Addis Ababa University

College of Natural And Computational Sciences

An Application Classification Framework for Information Leakage Detection on Android Platform.

Teklu Kuma Geda

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia
11-June-2021

Addis Ababa University
College of Natural Sciences

Teklu Kuma Geda

Advisor: *Dagmawi Lemma(PhD)*

This is to certify that the thesis prepared by *Teklu Kuma* , titled: *An Application Classification Framework for Information Leakage Detection on Android Platform* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name Signature Date .

Advisor: _____

Examiner:_____

Examiner:_____

Abstract

The growing of android based smartphone popularity is one of the reasons which is attracting the distribution of information stealing applications developed by attackers. As the latest android operating system versions are being updated to detect vulnerabilities, malware applications are shifting their patterns from looking malicious to looking like a good-ware application in order not to be detected easily. The use of machine learning is adapted in various information leakage detection techniques. Machine learning classifiers are widely used to model Android information leakage patterns based on their static features and dynamic behavior.

In order to overcome the problem of information leaking applications detection, in this thesis we proposed a machine learning based information leakage detection mechanism. Our proposed system utilizes the extracted features of samples of good-ware and malware applications to train classification model. The system extracts requested permissions, vulnerable application program interface calls, system calls sent in 30 seconds and intents, and uses them as features in various machine learning classifiers to build classification model. After performing various comparative analysis among classification algorithms and performance validation, we achieved high classification accuracy of 99.8 % using our high performing classification model.

Using the model as one of the major components, we have designed the classification framework to classify a random application as a leaker or non-leaker by extracting its feature at different state and add the extracted feature into the dataset of our classification model, since we have used incremental supervised learning. Using incremental supervised learning is helping our classification model to improve its performance from time to time as more applications are getting classified by our framework.

Keywords: Machine Learning, leakers, Anomaly detection

Dedication

To my Parents and Family

Acknowledgments

First of all I would like to thank my father Obbo Kuma Geda and My mother Adde Hawi Alemu, for always believing in me and encouraging me although they have no idea with what I am struggling. Secondly, I want to say “Thank you” to my advisor Dagmawi Lemma(PhD) for his continuous advisory and for being a great role model to me.

My gratitude also goes to my wife Martha Getachew and my daughter Fenet Teklu for being patient with me, encouraging and loving me. At last but not the least I want to thank my Sisters Dr. Tirsit, Eleni, and Brothers Melaku, Fikru, and Colleagues Fasil and Fuad for their continuous help and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ACRONYMS

ANN	Artificial Neural Network
DT	Decision Tree
IDS	Intrusion Detection System
IPS	Inter Process Communication
KNN	K-Nearest Neighbors
MIPS	Microprocessor Without Interlocked Pipelined Stage
NB	Naïve Bayes
OTP	One Time Password
PII	Personally Identifiable Information
RF	Random Forest
SMS	Short Message Service
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TPR	True Positive Rate
USSD	Unstructured Supplementary Service Data

1. INTRODUCTION

1.1 Background

Electronic fraud (or e-fraud for short) is a fraudulent action that is perpetrated using electronic technologies and devices, such as computers, Internet, email, telephone networks, and mobile phones according to Chen *et al.*[1]. Today's smart phones can be vulnerable to fraudulent action, through malwares such as virus, Trojan horse, Rootkits and information-leakage.

Leaker applications are an accessory used in e-fraud through which hackers are able to intrude into a secured system through users smart phone. In this thesis work, leaker application is used to refer a mobile application that snoop sensitive information on a smart phone and help hackers to intrude into secured system.

Mobile applications can be provided by various organizations to outreach services and enhance businesses. Automation of marketing and financial transaction can be achieved through development of mobile application to make the business easy and nearer to the user. As security is one of the reasons for automation in a lots of businesses, security demands a concern in application and overall platform design. Android operating system has a security mechanism to manage the activities of an application. Every android application should fulfill the security protocol set by operating system to get installed on the phone. If an android application to be installed shows a demand to access different services, data or information; an authority of permission granting or denial will be brought to the user.

Anti-malware and anti-virus applications are used to prevent and report commonly known security breaching attempt by malicious application. Applications which are developed for information leakage could be developed with the properties similar to good ware application in order to pass the security gate. Since, anti-malware and anti-virus applications commonly depend on patterns to identify the malicious activity, if an application will not show any commonly identifiable patterns then the security gate will be vulnerable.

Viber is one of well-known social networking application which was developed in 2010 and it is widely accepted social networking application all over the world. Viber was developed by MOSAD employee for the purpose of information gathering and lately sold to CIA for personal information tracking. Viber was known as a social networking application by user and it is used

by service providing vendor for gathering user's information in hand with social networking functionality according to Crager *et al* [2].

Information leaking applications commonly exist in the form of an application which is providing various services like, gaming, entertainment and photo editing to the user and perform leakage activities in hand, without the knowledge of the user. Mobile applications which are performing leakage activity are programmed not only to sniff the flowing information, but further could compromise sensitive information like bank information, personal data and password for e-fraud action.

Different research works have been done regarding with detection and prevention of leakage applications on android platform., but the difficulty of detecting and preventing them using a single software i.e. antivirus, antimalware, etc. still exists. Detection and prevention difficulty is because of the property an application shows before execution and during execution. Antimalware and Antivirus depends on patterns an application show before execution, but most of an application with leakage property could not show any malware or virus resembling patterns before execution.

This thesis work is proposed to come up with feature analysis of an android application before execution, during execution and during inter-process communication to classify an application as a leaker or not depending on the dataset that will be prepared from an extracted feature.

1.2 Motivation

Information leakage is accessory to electronic fraud which potentially leads to disagreement between the customer and the service provider. This in turn may lead to the company business destruction because of the service providing companies loss of trust from their customers.

Hackers can create a mobile application disguising useful service giving application, i.e. game application, entertainment application and etc. And, having such applications programmed with property of leakage information which will be used for the fraudulent action. During Installation of such malicious application, the smartphone might ask for the user's confirmation to install or not to install showing some of the privilege requirement by an applications. At this time, users are not commonly paying attention to the security details instead they will be eager to see what functionalities the application provides to them. Even if the user is able to give attention to the security alert from android platform, nowadays security alert on smartphone is like "The

application is from unknown source ". This does not mean the user should not have to use the application. Because, locally android applications can be provided by different business vendor to make life easy.

On the other hand the user can install the application not because he/she trust the application, but because he/she trusts the business vendor which provided the application. What if the business vendor add the leakage property to the application?

So, this work is initiated to create an efficient way of detecting information on android platform by tracking the intents that a particular application is sending to other application during inter-application and intra-application communication, and the system call that an application send to kernel for starting new child or parent process.

1.3 Statement of The Problem

Leaker mobile application can show the property of any other good ware application which will make it difficult to be detected easily. This is because of malware writers do use dynamic and reflection code to make application statically undetectable and also use crypto code for code obfuscation according to Gibler [3].

Different researches have been done recently to ease the detection of information leakage on an android platform, i.e. Drebin uses a bulk set of dynamic and static features to detect leakage and Li-Dai *et al.*[4]uses permission and API calls only. Besides the effective detection mechanism the author in uses, including system call tracking and dynamic permission could narrow the hole through which a leaker application could disguise the good ware one.

The real problem is when an application passes the security gate of an android platform by showing the good ware property like any other application and changes the behavior like using of dynamic permission, and sending of process creating system calls to the operating system with or without the knowledge of the user. If we track the intents from source to sink and follow process creating system calls, and use the dynamic permission request from each application and process the collected feature data we could detect the leakage attempt.

Victims of leaker applications can be exposed to leakage as most of security guaranteed mobile applications can be downloaded from some of the trusted service provider and there are also

applications which the user need to get from any other service providers for the sake of using the service.

When a user needs to install an application, the mobile security system will alert the user if the application is not from trusted source by giving an option to the user whether to install or not. But, suppose the application is from an organization which the user trusts and installs the application without any hesitation. Here, the user trusts the service providing vendor not the application.

- What if, the application contains property of information leakage and expose the users information and privacy to the hackers?
- What if, the service providing organization intentionally include the leakage activities in the application?

Information leakage applications are applications which provide different valuable services to the user such as, easy financial operations, personal profile storage and etc. On the other hand, leakage application sniff information for the hacking purpose and can take control over the user's device through the applications it is programmed with, and perform information theft according to the definition of Li-Dai *et al.*[4].

Information leakage applications can lead customers to lose trust on service providing organization. Customers can think like service giving organization is doing something wrong with the services they deserve to get, where service providing organization is thinking like the customers have done it wrong. For instance, information leakage applications can mimic the activity of the smartphone user and let the hackers to inter into the banking system through their phone; and transfer balance from the users bank account using user's password and other required information. Since, the transfer activity is not really performed by the user, the user will not have the knowledge of what is going on, so he/she will be able to complain and accuse the service providing bank for the lost money. This will lead the customer to lose trust on a bank.

1.4 Objectives

General Objective

The objective of this thesis work is to develop a framework for classification of an android application as a leaker or not, depending on their feature before execution and, during execution and inter-process communication.

Specific Objectives

For achievement of the general objective, there are specific objectives need to be achieved. i.e.

- Review of related literature.
- Extracting features of an application.
- Extracting system calls for creation and termination of processes for an application..
- Performing comparative analysis among three different classification algorithms using an extracted feature.
- Developing a framework of proposed detection mechanism of leakage application.
- Creating classification model which will be a component of our framework
- Evaluating the performance of the classification model

1.5 Methods

The methods that will be used in order to achieve the objectives are: Literature review, analysis of existing classification model and framework for classifying an application depending on properties and detailed study of an existing information leakage detection mechanism.

A. Literature Review

Related literatures on information will be reviewed from different sources such as published papers written by a variety of scholars and other materials to obtain understanding of the area in detail.

B. Simulation

Features of an application during execution and before execution is a major dataset for designing a classification model and an entire framework. In order to get those features and get an application into execution genymotion virtual android operating system simulation tool will be used.

C. Experimentation

Experimentation for preferring best classification algorithm and performing comparative analysis among the classification algorithms will be performed to recommend for our framework will be performed in this thesis work. Experimentation will also be used to create a training model which will be our major component of the framework.

1.6 Scope and Limitation

This research work is proposed to develop a framework through which an android application can be classified as leakage or not depending on their property before starting execution or under execution. The scope of this research is limited to:

- Identification of the privileges provided to an application during installation.
- Following if an application is using only permitted services.
- Tracking an application from pre execution to execution and collecting the features as a dataset.
- Usage of classification methods and classifying if an application is leaker or not depending on the provided dataset.
- Performing comparative analysis and recommending classification algorithms for similar problem.

The existing leakage problem and the proposed solution considers only android applications which will be attacked by sniffer applications. This is because as per 2014 statistics about 81.5% of the global mobile phones are using android operating system according to Malik *et al* [5].

The proposed work will not include the capability of identifying the leakage application and taking action on behalf of the user than classifying it as a leaker or not.

1.7 Application of Results

Information leakage applications which uses sensitive data from smartphone will affect both individuals and service providing organization. So, individuals who are using smartphone needs to have these application for protecting themselves from both information and financial theft. Besides, an organization which are providing anti-malware and anti-virus software will also be benefited from using this framework.

Banking service provider and financial institutions which are trying to make their service nearer to the customer will also be benefited from knowing whether an application is classified as a leaker or not. The hesitation of using different beneficial android applications like, mobile banking apps, personal photo gallery apps and etc. will also be enhanced depending on the functionality of this work.

1.8 Thesis Organization

The remaining part of the thesis is organized as follows: Chapter 2 starts with literature review on existing android security frameworks, architectures of leakage detection frameworks, nature of leaker application and features of preferred classification algorithm are presented. Review of related works is presented in Chapter 3 of the thesis. Works that have significant relation with this thesis are assessed. In Chapter 4 the framework design for classification of an application depending on their feature and behavior is presented. Each components of the architecture are discussed in details. Chapter 5 presents an implementation and experimentation of the proposed framework. Finally conclusion and future works are presented in Chapter 6.

2. LITERATURE REVIEW

Privacy leakage detection frameworks have been implemented by different researchers using different mechanisms. Signature based detection is one of those mechanisms which depends on creating a patterns for any behavior an application shows and including those patterns on anti-virus for preventing the intrusion and detecting them even after they got installed as pointed out by Egele *et al.*[6].

Android platform has its own security mechanisms, although most of privacy leaker application developers targeted security breaching methods especially for android system security.

2.1 Android Security Mechanism

Android security mainly focuses on protecting the users data, system resources and application isolation. Android uses sandbox, application signing and permission model for ensuring the security of the system according to Shabtai *et al*[7].

A. Sand Box

Isolation means by which Applications can be isolated from one another. Unique UID will be assigned for an application during installation. Set of permissions to access application files will be given during installation. Applications other than those who are permitted to be accessed won't access the application file as pointed out by Shultz *et al*[8]. If there is a possibility by which applications can be developed by the same developer and should access a specific files then they should be assigned the same UID. When the application is installed on the device, it runs in its own sandbox and other applications cannot access or interfere it as pointed out by Sami *et al.*[9].

B. Application Signing

Application signing is used to ensure the application security. It creates a certification between developers and their applications. Before placing an application into its sandbox, the application signing creates a relationship between the UID and the application. The applications couldn't be run on the Android without signing. With the same UID, that is, running in the same sandbox, the applications can share the permissions and communicate with each other. By using application signing, the application update process can be simplified. Since different versions of the same

application have the same certificate, the package manager can verify this certificate. Then, the old version is replaced, the new version can have the permissions already granted to the old version. What's more, the application signing can also ensure that an application cannot communicate with another unless using the ICC.

C. Permission Model

Permissions mechanism is used to make some restrictions when the applications want to access the sensitive API of the Operating system. An application is isolated when it is executed in sandbox. When the applications want to access some sensitive features such as, cameras, location, telephony and network. Android provides a permission model to achieve these goal.

The permissions have four levels, normal permissions, dangerous permissions, signature permissions and signature/system permissions. Normal permissions can be granted automatically; dangerous permissions are inferred to those granted by the users; signature permissions are granted within the same sandbox; signature or system permissions are granted to pre-installed applications or the applications installed by the root.

2.1.1Application Components and APK file Architecture

I. Android Application Components

For an application to be executed on an android platform it need to have a components like Activities, services, content provider, Broadcast Receivers and Intents.

- **Activities**

Usually an application contains list of Activities. It represents the Visual view of an application . Activities are an instances which are loaded every time the user is trying to interact with the application in a foreground. Activities can be termed as a face of an application. Activities involve various states. i.e.

- An activity is present in active or running states if it is in a foreground.
- An activity is in a paused state if it has lost a focus but still visible to an end user.
- An activity is in a stopped state if it is no longer visible to an end user.

Under the low memory conditions, Android system will often kill these no longer used apps to

free the memory so that memory can be used by other active applications. An android system contains various methods to facilitate the storage of state information of an activities on an android activity API.

