SCHOOL OF COMPUTING AND INFORMATICS

UNIVERSITY OF NAIROBI

MASTER OF SCIENCE IN DISTRIBUTED COMPUTING TECHNOLOGY

RESEARCH REPORT

STATIC ANALYSIS OF ANDROID LIBRARIES: DATA LEAKAGE

BY

KEVIN ATUNDA NYAKUNDI

P53/73139/2014

SUPERVISOR: DR. ELISHA ABADE

A research Report submitted to the Department of Information Communication and Technology in the School of Computing and Informatics in Partial fulfillment of the requirements for the award of the degree of Master of Science in Distributed Computing Technology of the University of Nairobi.

November, 2016

Declaration

I declare that this research is my original work and has not been presented in any other university/institution for consideration of any certification. This research has been complemented by referenced sources duly acknowledged. All the text, data including spoken words , graphics, pictures or tables have been borrowed from other sources, including the internet have been specifically accredited and referenced using the Harvard system and in accordance with anti-plagiarism

Signature______________________________________Date________________________________

Name	:	Kevin Atunda Nyakundi
Registration No	:	P53/73139/2014
Department	:	Computing and Informatics

Supervisor's declaration: This report has been submitted for appraisal with my approval as University Supervisor(s).

Signature____________________________________Date__________________________________

Name: DR. Elisha Abade.

Acknowledgement

First I begin by thanking God for his mercies and gift of life through each step of my academics right from when I stepped in baby class till now ,second I will thank my father ,mother and my two lovely sisters for being there to encourage me during this period of undertaking my project ,third my supervisor for accepting to be my supervisor and his time he spared in guiding me through, fourth I will take the school of computing for giving me an opportunity and enabling environment to undertake my study.

ABSTRACT

The possibility of android applications to spy on the users is real either intended or unintended. Considering that Java programs are based on large java libraries like Android SDK which must be understood to correctly reason about a program in static analysis. There is need to know its interaction with the library it uses. Analyzing these java libraries/Android libraries for each target application is highly inefficient and expensive. The study aimed at analyzing these libraries and in this case android Bluetooth library was analyzed and summaries computed. The summaries were later used to analyze randomly selected target applications for possible misuse with intention or unknowingly leaking user's data. For the analyzed application no possible leak was detected. The study also reviewed the current trends and developments in static analysis and proposed a new comprehensive android data leakage mitigation conceptual framework in static analysis. In this case survey of previous works and current was considered in validating the framework. Exploratory research approach was adopted for this study considering it is a tool that helps in understanding a phenomenon more and draw conclusions, lastly it helps build on what is already known or done by other researchers

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

Android data Leaks	-Genuine or Malicious applications gaining access to users Data and leaking it without user’s knowledge
Computed Summaries	-Methods that grouped according to their sensitivity in Regard to access to user’s data and facilitation of exit of The same data from the device to external servers or non-Secured location.
Sinks	-Methods that send data they obtain to external servers or To non-secure storage
Sources	-Methods that access or request for users data that is Considered sensitive for example getLocation
Static analysis	-program analysis of an application without executing it.
Call graphs	-is a directed graph that shows calling relationship among Subroutines and is produced by a program analysis tool.
Call graph Algorithms	-They are algorithms that are used to generate call graphs and they have to make tradeoffs between cost, complexity and accuracy

CHAPTER 1

1.0. INTRODUCTION

In the recent years android based devices mostly smart phones and tablets have experienced a steady growth, these can be attributed to android's operating system popularity, its flexibility, openness, development tools that can be downloaded for free and the ever growing support community online ranging from free online step by step tutorials, blogs, sites like GitHub and lastly android for developers website that has rich materials for developers. Which has resulted to an increase in the number of applications as of the start of 2015 there were over 1.2 million applications on the google play store market and also applications on third party android markets which do not implement any malware detection audits as stated by (Vigneri et.al (2015) ; Gascon et.al..(2013)).According to David Barrera(2012) an average Smartphone user installs 32 applications some for the same function ranging from communication, banking transactions and managing sensitive data that includes office documents , photos ,messages ,emails ,location ,address and others.

This growth and richness in private data and limitations in administrative device control by the smartphone users and security critical application like financial transactions has made it attractive to attackers, hackers and malware developers as supported by Parvez (2013). ,that are eager to lay their hands on users private data or sensitive information .Which makes it priority number one to secure this data by ensuring that applications do not leak users data without the user's consent or knowledge.

In securing these applications, several solutions have been suggested .One of them being static analysis, according to Gordon et.al (2015) static analysis is an attempt to analyze an application before execution for possible data leaks. This is where approximations of the possible behavior of a program are made.Tratt (2011) in his presentation defined static analysis with regard to malware detection as looking at a static program (source code or binary) and uncovering information, it is key in understanding the reasoning of the program without executing it, In this case in detecting data leakages in android applications static analysis frameworks attempt to analyze application for potentially sensitive information flows. Though there are challenges facing this approach; scaling to large applications and at the same time maintaining precision .In performing static analysis for data leakage in android applications ; the size, richness and complexity of the android API play a major role as accuracy is critical for a static analysis in trying to calculate security properties of an application as any inaccuracies could provide a hacker an opportunity to insert malicious flows that will result to them not being detected during analysis and imprecision of static analysis that leads to overestimations that results in many false alarms as supported by Xia et al. (2015),

A lot of research work has been done with regard to making static analysis efficient and applicable in android data leakages detection. According to Wel et al. (2014) due to inherent undecided ability attributes of determining code behaviors, each static analysis method must make a tradeoff between computing time, precision results and accuracy. Solutions have been centered in analyzing the android APIs and defining sensitive sources of data, some concentrating on mapping APIs and permissions they require which runs the risk of missed sources and sinks of sensitive data with a goal of reducing computational time and improving precision (Zheng et al.(2012);Shen et al.(2014);Gibler et al.(2012)).Though alike it does not answer the critical question on the role of Libraries that this applications rely on; what code that is actually called when a method is invoked and which implementers of that class are possible candidates and if among them there is a malicious one that will fetch data and send it to the attackers by considering interaction between the applications and the java and android libraries they rely on. To address this deficiency the study analyzed the android Bluetooth library and computed summaries that were used to analyze randomly selected target applications for possible misuse of the summaries to leak user’s data. Lastly the study also reviewed the current trends and developments in static analysis and proposed a conceptual framework.

1.1 Problem Statement

Despite static analysis being considered as the ultimate technique that can completely examine all data flows and detect possible data leakages in android applications, it generates false alarms and missed alarms due to it's over approximation and takes hours to examine a client application thus making it difficult to capture all usage patterns, enumerate or yield usable results.

1.2 Objectives

1. To investigate the current status and advancements in static analysis for android data leaks
2. Static analysis of android libraries and computing summaries that were to be used as black boxes when analyzing target applications.
3. Analyzing target applications with the computed summaries in the context of misuse of the analyzed library for data leaks

1.3 Research Questions

RQ1.What are the current developments and trends in static analysis of android application for data leaks detection?

RQ2.What is the current state of open source tools used in static analysis and how have they been adopted in android applications static analysis for data leaks detection?

RQ3.How can we get the code that is actually called when a method is invoked?

RQ4.which is the possible implementers of a class are possible candidates?

RQ5.Which possible implementations send data they get to an attacker?

RQ6.Which method in an application gets address and does it send it to an external server through the internet or write it to external storage?

RQ7.Which method in an application reads data from the external storage and sends it through Bluetooth?

1.4.1 Hypothesis

A complete analysis and computation of libraries implemented by android application can lead to a highly precise static analysis without knowledge of the code that will use the library later.

1.5.0 Significance of the research

This study is of great importance as it seeks to improve static analysis in solving data leakages in android based devices, as it seeks to ensure efficiency and effectiveness of static analysis that will go a long way in curbing data leakages.

1.6 Limitation of the study.

The study limited itself to one android library, this will not be able to provide a wide perspective of the libraries as this study provides an alternative that can be tried in future studies in studying the android libraries to predict possible data leakages in applications that will be implementing them.

1.7 Conceptual Framework

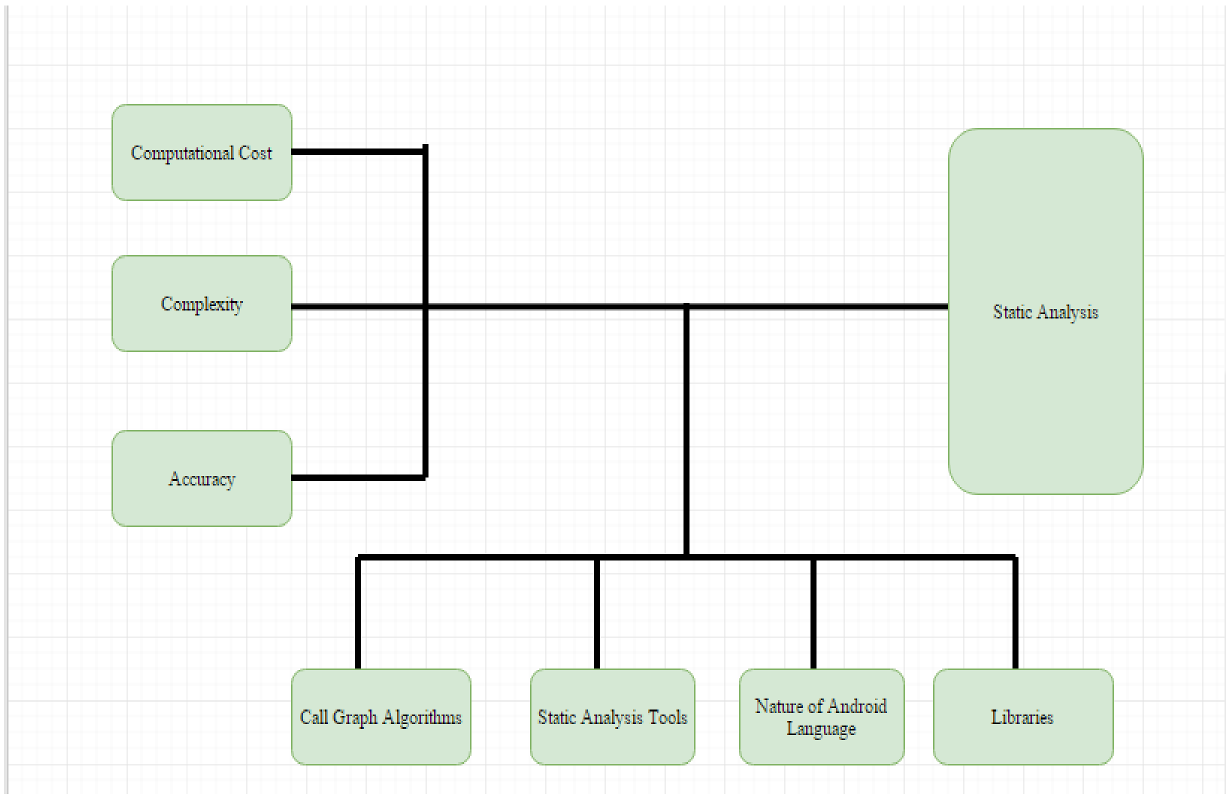


Figure 1: Conceptual Framework

Computational cost, Complexity, Accuracy are independent variables, static analysis is the dependent variable and lastly call graph algorithms, static analysis tools, nature of android language and libraries are moderating variables

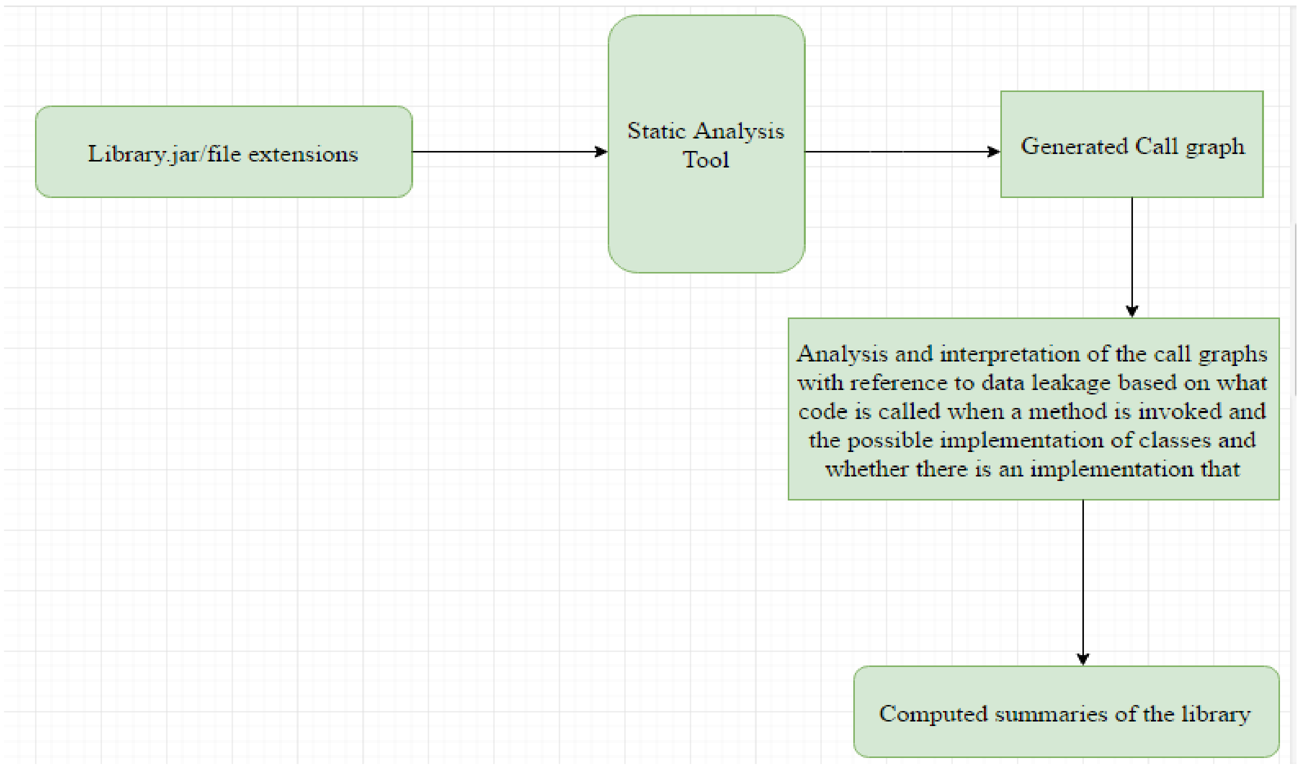

Library.jar/file extensions
Static Analysis Tool
Generated Call graph
Analysis and interpretation of the call graphs with reference to data leakage based on what code is called when a method is invoked and the possible implementation of classes and whether there is an implementation that
Computed summaries of the library

Figure 2: Work Flow

CHAPTER 2

2.0 Literature Review

Wal et al. (2014) in their study developed Amandroid which is a generic framework that conclusively points to information for all objects in android applications flows and context sensitive across components. They targeted android applications where they analyzed data flows within components and successful identified possible information leakages from sensitive sources to a critical sink by querying if there is data dependence in that chain from the source.

Zheng et al. (2012) in their study they proposed an in cross of static and dynamic analysis technique to reveal user interface based prompt conditions in android applications, they extracted expected activity switch paths by statically analyzing both activity and function calling relationships and for dynamically traversed each user interface elements by exploring the communication paths based on the sensitive APIs,it operated on simple principle where to find the right activity in the switch path a static switch path selector was used. In this case an app runs along unexpected activity call path where it triggers a sensitive behavior. The study borrows on their classification of APIs as sensitive for computing summaries of the android Bluetooth library.

Shen et al. (2014) proposed a flow permission mechanisms that extends the android permission system that incorporated semantic intelligence based on information flow, with an ability to indicate whether or not an application consists of a flow between a source and sink , checks the potential of an application to read private data and sending out through a sink though when it comes to information flows you cannot rule out missed information flows which can cause a lot of damage with regard to users private data thus need for a static analysis that can capture all patterns in this case information flows.

Li et al. (2014) in the their study they sought out to detect privacy leaks between components of components of android applications through inter component data flows, where they came up with IccTA tool that analyzes the context among the components to improve precision of the analysis to which the study will evaluate the applicability of their approach in achieving precision for the solution proposed by this study. Also in their static taint analysis technique where they used it to find out privacy leaks considering trails from sources, to statements sending data outside the application or device know as sink thus this will help in understanding what method is actually invoked and what code is actually implemented and if there is an implementation that really sends private data to an attacker.

