

May 2015

A Survey on Potential Privacy Leaks of GPS Information in Android Applications

Srinivas Kalyan Yellanki
University of Nevada, Las Vegas, yellas1@unlv.nevada.edu

Follow this and additional works at: <https://digitalscholarship.unlv.edu/thesesdissertations>

 Part of the [Library and Information Science Commons](#)

Repository Citation

Yellanki, Srinivas Kalyan, "A Survey on Potential Privacy Leaks of GPS Information in Android Applications" (2015). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 2449.
<https://digitalscholarship.unlv.edu/thesesdissertations/2449>

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

**A SURVEY ON POTENTIAL PRIVACY LEAKS OF GPS
INFORMATION IN ANDROID APPLICATIONS**

By

Srinivas Kalyan Yellanki

Bachelor of Technology, Information Technology
Jawaharlal Nehru Technological University, India
2013

A thesis submitted in partial fulfillment of the requirements
for the

Master of Science - Computer Science

**School of Computer Science
Howard R. Hughes College of Engineering
The Graduate College**

**University of Nevada, Las Vegas
May 2015**

© Srinivas Kalyan Yellanki, 2015
All Rights Reserved



THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Srinivas Kalyan Yellanki

entitled

A Survey on Potential Privacy Leaks of GPS information in Android applications

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

School of Computer Science

Dr. Ju- Yeon Jo, Ph.D., Chair Person

Dr. Ajoy K. Datta, Ph.D., Committee Member

Dr. Yoohwan Kim, Ph.D., Committee Member

Dr. Venkatesan Muthukumar, Ph.D., Graduate College Representative

May 2015

ABSTRACT

A Survey on Potential Privacy Leaks of GPS Information in Android Applications

By

Srinivas Kalyan Yellanki

Dr. Ju-Yeon Jo, Examination Committee Chair

Associate Professor, Department of Computer Science

University of Nevada, Las Vegas

Android-based smart phones are extremely common today. However, it is believed that nearly half of the Android devices are vulnerable to an attack that alters the functionality of an app with malicious software. The malware can collect users sensitive data from the phone. In particular, there are hundreds of location-based applications available nowadays in the Google Play store or other app stores. The very famous services called “Location Based Services” are used by many apps on mobile phones to track the geographical coordinates of the device. Such location information can be leaked to an attacker via malware.

In this thesis, we discuss different ways in which privacy can be breached in android applications and their countermeasures. The vulnerabilities, the method of detecting the information leakage, and the measures to control the security breach are discussed. The experimental results show how effectively those different countermeasures can help us in preventing the security breaches and information leakage of GPS data in android applications.

ACKNOWLEDGEMENTS

I would like to thank Dr. Ju-Yeon Jo, my research advisor for all the support and guidance she has offered me during the course of my graduate studies at University of Nevada, Las Vegas. Her encouragement and valuable suggestions have helped me immensely in seeking the right direction for this thesis. I would like to thank Dr. Yoohwan Kim, who deserves special recognition for his wholehearted guidance and support throughout my thesis.

I would also like to thank Dr. Ajoy K. Datta, Dr. Yoohwan Kim and Dr. Venkatesan Muthukumar for serving my committee and reviewing my thesis. I am grateful to Dr. Ajoy K. Datta for all his support and guidance through my Master's program.

This thesis has been a challenging experience to me and was accomplished through help of many people. In particular, I would like to express my sincere gratitude to Dr. Ajoy K. Datta for providing me assistantship.

My deepest gratitude to my parents Mallesh Yellanki and Vijaya Laxmi Yellanki for their love, care and opportunities they have provided me at every stage of my life. I would also like to thank my brother Sampath Yellanki, sister Sadhana Yellanki for their support and guidance in building up my career. Last but not the least; I would like to thank all my friends and well-wishers for their support.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENTS.....	iv
TABLE OF CONTENTS.....	v
LIST OF TABLES.....	ix
LIST OF FIGURES.....	x
CHAPTER 1 INTRODCUTION.....	1
1.1 Android History.....	1
1.2 Pre-Commercial Release Versions.....	3
1.2.1 Alpha Version.....	3
1.2.2 Beta Version.....	3
1.3 Outline.....	4
1.4 Motivation.....	5
1.5 Attacking Procedure.....	6
CHAPTER 2 ANDROID ARCHITECTURE.....	8
2.1 Android Security Architecture.....	8
2.2 System and Kernel Security.....	9
2.3 Application Security in Android.....	10
2.4 Sandbox.....	11

2.4.1 Functionality.....	13
2.5 Droidbox.....	14
CHAPTER 3 VULNERABILITIES.....	17
3.1 Vulnerabilities of Data Leakage in Android Applications.....	17
Data Leak through the Android App Structure.....	17
3.1.1 ContentProvider and Permission.....	17
3.1.2 Data Leak through Reverse Engineering.....	20
a) Using Dedexer Tool.....	21
b) Data Leak through Apk Manager.....	22
c) Proguard.....	23
3.2 Vulnerabilities of Data Leakage in Android GPS Applications...	24
CHAPTER 4 LEAKAGE DETECTION ANALYSIS.....	28
Detecting Leakage Analysis in GPS Applications.....	28
Approaches.....	29
4.1 LeakMiner - A Static Taint Analysis.....	29
Functionality.....	31
4.1.1 Fundamental activity lifecycle callbacks.....	31
4.1.2 Activity supplementary callbacks.....	32
4.1.3 Basic service lifecycle callbacks.....	32
4.1.4 Service supplementary callbacks.....	32

4.2 TaintDroid – a Dynamic Taint approach.....	33
4.3 Detection Leakage through Hybrid Analysis.....	38
4.4 Detection Leakage through Cloud based analysis.....	39
4.5 Leak Detection through Kynoid.....	41
4.6 DroidVulMon.....	43
4.6.1 Security Architecture for malicious App detection.....	43
4.7 Overall Comparison.....	46
CHAPTER 5 ANDROID APP- TRCK ME.....	48
5.1 Location Tracking Applications.....	47
5.2 Case Study.....	49
5.3 Personalized Android app – Track Me.....	51
Functionality.....	51
5.3.1 Welcome Screen.....	52
5.3.2 Go to application.....	53
5.3.3 Contacts.....	54
5.3.4 Choosing a Contact.....	55
5.3.5 Start Updating.....	56
5.3.6 Text Message Notification.....	57
5.3.7 Google Map View.....	58

CHAPTER 6 RECOMMENDATIONS FOR SECURE APPS.....	59
CHAPTER 7 CONCLUSION AND FUTURE WORK.....	61
BIBLIOGRAPHY.....	62
VITA	

LIST OF TABLES

Table 1 Malicious applications classification.....	24
Table 2 Information leakage reported by LeakMiner.....	33
Table 3 Comparison of various Taint Analysis techniques for android-I...36	
Table 4 Comparison of various Taint Analysis techniques for android-II..37	
Table 5 Privacy Leak Detection Frameworks-I.....	45
Table 6 Privacy Leak Detection Frameworks-II.....	46
Table 7 Comparison of different Location sharing apps on app store.....	49

LIST OF FIGURES

Fig 1. Android Software Stack.....	9
Fig 2. Sandbox Manager.....	12
Fig 3. AASandbox Architecture.....	13
Fig 4. An Example of Behavioral Graph.....	16
Fig 5. Database Sharing between apps.....	19
Fig 6. Relationship between apps and permissions.....	20
Fig 7. Conversion of Android app files.....	22
Fig 8. Reverse Engineering through Apk Manager.....	23
Fig 9. Potential Privacy Leaks of Android Application.....	26
Fig 10. Overall Architecture of LeakMiner.....	30
Fig 11. TaintDroid within the Android Application.....	34
Fig 12. SmartDroid Architecture.....	38
Fig 13. Paranoid Android Architecture.....	39
Fig 14. Core idea of Kynoid.....	41
Fig 15: System Architecture.....	44
Fig 16. Home Screen of app.....	52
Fig 17. Go to Application.....	53
Fig 18. Adding a Contact.....	54

Fig 19. Choosing a Contact.....	55
Fig 20. Start Updating.....	56
Fig 21. Text Message Notification to receiver.....	57
Fig 22. Google Map View.....	58

CHAPTER 1

INTRODUCTION

It is very common today that everyone is using smartphones. There are many location-based applications available now days in the play store or App store. The very famous services called “**Location Based Services**” are used by many applications to track the geographical coordinates of the device. The usage of this service is growing fast in the android applications.

1.1 Android History

In current mobile market, android is the most widely used operating system on smartphones. The android version history of the android operating system has begun with the release of the Android beta in November 2007 and then the first commercial version, Android 1.0, was released in September 2008[1]. Android is taken up by Google and currently is under ongoing development by Google and the Open Handset Alliance (OHA).

The most recent major Android release in operating system was Android 5.0 "Lollipop", which was released on November 3, 2014 by Google. Since April 2009, the nomenclature of Android versions have been developed under a confectionery-themed and released in an alphabetical order, beginning with

Android 1.5 "Cupcake"; the earlier versions 1.0 and 1.1 were not released under specific code names:

- Alpha (1.0)
- Beta (1.1)
- Cupcake (1.5)
- Donut (1.6)
- Eclair (2.0–2.1)
- Froyo (2.2–2.2.3)
- Gingerbread (2.3–2.3.7)
- Honeycomb (3.0–3.2.6)
- Ice Cream Sandwich (4.0–4.0.4)
- Jelly Bean (4.1–4.3.1)
- KitKat (4.4–4.4.4, 4.4W–4.4W.2)
- Lollipop (5.0–5.1)

Google announced that more than one billion activated Android devices were in use worldwide[2] on September 03, 2013 by the users. Recently in January 2015, Android devices accounted for approximately 62% of the US smartphone and tablet market.

1.2 Pre-Commercial Release Versions

The actual development of Android was started in 2003 by Android, Inc., and later was purchased by Google in 2005[3].

1.2.1 Alpha version

There were at least two internal releases of the software inside Google and the OHA before the beta version was released in November 2007 as per sources. For the milestones in internal releases, names of fictional robots were chosen, with various releases code-named Astro Boy, Bender and R2-D2 [4][5][6].

Dan Morrill created some of the first mascot logos, but the current green Android logo was designed by Irina Blok[7]. The project manager, Ryan Gibson, conceived the confectionary-themed naming scheme that has been used for the majority of the public releases, starting with Android 1.5 "Cupcake".

1.2.2 Beta version

The beta version was released on November 5, 2007[8][9], while the software development kit (SDK) was released on November 12, 2007[10]. The November 5 date is popularly celebrated as Android's "birthday"[11]. Public beta versions of the SDK were released in the following order[12]:

- November 12, 2007: m3-rc20a (milestone 3, release code 20a)^[13]
- November 16, 2007: m3-rc22a (milestone 3, release code 22a)^[14]
- December 14, 2007: m3-rc37a (milestone 3, release code 37a)^[15]
- February 13, 2008: m5-rc14 (milestone 5, release code 14)^[16]
- March 3, 2008: m5-rc15 (milestone 5, release code 15)^[12]
- August 18, 2008: 0.9^{[17][18]}
- September 23, 2008: 1.0-r1^{[19][20]}

1.3 Outline

In this survey, we discuss the different ways in which privacy can be breached in android applications. The few issues are:

1. Detection of GPS information leakage
2. Vulnerabilities
3. Privacy issues
4. Measures to control the leakage and security breach

The survey covers everything right from creation of an android application in an effective manner in order to prevent the security issues among the applications and then architecture of a typical android application then discusses the privacy issues in modern android applications followed by solutions to prevent the security breach.

This survey includes extensive experiments results that how effective different measures help us in preventing the detection of security breaches and leakage of GPS data in android applications with a high accuracy rate.

At last, we categorize the detection results in our survey and provide the best solution that provides the minimal leakage of GPS data in an android application.