- **Services**

Services are components that run in the background to perform long-running operations such as playing music, handle network transactions, interacting content providers etc. It does not have any UI. The music application, for example, will have a music service that is responsible for playing music in the background while the user is in a different application. Services can be started by other components of the app such as an activity or a broadcast receiver. Moreover, service can run in the background indefinitely even if the application is destroyed.

- **Content Provider**

Content providers make a specific set of the application's data available to other applications. They manage a shared set of application data. For example, contact information data is stored in a content provider so that other applications can query it whenever they require. A music player may use a content provider to store information about the current song being played, which could be further used by a social media application to update the user's 'current music playing' status.

- **Broadcast Receiver**

Broadcast receivers respond to broadcast messages from other applications or from the system itself. These messages are sometimes called events or intents. For example, the SMS app broadcasting information about an SMS has being received and let other applications know about the ongoing event. Broadcast receivers do not have a user interface and are generally used to act as a gateway to other components. They might, for example, initiate a background service to perform some work based on a specific event.

- **Intents**

Intents are asynchronous messages which allow application components to request functionality from other Android components. Intents allow users to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.

II. APK file Architecture

As the above listed components are the component of an android application, there are also components of APK file specifically. APK packages comprise of the key component of an Android

application which is the dex file created after compiling the bytecode of all the java source files in Android.

- **Android Manifest**

Android Manifest contains an essential information about applications to an android system. This information that is contained in android manifest is an information that the system must have before it can run any of an application's code. Every application comes with an AndroidManifest.xml file that informs the system about the app's components. The AndroidManifest.xml file contains information about application package, including components of the application such as activities, services, broadcast receivers, content providers etc. The Androidmanifest.xml also specifies application requirements such as special hardware requirements (e.g., accessing camera or GPS sensor), or the minimum API version necessary to run an app. To access protected components (e.g., location access, or access to sd card), an application needs to be granted permission. All necessary permissions must be defined in the app's AndroidManifest.xml. This way, during installation, the Android OS can prompt the user with an overview of used permissions after which a user explicitly has to grant the app access to use these components.

lib.

This directory usually contains compiled the code of various supporting libraries which are referred usually in the application components. This directory, in turn, has different directories representing the processor base for which the libraries are compiled for e.g., arm, x86, MIPS etc.

res.

This directory contains the resources which are used in the application. An example of resources can be various images required in the UI layout of an Activity.

assets.

This directory contains application assets. These can be accessed programmatically in Android using AssetManager.

classes.dex.

The .dex file generated after compiling the bytecodes of all the java source files in the applications.

resources.arsc.

This is a precompiled binary of the contents of resource directory mentioned above.

META-INF. In order for the system to maintain a unique identity of authors of applications, apps are required to be signed before installation in android. The contents of this directory contain the manifests and certificates of its digital signature.

2.2 Privacy leakage Detection

Leakage/privacy leak is any transfer of personal or phone-identifying information of the phone according Sanz *et al.* [11]. A sniffer is a piece of software that grabs all of the traffic flowing into and out of a computer attached to a network as defined by Chen, *et al.*[1].Smart phones can be considered as one of the computing device that will be connected to the network in order to perform different operations like, processing banking information, interacting with people in social media, updating the existing dynamic applications.

Transmission of sensitive data by itself may not be considered as a leakage or privacy leak. Whether it is privacy leak or not will be decided depending on if the transmission is user intended or not. There is a possibility of users sending sensitive data outside. But if applications are sending users information it is not intended transmission which means the application is sniffer.

The irregular transmission of sensitive data performed by an app, which is unknown to users and irrelevant to the function user enjoys, is defined as unintended data transmission, or privacy according to Koli [12].

Sniffers are available for several platforms in both commercial and open-source variations. Some of simplest packages are actually quite easy to implement in C or Perl, use a command line interface and dump captured data to the screen. More complex projects use a GUI, graph traffic statistics, track multiple sessions and offer several configuration options. Sniffers are also the engines for other programs. Intrusion Detection Systems (IDS) use sniffers to match packets against a rule-set designed to flag anything malicious or strange. Network utilization and monitoring programs often use sniffers to gather data necessary for metrics and analysis. Law enforcement agencies that need to monitor email during investigations, likely employ a sniffer designed to capture very specific traffic Chen, *et al.* [1].

The other way of privacy leakage depends on the capabilities that are possessed by modern mobile phones, i.e. communication, computing and sensing as pointed out by Gibler *et al.* [3]. The sensing capabilities of mobile phones come from the audio, video, and location sensors in the form of

microphones, cameras, and GPS receivers. While these sensors enable a variety of new applications, they can also seriously jeopardize user privacy which is leakage.

2.2.1 Methods of detecting privacy leakage

Some of an android applications has a features of frequently sending users' private data outside the device carelessly or intentionally. A large number of methods have been proposed to detect these privacy leakage, including static and dynamic analysis methods according to Crager, *et al.*[2].

Those two methods are only able to identify part of private data vulnerabilities due to the dynamic features in codes and code coverage problems. AndroidLeaks, a static analysis framework for automatically finding potential leaks of sensitive information in Android applications on a massive scale. information, they may leak it carelessly or maliciously. Google's Android operating system provides a permissions-based security model that restricts an application's access to the user's private data. Each application statically declares the sensitive data and functionality that it requires in a manifest, which is presented to the user upon installation. However, it is not clear to the user how sensitive data is used once the application is installed according to Hintea *et al.*[4].

The main methods of analyzing whether an application is leakage/leaking the PII(Personally Identifiable Information) or not is to use the analysis method. Concerning this, the developers built framework tools according to two methods: i.e. Static Analysis and Dynamic Analysis according to Malik *et al.*[5].

Before analysis, detection of the intrusion is the main concern. There are two types of intrusion detection techniques i.e. misuse detection technique and anomaly detection technique. In misuse detection technique the intrusion is defined based on predefined signature. In anomaly intrusion detection technique we analyze the behavior of intrusion through the events raised by intrusion and detect intrusion.

In anomaly detection technique, the software analysis is of two types, Static analysis and Dynamic analysis.

Static analysis is an analysis method for mobile application in non-runtime environments. Egele, *et al.* [6] presented PIOS using this analysis to detect . During the experimentations, the researchers

tested more than 1400 iPhone apps and they found that more than half of them had leaked the unique ID of the mobile devices. Therefore, this data allows third parties to create users' profile without their knowledge. It is worth nothing that in this experiment, some apps were tested that were on jailbreak iPhone firmware and did not have any privacy. In static analysis frameworks, analysis is conducted by disassembling the APK packages and analyzing the decompiled code.

Dynamic analysis is the real time analysis method consisting in testing and analyzing software or mobile apps to detect the vulnerabilities. Shabtai, *et al.*[7] identified the privacy of 226 Android apps. The results have been divided into three categories, and the report shows that 46.5% of apps leaked private data.

On the other way although some of the literatures suggest usage of constructed effective behavioral representation will help to identify and detect the leaking application on an android platform. This mechanism uses the system call to depict the software behavior because the usage of system API captures the characteristics of the interaction between an app and the android system as pointed out by Schultz, *et al.*[8]. However, system calls do not appear to be a good standard for judging malicious/ behaviors because it is hard to distinguish the high privileged apps from malicious applications according to Sami, *et al.*[9].

2.2.2 Privacy Preventive Frameworks Deployment

Researchers are using various deployment methods for the implementation of frameworks which are developed for analysis and detection of leaking property of applications on an android platform.
Android application: Frameworks that will analyze and detect regular Android applications can be deployed as an android application.
APK file: Frameworks that provides an Android application (APK) for deployment.
OS modification: Frameworks that are deployed on Android OS. In this case, the Android OS needs to be patched with framework files. For example, Taint-Droid and IntelliDroid require modification of Android OS.
WebApp/Script Proposals that are available in the form of the web application or scripts/package.

2.2.3 Leaking Application analysis and Detecting Frameworks

Although there is further categorization of privacy frameworks, they can be viewed as two. i.e. Analysis and Detecting frameworks.

A. Analysis Frameworks

Frameworks which helps in analyzing android applications for privacy sensitive leaks, application internal mechanism. eg.(Intellidroid and TaintDroid both belongs to this category).

B. Detection Frameworks

Frameworks that help in detecting and reporting privacy leaks or malicious behavior in android application(eg. TaintDroid). Some of the frameworks which are implemented by researchers, for analysis and detection of privacy in android platform are; Intellidroid, Droidsafe, Flowdroid, DroidAPIMiner, DroidRanger, RiskRanker, Androguard, DroidChecker and etc. Those frameworks are mainly Statically analyzing frameworks. Still there are frameworks which are analyzing and detecting privacy s through dynamic analysis which involves runtime analysis and detection of android applications.

Researchers who are associated with each of the frameworks have been listed out the behavior and tested result of the frameworks. However, we intended to deal with some of the frameworks which are more related to the detection and prevention of privacy leakage in android platform.

I. TaintDroid Leakage Detection Framework

Taintdroid which is privacy leakage detection mechanism, performs the evaluation on 30 android applications out of 358 applications which require Internet access for performing an activity from 1,100 applications that were freely available. According to the evaluation result android applications can use Location, Camera, Audio and Phone state, depending on the permission that can be granted by the user during installation. After dynamic taint analysis performed, some of the applications sent out the phone number, IMSI, and ICC-ID along with the geo-coordinates to the app's content server according to Naser P, *et al.2*[10].

Most of the reviewed works have been able to analyze an application either statically or dynamically. This analysis can be effective to some extent. i.e. Statically analyzing detection mechanisms will help by deciding whether an application contains leaking property or not before installing on a device. But the problem is, an application is not analyzed on a real device, instead it is analyzed using a simulation devices which will not guarantee that if an application will start leaking after installation. Similarly, dynamically analyzing detection mechanisms can detect

whether an application has a property or not by analyzing an application on running. Even if this mechanisms is helping in detecting an application that is leaking during execution it will not prevent the execution. Of course, it might avoid further execution of an application after detection it's property, but it works on the detection level of security stage.

This work is mainly focuses on the prevention of leaking application from execution even after being installed. During installation the mobile system will notify the user as some of the applications need an access of some services from the device like; microphone, location , audio and camera. The user has an authority to allow or deny the request. But some applications may change their behavior after passing the installation gate. Those, change involves access of services other than the permitted one after installation. So, our target is to follow the execution of an application and their need of access. Then after comparing the initially permitted service request with the recent need of service access, if the needed service access is not belongs to the already permitted list of services(Permission Catalog) then the user will be notified through AO(Alerting Operation).

TaintDroid is a framework which provides the analysis and detection of privacy leakage on Android platform. TaintDroid according to Naser P, *et al.* [10] is the first dynamic analysis engines introduced for Android Apps. It performs taint tracking to precisely analyze how private data is obtained and released at runtime. In achieving this, it adopts an efficient way to handle taint storage. It also defines taint propagation rules on Dalvik instructions across API calls. As TaintDroid handles taint analysis of Dalvik instructions across API calls at runtime, it is resistant to Java reflection and code encryption. In addition, TaintDroid can be loaded into real devices, allowing for real-time monitoring of actual hardware and sensors. These advantages have made TaintDroid be used widely in Android app behavior analysis.

TaintDroid is an easy to use with a static analysis tool and can be easily integrated with any supporting static analysis framework. For instance, IntelliDroid using TaintDroid can trigger and detect sensitive- privacy leaks. In original work, authors show that IntelliDroid's event chain detection and device framework interface input injection enabled it to effectively generate inputs that trigger targeted APIs in a corpus of malware.

TaintDroid is different from another frameworks as it is performing information flow tracking, dynamic taint tracking and analysis of the flowing information(taint).

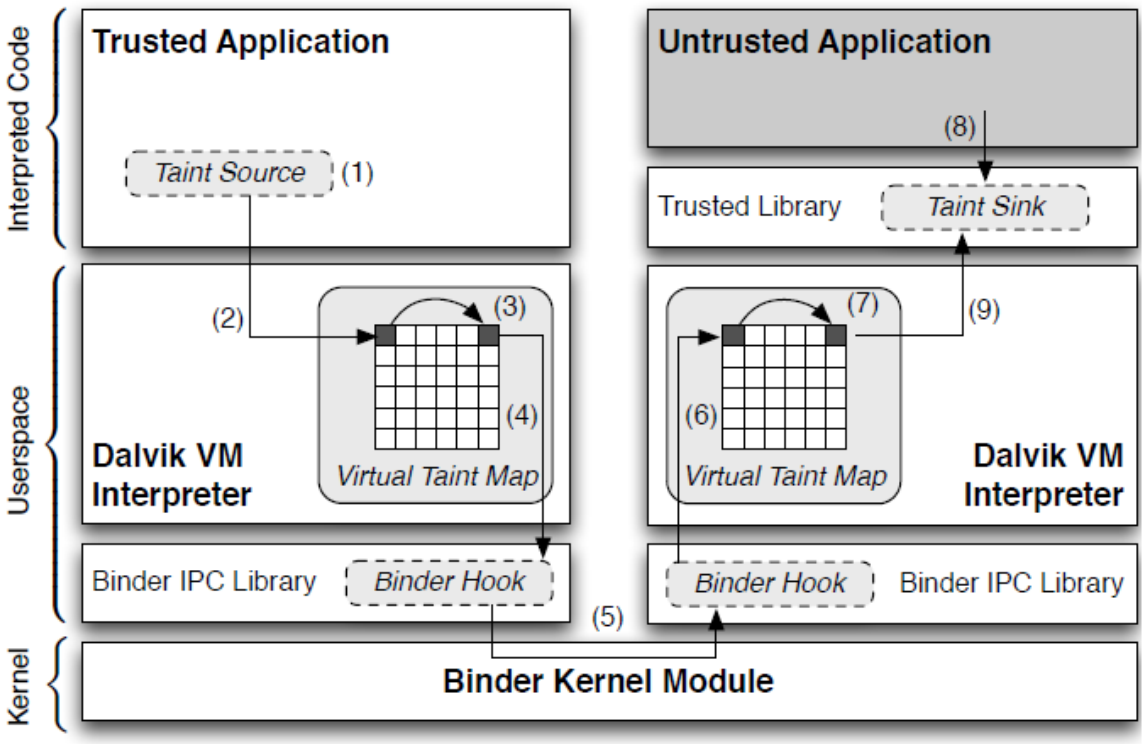

Figure 2.1:*TaintDroid Overall Architecture*

II. AppIntent Leakage Detection Framework

The basic idea of AppIntent is to use symbolic execution to generate aforementioned event sequence, but straight forward symbolic execution proves to be too time consuming to be practical. A major innovation in AppIntent is to leverage the unique Android execution model to reduce the search space without sacrificing code coverage according to Koli *et al*, [12].

For Identifying information flow and reporting as it is a privacy , first it should be identified whether the information transmission is user intended or not. Because, there is a possibility of users transmitting sensitive information. Unfortunately, due to the complex nature of user intention and different/unpredictable settings of different apps, it is almost impossible to have automated method to determine user intention. But, Alternatively it is practical to design an automated tool to provide a human analyst with the context information in which the data transmission occurs. Intuitively presented context information will make the task of the human analyst easier in determining if the transmission is user intended Koli *et al*,[12].