Ali and Lhotak (2012) in their study they acknowledge the most common approach of constructing a call graph for a whole program analysis is to neglect all the consequences of the library code and all the calls that it invokes to the application. In generating call graphs in static analysis of android applications for possible data leaks, the possibility of missed paths and misused library code by malicious developers or knowingly or unknowingly use of advertisement libraries by developers exposing users private data to advertisement firms. Having this in mind makes that common approach unsound and unusable. In solving this they developed a CGC framework that generated a sound call graph that overestimates set of target at each call site in context of analysis scope and a set of reachable's for the application part of a program but does not analyze the library code instead makes assumptions about the library code by generating a summary node that represents methods in the library o and invoking separate compilation assumption argument, where they argue that the distinction between an application and the library it uses is not discretionary which they also acknowledge that in the case analysis scope was a set of classes at that point the call graph would be very imprecise. They concluded by saying that "separate compilation assumption is sufficient to construct a precise call graph" but considering the possibility of call backs there is need to know which code is called and possible implementations, they also recommend definition of multiple libraries and their dependencies with each and own library points to set .This study leveraged on this study to support its hypothesis.

CHAPTER 3.

3.0 Research Methodology

The study embraced Exploratory research methodology, literature survey was conducted where research papers, article, journals, reports ,online sources and books were selected ,the year of publication was considered where literature that was published between year 2010-2016 was considered and only where necessary literature that was published in early years was considered ,Key words were used in selecting and reviewing this literature; words like static analysis ,program analysis, Bluetooth security ,construction of call graphs in object oriented, android data leakages, android application security ,open source static analysis tools,Wala framework, Soot framework,javacg. Online documentations were considered too, in this case android developer website or documentation, java documentations from oracle website and information from technology websites. Kothari page 36.(2004) exploratory research "happens to be the most simple and fruitful method of developing hypothesis" ,priority number one in exploratory is to "discovery of ideas and insights and such design is appropriate for such studies as it must be flexible enough to provide opportunity for considering different aspects of a problem under study" and for survey of relevant literature states that it's "the most simple and fruitful method of formulating precisely the problem under study or developing a hypothesis and allows reviewing of the previous hypothesis and also helps the researcher build upon the work already done by others and also help application of concepts and theories developed in different research contexts to the area of study."

3.1 Research Design

Literature survey or review was conducted .The reviewing bit started start by searching literature from several digital libraries such as research libraries offline and online, Google Scholar, Android support websites and blogs through searching using keywords. Assessment of the found literature will be done through going through the abstract and conclusions and references will be considered for follow up this will help to expand understanding by obtaining more explanation of the reviewed concepts, next was reading and recording concepts from the selected literature the main sources being books, journals, articles, technical reports, online sources and content produced by various tools which is documented

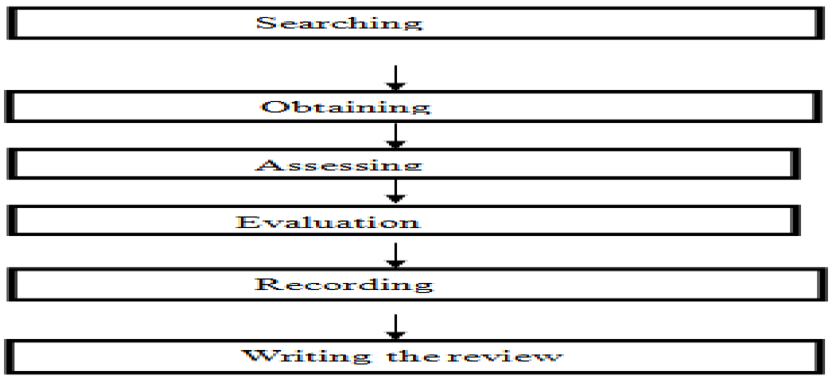


Figure 3: Data Collection Procedure

3.3 Sampling

The study featured various android Libraries that were to be selected for analysis and having in mind that to perform static analysis on android libraries you will need a library that is fully implemented and considering that android libraries do not come with implementations , a simple random sampling approach was adopted as it increases the chances of every possible Library to be equally selected to increase versatility of the sample size in this case 10 libraries were selected as the population where one of them is selected based on whether it had full implementation. This sample is easy to define but challenging to do.

Library	Availability of Implementation
Twitter	API
Instagram	API
Dropbox	API
Facebook	API
Android Bluetooth Library	Full implementation .jar
android GPS library,	API
Map Library	API

Advertisement Library	API
Wifi Library	API
Camera Library	API

Table 1 **:Sample of Libraries**

Sample size=n/N

N=10 n=1.

3.4 Data Collection tools and procedures

Qualitative data was collected from literature review of secondary sources data and primary sources considering generated data from java call graph suite program tool, considering it was in text form, qualitative was best suited for collection and analysis of data.

3.5 Data Analysis presentation and interpretation

Considering the study used one predictor, a continuous predictor variable will be used to predict a continuous criterion variable where.

$$Y=a+bx$$

Where Y=predictor criterion

x=an individual's score on the predictor variable

a= a constant calculated from the scores of all participants

b=the coefficient indicating the contribution of the predictor to the criterion

And for presentations graphs and charts were used to provide a visual representation of the relationships of variables.

A high degree of correlation between two variables does not imply that one causes the other, existence of a high negative correlation permits prediction, the correlation coefficient does not indicates the percentage of relationship between the variables, the correlation coefficient indicates the amount of common variance shared by the variables, common, or shared, variance will indicate the extent to which variables vary systematically.

3.6 Validity

In terms of validity of the study most specific the research design and methods for the literature review, the study considered the latest relevant sources were covered which ensured up to date finding and true representation of the current state without omission and mostly the recent secondary sources and follow up of the referenced materials and the coding techniques used ensured accurate representation of facts. For the second objective the required minimum population size has been selected to ensure generalization of findings and also in the case of time available to conduct the study, is enough to cover all the aspects of the study and all the data collection tools have been approved from other studies as being effective.

CHAPTER 4

4. 0 Introduction

4. 1 JAVA –CALL GRAPH OR JAVACG SET UP.

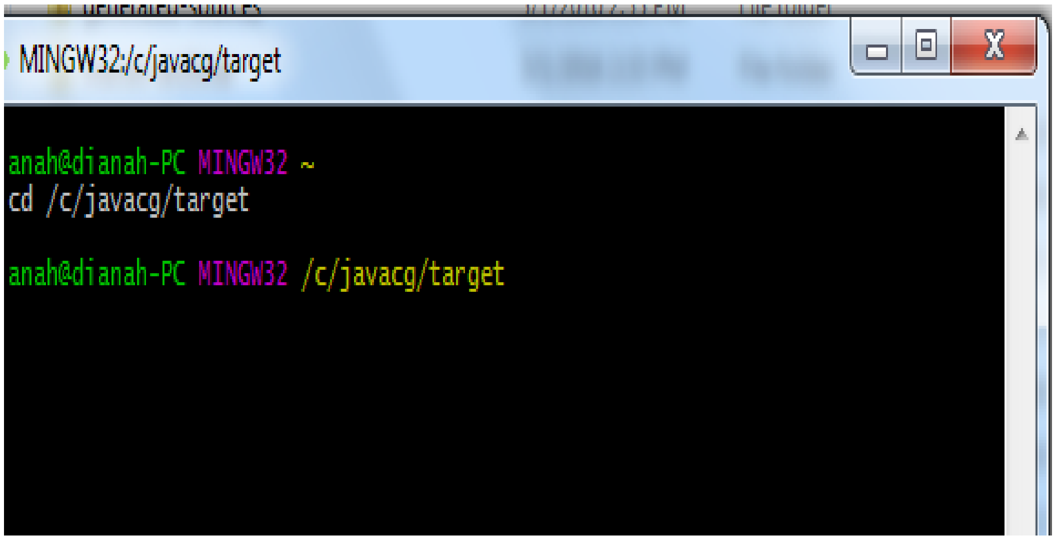
- After installing maven
- Run mvn install this produces a target directory with jars

  Javacg-0.1-SNAPSHOT.jar

  Javacg-0.1-SNAPSHOT.jar for static

  Javacg-0.1-SNAPSHOT.jar for dynamic
- Run javacg static from the "command prompt".

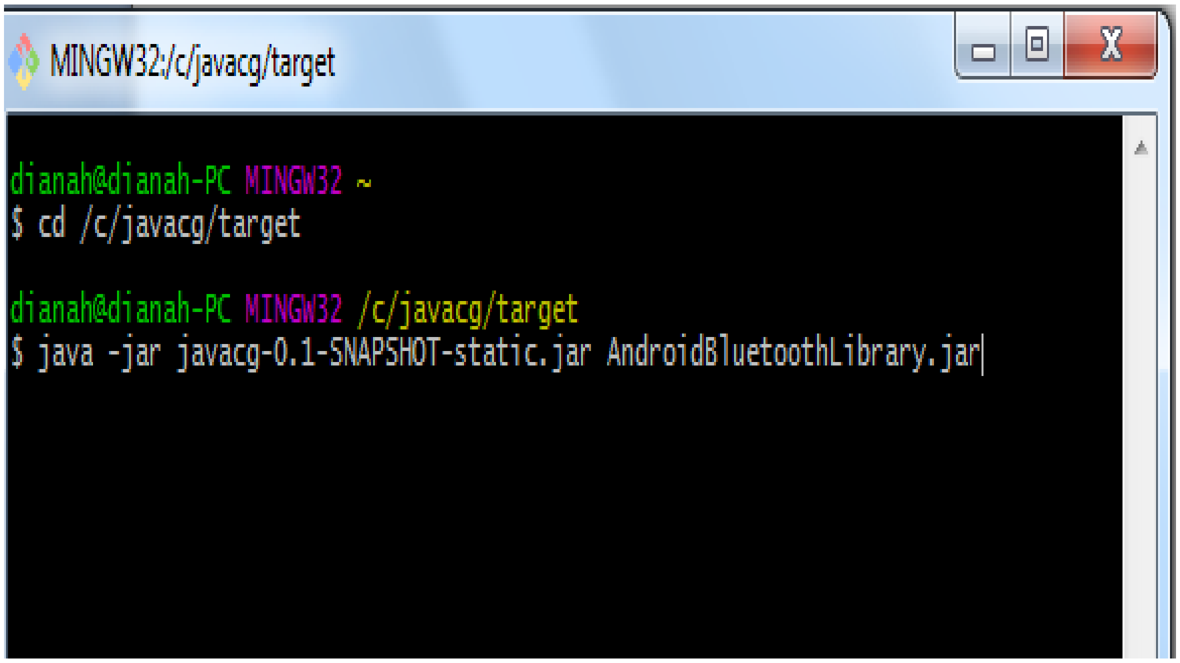
  Java –jar javacg-0.1-SNAPSHOT-static.jar AndroidBluetoothLibrary.jar



```
anah@dianah-PC MINGW32 ~
cd /c/javacg/target

anah@dianah-PC MINGW32 /c/javacg/target
```

Figure 4: Screenshot of set up



```
dianah@dianah-PC MINGW32 ~
$ cd /c/javacg/target

dianah@dianah-PC MINGW32 /c/javacg/target
$ java -jar javacg-0.1-SNAPSHOT-static.jar AndroidBluetoothLibrary.jar
```

Figure 5: Screenshot of the setup of JAVACG

Interpretation of the Results

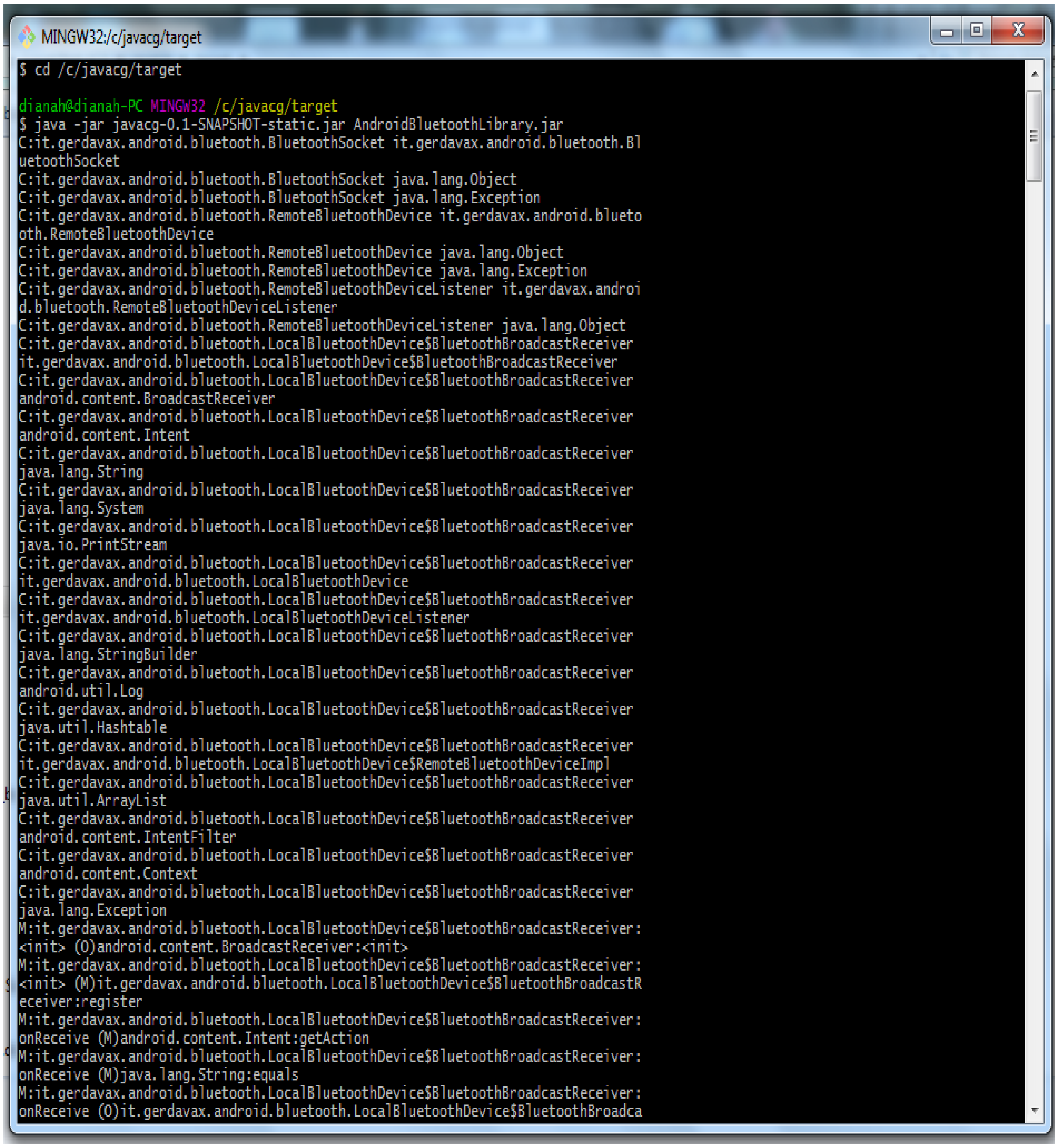
```
M: class1: (typeofcall) class2:
```

The line means that `method1` of `class1` called `method2` of `class2`. The type of call can have one of the following values.

- M for invoke virtual calls
- I for invoke interface calls
- O for invoke special calls
- S for invoke static calls

```
C: class1 class2
```

This means that some method(s) in class1 called some method(s) in class2.


