Computed Summaries of Android Bluetooth Library: Data Leakages Detection

Kevin A. Nyakundi
School of Computing and Informatics
University of Nairobi, Kenya

Elisha Abade
School of Computing and Informatics
University of Nairobi, Kenya

ABSTRACT
Static analysis has been used for assessing android applications for possible data leaks both known and unknown. Due to large size of applications and the libraries that they depend on, it's expensive to perform whole program analysis which leads to either ignoring or making assumptions of the effect of the library that puts into question the soundness of the results. Missed paths are generated that lead to false alarms and missed paths that in return allow possible leaks evade detection. The study computed Android Bluetooth Library 2.1 summaries that were successful used to analyze twenty target applications and no possible data leak was detected. Exploratory approach was used to answer the research questions and lastly java-call graph suite of programs was used to construct a call graph of the library and Dexter android static tool for applications.

General Terms
Android Applications

Keywords
Static Analysis, data leakages, Android Libraries

1. INTRODUCTION
Static analysis has been used to detect possible data leakage in android applications According to [7] static analysis is an attempt to analyze an application before execution for possible data leaks. It involves approximations of the possible behavior of a program.

Despite static analysis being the de-facto technique that can exhaustively examine all data flows and detect possible data leakages in android applications, it generates false alarms and missed alarms due to its over approximation and requires minutes or even hours to examine a real application thus making it difficult to capture all usage patterns, enumerate or yield usable results

A lot of research work has been done with regard to making static analysis efficient and applicable in android data leaks detection. According to [17] due to inherent undecided ability nature of determining code behaviors, any static analysis method must make a tradeoff between computing time and precision of results. In this case a decision has to be made whether to perform a whole program analysis or partial where the effect of the libraries the applications depend on is ignored or assumptions made without analyzing them.

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 [2, 8, 15] produced a sound partial call graph but didn't analyze the library based on separate compilation assumption which left the critical questions on the role of the libraries unanswered. What code 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 attacker .The study answers this questions and evaluates the hypothesis that A complete analysis and computation of library summaries implemented by android application can lead to a highly precise static analysis without knowledge of the code that will use them later. Android Bluetooth Library 2.1 was considered for this study.

2. RELATED WORK.
2.1 Placeholder library
In this study they did build up on their previous work [2] where they generated a partial call graph using CGC tool based on separate compilation assumption [1]. In this study they evaluated the possibility of integrating the separate compilation assumption into whole program analysis frameworks. They presented Averroes tool that generated a placeholder library that replaces the original library that has the constraints that are derived from separate compilation assumption. It assumes that any code of the library that is not analyzed is capable of anything .The placeholder library has three kinds of classes; Referenced library classes, concrete implementation classes and Averroes library class. Averroes performs whole program analysis by generating a sound and a precise call graph without analyzing the library and instead generates a placeholder library that is smaller as compared to the library.

2.2 Separate compilation assumption
In their study they acknowledge the most common approach of building a call graph for a whole program is to ignore all the effects of the library code and all the calls that it makes to the application [2]. 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 tool that generates a sound call graph that overestimates the set of target at each call site in the analysis scope and a set of reachable 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 division between an application and the library it uses is not arbitrary which they also acknowledge that if the analysis scope was a set of classes then 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.

2.3 Unsoundness of android Call graph generation tools analysis

Reviewed the unsoundness of call graphs in android static analysis tools where they define a sound call graph as one with all methods of a client application and how critical it's for accuracy of analysis results and in the case of static analysis of android applications for possible data leaks accurate results are crucial [16] .In this study they proposed and implemented a novel approach that automatically identifies unsoundness, where they evaluated dynamic call graph against a static call graph. They mapped edges of the two graphs and all methods present in a dynamic call graph and missing in static one resulted to it being classified as unsound.

3. METHODOLOGY

The study proposes computation of library summaries and use of the computed summaries to analyze target application. The diagrams below represent the conceptual framework of the solution and experiment set ups.Java-call graph suite of programs was used to analyze the library while Dexter android static analysis tool was used to analyze target applications

3.1 Computation of summaries of android libraries

The goal is to find out which code is called when a method is invoked, the possible implementation of classes, whether there is an implementation that can led to data being sent to hackers or to unsecure storage space where it can be accessed by malicious applications and the possibility of analyzing a library without knowledge of the client application that will use it later. Lastly the evaluation of computed summaries with target application

3.2 Conceptual framework

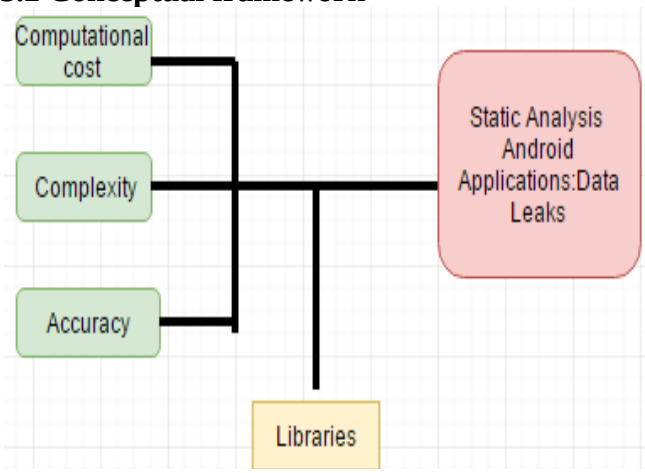

Figure 1.Conceptual Framework

Computational cost, Complexity, Accuracy are independent variables, static analysis is the dependent variable libraries are moderating variable