AppIntent derives the input data and user interaction inputs that lead to the transmission. The context information of the transmission shown to the analyst is in the form of a sequence of UI manipulations (i.e., GUI screens along with the highlighted GUI controls that indicate the supposed user operations) that is captured from a controlled execution of the app with the derived input data and user interaction. By looking at the displayed UI manipulations, a human analyst can then make a judgement.

Although AppIntent is helping in identifying whether an application privacy property is user intended or not, through human analysts; it is not possibly performing the prevention of privacy s. Instead, it is working on identification of privacy and classifying as whether transmission is user intended or not.

III. Glass Box Privacy Leakage Framework

This work is mainly focused on automatic analysis and detection of android applications before they are installed by the user. The author of this work proposes as analysis of an applications behavior on virtual device is a waste. Since, currently existing emulators cannot emulate SIM card, Camera, microphone and GPS. So usage of dynamic application analysis prototype is proposed. i.e. Glass Box. It executes applications on real devices in a monitored and controlled environment.

This prototype tested using various application testing strategies . i.e. UI brute force, Monkey, broadcast events, Real SMS/Call and Text Fields filling.

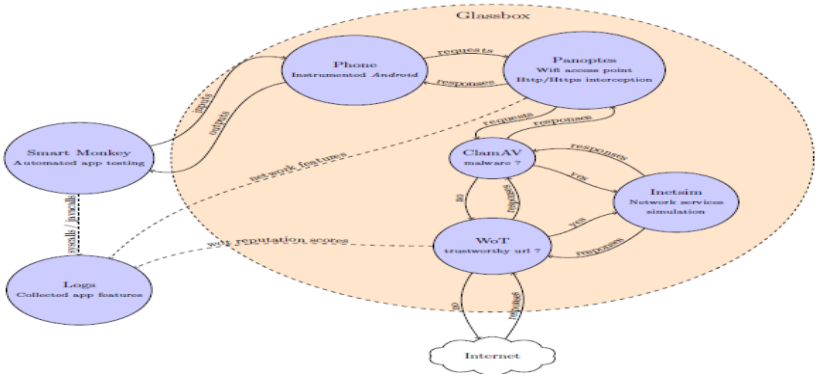

Figure 2.2:*Glass box Overall Architecture*

Since the rise of Android dynamic analysis systems, the use of system calls have been the leading approach. System calls are the functions of the kernel space, available to the user space. It gives

the capacity to manipulate hard drive files or to control processes. System calls can describe a program behaviors, from a low level perspective.

2.3 Features of An Android Application

There are common features that an applications are targeting to get an access to files and information of the system and users as depicted on Shabtai *et al.*[7]. On Table 2.1, different features of an android application has been discussed thoroughly.

Table 2.1: *Features of An Android Application*

SNo.	Features	Description
1.	Permission	The model of security in android is mainly based on permissions. A set of permission is required by application to perform its intended task. Permission are used to allow or restrict an application access to restricted APIs and resources, and granted by the users at installation or runtime. Malicious software tends to request more permission then required. Thus, 250 different permissions were identified and are used during binary vector generation. If a specific permission; e.g. SEND SMS, is requested by the app; it represented by 1 in its binary vector while it is represented by 0 if it is not requested by the app.
2.	API Calls	APIs are classes and interfaces that enable apps to interact and lunch functionality of the underlying android system. Certain API call allow access to sensitive data or resources of the smartphone and are frequently found in malware samples. Such malicious API were identified and comprehensive list of 70 such API is prepared few of them are as follows; getDeviceId(), getSubscriberId(), sendTextMessage(), Runtime.exec(), cipher.getInstance() etc.
3.	Is-crypto code	IS-CRYPTO CODE is set to 1 during binary vector generation phase if it detects cryptography related code in the apps. The encryption process of cryptography is used for obscuring information to make it unreadable without special knowledge.

4	Is-native code	IS NATIVE CODE field value is the indication of an application using native libraries. Native libraries contain native code which is compiled to binary codes and run directly on operating System. Native code allow developer to access some of processor features which are not accessible through Android SDK.
5.	Is-Reflection code	IS REFLECTION CODE value is set to True if application uses reflection to dynamically call methods. Reflection code is commonly used by programs to achieve the ability to examine or modify the runtime behavior of application running in DVM.

I. System call

The system call provides an interface to the operating system services. There are 5 different categories of system calls: process control, file manipulation, device manipulation, information maintenance, and communication. Figure 2.3 shows the working principles of system call on the system that have a kernel and user level

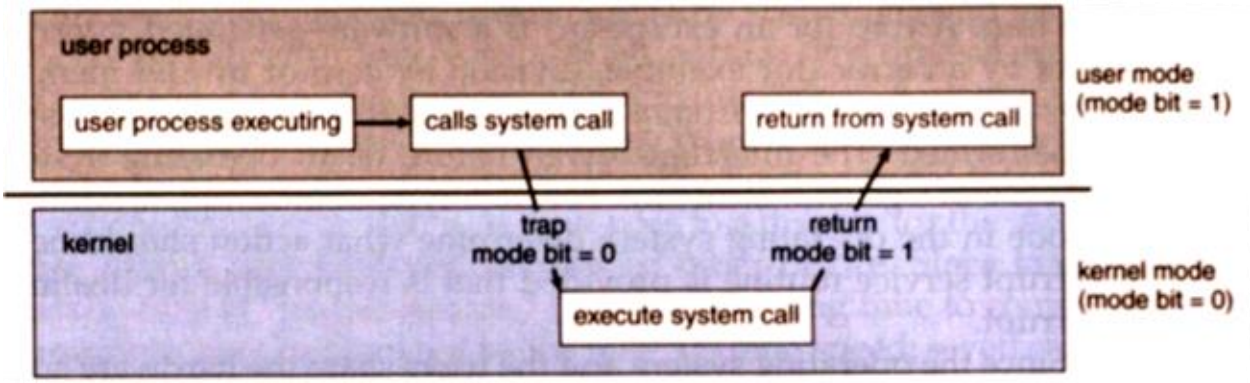

Figure 2.3:*System call*

Although both system call and other dynamic features like dynamic permission can be extracted at a time when the program is in execution, they can be extracted in a different way. Because, we have used Droidbox according to Irolla, *et al.*[17] for execution of an android applications on emulator and to extract the dynamic feature of them. But for extracting the system call, we need to have a tool which trace the execution starting with the first process of an application which send

the system call to kernel. Though, this tool is call strace as pointed out by Wenjia *et al*,[18], which is one of the module in Andropytool.

II. Permission

Google categorizes Android permissions into four threat level:

Normal permission: include lower-risk permissions which control access to API calls that are not particularly harmful. The system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval like SET ALARM.

Dangerous permission: regulate access to the potential harmful API calls that would give access to private user data. For example, permissions to read the location of a user ACCESS FINE LOCATION or WRITE CONTACTS are classified as dangerous.

Signature permission: protect access to the most dangerous privilege. The system grants the permission only if the requesting application is signed with the same certificate as the application that declared the permission.

Signature/System permission: A permission that the system grants only to applications that are in the Android system.

Permission, as one of the static feature of an android application it has a relevance in deciding whether an application has a malicious property or not. As pointed out by comparison of the top most requested permissions by malware and benign application were analyzed and presented. Figure 2.4 shows the permission frequency usage of malware and benign application.

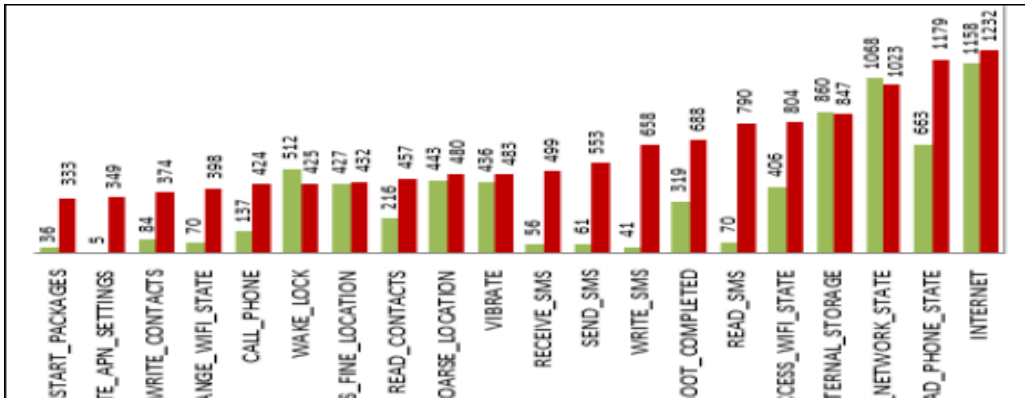

Figure 2.4*: Permission Frequency Usage*

The relevance of one of the static feature of an android application is analyzed well. This is a major input for our work, on analysis and detection of data by combining it with other features like system call and API calls.

III. API Calls

The Android platform provides a framework API that Apps can use to interact with the underlying Android system. The framework API consists of a core set of packages and classes. Because most mobile applications uses large number of APIs, It is one of the determining feature of an application for what it is performing. i.e, leaking or not.

API calls can be extracted just like permission and can be represented as a binary vector of API calls. namely A, where Ai = 1 if and only if the API is used in the application and $A_i = 0$ if corresponding application does not use the API.

2.4 Classification Algorithms

For classifying an item first we need to have select which kind of classification algorithm we are going to implement. There are different kind of classification algorithms which suits the problem like we are dealing with. ANN(Artificial Neural Network), KNN(K-Nearest Neighbor) and SVM(Support Vector Machine) are among the classification algorithms that suit supervised learning.

I. Artificial Neural Network

ANN is a type of artificial intelligence that limits some functions of the person mind. ANN has a normal tendency for storing experiential knowledge. An ANN consists of a sequence of layer; each layer consists of a set of neurons. All neurons of every layer are linked by weighted connections to all neurons on the preceding and succeeding layers. ANN is a computational model inspired by the biological neural network. It could be considered as a weighted directed graph in which nodes are neurons and edges with weights are connection among the neurons. Each artificial neuron computes a weighted sum of its input signals and generates an output, based on certain activation functions, such as piecewise linear, sigmoid, Gaussian, etc. It consists of one input layer, one output layer, and depending on the application it may or may not have hidden layers.

The number of nodes at the output layer is equal to the number of information classes, whereas the number of nodes at the input is equal to the dimensionality of each pixel. Feed forward ANN with the back propagation learning algorithm is most commonly used in ANN literature. In the learning phase, the network must learn the connection weights iteratively from a set of training samples. The network gives an output, corresponding to each input. The generated output is compared to the desired output. The error between these two is used to modify the weights of the ANN. The figure 1 shows the architecture of artificial neural network. The training procedure ends when the error becomes less than a predefined threshold. Then, all the testing data are fed into the classifier to perform the classification.

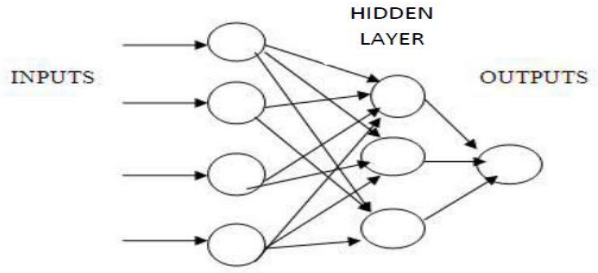

Figure 2.5*: Artificial Neural Network Structure*

II. Support Vector Machine(SVM)

A support vector machine builds a hyper plane or set of hyper planes in a high- or infinite dimensional space, used for classification. Good separation is achieved by the hyper plane that has the largest distance to the nearest training data point of any class (functional margin), generally larger the margin lower the generalization error of the classifier. Support vector machines are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis as pointed out by Wenjia, *et al.* [14]. The method is presented with a set of labeled data instances and the SVM training algorithm aims to find a hyper plane that separates the dataset into a discrete predefined number of classes in a fashion consistent with the training examples. The term optimal separation hyper plane is used to refer to the decision boundary that minimizes misclassifications, acquired in the training phase. Learning means to the iterative process of finding a classifier with optimal decision boundary to separate the training patterns (in potentially high-dimensional space) and then to separate

simulation data under the same configuration dimensions) Wei, *et al.*[27]. Support vector machine (SVM) is a learning machine that seeks the best compromise between the complexities of the model and learning ability according to the limited sample information, it is suitable for the machine learning in small samples circumstances, and overcomes the insufficient problem of typical negative type data.

In SVM classifier implementation 1V1(One-Versus-One) classical multi-class classification approach is used. This is because It evaluates all possible pairwise classifiers and thus induces k(k −1)/2 individual binary classifiers. Applying each classifier to a test example would give one vote to the winning class. A test example is labeled to the class with the most votes. The size of classifiers created by the one-versus-one approach is much larger than that of the one-versus-rest approach.

III. KNN(K-Nearest Neighbor)

The K-Nearest Neighbor is a kind of classifier which can train and test data at the same time. KNN classifier is instance based classified that performs classification of unknown instances by relating unknown to known by using distance or similarities function. It takes K nearest point and then assigns a class of majorities to the unknown instance as written by Zhang *et al.*[28]. KNN is one of the simplest of classification algorithms available for supervised learning. The idea is to search for closest match of the test data in feature space.

3. RELATED WORK

Android platform are turned out to be a potential target for malware and e-fraud[1]. Different works have been done regarding with the e-fraud which are mainly targeted smartphones. In this thesis work we have categorized the our related works into two. i.e. on application analysis and detection frameworks depending on blacklisting and parametrizing, and classification frameworks for an application analysis and detecting depending on their feature from different states.

3.1 Related works on Leakage Analysis and Detection Depending on Blacklisting and Parametrizing.

Gibler *et al*, [3] pointed out that a general framework for ensuring good privacy on different mobile platform is proposed for snooping attack on users by leakage on their mobile phone sensor such as microphone, camera and GPS receiver.

Gibler *et al.* [3] considered the leakage attack that violate users privacy by leakage on the sensor of mobile phone, i.e. Microphone, camera and GPS reviver. On this work the difference of sensor-leakage attack challenges that is posed by other malware is illustrated briefly. But, Unlike the proposed work the authors mentioned as their work doesn't give concern about the leakage attack that steal the confidential information, since the authors thought as the problem is the one which is concerned with the computer problem and can be applied to the different mobile platform.

This thesis work will mainly focus on the confidential information theft through sniffer/leaker application to fill the noticed gap in the work prepared by Hintea *et al.*[4] and also will be concerned about the sensor-leakage attacks in general.

Hintea *et al*,[4] proposed the real time on-device identification and detection of SMS sniffers and catchers, which need to be executed on device and detection method must work on a non-rooted phone, because as per the authors research, non-rooted device represent a common target and an appropriate detection methods are definitely needed for this class of device.

According to Malik *et al.*[5] consider the interception and catching of incoming and outgoing SMS(Short Message Service). The interception of the incoming and replied message will lead to theft of security information like OTP(One time password) which will be sent by different service providing organizations.

The proposed work will be able to consider the prevention and detection of SMS sniffer in addition to USSD information leaker and information stealer from any executing program on the smartphone.

