_precision3 = average_precision_score(y_test, y_score3)
```

Figure 5. 4 Sample code for accuracy of the classifier

CHAPTER 6

6 Evaluation, Result, and Discussion

In this chapter, the evaluation of the proposed Android malware detection using machine learning approach prototype on mobile devices have been described.

6.1 Performance of Classification Models

Three different types of machine learning algorithms such as support vector machine; decision tree and Naïve Bayesian were proposed. These three different machine learning algorithms have different detection performance. The algorithm with high classification accuracy is selected to provide a better Android malware detection system. After the implementation of the dataset on these algorithms, the true positive (TP) value, false positive (FP) value, accuracy, precision, and recall have been calculated.

```
Average precision score for SVM: 92.72 %

Average precision score for Naive Bayes: 78.63 %

Average precision score for Decision Tree: 81.27 %
CLASSIFICATION ALGORITHM         ACCURACY
        Decision Tree             86.0 %
        Support Vector Machine    88.0 %
        Naive Bayes               81.0 %
```

Figure 6. 1 Comparison result for classifier algorithms

The accuracy and precision calculated using data test for these algorithms are presented in table 6.1.

Table 6. 1 Important parameters and their calculation result

Classifier algorithm	***Parameter***	***Calculated value (%)***
SVM	Average precision score	92.72
	Accuracy	88.00
Naive Bayes	Average precision score	78.63
	Accuracy	86.00
Decision Tree	Average precision score	82.63
	Accuracy	81.00

The training data was applied on three different algorithms (SVM, NB, and DT), and compute the accuracy and compared the values. Figure 6.1 shows the comparison between these algorithms. From the figure, the quality of the training data was clearly justified. The comparison result also tends us to apply a support vector machine classification algorithm to develop client-server Android malware detection system using machine learning approach.

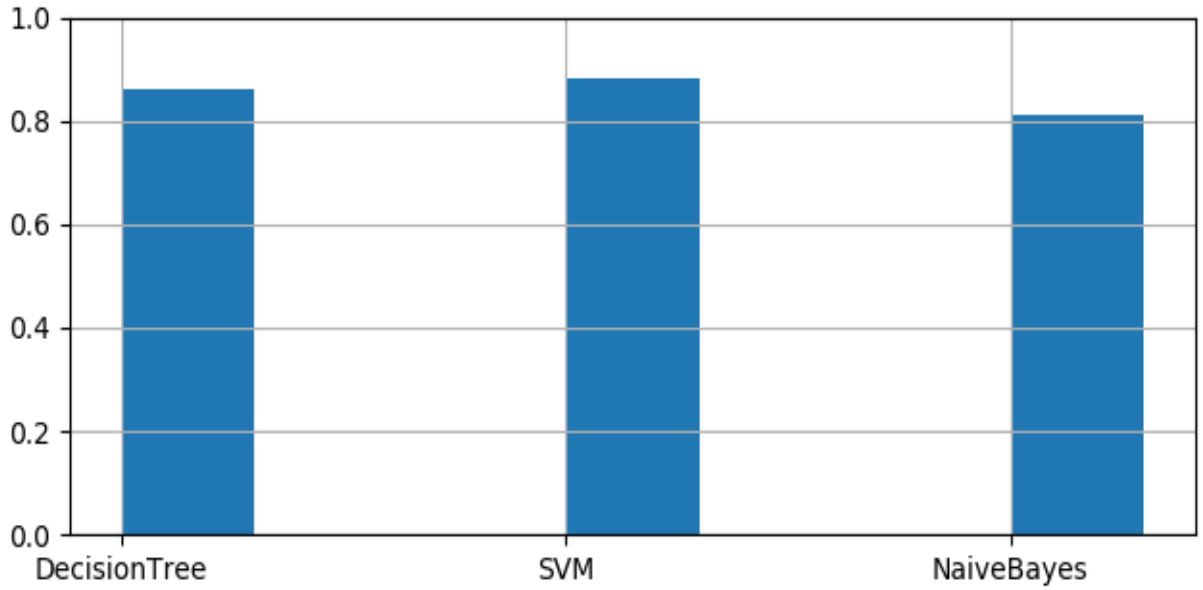

Figure 6. 2 Quantifying the quality of classifier implementation result chart

6.2 Feature Vector Construction Result

In this study, several permissions from android application package (APK) files particularly from AndroidManifest.xml are retrieved. Binary number (0 or 1) is assigned to the selected features. If a given permission exists in a given app is valued at 1 otherwise is 0. Fig. 6.2 shows a given sample used in this paper which includes 329 features.

```
==========FEATURE VECTOR OF THE APPLICATION==========
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 6. 3 Feature vector

6.3 Performance in Detection and Prediction of the system

Case 1: If the SHA-256 value of the given application is existing in local database (or SQLite database):

The process of extracting SHA-256 value and comparing the value with malicious lists found on local database (or SQLite database) is a type of signature-based Android malware detection. In this study, the system can generate the SHA-256 value of the given application to detect the incoming application fast. In this case (case 1), the user can identify the malicious application on client within short period of time. Figure 6.3 and Figure 6.4 illustrate how unknown application is detected using local database or using modeless learning.