1.4 Motivation

Almost nearly half of the Android devices are vulnerable to an attack that could replace app functionality with malicious software that can steal and gather sensitive personal data from a phone.

Major android developer giants like Google, Samsung and Amazon have released patches for their own devices, but 49.5 percent of Android users are still vulnerable according to Palo Alto Networks[21], which suffered the problem. Google said it has not detected attempts to exploit the flaw.

An infected application can be installed using the vulnerability, called "Android Installer Hijacking". This will have full access to a device, including personal sensitive data such as usernames and passwords, wrote Zhi Xu, a senior staff engineer with Palo Alto.

The vulnerability only affects apps that are installed from a third-party play store or websites. It is highly recommended to be more cautious while downloading such apps from third party websites.

The apps that are downloaded from third party websites place their APK installation files in a device's unprotected or unmonitored local storage, such as an SD card.

From these places, a system application called PackageInstaller finishes the installation. The flaw allows an APK file to be modified or replaced which may be malicious, during installation without anyone knowing.

1.5 Attacking Procedure

Install and attack flow would be like this: User downloads an app what appears to be a legitimate application from the third party website or app store. The application after downloading and before installation asks for certain permissions on the device.

During that process, it was possible to swap or alter the APK file in the background because the PackageInstaller fails to verify it. After the user clicks the install button, the PackageInstaller can actually install a different malicious app with an entirely different set of permissions.

The main problem with the Android devices does not need to be rooted for the attack to work or to inject a malware, although rooting also make devices more vulnerable.

When the major flaw was detected, in January 2014, approximately 90 percent of all Android devices were affected. That has since dropped to 49.5 percent, but still many devices have not been patched.

The recent Palo Alto's exploit was a huge success against Android versions 2.3, 4.0.3 to 4.0.4, 4.1.x, and 4.2.x. The 4.4 version of Android devices fixes the issue and some Android 4.3 devices may still be affected and since some manufacturers have not patched yet.

In order to overcome these Malwares, Google has published a patch, and Amazon recommends its users to download the latest version of the Amazon AppStore, so that they update their Fire devices.

CHAPTER 2

ANDROID ARCHITECTURE

2.1 Android Security Architecture

Android is the most modern mobile platform designed and is open in the market. All the android applications use the most advanced hardware and software to offer variety of innovative features that value to customers. In order to protect that value, the coding platform must provide an application environment that provides the security for users, data and applications on device.

In order to secure the open platform, it requires a strong security and rigorous security programs. This can be achieved by android by adopting the multi-layered security and also provides protection for all the users using the platform. Android security system is designed with device users in mind.

This design includes the expectation that malicious app attackers would try to perform attacks such as social engineering attacks[1] to make users to install malware. The android security system is designed to reduce the both chances of these attacks and to limit the impact of attack on any app.

2.2 System and Kernel Security

At the operating system level, android comprises Linux kernel and secure inter-process communication (IPC). Android provides security to these two major components to achieve secure and safe communication between apps running in different processes. These security features at the Operating System level make sure that even the code is severely restricted by the Application Sandbox.

The fig.1 below shows the Android Software Stack with bottom up approach depicting various levels of Android development.

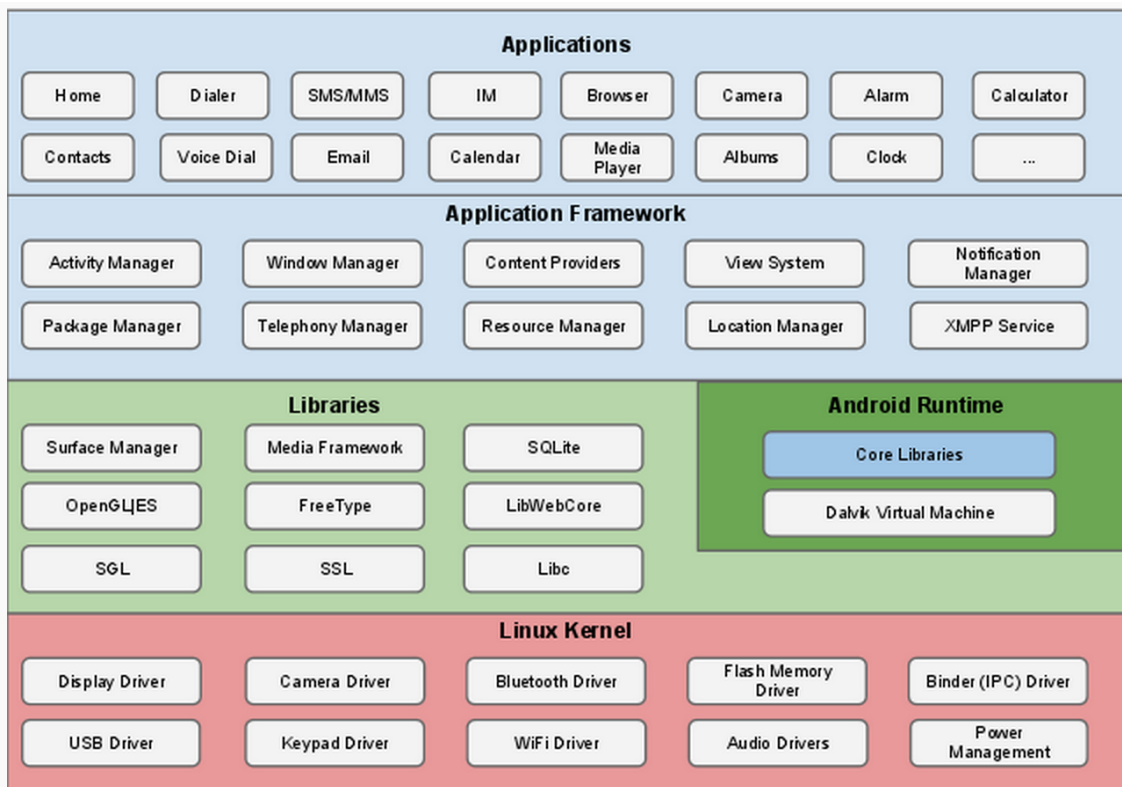


Fig 1. Android Software Stack[40]

2.3 Application Security in Android

Android core operating system is completely based on the Linux kernel. The most android applications are programmed in Java programming language and run in the Dalvik Virtual Machine (DVM). All the android applications are installed from a single file with the .apk extension.

The android application building blocks are:

- **AndroidManifest.xml:** The AndroidManifest.xml file is the metadata file that is considered as control file that tells the system, what to do with all top-level components in an application. This file specifies what all permissions are granted to the app.
- **Activities:** An Activity is usually includes displaying a UI to the user and typically application's Activities is the entry point to an application.
- **Services:** Services are the body or execution part of code that runs in the background. The other components "bind" to a Service and trigger methods on it via remote procedure calls.
- **Broadcast Receiver:** A BroadcastReceiver is an object that is instantiated when intent is issued by the operating system. An app may register a receiver for the low battery message and change its behavior based on that specific-information.

Security features are to be considered while developing Android applications to lessen the security breach. This can be achieved by encrypting the file system so that it keeps device safe during any theft or loss. This can be achieved by a concept called “SandBox”.

2.4 Sandbox

This Sandbox technique[26] helps users to isolate the app data and codes from the other applications on the device. So that developers can define their own permissions that are specific to their own applications.

Due to this isolation of application data and codes with the other applications, the scope of influencing malicious applications is very less. This Sandbox isolates apps data and code accessing from other apps on the device so that it can prevent any vulnerabilities of malicious attempts by other apps.

But in real scenario, there is a necessity of accessing Phone book or Photo Gallery from the messenger applications on the device by which there is a large scope for malicious apps to attack the applications data or code.

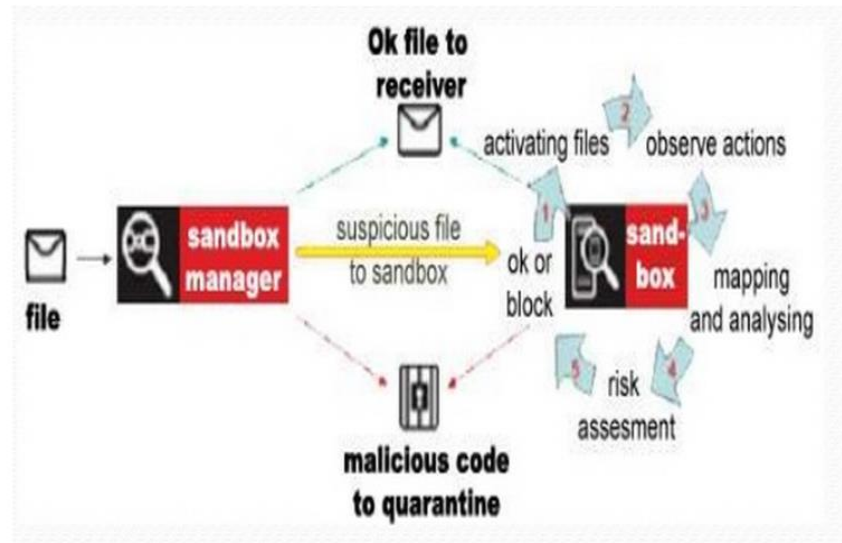


Fig 2: Sandbox Manager[26]

The above figure.2 depicts the overall architecture of the Android Sandbox Manager. To overcome this, an API called ContentProvider is responsible for authenticating and managing accesses to the databases of other apps. Permissions and ContentProvider are described in further detail in the following section.

For any unusual or malicious activity detection; knowledge of application’s characteristics is essential. There are two most common and best practices that exists in mobile software for analyzing the application’s activity. They are “**Static and Dynamic analysis of software**”.

These two techniques have many advantages and disadvantages in terms of their evaluation. Static analysis involves various binary forensic techniques, including decompilation, decryption, pattern-matching and static system call analysis.

Functionality

The main functionality of these techniques is to prevent running the potentially malicious software on the devices. The Static Analysis usually filters the binaries with malicious patterns, known as Signatures.

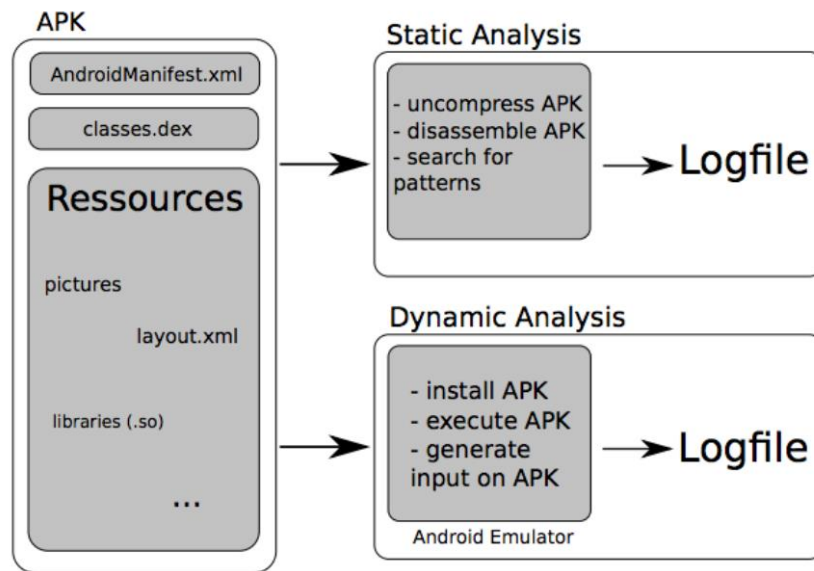


Fig.3 AASandbox Architecture[26]

The above fig.3 shows the architecture of Sandbox with both the Static and Dynamic Analysis. Due to the direct comparison with the predefined signatures, the static analysis is very fast and simple.

Almost all the anti-virus software are based on this approach. Difficulty with the Static Analysis is that each and every signature/malicious pattern must be known in advance during the analysis. By which it is impossible to detect new malware or malicious code entry from any external malicious application.