Sapna Malik *et al.*[5], has proposed a system that exploits the rich information in the manifest files, produces feature vectors automatically, and uses state-of-the-art machine learning algorithms to classify applications as malicious or benign. This system is called Manilyzer.

Third party connection that can be made by an android application/mobile application can make the transmission as the app did not violate the access permission.

SMASheD is the first framework that can sniff and manipulate protected sensors on unrooted android devices without user awareness, without constant device-PC connection and without the need to infect the PC according to Egele *et al.*[6].

In SMAShed the authors hide the program which will sniff users information with another application having the service of ADB(Android Debug Bridge). This will let the program to get the permission of installation easily.

As a research gap, this work considered only information leakage applications which will stealthily sniff(read) and manipulate(write to) many of the android sensors which is working only when Internet is connected. But, the proposed work considers the existence of on device sniffer applications which will be able to sniff and manipulate anytime with no need of Internet connection.

Additionally, The proposed work is different from any of the above related works by considering the leakage activities which are able to control the root of the device and manage any other android subsystems. This means, the proposed work will be able to prevent the leakage activities targeting the rooted device.

3.2 Related works on Leakage Analysis and Detection, and Classification

Shabtai *et al* [7], introduced signature based methods in mid 90s are commonly used in malware detection. The major weakness of this type of approaches is its weakness in detecting metamorphic and unseen malware.

Instead of using predefined signatures for malware detection, data mining and machine learning techniques provide an effective way to dynamically extract malware patterns as indicated in Shultz *et al*[8]. Coherently Sami *et al*[9] has used data mining and features generated from windows

executable API calls. The authors in this work achieved good results in a very large scale dataset with about 35,000 portable executable files. Another behavioral foot printing method also provides a dynamic approach to detect self-propagating malware according to Peiravian *et al* [10].
The dynamic analysis was used to capture the system call while static analysis was used to capture permission, and all the captured and permission were classified using covering algorithm as pointed out by Sanz, B *et al* [11]. In this work, the authors uses the permission and system call to create pattern of classification on android application. Though, 32 different pattern of system call and permission were analyzed to identify GPS exploitation on android platform. As most of the permission were captured through static analysis, permission request and patterns that could be created when an application is on execution were not analyzed in this work.

Koli *et al* [12] used permission and application's metadata to classify an application as a malware or good ware using Apriori model. The mentioned model is the one which the authors built as a plugin to the existing weka models. Although the approach the authors used and the proposed model is good in classifying application, the behavioral change of an android application during execution is not considered this work.

Sahs *et al* [15] considered classification as a major technique of detecting a malware android application from benign one. Additionally, the authors did mention SVM is the best algorithm to classify an application. But, in this work even if the authors use API call as one of the dataset from which they got it from metadata online. This, doesn't totally show that the application is benign or malware, since it just depends on the static feature of an application.

Koli *et al*[12] introduced RanDroid, a system that employs various machine learning techniques ie; Support Vector Machine(SVM), Decision Tree (DT), Nave Bayes (NB) and Random Forest (RF) to perform malware classification. It makes use of comprehensive static analysis approach of application. The system uses permissions, API calls along with presence of key app's information which were not considered in most of previous proposed approaches, such as: crypto code, dynamic code, native code, reflection code, and database as a features set to generate binary vector from Android application samples of identified malware and good ware applications and adopt machine learning to perform malware classification. Although this work, consider a lots of learning

techniques and uses different native, static and dynamic codes for the purpose of classification. But, from the tool used to extract the specified data, strace or other system call extraction tool were not used. System call is one of the determining feature to classify android application as good ware or not. Most of the malicious activities can be happening during the communication between the process or components within the process.

In literature researchers extensively used machine learning techniques to model Android malwares patterns based on their static features and dynamic behavior to avoid difficulty of manually craft and update detection pattern for android malware and successfully discriminate Android malware samples form benign application. perform static and/or dynamic analysis to extract features (such as permission and API calls) from applications and employ machine learning techniques to perform malware classification.

Li *et al.* [14] presents a SVM-based approach to detect malware in android platform. It is based on static analysis and uses risky permission combination and vulnerable API calls as feature to train SVM algorithm. Drebin proposed a static analysis based framework that extracts a set of features from the apps AndroidManifest.xml and disassembled code to generate features vector.

SVM was applied on the dataset to learn a separation between the two-classes of apps (benign and malicious). The system was tested with 123,453 benign and 5,560 malwares. MobieSandBox system is based on combination of static and dynamic analysis where results obtains from static analysis are used to guides dynamic analysis and extend execution code coverage. It also uses techniques to log native calls and is successful to claim that 24% of all applications in Asian third party market uses native call in their code.

Droidmat is also based on static analysis detects malware through analyzing AndroidManifest.xml and tracing systems calls. It first extracts different features from the apps android-Manifest.xml such as: permissions, and intent messages. Then, it marks the apps components; activity, service, and receiver as initial points to trace the API calls that are related to the permissions. DroidMat uses permissions, components, intents, and usage of the API calls as feature set and applies K-means algorithm to model malware while the number of clusters are determined by singular value decomposition (SVD).

According to Chen *et al* [1] the genuine applications are downloaded from google store and some malware application data were downloaded from standard dataset. Here, the authors are only evaluating the model they built to classify as malware or benign depending the extracted feature. i.e. permission and API calls. The authors already knew the application whether it is malware or not, but they want to know whether the classifier classify it correctly or not. In our concern, if the user install an application just by trusting the source and the application is unidentifiable by its signature, there should be a mechanism to classify even benign application depending on their behavior on execution.

3.3 Summary

The related works that were reviewed in preparing the thesis report and working on the frameworks have similarities with our work. As an example we can take DroidMat which is one of the related work, but the big difference among our work and DroidMat is It uses features of an application at their static state. Which means the extraction of the feature can be done using static analysis specifically on the manifest file of an application.

Using machine learning is one of the major relativity among our works and the reviewed works in this section. Some of the reviewed works uses SVM, which is pointed out as it is preferable one in order to perform classification and build a classification model.

Our thesis work is different from the related works discussed, as it considers the feature of an Application at different states i.e. pre-static, static and dynamic states. Additionally, our framework considers the frequency and behavior of system calls that an application send to kernel during execution. Signature and pattern based detection and prevention mechanisms mostly considers the attribute of an application before execution, which is not that enough for judging an application or classifying it. But, the tools, algorithms and methods used by most of the related work was impressive and considered as the best one by us.

Generally, most of the preferred journals and articles that are reviewed as a related work to ours have considered the big problem the security of a smartphone has. The authors on the reviewed work has also pointed out the solution like using classification model for identifying an application and putting a flag on them to show their behavior and detecting them. But most of them were limited to analyze the features of an application to the static level features which

will let an application to perform leakage by disguising the good-ware behavior during their static state and change their behavior later on during execution.

The proposed framework is capable of detecting the disguising applications using the combination of features of an applications at different states. Besides using the features from different states, our framework also recommends the best classification algorithm to use after performing an experimentation which involves comparative analysis among three different classification algorithms to recommend the one with best performance during experimentation.

4. APPLICATION CLASSIFICATION FRAMEWORK FOR INFORMATION LEAKAGE DETECTION

In this chapter the framework design for application classification depending on the collected dataset at different state of an application will be discussed thoroughly. Some of the selected features, which are the components of the dataset having the determining capability in order to classify an application as a leaker or not will be described.

4.1 Overview

Android application classification is one of the different leakage detection and prevention mechanisms through which applications can be classified as a leaker or not depending on different attributes they got at varied states. Signature based malware detection focuses on the patterns of an application to categorize an application as a malware or not. Mostly the dataset collection for signature based detection from the applications manifest file which is the static feature of an application. The static feature of an application might be better to detect privacy leakage early.

The need of having another detection mechanism and proposing this work is to detect an application which have ability of not getting detected at their static state, but shows the property of privacy leakage during the execution phase. So, we believed that there should be new way of combining the feature of an application at different state in order to classify an application as a leaker or not.

Figure 4.1, Shows an entire framework for classifying new application into leaker or not depending on analyzed features extracted before performing the classification. Each component of the framework will be discussed thoroughly and one by one in the coming sub-sections of this section.

Unlike an experimentation process used to form a classification model(one of our framework component that contains the training model), on our framework we can give a random application which as an input , we didn't have a flag for it which shows whether it is a leaker or not. So the main classifier component of our framework uses the classification model for reference in order to classify an application or give a flag of showing if it is a leaker or not.

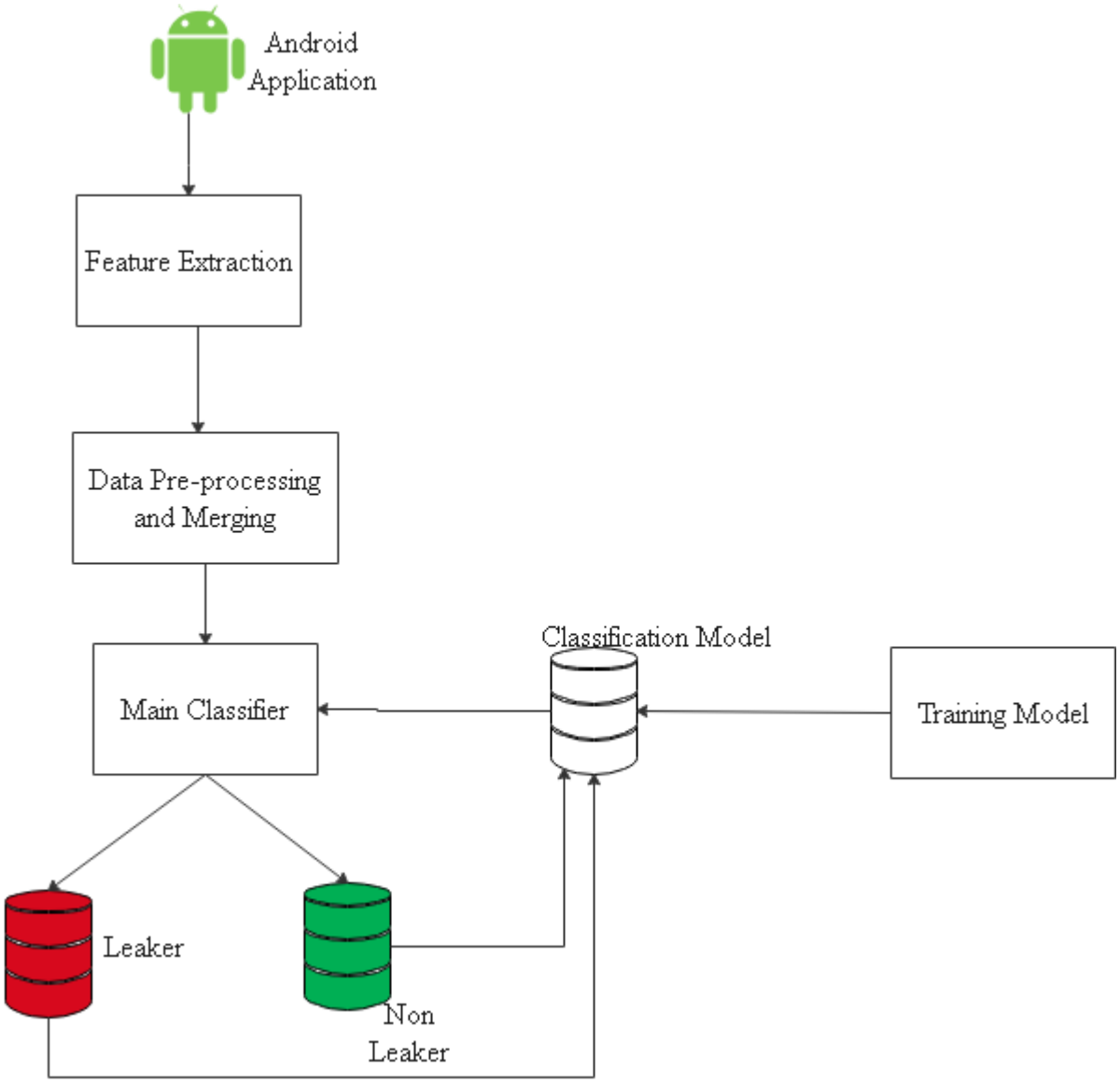

Figure 4.1: *Proposed Framework*

In Figure 4.1, the major components of the framework are shown as per the data flow among them. The first activity, which is taking an input will take a random application. The random application shouldn't necessarily be known by the system although he have thousands of applications used for training one of the components which is training model component. Using the feature extraction processes which is shown on the next sub-sections the feature of an application at different states are collected at the feature extraction component of our framework.

An extracted feature can be pre-processed at the preprocessing component of the framework which leads to the identification of irrelevant data and prepare the dataset for three different states with related attributes. Later on the preprocessed data will be merged to create the dataset for a specific application and feed it into the main classifier which depends on the already trained classification model.

As we used an incremental supervised learning to train the classification model we can increment the dataset on which the classification model depends on. So after the main classifier classifies an application into a leaker or not , the dataset for a specific application containing a flag that shows whether an application is a leaker or not will be added to the main dataset to increment the accuracy of learning in a training model.

4.2 Feature Extraction

In designing the structure of the proposed classification framework for categorizing an application whether they are leaker or not is done using machine learning. Using machine learning classification algorithms is preferable in order to make the framework grow its performance from time to time as the learning process continues. Among different learning model we preferred to use incremental supervised learning approach .

The framework involves the feature extraction, pre-processing of the extracted data and training the machine in order to make decision whether an application has a leaking property or not. In order to create dataset for feeding classifier, we have implemented extraction algorithm for static feature analysis process and used an extraction tool with proper algorithm for the rest i.e. Dynamic analysis and System call extraction. Figure 4.2 Shows the detailed feature extraction component of our framework.

