KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Privacy Implications of Android Applications in the Presence of Third-Party Libraries

Roger Bernat Ribas Manero

Thesis submitted for the degree of
Master of Science in
Electrical Engineering, option
Electronics and Integrated Circuits

Thesis supervisor:
Prof.dr.ir. Bart Preneel
Prof.dr.ir. Marian Verhelst

Assessor:
Prof.dr.ir. Claudia Diaz

Mentor:
Michael Herrmann

Academic year 2013 – 2014

Preface

"Hoy es siempre todavía" - Antonio Machado
"Fent i desfent, aprèn l'aprenent." - Ferran Boixadós Escalé

After an academic year of hard work, my master thesis is finally finished. With it, I conclude my Master Degree and, therefore, a stage of my live. This time has been an amazing trip of auto-discovery. There were moments were I had to face several difficulties that completely overtook me but I always came up with a solution. I will always remember that sweet moment in which your written code works as expected.

I would like to thank the KU Leuven for offering me the opportunity of doing my Master Thesis with them. Specially, I would like to thank Prof. Claudia Diaz who accepted my application from the very beginning. Also, my advisor Michael Hermann from whom I have learned a lot and without whom I could never have reached this point. Furhter, my advisor in Spain Prof. Jordi Forné for helping and guiding me from the distance.

I would like to thank my colleges from the COSIC department Marc Juárez and Günes Acar for their help and interest. Furthermore, I would like to thank all the people from the COSIC.

I would like to thank my family and my girlfriend Veronica for all their support during this year, backing me up every time I could not find a way to continue.

Finally, I would like to thank all the incredible people I have met during this year. Specially, my room-mates Xavi Garcia, Laura Forriol and Josu Saetero and my good friend Esra Tiger.

Thank you all!
Moltes gràcies a tots!
Dank u allen!

Roger Bernat Ribas Manero

Contents

Abstract

Mobile devices, such as smart phones and tablet computer, become increasingly popular. One reason that makes these devices so popular, is the availability of mobile applications that allow to add a wide range of functionality to the operating system of a mobile device. Recently, concerns have been raised with respect to the private information that these mobile applications are able to access. Usually, privileges can be granted to a mobile application in order to limit the data they have access to. However, since mobile applications generally do not run if not all privileges are granted, users tend to grant all necessary privileges. From a privacy perspective, this is particularly an issue, because a mobile application usually consists of several parts: One part written by its developers and several external libraries written by third parties.

In this work, we present *3Ov*, a tool that analyses the source code of mobile applications in order to identify the privileges necessary for the mobile application itself and the privileges that are necessary for the third party libraries. The ultimate goal of 3Ov is to find permissions that needs to be granted to a mobile application solely because the third party library requires it. We provide the evaluation of 89 applications of a proof of concept. Our results show that external libraries increase the privileges that need to be granted to the application. Since this means that users allow potentially many parties to access their private data, we discuss protecting possibilities on the example of location data. We propose *LPDroid*, a tool that enables to embed location privacy protecting mechanisms in the operating system itself. In particular, LPDroid allows the user to choose the level of protection and the operating system obfuscates the user's location data automatically to all mobile applications accessing location data.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

AVDM	Android Virtual Device Manager
ADT	Android Development Tools
API	Application Programming Interface
APK	Android Package
AOSP	Android Open Source Project
DDMS	Dalvik Debug Monitor Server
DVM	Dalvik Virtual Machine
GPL	General Public License
GPS	Global Positioning System
GUI	Graphic User Interface
HAL	Hardware Abstraction Layer
IMEI	International Mobile Station Equipment Identity
IMSI	International Mobile Subscriber Identity
IPC	Inter Process Communication
JNI	Java Native Interface
LBA	Location Based Applications
LBS	Location Based Service
LIFO	Last In First Out
LPPM	Location Privacy Preserving Mechanism
LSBS	Location-Sharing-Based Service
MITM	Man In The Middle
NDK	Native Development Kit
NMEA	National Marine Electronics Association
OOM	Out Of Memory
OSN	Online Social Network
POI	Point Of Interest
QoS	Quality of Service
RPC	Remote Procedure Calls