2.5 Droidbox

We dedicate this section to a detailed description of Droidbox[24] and its capabilities. DroidBox[25] is a dynamic analysis tool for Android applications targeting Android version 2.1.

The tool is based on TaintDroid[74] for detecting information leaks but has been extended, by modifying the Android framework, to monitor API calls of interest invoked by an application.

Applications are executed within the Android SDK emulator and logs are issued for each monitored behavior and collected in the host operating system. A text-based report is generated after analysis has ended and provides a summary of the execution.

On mobile phones, malware has been discovered that listens for incoming SMS and forwards this information to the attacker. In, TaintDroid was used to track sensitive data originating from the phone's database.

DroidBox can extend this approach by adding and modifying output channels throughout the Android framework to detect leaks via outgoing SMS and to disclose full details of the network communication, not only in network leak scenarios.

The file `AndroidManifest.xml`, included in the Android package, contains permissions that are needed for the application to interact with the operating system, for example, connecting to the Internet, sending SMS, making calls and receiving incoming SMS.

Applications that need to interact with any resources must declare the appropriate permission in the manifest file. It has been demonstrated that malicious Android applications can circumvent the permission policies and [23], thus DroidBox compares each monitored operation that requires any permission with the package's manifest file to check if any permission policy has been bypassed.

Some malicious Android applications can evade anti-virus software by performing obfuscation and changing themselves during run-time [27]. Obfuscation may include cryptographic functions applied to the data. DroidBox is designed to detect applications as they invoke cryptographic keys or perform encryption or decryption on the data

Malicious Android applications can perform phone calls or send SMS to premium rate numbers that are declared by the attacker.

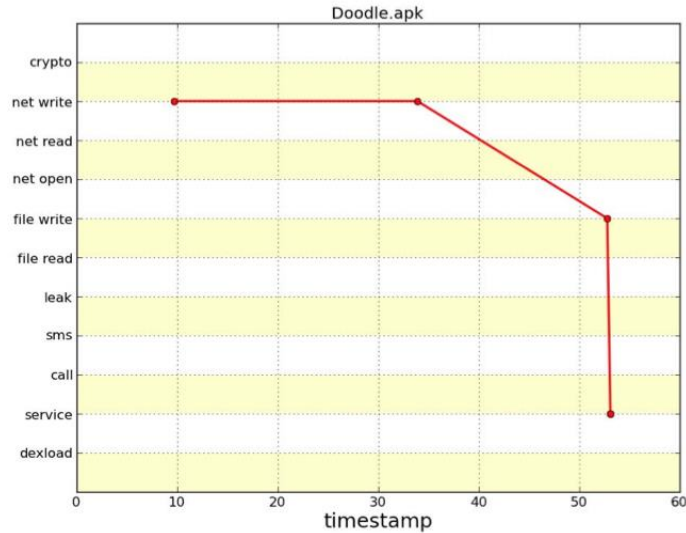


Fig 4. *An Example of Behavioral Graph[24]*

The above figure shows the behavior graph produced by Droidbox describes the temporal order of the operations. DroidBox can disclose these operations by listening to API calls when SMS and calling methods are invoked by a sample.

Figure 4 shows the output of the sample doodle.apk that was executed for 60 seconds. On the x- axis the time of the monitored operation is shown, while the y- axis describes what kind of operation type was monitored. This graph is generated by picking operations and timestamps for each operation from the sandbox log and plotting them.

CHAPTER 3

VULNERABILITIES

Vulnerabilities of Data Leakage in Android Applications

Apps on android devices are highly vulnerable for data leakages because the android platform is open for developers and the android security allows users to download apps from third party websites or app stores. Therefore data leakage from android apps is very common and has high scope for stealing the personal sensitive data on phone.

Data leakage in android applications can be done in many ways. Few ways are listed below in detail[55].

3.1 Data leak through the Android App Structure

Personal sensitive data leakage in android application is done mainly through the app structure. An android app is undergoes many steps during the development [28]. Different ways in which data leakage occurs is discussed in following sections:

3.1.1 ContentProvider and Permission

There are many applications[29] installed on a device and each application knows the structure and file system of other applications and also knows how to interact

and access to the databases of other applications.

In order to grant access/permissions to the other applications databases, android interleaves ContentProviders[30], which is based on client-server model. So not all the applications cannot access the databases of other applications until permission is acquired from ContentProviders.

There are some applications that run on server. These applications also require databases to access the content. So with the help of ContentProviders, apps that run on server share their content to other applications on the device.

The server applications provide a unique URI to identify their database and specify these URI in AndroidManifest.xml or ContentProvider file. The applications that are so called client applications make use of these URI by making requests to send queries to a particular database of a server app.

So, the client apps in order to use the server-side databases it should know the names, table structures and URI of the database. Unlike ContentProviders at server-side there are ContentResolvers at client-side.

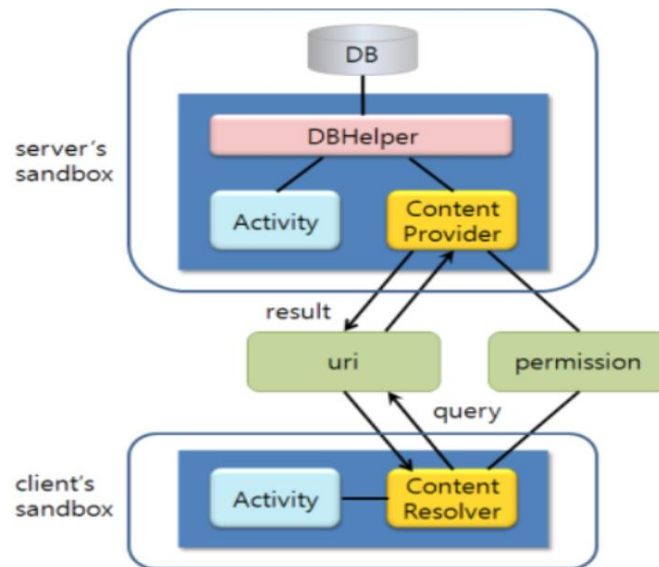


Fig 5: Database Sharing between apps[29]

The figure 5 explains how apps on android devices share resources like database. These ContentResolvers are used by client apps to communicate with the server-side ContentProvider. Queries and URIs are sent to server-side ContentProvider using these ContentResolvers.

These ContentProviders and ContentResolvers serve as intermediate for sharing databases between server and client apps. When these ContentResolver sends queries to the ContentProvider at server-side, the ContentProvider receives results from the database via DB Helper and send these results back to the ContentResolvers.

In order to access major resources or system functions, all the android applications see permission through the AndroidManifest.xml file[31] which is a unique file that holds all the permissions of Android System.

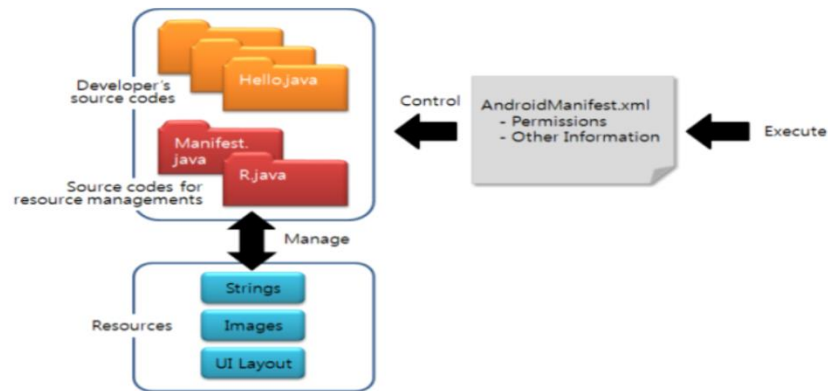


Fig 6: Relationship between apps and permissions[29]

The figure 6 explains the relationship between apps on android devices and how the permissions are granted. Now the applications that run on client side send query requests to ContentProvider through the ContentResolver to access the database on server.

This Android System checks for the permissions required to access or to use the database by client apps. The service from a particular database may be denied or granted depending upon the permission levels for a particular client request.

3.1.2 Leakage of Data through Reverse Engineering

In order to access the databases of other applications, the assigned URIs to every database are used, queries that require data tables and information of fields attributes and finally permissions required by server apps are needed.

The address books database, it is provided as default to share within the apps. In this case not only URIs but also the access procedures or functions are also offered via System APIs. Data leakage in android apps through Reverse Engineering can be done in 3 different ways:

a) Using Dedexer Tool

The Java source files are converted into dex files while building an app in android. Dedexer is a tool that compiles dex files into Java source files. This is nothing but reverse compiling the files.

While developing an Android app, the Java source code is compiled (.java) to (.class) files, which are called as byte code based on JVM (Java Virtual Machine). These (.class) files are then converted into a dex type file using Dedexer tool.

The simulator that runs apps on Android devices is called Dalvik[32], which reads the byte codes of this dex file. All together the dex files, xml files and resources grouped together and compressed into Java archive (jar file) format and finally to generate an apk file (.apk).

There is one more important file named R.Java, which is a system generated resources file during the creation of app.

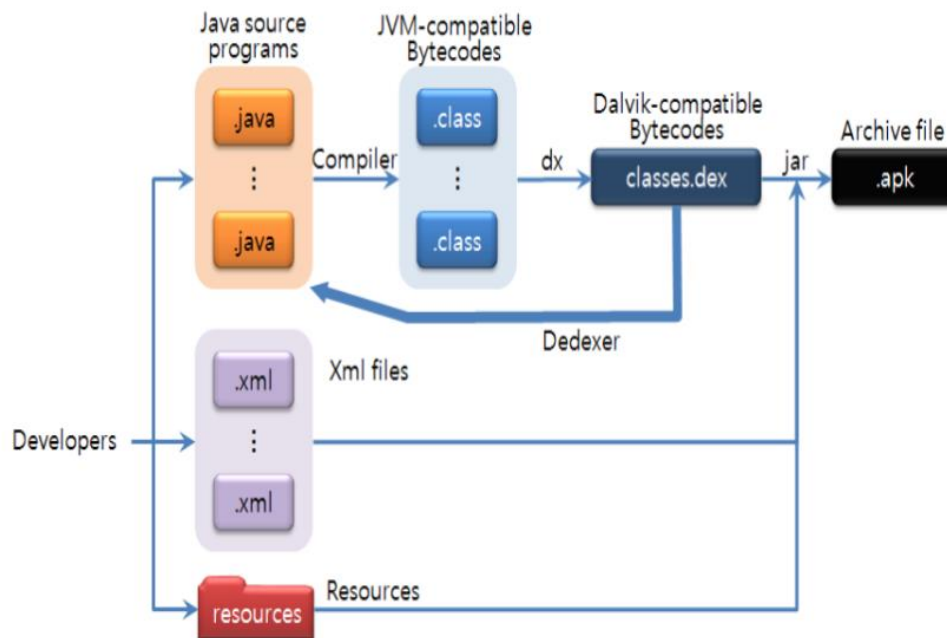


Fig 7: Conversion of Android app files[29]

The figure 7 explains the conversion of android apps files in an android device. This Dedexer reconfigures not only user written Java source programs but also R.Java. Still Dedexer doesn't completely restore all the files through reverse engineering process.

b) Data Leakage through Apk Manager

Dedexer may be accounted as one of the best tool for attackers to manipulate the Java source file through reverse engineering the dex files into Java Source files.

As mentioned earlier, the Dedexer does not support a complete restoration. And also, it restores Java programs only without supporting xml files and resources. That is the reason it is difficult to complete operable apps by analyzing the existing apps and turning them into malicious codes.

But this ApkManager reverses and compiles original Java programs into Smali-based[33] ones, which not only read just source programs but also AndroidManifest.xml.