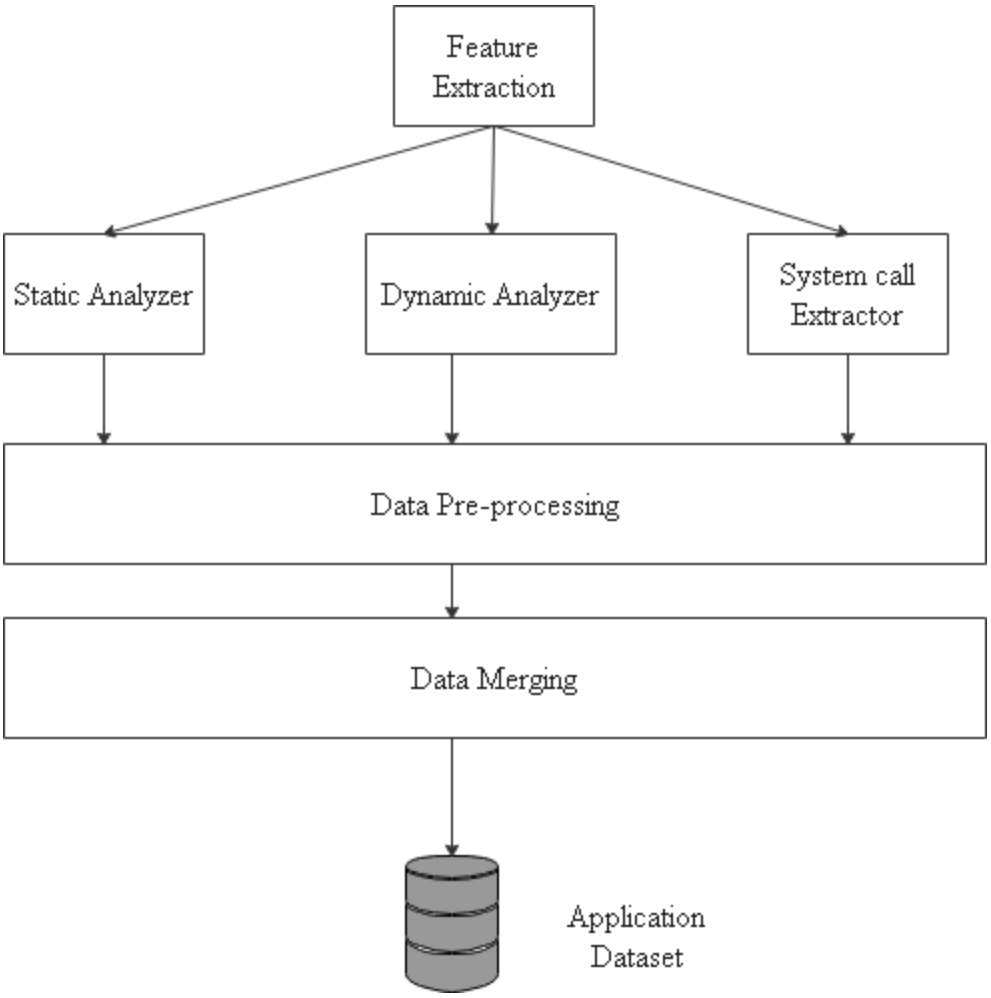

Figure 4.2: *Feature Extraction*

Feature extraction component can be from three different states. Before execution, during execution and from the system calls that can be sent to the kernel part of the system.

The extracted features can be organized and used for classifying an application either as a leaker or not after getting pre-processed and data from different sources are merged together. At the end of the feature extraction process our framework has an application data set which will be used by main classifier.

4.2.1 Static Feature Analysis

Static feature extraction in this thesis is presented depending on the feature extraction of FlowDroid which in turn depends on IFDS framework, for tracking data flow. IFDS stands for Inter-procedural, Finite Distributive subset which is used for static data flow analysis. According

to flowdroid is a static data flow analysis tool for Android apps and Java programs. When we go deep into the structure of static data flow analysis for extracting the static feature of an android application we used aliases of different flowdroid algorithms.

Among the different alias strategy that has been used by FlowDroid, in this thesis we have used the PtS based aliasing analysis algorithm. This is because the number of queries to be made can be high depending on the maximum access path length during analysis. Algorithm 4.1 shows pts

algorithm which is used to extract features in flowdroid application. This algorithm is preferred among four different algorithms used in flowdroid based on the suitability it has for extracting features with malicious property at the static state of an application.

Algorithm 4.1: *PtS-Based Alias Algorithm in Flowdroid*

```
Require: Method body b, tainted access path ap
Ensure: Injection of alias taints into forward taint tracker
1: Q   [ ap ]{Initial queue of aliasing access paths}
2: while Q 6= empty do
3: curAPQ.pop()
4: for each Unit u 2 b do
5: if isInstanceInvokeStatement(u) then
6: if curAP.hasPrefix(u.base) then
7: injectIntoForwardSolver(append(u.base, curAP.suffix), u)
8: else if isInvokeStatement(u) then
9: for each Parameter p 2 u.getParams() do
10: if curAP.hasPrefix(p) then
11: injectIntoForwardSolver(append(p, curAP.suffix), u)
12: else if isAssignment(u) then
13: if isHeapObject(u.rightOp) ^
hasNonEmptyIntersetion(pts(u.rightOp), pts(curAP)) then
14: Q  append(u.getLeftOp, curAP.suffix)
15: if isHeapObject(u.leftOp ^
hasNonEmptyIntersetion(pts(u.leftOp), pts(curAP)) then
16: injectIntoForwardSolver(append(u.getLeftOp, curAP.suffix), u)
```

As depicted on Algorithm 4.1, the PtS-based alias analysis only iterates over the statements inside the current method to check for potential aliases.

Algorithm 1 shows the PtS-based aliasing algorithm in detail. It is called for a method body b and an incoming access path ap. This incoming access path is the one that was originally tainted though the assignment to the heap object for which the alias analysis was invoked. The algorithm takes

this access path as the first element of its worklist Q in line 1. As long as the worklist is not empty, the algorithm iterates over all statements in the current method. We will first discuss how assignments are handled (line 12). If the current statement is of the form a = b.c and the current access path curAP is e.f.g, the algorithm must check whether the points-to sets of b.c (the assignment's right side) and e.f (the prefix of the current access path) have a non-empty intersection as shown in line 13. If this is the case, a new alias has been found which must be added to the taint set. In line 14, the access path of the newly-found alias is computed by taking the right side of the assignment (b.c in the example) and appending the remaining fields of the access path that are not covered by the field access (g in the example). In the example, the new alias b.c.g is added to the worklist. It is not immediately passed to the forward taint analysis, though. Aliases are only manifested when they are actually accessed in this alias algorithm.

The check whether an alias is accessed is done in line 15. If the left side of the assignment aliases with the prefix of the current access path (based on a non-empty intersection of the respective points-to sets as explained above), the forward taint analysis is triggered. Similar to the case before in which a new alias was discovered, the algorithm adds the remaining fields to the access path. It then injects this access path into the forward taint propagation as a new taint.The second possibility for referencing a tainted access path aside from assignments is to use the base object of a tainted access path in a method invocation, be it as the base object of a virtual method call (line 5 in the algorithm) of as a parameter (line 8 in the algorithm). In both cases, the alias algorithm derives a new access path with the respective base object from the method call and the field list from the incoming access path which it passes to the forward taint analysis (lines 7 and 11). The forward taint analysis then takes care of mapping these access paths into the context of the callee where the alias analysis is triggered again if necessary (i.e., when the taint analysis has reached another assignment to a heap object in the callee).

4.2.2 Dynamic Feature Analysis

Dynamic feature extraction in this thesis is intended in extracting the dynamic feature of an application which is the feature of an application on execution. The feature of an application on execution is one of the determining factor for detecting an application showing leakage property just after passing the static analysis.

Dynamic analysis is performing by tracking the taints. In this thesis the dynamic feature extraction can be done using droidbox, which depends on the taintdroid framework. Taintdroid has the feature of tracking information flow in a real time according to Erick, *et al.* [18].

The DroidBox Analysis was developed to detect applications that leak sensitive information, use services that cost money, or use suspicious functions as dexClassLoader . The dexClassLoader is often used by malware to dynamically load a malicious APK that was previously downloaded via the network. But in dynamic feature extraction the main component of the Analysis is the DroidBox Plugin, which uses the DroidBox system image.

In our thesis work DroidBox Plugin is to integrate the existing Android application sandbox tool named DroidBox into the framework and to extend its functionality. After performing evaluation on DroidBox during experimentation phase of the thesis work, we preferred to use Droidbox Plugin for performing dynamic analysis which will provide us with features of an application after executing an application on emulator.

4.2.3 System Call Analyzer

Besides the dynamic features of an application, system call is another way of detecting an application's leakage behavior. In order to capture the system calls that an application send to the kernel level of the system in this thesis we implemented the strace.

Strace has plenty of functionalities like debugging programs, troubleshooting programs, Intercept System calls by a process, record system calls by a process, signals received by a process and trace running processes. Among the functionalities of strace we specifically incorporated process system call recording, signal's received by process and tracing running processes.

In this framework, an information from the output of the strace execution which is system call is one of the unique property. We believe that system call is a huge data component to classify an application as a leaker or not in collaboration with dynamic and static data.

4.3 Data Pre-processing and Merging

4.3.1 Data Pre-processing

Data pre-processing involves removing outliers and organizing a relevant data from an extracted features. In our framework development, we incorporated feature selection algorithm for selecting

relevant features of an applications. As the features of an application has strong relation with the output result, we preferred to use univariate selection technique. So before performing data merging, our framework performs the selection of relevant features for three different states of an application.

Univariate selection technique for pre-processing the features of applications, using python based scikit-learn library which provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features.

The use of univariate selection technique is to reduce the outfitting of the data which means reducing the redundancy to make the opportunity of making decision based on irrelevant data. On the other hand using univariate selection technique enhances the shortage of training time and also improves the accuracy of classification. **Algorithm 4.2** shows the python based implementation algorithm for univariate statistical test which is based on the dataset we have extracted in our classification model experimentation.

Algorithm 4.2: *Feature Selection using Univariate Statistical Test*

```
1) # Feature Selection with Univariate Statistical Tests
2) from pandas import read_csv
3) from numpy import set_printoptions
4) from sklearn.feature_selection import SelectKBest
5) from sklearn.feature_selection import f_classif
6) # load data
7) filename = 'ApplicationclassificationDataset.csv'
8) names = [400 features of 3160 applications]
9) dataframe = read_csv(filename, names=names)
10)     array = dataframe.values
11)     X = array[:,173]
12)     Y = array[:,3160]
13)     # feature extraction
14)     test = SelectKBest(score_func=f_classif, k=4)
15)     fit = test.fit(X, Y)
16)     # summarize scores
17)     set_printoptions(precision=3)
18)     print(fit.scores_)
19)     features = fit.transform(X)
20)     # summarize selected features
21)     print(features[0:5,:])
```

On an **Algorithm 4.2** the scikit-learn library provides the SelectKBest class (on Line 3)that can be used with a suite of different statistical tests to select a specific number of features. On (line 7) we have the amount of data we obtain during extraction. For 3160 applications within limited time frame we obtained about 400 attributes for each, which was saved as a dataset so we feed this dataset (on line 7).

Line 14 of the Algorithm shows that we have implemented the test which uses the selectKBest function to reduce the redundant feature and fit them into the value of the array in X (on line 11) and Y (on line 12)

So using this algorithm our classification model and our entire framework performed the feature selection from the extracted feature.

4.3.2 Data Merging

After performing data pre-processing for an extracted features from three different states of an application, merging of those features are performed. As similar number of columns are available after performing data pre-processing, merge join technique are used in this dataset preparation mechanism. So features from three different states were combined together and will be feed to the main classifier.

Merge Join technique which is used in our dataset preparation process joins two or more database to create the merged version of them. The merge-join algorithm involves two different steps. The first step is to sort the joining tables by joining attributes which is shown on an **Algorithm 4.3**. The joining attributes are working for the tables from different sources. An **Algorithm 4.3** shows an implementation of the first step which is preparing the joining tables according to the joining attributes.

Algorithm 4.3: *Comparison among datasets from different sources*

```
1) posts.sort(Comparator.comparing(Post::getId));
2) postComments.sort((Ds1, Ds2,Ds3)
3)  {
4)  int result = Comparator
5) .comparing(PostComment::getPostId)
6) .compare(Ds1, Ds2,Ds3);
7) return result != 0 ? result : Comparator
8) .comparing(PostComment::getId)
9) .compare(Ds1, Ds2, Ds3);
10)     });
```

As shown on **Algorithm 4.3** After getting the ID of each columns sorting will be performed as depicted on line 2. After sorting on a similar way our algorithm perform the comparison between the three different tables. So the aim of this step is to perform the comparison among the three tables.

The second step in merging data from different source is to iterate within the three tables to check if they could fulfill the joining condition. The **Algorithm 4.4** shows the iteration among the tuples of the table from different sources.

Algorithm 4.4: *Joining tables according to the Joining condition*

```
1) List<Tuple> tuples = new ArrayList<>();
2) int postCount = posts.size(), postCommentCount =
   postComments.size();
3) int i = 0, j = 0, k=0;
4) while(i < postCount && j < postCommentCount && k<
postcomment2) {
5) Post post = posts.get(i);
6) PostComment postComment = postComments.get(j);
7) PostComment postComment2= postComments.get(k);
8) if(post.getId().equals(postComment.getPostId())) {
9) tuples.add(
10) .new Tuple()
11) .add("post_id", postComment.getPostId())
12) .add("post_title", post.getTitle())
13) .add("review", postComment.getReview())
14) );
15) j++;
16) } else {
17) i++;
18) k++;
19) }
20) }
```

An **Algorithm 4.4** Shows an iteration within three different tables which are indexed as i, j, k and creating the similarities among them depending on the joining condition.

In creating the classification framework which depends on the classification model which is created using experimentation we have used merge joining dataset preparation as our datasets are from three different sources for achieving the accurate classifying using features of an android application at three different states.

4.4 Main Classifier

The main classifier is one of the core operating component of our framework. Main classifier performs the major activity of our classification framework As shown on Figure 4.3, the main classifier uses the classification model which we created using our own data set and uses specific classification algorithm after performing comparative analysis among three different classification algorithms. Figure 4.3 shows the main classifier task and it's dependency on the classification model.

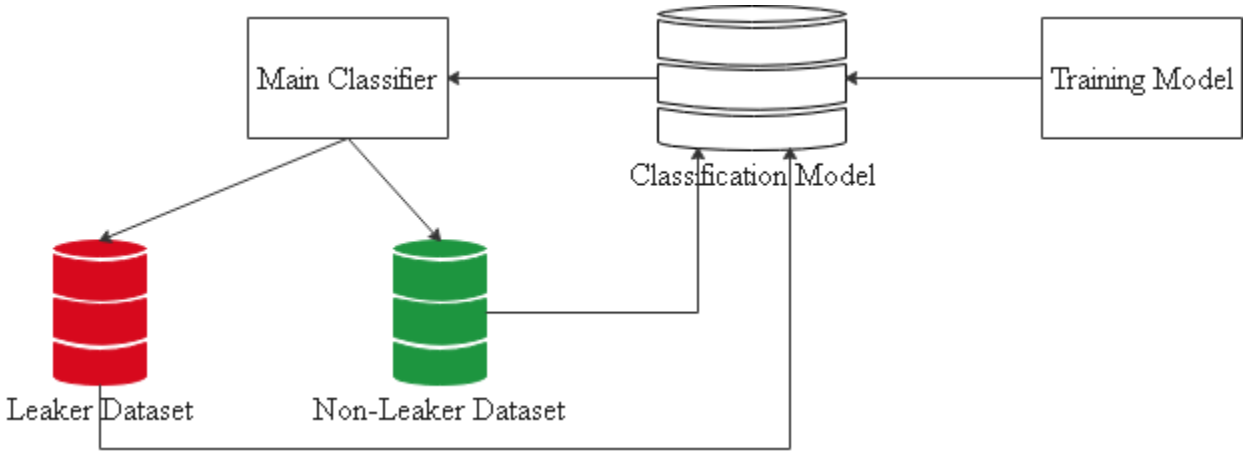

Figure 4.3: *Main Classifier of the framework*

Figure 4.3 shows that the decision of classification of newly inputted application as a leaker or not is done depending on the already created and trained classification model. The trained classification model on Figure 4.3 contains the data set and recommended classification algorithm which helps the framework classify an application correctly depending on the feature it shows at different state. An application classification model was put together after performing an experiment.