SDK	Software Development Kit
SSL	Secure Socket Layer
TLS	Transport Layer Security
URI	Uniform Resource Identifier

Chapter 1

Introduction

1.1 Motivation

As of May 2014, there are 6.91 mobile subscriptions in the world, equivalent to the 95.5% of the world's population[1]. By the end of 2014, 1.75 billion people are expected to be using smart-phones, mobile phones with enhanced capabilities and improved connectivity[2]. Every mobile phone necessitates the use of an operating system (OS). Concretely, smart-phones rely on Android (79% of smart-phones), iOS (14.2%), Windows Phone (3.3%) and Black Berry (2.7%). Smart-phones' popularity can be mostly contributed to their better hardware and the growing use of mobile applications. These applications can be included within the OS or can be developed by the device vendor, third-party companies and/or amateur developers. Usually, every OS has its own *market* where developers can upload their applications and make them available, free or at a certain price, to allow users to download, install and use them. Further, the success of an OS mainly depends on its applications market. With more than 1 200 000 applications as of May 2014[3], the Google Play market is the leader market. Almost 1 000 000 of these applications are free to be downloaded from the market[4]. Indeed, for every paid application downloaded on the market, users download 82 free applications [1].

When downloading an Android application from the Google Play market, users first have to allow the application certain permissions. Only then, may users download and install it. As it will be explained later, these permissions can provide applications with access to users' sensitive data. Related work has already shown that Android applications declare further permissions in their manifest file to allow third-party libraries access sensitive data [5, 6, 7, 8]. However, our work tries to relate this tendency to declare further permissions with the different categories of

[1] Mobile phones statistics website: `http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a\#subscribers`

[2] Online article about smart-phones popularity: `http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536`

[3] Android applications statistics: `http://www.appbrain.com/stats/number-of-android-apps`

[4] Android applications statistics: `http://www.appbrain.com/stats/free-and-paid-android-applications`

third-party libraries. For the moment, we will refer to this permissions with the term *over-privileges*.

Naturally, application developers need to find some way to generate revenue, if they are not directly charging for their product. Running in parallel to the growing popularity of smart phones is the ever growing number of mobile advertising business. These businesses, including the likes of AdMob, Millenial Media, InMobi, Chartboost etc. provide advertising libraries which include different mechanisms through which application developers can embed advertising banners in the devices' user interfaces during the application run-time. We will refer to these libraries as third-party advertising libraries. Developers use these third-party advertising libraries to generate revenues through the usage of their applications. Stevens et al. [2] hypothesize that the more precise the target of the advertisement is, the more personalized the advertisement will be and, hence, the more revenue for both the developer and the advertising company. So far, all we can say is that it is likely that the more personalized an advertisement is, the more likely it is for a user to buy the advertised product. Therefore, advertising companies can ask for higher prices to the contractors and hence the developers can generate more revenue. Advertising libraries are not the only third-party libraries used by application developers. Development tools libraries as well as Social Software Development Kits (SDKs) are also common in the applications. The former allow Android developers to enhance their applications functionality by extending the Android Application Programming Interfaces (APIs). The latter allow developers to interact with Online Social Networks (OSNs). For instance, a browser-type application could use a library that managed HTTP-type transactions such as Apache. For another example, a game-type application that wanted to share users' score with their friends on Facebook could use the Facebook SDK. Although at first sight we could think that Development tools libraries are not likely to gather information from the users, it would make sense to think that Social SDKs might do it. The fact that, for the last years, several cases of personal data trading between OSNs, governments and companies have been uncovered, illustrates this point [3, 4].