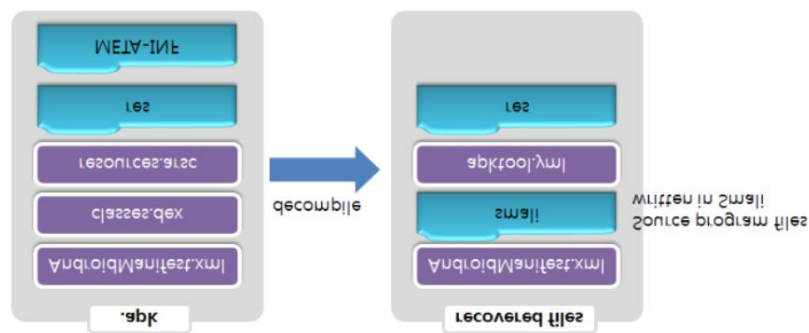


Fig 8: Reverse Engineering through Apk Manager[29]

The above figure 8 explains how the data leakage is done through the Reverse Engineering using the Apk Manager.

c) Proguard

Android system provides a feature called Proguard. This Proguard is used to prevent the reverse-engineered result[34] from being used for malicious codes or malware susceptible codes that influence the app.

On including the proguard.cfg file in the Android project can use this feature. This Proguard deletes the less important codes to optimize and reduce program volume, changes class names, fields and classes into meaningless ones for vagueness.

Even though it takes some time to determine the meaning and context of changed names when vague sources are restored by reverse engineering, the analysis is definitely not impossible, as logics are not changed, which are the limitation of vagueness.

3.2 Vulnerabilities of Data Leakage in Android GPS applications

Many apps on the device use the GPS[57] information for navigation. But before requesting and getting the information it needs to get the access permissions from the user. After too many studies, we discovered that vulnerable apps generally acquire the user's private data at first and then sent them through network or SMS.

Malicious applications are classified into two types[35]:

	Request only Location Information	Request extra Information
Send information via Network	Low risk	High risk
Send information via SMS	High risk	Low risk

Table 1: *Malicious applications classification[35]*

The above table 1 defines the classification of malicious applications. We define the data leaks that contain only GPS information as low-level risk behavior and the data leaks that contain more private information as high-level risk behavior.

Based on the triggering condition of the leaking behavior, we categorize the privacy leaking behavior into three types.

1. The behavior triggered by user interaction with the application.
2. The behavior triggered automatically by background services.
3. The behavior triggered automatically by background services and user Interaction.

The risk of leak triggered by background services on a device is only considered to be superior to other behaviors[35]. There are few papers that are manually analyzed on android malicious applications and noticed that most malicious applications are risky in behavior that mainly focus on collecting and sending the users private information.

For example, FakeFlash is an Android Malware in the disguise of an Abode flash player application that collects user's phone number and phone's IMEI information and then send this data by posting information to remote server[58]. So a complete path from getting privacy data to sending privacy data should be included in a complete attack model.

In the below figure, we define the following specific behaviors as the potential privacy leaks of Android application.

1. Collect user's private information using android framework API.
2. Transform private information into another form.
3. Send the transformed information to remote phone or remote server.

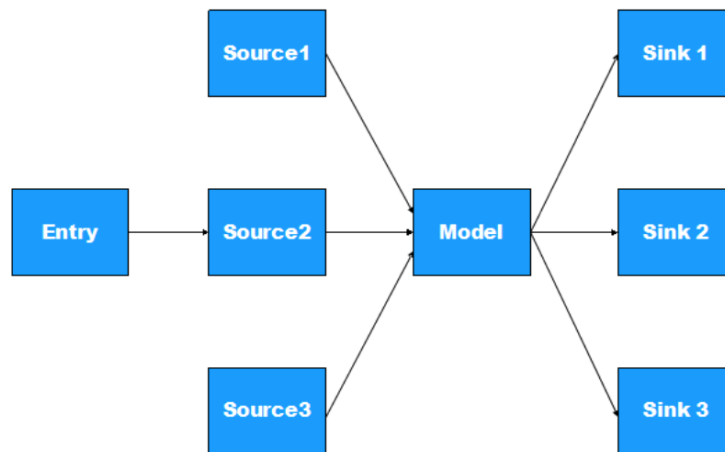


Fig 9: *Potential Privacy Leaks of Android Application*[35]

The above figure 9 explains the potential privacy leaks of android applications. In the attack model, we define *Entry block* as the triggering action, which then leads to privacy data leak, *Sources block* as the action, which collects sensitive information, sinks block as *Privacy sending action*.

A path that is connecting a source and a sink is considered as a *Confirmed path*. There is a leakage detection called “Brox” which detects location leakage information.

This tool is based on Dalvik-opcode[36] specification and uses data flow analysis framework equipped with context-sensitive, flow-sensitive and inter-procedure techniques to detect potential location information leakage path in Android malicious apps.

This Brox is also a static privacy leak detection framework for Android apps. Got inspired by WALA[56], that provides static analysis capabilities for Java byte-code, JavaScript and related other languages; this Brox inherits its effectiveness and correctness by using inter-procedure analysis framework.

CHAPTER 4

Detecting Leakage Analysis in GPS Applications

Android is the most widely used Mobile operating system on many smartphones. There are many information flow tracking and information leakage detection techniques that are developed on Android operating system.

The most commonly used technique is **Taint analysis**. This Taint Analysis is a data flow analysis technique, which tracks the flow of private or sensitive GPS information and its leakage.

There are varieties of apps available for different platforms and device configurations. Apps are made for PC users and also available for Mobile users. These android apps manipulate personal data such as contact and SMS, GPS information and leakage of such sensitive information may cause great loss to the android users.

Therefore, detecting sensitive GPS information leakage on Android is in urgent need. However, till now, there is still no complete perfect solution available to get rid of this scenario to Android markets.

A famous approach called **State-of-the-art** is used for detecting Android GPS information leakage by applying dynamic analysis on user site, thus they introduce large runtime overhead to the Android GPS apps.

The main major difference between both the cases is, static analysis techniques look at the complete program source code and all possible paths of execution before its run, whereas dynamic analysis looks at the instructions executed in the program-run in the real time.

Approaches

There are many different approaches proposed to detect and determine the leakage of GPS data in an android device. This detection analysis can be done in two different ways. By static analysis and dynamic analysis. Few of the detections techniques are proposed below:

4.1 LeakMiner - A Static Taint Analysis

There are different approaches available to detect the leakage in android applications[37]. LeakMiner is one among them, which detects leakage of sensitive GPS information on Android with a static taint analysis. LeakMiner analyzes Android apps on market site unlike Dynamic analysis.

Therefore it does not introduce any runtime overhead to normal execution of target applications on a device. Besides, this LeakMiner technique can detect sensitive information leakage before apps are available to users, so vulnerable apps can be identified and removed from play store before users download them.

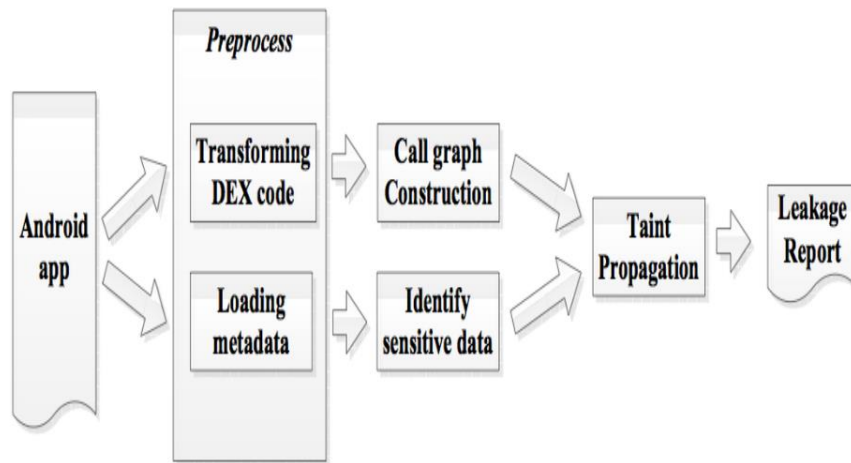


Fig 10: Overall Architecture of LeakMiner[38]

The above figure 10 explains the overall architecture[38] of LeakMiner technique, which is a static analysis for detecting the data leakage in android applications. Starting with android app and it traverses in a reverse manner and produces the Leak report.

Functionality

Unlike others programs, Android apps[46] doesn't contain the main function, which is usually named the root or main function in C/C++ or Java. Instead of consisting single entry point, Android apps may have multiple entry points. These entry points are pre-defined by service interfaces and Android activity.

Our static taint analysis technique first build call graphs[59] which starts at these one of the entry points. Then a new root function node is used to link these call

graphs by constructing function call edges from the root node to each of the entry nodes. The entry points that were traced by static taint analysis are listed below:

4.1.1 Fundamental activity lifecycle callbacks

Android provides six basic lifecycle callbacks that are *onCreate*, *onStart*, *onResume*, *onPause*, *onStop*, and *onDestroy*. User can override any of this callback through any Android activity to do an appropriate work. These are the hooks that are invoked by Android activity manager when the state of activity changes. These callback functions are fundamental activity entry points for every android app.

4.1.2 Activity supplementary callbacks

Fundamental activity[46] lifecycle callbacks are triggered each time when the state of activity is changed. Besides, Android provides supplementary hooks, which assist the function of Android system for efficient resource management.

For example, when the system destroys or kills an activity in order to restore memory, a function *onSaveInstanceState* is invoked to save the current state of activity. A corresponding *onRestoreInstanceState* hook is invoked to restore the state when user navigates to the activity.

4.1.3 Basic service lifecycle callbacks

Like activities, Android services also have life callbacks, which are automatically triggered by android service manager such as *onStartCommand*, *onBind*, *onCreate*, and *onDestroy*[46]. These callback services are the basic entry points for analyzing the service application behaviors.

4.1.4 Service supplementary callbacks

This Service supplementary callbacks are usually triggered when the configuration is altered (eg. *onConfigurationChanged*), or when the resources are exploited (eg. *onLowMemory*).

Leakage Source	LeakMiner Report	True Leak age
Device ID	278	127
Phone Information	53	50
Location	35	27
Contacts	12	12
Total Detected	305	145

Table 2: *Information leakage reported by leakminer about half of the reported informaton leakage is true information leakage.[46]*

These functions are infrequently triggered, and our analysis includes such interfaces for the sake of code coverage. The above table 2 explains the information leakage reported by LeakMiner in which about half of the reported information leakage is true information leakage.

4.2 TaintDroid – a Dynamic Taint approach

TaintDroid is a dynamic taint tracking[39] system for Android devices available system-wide. Simultaneously, this TaintDroid[65] approach can track multiple sources and sinks. When the sensitive information leaks through the system at the run time, the Android phone users are get notified.

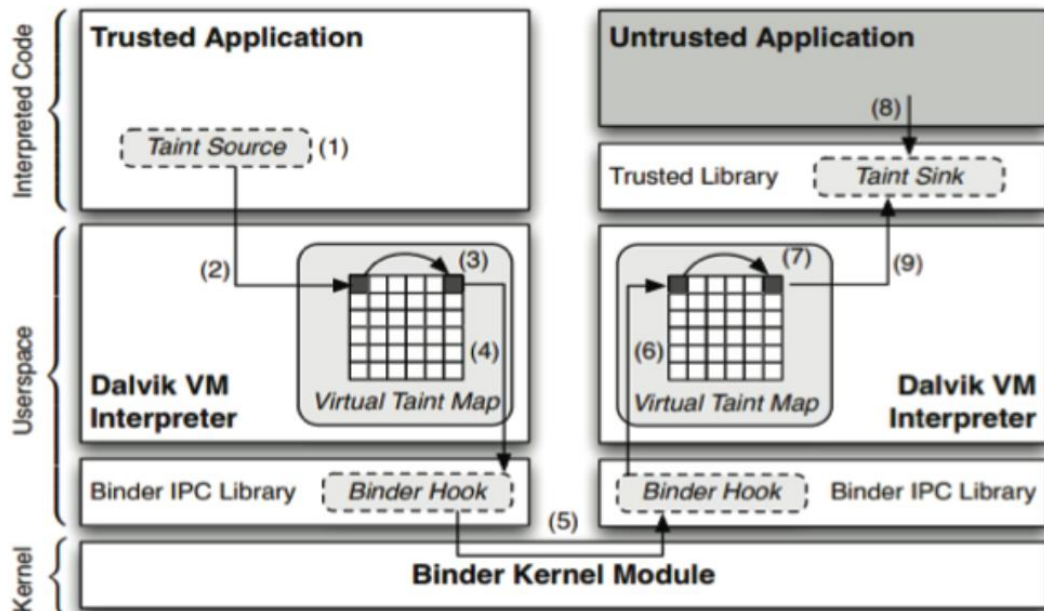


Fig 11: TaintDroid within the Android Application[60]

The above figure 11 explains the TaintDroid structure[40] in an android application. The main functionality of TaintDroid is achieved by altering the Dalvik virtual machine of Android. It introduces variable level taint tracking in android system using the shadow variables.