MINGW32:/c/javacg/target

```
$ cd /c/javacg/target

dianah@dianah-PC MINGW32 /c/javacg/target
$ java -jar javacg-0.1-SNAPSHOT-static.jar AndroidBluetoothLibrary.jar
C:it.gerdavax.android.bluetooth.BluetoothSocket it.gerdavax.android.bluetooth.Bl
uetoothSocket
C:it.gerdavax.android.bluetooth.BluetoothSocket java.lang.Object
C:it.gerdavax.android.bluetooth.BluetoothSocket java.lang.Exception
C:it.gerdavax.android.bluetooth.RemoteBluetoothDevice it.gerdavax.android.blueto
oth.RemoteBluetoothDevice
C:it.gerdavax.android.bluetooth.RemoteBluetoothDevice java.lang.Object
C:it.gerdavax.android.bluetooth.RemoteBluetoothDevice java.lang.Exception
C:it.gerdavax.android.bluetooth.RemoteBluetoothDeviceListener it.gerdavax.androi
d.bluetooth.RemoteBluetoothDeviceListener
C:it.gerdavax.android.bluetooth.RemoteBluetoothDeviceListener java.lang.Object
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
android.content.BroadcastReceiver
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
android.content.Intent
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
java.lang.String
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
java.lang.System
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
java.io.PrintStream
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
it.gerdavax.android.bluetooth.LocalBluetoothDevice
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
it.gerdavax.android.bluetooth.LocalBluetoothDeviceListener
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
java.lang.StringBuilder
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
android.util.Log
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
java.util.Hashtable
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
java.util.ArrayList
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
android.content.IntentFilter
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
android.content.Context
C:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver
java.lang.Exception
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:
<init> (O)android.content.BroadcastReceiver:<init>
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:
<init> (M)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastR
eceiver:register
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:
onReceive (M)android.content.Intent:getAction
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:
onReceive (M)java.lang.String:equals
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:
onReceive (O)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadca
```

Figure 6: Screenshot of javacg results

4.2 DEXTER STATIC ANDROID APPLICATION ANALYSIS TOOL

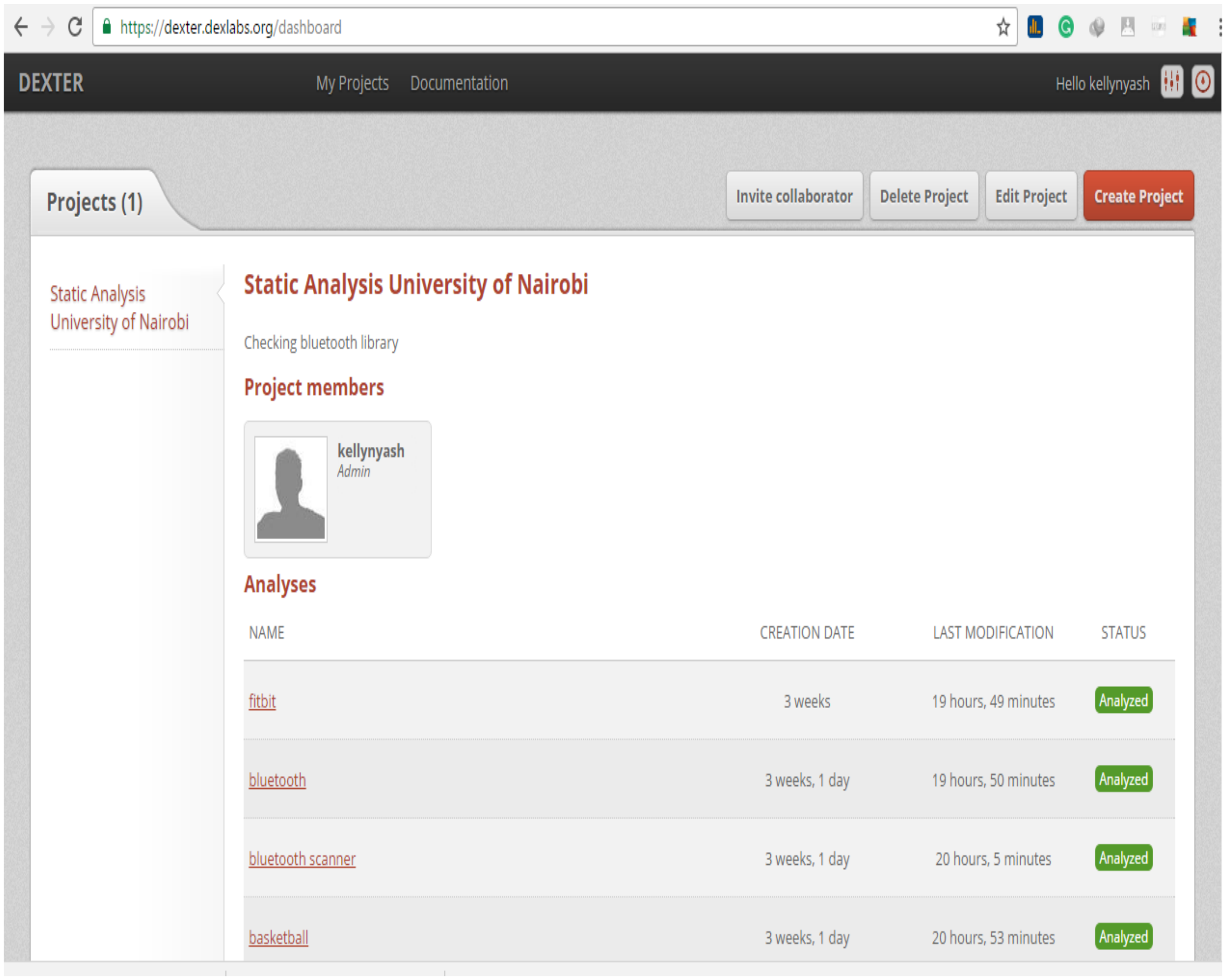
Dexter is an online based static analysis tool for android applications for possible malicious behaviors, thus it’s a malware analysis tool.

It resolves complex relations between objects, helps analyze objects of applications, like methods, classes and packages by generating call graphs or control flow graphs.

It decompiles dalvik bytecode, to enable looking at the internal of an application as it converts it to java class files thus the source like level.

4.2. 1 Steps of Using Dexter

1. Create an Account
2. Confirmation Mail link
3. Create a new project
4. Upload the Apk you wish to analyze
5. Wait until the process is done and you will have your results ready



https://dexter.dexlabs.org/dashboard

DEXTER My Projects Documentation Hello kellynyash

Projects (1)

Invite collaborator Delete Project Edit Project Create Project

Static Analysis University of Nairobi

Static Analysis University of Nairobi

Checking bluetooth library

Project members

kellynyash
Admin

Analyses

NAME	CREATION DATE	LAST MODIFICATION	STATUS
fitbit	3 weeks	19 hours, 49 minutes	Analyzed
bluetooth	3 weeks, 1 day	19 hours, 50 minutes	Analyzed
bluetooth scanner	3 weeks, 1 day	20 hours, 5 minutes	Analyzed
basketball	3 weeks, 1 day	20 hours, 53 minutes	Analyzed

Figure 7: Screenshot of Dexter

4. 3 Research questions and findings.

RQ1.What are the current developments and trends in static analysis of android applications for data leakages ?

a).Use of call graphs as a foundation for inter-procedural analysis .Frameworks and tools have been developed to construct call graphs for either whole program analysis or partial call graphs .In the context of data leakage analysis in android apps ,whole program analysis call graphs include the libraries that the applications interact with while partial program analysis call graphs involve construction of only application call graphs minus the libraries it interacts with or by making assumptions of the effects of the library.

They model the calling relationships between functions and they are used to find paths from one function to another,for example for finding out whether an application can leake users data in static API analysis where API functions are classified as sources of data and sinks of data meaning functions that can send information out of the device thus finding a path from G(f,g) f(source of data) to g(Sink of data)=data leakage..
source of these data(Bartel et al.(2014);Gibler et al.(2012);Mann et al.(2012);Zhang et al.(2014);Yan et al.(2014);Rayside et al.(2000);Babu et al.(2013);Grove and Chamber (2001);Mangal et al.(2014);Honar et al.(2010);Lam et al(2011);Grove et al(1997);Milanova et al.(2002);Sawin and Rountev(2011);Zhang and Ryder(2006);Liang(2014) and other referenced sources)

b).Call graph generation /construction algorithms.These algorithms overapproximate the possible result set to each call site,they are proposed by having in minds the steps made by previous call graphs based on precision,cost and accuracy.

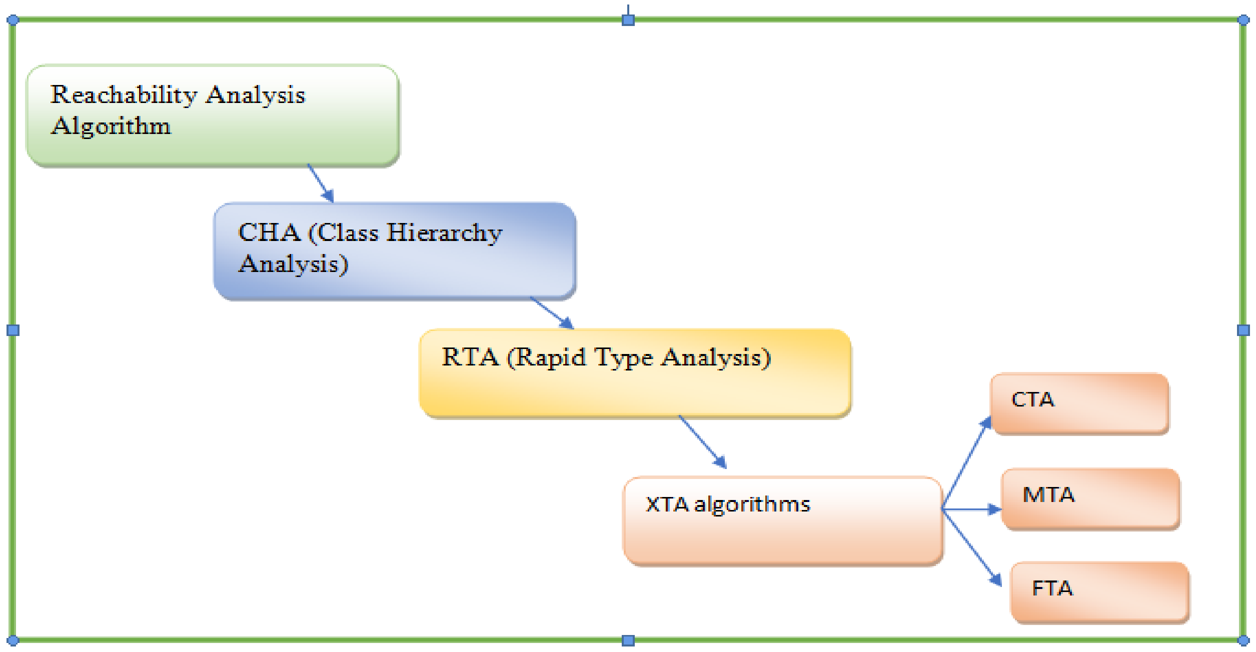

Reachability Analysis Algorithm
CHA (Class Hierarchy Analysis)
RTA (Rapid Type Analysis)
XTA algorithms
CTA
MTA
FTA

Figure 8: Advancements in Call Graph Generation Algorithms

Reachability Analysis is based on name resolution,produces large sets of recheables as compared to other algorithms ,it computes reachable methods from a call site and adding them to set of reachables.It does not take into account the method parameters,method signature or return types.It was the first algorithm to be used to construct call graphs of a program and it is considered very imprecise.

Class Hierachy Analysis is an advancement of Reachability analysis algorithm,produces less set of reachables as compared to RA,it takes into account the method signature and also constructs the whole program class hierachy before executing it.It is more precise as compared to Reachability Analysis.

Rapid Type Analysis improves Class Hierachy Analysis by considering class instantion information besides the class hierarchy information that adding to sets for consideration,meaning less reachable methods as compared to CHA algorithm.It is considered more precise as compared to CHA.

XTA()/CTA(Class Type Analysis) it is an improvement of RTA,the algorithm uses multiple set variables that range over a set of classes,these set variables are associated with program entities such as classes/methods and fields.Under XTA we have CTA,MTA,FTA. Read more in (Tip and Palsberg,(2000)).

Sources[Bartel et al.(2014);Gibler et al.(2012);Mann and Starostin(2012);Zhang et al.(2014);Yan et al.(2014);Lam et al.(2011) and others referenced] and for the algorithms[Rayside et al.(2000);Babu et al.(2013);Grove et al.(2001);Mangal et al.(2014)].

c).Generation of partial call graph or whole program call graphs for static analysis of android applications as a trade off to save cost and complexity (Ali et al.(2013;Ali and Lhotak(2012);Yan et al.(2012);Karim Ali(2014),Balatsouras et al.(2013));

RQ2.What are the available open source tools to analyze android and java applications and libraries ?

Tool/Framework	Type of Analysis	Features	Usability/Popularity
SOOT/SPARK	-Construction of Call graphs -Point to Point Analysis -Taint Analysis -Def/Use Chains -Interprocedural Analysis -Intraprocedural Analysis	Takes Java bytecode,android bytecode,Jimple and Jasmin as inputs and outputs it can also transform from one translation to another for example from android to java	40%
WALA	- class hierarchy analysis -Interprocedural	Supports java and javascript	30%

	dataflow analysis -Context-sensitive tabulation-based slicer -Pointer analysis and call graph construction		
DOOP	-pointer analysis -sophisticated relection Analysis -precise exception analysis -subset-based (or inclusion-based) analyses - a context-sensitive heap abstraction (also known as heap cloning or heap specialization) -context sensitive pointer Analysis -flow-insensitive pointer analyses -field-sensitive analyses		15%
JAVA-CALLGRAPH	-generates Call graphs -Static Call graphs and Dynamic		5%
Others eg.AndroidWarn,	-detects and warns the user about potential		10%

	malicious behaviors of an Android application. Performs static analysis of the application's Dalvik byte code		

Table 2: Open source tools

As documented by wala framework,soot framework,java callgraph framework

20 papers were consisdered in coming up with the above percentage were 8 papers used soot, while 6 used wala,while three referenced doop,while 2 are other tools and frameworks while Java Call graph suite programs is 5% this can be attributed to it not allowing the user any control of the analysis or even define the analysis scope as compared to other tools which are even more advanced and this study preferred this tool as it is simplified and in the study we do not define the analysis scope .

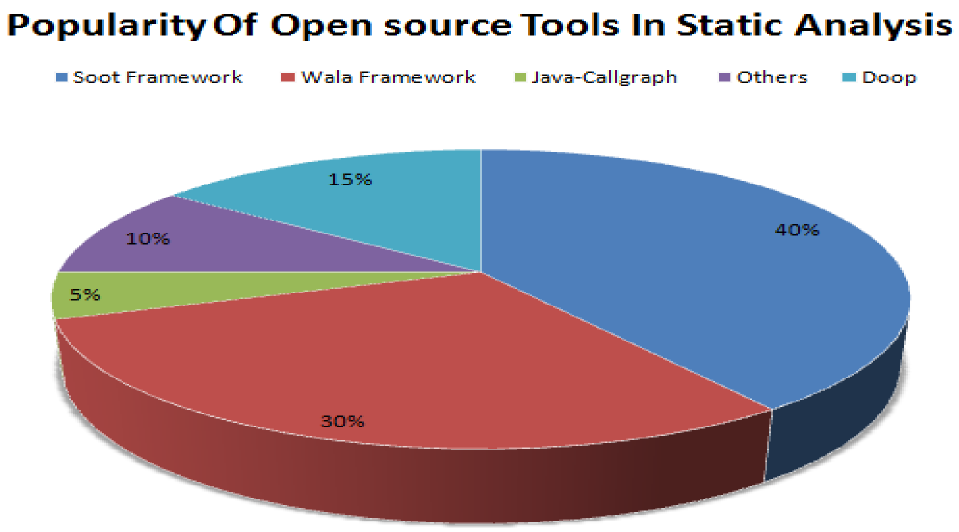


Figure 9: Popularity of open source static analysis tools pie chart

RQ3.How can we get the code that is actually called when a method is invoked?