After performing an experimentation the incremental supervised learning is preferred because, Adding an applications data into dataset after being classified as leaker or not increases the number of applications data in a dataset, which will in turn increase the accuracy of classification algorithm. So our framework is self-developing, which increases the classification accuracy as more application is put to be classified.

4.4.1 Classification Model

Classification model is a major component of android application classification framework. An entire framework depends on the experimentally proved principles of the classification model. We trained the classification model which is used in this framework.

In order to train and apply learning we have used an experimentation for extracting features of an application and prepare our own dataset. As we already flagged an applications dataset as a leaker or not leaker in our dataset we have trained our model using three different classification algorithms and preferred one among the rest through comparative analysis.

The classification model has a training component and a dataset. Incremental supervised learning is used to update and add the data into the dataset as the classification is performed by our framework. The two components of the classification model are dataset and training model

A. Dataset

In a research that involves machine learning, we need a training data set. It is the actual data set used to train the model for performing various actions. In context of this work, the dataset is used to train the model to classify applications as leaker/malware and non-leaker/benign.

Depending on the data from dataset the input will be given to the classifier model and the classifier model will tell whether an application is leaker or not after analyzing and learning the features from the dataset. In our case features of more than three thousands of an application was extracted at three different states using static feature extraction algorithm, droidbox and strace tools.

The data from system call extraction, static analyzer and dynamic analyzer of an android application combined together to create the pattern for classifying an application as leaker or not.

In order to get the most determining static property of an application, one of the AndroPyTool component, i.e Flowdroid is used to extract the static property of an application specifically, permissions from manifest file. Then, we used droidbox and strace to extract the dynamic feature of an application.

System call extraction is done by installing genymotion i.e virtual environment for android application. On this virtual environment nex 5 android OS was installed. For an application to send

their system call Monkeytool was used as a built in component of genymotion. As we execute an application on nex 5 android OS, monkeytool is used to execute different application, so strace can extract the each system call from each execution.

Although system call is one of the feature which is more determinant for classifying application as a leaker or not, there are other features which are also determining the property of an application . so, in this work we used droidbox to extract the rest of dynamic feature of an application.

We believe that those features other than system call is necessary to identify whether an application has property or not. Because, if we see permission which is mainly static feature of an application could be extracted in dynamic analysis since there are another category of it which is native permission.

In order to classify an extracted feature and train our model, SVM, ANN and KNN are used in this work. So 70% of data in dataset is used for training purpose, and 15% for evaluation and 15% for testing; the model will learn to easily classify features depending on the most significant features. An application having leakage property were not always be detected by anti-malware or any other malware detection signature based detector. Because, unlike any other malware an application with leakage property may not show any malicious property, they rather show the property of normally functioning application. Sometimes, they perform their activity when they are being executed for performing the functionality they are intended to perform.

B. Training Model

The training model of our application classification framework is experimented with an application from different sources having the property of leaker or not. **Figure 4.4** Shows the experimental process of developing the model which will be more elaborated on an experimentation part of this document i.e. Section 5. The training model uses the dataset prepared for training the model which contains 3160 applications of which 107 of them have the behavior of leaking information from users smart phone. So we have used experimentation to prove if our training model is able to classify those dataset into leaker or not for 3160 applications.

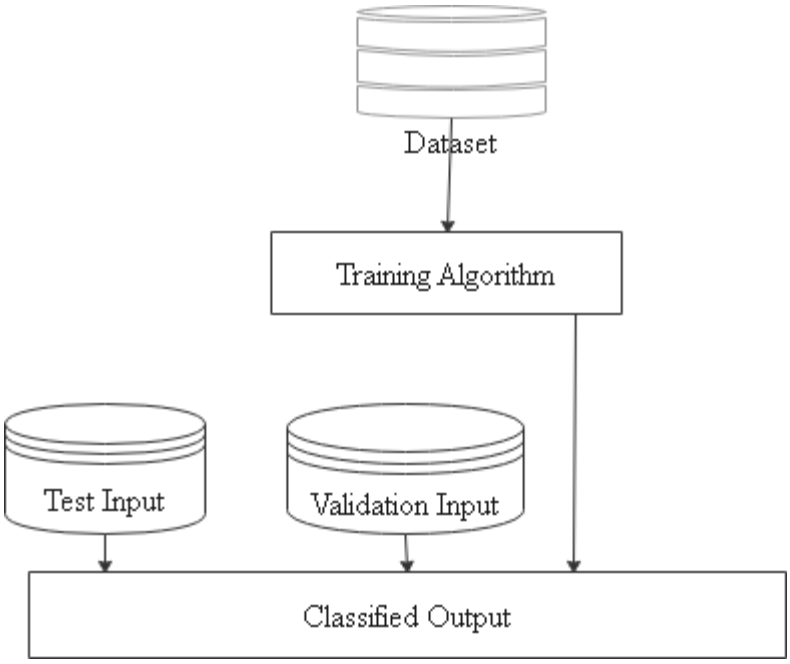

Figure 4.4*: Classification Model Experimentation Process*

In Figure 4.4, the step by steps experimentation process are shown. Dataset identification were done by specifying the extraction source of the dataset for training our model, while feature set selection process is for preprocessing and categorizing the obtained data as a training set, testing set and validation set for which it will be provided to the training algorithm. In the experimentation process comparative analysis among three different training algorithm were performed i.e. SVM, ANN and KNN. Then the output vectors were set and we recommended the right classification algorithm for our framework depending on the comparative analysis among the algorithms and the entire performance of our model.

As shown in Figure 4.4, The data set which is extracted from different sources, pre-processed and merged together is used to train our classification model which is the main input of our entire framework. During the experimentation process for creating the classification model, we have used 70% of our dataset for training our model, 15% of them for testing and 15% of them for validation purpose.

The component which is called training model in our main framework is built through experimentation performed on 3160 android applications and features we extracted to prepare our own dataset. The entire framework experimentation and an experimentation for the specific

component of our entire framework which is Training model is also discussed in depth on the experimentation section of this thesis.

An experimentation process for creating the classification model is performed totally on a virtual environment. As the classification accuracy of an entire framework depends on the classification performance the model which is major component of our framework has, experimenting our entire classification framework totally depends on our classification model.

Experimentation for classification model involves three different sets of data, i.e. Training Dataset, testing dataset and validation dataset. As we have used incremental supervised learning for training our system to learn the patterns of an application we already know that out of 3160 applications 107 of them have a leakage property. After performing an experiment we have achieved classification accuracy of 99.8% using artificial neural network training algorithm. Among the 3160 applications the model classified 3048 applications as non-leaker out of 3053 and 105 of them as a leaker out of 107. This means the training model is proved to have highest accuracy of classifying an application depending on their extracted feature which will be even increased after adding newly classified application's dataset to the main classification model as we are using incremental supervised learning.

5. EXPERIMENTATION

An experimentation on classification framework of an android application whether they have leakage property or not, depending on their dynamic, static and pre-static feature that can be extracted is covered in this section of the thesis. An experimentation process involves extraction of features from specific application for which we didn't knew whether it has a leakage property or not.

After performing an experimentation on creating a classification framework and a classification model which is one component of our framework, performance evaluation of an entire framework was done by extracting the feature of new application and classifying it into one of the two categories, then adding them into our dataset.

5.1 Feature Extraction and Dataset preparation

As two categories of anomaly detection mechanisms were used in this classification model, we extracted different features of an android application on execution and at application level.

3160 different applications from which 3053 are an application which are considered as an application not showing the property and downloaded from google play store and 107 applications showing property and downloaded from drebin site.

For each application there are about 400 features, that can be extracted within 30 second of execution on a virtual device. From those features only 173 of them were selected depending on their priority of determining whether an application has property or not.

173 features for each application is preferred to be used, as the rest of the features are either reputation of the existing feature while the monkey tool enables the virtual device to use the same system call repeatedly and the concern of this work not classifying the applications depending on the reputation of the system call used rather it is using the combination of the features at execution and before execution.

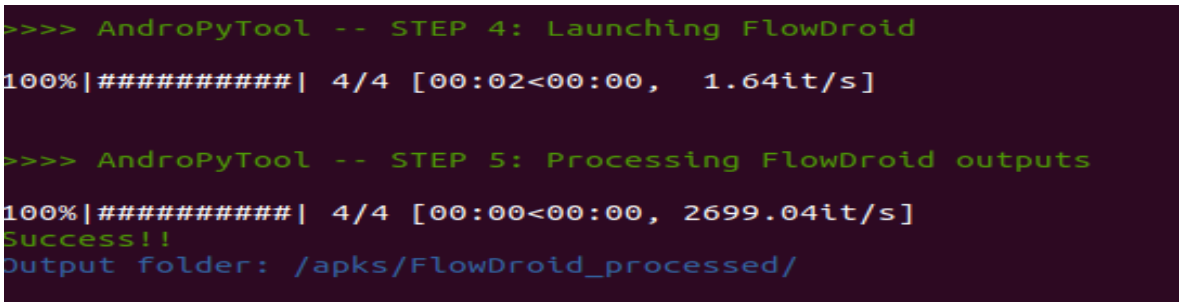

Figure 5.1: *Static and pre static feature extraction*

Figure 5.1 shows the static and pre static feature extraction using flowdroid which is one of the main component of AndropyTool. The Figure shows the feature extraction for 4 application from google play store. The feature extraction of 4 application is just for the sample.

Before performing the merging of dataset we also performed the extraction of features of an application on execution(on virtual device). Nexus 5 is used smartphone on a virtual device. The device is preferred as some of the application from google play store cannot be executed since they are not been programmed to be executed on the older version and some of them cannot be executed on latest version of an android device. So nexus 5 is preferred as it is better in mediating both problems

Figure 5.2, shows a number of applications on execution on our virtual environment. In order to get system call feature of an application we need to get the process ID of an application on execution. So we need to get the list of an application on execution..

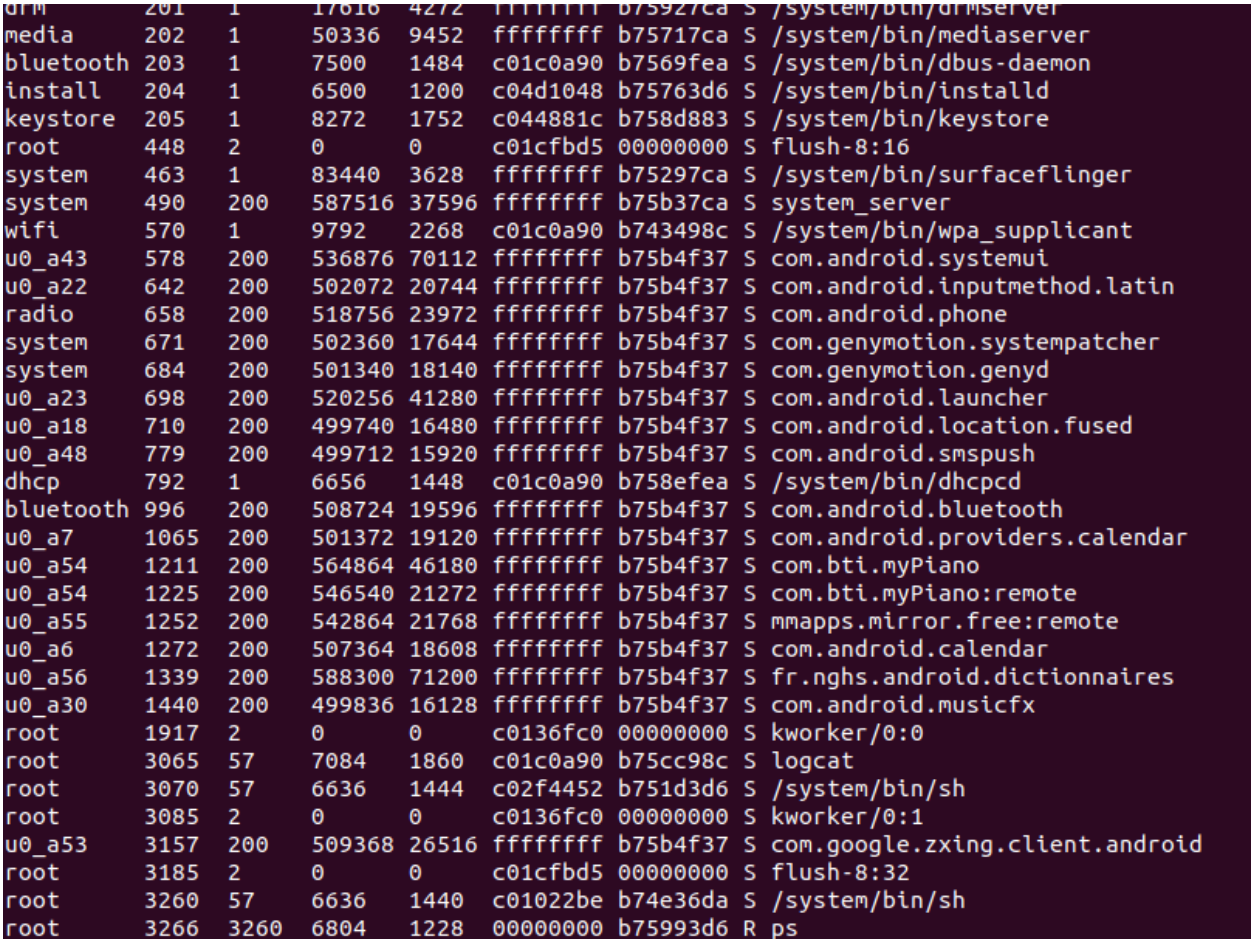

Figure 5.2: *List of an application on execution*

Figure 5.2 Shows an application in execution on an android virtual device. In order to extract the feature of an application on execution we used andropytool specially strace for extraction system call and dynamic permission. Strace is one of the component of andropytool which is used to extract system call and Droid box is also another component of andropytool which is used to extract other dynamic features of an applications