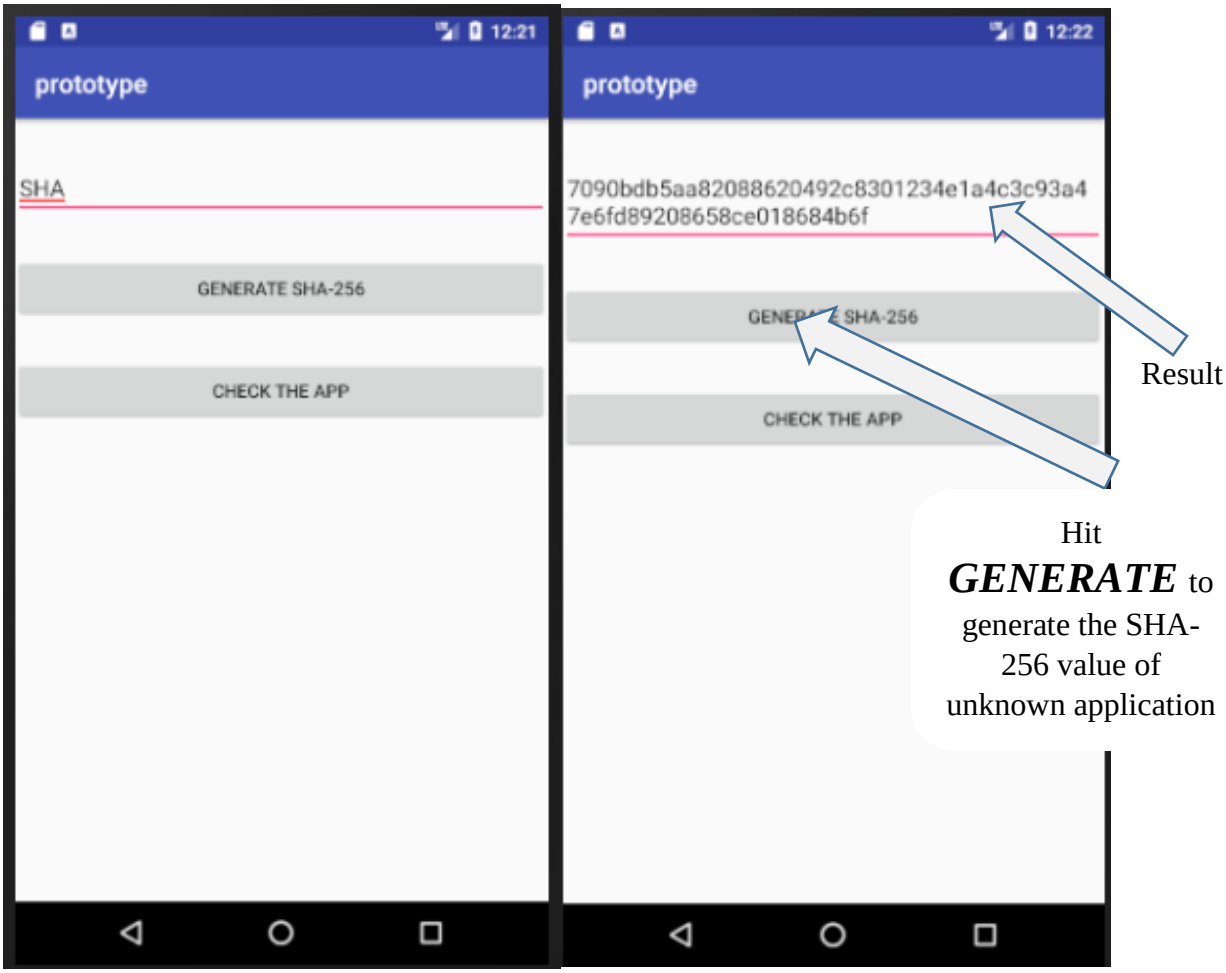

Figure 6. 4 (a) AMDMA User interface. (b) Generated SHA-256

As we have discussed in chapter five, the generate module converts the given Android application into its equivalent hash value that is SHA-256 value. Every APK file has a SHA-256 value which is unique each application. In our proposed system, when the user hit generate button, it converts a given APK file into equivalent SHA-256 value and display the result as shown in a figure 6.3(b).

When the user selects check label, the system can use generated SHA-256 and compare the value with existing lists on local database. If generated SHA-256 is existing in the list, the system informs that the given application is malicious and recommend the user to cancel the installation of the application as shown in the figure below.

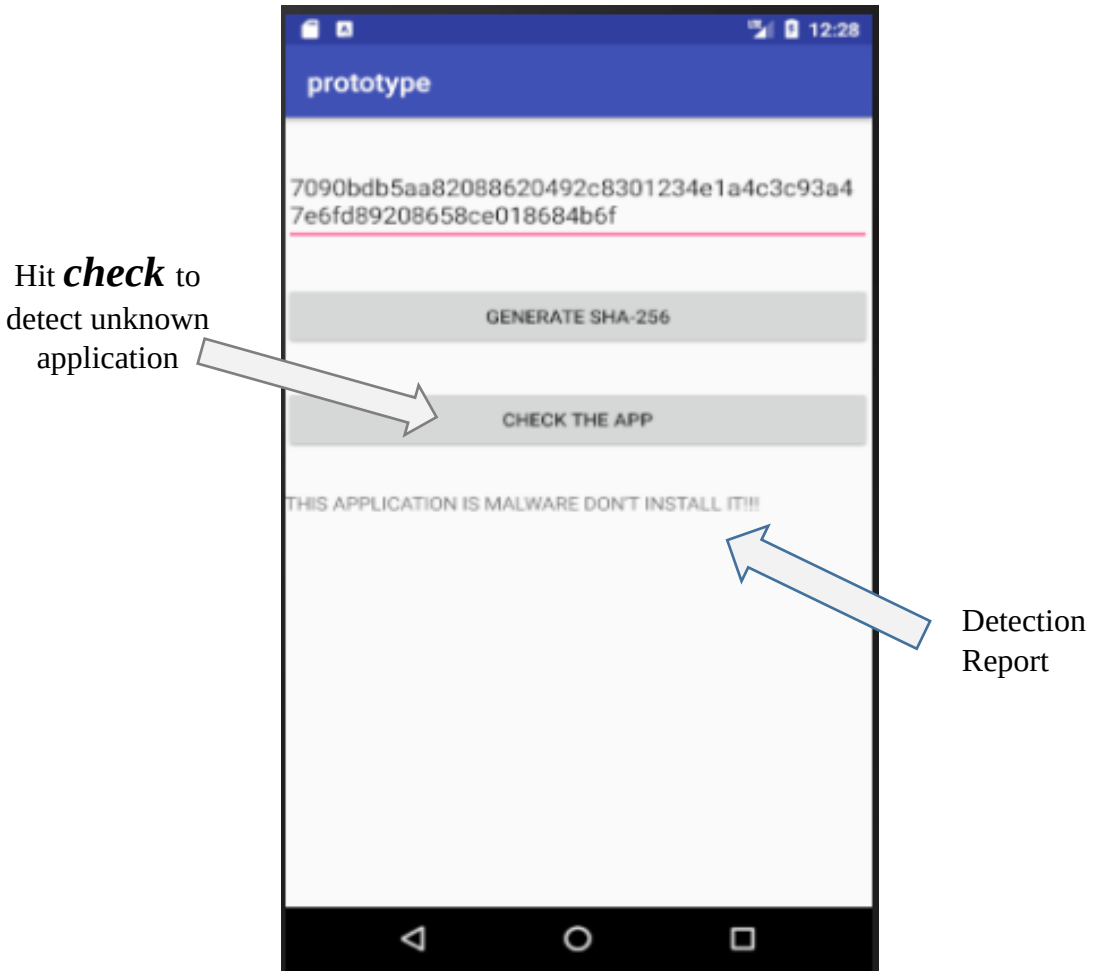

Figure 6. 5 Detection result when SHA-256 is exists on local database.

Case 2: If the SHA-256 value of the given application is not existing in local database (or SQLite database):

In this case the system has been checked the generated SHA-256 value in the local database, but the value is not found in the list. Even though the SHA-256 of the application is not found in the local database, it is difficult to decide that the application is normal application. In such case, the system requests the server for more malware analysis.