For this study we moderated Libraries without constraints on cost, complexity and improving accuracy .A more precise static analysis which in the context of android data leakages means a sound analysis of possible data leaks

3.3 Experiment set up

3.3.1 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

3.3.2 Steps of using Dexter

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

4. RESULTS

RQ 1. What code is called when a method is invoked?

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

Is the output from the generated results of call site and reachable methods in this case we have a method(onReceive) from (LocalBluetoothDevice)implementing (BluetoothBroadcastReceiver) invoking method equals of class String from library lang.

When method equals is invoked the following code is called

```
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;
    }
```

It returns true or false

RQ 2.which are the possible implementers of a class are possible candidates?

a).BluetoothSocket:connect() .getInputStream(),getConnectionType(),getInputStream,getRemoteDevice(),getOutputStream(),isConnected(),Close().

b)RemoteBluetoothDevice:getName(),getAddress(),getRSSI() ,getDeviceClass(),pair(String s),isPaired(),setPin(),BluetoothSocket openSocket (int i) throws Exception,setListener(RemoteBluetoothDeviceListener remotebluetoothdeviceListener)

c).RemoteBluetoothDeviceListener: paired () and pinRequested ().

d).LocalBluetoothDeviceListener :enabled(),disabled(),scanStarted(),scanCompleted(ArrayList arrayList)

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

Connect(),getConnectionType(),getOutputstream(),getRemoteBluetoothDevice(),getInputstream(),getName(),getAddress(),getRSSI(),setPin(),pair(Strings),pair(),isEnabled(),getPort(),getManufacturer(),setPin().

RQ6. How applicable are these computed summaries in analyzing target applications?

The applications without Bluetooth permissions are not considered for further analysis based on the computed summaries and in this case getAddress() was considered as it gets MAC address of the adapter and from previous studies its grouped as private users data.

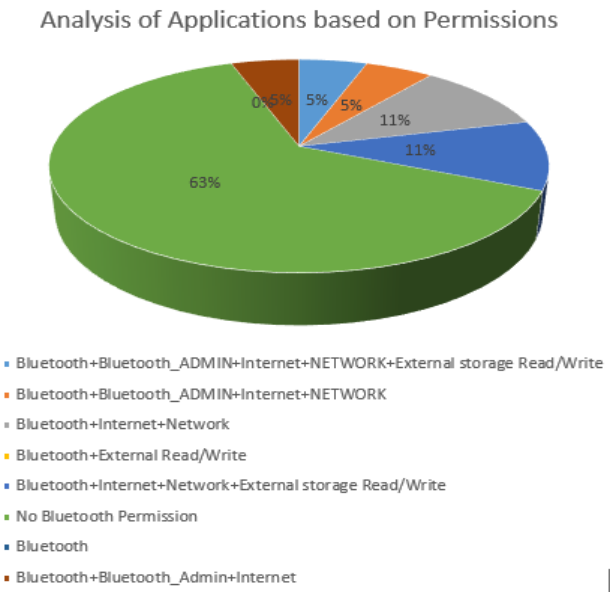

Figure 2.Analysis of applications based on permissions

Table 1.Results of analyzed applications

Number of Applications analyzed	Number of identified Data Leaks	Recommendations
5	0	Further Analysis of Fitbit Application

4.1 Summary of the results

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.