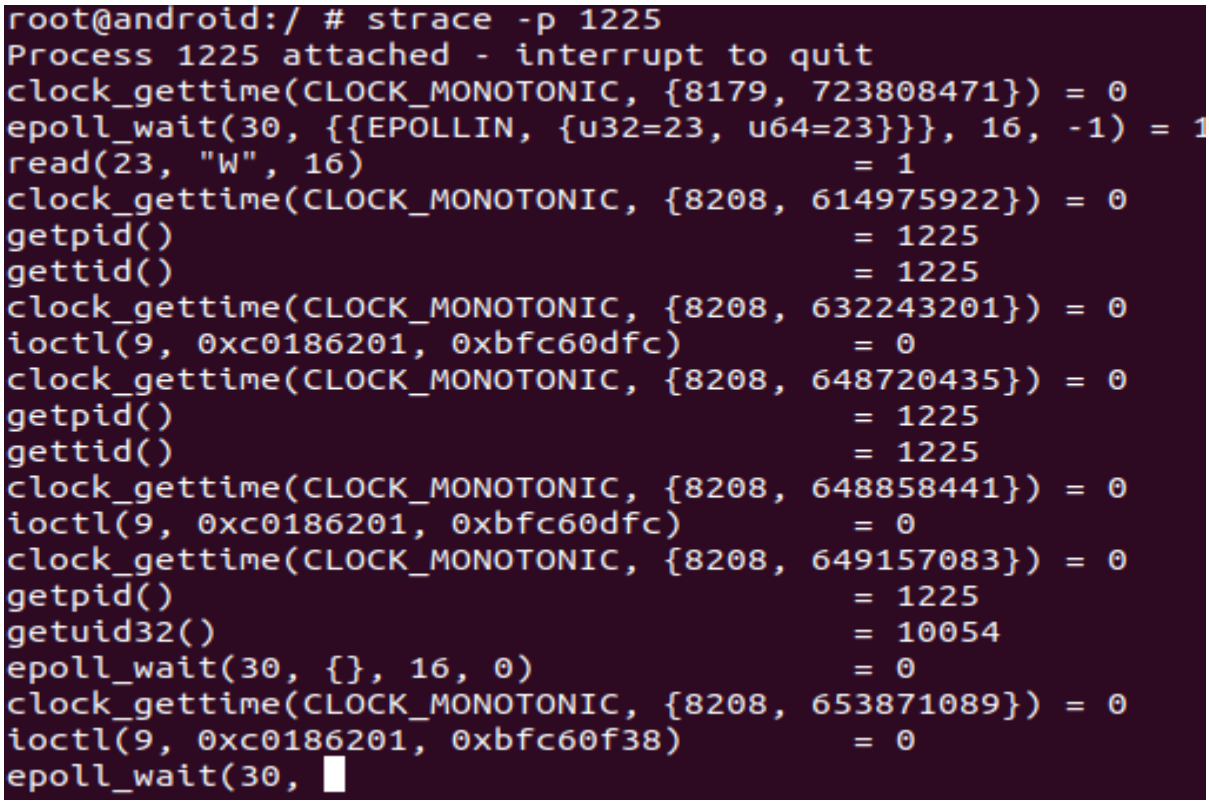

Figure 5.3: *System Call Extraction*

System call is one of the determining feature of an application, since it is the way of creating other processes to perform an activity. Dataset preparation tools that were used in feature extraction are also capable of preprocessing the data after extraction. This kind of preprocessing involves, creating two class type of data depending on whether an application uses the specified feature or not.

5.2 System Setup and Configuration

Matlab 2018a were used for implementation of the proposed model. This tool involves plenty of classification learner algorithms which could be used on different situations and for different learning categories(supervised and unsupervised). In this work we used, mainly the supervised learning classifier algorithms i.e. SVM and ANN, and evaluated both for the efficiency they have.

Depending on the dataset storage and processing speed required during the feature extraction we preferred to use the system with the memory capacity of 8GB, 1TB hard disk, with core i7 processor technology.

5.3 Evaluation Metrics

To evaluate the effectiveness of proposed model, confusion matrix shown in Table 5.1 is used which provide summary of prediction results on a classification problems.

Table 5.1: *Summary of Prediction Results*

	Predicted as Malicious	**Predicted as Benign**
Actual malicious	True Positive	False Negative
Actual Benign	False Positive	True Negative

Recall rate is the rate of correctly sensing an instance as malicious whereas false positive rate(FPR) is defined as the false identification of benign application as malicious. The Accuracy value define how precise the classifier classifies the instance in the right class while the F-measure is the harmonic mean of Recall and Precision.

Accuracy = TP+TN/TP+TN+FP+FN

Recall rate=TP/TP+FN

false positive rate=FP/TN+FP

Precision=TP/TP+FP

F-measure=2*Recall*Precision/(Recall + Precision)

5.4 Results

The system model we introduced was evaluated in two ways with the previously existing classification models. One is the dataset that was fed to the model. In this work we used the system calls and other dynamic feature of the application. On the other way, we evaluated the classification accuracy of the model, by using two different supervised learning classification algorithms. i.e. we used Artificial neural network and Support vector machine.

We provided the input dataset extracted, and the target output data expected for our model to learn classification. Out of the input data, i.e. 3160 for both malicious and benign, 15% of them which is 474 were used for validation and 15% of them were used for testing. 70% of the inputted dataset were used to train our model.

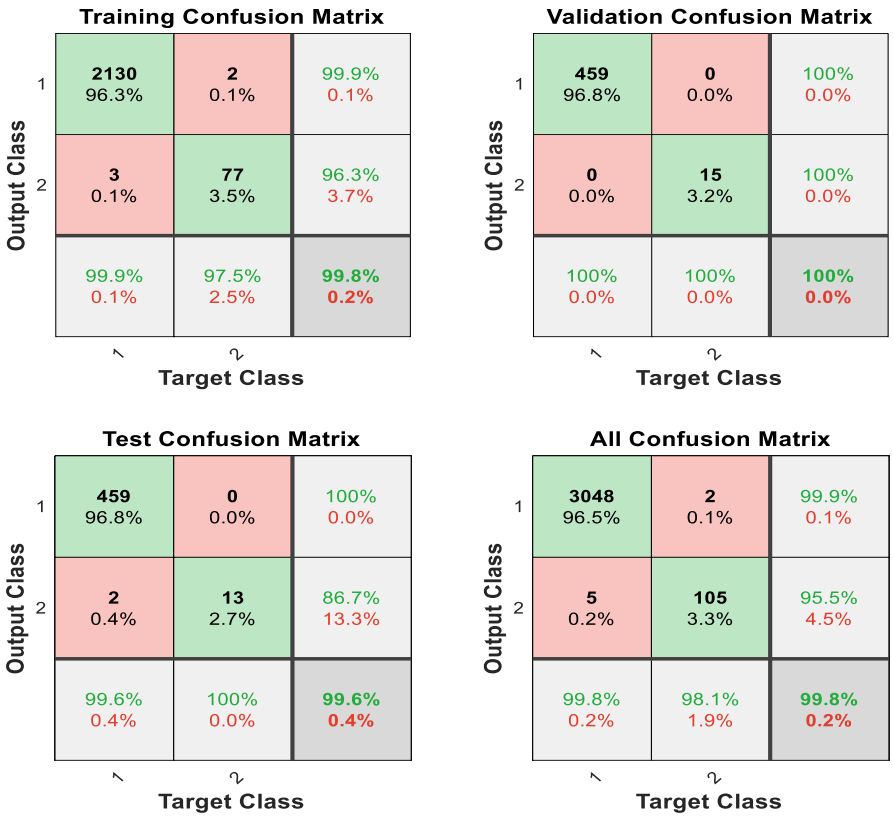

Figure 5.4:*Confusion Matrix Using ANN*

From figure 5.4, we concluded that the proposed model is trained well and can classify an application depending on their static, pre-static and dynamic feature with an accuracy of 96.5%. So we can say that an our model is well trained for classifying an application. The accuracy value mentioned i.e. 96.5% is the True positive value of the model which refers to the accurately classified feature of 3048 application as a non-leaker. Among the training dataset feed to the model which we already know as they are non-leaker, our model classified 5 of them as they still have a leaking property. On the other hand, the True-Negative value of the model is about 3.3% of which 105 of a 107 applications which we actually know they have a leaker property were classified as they have leaking property , while 0.1% of the entire application are classified as they didn't have a leaking property. So the model has 99.8 % classification accuracy after being trained.

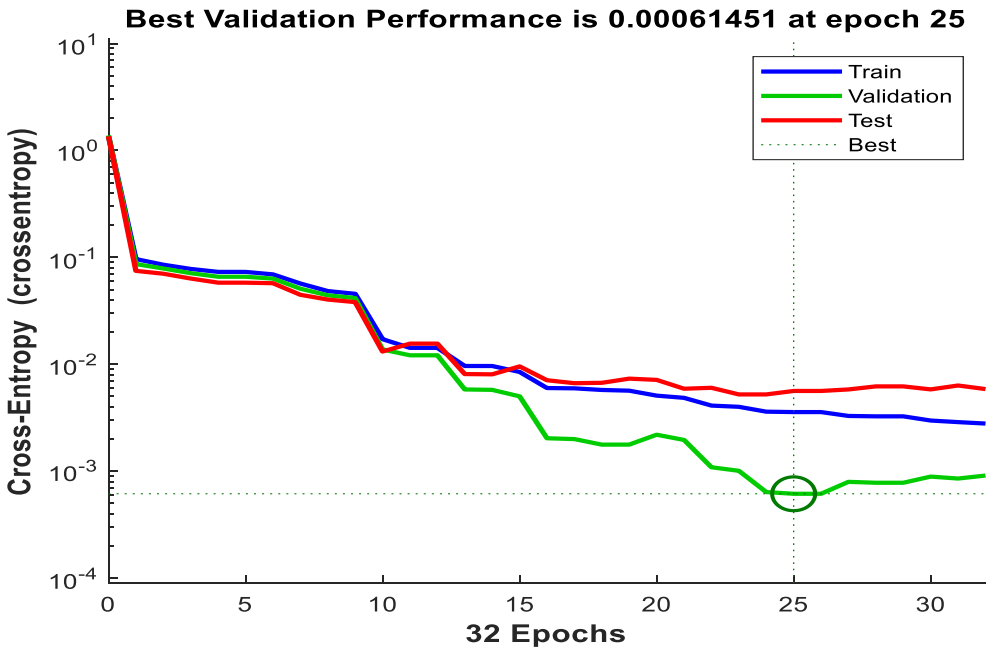

Figure 5.5: *Neural Network Performance*

Figure 5.5, Is the best validation performance the proposed model achieved using neural network. In this experimentation process we allowed our training process to repeat for 32 epochs for which we achieved the best validation performance on 25th.

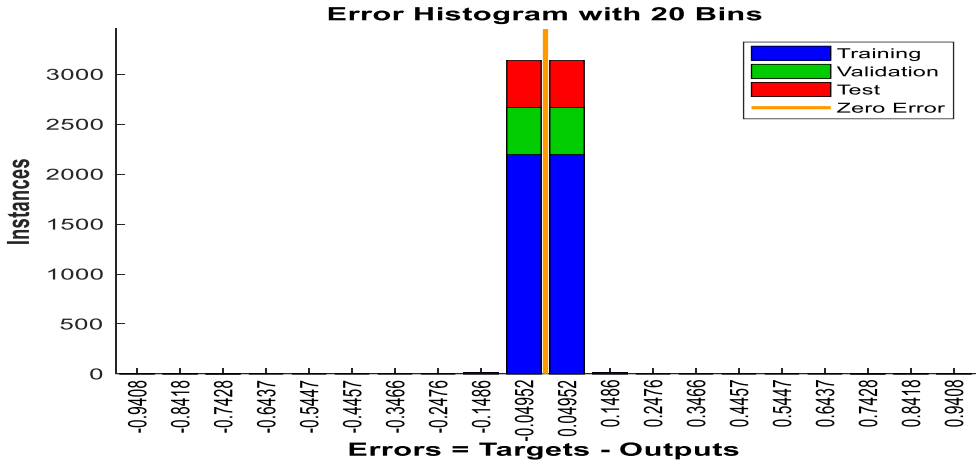

Figure 5.6: *Error Histogram*

For the same dataset we have used while we implement the classification model using ANN, we tried support vector machine for model implementation in order to achieve better classification accuracy in this work. Indeed we achieved 99.8% of accuracy by performing 26 obs/sec in 34.546sec training time.

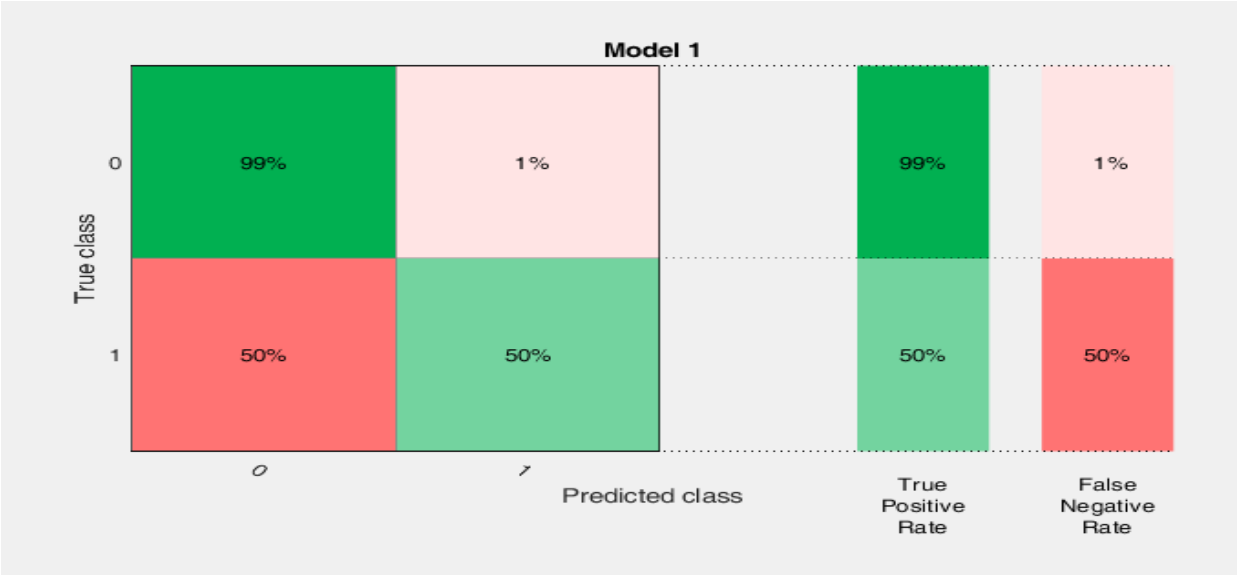

Figure 5.7: *Predicted Class, True Positive rate and False Negative rate Using SVM*

The Confusion matrix for evaluation of the model 1 is shown on figure 5.7, depicted that 99% of the training data were classified correctly. This model is performing less compering with the former model that uses ANN and achieved 99.8% classification accuracy.

Using SVM in the training process leads us to misclassify around 1% of our dataset during experimentation process of creating our classification model .