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:<init>
(M)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:register.

```
public void register(Context context)
        {
                if(!registered)
            {
                Log.d("BluetoothBroadcastReceiver", "Registering");
                IntentFilter intentFilter = new IntentFilter();

intentFilter.addAction("android.bluetooth.intent.action.BLUETOOTH_STATE_CHANG
ED");

intentFilter.addAction("android.bluetooth.intent.action.REMOTE_DEVICE_FOUND")
;

intentFilter.addAction("android.bluetooth.intent.action.DISCOVERY_COMPLETED")
;

intentFilter.addAction("android.bluetooth.intent.action.DISCOVERY_STARTED");

intentFilter.addAction("android.bluetooth.intent.action.PAIRING_REQUEST");

intentFilter.addAction("android.bluetooth.intent.action.BOND_STATE_CHANGED_AC
TION");
                context.registerReceiver(this, intentFilter);
                registered = true;
            } else
            {
                Log.d("BluetoothBroadcastReceiver", "Already registered");
            }
        }
```

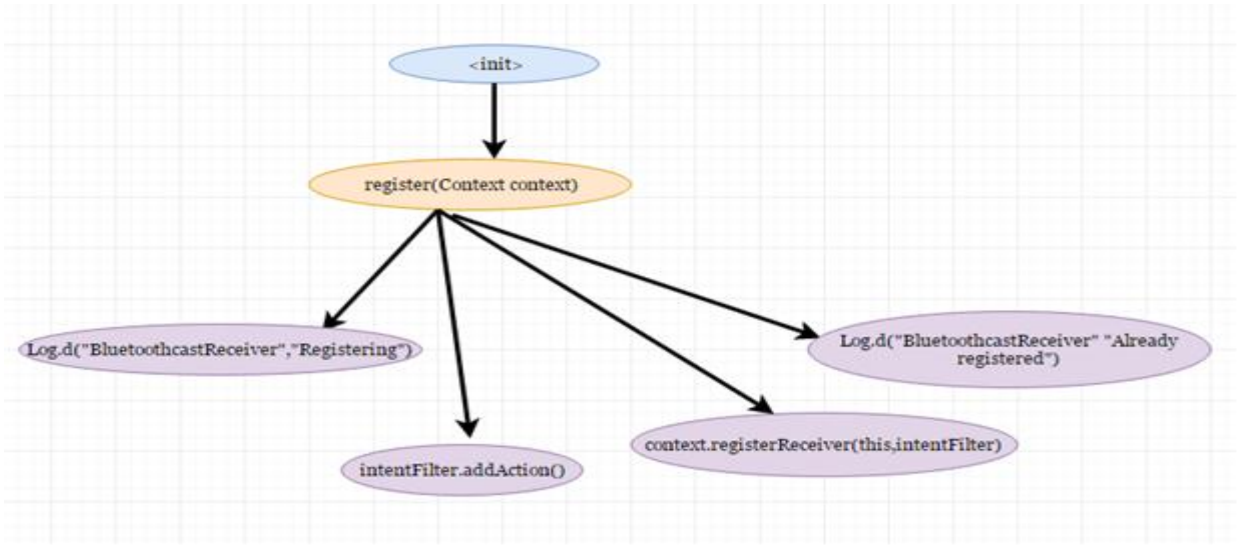

<init>
register(Context context)
Log.d("BluetoothcastReceiver","Registering")
intentFilter.addAction()
context.registerReceiver(this,intentFilter)
Log.d("BluetoothcastReceiver" "Already registered")

Figure 10: Call site

True is returned if the device is successfully registered and already registered if the device was previsiously registered.

RQ4.which are the possible implementers of a class are possible candidates ?

a) BluetoothSocket

	Possible implementers	**Explaination**
1.	Connect()	a connection with a remote device/opens sockets
2.	getConnectionType()	gets the type of connections either L2CAP/RFCOMM
3.	getOutputStream()	get the output stream associated with this socket (Sink or exit)
4.	getRemoteDevice()	device it's connected to
5.	isConnected()	checks the status of the socket
6.	Close()	closes the associated object and releases the system resources its holding
7.	getMaxTransmitPacketSize()	Maximum supported transit packet size
8.	getMaxReceivePacketSize()	Maximum supported receive packet size
9.	getInputStream()	gets input stream for the associated socket(Source)

Table 3:BluetoothSocket Possible implementers

b) RemoteBluetoothDevice

	Possible implementers	**Explanation**
1.	`getName()`	Returns the friendly Bluetooth name of the remote device
2.	`getAddress()`	Returns the hardware address of the Bluetooth adapter MAC address.

3.	`getRSSI()`	Obtains the received signal strength
4.	`getDeviceClass()`	Obtains the Bluetooth class of the remote device
5.	`setPin(String s)`	Sets the pin during pairing and requires Bluetooth_Admin permission.
6.	`pair()`	It allows connecting of two or more devices through request for a pin code.
7.	`pair(String s)`	
8.	`isPaired()`	Checking whether the devices are paired /Checking status

Table 4:RemoteBluetoothDevice Possible implementers

c). RemoteBluetoothDeviceListener

	Possible implementers	**Explanation**
1.	Paired()	Queries paired devices
2.	pinRequested()	Requests for the Pin

Table 5: RemoteBluetoothDeviceListener Possible implementers

d.) LocalBluetoothDeviceListener

	Possible implementers	**Explanation**
1.	Enabled()	Enables the adapter if it is un enabled
2.	Disabled()	Disables the adapter
3.	scanStarted()	Starts scanning or discovery of within range devices
4.	scanCompleted(ArrayList arraylist)	The scan completes with a list of with range Bluetooth devices

Table 6: RemoteBluetoothDeviceListener Possible implementers

d.) LocalBluetoothDevice

	Possible implementers	**Explanation**
1.	isEnabled()	Returns true if the adapter is enabled
2.	getPort()	
3.	getManufacturer()	
4.		

Table 7: LocalBluetoothDevice Possible implementers

RQ5.Is there an implentation that can lead to data being sent to an attacker ?

	Method	**Source**	**Sink**	**Sensitivity**
1.	Connect( )		✓	High
2.	getConnectionType()	✓		High
3.	getOutputStream()	✓		High
4.	getRemoteBluetoothDevice()	✓		Low
5.	isConnected()		✓	Low
6.	Close()		✓	Low
7.	getMaxTransmitPacketSize()	✓		Low
8.	getMaxReceivePacketSize()	✓		Low
9.	getInputStream()	✓		High

10.	`getName()`	✓		high
11.	`getAddress()`	✓		high
12.	`getRSSI()`	✓		high
13.	`getDeviceClass()`	✓		
14.	`setPin(String s)`		✓	High
15.	`pair()`		✓	High
16.	`pair(String s)`		✓	High
17.	`isPaired()`		✓	
18.	Paired()		✓	
19.	pinRequested()		✓	
20.	Enabled()		✓	Low
21.	Disabled()		✓	Low
22.	scanStarted()		✓	Low
23.	scanCompleted(ArrayList arraylist)		✓	Low
24.	isEnabled()		✓	Low
25.	getPort()	✓		High
26.	getManufacturer()	✓		High
27	setPin		✓	High

Table 8: Methods Classification

Sensitivity was classified as either low risk or higher risk ,by considering sensitivity of the data involved in terms personnal information of a user.

Sources ((*Android 7.0 Nougat!*, no date))

RQ6.How applicable are these computed summaries?

	Application Name	**Bluetooth Permission**	**Bluetooth_ Admin Permission**	**Recommendations**
1.	Basketball_101.apk	✓	✕	Further analysis
2.	Classic_Drums_2.5.apk	✕	✕	No further Analysis
3.	WhatsApp_Messenger_v2.16.247_apkpure.com.apk	✓	✕	Further Analysis
4.	BeautyPlus_Selfie_Editor_v6.2.6_apkpure.com.apk	✕	✕	No further Analysis
5.	Soccer_sinGoo_103.apk	✓	✕	Further Analysis
6.	World_war_Clash_of_Zombies_v1.1_apkpure.com.apk	✕	✕	No Further Analysis
7.	Bluetooth_V1.5_apkpure.com.apk	✓	✓	Further Analysis
8.	Bluetooth_scanner_v1.1_apkpure.com.apk	✓	✓	Further Analysis
9.	M_Chapaa_v1.2_apkpure.com.apk	✕	✕	No further Analysis
10.	M_Money_v1.0_apkpure.com.apk	✕	✕	No Further Analysis
11	Messenger_v82.0.0.17.75_apkpure.com.apk	✓	✕	Further Analysis

.				
12.	Airtel_mHealth_v1.9_apkpure.com.apk	✕	✕	No further Analysis
13.	Thesaurus_Synonyms_offline_v1.0.12-en.apk	✕	✕	No further Analysis
14.	Kopo_kopo_v2.2.2_apkpure.com.apk	✕	✕	
15.	Fake_GPS_Location_Spoofer_Free_v4.3.5_apkpure.com.apk	✕	✕	No further Analysis
16.	Fitbit_v2.29_apkpure.com.apk	✓	✓	Further Analysis
17.	PayPal_v6.5.1_apkpure.com.apk	✕	✕	No further Analysis
18.	OLX_v5.0.6_apkpure.com.apk	✕	✕	No further Analysis
19.	Love_cal_dnn.apk	✕	✕	No further Analysis
20				

Table 9: Table of Summaries

4.3.1 Further Analysis of the Recommended Applications

1. Is there a method that gets address and sends it to an external server through the internet or write it to the external storage?

2. Is there a method that reads data from the external storage and sends it through Bluetooth?

App Name	Number of Reachable Methods	Summary of findings	Qn1.	Qn2.

Basketball_101.apk	36482	The application requests for Bluetooth permissions, that allows it to connect to all paired devices .The application has access to the internet and also has access to network information getAddress(),getDefault(),getBonded(),getName,append(),toString(),getMethod(),checkCallingpermission().	×	×

Table 10:Basketball_101.apk analysis summary

App Name	Number of Reachable Methods	Summary of findings	Qn 1.	Qn 2.
WhatsApp_Messenger_v2.16.247_apkpure.com.apk	52872	It has access to Bluetooth, allowing applications to connect to paired Bluetooth devices but it does not have Bluetooth_ADMIN permission which allows it to discover and pair Bluetooth devices. It has access to internet and can write to external storage	×	×

Table 11: Summary of WhatsApp Analysis

Application Name	Reachable Methods	Summary of findings	Qn 1.	Qn 2.
Bluetooth_V1.5_apkpure.com.apk	1118	The application can pair to any device ,discover and pair and it open network sockets isenabled(),getDefaultAdapter(),enable(),disabled().	×	×

Bluetooth_scanner_v1.1_apkpure.com.apk	1481	The application has permission to pair any device,discover and pair devices and lastly it has permission to open network sockets getName(),getBondedState(),getAddress( ),append,toString(),startDiscovery(),isDiscovery(),getDefaultAdapter(),enable(),cancelDiscovery()	✕	✕
Fitbit.apk	64907	The application has access to networks information, internet, Bluetooth ,Bluetooth_Admin meaning it has permission to discover and pair to devices ,internet permission to open network sockets ,Read_External storage is indicated as unknown. Implements BluetoothManager,BluetoothGatt,BluetoothGattServerCallback,BluetoothGattCallback,BluetoothGattCharacteristic,BluetoothGattDescriptor, BluetoothAdapter,,BluetoothGattServer, BluetoothDevice, BluetoothGattService,	✕	✕

N/B

Table 12: Bluetooth_V1.5_apkpure.com.apk, Bluetooth_scanner_v1.1_apkpure.com.apk ,Fitbit.apk analysis summaries

Fitbit--- Further analysis should be done with regard how the app communicates with fitness devices whether the data it obtains ends up being sent to unauthorized servers

4.4 Summary of findings

The following previous work was considered in this study. Static analysis of object oriented languages considering that android applications implement OOP principles. Static analysis of android applications for malicious behaviors includes applications that leak data without intending to and malicious applications. Challenges and solutions for static analysis of android applications, this included the limitations of static analysis in being able to capture all program patterns and the challenges posed by the nature of android. Static analysis for extracting permissions where API call graphs are generated and mapped against the required set of permissions in solving the permissions gap problem, here API functions are classified as sources and sinks and by having a path from a source and a sink is equivalent to a data leak. The study of android applications security works was also reviewed, the android security models and definition of its limitations and possible vulnerabilities which are crucial in static analysis considering it's performed without running the applications or libraries.

What's clear from collected data is the key role played by the call graphs in static analysis of android application for possible data leaks, where they have been used for performing taint analysis, point to point analysis, entry point analysis, data flow analysis, class hierarchy analysis, context sensitive analysis, information flow analysis and others that are mentioned in other studies and they all use call graphs as starting points for analysis. The static call graph helps in knowing procedures that are calling other procedures and from which point of a program. From the reviewed literature it has been noted that a precise call graph leads to precise static analysis which in return means accuracy of data leaks detection in android applications also it has been noted that static analysis due to it being an over approximation, it takes hours in generating call graphs and computing summaries of applications and large set of reachable methods resulting to imprecise results ,From the surveyed literature much has been achieved in ensuring the generated call graphs are sound to ensure sound static analysis of applications which will result to positive results in detecting data leaks.

Static analysis has faced challenges that are noted, starting from the nature of android being implemented using java language which is an object oriented language that comes with various features that are good though pose a challenge to static analysis of applications or libraries for possible data leaks. They range from polymorphism, reflection, inheritance, multithreading and static and dynamic binding, method overriding, method loading, encapsulation, object cloning and lastly the android components. The large size of applications and libraries also pose a challenge during analysis considering the thousands of codes and as it was seen during analysis of the applications averagely 40,000 to more than 100,000 method references.

Various efforts from the research community and security experts have been recorded in literature in ensuring the effectiveness static analysis this has been done through advancements in call graph algorithms, static analysis tools and frameworks, tackling challenges brought about by the nature of android language ,human aspect in analysis and lastly the Libraries that applications interact with.

The study considered literature on call graph algorithms for object oriented algorithms, revised Algorithm for incremental call graph, call graph construction frameworks. The most common algorithms are the likes of RA algorithm which only takes into account the name of function to get a set of reachable methods from a call site, the advancement of it was the CHA algorithms that added the method signature and class hierarchy generation before starting analysis. RA analysis .XTA/CTA, KRB, Anderson and .Their ranking can be seen from fig 3. Preciseness of the algorithms comes at a cost of computation and complexity. What is clear from the analyzed data is that different algorithms and program representation choices affect precision and cost.

From the reviewed works, soot framework website and email lists, wala website, doop framework website and javacg resipository github. Tools that have been lately been used in static analysis were identified such as the most popular one being soot framework that is mostly used by researchers and other users who are interested in call graph construction, point to point analysis, taint analysis and interprocedural analysis and its popularity can be attributed to its support for android bytecode as compared to other frameworks and tools like Wala that supports java bytecode and JavaScript, Doop framework which is a point or point analysis of java programs supports java bytecode and lastly java call graph suite of programs that supports java bytecode to generate static and dynamic call graphs. Lastly the ease or user friendliness of these tools, through interaction of these tools and as a first timer in static analysis, soot framework for the beginners maybe not be the first tool to start with as it requires at least previous knowledge in static analysis and considering that android libraries and android applications do not have a main method as compared to java call graph which does not require custom main method or previous knowledge of static analysis.

Java Call graph suite of programs was used to construct a call graph of the android bluetooth library 2.1.The generated Call graphs aided this study to find out the code that is actually called when a method is invoked. Where M: class-A (type of call whether its virtual call/interface call or special call) class: M .Which means that method1 of class-A calls method2 of class-B and for the type of calls

- ✓ M for invokevirtual calls
- ✓ I for invokeinterface calls
- ✓ for invokespecial calls
- ✓ S for invokestatic calls

For the Classes

`C: Class-A Class` -B

Which means that some method(s) in class-A called some method(s) in class-B?