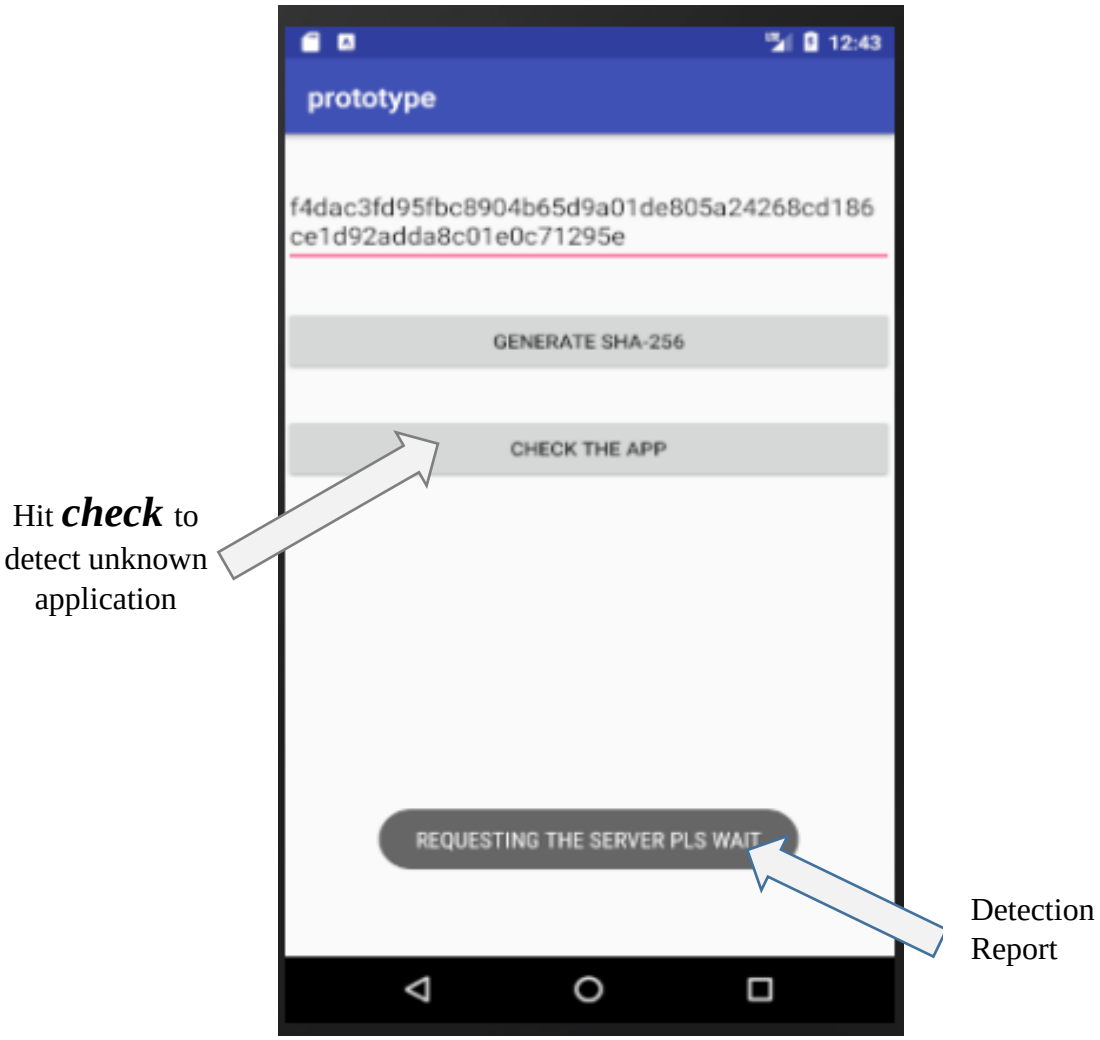

Figure 6. 6 Detection result when SHA-256 is does not exists on local database.

Here, the system displays 'REQUESTING THE SERVER PLS WAIT' message to inform the user that the system is connecting with the server. The client passes the APK file path and SHA-256 value for the server and the server continue malware detection process.

Case 3: If the SHA-256 value of the given application is existing in Cloud database (or MongoDB database):

This case can be occurred when a client requests for detecting unknown application that have been detected yet by any other client. To reduce the delay of time to respond for the request, there is a module called ***automatic check*** which is used to check server database before other modules are run. Case 2 shows that, the SHA-256 value of a given application 'fb37242686173e12a9734b1a19706f3db497236b48530792d1cda8298bf9f75e' doesn't exist on local database, but figure 6.5 shows that this value is exists on server database. In such scenario, the server responds the request for the client as much as possible in short period of time.

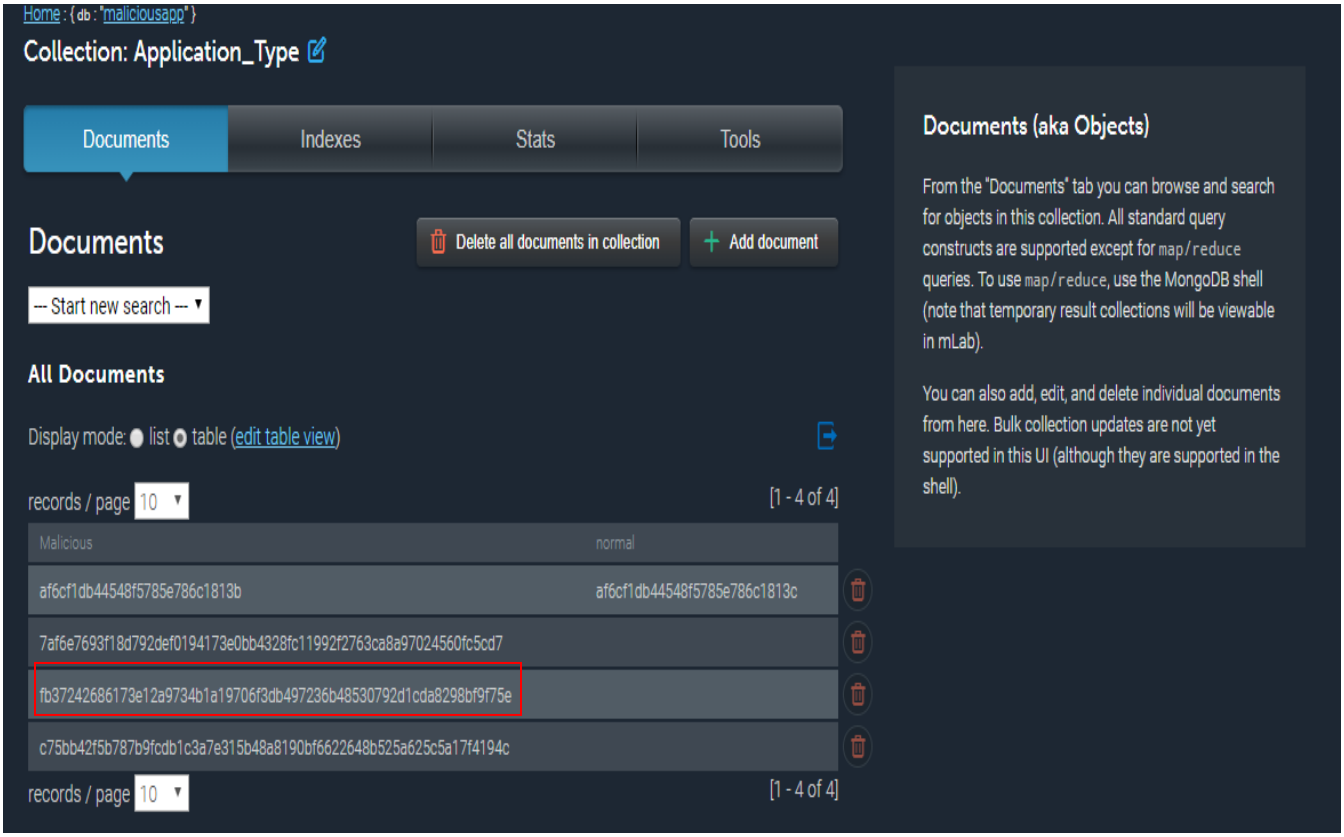

Figure 6. 7 MongoDB database

6.4 Comparison with Antivirus Scanners

The similarity and differences between the proposed system and existing Android antivirus applications are shown in follow Table 6.2.