The size of each variable is doubled from 32 bits to 64 bits with modifying the stack format. These additional 32 bits are used to store the taint tag ID. These taint tag ID's are the unique identifiers to identify the sensitive information such as Geographical Location coordinates and IMEI.

This TaintDroid offers various levels of tracking in android system. It does variable level tracking for the native code in the system. For the secondary storage files it does File level tracking.

It does the message level tracking for the Inter Process Communications and finally it does the method level tracking for the native code. The taint tags are added to the sources in the taint system and when it reaches the sinks in taint system, these tags are processed to recognize and find out which information is leaked through at the corresponding sink.

Each application in android environment has their own sandbox with its own User ID, Dalvik Virtual Machine instance and respective set of permissions assigned to it.

In a trusted application the Taint source is marked, which is mapped in virtual taint map by the new or modified Dalvik virtual machine. The Binder IPC is responsible for the inter-app communication and that carries the tainted data to binder hook of untrusted app.

This tainted data is then mapped and propagated in corresponding virtual taint map. This virtual map is of untrusted application according to data flow rules. When the untrusted application triggers a taint sink specified library, the particular tag from the tainted data is retrieved and the event is reported directly to user.

This TaintDroid technique tracks information flows at real-time for the privacy monitoring. TaintDroid has 14% performance overhead on a CPU bound micro-benchmark.

The TaintDroid implementation requires constructing a ROM i.e., patched version of Android operating system in a customized android release. CyanogenMod ROM is an OS similar to Android available in market. This CyanogenMod ROM's has been released in some of the Samsung Galaxy devices.

This TaintDroid has been integrated with this CyanogenMod ROM. There is a successful author who presented collection of attacks on TaintDroid exploring its effectiveness and limitations. The author applied generic classes of anti-taint methods to overcome TaintDroid.

Approach	Static/Dynamic Analysis	Sensitive information Sources defined	Implicit flows
Taint Droid	Dynamic Analysis	5 sources 32 possible	Method level tracking
AppFence[41]	Dynamic Analysis	12 sources predefined	Not analyzed
Kynoid[42]	Dynamic Analysis	2^{32} possible	Not analyzed
LeakMiner	Static Analysis	6 sources predefined	Not analyzed
TrustDroid[43]	Static Analysis	Not mentioned	Not analyzed
FlowDroid[44]	Static Analysis	Exhaustive list	Analyzed[45]

Table 3: *comparison of various Taint Analysis Techniques for android[37]*

	Inter app Taint Propagation (IPC)	Deployment	Open Source availability
Taint Droid	Message level taint tracking	Customized Android version	Available
AppFence	Not mentioned	Customized Android version	Available
Kynoid	Inter Process Tracking	Customized Android version	Not available
LeakMiner	Not analyzed	Deployed on computer	Not available
TrustDroid	Not analyzed	Can be deployed on computer as well as phone	Not available
FlowDroid	Not analyzed	Deployed on computer	Available

Table 4: *comparison of various Taint Analysis Techniques for android with other techniques[37]*

The above tables 3 and 4 explain comparison of various Taint Analysis[62] Techniques for android with other techniques. Among all the approaches from table 3 only the TaintDroid approach follows the Method level tracking and all other approaches are not analyzed. From table 4 only the TaintDroid approach and AppFence approach has the open source availability.

From table 4, all the approaches including the TaintDroid can be deployed in a customized android version whereas the FlowDroid approach is deployed on computer instead of mobile phone.

4.3 Detection Leakage through Hybrid Analysis

Hybrid analysis is the combination of both static and dynamic analysis to enhance the privacy leak detection. The overview of hybrid privacy[72] detection tool known as SmartDroid[66] is shown in the below figure. The leak detection of SmartDroid varies at different levels of the application.

At the higher levels it implements a static path selector that uses static analysis to extract the expected activity of switching paths by analyzing function CFGs and activity.

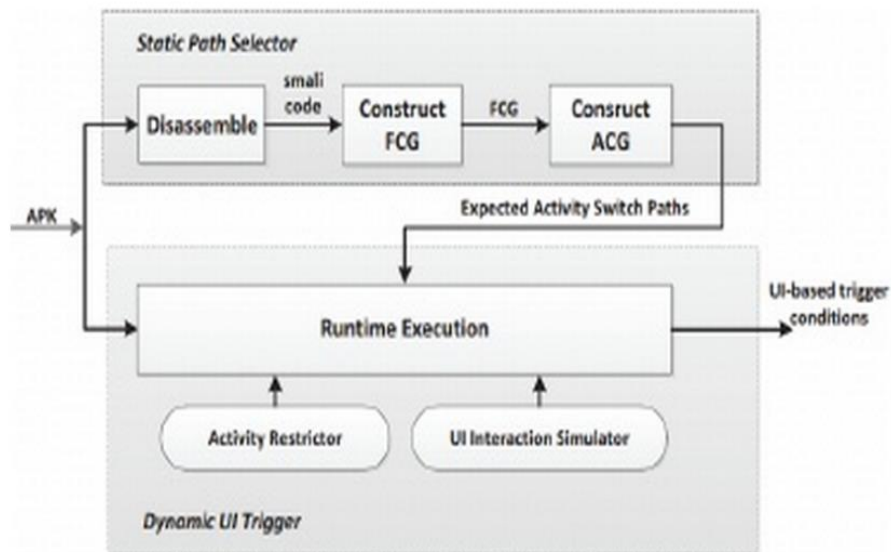


Fig 12: *SmartDroid Architecture*[66]

The above figure 12 explains the SmartDroid Architecture for an android application. Then the dynamic UI[67] is invoked and then traversed each UI element. This is done to reveal privacy sensitive trigger conditions according to the static analysis reports[68] i.e., activities and switching paths.

4.4 Detection Leakage through Cloud based analysis

All the mobile devices have limited memory and are severely restricted in resources. Due to this performing the privacy leak detection on these devices is an issue. So, the researchers came up with a solution and proposed a new cloud based analysis model[71].

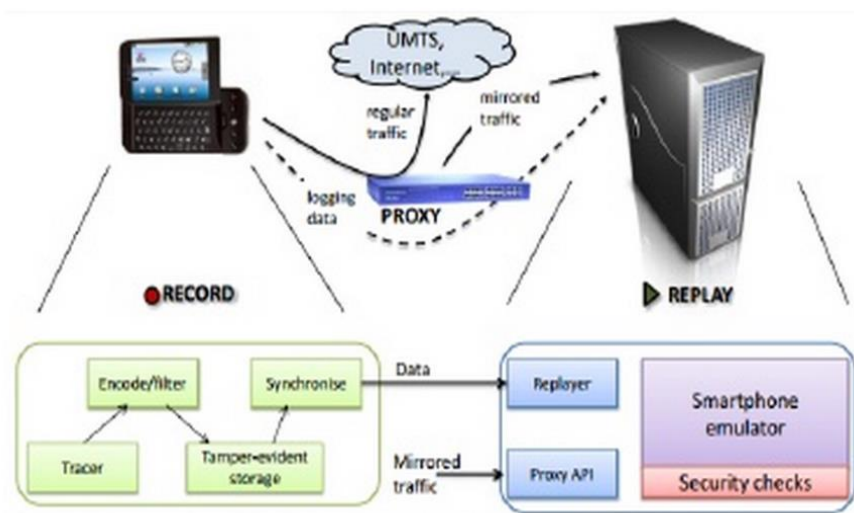


Fig 13: *Paranoid Android Architecture*[71]

The above figure 13 explains the structure of Paranoid Android Architecture[49] in an android device. Architecture of one such technique and tool names Paranoid android was proposed.

Based on this model[73], performing privacy detection on mobile devices has been eliminated. The architecture is illustrated in the above figure. The working of this model is, the cloud[69] includes running synchronized replica of the mobile device on a cloud-based server.

Since the server does not have any memory and mobile device like constraints, the privacy leak detection analysis that would be too complex to run on a mobile device can be performed successfully. In this model there is a Tracer available in

mobile device that collects all the necessary information required to perform all the mobile application executions.

This tracer transmits the information over the encrypted channel to the cloudbased Replayer[70]. It re-executes the application in the smart phone emulator. Then eventually the privacy checks within the emulator can be performed on the server.

4.5 Leak Detection through Kynoid

Recently Daniel Schreckling proposed Kynoid[42], which is real time enforcement of fine-grained, user defined and datacentric security policy proposed in android. It is based on user defined security policies defined for data items stored in shared resources. Core idea of Kynoid is to implement a middleware between application and the data as shown in Fig to provide policy enforcement functionality.

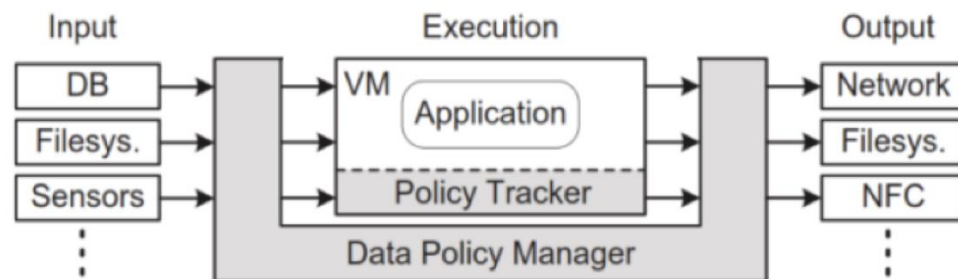


Fig 14: Core idea of Kynoid[61]

The above figure 14 explains the core idea of Kynoid in android architecture. Kynoid is based on TaintDroid to integrate a lightweight policy tracker in

sandboxing mechanism of Android[42]. It tries to make the TaintDroid approach fine grained to support practical permission system, which allows critical and non-critical data.

TaintDroid supports 32 different tags in 32 bits field introduced in shadow variable, which can refer to at most 32 different data sources. Whereas, Kynoid uses this 32 bits field for identifiers, each variable in Android can be assigned with different ID, which again mapped with policy.

It allows Kynoid finer grained tracking by having total 2^{32} mappings for security policies. But this creates tremendous amount of run time and memory overhead, which is addressed by using dependency graph in Kynoid.

Dependency graph is evaluated at sink to derive exact security policy. Kynoid blocks the connections that are leaking information at monitored sinks as per policies defined. Modifying this Dalvik VM for taint tracking and Kynoid system service for policy database and ID mapping does implementation.

For inter-process policy tracking, identifiers of source variables are mapped to the identifiers of destination variable of another app. Sinks are monitored in similar kind of architecture that of TaintDroid to detect information leak.

Kynoid claims to be giving competitive performance on benchmark tests against TaintDroid while providing finer granularity of taint tracking policy, but it exists

only as a prototype implementation. Also, Kynoid needs to analyze impact of indirect flows to the overall performance.

4.6 DroidVulMon

In android the security-related vulnerability for mobile terminal suggests ‘DroidVulMon’ system to detect and respond attacks in order to prevent information leakage caused by malicious app.