This call graphs that represent call relationship from call site to possible reachable enables the user find out what code is called when a method is called and it creates paths

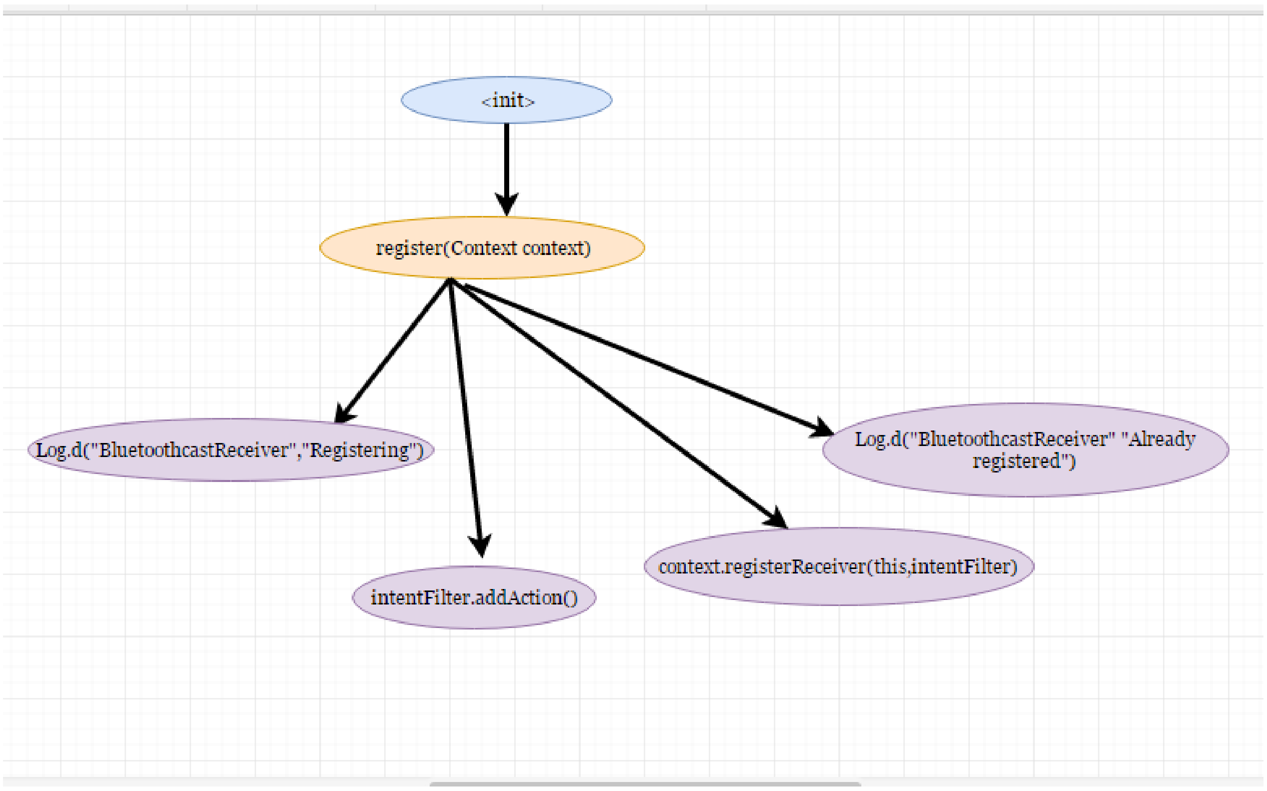

<init>
register(Context context)
Log.d("BluetoothcastReceiver","Registering")
intentFilter.addAction()
context.registerReceiver(this,intentFilter)
Log.d("BluetoothcastReceiver" "Already registered")

Figure 11:Call site

A=Call site (Register (Context context)) Method

{Log.d("BluetoothcastReceiver","Registering"),intentFilter.addAction(),context.registerReceiver (this,intentFilter),Log.d("BluetoothcastReceiver""Already registered"), Register }$\in$ Reachables They are considered as reachables from call site register( ).

All the possible implementers of classes that are declared as interfaces being BluetoothSocket,remoteBluetoothDevice,remoteBluetoothListener,localBluetoothDeviceListener ,localBluetoothDevice that are implemented by BluetoothImp and remoteBluetoothDeviceImpl.This implies that all the methods of these interfaces have to be implemented if not so declared as abstract by the classes implementing them.

Knowing whether there is a possible implementation that results in data leaks. Previous work in the definition of sensitive flows, data flow analysis, taint analysis, path analysis and work that has been done with regard to bluetooth security and possible threats and vulnerability were considered. This works helped in defining the sensitive functions that if they are implemented by either intentionally or unknowingly can be exploited by the attackers and to send data to the attacker's servers who in turn use the information obtained to either launch an attack to steal the user's data or track the user. Despite the requirement for these functions to have permission to access the bluetooth resource it is possible for them to rely on malicious applications or poorly developed application to leak the device information in this case address, name and profiles. The ability of the device to create a connection, open sockets, allows input streams and output streams. The code that is actually called when they are invoked is analyzed and by considering what is returned by the function, example for getAddress () address is returned, getName () name is returned and also parameters are checked for example the address parameter, if it's a parameter in within the implemented code: `getDeviceClass ()` invokes `deviceClass = getRemoteClass (address);`.Lastly works that have been done in classifying sources and sinks were considered as they give all sources and sinks in android framework ranging from locations, phone, Bluetooth, database ,wifi , contacts, email and internet.

Twenty applications were selected randomly, considering the availability of the APKs. The APK files were downloaded from online **https://apkpure.com/app.**After obtaining the apks they were uploaded into Dexter platform for analysis and averagely it took 15 minutes for the analysis to complete. The first to be noted is the permissions the application asks for whether it has access to Bluetooth/Bluetooth_Admin , internet , access to network status , external storage for read or writes purposes. Those with the permissions were selected for further analysis while those without these permissions were not considered for further analysis as Bluetooth can only be accessed by those applications with permissions.

The four applications that were recommended for further analysis were analyzed based on the computed summaries, where the methods were analyzed considering the callee and callers of the methods. For example getAddress was considered as the source of data "address" and whether there were callers that requested for the address and the destination. Also the getInputStream and getOutputStream was considered and the possibility of them exiting the device, toString, append, getDefaultAdapter were also considered and other sinks

4.5 Discussion

The growth of android applications is evident and the curve will always be rising with time .These applications will likely be at the center of current and future privacy concerns considering that almost every aspect of our lives is on our smartphones ranging from our communication,social life,banking/economy and even our political life.The richness of these applications has attracted hackers who are targeting users data through leaking them to external servers or external storage where there are no security checks or unprotected areas,also the likely hood of genuine applications being able to leak or be used by malicious applications to leak data due to vulnerabilities that exist within them as supported by(Rastogi et al.(2013);Liang(2014)).

The first question that we will ask ourselves is whether android does provide enough security mechanism to protect the users and the answer to this is evident that the protection framework faces challenges that range from poor programming practices by the developers who will declare so many permissions that the application does not need,allowing them to be used by other malicious applications,developers using advertisment libraries without full knowledge what risk they pose to the users,developers allowing applications write to unsecured storage allowing that data to be fetched by malicious applications that leak them as supported by Felt et al.(2011) . The security framework leaves the responsibility of ensuring that data is not leaked to the users who are tasked with the resposibility of accepting the permissions the applications are asking for before installation.Its evident that most users do not understand the permissions they are giving these applications and their capabilities or they do not have enough information what they do with the permissions and do not have control after installing them.Felt et al.(2011) agrees that users are unable to evaluate the requests and have little choice in regard to which specific permissions of an app they grant and discard others as the only option is either to agree to all or reject installation of the application. Thus escalating the issue of data leaks in android applications.

Next stage the applications get their way to android markets or play stores where they are downloaded by the users,the play store admnistrators have a responsibility to protect the users from malicious apps that are after their data and poorly developed applications that risk users data.Taking into consideration the limitations or risk posed by the naïve user with regard to security,the sole responsibilty to protect the user at this stage is taken by the admnistarors .It is evident that due to the openness of android it allows installation of application from third party markets that do not perform any security checks or analysis as supported by (Neuner et al.(2013)).It has also been cited that even the android google playstore despite having security checks its rate of success is fairly low and thus it can be bypassed thus making it a challenge to detection of possible data leaks and Neuner et al.(2013) agrees with this statement and also adds by saying that due to the huge volumes of apps being published considering that google play store alone has over one million applications and over 20,000 new applications released every month.

With all these in consideration it makes it necessary for the research community and security experts to come in and try to protect smartphone users from possible leaks of their private data.This stage is considered as the last stage in the cycle to possiblity detect possible leaks ,if it fails at this point them the damage will be grave ,making the users be victims of malicious developers and poorly done applications.

Static analysis is one of the approaches adopted by researchers and the security experts in analyzing android applications for possible data leaks.It is considered as the defacto technique for analysis, as the application cannot modify its behaviour at the testing table as considered to dynamic analysis.The main goal of static analysis is to detect possible leaks and minimise the false positives,maximizing true postives,avoiding false negatives and maximizing true negatives and regardless the approach of static analysis whether it is taint analysis,context sensentive analysis,data flow analysis,information flow analysis,point to point analysis the objectives are the same;detect possible leaks without missing any at a considerable cost of computation ,Preciseness,Accuracy and cost of computations are considered, lastly complexity.

Static analysis faces challenges that are as a result of the nature of android /java language that makes generated call graphs imprecise or with missing paths.The implementation of OOP principles like polymorphism,inheritance,multhreading ,virtual calls and reflection has become an obstacle derailing the efforts of ensuring static analysis is efficient and effective in determining possible data leakes in android applications and as supported by Li et al,(2016,pg 1) where they state that one must account for android features to ensure both sound and complete static analysis..Reflection allows dynamic code generation and makes it possible to instantiate new objects and invoke methods from the name of the classes or methods and some work has been intiated in solving this by performing string analysis of the string values more can be found Kim et al(2012) thus it is possible for malicious developers to try and exploit this ,to evade detection of malicious implementations that will likely lead to data leakage.Polymorphism.

Java virtual calls means that a method invoked can only be determined by a dynamic type for example

```
              Class Y{
Public void foo(){
}
Class X extends Y{
Public void foo(){
}
}
When invoked Y object=new X()
Object.foo(); X.foo()will be invoked and not Y,considering the dynamic type of obj is X.
```

The aspect of reflection methods in java also makes generated call graphs imprecise having in mind the missed patterns ,reflection is the ability of examining or modifying the run time behavior of a class at run time. Multi-threading also leads to data that was retrieved by the main thread being released to the child thread that will send it via a network or an open socket.
Call graphs are key when it comes to static analysis of programs ,according to Ismail pg 1(2009) call graphs are powerful tools for program analysis as they help in understanding calling relationships between program methods. They link call sites to target methods, in this case all possible calls of a program that lead to multiple target methods. This results to overapproximation and with reference to android data leakages this is not good news considering the many false alarms and the ability to handle large application or even the libraries.This is a motivating factor for malicious developers who are determined to make sure that their malicious applications evade detection .

Call graph algorithms are considered conservative as they are overestimating the possible calls from a call site. The advancement of these algorithms is purely based on how precise the call graph can be thus the same time increasing accuracy at the cost of computing resources and complexity. Mainly the reduced number of reachable methods has been observed these has been effected through use of sets and the more the sets the less the number of reachable, improved accuracy and more precise but a question can be posed with regard to static analysis of android applications for possible data leaks is there a possibility that a set may not include sensitive methods or call sites that can results to leakage of users data?.

Open source tools for constructing call graphs; have been developed to support various static analysis techniques for program optimization, quality code assurance and assessing security and compliance. Selecting a tool to use is based on the type of analysis you want to do, the possible inputs and outputs in this case we have android bytecode or java bytecodes, file extension either apk,jar or java class. Conversion form one file extension to another may lead to a distorted file thus the importance of using a tool that supports the current file extension and lastly prior knowledge of the tools and static knowledge is also a major factor considering that most frameworks will require the researcher to set the scope of analysis and even create a custom entry main method. This will affect the results that will be generated by this frameworks and it poses a great danger if it results to undetected data leaks. Support community and updates are of importance when selecting a tool to use as they will capture recent developments.

For the computed summaries of android Bluetooth Library, connect () creates a connection with a remote device/opens sockets thus creating a link for both the bluetoothSocket and the bluetoothSocket Server and calls getInputStream () and getOutputSream () that opens the input and output streams and according to Pandey and Khare(2014) a device with its Bluetooth turned on or is discoverable it may be vulnerable to Bluejacking and Bluesnarfing if there is a vulnerability in the vendor’s software.

getInputStream () gets input stream for the associated socket can be classified as a source that allows injection of data streams or in this case users’ data and according to Gordon et al, (2015, pg 2) in their newly developed sensitive information flow system they classified API functions that get users data or takes inputs as sources that end up sending the information they get to sinks that are considered as exits of the obtained data to external servers in this case to attackers.

getOutputStream() get the output stream associated with this socket (Sink or exit) in Gordon et al,(2015) defines sinks as API call that may leak information or allow information to exit the device.

getRemoteDevice() device it’s connected to,

```
    BluetoothDevice                                                       device
=mBluetoothAdapter.getRemoteDevice(address);

       BluetoothSocket tmp =null;

       BluetoothSocket mmSocket =null;

//get bluetooth socket for a connection with a given bluetoothdevice

          try{


tmp=device.createRfcommSocketToServiceRecord(MY_UUID);

   Method    m=    device.getClass().getMethod("createRfcommSocket",new
Class[]{int.class});

           Tmp=(BluetoothSocket)m.invoke(device,1);

}catch(IOException e){

Log.e(TAG,"create() failed",e);

}

mmSocket =tmp;
```

This is an extract from a simple open source code android app by janosgyerik (2016)

The possible implementation of getRemoteDevice where the MAC address of a remote device is obtained through device discovery or obtained from the bonded devices through getBondedDevice().Next step the socket for communication is opened using createRfcommsocket and all this requires require the application to have bluetooth permissions but according to Zhou and Jiang (2013,pg 3) a malicious application may not request any permissions to access the bluetooth services but can be able to access the MAC address of a device through other vulnerable application. The malicious application that has internet permissions will send the MAC address to a remote server of the attacker who will use the MAC address to launch a bluetooth attack to steal users data without their knowledge.

Titze,Stephanow and Schuette,(2013,pg 4) states that despite the security model implemented by android there exists a security breach that springs from programming errors caused by inexperienced developers that leads to vulnerable applications that may end up leaking users data or being exploited by malicious application in this case by obtaining the MAC address of the device and sending it to an external server

If an applications has access to internet ,Bluetooth , Bluetooth_Admin and has permission to write and read from external storage that is considered insecure. This means that it is possible of an application to request for MAC address of the Bluetooth adapter through the getAddress() method and either deposit it to the external storage or to a server online. Also the getInputStream () and getOutputStream() that can get data from external storage and send it via the Bluetooth as it makes a call to connect() which opens up the network sockets that will allow data leave the device. And lastly it can send data via the internet(), for the sinks append(),toString() were considered in this case. To exhaustively vet the applications some other API methods of various classes were considered like

The generated call graphs model the calling relationship within the android Bluetooth Library 2.1, it represents calls made from call sites and the possible calls that are reachable from the call site, we are actually able to get the code that is called when a method is invoked and with all this information we are able to determine what is returned when a method is invoked .In static analysis for android data leakages or possible data leakages methods are analyzed based on their ability to have access to sensitive users data and the possibility that same data can exit the device using another method or the possible misuse of a function or method and lastly the sensitivity of the function in regard to users data.

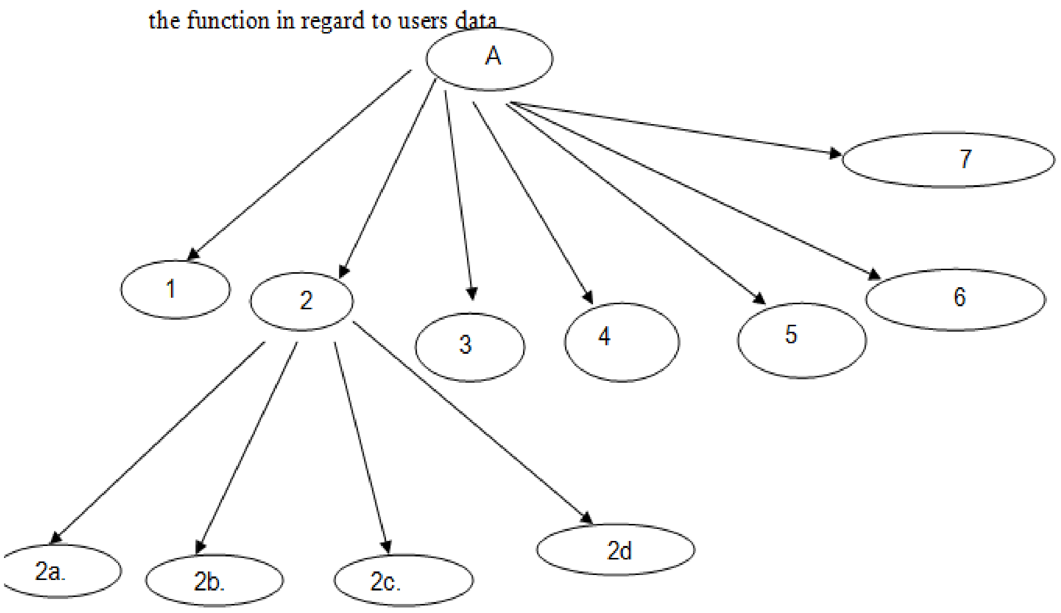


Figure 12: Reachable methods from call site

A=Call site OnReceive() Method

1=getAction 2.processDiscoveryStarted() 3.equals() 4.processRemoteDeviceFound() 5.processDiscoveryCompleted() 6.processPairingRequested() 7.processBondStateChanged()

2a.Println() printing next line 2b.access3() 2c.access4().scanStarted().