Table 6. 2 Performance Comparison between Antivirus applications and proposed mechanism

Comparison	***Antivirus Applications***	***CSAMDS (proposed Mechanism)***
Detection Approach	Signature-based	Signature-based Permission-based
Objective	Scan all applications that installed on Android mobile device	Detect malicious application before installation
Methods	Check all the applications installed on the device by name. Compare the name of the application with existing malware application list.	Generate the signature of the application and compare with the existing malware app list. If the application is new application it requests the server for malware analysis
Limitations	Cannot detect unknown malware.	Probably misclassification

The above table describes the comparison between existing system and suggested mechanism based on detection mechanism. Many antivirus developers try to overcome the Android security issue by developing antivirus applications. However, various problems are found in the detection process. Google play store by itself has a policy which identify the application when the developers try to upload the legitimate application to reduce the intruders attack. Since the mobile antivirus applications detect the applications from google play store, it is difficult to say that existing Android antivirus applications have extreme value in detecting malicious application.

The proposed mechanism adds a value compared with antivirus applications, by detecting the applications that exists not only on Google play store but also that exists on the third-party Android application markets. Table 6.3 illustrates the problem found in antivirus android applications that have been solved in this study.

Table 6. 3 Advantages of CSAMDS over Antivirus Android applications

Antivirus Android applications	***CSAMDS (proposed mechanism)***
Only scan application stored on Google Play Store	Detect applications that found on every Android application market
They detect only by identifying the name of the application installed on the device	Detect by using Machine learning
Only known applications are detected	The unknown application also can be detected using machine learning.
Since the Android application can scan applications after installation, they cannot be scan the system that affected for the malicious process during installation	Since the proposed system, that detects malicious application using machine learning, detect the malware before installation, the whole device system can be safe from threat.
Antivirus applications are not meant for staying safe on Android devices	It is important for staying safe on Android devices.

6.5 Performance Comparison with Existing Mechanism

Table 6. 4 Performance Comparison

Comparison	***Existing Mechanism***	***CSAMDS (proposed Mechanism)***
Experimental Platform	Android	Android
Learning algorithm	SVM	SVM
Methods	Generate the signature of the application and compare with the existing malware app list. If the application is new application, ***it submits to the learning model***	Generate the signature of the application and compare with the existing malware app list. If the application is new application, ***first it checks server database.***
Availability	Less	More
Speed	Slower	Faster

The differences and similarity between the proposed technique at chapter four of this paper and the technique developed by Long Wen, and Haiyang Yu [49] are shown in follow Table 6.4. Both techniques used client-server approach and collected and analyzed the permission from AndroidManifest.xml. However, the analysis of the applications have been conducted and store on the signature database using cloud database to improve the avaialabilty and speed of the detection system.

6.5.1 Analysis of the Algorithms

Analysis of the proposed algorithm with the related approach in case of n number of users request for malware analysis that just detected. In the case of unknown application or unknown malware, our system provides the user to request server. Even if the requested application specification not found on the local database with high probability, it may the specification of the application exist on the the server database. The following algorithm is designed for such probability.

Table 6. 5 Proposed Algorithm

Algorithm 3
1. Int n
2. String Result
3. If SHA-256 exists in Mongo DB
4. If SHA-256 exists under the malicious column
5. Result ← “The application is Malicious”
6. Else
7. Result ← “The application is Normal”
8. End if
9. Return Result
10. For i ← 1 to n do
11. Respond the request

In case of related approach, when the server receive request from client, all the modules found on business logic such as feature extraction; feature selection; vector generation and prediction module can be executed [51] [49] [54]. The algorithm which is used in the case of related approach

is shown on table 6.6 below, when n is a number of clients request the server for the same application that is detected yet by another client.

Table 6. 6 Existing Algorithm

Algorithm 4

```
1.  String result
2.  String file1← AndroidManifestF.xml
3.  String file2 ← gathertPermissions.txt
4.  For i ← 1 to n do
5.          While line in file1:
6.                  If "android. Permission" in line
7.                      permission2 ← re.search ("android\.permission\. [\w.]+", line)
8.                      file2 ← permission2
9.                  End if
10.         End While
11.         Return file2
12.         If the file2 is not null
13.             String s ← StandardPermissions.txt
14.             For line in s:
15                  array2 ← line of s
16              End for
17              Open gathertPermissions.txt as f:
18              For line in f:
19                  array1← line
20              For i in range (0, Len (array2)) do
21                  For j in range (0, Len (array1)) do
22                      If array2 [i] in array1
23                          Fi ← 1
24                      Else
25                          Fi ← 0
26                      Increase j
```

```
27              array3 ← fi
28              End if
29              End for
30              Increase i
31          End for
32          Return array3
33          Int Detection ← np.array (array3)
34          Prediction ← svma.predict (Detection)
35          If prediction is 1
36              Result ←" the application is Normal"
37          Else
38              Result ←" the application is malicious"
39          End if
40          Return Result
41          Respond the request
```

This algorithm has been used in some cases like when requested application is not detected yet by any other user or if the specification of the unknown app is not registered both on local and server databases.

Analyzing time complexity of proposed algorithm in this case

Running time of specific algorithm is based on different metrics such as:

- Single vs multiprocessor X
- Read/write speed to memory X
- 32-bit vs 64-bit X
- Input size → in case of our study number of clients ✓

When one talk about time complexity, in this case the first three metrics is not required. The only quantifying the performance of the algorithm depend up on how the algorithm behaves for several of inputs or how the time taken by the algorithm grows are worried. The calculation of time

complexity of both algorithms (proposed solution and related approach) is presented in the table 6.5.

Table 6. 7 Running time

Proposed Mechanism			***Existing System***		
Line	Cost	No. of times	line	Cost (line)	No. of times
1	1	1	1	1	1
2	1	1	2	1	1
3	1	1	3	1	1
4	1	1	4	2	n+1
5	1	1	5	1	$(n+1)^2 = n^2 + 2n + 1$
6	1	1	6	1	$(n+1)^2 = n^2 + 2n + 1$
7	1	1	7	1	$(n+1)^2 = n^2 + 2n + 1$
8	2	n+1	8	1	$(n+1)^2 = n^2 + 2n + 1$
9	1	N	9	1	N
			10	1	N
			11	1	N
			12	1	$(n + 1)^2 = n^2 + 2n + 1$
			13	1	N
			14	1	N
			15	1	$(n+1)^2 = n^2 + 2n + 1$
			16	1	N
			17	2	N
			18	2	N
			19	1	N
			20	1	N
			21	1	N
			22	1	N
			23	1	N
			24	1	N
			25	1	N
			26	1	N
			27	1	N
			28	1	N
			29	1	N
			30	1	N
			31	1	N
			32	1	N
		2(n+1)+n+7	33	1	N
Total	10	3n+9	Total		$3+ 2(n+1) + 25n + 6(n^2 + 2n + 1)$ $=3+2n+2+25n+6n^2+12n+6$ $=11+39n+6n^2$