In order to conduct such personalized advertising, advertising companies, and therefore developers, need to gather information from their users. For the rest of this work, we encompass the gathered information within the term *sensitive data* which includes: Users' location, contacts list, phone number, passwords, messages and devices' International Mobile Station Equipment Identity (IMEI), International Mobile Subscriber Identity (IMSI), phone ID number etc. Third parties can then access this data and send it to third party servers for user profiling or personalized advertisement targeting. For instance, a third-party advertising library can determine the location of a certain user and, therefore, advertise shops, restaurants and other establishments in the surrounding area.

If applications have been shown to access users' sensitive data and send it to third-party servers, it would be necessary to provide a tool to protect users against such an act. The protection of the plethora of information that is considered to be sensitive was infeasible due to time constraints. Consequently, we decided to focus on one the location data privacy as location coordinates potentially reveal

many important information about an individual. For the last years, Location Based Services (LBSs) have become more and more popular. Through them, it is possible for users to have access to certain services related to their current location. The use of LBSs has increased lately due to embedded Global Positioning Systems (GPS) chips in smart-phones and the popularity of wireless data connections. Examples of LBSs include mapping applications (e.g. Google Maps), GPS navigation (e.g. TomTom) and location-aware social networks (e.g. FourSquare). The growing exposure of users' location data leads towards a series of privacy issues. For instance, location data can be used to infer individual's home or work location, sexual preferences, political views, religious inclinations, etc.

1.2 Objectives, Scope and Main Conclusions

Firstly, we will introduce *Third Party Over-privileges* (*3Ov*), a tool that statically analyzes Android applications for finding over-privileges. Additionally, it also searches for the presence of unnecessary and missing permissions. The former are those permissions that are granted to the application but are not required by any of its method calls. The latter are those permissions that should be granted to the application but are not. The implementation of 3Ov required to be aware of all the third-party libraries from the different categories: *Development tools libraries*, *Advertising libraries* and *Social SDKs*. Hence, we started out with the lists from the Appbrain project[5] and included in 3Ov a tool to find new third-party libraries present in the analyzed applications. Further, 3Ov requires the usage of *PScout*, a tool that analyzes the Android source code and maps all the methods to their required permission. 3Ov will identify the method calls in the analysed applications and will try to link these method calls to the permissions they require according to PScout.

Secondly, we will introduce *Location Privacy Droid* (*LPDroid*), an extension of *TaintDroid* that provides users with further mechanisms to protect their location data. LPDroid introduces a set of tools that allow users to communicate with Android's location framework and modify its behaviour for the sake of users' location data protection. It is meant to pave the future research for the implementation of Location Privacy Preserving Mechanisms (LPPMs) in Android devices as it will be addressed in Chapter 6.

As there is not an official definition of the concept *over-privilege*, in this work we provide five different heuristics and analyse them with 3Ov. As a proof of work, we inspect a set $\mathcal{B}$ of 89 applications for the presence of over-privileges considering each heuristic. In all the situations, 3Ov identifies the presence of over-privileges in the analysed applications. To the best of our knowledge, this is the first time, a work relates the presence of over-privileges with the use of Advertising libraries, Social SDKs and Development tools libraries.

[5]Statistics about Android third-party libraries:
`http://www.appbrain.com/stats/libraries/overview`

With the implementation of LPDroid we show that the correct place to implement LPPMs is the Android location framework rather than in any other level of the Android OS stack. Further, we introduce the necessary mechanisms to allow Android developers to implement different LPPMs and allow users to control their behaviour. Moreover, we provide three dummy mechanisms that can be considered as an example of that procedure.

The rest of this document is organized as follows:

Chapter 2 is dedicated to review the related work. This encompasses the previous work that has been conducted on the presence of over-privileges in Android applications as well as the leakage of sensitive data by third-party libraries. Further, it covers the previous studies on the field of location privacy.

Chapter 3 provides the theoretical background related to Android's internals and Android applications necessary to understand the following chapters.

Chapter 4 introduces 3Ov, a static analysis tool that seeks for over-privileges in Android applications, and presents the outcome of its analysis. The tool consists in 6 different steps which will be covered in different sections: crawling the Google Play market, downloading the applications, parsing Android's permissions, reverse-engineering the applications and generating Android Manifest type files.