The proposed scheme is different from the traditional scheme in the sense that the proposed scheme enables to monitor and detect existence of malicious app for multiple terminals while the traditional scheme supports to detect malicious app for only one mobile terminal.

The proposed scheme, named as DroidVulMon, enables to collect information related to system, service, process and network from multiple terminals so that it can detect rooting attacks and security vulnerability.

4.6.1 Security Architecture for malicious App detection

The Architecture about the proposed system, DroidVulMon[47] can be described as a follow Fig. One of main part is a server which consists of server application and vulnerability checking server. Vulnerability checking server is continuously cooperating with client.

Specifically, the security vulnerability-checking engine of vulnerability server sends or receives data from security vulnerability monitoring engine in client. They exchange data including engine update information and manage verification logic with each other.

Additionally, the server has security-checking DB, which is used for storing the entire data to/from server and the data maintained in DB will be utilized for criteria to estimate whether an app is malicious app or not. As mentioned above, client has vulnerability monitoring engine.

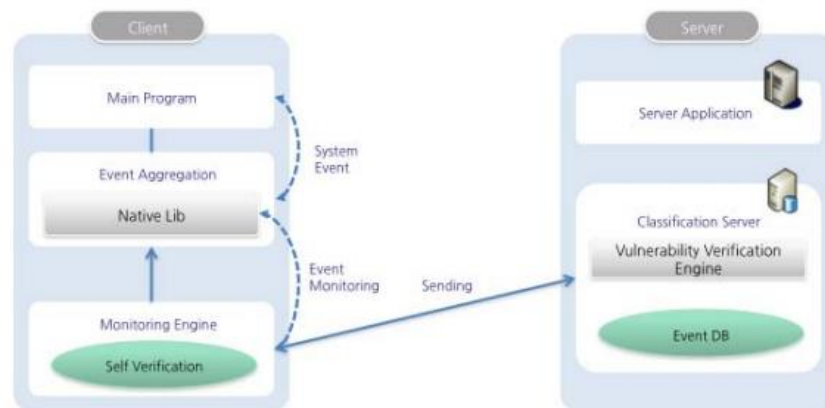


Figure 15: System Architecture[47]

The above figure 15 explains the system architecture of DroidVulMon[47] for an android application. The main purpose of the engine is to verify and protect the app in conjunction with server's vulnerability check engine. If the malicious app is

DroidVulMon aims to detect malicious app and security related vulnerability for 'n' mobile terminals. To meet this goal, the DroidVulMon is designed effectively to share data collected from n mobile terminal by client and server.

The main purpose of the engine is to authenticate and protect the app in conjunction with server's vulnerability check engine. If the malicious app is detected, events will be collected.

Those events will be sent to a native library, which is an agent to detect rooting attack and the native library along with main program keep sharing events, rooting attack monitoring results and information about abnormal app.

4.7 Overall Comparison

The below table 5-I/II explains the comparison of different types of Detection analyses. All the techniques are applied on Android platform except the ProtectMyPrivacy technique, which is for iOS device. Static data flow technique is implemented by most of the approaches like LeakMiner, AndroidLeaks, AppIntent. Whereas Droidtest, IntentFuzzer and TISSA use dynamic data flow technique.

Majority of android apps are tested on AndroidLeaks approach, which is around 25,976. Approximately 57,299 leaks are found in applications; 63.51% leaks are found in ad code. Moreover, 92% leaks are related to phone data, 5.94% leaks are of Location data.

	Platform	Technique	No. of Tested Apps
LeakMiner	Android	Static Flow Data	1,750
PCLeaks	Android	Static intra Component analysis	2,000
DroidTest	Android	Dynamic Data Flow	50
AndroidLeaks	Android	Static Data Flow	25,976
AppIntent	Android	Static data Flow	1,000
IntentFuzzer	Android	Dynamic capability leak	2,183
IccTA	Android	Static intra component Analysis	3,000
TISSA	Android	Dynamic data flow	24
Mobile Forensics of Privacy Leaks	Android	Correlate User actions to leaks	226
Woodpecker	Android	Capability Leaks	953
ProtectMyPrivacy	iOS	Crowdsourcing	685

Table 5: Privacy Leak Detection Framework-I[71]

	Summary
LeakMiner	It is found that 127 app leaks device ID, 50 apps leaks phone info, 27 apps leaks Location and 12 apps leaks contacts.
PCLeaks	Nearly 986 component leaks are found. While 534-activity launch leaks are found. Moreover, broadcast injection leaks are 245 and activity-hijacking leaks are 110.
DroidTest	It is found that most app leaks model number, subscriber ID, Location, Mobile number
AndroidLeaks	Approximately 57,299 leaks are found in applications; 63.51% leaks are found in ad code. Moreover, 92% leaks are related to phone data, 5.94% leaks are of Location data.
AppIntent	It is found that 140 apps have potential data leaks, 26 apps leaks data unintentionally, 24 apps leaks Location, 1 app leaks SMS
IntentFuzzer	It is found that more than 50% of applications leak capabilities or permissions related to network state, Location, internet connection
IccTA	It is found that 425 applications leak information directly. These leaks are related to Device and Location data
TISSA	It is found that 14 apps leak Location and 13 leaks device ID.
Mobile Forensics of Privacy Leaks	It is found that 9 different kinds of data is leaked by applications, 34 apps leaks data due to user actions on widgets 14 leak on Location.
Woodpecker	Explicit capability leaks are found in trustworthy applications.
ProtectMyPrivacy	It is found that 48.43% applications access identifier of device, 13.27%access locations, 1 app leaks contacts.

Table 6: Privacy Leak Detection Framework-II [71]

The above table 5-I explains the statistics and comparison of different types of Detection analyses Least number of apps was tested on TISSA and DroidTest approaches, which are 24 and 50 apps respectively.

It is found that 14 apps leak Location and 13 leaks device ID through the TISSA approach whereas it is found that most app leaks model number, subscriber ID, Location, Mobile number through the DroidTest approach.

Dynamic capability leak technique is applied on IntentFuzzer and number of apps tested was 2,183. It is found that more than 50% of applications leak capabilities or permissions related to network state, Location, internet connection.

Capability Leaks are applied on Woodpecker testing in total 953 apps. Explicit capability leaks are found in trustworthy applications. Crowdsourcing is applied on ProtectMyPrivacy is tested on 685 number of applications. It is found that 48.43% applications access identifier of device, 13.27% access locations, app leaks contacts.

Static intra component Analysis is applied on IccTA testing on 3000 apps. It is found that 425 applications leak information directly. These leaks are related to Device and Location data.

Static intra Component analysis is applied on PCLeaks with tested on 2000 apps. Nearly 986 component leaks are found. While 534-activity launch leaks are found. Moreover, broadcast injection leaks are 245 and activity-hijacking leaks are 110.

CHAPTER 5

Android App – Track Me

5.1 Location Tracking Applications

In this chapter, we discuss about apps that records the GPS coordinates and shares them to the dear ones. There are bunch of applications available in play store, which works on these GPS.

The below table 6 gives the comparison between the Location sharing apps available in the app store. The table compares the apps with the type of language developed, type of webserver used and databases. The table also talks about the pros and cons of the apps.

All the apps developed on Android uses either Java or C# as programming languages for development. Usually most of the apps are developed on Java because development on Java is more feasible. Java doesn't less configuration system for development.

A common database used for app development in android is SQLite. By the nomenclature it conveys as a Lite database. It's not like a usual database. Since mobile has very less memory when compared to the desktop applications.






	 Family Locator	 Friend Location Finder	 Friend Finder	 Family Locator	 Friends Finder
Database	SQLite[51]	SQLite	H2 DB	SQLite	SQLite
Language	Java	Java	C#	Java	Java
Webserver	GCM Http[52]	GCM Http	GCM Http	GCM Http	GCM Http
Pros and cons	<p>Pros: Networking app which also includes: 1.Location Sharing 2.Circle 3.Places 4.Premium</p> <p>Cons: 1.Needs Internet compulsory on phone to track. 2.Need Smartphone</p>	<p>Pros: Shares device ID to establish connection and receives location updates.</p> <p>Cons: 1.Needs Internet compulsory to track. 2.Need Smartphone</p>	<p>Pros: Very perfect & accurate in giving the location updates.</p> <p>Cons: 1.App can only send updates to 2 contacts which is limited. 2.Not on iOS, need Internet to run this application. Need Smartphone</p>	<p>Pros: Networking app which also includes: 1.Location Sharing 2.Circle 3.Message sharing</p> <p>Cons: 1.Need Internet to run this application. 2.Need Smartphone.</p>	<p>Cons: 1.Need Internet to run this application. 2.Need smartphone. 3.Difficult to understand the Interface</p>

Table 7: Comparison of different Location sharing apps on app store

5.2 Case Study

Sygy GPS Application

The Sygy GPS application[50] allows the users to take pictures using an Activity developed in house, instead of reusing the regular Android camera application.

To do this, the Activity called CameraActivity first registers a callback using the Camera.takePicture function. The system triggers the callback function when the picture is captured and attaches the actual byte array. It then calls setResult function and finish, sending the raw picture to the caller.

The Activity has no intent-filter because the exported attribute is not set, the default value is set to false. The Activity could only be exploited by another application signed by the same developer, so then we classify it as low risk.

As of now, none of the other applications of the same developer currently in the play store seem to invoke this Activity function, but this may change in the future. This type of vulnerability is difficult to detect statically because the source is not in the application code. The application passes a function to the camera API and the operating system calls that function with tainted parameters.

The Intent passes through a message queue, from where it is forwarded to the correct application handler. Identification of this vulnerability was possible because we analyze the application together with the Android libraries and manually added a rule to Permission Flow that marks the function that distributes the Intents to handlers as having a tainted Intent parameter.

5.3 Personalized Android app – Track Me

This section discuss about an android app that is developed and entitled “Track Me” records your Geographical Longitudinal and Latitudinal location coordinates from the app using the GPS[54].

Initially, the app has to set a mobile number to someone whom you want to share your location coordinates. And then once the app is triggered, it sends the GPS coordinates to that corresponding mobile number through the app[53].

Functionality

The app sends geographical coordinates along with the address and a Google Maps link[54]. So, the app users may need not to have a smart phone on both sides. It works perfectly even if only the sender has a smartphone and the receiver does not have one.

This app keeps on send the geographical coordinates to the receiver in the form of text message for every constant interval of time. Once the app is triggered, it keeps running in the background and sends the text message to the receiver until we hit the stop command in the app.

5.3.1 Home Screen

Initially, the app is supported for only android devices. The device requires minimum OS Android 2.2 running on android devices.

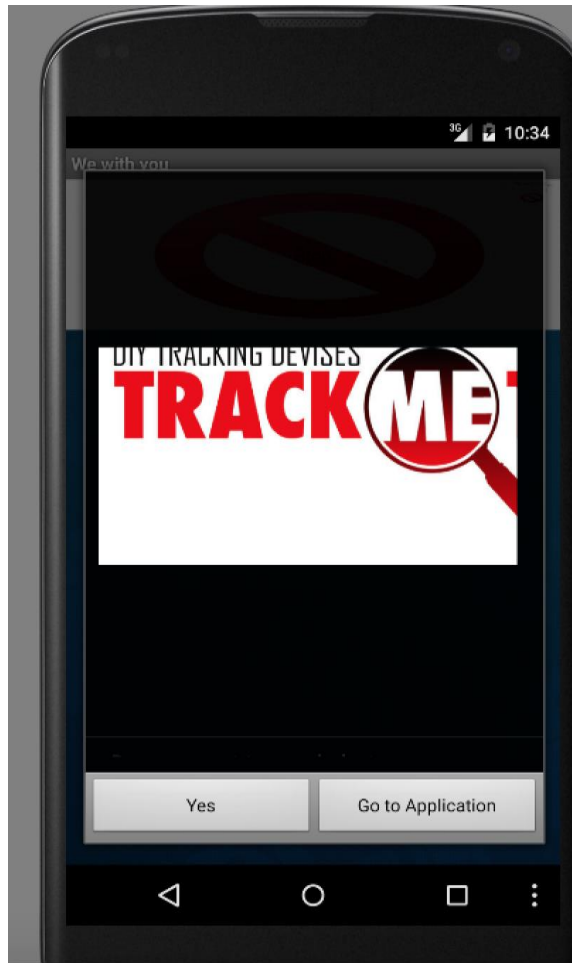


Fig 16: *Home Screen of app*

The above figure 16 is the welcome screen for the app. Whenever the user triggers the app from the device, this screen pops up asking to start newly or continue to send the text messages to the previous number that has been set on the device.