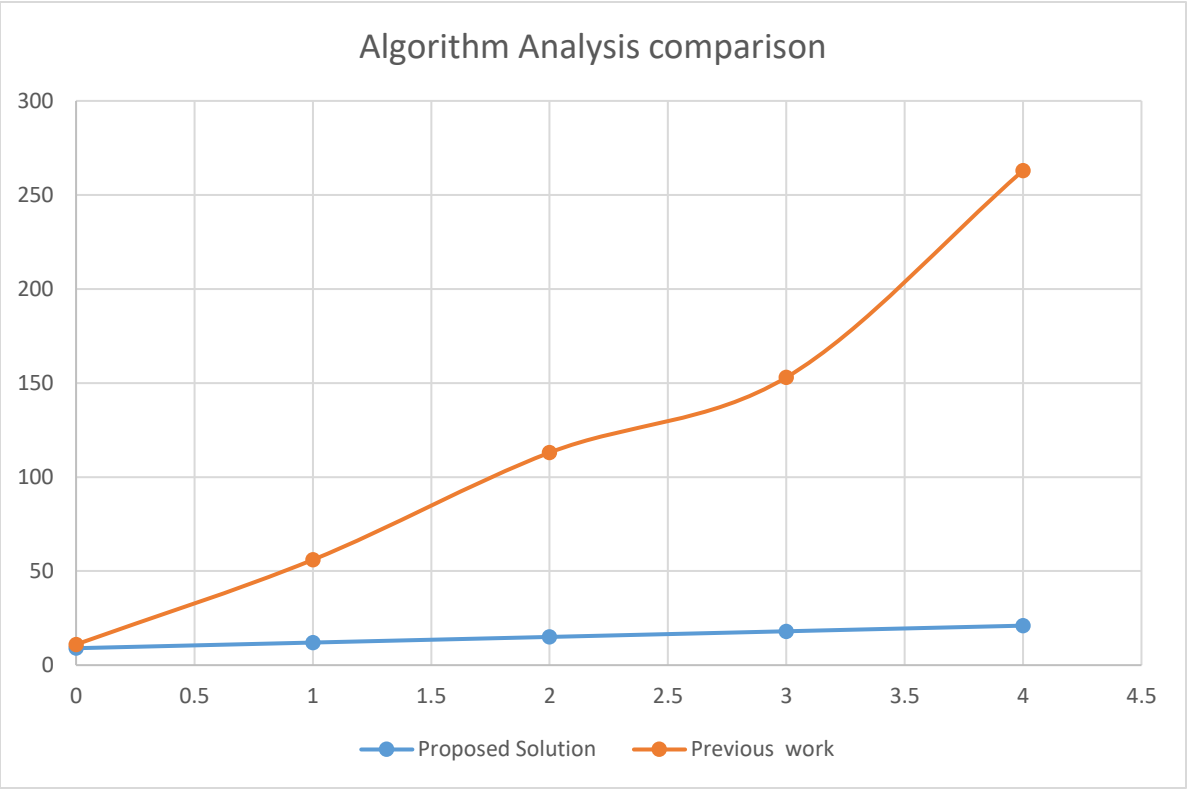

Figure 6.2 Algorithm analysis comparison between our system and related approach

6.6 Discussion

Both signature and permissions are one of the special keys for security on the Android operating system. According to the contribution obtained from previous research, a permission-based mechanism can provide a quick filter to identify the type of applications; that must be associated with a second approach, such as signature comparison, to make a complete analysis for a malicious application. This result is justified with interesting performance indicators such as true positive, false positive, true negative and false negative. For the work on the signature comparison, the solution helps the user to get a response in a short period of time. On the hand permission analysis is the solutions help the user to make the detection of unknown malware.

The support vector machine (SVM) learning algorithm was proposed for detecting Android malware detection. Unlike the previous approach which uses selected features, all permissions requested by both normal and malicious application are combined to develop classifier. The data obtained from literature review shows that support vector machine has better classification

accuracy [49]. The first experiment result also shows that, the support vector machine algorithm has better classification algorithm compared with decision tree and Naive Bayes algorithms.

The second experiment result shows that, proposed mechanism saves a lot of time during the process of checking the signature of the application that exists in server database, in this experiment it is very important to show the ability of the system to save the time, so the input of this experiment will be the same application on two different user's Android mobile, the goal is to detect the malicious application on the second device in short period of time.

First, assume that the application is unknown application that doesn't exist neither on local database nor on server database, the dot APK files were analyzed using machine learning model when the first user request for the analysis. Unfortunately, the second user requests for analysis the same application that was analyzed and the result is stored on database. During this time our system automatically check the database and give response for the request (figure 6.6). This process saves a lot of response time compared to existing mechanism.

The second experiment result, shows that the proposed mechanism improves the availability of the system during the process of requesting for remote server, in this experiment the cloud database (figure 6.6) that store the signature of the Android applications was developed. As described in figure 6.6, mongo DB database called ***malicious app*** using mlab was created which is used to store the signature of the Android applications. In this experiment, it is important to show the ability of detecting the application who's the signature of it is exists on server database being everywhere.

In the previous work, there is time consumption and less availability of the system. This study contributes high availability and time saving when the client requests the server. Section 6.3 case 3, illustrates how the system save the time more than existing system.

In this thesis work, **four research questions** are answered by using both signature-based and permission-based approach by the aid of machine learning approach. In our first research question, informing the Android device users, whether the application is malicious application or not is questioned in order to develop the system that detect the application and render the user the detection result. Even if existing Android antivirus application tries to scan Android applications, they didn't inform the users whether the application is normal or not before the install on their device. In this study, the detection result is reported for the users before they install the application.

In the second research question, the response time is an issue in Android malware detection system. In previous studies, there was no study that is questioning the most suitable technique in which the response time is reduced. However, there are studies using client-server method without questioning its detection performance [53] [49]. To improve the detection performance in client-server method new algorithm (Table 6. 5), that parse server database before running machine learning model was developed. There was no study that used this algorithm (Table 6.5) in the detection of malicious applications.

Quantifying the quality of the classifier is the other research question designed in order to compare the classifier algorithm. Machine learning classifier algorithms have different classification performance. The classification accuracy, one type of metrics for classifier performance, was calculated for three different types of classifier such as NB, SVM, and DT. Among those algorithms, SVM has better classification accuracy than both NB and DT.

In the last research question, the usage of cloud database and client-server method using machine learning algorithms in the detection of malicious applications is asked. Although many researchers developed various mechanism for Android malware detection using related approach with proposed technique, in the proposed mechanisms better approach was designed. Improving the availability and decreasing the response time of the system was the main contribution of this study. The more secured hashing algorithm called SHA-256 [67] was used to advance the detection performance based on signature.

CHAPTER 7

7 Conclusions and Future Work

In this chapter, the summary of the proposed Android malware detection using machine learning approach and future research directions have been described.

7.1 Conclusions

Currently, there are different types of smart mobile devices available in the market which have their own OS and application features. Android OS is the popular software stack for smartphone devices which includes an operating system, middleware, and application. The popularity of this operating system tends the idea of the intruders to attack the Android mobile devices. Malware is the critical issue for Android mobile devices because a large number of applications that are available on application markets are easily repackaged and allows the attackers to insert the malicious code into legitimate Android applications. Hence, different researchers develop Android malware detection system using different approaches.

In this study, the SHA-256 signature, and permissions of Android applications are used to detect malware in Android based mobile platform. The Android malware detection system using machine learning approach was implemented based on SVM, different from traditional system. The comparison result shows that, SVM have better classification accuracy other than supervised classification algorithms. The system detect new incoming Android application or unknown Android application based on machine learning.

The client-server Android malware detection system improves the detection performance than either detection system on the device or detection system on the server. In the client-server approach, detection on client using signature database (SQLite) is important for time saving. On the other hand, developed machine learning model including other business logic can run on the server. Server is important because of the maximum memory capacity of mobile device is 128GB, it is difficult to run various modules developed for malware detection.

Mongo DB database is used as backend of the system to increase the availability of the system. We get the advantage of system availability by using cloud database called mongo DB database, non-query database which is used to store data on the cloud. Since this database duplicate itself, the client requesting the server not only requesting the server nearest to it but also can get the response in short period of time.