Chapter 5 is devoted to introducing LPDroid, an extension of TaintDroid that provides users with further capabilities to protect their location data, and its implementation on a real device. Firstly, we will introduce the current tools available in the Google play market to protect users' location data. Secondly, in front of the drawbacks presented by those applications, we will go deeper into the Framework level. Finally, we will analyse the lowest layers of the Android OS to check whether it can influence user's location privacy or not.

Chapter 6 presents the future work that can be considered as an extension of this thesis. Firstly, we will consider all the enhancements that can be carried out to improve 3Ov. Secondly, we will introduce a way to improve the implemented mechanisms through the implementation of a mechanism to provide *geo-indistinguishability*.

Chapter 7 presents the main conclusions of the thesis.

Chapter 2

Related Work

In this chapter we will discuss the previous work that concerns our research. To this end, we will divide the chapter into two different sections. In Section 2.1, we will address all the work related to analysing the Android Open Source Project (AOSP) as well as the applications from the Google Play market. In Section 2.2, we will focus on the work dedicated to the study of location privacy.

2.1 Privacy Implications of Android Applications

Adrienne et al. [5] introduce Stowaway, a tool that detects over-privileges in Android applications. They determine the set of API calls that an application uses and then maps those API calls to permissions. Stowaway was tested in 900 applications and it identified 323 (35.8%) over-privileged applications. In front of those results, Adrienne et al. justify the tendency to over-privilege applications by a series of common developer errors: mistaken permission names, deputies, related methods, copy and paste, deprecated permissions, testing artefacts and so on. Our work extends this approach in the way that it distinguishes between when these over-privileges are likely to happen due to developer errors (such as the ones mentioned before) and when they are likely to happen due to the presence of third-party libraries.

Another approach is the one followed by Theodore et al. [6] in which they relate the presence of privileges with the Advertising Libraries. They reconstruct the API for 103 advertising libraries and study how the privacy leaking API calls from the top 20 libraries are used. Our work extends theirs by considering also Development tools libraries and Social SDKs. Felt et al. [7] manually analyse 956 Android applications seeking for over-privileges and analyse their permissions to evaluate whether application permissions are effective at protecting users' sensitive date. Their results indicate that application permissions can have a positive impact on system security when applications' permission requirements are declared up-front by the developer. Vidas et al. [8] introduce a tool that allows developers fulfilling the least privilege principle by performing an over-privilege analysis on their applications source code and informing the user about the presence of over-privileges. In a different approach, Theodore et al. [11] investigate changes over time in the behavior

of Android Advertising libraries observing how most of the guaranteed permissions increase over the years. For instance, the `INTERNET` permission has increased from a presence of less than a 10% in 2008 to more than a 60% in 2012.

Grace et al. [9] develop AdRisk, a platform to systematically identify potential risks in Android applications, ranging from uploading sensitive information to remote servers to executing untrusted code from Internet sources. Equally, Enck et al. [10] apply the Fortify Java tool with the same goal. Both show the leakage of sensitive information by third-party libraries.

Against this increasing tide of over-privileged applications, Shashank et al [12]. propose Flow Permissions, an extension to the Android permission mechanism. It allows users to examine and grant permissions within an application during the run-time. For instance, if a user does not want to share her location with the application, she can block the `ACCESS_LOCATION` type permission. In the same direction Ilias et al. [13] present a privacy protection framework that aims to achieve an equilibrium between the developer's revenue through advertising libraries and the user's privacy. The proposed framework is based on the establishment of a feedback control loop that adjusts the level of privacy protection on mobile phones, in response to advertisement generated revenue. They present a similar approach to the one introduced by Stevens et al. [2] and implement a framework to limit the amount of sensitive data that applications can access. Shashi et al. [14] propose AdSplit, an extension of Android that allows applications and their advertisements to run as separate processes eliminating the need of applications to request permissions on behalf of their advertising libraries.