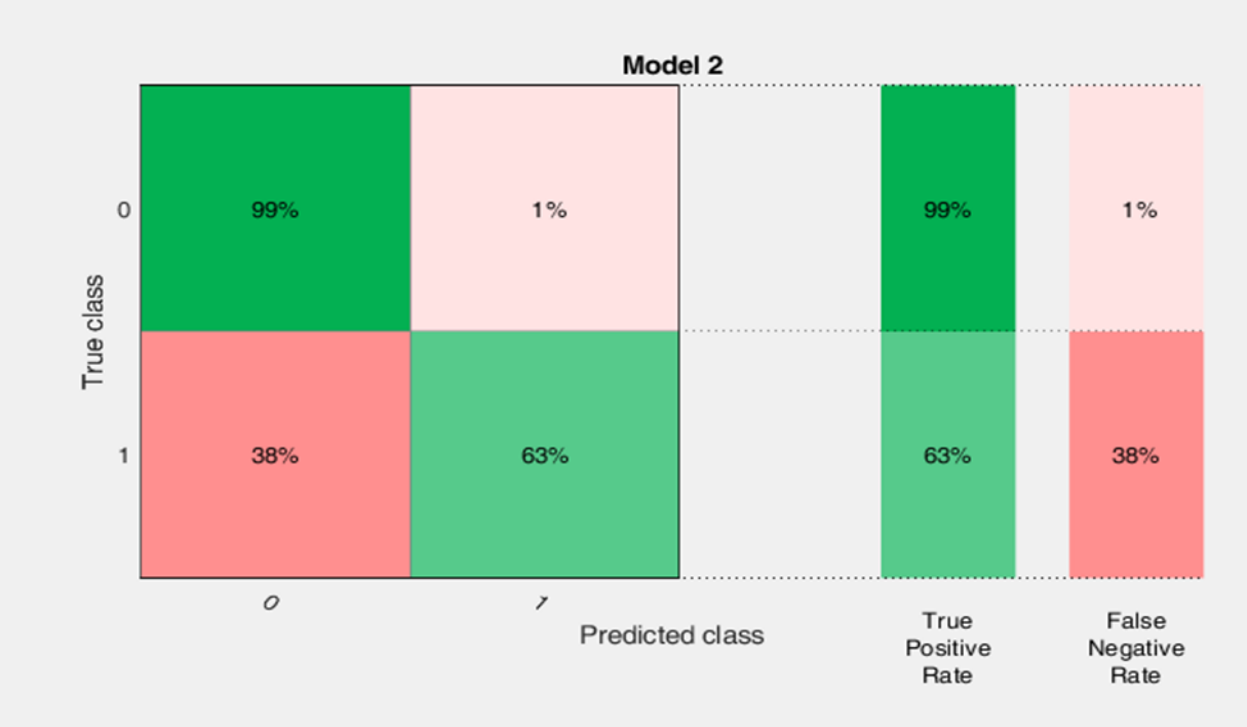

Figure 5.8: *Confusion Matrix Using KNN*

Figure 5.8, shows that KNN algorithms is used to implement the classification model too, although the accuracy achieved using KNN and SVM is the same, KNN uses more observation at a time i.e. 37 obs/sec and consume less time to train i.e. 23.979 sec. In order to take the best classifier among KNN and SVM we need to take another factor as their accuracy is similar. So, we decided to use the ROC curve. The ROC curve shows true positive rate versus false positive rate for the currently selected trained classifier. The marker on the plot shows the performance of the currently selected classifier. The marker shows the values of the false positive rate (FPR) and the true positive rate (TPR) for the currently selected classifier.

A perfect result with no misclassified points is a right angle to the top left of the plot. A poor result that is no better than random is a line at 45 degrees. The Area Under Curve number is a measure of the overall quality of the classifier. Larger Area Under Curve values indicate better classifier performance.

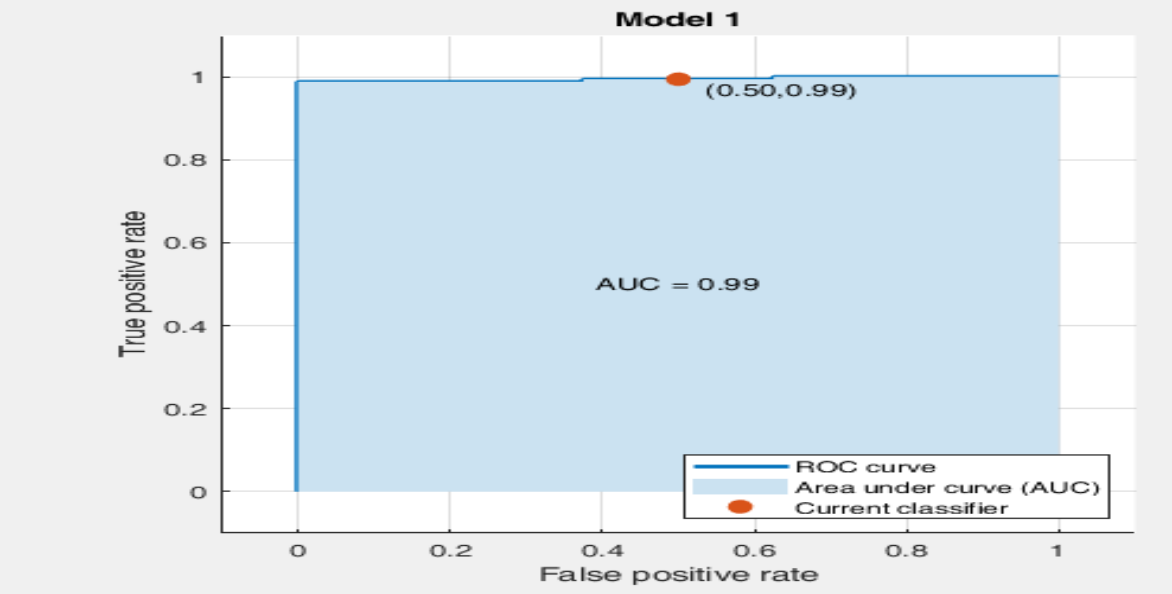

Figure 5.9: *ROC curve for SVM*

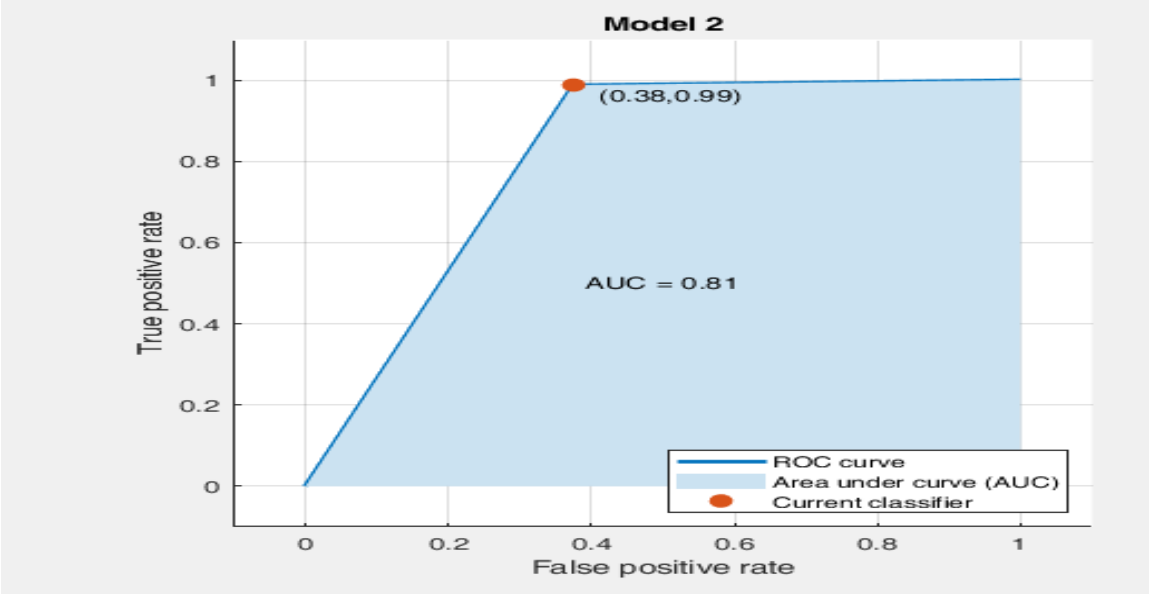

Figure 5.10: *ROC curve for the model using KNN*

According to the implementation result and figure 5.9 and 5.10 we can observe that the classification model 1 i.e. SVM is performing accurate and better KNN, but the classification accuracy of the first model, which uses ANN algorithm has better classification accuracy.

6. CONCLUSION, RECOMMENDATIONS AND FUTURE WORKS

In this work, android information leakage detection and classification of an application as a leaker or not were proposed to be achieved through the activity an application and features that can be shown at different states, i.e. pre-static, static and dynamic state. So, feature of different application have been extracted to train and build classification model by using various incremental supervised learning technique which can automatically distinguish an application as a leaker or not depending on the property of an application extracted to the data, then increment the number of data in a dataset. .

Experiment result shows that the classification model is able to identify an application with property in accurate manner i.e. 99.8% accuracy of classification. It also verify the fact that the use of mentioned information in feature set helps to achieve better result.

Generally, in this work a classification framework which is used to identify the leaker application can classify an application as a leaker or not depending on the classification model, which the performance were analyzed through experimentation. So, the classification model which is used by main classifier is a proved classification model which will be used to classify an application as a leaker or not. Additional, this model's performance will be enhanced as newly classified applications by framework will be added from time to time.

In the presented framework we used features which are mainly exposes the information on an android platform by extracting them at runtime and before execution. But, as we have used the intent tracking for specific application just only for 30 seconds, and didn't track the intents from source to sink, we couldn't perform the entire control flow graph(CFG)analysis and deep native code analysis. So they could be the main topic of concern for future work which will help us to achieve better accuracy.

Bringing the prevention and detection of information depending on the features used for modelling the classifier, to the android device level will also be the future work that will help for better achievement of this work.

REFERENCES

[1] Hongyi Chen, Ho-fung Leung, Biao Han "Automatic Privacy Detection for Massive Android Apps via a Novel Hybrid Approach" May 2017 DOI:10.1109/ICC.2017.7996335, Conference: ICC 2017 - 2017 IEEE International Conference on Communications

[2] Crager, Kirsten - Maiti, Anindya- Jadliwala, Murtuza, He Jibo- 2017/04/29 - Information Leakage through Mobile Motion Sensors: User Awareness and Concerns-10.14722/eurousec.2017.23013

[3] Clint Gibler, Jonathan Crussell, Jeremy Erickson , and Hao Chen "AndroidLeaks: Automatically Detecting Potential Privacy Leaks In Android Applications on a Large Scale" University of California, Davis.

[4] Hintea, Diana; Taramonli, Chrysanthi; Bird, Robert; and Yusuf, Rezhna, "Forensic Analysis of Smartphone Applications for Privacy Leakage" (2016). Annual ADFSL Conference on Digital Forensics, Security and Law. 7.

[5] Sapna Malik and Kiran Khatter, " AndroData: A tool for static and Dynamic Feature Extraction of Android Apps, " *International Journal of Applied Engineering Research*, ISSN 0973-4565 Vol. 10 No. 94(2015).

[6] Egele, m., kruegel, c., kirda, e. & vigna, G. PiOS: Detecting Privacy Leaks in iOS Applications. NDSS, 2011.

[7] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, andromaly: A behavioral malware detection framework for android devices J. Intell. Inf. Syst., 38(1):161190, 2012.

[8] M. Schultz, E. Eskin, and E. Zadok. Data mining methods for detection of new malicious executables. In Security and Privacy Proceedings IEEE Symposium, pages 38 49, May 2001.

[9] A. Sami, H. Rahimi, B. Yadegar, N. Peiravian, S. Hashemi, A. Hamze, Malware Detection Based On Mining API Calls The 25th ACM Symposium on Applied Computing-Data Mining Track, March 2010.

[10] Naser Peiravian and Xingquan Zhu Dept. of Computer & Electrical Eng. and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA {npeiravi, xzhu3}@fau.edu, Machine Learning for Android Malware Detection Using Permission and API Calls.

[11] Sanz, B.; Santos, I.; Laorden, C.; Ugarte-Pedrero, X.; Bringas, P.G.; Álvarez, G. Puma: Permission usage to detect malware in android. In International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions; Springer: Berlin/Heidelberg, Germany, 2013; pp. 289–298.

[12] J. D. Koli Scientific Analysis Group Defence Research & Development Organisation (DRDO) New Delhi, India. RanDroid:Android Malware Detection Using Random Machine Learning Classifiers

[13] Sahs, Justin, and Latifur Khan. "A machine learning approach to android malware detection." Intelligence and security informatics conference (eisic), 2012 european. IEEE, 2012.

[14] Li, Wenjia, Jigang Ge, and Guqian Dai. "Detecting malware for android platform: An svm-based approach." Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on. IEEE, 2015..

[15] Chan, j. j. k., tan, k. w., jiang, l. & balan, r. k. The Case for Mobile Forensics of Private Data Leaks: Towards Large-Scale User-Oriented Privacy Protection. 2013. 4th Asia-Pacific Workshop on Systems (APSYS).

[16] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In ACM Transactions on Computer Systems (TOCS'14), New York, NY, USA, 2014.

[17] Irolla, Paul & Filiol, Eric. (2017). Glassbox: Dynamic Analysis Platform for Malware Android Applications on Real Devices. 610-621. 10.5220/0006094006100621.

[18] Di Cerbo, F.; Girardello, A.; Michahelles, F.; Voronkova, S. Detection of malicious applications on android os. In Computational Forensics; Springer: Berlin/Heidelberg, Germany, 2010; pp. 138–149.

[19] Di Cerbo, F.; Girardello, A.; Michahelles, F.; Voronkova, S. Detection of malicious applications on android os. In Computational Forensics; Springer: Berlin/Heidelberg, Germany, 2010; pp. 138–149.

[20] Arp, Daniel, et al. "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket." NDSS. 2014.

[21] Spreitzenbarth, Michael, et al. ”Mobile-sandbox: having a deeper look into android applications.” Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, 2013.

[22] Wu, Dong-Jie, et al. ”Droidmat: Android malware detection through manifest and api calls tracing.” Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on. IEEE, 2012.

[23] Ilaria Pertot, Tsvi Kuflik, Igor Gordon, Stanley Freeman, Yigal Elad, Identificator: A web-based tool for visual plant disease identification, a proof of concept with a case study on strawberry, Computers and Electronics in Agriculture, Elsevier, 2012, Vol.88, p.144-154.

[24] Asma Akhtar, Shoab A. Khan, Arslan Shaukat, Aasia Khanum. Automated Plant Disease Analysis (APDA): Comparison of Performance of Machine Learning Techniques. 978-1-4799-2293-2/13, 2013, IEEE DOI 10.1109/FIT.2013.19.

[25] Mohammed J. Islam et al. (2007). Investigating the Performance of Naive- Bayes Classifiers and K- Nearest Neighbour Classifiers. IEEE 2007, 0-7695-3038-9/07, DOI 10.1109/ICCIT.2007.148.

[26] Arp, Daniel, et al. ”DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.” NDSS. 2014.

[27] Li, Wenjia, Jigang Ge, and Guqian Dai. ”Detecting malware for android platform: An svm-based approach.” Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on. IEEE, 2015.

[28] Wei, Y., Zhang, H., Ge,L., and Hardy, R. On behavior-based detection of malware on android platform. In Proceedings of the IEEE Global Communications Conference (GLOBECOM) (Dec 2013), pp. 814–819.

[29] Rastogi, V., Chen, Y., and Jiang, X. Catch me if you can: Evaluating android anti-malware against transformation attacks. IEEE Transactions on Information Forensics and Security 9, 1 (Jan 2014),