Our experimental results indicate that our developed framework has better detection performance using the features collected and cloud concept: permission combinations used by the application, mongo DB database used to store the signature of the detected Android applications. From the experiments we realize that Android permission combination analysis and the finger print information of the application can provide effective method to determine whether the application is normal or malicious application.

7.2 Recommendations for Future Work

The weakness and limitation of approaches and techniques developed in this research work have indicated the following points as a recommendation for future work.

- Including other features such as: API calls, system calls or function calls in modeling the classifier to achieve a higher detection rate as well as reducing the false negative and false positive rates.
- Automatically updating the local database for user of the system based on their preference to improve the speed of detection more.

References

[1] Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weisee, Y. , ""Andromaly": a behavioral malware detection framework for android devices," *Journal Intelligent Systems*, 2012.

[2] S. V. K. M. Narmatha, "Study on Android Operating System And Its Versions," *International Journal of Scientific Engineering and Applied Science (IJSEAS)*, vol. 2, no. 2, pp. 439-444, February 2016.

[3] FRAMINGHAM, "Worldwide Quarterly Mobile Phone Tracker," April 27, 2017.

[4] J. Poushter, "Smartphone Ownership and Internet Usage Continues to Climb in Emerging Economies," Pew Research Center, February, 2016.

[5] Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weisee, Y. , "a behavioral malware detection framework for android devices," *Journal Intelligent Systems*, 2012.

[6] T. Point, Android Application Development, 2014.

[7] M. M. K. Suhas Holla, "ANDROID BASED MOBILE APPLICATION DEVELOPMENT and its SECURITY," *International Journal of Computer Trends and Technology*, vol. 3, no. 3, 2012.

[8] Mu Zhang, Heng Yin, Android Application Security, Spring, 2016.

[9] A. A.M. Memon, "tomorrow's mobile malware threat," *Colluding apps*, vol. 103, pp. 77-81, 2015.

[10] L. O. M. Security, "Look Out Mobile Threat Report," Augest 2011.

[11] "Statista," [Online]. Available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. [Accessed 12 5 2018].

[12] I. Muttik, "Malware mining," in *Proc. 21st Virus Bulletin International Conference*, Barcelona, Spain., 2011.

[13] shahid, "Mobile Application Development - An Insight," 12 10 2014. [Online]. Available: https://www.devsaran.com/blog/mobile-application-development-insight. [Accessed 2 10 2017].

[14] K. Divya and S. Venkata KrishnaKumar, "COMPARATIVE ANALYSIS OF SMART PHONE OPERATING SYSTEMS ANDROID, APPLE iOS AND WINDOWS," *International Journal of Scientific Engineering and Applied Science (IJSEAS)*, vol. 2, no. 2, pp. 432-438, 2016.

[15] A. A. Sheikh, P. T. Ganai,N. A. Malik,and K. A. Dar, "Smartphone: Android Vs IOS," *The SIJ Transactions on Computer Science Engineering & its Applications (CSEA)*, vol. 1, no. 4, pp. 141-148, 2013.

[16] "Windows Phones 8.1 Security Overview," Windows Phone ins, 2014.

[17] R. T, "Mobile OS - Features, Concepts and Challenges for Enterprise Environments," in *SNET Project Technische University*, Berlin, 2014.

[18] O. O. Okediran, O. T. Arulogun, R. A. Ganiyu and C. A. Oyeleye, "Mobile operating systems and application development platforms: A survey," *International Journal of Advanced Networking and Applications,* vol. 6, no. 1, pp. 2195-2201, 2014.

[19] Enck. W , Ongtang.M, and McDani, "lightweight mobile phone application certification," in *09 proceedings of the 16th ACM conference on computer and Communication Security.*, USA, 2009.

[20] Enc, William, et al, "A Study of Android Application security.," *USENIX security symposium*, 2011.

[21] "Android operating system framework architecture images," Google, [Online]. Available: https://www.google.co.uk/search?q=Android+operating+system+framework. . [Accessed 12 5 2017].

[22] Suleiman Y. Yerima, Sakir Sezer Igor Muttik, "A New Android Malware Detection Approach Using Bayesian Classification," 2016.

[23] J.Six, "Android Application Security: process, permissions, and other safe guards," *O'Reilly Media*, pp. 25-26, 2011.

[24] C. Lueg, "G Data Security Blog," Software AG, 27 04 2017. [Online]. Available: https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malware-samples-every-day. [Accessed 12 10 2017].

[25] V. Grampurohit, "Android App Malware Detection," Hyderabad, India, 2016.

[26] Fairuz Amalina Narudin · Ali Feizollah, Nor Badrul Anuar · Abdullah Gani, "Evaluation of machine learning classifiers for mobile malware," *Mobile Cloud Computing,* 2014.

[27] You Joung Ham, Daeyeol Moon, Hyung-Woo Lee, Jae Deok and Jeong Nyeo Kim, "Android Mobile Application System call Event Pattern Analysis for Determination of malicious Attack," *International Journal of Security and its Applications,* vol. 8, pp. 231-246, 2014.

[28] "Computer Virus Timeline," Sandbox Networks, Inc., publishing as Infoplease., 2000-2017. [Online]. Available: https://www.infoplease.com/science-health/computers/computer-virus-timeline. [Accessed 19 May 2018].

[29] F. Ruiz, "Securing Tommory Today," McAfee, 04 Oct 2012. [Online]. Available: https://securingtomorrow.mcafee.com/mcafee-labs/fakeinstaller-leads-the-attack-on-android-phones/. [Accessed 20 May 2018].

[30] J. F¨urnkranz et al, "Foundations of Rule Learning," *Cognitive Technologies,* pp. 1-17, 2012.

[31] P. Simon, Too Big to Ignore: The Business case for big data, Wiley: ISBN, 2013.

[32] "WikiVisually," Wikipedia, 4 May 2018. [Online]. Available: https://wikivisually.com/wiki/Machine_learning. [Accessed 5 May 2018].

[33] Marco Varone,Daniel Mayer, Andrea Melegari, "Expert System," 4 May 2018. [Online]. Available: http://www.expertsystem.com/machine-learning-definition/. [Accessed 5 May 2018].

[34] Taiwo Oladipupo Ayodele, "Types of Machine Learning Algorithms," in *New Advances in Machine Learning*, InTech, 2010, pp. 19-48.

[35] Abhishek Thakur, Artus Krohn-Grimberghe, "AutoCompete: A Framework for Machine Learning Competition," *International Conference on Machine Learning 2015*, vol. 1, p. 4, 8 Jul 2015.

[36] Kajaree Das, , Rabi Narayan Behera, "A Survey on Machine Learning: Concept, Algorithms and Applications," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, no. 2, pp. 1301-1309, 2017.

[37] Dr.Roger Brooks et al., "Guavus," Tholes company, 6 October 2017. [Online]. [Accessed 12 5 2018].

[38] D. Akhtar, Practical Reinforcement Learning, UK: Packet Publishing Ltd, 2017.

[39] K.-L. Du, M.N.S.Swamy,, "Fundamental of Machine Learning," in *Neural Networks and Statistical Learning*, Londol, Springer, 2014, pp. 15-65.

[40] A. N. a. M. Biswas, Reinforcement Learning, Kolkata,West Bengal, India, 2018.

[41] WIKIPEDA, "List of Machine Learning Concepts," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Outline_of_machine_learning.