{1,2,3,4,5,6,7,A }∈ Reachables and {2a,2b,2c,2d,2} ∈ Reachables

Classes implement interfaces meaning that a class has to implement all the methods belonging to the interface unless its declared abstract.In this case we have private class BluetoothSocketImpl implements BluetoothSocket thus BluetoothSockets methods are possible candidates for implementation by bluetoothsocketImpl the same thing with private class RemoteBluetoothDeviceImpl implements RemoteBluetoothDevice.Having successfully obtained the possible implementers of a class we are able to review all the methods with an aim of being able to identify any of them that can be used maliciously or its implementation can lead to user's private data being compromised or being sent out of the device via the Bluetooth device.

Lastly in supporting this study, findings and hypothesis we considered what was done by Ali and Lhotak (2012) which this study was partly extension of what they did. In their study as captured in chapter one and two they produced a partial call graph that soundly over approximated the set of targets of every call site during static analysis scope and a set of reachable functions in the analysis scope. They produced a node of the libraries and avoided analyzing them. They based their study on the separate compilation assumption from which they deduced specific restrictions on how the library interacts with the application using it. The inability of the library calling a method, accessing a field or instantiating a class of an application of which the library author has no knowledge of the method, field or class ,having in mind the library can be compiled without having knowledge of the application.

These supports the study’s argument that it is possible to analyze the application separately and compute summaries of possible use without knowledge of the application that will use it. In their efforts to ensure they generate a sound call graph which from Chapter one and two the computation cost has to be considered, accuracy and complexity and this informed their decision to moderate the library aspect and a void the whole program call graph which is considered expensive and armed with this in mind .The study further moderates the aspect of the library by computing summaries based on answering which code is actual called when a method is invoked and classifying them according to sensinsitive nature by finding out which classes are implemented and the possibility of having any of their implementation leading to possible data leakage in android applications that implement them. Thus with these summaries and the code that is actually called will improve the preciseness of static analysis without any strain on the cost and complexity because the summaries will be readily available.

Code extract from Ali and Lhotak(2012,pg 692)

```
Public class Main{

Public static void main(){

MyHashMap<String,String>myHashMap=new MyHashMap<String,String>();

System.out.println(myHashMap);

}

}
```

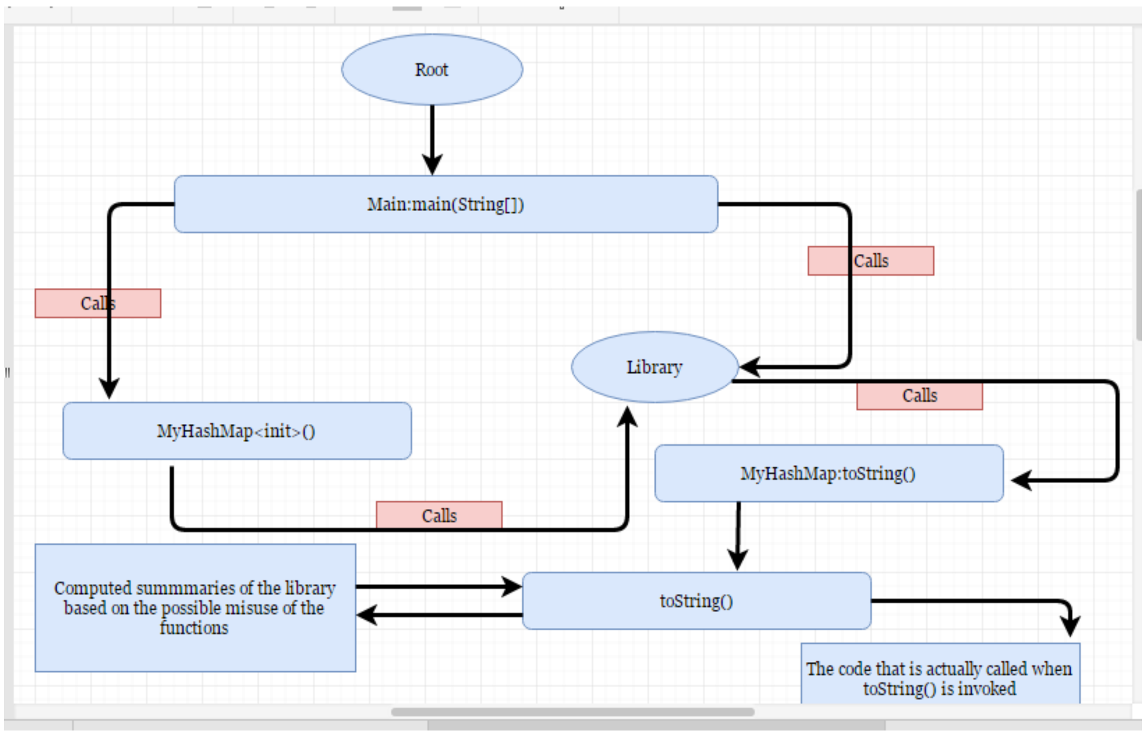

Root
Main:main(String[])
Calls
Calls
Library
Calls
MyHashMap<init>()
MyHashMap:toString()
Calls
Computed summmaries of the library based on the possible misuse of the functions
toString()
The code that is actually called when toString() is invoked

Figure 13: Analysis of target Applications Using Computed Summaries.

4.6 Proposed Android Data Leakages Mitigation Conceptual Framework in Static Analysis

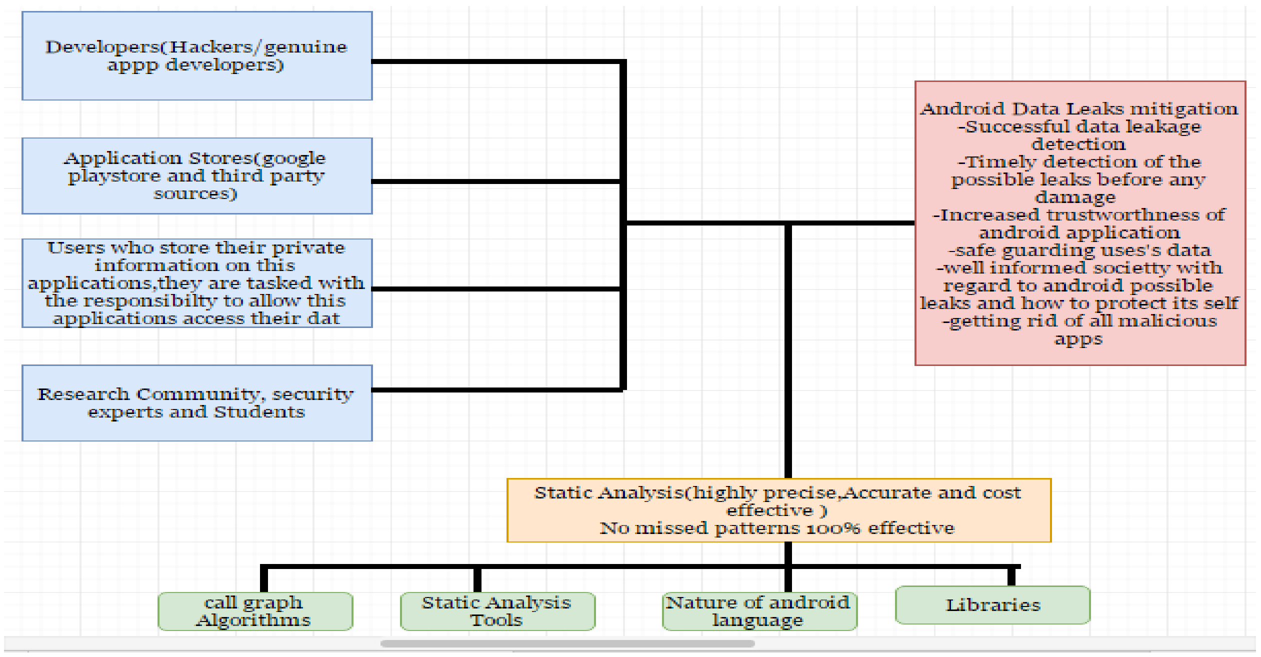

Developers(Hackers/genuine appp developers)
Application Stores(google playstore and third party sources)
Users who store their private information on this applications,they are tasked with the responsibilty to allow this applications access their dat
Research Community, security experts and Students
Android Data Leaks mitigation -Successful data leakage detection -Timely detection of the possible leaks before any damage -Increased trustworthness of android application -safe guarding uses's data -well informed societty with regard to android possible leaks and how to protect its self -getting rid of all malicious apps
Static Analysis(highly precise,Accurate and cost effective) No missed patterns 100% effective
call graph Algorithms
Static Analysis Tools
Nature of android language
Libraries

Figure 14: Proposed Android Data Leakages Mitigation Conceptual Framework in Static Analysis

Lastly the study proposes and validates the below conceptual framework for future studies and testing in android data leakage studies

4.6.1 Independent Variables

4.6.1.1 Developers(Genuine developers and malicious developers)

The developers of android applications have a responsibility of protecting the users from either intended or unintended data leaks. It is evident from literature that developers are contributors to data leaks in android applications. According to Bartel et al.(2014) developers often over estimates the required permissions by adding many permissions which an application does not need, these permissions allow the application to access users private data thus exposing the users to malicious applications that will exploit the permissions, they confirm the possibility of injected malware to use these declared permissions for malicious goals. Developers using advertisment libraries without full knowledge what risk they pose to the users,developers allowing applications write to unsecured storage allowing that data to be fetched by malicious applications that leak them as supported by Felt et al.(2011).

4.6.1.2 Application Stores(google play store and third party markets).

Once the developers are done with developing the applications the next stop for the application is application stores like google application online market and third party markets. Having in mind the challenges posed by the developers in android data leaks mitigation it’s the responsibility of the administrators of these online stores to protect the users from these applications that have the potential of leaking their data. It is evident from literature that despite having a centralized distribution of applications there is a threat posed by other unofficial application markets that distribute same genuine and malicious applications that have no review of the applications thus posing a threat to the users as supported by Shaerpour et al.(2016).According to Schmidt et al.(2009) as cited by Shaerpour et al.(2015) there is a practice of repackaging malware free applications acquired from google play store and uploading them to third party markets which poses a threat to android data leaks mitigations where malware can be attached to the genuine applications by planting loopholes or hide malicious payloads.

4.6.1.3 Android Smartphone Users

The smartphone users have a bigger task in ensuring that they do not expose their private data for possible leaks by ensuring that they install applications from trusted markets and they accept or reject applications they install by reviewing the permissions they request for. Having in mind the challenges posed by the developers and android markets. Evident from literature shows a threat posed by the user to android data leak mitigation, according to Shaerpour et al.(2016) users are likely to install free applications instead of purchasing and most of the time these free applications access users private data and sends it to advertisement firms for financial gains ,despite the permissions security mechanism in android the users download malicious applications as they have little knowledge of how much or to what extent a particular permission exposes their personal data and most of them ignore the permissions altogether. Felt et al.(2011) agrees that users are unable to evaluate the requests and have little choice in regard to which specific permissions of an app they grant and discard others as the only option is either to agree to all or reject installation of the application. Thus escalating the issue of data leaks in android applications.

4.6.1.4 Research community and security experts

The research community and security experts in android data leak mitigation field of study have the responsibility to ensure zero data leak as they are considered as the" saviors" when it comes to leaks .There failure will led to unimaginable damage. Bartel et al. (2014) there is need to address issues to do with permission gaps in android permissions security mechanisms, Li et al. (2015).Xia et al.(2015) need for applications audits, Need to explore the information flows in android applications as supported by Gordon et al.(2015) to discover all potential sensitive flows.

4.7.0 Moderating Variable

4.7.1 Static Analysis

When developers assign permissions to applications, it has been reported in Bartel et al. (2014) that developers assign more permissions than actually needed permissions which end up being exploited by maliciously to leak uses private data. There is need for a precise mapping of API functions and the permissions they require and this is achieved through advanced class hierarchy and field-sensitive set of analysis that extract this mapping achieved through precise static analysis Bartel et al. (2014).

In curbing sensitive information leaks in android applications that are as a result of malicious or poor coding that led to misuse of advertisement libraries by the developers poses a major threat or risk to android ecosystem as supported by Gordon et al. (2015) where they addressed these problem by coming up with Droid safe which is a static analysis information flow tool that analyses information flows in an application to detect potential sensitive information leaks

In dealing with problems that come along with third party applications downloaded from third party online stores, where it has been reported of no vetting policies of applications they allow on their store thus exposing the users to potential leaks of their private data considering the habit of users downloading free applications and disregard of permissions requested by these applications. In solving these problem Kim et al. (2012) developed SCANDAL static analyzer that detects privacy leaks in android applications by determining if there is any flow of data from a source to a sink.

CHAPTER 5

5.1 Introduction

Findings of this study with regard to the objectives and research questions are summarized and conclusions generalized based on the findings of the study as presented. The strengths and limitations of this study are considered and suggestions for further studies are presented. The chapter concludes with recommendations to the research community, android based device users, android play store administrators, managers and lastly security experts .

5.2 Conclusion

5.2.1 Objective 1: To investigate the current status and advancements in static analysis for android data leaks.

The findings in this study paint a picture of a determined research community and security experts in solving android data leakages and in this case spending time in improving static analysis techniques. They all acknowledge that static analysis is the de-facto technique in understanding a program. Any malicious, misuse, coding errors or vulnerabilities in android applications that lead to data leakage can be detected before an application is released or after release and it is not easy to evade detection as compared to dynamic analysis. They also acknowledge the role of call graphs, which are considered as starting points in static analysis. Efforts have been made in improving the generated call graphs, through improved algorithms that leads to precise call graphs that can capture all program patterns and at the same time considering computation cost. Same time the advent of open source tool to support static analysis is also reported and much has been achieved considering soot framework that supports android byte code and other tools for android static analysis tool. The role of libraries that android applications interact with is very clear from the findings and it has been accepted that their effect when it comes to static analysis of android applications for possible data leaks cannot be ignored. The possibility of misuse of the libraries by malicious developers and genuine developers either knowingly or unknowingly has been acknowledged and need to mitigate their effect is an agreed fact among the research community and security experts.

5.2.2 Objective 2: **Static analysis of android libraries and computing summaries that were to be used as black boxes when analyzing target applications.**

From the generated call graph of the android Bluetooth 2.1 library it is possible to get the code that is actually called, when a method is invoked and from each call site it is possible to get all reachable methods and their code respectively. In this case if an application makes a call to the library and the call makes a call within the library where the other call makes a call back to the application without knowledge of the method that was called first in what is referred to as call backs .It is possible to see this pattern thus solving the missing pattern of a call graph and callback challenge.

It is possible to know all the methods that are likely to be implemented within the library or the application through the provided APIs and considering that it's through them the developer can interact with the library. The API analysis does not have callbacks that are made within the Library thus making Library analysis superior as compared to API analysis. Lastly the possibility of computing each of the possible methods with regard to the ability of their implementations leading to data leaks whether as sources or sinks and using them to analyze target application without knowledge of the applications that will use them later.

5.2.3 Objective 3: **Analyzing target applications with the computed summaries in the context of misuse of the analyzed library for data leaks.**

For all the applications that that were analyzed none of them leaks address (MAC address) of the Bluetooth adapter and none of the functions fetched data from the external storage and sent it through Bluetooth or to external servers. The analysis was successful and it is a confirmation that it's possible to use computed summaries and also what comes out clear is a role that is played by human , results have to be interpreted by the security analyst, researcher and applications market store administrators. Lastly from the findings it's clear that address (MAC address) and data leaks via Bluetooth are not common but still there is possibility of it being exploited mostly in fitness applications like Fitbit and others

5.2.4 Hypothesis

A complete analysis and computation of libraries implemented by android application can lead to a highly precise static analysis without knowledge of the code that will use the library later.

The findings report the possibility of being able to perform a complete analysis of an android libraries without having knowledge of the code or program that will use it later that will lead to precise static analysis considering that it's possible to extract what code is called when a method is invoked, possible implementations of classes and lastly computation summaries according to their sensitivity .In conclusion preciseness of static analysis is a continuous process and we are not yet there.

5.3 Recommendations.

The following recommendations

- Considering the importance of a precise static analysis in android data leakages and the role of computed summaries of the libraries that are used by this applications ,the study suggests more summaries of other libraries to be computed then validated by experimental studies with target applications and compared with other techniques.
- From the findings soot framework is rated as one of the most popular open source tool used in static analysis of android applications for possible data leakages, the study suggest a repeat of this study using soot framework and comparing the findings with this study.
- Considering the critical role played by call graph generation algorithms, the study suggest the possible evaluation of the current algorithms on various existing tools and frameworks in analyzing large android libraries.
- From the findings it is very clear that the nature of android language is key in coming up with a precise static analysis of android applications and libraries .There is need to conduct first a systematic literature review of what has been achieved so far and later a focus on what has not been done .
- Considering the current advancements in call graph construction algorithms where the number of sets used is directly propositional to the number of reachable methods. There is need to determine if there is a possibility of these sets leaving out some methods ,which will lead to possible data leakage in an application to evade detection during

static analysis. Explore also the possibility of a malicious developer with the knowledge of these sets trying to exploit this.