Either you can select “Go to Application” or “Yes”. If the user hits the “Go to Application” button then the app, it redirects to the new contact page to add a new contact number to which the geographical coordinates updates are to be sent.

5.3.2 Go to Application



Fig 17: *Go to Application*

The above figure 17 shows a screen that pops up when the user hits the “Go to Application” button during the welcome screen. If the user wants to set a new number to which the coordinates has to be sent then hits the contacts button.

5.3.3 Contacts

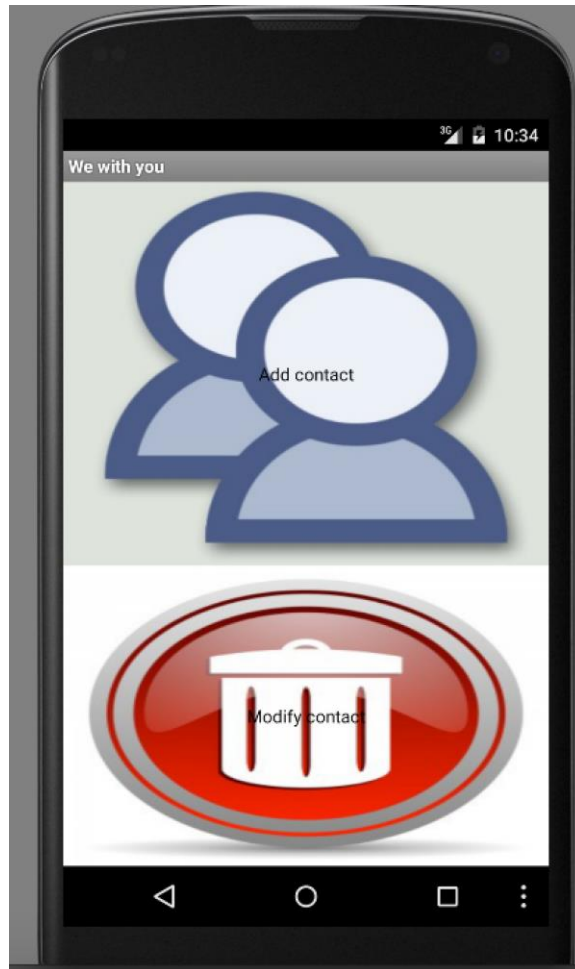


Fig 18: *Adding a Contact*

The above figure 18 shows a screen to add a new contact to whom the user wants to send the updates of his/her location. A new contact can be added or an existing contact can be chosen.

5.3.4 Choosing a Contact

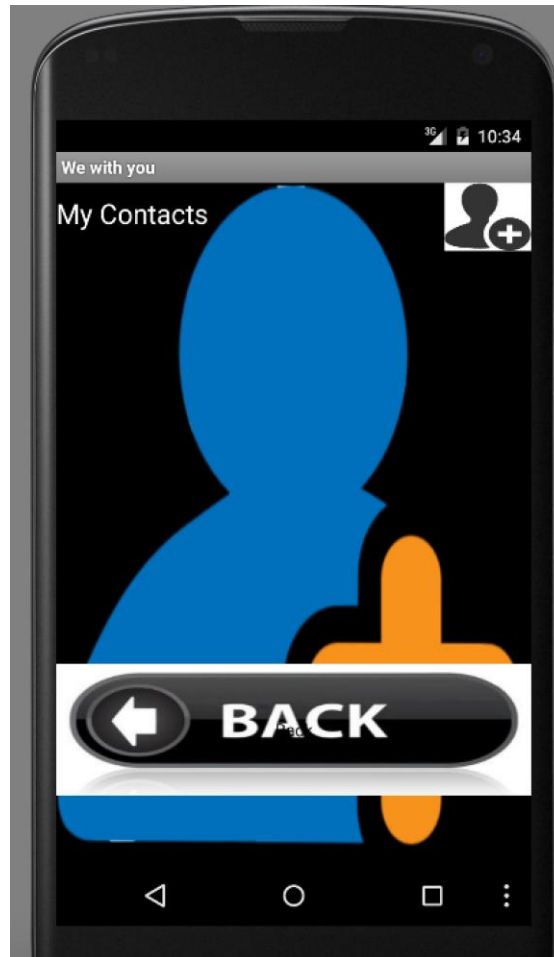


Fig 19: *Choosing a Contact*

The above figure shows a screen to pick a contact from the phone directory to whom the user has to share the location coordinates and after choosing the contact, the user hits back button to return to Main menu.

5.3.5 Start Updating



Fig 20: *Start Updating*

The above figure 20 shows a screen saying the location updating has been started and it keeps sending the text messages to the number, which we have set in the app. The app keeps on sending text messages to the receiver for every two minutes of interval.

5.3.6 Text Message Notification

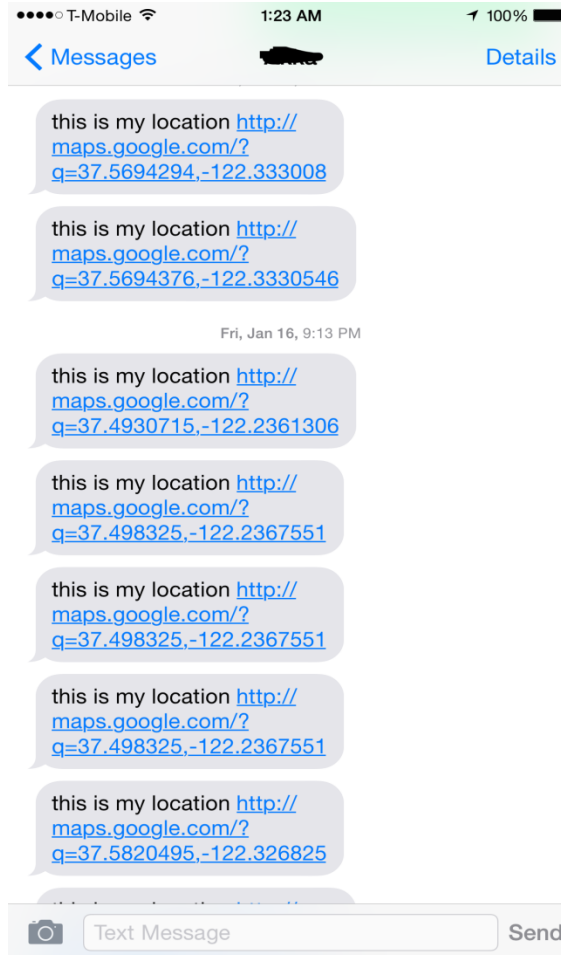


Fig 21: *Text Message Notification to receiver*

The above figure 21 shows a format of the text notification[53] sent to the receiver when the app is triggered from the sender mobile. The message contains a link which when pressed redirects to the exact location on the Google Maps.

5.3.7 Google Map View

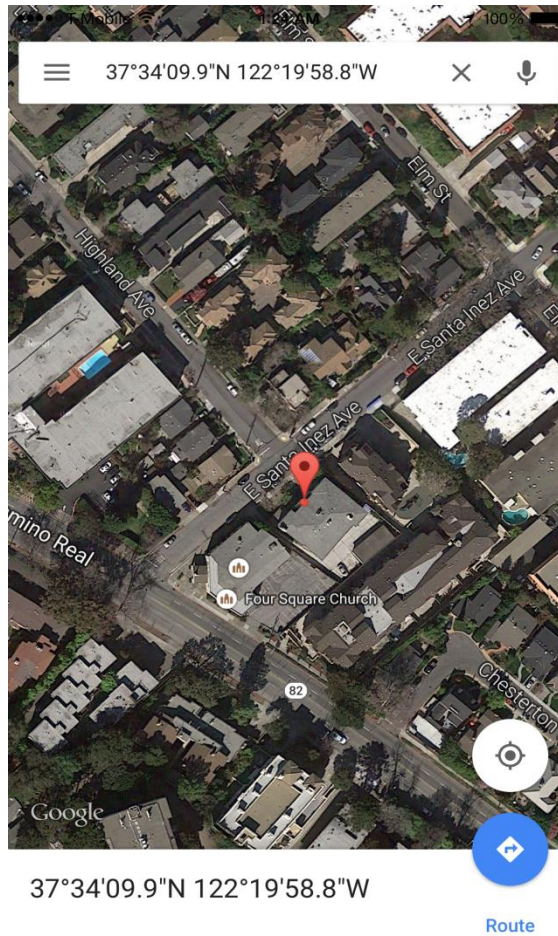


Fig 22: *Google Map View*

The above figure 22 shows a screen, which is redirected, to the Google Maps[54] when the link in the text message [53] is pressed. By default the app navigates to the Google Maps and pins your exact location.

CHAPTER-6

RECOMMENDATIONS FOR SECURE APPLICATIONS

The best and efficient advice for security-aware Android developers is to pay close attention to the configuration of their application (particularly, any combination of parameters listed without additional checks, vulnerable). If such a parameter combination is needed for functionality reuse or other constraints, here are some ways of maintaining security:

A better approach is to always request an explicit user confirmation for the invocation of any Activity that may be part of an inter-application flow and the user should be informed to which caller the information will be sent.

This method has the disadvantage that it decreases the ease-of-use of the application and to enforce that callers of your Activities own certain permissions, developers can use either declarative permission requirements in the application manifest and dynamic permission checks using `checkPermission` calls.

All the developers should consider using work-around for sending sensitive information over inter-component boundaries. For example, several of the applications analyzed leak information from an ordered set of items such as GPS data, contact names/phone number or zip code.

For these kind of applications there is no need for complex mechanisms to avoiding the vulnerability. It may be sufficient to return an integer index to the information database, instead of the actual information and the caller would need to query the database to obtain the actual information.

Passing sensitive information over the inter-component boundaries of the same application in an encrypted form is recommended to protect against unintended callers, but it does not help if an attacker has compromised another application with which the current application shares the user id.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Many android applications have access to other types of sensitive information that are not protected by standard Android permissions. In order to protect this kind of information, developers should define custom permissions to the apps, but because of the coarse-grained nature of the custom permissions (assigned to applications as opposed to APIs) it is not possible to automatically identify the taint sources for such information. The method is completely automated and it is based on coupling rule-based static **taint analysis** with automatic generation[63] of rules that specify how permissions can leak to unauthorized applications. Most mobile operating systems are included in this category and can benefit from the proposed new application of taint analysis[64].