[42] Settouti, N., Bechar, M.E.A, Chikh, M.A.:, "Statistical Comparisons of the Top 10 Algorithms in Data Mining for Classification Task.," *International Journal of Interactive Multimedia & Artificial Intelligence (IJIMAI)* , pp. 46- 51, 2016.

[43] R. Shipley, "The best anti-malware software of 2017," 2017.

[44] A. Aman, "A A Model Development for Software Defect Prediction: Using Feature Reduction Method," astu, Adama, 2018.

[45] F. Abdurahman, "Lightweight Security Auditing Tool for Android Smart Mobile," AAU, Addis Ababa, 2014.

[46] Zhaoguo Wang, Chenglong Li, Zhenlong Yuan, Yi Guan, Yibo Xue, "DroidChain: A novel Android malware detection method based on behavior chains," in *Pervasive and Mobile Computing*, 2016.

[47] Fauzia Idrees, Muttukrishnan Rajarajan, Mauro Conti, Tom Chen, Yogachandran Rahulamathavan, "PIndroid: a novel android malware detection system using ensemble learning methods," *Computers & Security*, 2017.

[48] Kabakus Abdullah Talha, Dogru Ibrahim Alper , Cetin Aydin , "Permission-based Android malware detection," *Digital investigation*, vol. 13, pp. 1-14, 2015.

[49] Long Wen, and Haiyang Yu, "An Android malware detection system based on machine learning," *AIP Conference Proceedings*, 2017.

[50] Shifu Hou, Aaron Saas, Yanfang Ye, and Lifei Chen, "DroidDelver: An Android Malware Detection System Using Deep Belief Network Based on API Call Blocks," *WAIM 2016 Workshops*, pp. 54-66, 2016.

[51] Dixon, B., Mishra, S., "Power Based Malicious Code Detection Techniques for Smartphone.," *In: 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 142-149, 2013.

[52] Fei Tonga, Zheng Yana, "A hybrid approach of mobile malware detection in Android," *Journal of Parallel and Distributed Computing.* , 2016.

[53] Wei-Ling Chang, et al, "An Android Behavior-Based Malware Detection Method using Machine Learning," *ICSPCC2016*, 2016.

[54] Mohsen Damshenas, Ali Dehghantanha, Kim-Kwang Raymond Choo & Ramlan Mahmud, "M0Droid: An Android Behavioral-Based Malware Detection Model," *Journal of Information Privacy and Security*, vol. 11, pp. 141-157, 2015.

[55] Shina Sheen n, R.Anitha,V.Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach," *Neurocomputing*, vol. 151, pp. 905-912, 2015.

[56] Tong Shen, Yibing Zhongyang, Zhi Xin, Bing Mao and Hao Huang, "Detect Android Malware Variants using Component Based Topology Graph," *In: 2014 IEEE 13th International Conference on Trust, Security and Privacy in computing and Communications (TrustCom)*, pp. 406-413, 2014.

[57] Sun, M., Li, X., Lui, J., Ma, R. Liang, Z., "Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android," *IEEE Transactions on Informantion Forensicsand Security*, vol. 12, no. 12, pp. 1103-1113, 2017.

[58] Amamra, A., Talhi, C. & Robert, J. M., "Smartphone Malware Detection: From a Survey Towards Taxonomy," 2012.

[59] E. M. V8, "Professionally diagram and communicate with essential Edraw solution," Edraw soft, 2004 - 2016.

[60] XDA-Developers, "xda," Leaseweb, 15 March 2017. [Online]. Available: https://www.xda-developers.com/decompile-and-modify-apks-on-the-go-with-apktool-for-android/. [Accessed 3 June 2018].

[61] Dinesh V. Rojatkar, Ganesh M. Jengathe,Ashwini B. Khairnar, Swapnil A. Lengure, "ANDROID APPLICATION DEVLOPMENT SOFTWARE – ANDROID STUDIO AND ECLIPSE," *INTERNATIONAL JOURNAL FOR ENGINEERING APPLICATIONS AND TECHNOLOGY*, vol. 3, pp. 9-12, 2016.

[62] M. Solutions, "Advantages and Disadvantages of Python Programming Language," 23 Apr 2017. [Online]. Available: https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121. [Accessed 3 Jun 2018].

[63] Mark Wilcox, Stijin Schuermans, Christina Voskoglou, and Alexandre Soboleviski, "State of the Developer Nation Q1 2017," 2017.

[64] Mila, "Cantiago Mobile," 2016. [Online]. Available: http://contagiominidump.blogspot.com/2016/. [Accessed 05 7 2017].

[65] V. Manjunath, "Reverse Engineering of Malware on Android," *Info Scince*, 2011.

[66] AWS, "Amazon Machine Learning Developer Guide," August 2nd, 2016.

[67] C.G Thomas, Robin Thomas Jose , "A Comparative Study on Different Hashing Algorithms," in *International Journal of Innovative Research in Computer and Communication Engineering* , 2015.

[68] Abawajy, J., Kelarev, A , "Iterative Classifier Fusion System for the Detection of Android malware," *IEEE Transaction on Big Data,* vol. 5, pp. 1-12, 2017.

[69] Dr. Sudhir B. Jagtap, Dr. Kodge B.G, "Census Data Mining and Data Analysis Using WEKA," in *International Conference in " Emerging Trends in Science, Technology and Management* , Singapore, 2013.

[70] "PC," Encyclopedia, [Online]. Available: https://www.pcmag.com/encyclopedia/term/43578/function. [Accessed 27 4 2018].

[71] I.Muttik, "Malware mining," in *21st virus Bulletin International Conference,* Barcelona, oct.2011.

[72] Wei-Ling Chang, et al, "An Android Behavior-Based Malware Detection Method using Machine Learning," *IEEE,* 2016.

[73] F. Abdurahman, "Lightweight Security Auditing Tool for Android Smart Mobile," AAU, Addis Ababa , 2014.

Appendixes

Appendix A: Sample Code for local database connectivity.

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mDBHelper = new DatabaseHelper(this);

        try {
            mDBHelper.updateDataBase();
        } catch (IOException mIOException) {
            throw new Error("UnableToUpdateDatabase");
        }

        try {
            mDb = mDBHelper.getWritableDatabase();
        } catch (SQLException mSQLException) {
            throw mSQLException;
        }
```

Appendix B: Sample Code for detection using signature database on device.

```
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String value1 = editText.getText().toString();
                String product = "";

                Cursor cursor = mDb.rawQuery("SELECT status FROM
application_info where signature ='"+value1+"'", null);
                cursor.moveToFirst();
                while (!cursor.isAfterLast()) {
                    product += cursor.getString(1) + " | ";
                    cursor.moveToNext();
                }
                cursor.close();

                textView.setText(product);

                if(textView!=null){
                    textView.setText("THIS APPLICATION IS MALWARE
DON'T INSTALL IT!!!");
                }
                else {
```

```
connectToServer("C:\\Users\\user\\Desktop\\Thesis
Prototype\\com.uc.browser.en_v10.7.8-101_Android-2.3.apk");
                }
            }
        });
        generate_btn.setOnClickListener(this);

    }

    @Override
    public void onClick(View v) {

        if(v.getId()== R.id.generate) {

            SHA256("C:\\Users\\user\\Desktop\\Thesis
Prototype\\com.uc.browser.en_v10.7.8-101_Android-2.3.apk");
            String value1 = editText.getText().toString();
        }

    }
    public String SHA256(String hash_value) {
        try {
            java.security.MessageDigest md =
java.security.MessageDigest.getInstance("SHA-256");
            byte[] array = md.digest(hash_value.getBytes());
            StringBuffer sb = new StringBuffer();
            for (int i = 0; i < array.length; ++i) {
                sb.append(Integer.toHexString((array[i] & 0xFF) |
0x100).substring(1,3));
            }
            String mdvalue = sb.toString();
            editText.setText(mdvalue);
            return mdvalue;

        } catch (java.security.NoSuchAlgorithmException e) {
        }

        return null;
    }

    private void connectToServer(String s) {
    }
}
```

Appendix C: Sample Code to access SQLite database.