- The future studies should also explore the possibility of attackers using native code to evade detection of malicious application from being detected during static analysis through experimental studies.
- Future studies should explore to studies that compare android libraries analysis versus API analysis
- Future studies to implement and test the proposed conceptual framework.

6.0 Reference

Posted and Paul, J. (2013) *How clone method works in java?* Available at: http://javarevisited.blogspot.co.ke/2013/09/how-clone-method-works-in-java.html (Accessed: 23 May 2016). SpecificationTheJava™Language (2016) *Class object*. Available at: https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html (Accessed: 23 May 2016)**.**

Bluetooth Socket (no date) Available at: https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#TYPE_RFCOMM (Accessed: 23 May 2016). *BluetoothDevice* (no date) Available at: https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html (Accessed: 23 May 2016)**.** Motorola (2001) *RemoteDevice (bluecove 2.1.0 API)*. Available at: http://bluecove.org/bluecove/apidocs/javax/bluetooth/RemoteDevice.html (Accessed: 23 May 2016)**.**

Sable.github.io. (2016). *A framework for analyzing and transforming Java and Android Applications*. [Online] Available at: https://sable.github.io/soot/ [Accessed 3 Jun. 2016].

Bartel, A., Klein, J., Monperrus, M. and Le Traon, Y. (2014) 'Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges and Solutions for Analyzing Android', *IEEE Transactions on Software Engineering*, 40(6), pp. 617–632. doi: 10.1109/tse.2014.2322867.

Gibler, C., Crussell, J., Erickson, J. and Chen, H. (2012) 'AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale', in *Lecture Notes in Computer Science*. Springer Science + Business Media, pp. 291–307.

Mann, C. and Starostin, A. (2012) 'A framework for static detection of privacy leaks in android applications', *Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12*, . doi: 10.1145/2245276.2232009.

Zhang, P.H., Li, J.Z., Shao, S. and Wang, P. (2014) 'PDroid: Detecting Privacy Leakage on Android', *Applied Mechanics and Materials*, 556-562, pp. 2658–2662. doi: 10.4028/www.scientific.net/amm.556-562.2658.

Yan, M., Mu, Y., He, Y. and Liu, A. (2014). The Analysis of Function Calling Path in Java Based on Soot. *AMM*, 568-570, pp.1479-1487.

Rayside, D., Reuss, S., Hedges, E. and Kontogiannis, K., 2000. The effect of call graph construction algorithms for object-oriented programs on automatic clustering. In *Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on* (pp. 191-200). IEEE.

Babu, R., Abraham, G. and Borasia, K., 2013. KRAB Algorithm-A Revised Algorithm for Incremental Call Graph Generation. *arXiv preprint arXiv:1303.0908*.

Grove, D. and Chambers, C., 2001. A framework for call graph construction algorithms. ACM Transactions on Programming Languages and Systems (TOPLAS), 23(6), pp.685-746.

Mangal, R., Naik, M. and Yang, H., 2014. A correspondence between two approaches to interprocedural analysis in the presence of join. In Programming Languages and Systems (pp. 513-533). Springer Berlin Heidelberg.

Honar, E. and Mortazavi Jahromi, S.A., 2010. A Framework for Call Graph Construction.

Lam, P., Bodden, E., Lhoták, O. and Hendren, L., 2011, October. The Soot framework for Java program analysis: a retrospective. In Cetus Users and Compiler Infastructure Workshop (CETUS 2011).

Grove, D., DeFouw, G., Dean, J. and Chambers, C., 1997. Call graph construction in object-oriented languages. ACM SIGPLAN Notices, 32(10), pp.108-124.

Milanova, A., Rountev, A. and Ryder, B.G., 2002. Precise Call Graph Construction in thePresence of FunctionPointers. P roceed ing so f th e 2n d In ter n atio n a l W or k s h o p o nS o u rce C odeA n a lys is a n d M a nipulatio n, Montreal, Canada, O ct.

Sawin, J. and Rountev, A., 2011, September. Assumption hierarchy for a CHA call graph construction algorithm. In Source Code Analysis and Manipulation (SCAM), 2011 11th IEEE International Working Conference on (pp. 35-44). IEEE.

Zhang, W. and Ryder, B., 2006, September. Constructing accurate application call graphs for Java to model library callbacks. In Source Code Analysis and Manipulation, 2006. SCAM'06. Sixth IEEE International Workshop on (pp. 63-74). IEEE.

Enck, W., Octeau, D., McDaniel, P. and Chaudhuri, S., 2011, August. A Study of Android Application Security. In USENIX security symposium (Vol. 2, p. 2).

Rastogi, V., Chen, Y. and Enck, W., 2013, February. AppsPlayground: automatic security analysis of smartphone applications. In Proceedings of the third ACM conference on Data and application security and privacy (pp. 209-220). ACM.

Liang, S., 2014. Static analysis of Android applications (Doctoral dissertation, The University of Utah).

Grishchenko, I., Maffei, M. and Hammer, C., 2014. Static Analysis of Android Applications (Doctoral dissertation, Universität des Saarlandes Saarbrücken).

Zhao, D., Miao, L. and Zhang, D., 2015. Reusable Function Discovery by Call-Graph Analysis. Journal of Software Engineering and Applications, 8(4), p.184.

Sharir, M. and Pnueli, A., 1978. Two approaches to interprocedural data flow analysis. New York University. Courant Institute of Mathematical Sciences. ComputerScience Department.

Neuner, S., Van der Veen, V., Lindorfer, M., Huber, M., Merzdovnik, G., Mulazzani, M. and Weippl, E., 2014. Enter sandbox: Android sandbox comparison. arXiv preprint arXiv:1410.7749.

Li, L., Bartel, A., Klein, J. and Le Traon, Y., 2014. Detecting privacy leaks in Android Apps.

Rastogi, V., Chen, Y. and Jiang, X., 2014. Catch me if you can: Evaluating android anti-malware against transformation attacks. Information Forensics and Security, IEEE Transactions on, 9(1), pp.99-108.

Apvrille, A. and Strazzere, T., 2012. Reducing the window of opportunity for Android malware Gotta catch’em all. Journal in Computer Virology, 8(1-2), pp.61-71.

Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., Wang, X.S. and Zang, B., 2013, November. Vetting undesirable behaviors in android apps with permission use analysis. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 611-622). ACM.

Egele, M., Brumley, D., Fratantonio, Y. and Kruegel, C., 2013, November. An empirical study of cryptographic misuse in android applications. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 73-84). ACM.

Zhang, L., Niu, Y., Wu, X., Wang, Z. and Xue, Y., 2013. A3: Automatic Analysis of Android Malware. In International Workshop on Cloud Computing and Information Security.

Dua, L. and Bansal, D., 2014. TAXONOMY: MOBILE MALWARE THREATS AND DETECTION TECHNIQUES. International Journal of Computer Science & Information Technology, 6.

Shaerpour, K., Dehghantanha, A. and Mahmod, R., 2013. Trends in android malware detection. The Journal of Digital Forensics, Security and Law: JDFSL, 8(3), p.21.

Kim, J., Yoon, Y., Yi, K., Shin, J. and Center, S.W.R.D., 2012. ScanDal: Static analyzer for detecting privacy leaks in android applications. MoST, 12.

Enck, W., Octeau, D., McDaniel, P. and Chaudhuri, S., 2011, August. A Study of Android Application Security. In USENIX security symposium (Vol. 2, p. 2).

Gordon, M.I., Kim, D., Perkins, J.H., Gilham, L., Nguyen, N. and Rinard, M.C., 2015. Information Flow Analysis of Android Applications in DroidSafe. In NDSS.

janosgyerik (2016) *Janosgyerik/bluetoothviewer*. Available at: https://github.com/janosgyerik/bluetoothviewer (Accessed: 13 June 2016).

Smaragdakis, Y., Balatsouras, G., Kastrinis, G. and Bravenboer, M., 2015. More sound static handling of Java reflection. In Programming Languages and Systems (pp. 485-503). Springer International Publishing.

Li, Y., Tan, T., Sui, Y. and Xue, J., 2014. Self-inferencing reflection resolution for Java. In ECOOP 2014–Object-Oriented Programming (pp. 27-53). Springer Berlin Heidelberg.

Rayside, D., Reuss, S., Hedges, E. and Kontogiannis, K., 2000. The effect of call graph construction algorithms for object-oriented programs on automatic clustering. In Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on (pp. 191-200). IEEE.

Babu, R., Abraham, G. and Borasia, K., 2013. KRAB Algorithm-A Revised Algorithm for Incremental Call Graph Generation. arXiv preprint arXiv:1303.0908.

Grove, D. and Chambers, C., 2001. A framework for call graph construction algorithms. ACM Transactions on Programming Languages and Systems (TOPLAS), 23(6), pp.685-746.

Kim, J., Yoon, Y., Yi, K., Shin, J. and Center, S.W.R.D., 2012. ScanDal: Static analyzer for detecting privacy leaks in android applications. MoST, 12.

Ismail, U., 2009. Incremental call graph construction for the Eclipse IDE. University of Waterloo Technical Report.

Bauer, V. (2014) A categorized directory of free libraries and tools for Android. Available at: https://android-arsenal.com/tag/94 (Accessed: 3 June 2016).

Bhat, S.A. (2012) ‘A practical and comparative study of call graph construction Algorithms’, IOSR Journal of Computer Engineering, 1(4), pp. 14–26. doi: 10.9790/0661-0141426.

BluetoothSocket (no date) Available at: https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html (Accessed: 8 June 2016).

document, B.S., Inc (no date) Bluetooth development portal. Available at: https://developer.bluetooth.org/TechnologyOverview/Pages/GATT.aspx (Accessed: 8 June 2016).

Doop / framework for java pointer analysis (2016) Available at: http://doop.program-analysis.org/ (Accessed: 3 June 2016).

gousiosg (2013) Gousiosg/java-callgraph. Available at: https://github.com/gousiosg/java-callgraph/blob/master/README.md (Accessed: 3 June 2016).

Deshpande, S. and Dharmadhikari, S.C., 2016. Analysis on Camera Attacks and their Defenses on Android Smartphones. European Journal of Advances in Engineering and Technology, 3(3), pp.26-29.

Tip, F. and Palsberg, J., 2000. *Scalable propagation-based call graph construction algorithms* (Vol. 35, No. 10, pp. 281-293). ACM.

Gordon, M.I., Kim, D., Perkins, J.H., Gilham, L., Nguyen, N. and Rinard, M.C., 2015. Information Flow Analysis of Android Applications in DroidSafe. In *NDSS*.

Enck, W., Octeau, D., McDaniel, P. and Chaudhuri, S., 2011, August. A Study of Android Application Security. In *USENIX security symposium* (Vol. 2, p. 2).

Ali, K. and Lhoták, O., 2013, July. Averroes: Whole-program analysis without the whole program. In *European Conference on Object-Oriented Programming* (pp. 378-400). Springer Berlin Heidelberg

Ali, K. and Lhoták, O., 2012, June. Application-only call graph construction. In *European Conference on Object-Oriented Programming* (pp. 688-712). Springer Berlin Heidelberg.

Yan, D., Xu, G. and Rountev, A., 2012, June. Rethinking soot for summary-based whole-program analysis. In *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis* (pp. 9-14). ACM.

Li, Y., Tan, T., Sui, Y. and Xue, J., 2014, July. Self-inferencing reflection resolution for Java. In *European Conference on Object-Oriented Programming* (pp. 27-53). Springer Berlin Heidelberg.

Barros, P., Just, R., Millstein, S., Vines, P., Dietl, W., d'Amorim, M. and Ernst, M.D., 2015. Static analysis of implicit control flow: Resolving Java reflection and Android intents (extended version). *University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, Tech. Rep. UW-CSE-15-08-01*

Balatsouras, G. and Smaragdakis, Y., 2013, October. Class hierarchy complementation: soundly completing a partial type graph. In *ACM SIGPLAN Notices* (Vol. 48, No. 10, pp. 515-532). ACM.

Rountev, A., Kagan, S. and Marlowe, T., 2006, March. Interprocedural dataflow analysis in the presence of large libraries. In *International Conference on Compiler Construction* (pp. 2-16). Springer Berlin Heidelberg.

Bravenboer, M. and Smaragdakis, Y., 2009. Strictly declarative specification of sophisticated points-to analyses. *ACM SIGPLAN Notices*, *44*(10), pp.243-262.

Bodden, E., Sewe, A., Sinschek, J., Oueslati, H. and Mezini, M., 2011, May. Taming reflection: Aiding static analysis in the presence of reflection and custom class loaders. In *Proceedings of the 33rd International Conference on Software Engineering* (pp. 241-250). ACM.

Android 7.0 Nougat! (no date) Available at: https://developer.android.com/index.html (Accessed: 20 September 2016).

A framework for analyzing and transforming java and Android applications (no date) Available at: https://sable.github.io/soot/ (Accessed: 14 September 2016).

Jiang, Y.Z.X., 2013, February. Detecting passive content leaks and pollution in android applications. In Proceedings of the 20th Network and Distributed System Security Symposium (NDSS).

Parvez, MAD&J 2013, 'Evaluating Smartphone Application Security: A Case Study on Android', *Global Journal of Computer Science and Technology Network, Web & Security*, vol 13, no. 12, pp. 9-15.

Luigi Vigneriy, JCIPAOH 27th april 2015, 'Taming the Android AppStore: Lightweight Characterization of Android Applications', Research Report RR-15-305, Networking and Security department , EURECOM,

Payet, É. and Spoto, F., 2012. Static analysis of Android programs.*Information and Software Technology*, *54*(11), pp.1192-1201.

Shen, T., Zhongyang, Y., Xin, Z., Mao, B. and Huang, H., 2014, September. Detect android malware variants using component based topology graph. In*2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications* (pp. 406-413). IEEE

Gascon, H., Yamaguchi, F., Arp, D. and Rieck, K., 2013, November. Structural detection of android malware using embedded call graphs. In*Proceedings of the 2013 ACM workshop on Artificial intelligence and security*(pp. 45-54). ACM.

Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., Wang, X.S. and Zang, B., 2013, November. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 611-622). ACM.

Enck, W., Octeau, D., McDaniel, P. and Chaudhuri, S., 2011, August. A Study of Android Application Security. In *USENIX security symposium* (Vol. 2, p. 2).

Schmidt, A.D., Bye, R., Schmidt, H.G., Clausen, J., Kiraz, O., Yuksel, K.A., Camtepe, S.A. and Albayrak, S., 2009, June. Static analysis of executables for collaborative malware detection on android. In *2009 IEEE International Conference on Communications* (pp. 1-5). IEEE.

Yang, Z. and Yang, M., 2012, November. Leakminer: Detect information leakage on android with static taint analysis. In *Software Engineering (WCSE), 2012 Third World Congress on* (pp. 101-104). IEEE.

.

Grove, D., DeFouw, G., Dean, J. and Chambers, C., 1997. Call graph construction in object-oriented languages. *ACM SIGPLAN Notices*, *32*(10), pp.108-124.

Grove, David, and Craig Chambers. "A framework for call graph construction algorithms." *ACM Transactions on Programming Languages and Systems (TOPLAS)* 23, no. 6 (2001): 685-746.

Elish, K.O., Yao, D. and Ryder, B.G., 2012, May. User-centric dependence analysis for identifying malicious mobile apps. In *Workshop on Mobile Security Technologies*.

Mahmood, R., Mirzaei, N. and Malek, S., 2014, November. Evodroid: Segmented evolutionary testing of android apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 599-609). ACM.

7.0 Appendix.

RQ3.The codes that are called when methods are invoked in android bluetooth Library?