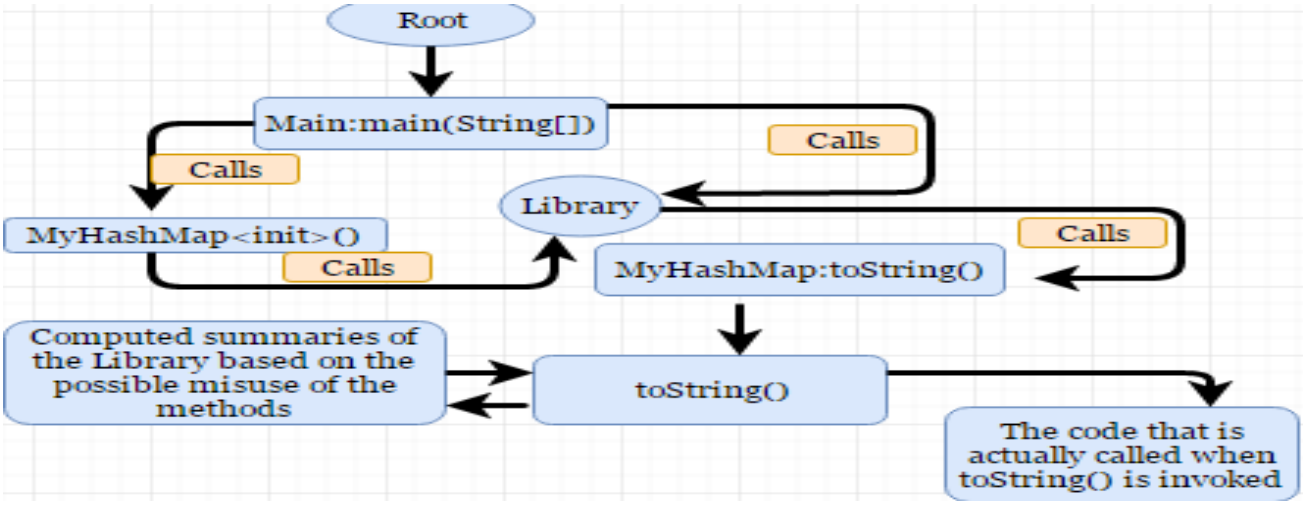

Figure 3. Computed Summaries of Libraries

For the possible candidates of a class, the main source code was considered with the aid of the generated call graph, where interfaces were considered. Class connect implements BluetoothScoket, Class <init> implements remoteBluetoothDevice, private class BluetoothSocketImpl implements BluetoothSocket, private class RemoteBluetoothDeviceImpl implements RemoteBluetoothDevice.

The results of whether there is an implementation that results to user's data being sent to hackers or misuse that will result data being leaked through the external storage where no protection measures are implemented thus can be picked by malicious applications.

In this case implementations were classified into two groups either as sources or sinks as defined in previous studies like [13].

They aided the computation of summaries and also consideration of whether the methods require permissions to access data or open sockets, use Bluetooth, read and write to external storage, internet permissions which are useful to know whether an application sends data to external servers.

Analysis of target applications was done, the first step was to find out whether among the twenty applications that were selected for analysis have Bluetooth permission if not they were not considered for further analysis and an assumption was made that they do not have ability to access Bluetooth either through shared id's or through other applications. After this consideration only 5 applications were analyzed with computed summaries of the library and getAddress() was considered as the private users data that can be possibly be obtained from the users device and sent out without users permission either through internet Urls or to a file in the external storage. The getOutputStream (), getInputStream() were also considered whether the data they obtain is sent out through urls or to the external storage, also whether they read data from the external storage and send it out to external servers by opening sockets through connect( ).From the findings none of the analyzed applications leak data through the misuse of android Bluetooth library though the study recommends further analysis on the Fitbit application considering it reads data to and from the external storage

5. DISCUSSION

In support of this study, findings and hypothesis we considered what was done by [2] which this study is partly an extension of what they did. In their study 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 ,considering that the library can be compiled without knowledge of the application.

This 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 the computation cost, accuracy and complexity has to be considered 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 sensitive 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 and can be used to analyze applications that implement them.

Code extract from [2].

```
Public class Main{

Public static void main(){

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

System.out.println(myHashMap);

}}
```

6. CONCLUSION

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 computing summaries according to their sensitivity .Lastly it's not practically possible to achieve 100% preciseness and thus preciseness is a continuous process

7. FUTURE WORKS

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 these 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.
- The study also suggests new call graph algorithms for large java libraries like JDK and android SDK.
- The study suggests a repeat of this study using soot framework and comparing the findings with this study.
- The study suggests computation of summaries of the recent versions of android Bluetooth library and consideration of evaluation of computed summaries with malicious applications.

8. ACKNOWLEDGMENTS

Sincerely thank the Dexter labs organization for providing an open source android static analysis tool that we used for analyzing the applications and lastly the GitHub specifically Georgious Gousious for his java call graph: java call graph utilities suite that we used to analyze the android Bluetooth library

9. REFERENCES

[1].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.

[2].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.

[3].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.

[4].Ali, K., 2014. *The Separate Compilation Assumption* (Doctoral dissertation, University of Waterloo).

[5.]Allen, N., Krishnan, P. and Scholz, B., 2015, June. Combining type-analysis with points-to analysis for analyzing Java library source-code. In*Proceedings of the 4th ACM SIGPLAN International Workshop on State Of the Art in Program Analysis* (pp. 13-18). ACM.

[6].Smaragdakis, Y., Balatsouras, G., Kastrinis, G. and Bravenboer, M., 2015, November. More sound static handling of Java reflection. In *Asian Symposium on Programming Languages and Systems* (pp. 485-503). Springer International Publishing.

[7].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*.

[8].Gibler, C., Crussell, J., Erickson, J. and Chen, H., 2012, June. AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale. In *International Conference on Trust and Trustworthy Computing* (pp. 291-307). Springer Berlin Heidelberg.

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

[10].Gibler, C., Crussell, J., Erickson, J. and Chen, H., 2012, June. AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale. In *International Conference on Trust and Trustworthy Computing* (pp. 291-307). Springer Berlin Heidelberg.

[11].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.

[12].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, Campus SophiaTech, 1504.06093v2, EURECOM, Sophia Antipolis cedex,France

[13].Steven Arzt and Eric Bodden 2014, *Secure software Engineering group*, viewed 20 January 2015, <http://sse.ec-spride.de/>.

[14].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.

[15].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.