```
public class DatabaseHelper extends SQLiteOpenHelper {
    private static String DB_NAME = "MyExternalDatabase.db";
    private static String DB_PATH = "";
    private static final int DB_VERSION = 2;

    private SQLiteDatabase mDataBase;
    private final Context mContext;
    private boolean mNeedUpdate = false;

    public DatabaseHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
        if (android.os.Build.VERSION.SDK_INT >= 17)
            DB_PATH = context.getApplicationInfo().dataDir +
"/databases/";
        else
            DB_PATH = "/data/data/" + context.getPackageName() +
"/databases/";
        this.mContext = context;

        copyDataBase();

        this.getReadableDatabase();
    }

    public void updateDataBase() throws IOException {
        if (mNeedUpdate) {
            File dbFile = new File(DB_PATH + DB_NAME);
            if (dbFile.exists())
                dbFile.delete();

            copyDataBase();

            mNeedUpdate = false;
        }
    }

    private boolean checkDataBase() {
        File dbFile = new File(DB_PATH + DB_NAME);
        return dbFile.exists();
    }

    private void copyDataBase() {
        if (!checkDataBase()) {
            this.getReadableDatabase();
            this.close();
            try {
                copyDBFile();
            } catch (IOException mIOException) {
                throw new Error("ErrorCopyingDataBase");
            }
        }
    }
```

```
    private void copyDBFile() throws IOException {
        InputStream mInput = mContext.getAssets().open(DB_NAME);
        OutputStream mOutput = new FileOutputStream(DB_PATH +
DB_NAME);
        byte[] mBuffer = new byte[1024];
        int mLength;
        while ((mLength = mInput.read(mBuffer)) > 0)
            mOutput.write(mBuffer, 0, mLength);
        mOutput.flush();
        mOutput.close();
        mInput.close();
    }

    public boolean openDataBase() throws SQLException {
        mDataBase = SQLiteDatabase.openDatabase(DB_PATH + DB_NAME,
null, SQLiteDatabase.CREATE_IF_NECESSARY);
        return mDataBase != null;
    }

    @Override
    public synchronized void close() {
        if (mDataBase != null)
            mDataBase.close();
        super.close();
    }
    @Override
    public void onCreate(SQLiteDatabase db) {

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        if (newVersion > oldVersion)
            mNeedUpdate = true;
    }
}
```

Appendix D: Sample Code for Server side.

```
import pprint
import re
import hashlib
import numpy as np
import xml.etree.ElementTree as ET
from numpy import genfromtxt
from pymongo import MongoClient
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import classification_report
from sklearn import preprocessing, svm

#Extracting android.permission features from the Android Manifest file of Facebook.apk
#Feature_extraction Function{
MONGODB_URI= "mongodb://glucose:guleex2119@ds137019.mlab.com:37019/maliciousapp"
client = MongoClient(MONGODB_URI, connectTimeoutMS=30000)
db = client.get_database("maliciousapp")
Application_Type = db.Application_Type
print("Successfully connected with CloudDatabase!")

#A METHOD USED TO GENERATE THE SHA-256 OF APK FILE
filename = input("Enter the input file name: ")
def sha256_checksum(filename):
    sha256_hash = hashlib.sha256()
    with open(filename,"rb") as f:
        # Read and update hash string value in blocks of 4K
        for byte_block in iter(lambda: f.read(4096),b""):
            sha256_hash.update(byte_block)
            hash1=sha256_hash.hexdigest()
       # print(sha256_hash.hexdigest())
        return hash1
record = {
   "malicious": sha256_checksum(filename),
}
hash_value=sha256_checksum(filename)
```

```
#A METHOD USED TO LOOKUP THE HASH VALUE IN MALWARE ATTRIBUTE
def chck_malware(hash):
    result=Application_Type.find({"malicious":hash_value})
    if result.count()>0:
        msg = "Malware"
        print(msg)
    else:
      chck_normal(hash)

#A METHOD USED TO LOOKUP THE HASH VALUE IN NORMAL ATTRIBUTE
def chck_normal(hash):
    result=Application_Type.find({"normal":hash_value})
    if result.count()>0:
        msg = "Normal"
        print(msg)
    else:
        Feature_extraction()
        array3=Vector_Generation()
        Predict_App(array3)

#A METHOD USED TO PUSH THE RECORD TO MALWARE ATTRIBUTE
def pushRecord_MALICIOUS(record):
    result=Application_Type.find({"malicious":hash_value})
    if result.count()>0:
        print ("Exists in DB")
    else:
        Application_Type.insert_one(record)

#A METHOD USED TO PUSH THE RECORD TO NORMAL ATTRIBUTE
def pushRecord_NORMAL(record):
    result=Application_Type.find({"normal":hash_value})
    if result.count()>0:
        print ("Exists in DB")
    else:
        Application_Type.insert_one(record)

#A METHOD USED TO EXTRACT FEATURE FROM AndroidManifest.xml
def Feature_extraction():
    file2 = open("ExtractedPermission.txt", "w")
    root = ET.parse("AndroidManifestF.xml").getroot()
```

```
    permissions = root.findall("uses-permission")
    for perm in permissions:
        for att in perm.attrib:
            permission= perm.attrib[att]+'\n'
            print("{}\n".format(perm.attrib[att]))
            file2.write(permission)
    file2.close()

#A METHOD USED TO CONSTRUCT FEATURE VECTOR
def Vector_Generation():
    array1 = []
    array2 = []
    array3 = []
    with open("StandardPermissions.txt", "r") as s:
        for line in s:
            array2.append(line)
    with open("ExtractedPermission.txt", "r") as f:
        for line in f:
            array1.append(line)
        i = 0
        j = 0
        for i in range(0, len(array2)):
            for j in range(0, len(array1)):
                if array2[i] in array1:
                    fi = 1
                else:
                    fi = 0
                j += 1
            array3.append(fi)
            i += 1
        print("====================FEATURE VECTOR OF THE APPLICATION==============")
        print(array3)
        return array3

#A METHOD USED TO CLASSIFY THE ANDROID APPLICATION
def Predict_App(array3):
    print("=========================CLASSIFICATION INFO==============")
    df = pd.read_csv('datasets.csv')
    df.replace('?', -99999, inplace=True)
    x = np.array(df.drop(['Class'], 1))
    y = np.array(df['Class'])
```

```
    x_train, x_test, y_train, y_test, = cross_validation.train_test_split(x, y, random_state=0)
    svma=svm.SVC()
    svma.fit(x_train, y_train)

    detection = np.array(array3)
    detection = detection.reshape(1, -1)
    prediction = svma.predict(detection)
    if prediction == 1:
        msg= 'This appliction is normal'
        print(msg)
        pushRecord_NORMAL(record)
    else:
        msg= 'This appliction is malicious'
        print(msg)
        pushRecord_MALICIOUS(record)
    return msg
sha256_checksum(filename)
chck_malware(hash)
```