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:<init>
(M)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:register.

```
public void register(Context context)
        {
                if(!registered)
            {
                Log.d("BluetoothBroadcastReceiver", "Registering");
                IntentFilter intentFilter = new IntentFilter();

intentFilter.addAction("android.bluetooth.intent.action.BLUETOOTH_STATE_CHANG
ED");

intentFilter.addAction("android.bluetooth.intent.action.REMOTE_DEVICE_FOUND")
;

intentFilter.addAction("android.bluetooth.intent.action.DISCOVERY_COMPLETED")
;

intentFilter.addAction("android.bluetooth.intent.action.DISCOVERY_STARTED");

intentFilter.addAction("android.bluetooth.intent.action.PAIRING_REQUEST");

intentFilter.addAction("android.bluetooth.intent.action.BOND_STATE_CHANGED_AC
TION");
                context.registerReceiver(this, intentFilter);
                registered = true;
            } else
            {
                Log.d("BluetoothBroadcastReceiver", "Already registered");
            }
        }
```

True is returned if the device is successfully registered and already registered if the device was previsiously registered.

M:
it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:onReceive
(M)android.content.Intent:getAction.

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:onReceive
(M)java.lang.String:equals.

```
    public boolean equals(Object anObject) {
      if (this == anObject) {
          return true;
      }
      if (anObject instanceof String) {
          String anotherString = (String)anObject;
          int n = count;
          if (n == anotherString.count) {
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = offset;
            int j = anotherString.offset;
            while (n-- != 0) {
                if (v1[i++] != v2[j++])
                  return false;
            }
            return true;
          }
      }
      return false;
    }
```

This method is invoked by onReceive method in class BluetoothBroadcastReceiver and it returns true or false based on

```
if (action.equals(AndroidBluetoothConstants.DISCOVERY_STARTED_ACTION)) {
                              processDiscoveryStarted();
                      }                        else                           if
(action.equals(AndroidBluetoothConstants.REMOTE_DEVICE_FOUND_ACTION)) {
                              processRemoteDeviceFound(intent);
                      }                        else                           if
(action.equals(AndroidBluetoothConstants.DISCOVERY_COMPLETED_ACTION)) {
                              processDiscoveryCompleted();
                      }                        else                           if
(action.equals(AndroidBluetoothConstants.PAIRING_REQUEST_ACTION)) {
                              processPairingRequested(intent);
                      }                        else                           if
(action.equals(AndroidBluetoothConstants.BOND_STATE_CHANGED_ACTION)) {
                              processBondStateChanged(intent);
                      }                        else                           if
(action.equals(AndroidBluetoothConstants.BLUETOOTH_STATE_CHANGED_ACTION)) {
                              processBluetoothStateChanged(intent);
                      }                        else                           if
(action.equals(AndroidBluetoothConstants.REMOTE_NAME_UPDATED_ACTION)) {
```

```
                                processRemoteNameUpdated(intent);
                        }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:onReceive (O)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processDiscoveryStarted.

```
private void processDiscoveryStarted()
        {
            System.out.println("Discovery started");
            LocalBluetoothDevice._localDevice.listener.scanStarted();
        }
```

Scanning for the remote device to connect to by the LocalBluetoothDevice and it's invoked by onReceive.

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:onReceive (O)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processRemoteDeviceFound.

```
private void processRemoteDeviceFound(Intent intent)
        {
            String                          address                           =
intent.getStringExtra("android.bluetooth.intent.ADDRESS");
            LocalBluetoothDevice._localDevice.devices.add(address);
        }
```

The processRemoteDeviceFound(Intent intent) invoked by the onReceive() method gets the remote device's address and adds it to the localdevice list of address of registered devices.

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:onReceive (O)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processDiscoveryCompleted

```
 private void processDiscoveryCompleted()
        {
            if(LocalBluetoothDevice._localDevice.listener != null)
            {

LocalBluetoothDevice._localDevice.listener.scanCompleted(LocalBluetoothDevice
._localDevice.devices);
            }
        }
```

ProcessDiscoveryCompleted() invoked by onReceive method checks whether there is a device to be registered during scanning through the LocalDevice.listener,if it's null it calls scanCompleted(LocalBluetoothDevice._localDevice.devices)

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:onReceive (O)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processPairingRequested.

```
private void processPairingRequested(Intent intent)
        {
            String                          address                          =
intent.getStringExtra("android.bluetooth.intent.ADDRESS");
            Log.d("LocalBluetoothDevice",      (new      StringBuilder("Pairing
requested for ")).append(address).toString());
            RemoteBluetoothDeviceImpl          remoteBluetoothDevice          =
(RemoteBluetoothDeviceImpl)LocalBluetoothDevice._localDevice.remoteDevices.ge
t(address);
            if(remoteBluetoothDevice != null)
            {
                remoteBluetoothDevice.notifyPairingRequested();
            }
        }
```

Pairing request is made from a local bluetooth device to a remote device by appending the device address and if the remote device accepts pairing the pairing request notification is sent.

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:onReceive (O)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processBondStateChanged.

```
private void processBondStateChanged(Intent intent)
        {
            String                          address                          =
intent.getStringExtra("android.bluetooth.intent.ADDRESS");
            int                     previousBondState                      =
intent.getIntExtra("android.bluetooth.intent.BOND_PREVIOUS_STATE", -1);
            int                             bondState                      =
intent.getIntExtra("android.bluetooth.intent.BOND_STATE", -1);
            Log.d("BluetoothBroadcastReceiver",                            (new
StringBuilder("processBondStateChanged()              for              device
")).append(address).toString());
        }
```

These method is called to bond a previously bonded device by checking the previous bonded status and the current bond state.It then appends the address of the paired device.

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:onReceive (O)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processBluetoothStateChanged

```
private void processBluetoothStateChanged(Intent intent)
        {
            Int                    previousBluetoothState                    =
intent.getIntExtra("android.bluetooth.intent.BLUETOOTH_PREVIOUS_STATE", -1);
            int                        bluetoothState                        =
intent.getIntExtra("android.bluetooth.intent.BLUETOOTH_STATE", -1);
            Log.d("BluetoothBroadcastReceiver",                          (new
StringBuilder("processBluetoothStateChanged():
")).append(bluetoothState).toString());
            if(LocalBluetoothDevice._localDevice.listener != null)
            {
                switch(bluetoothState)
                {
                case 2: // '\002'
                    LocalBluetoothDevice._localDevice.listener.enabled();
                    break;

                case 0: // '\0'
                    LocalBluetoothDevice._localDevice.listener.disabled();
                    break;
                }
            }
        }
```

Checks whether the bluetooth state changes from enabled to disabled by appending the current state of the bluetooth device then its checked using switch through the device listener.

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processDiscoveryStarted (M)java.io.PrintStream:println

```
public void println() {
      newLine();
    }
```

```
Println() is invoked by processDiscovery()  to display a message and print a
newline
System.out.println("Discovery started");
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processDiscoveryStarted (S)it.gerdavax.android.bluetooth.LocalBluetoothDevice:access$3

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processDiscoveryStarted (S)it.gerdavax.android.bluetooth.LocalBluetoothDevice:access$4

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processDiscoveryStarted(I)it.gerdavax.android.bluetooth.LocalBluetoothDeviceListener:scanStarted

```
public abstract void scanStarted(){


} an interface call is invoked.
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processPairingRequested (M)java.lang.StringBuilder:append

```
public StringBuilder() {
      super(16);
    }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processPairingRequested (M)java.util.Hashtable:get

```
public synchronized V get(Object key) {
      Entry tab[] = table;
      int hash = key.hashCode();
      int index = (hash & 0x7FFFFFFF) % tab.length;
      for (Entry<K,V> e = tab[index] ; e != null ; e = e.next) {
          if ((e.hash == hash) && e.key.equals(key)) {
            return e.value;
          }
      }
      return null;
    }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processPairingRequested(M)it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl:notifyPairingRequested

```
 void notifyPairingRequested()
    {
        if(listener != null)
        {
            listener.pinRequested();
        }
    }
```

Sends a notification for a pairing request and if there is another device on the other end a pin request is sent

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processRemoteDeviceFound (M)java.util.ArrayList:add

```
public boolean add(E e) {
      ensureCapacity(size + 1);  // Increments modCount!!
      elementData[size++] = e;
      return true;
    }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processDiscoveryCompleted(I)it.gerdavax.android.bluetooth.LocalBluetoothDeviceListener:scanCompleted

```
public abstract void scanCompleted(ArrayList arraylist);
interface call
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processBluetoothStateChanged(I)it.gerdavax.android.bluetooth.LocalBluetoothDeviceListener:enabled

```
public abstract void enabled();
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:processBluetoothStateChanged(I)it.gerdavax.android.bluetooth.LocalBluetoothDeviceListener:disabled

```
public abstract void disabled(); interface call
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothBroadcastReceiver:close (M)java.lang.Exception:printStackTrace.

```
public void printStackTrace() {
        printStackTrace(System.err);

catch(Exception e)
        {
            e.printStackTrace();
        }
    }
```

It prints a throwable and its backtrace to a standard error stream
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:<init>
(M)java.lang.StringBuilder:toString

```
public String  toString() {


        // Create a copy, don't share the array

        return new String(value, 0, count);


    }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:<init>
(M)it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:connect

```
void connect()
            throws Exception
        {
bluetoothSocketClass =Class.forName("android.bluetooth.RfcommSocket");
bluetoothSocketObject = bluetoothSocketClass.newInstance();
Method createMethod = bluetoothSocketClass.getMethod("create", new Class[0]);
  createMethod.invoke(bluetoothSocketObject, new Object[0]);
     Method connectMethod = bluetoothSocketClass.getMethod("connect", new
Class[] {
               java/lang/String, Integer.TYPE
            });
            connectMethod.invoke(bluetoothSocketObject, new Object[] {
                remoteBluetoothDevice.address, Integer.valueOf(port)
            });
        }
```

This method opens a socket link when it returns without throwing an exception

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:connect
(M)java.lang.reflect.Method:invoke

```
 public Object  invoke(Object proxy, Method method, Object[] args)

      throws Throwable;

}

It implements

connectMethod.invoke(bluetoothSocketObject, new Object[] {

                remoteBluetoothDevice.address, Integer.valueOf(port)

            });

        }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:connect
(M)java.lang.Class:getMethod.

```
Method connectMethod = bluetoothSocketClass.getMethod("connect", new Class[]
{
                java/lang/String, Integer.TYPE
            });
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:connect
(S)java.lang.Integer:valueOf

`remoteBluetoothDevice.address, Integer.valueOf(port);`

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:connect (M)java.lang.reflect.Method:invoke

```
  connectMethod.invoke(bluetoothSocketObject, new Object[] {
                remoteBluetoothDevice.address, Integer.valueOf(port)
            });
        }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:getInputStream (M)java.lang.Class:getMethod

```
Method getInputStreamMethod =
bluetoothSocketClass.getMethod("getInputStream", new Class[0]);
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:getInputStream (M)java.lang.reflect.Method:invoke

```
Object returnValue = getInputStreamMethod.invoke(bluetoothSocketObject, new
Object[0]);
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:getOutputStream (M)java.lang.Class:getMethod

```
Method getOutputStreamMethod =
bluetoothSocketClass.getMethod("getOutputStream", new Class[0]);
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$BluetoothSocketImpl:getOutputStream (M)java.lang.reflect.Method:invoke

```
Object returnValue = getOutputStreamMethod.invoke(bluetoothSocketObject, new
Object[0]);
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl:getName (M)it.gerdavax.android.bluetooth.LocalBluetoothDevice:getRemoteName.

```
public String getName()
    {
        if(name == null)
        {
            try
            {
                name = getRemoteName(address);
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }
        }
        return name;
    }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl:getDeviceClass (M)it.gerdavax.android.bluetooth.LocalBluetoothDevice:getRemoteClass

```
public int getRemoteClass(String address)
        throws Exception
    {
        Method getRemoteClassMethod =
bluetoothServiceClass.getMethod("getRemoteClass", new Class[] {
            java/lang/String
        });
        Integer returnValue =
(Integer)getRemoteClassMethod.invoke(bluetoothService, new Object[] {
            address
        });
        return returnValue.intValue();
    }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl:pair (S)java.lang.String:valueOf

```
  Log.d("LocalBluetoothDevice", (new
StringBuilder(String.valueOf(address))).append(" is already
paired").toString());
                listener.paired();
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl:pair (S)it.gerdavax.android.bluetooth.LocalBluetoothDevice:access$1

```
 else
  {
LocalBluetoothDevice.access$1(LocalBluetoothDevice.this, address);

             }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl:isPaired (S)it.gerdavax.android.bluetooth.LocalBluetoothDevice:access$2

```
int bondState = LocalBluetoothDevice.access$2(LocalBluetoothDevice.this,
address)
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl:openSocket (M)java.util.Hashtable:containsKey.

```
        Integer portKey = new Integer(port);
        address socket;
        if(sockets.containsKey(portKey))
        {
            socket = (sockets)sockets.get(portKey);
        } else
        {
            socket = new (LocalBluetoothDevice.this, this, port);
            sockets.put(portKey, socket);
        }
        return socket;
    }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl:openSocket (M)java.util.Hashtable:get

```
socket = (sockets)sockets.get(portKey);
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice$RemoteBluetoothDeviceImpl:openSocket (M)java.util.Hashtable:put

```
sockets.put(portKey, socket);
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:initLocalDevice (M)android.content.Context:getSystemService

```
bluetoothService = context.getSystemService("bluetooth");
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getAddress (M)java.lang.Class:getMethod

```
Method getAddressMethod = bluetoothServiceClass.getMethod("getAddress", new
Class[0]);
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getAddress (M)java.lang.reflect.Method:invoke

```
return getAddressMethod.invoke(bluetoothService, new Object[0]).toString();
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getName (M)java.lang.Class:getMethod

```
    Method getNameMethod = bluetoothServiceClass.getMethod("getName", new
Class[0]);
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getName (M)java.lang.reflect.Method:invoke

```
          return getNameMethod.invoke(bluetoothService, new
Object[0]).toString();
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getManufacturer (M)java.lang.Class:getMethod

```
Method                         getManufacturerMethod                        =
bluetoothServiceClass.getMethod("getManufacturer", new Class[0]);
```

```
return
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getManufacturer (M)java.lan
g.reflect.Method:invoke

```
getManufacturerMethod.invoke(bluetoothService,new Object[0]).toString();
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getCompany (M)java.lang.Obj
ect:toString

```
Method getCompanyMethod = bluetoothServiceClass.getMethod("getCompany", new
Class[0]);
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getCompany (M)java.lang.Cla
ss:getMethod
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getCompany (M)java.lang.ref
lect.Method:invoke

```
return getManufacturerMethod.invoke(bluetoothService, new
Object[0]).toString();
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:scan (M)java.lang.Class:get
Method
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:scan (M)java.lang.reflect.M
ethod:invoke
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:scan (M)java.util.ArrayList
:clear

```
public String getRemoteName(String address)
        throws Exception
    {
        Method getRemoteNameMethod =
bluetoothServiceClass.getMethod("getRemoteName", new Class[] {
            java/lang/String
        });
        return getRemoteNameMethod.invoke(bluetoothService, new Object[] {
            address
        }).toString();
    }
```

M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getRemoteName (M)java.lang.
Class:getMethod
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getRemoteName (M)java.lang.
reflect.Method:invoke
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getRemoteName (M)java.lang.
Object:toString
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getRemoteClass (M)java.lang
.Class:getMethod
M:it.gerdavax.android.bluetooth.LocalBluetoothDevice:getRemoteClass (M)java.lang
.reflect.Method:invoke

```
public String getRemoteName(String address)
        throws Exception
    {
        Method getRemoteNameMethod =
bluetoothServiceClass.getMethod("getRemoteName", new Class[] {
            java/lang/String
        });
        return getRemoteNameMethod.invoke(bluetoothService, new Object[] {
            address
        }).toString();
    }
public RemoteBluetoothDevice getRemoteBluetoothDevice(String address)
    {
        RemoteBluetoothDeviceImpl remoteBluetoothDevice;
        if(remoteDevices.containsKey(address))
        {
            remoteBluetoothDevice =
(RemoteBluetoothDeviceImpl)remoteDevices.get(address);
        } else
        {
            remoteBluetoothDevice = createRemoteBluetoothDevice(address);
            remoteDevices.put(address, remoteBluetoothDevice);
        }
        return remoteBluetoothDevice;
    }
    private RemoteBluetoothDeviceImpl createRemoteBluetoothDevice(String
address, int deviceClass, String rssi)
    { RemoteBluetoothDeviceImpl impl = new RemoteBluetoothDeviceImpl(address,
deviceClass, rssi);
        return impl }
    private RemoteBluetoothDeviceImpl createRemoteBluetoothDevice(String
address)
    {RemoteBluetoothDeviceImpl impl = new RemoteBluetoothDeviceImpl(address);
        return impl;
    }}
```