BIBLIOGRAPHY

- [1] Android version history "http://en.wikipedia.org/wiki/Android_version_history"
- [2] "Android device activation numbers reach 1 billion worldwide" Phandroid.com. September 3, 2013. Retrieved [November 2, 2013].
- [3] Elgin, Ben (August 17, 2005). "Google Buys Android for Its Mobile Arsenal". *Bloomberg Businessweek*. Bloomberg. Archived from the original on February 24, 2011. Retrieved [February 20, 2012].
- [4] "A History of Pre-Cupcake Android Codenames". Android Police. September 17, 2012. Retrieved [October 20, 2013].
- [5] "Dianne Hackborn". Google+. September 1, 2012. Retrieved April 8, 2013.
- [6] "Dan Morrill". Google+. January 2, 2013. Retrieved January 5, 2013.
- [7] Breeze, Mez. "The designer behind the logo". *TheNextWeb.com* (TNW). Retrieved [August 14, 2013].
- [8] "Google Launches Android, an Open Mobile Platform". Google Operating System. [November 5, 2007].
- [9] "Live Google's gPhone Open handset alliance conference call"(transcript). Gizmodo. November 5, 2007. Retrieved [February 8, 2013].
- [10] "Google releases Android SDK". Macworld. November 12, 2007. Retrieved [February 8, 2013].
- [11] "Android's 5th Birthday Celebration: European Best-of-Best Hackathon Series". Devfest.info. October 2012. Retrieved [January 5, 2013].
- [12] "SDK Archives". *developer.android.com*. Retrieved [March 7, 2015].
- [13] "Android 0.5, Milestone 3—the first public build - The history of Android". Ars Technica. June 16, 2014. Retrieved [March 7, 2015].
- [14] "Android: the first week". Android Developers Blog. November 16, 2007. Retrieved [January 24, 2013].

[15] "Life can be tough; here are a few SDK improvements to make it a little easier". Android Developers Blog. December 14, 2007. Retrieved [January 24, 2013].

[16] "Android SDK m5-rc14 now available". Android Developers Blog. February 13, 2008. Retrieved [January 24, 2013].

[17] "Announcing a beta release of the Android SDK". Android Developers Blog. August 18, 2008. Retrieved [January 24, 2013].

[18] "Android 0.9, Beta—hey, this looks familiar! - The history of Android". Ars Technica. June 16, 2014. Retrieved [March 11, 2015].

[19] "Announcing the Android 1.0 SDK, release 1", Android Developers Blog. September 23, 2008. Retrieved [January 24, 2013].

[20] "Android 1.0—introducing Google Apps and actual hardware - The history of Android". Ars Technica. June 16, 2014. Retrieved [March 11, 2015].

[21] Android flaw puts personal data at risk for millions

“http://www.computerworld.com/article/2900845/android-flaw-puts-personal-data-at-risk-for-millions.html?phint=newt%3Dcomputerworld_data_management&phint=idg_eid%3Db724b568c8ffdb142ea04b55824060f3#tk.CTWNLE_nlt_data_mgmt_2015-03-25”

[22] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege Escalation Attacks on Android Information Security." vol. 6531, M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, Eds., ed: Springer Berlin / Heidelberg, 2011, pp. 346-360.

[23] A. Lineberry, D. L. Richardson, and T. Wyatt. (2010, Dec 29). *THESE AREN'T THE PERMISSIONS YOU'RE LOOKING FOR*. Available: <http://dtors.files.wordpress.com/2010/09/blackhat-2010-final.pdf>

[24] MoutazAlazab, VeelashaMoonsamy Lynn Batten and RonghuaTian, Patrik Lantz “Analysis of Malicious and Benign Android Applications” 2012 32nd International Conference on Distributed Computing System Workshops

[25] P. lantz. (2011). *Project 5 - DroidBox: An Android Application Sandbox for Dynamic Analysis*. Available: <http://www.honeynet.org/gsoc/slot5>

[26] Sandbox, ”<http://www.slideshare.net/anushatuke1/android-sandbox>”

- [27] A. Shabtai, U. Kanonov, Y . Elovici, C. Glezer, and Y . Weiss, "“Andromaly”: a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, pp. 1-30,[October, 29 2011].
- [28] Android Security, <http://developer.android.com/training/articles/security-tips.html>
- [29] Taenam Cho, Jae-Hyeong Kim, Hyeok-Ju Cho, Seung-Hyun Seo, Seungjoo Kim , Vulnerabilities of Android Data Sharing and Malicious Application to Leaking Private Information
- [30] ContentProvider, <http://developer.android.com/intl/ko/reference/android/content/ContentProvider.html>.
- [31] Android Permission, <https://android.googlesource.com/platform/frameworks/base/+/master/core/res/AndroidManifest.xml>
- [32] Dalvik, [http://ko.wikipedia.org/wiki/%EB%8B%AC%EB%B9%85_\(%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4\)](http://ko.wikipedia.org/wiki/%EB%8B%AC%EB%B9%85_(%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4)).
- [33] Smali, <http://code.google.com/p/smali/>.
- [34] Android Proguard, <http://developer.android.com/tools/help/proguard.html>.
- [35] Siyuan Ma, Zhushou Tang, Qiuyu Xiao, Jiafa Liu, Tran Triet Duong, Xiaodong Lin, Haojin Zhu” Detecting GPS Information Leakage in Android Applications” Globecom 2013 - Communication and Information System Security Symposium
- [36] T. Watson, “T.j. watson libraries for analysis,” [http://wala.sourceforge.net/wiki/index.php/Main Page](http://wala.sourceforge.net/wiki/index.php/Main_Page).
- [37] Bhushan Lokhande, Sunita Dhavale, “Overview of Information Flow Tracking Techniques Based on Taint Analysis for Android”
- [38] Zhemin Yang and Min Yang, “LeakMiner: Detect Information Leakage on Android with Static Taint Analysis”, In Software Engineering (WCSE), 2012 Third World Congress on, pp.101–104, 2012

- [39] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, Anmol N. Sheth. "TaintDroid: An InformationFlow Tracking System for Real-time Privacy Monitoring on Smartphones" 9th USENIX Symposium on Operating Systems Design andImplementation (OSDI' 10) 2010.
- [40] Android Security Overview,
<https://source.android.com/devices/tech/security/index.html> [Oct. 24,2013]
- [41] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall, "These Aren't the Droids You're Looking For",Retrofitting Android to Protect Data from Imperious Applications In Proc. of ACM CCS, [October 2011]
- [42] Daniel Schreckling, Johannes Kostler, Matthias Schaff, "Kynoid: Realtime enforcement of fine-grained, user-defined, and data-centric security policies for Android", information security technical report 17, pp.71-80, 2013
- [43] Zhibo Zhao and F.C.C. Osono, "Trustdroid: Preventing the use of smartphones for information leaking in corporate networks through the used of static analysis taint tracking". In Malicious and Unwanted Software (MALWARE), 7th International Conference on, pages 135– 143, 2012
- [44] Christian Fritz, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Jacques Klein, Alexandre Bartel, Yves le Traon, Damien Ochteau, Patrick McDaniel, "Highly Precise Taint Analysis for Android Applications", EC SPRIDE Technical Report. Nr. TUD-CS-2013-0113. [May, 2013]
- [45] FlowDroid Now Supports Implicit Flows,
<http://sseblog.ecspride.de/2013/10/flowdroid-implicit-flows/Oct.01,2013>
[Oct.25,2013]
- [46] ZheMin Yang, Min Yang "LeakMiner: Detect information leakage on Android with static taint analysis", 2012 Third World Congress on Software Engineering.
- [47] You Joung Ham, Hyung-Woo Lee, Jae Deok Lim, Jeong Nyeo Kim, "DroidVulMon - Android based Mobile Device Vulnerability Analysis and Monitoring System", 2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies
- [48] "A framework for static detection of privacy leaks in android applications",
<http://dl.acm.org/citation.cfm?id=2232009>

- [49] Geogios Portokalidis, Philip Homburg, Kostas, Herbert Bos “Paranoid Android: versatile protection for smartphones”, <http://dl.acm.org/citation.cfm?id=1920313>
- [50] Sygic GPS Application, “<https://www.sygic.com/gps-navigation/features>”
- [51] Android Database,
<https://developer.android.com/reference/android/database/sqlite/package-summary.html>
- [52] Android server,
<http://developer.android.com/reference/com/google/android/gcm/server/package-summary.html>
- [53] SMS in Android,
<http://developer.android.com/reference/android/telephony/SmsManager.html>
- [54] GPS in Android,
<http://developer.android.com/reference/android/location/GpsStatus.html>
- [55] Seung-Hyun Seo, Dong-Guen Lee, Kangbin Yim, “ Analysis on maliciousness for mobile application,” IMIS 2012, pp.126-129, 2012. 7.
- [56] C. Gibler, J. Crussell, J. Erickson, and H. Chen, “Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale,” *Trust and Trustworthy Computing*, pp. 291–307, 2012.
- [57] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, “Riskranker: scalable and accurate zero-day android malware detection,” in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 281–294.
- [58] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, “Unsafe exposure analysis of mobile in-app advertisements,” in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2012, pp. 101–112.
- [59] J. Dean, D. Grove, and C. Chambers. Optimization of objectoriented programs using static class hierarchy analysis. In Proc. ECOOP’95, pages 77–101, 1995.

- [60] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, Anmol N. Sheth. "TaintDroid: An Information- Flow Tracking System for Real-time Privacy Monitoring on Smart phones" 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI' 10) 2010.
- [61] Daniel Schreckling, Johannes Kostler, Matthias Schaff, "Kynoid: Real- time enforcement of ne-grained, user-dened, and data-centric security policies for Android", information security technical report 17, pp.71- 80, 2013
- [62] Zhibo Zhao and F.C.C. Osono, "Trustdroid: Preventing the use of smartphones for information leaking in corporate networks through the used of static analysis taint tracking". In Malicious and Unwanted Software (MALWARE), 7th International Conference on, pages 135– 143, 2012
- [63] Christian Fritz, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Jacques Klein, Alexandre Bartel, Yves le Traon, Damien Oceau, Patrick McDaniel, "Highly Precise Taint Analysis for Android Applications", EC SPRIDE Technical Report. Nr. TUD-CS-2013-0113. May, 2013
- [64] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen, "AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale", Proceeding TRUST'12 Proceedings of the 5th international conference on Trust and Trustworthy Computing pp.291-307, 2012
- [65] Golam Sarwar (Babil), Olivier Mehani, Rokhsana Boreli, Mohamed-Ali Kaafar, "On the Effectiveness of Dynamic Taint Analysis for Protecting Against Private Information Leaks on Android-based Devices", SECURE, 10th International Conference on Security and Cryptography 2013
- [66] Cong Zheng, Shixiong Zhu, Shuaifu Dai, Guofei Gu, Xiaorui Gong, Xinhui Han, Wei Zou "SmartDroid: an Automatic System for Revealing UI-based Trigger Conditions in Android Applications"
- [67] S. Dienst and T. Berger. Mining interactions of android applications static analysis of dalvik bytecode. Technical report, Department of Computer Science, University of Leipzig, Germany, May 2011. Technical Note.
- [68] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In Proceedings of the 9th USENIX conference on Operating systems design and implementation, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

- [69] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis “Paranoid Android: Versatile Protection For Smartphones”
- [70] T. Leblanc and J. Mellor-Crummey. Debugging parallel programs with instant replay. *IEEE Transactions on Computers*, 36(4):471–482, 1987.
- [71] Muhammad Haris, The Hong Kong University of Science and Technology ”Privacy Leakage in Mobile Computing: Tools, Methods, and Characteristics ”
- [72] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, “Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications,” in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2012, pp. 93–104.
- [73] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, “Paranoid android: versatile protection for smartphones,” in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 347–356. s
- [74] William Enck, Peter Gilbert, Byung-gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and A. N. Sheth, "TaintDroid: An Information- Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *proceeding of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)*, Vancouver, Canada 2010.

VITA
Graduate College
University of Nevada, Las Vegas
Srinivas Kalyan Yellanki

Degrees:

Bachelor of Technology in Computer Science, 2013

Jawaharlal Nehru Technological University

Master of Science in Computer Science, 2015

University of Nevada Las Vegas

Thesis Title: A Survey on Potential Privacy Leaks of GPS information in Android Applications

Thesis Examination Committee:

Chair Person, Dr. Ju-Yeon Jo, Ph.D

Committee Member, Dr. Yoohwan Kim, Ph.D

Committee Member, Dr. Ajoy K. Datta, Ph.D

Graduate College Representative, Dr. Venkatesan Muthukumar, Ph.D