A part from that, Kathy et al. [15] present PScout, a tool that extracts the permission specifications from the Android source code. Basically, PScout relates all the API calls from the Android OS to their required permissions. For example, it relates the `getLastKnownLocation()` method with an `ACCESS_LOCATION` type permission. The Android documentation is far away of being complete. There are lots of undocumented and internal APIs. PScout is aware of that and includes the methods from these APIs in its analysis.

Enck et al. [16] develop TaintDroid, an extension of Android capable of tracking sensitive data leakage. Finally, Hornyack et al. [17] introduce AppFence an extension of TaintDroid that implements two privacy controls that (1) replace sensitive data by obfuscated data and (2) block network transmissions that contain sensitive data. Further information about TaintDroid and AppFence can be found in Section 5.2.

2.2 Location Privacy

A great variety of LPPMs have been proposed in the literature. One big group is the one that encompass the obfuscation-based LPPMs. As shown in [18], obfuscation-based LPPMs cover various methods that reduce the accuracy and/or precision of the events' spatiotemporal information: *location hiding*, *perturbation*, *adding dummy regions* and *reducing precision*. The *location hiding* consists of not sending the location data. Some LPPMs propose to change pseudonyms after a period of location

hiding [21, 22, 23]. LPPMs that rely on *perturbation* submit a location different from user's actual location [24]. The *adding dummy regions* strategy [25] consists of submitting fake locations along with the user's actual location. Finally, with the *reducing precision strategy* [26, 27] a user would send a cloak region that contains her current location, but she would not reveal her precise whereabouts. The key limitation of this approach is that it does not distinguish between the information revealed to trusted parties and to the service provider. Hence, in order to protect their location information towards the service provider, users must lower the quality of location information shared with their trusted parties.

In front of that drawback, other approaches for implementing LPPMs have been proposed such as *differential privacy* and privacy based on cryptographic primitives. Dong et al. [28] propose to use proxy-encryption, which guarantees that the service provider is not able to learn the location update and, furthermore, that the ciphertext can be modified by the service provider such that only intended receivers are able to successfully decrypt. Zhong et al. [29] propose three protocols for friend nearby notification. In this setting, users are notified if a friend is in close proximity. Bilogrevic et al. [30] propose two protocols to allow users to calculate a fair rendez-vous point in a privacy preserving manner. Finally, some works apply Private Information Retrieval (PIR) in which the users retrieve information (e.g., points of interest) related to their surroundings [31, 32]. PIR could in principle be employed to build privacy-preserving LSBS.

Shokri et al. [19] propose a method to find the optimal LPPM for a LBS given each user's service quality constraints against an adversary implementing the optimal inference algorithm. Such LPPM is the one that maximizes the Expected Distance Error that the optimal adversary incurs in reconstructing the actual location of a user, while fulfilling the user's Quality of Service (QoS) requirement. This strategy is extended by Hermann et al. [20] by considering bandwidth constrains. Their results show that the maximum value of privacy a user can enjoy can be reached by either sufficiently relaxing the quality loss or the bandwidth constraints, or by choosing an adequate combination of both constraints.

In [18], Shokri et al. provide a formal framework for LPPM evaluation. This framework considers the prior information an attacker may have and the various attacks she can perform. Further, they clarify the difference between three aspects of the adversary's inference attacks: their *accuracy*, *certainty*, and *correctness* (Expected Distance Error). They show that the latter determines the privacy of users.

There are four accepted metrics to measure the notion of location privacy: *Expected Distance Error*, *k-Anonymity*, *Differential Privacy* and *Transformation-based approaches*. *Expected Distance Error* states that the expected estimation error of the adversary is the metric of users' location privacy. *k-Anonymity* states that the user's location should be indistinguishable among a set of k points. *Differential privacy* is a notion of privacy inherited from the world of Databases. It aims to provide means to maximize the accuracy of queries from statistical databases while minimizing the chances of identifying its records. Concretely it requires that the probability that a query returns a value v when applied to a database D, compared to the probability to report the same value when applied to and *adjacent* database

D' should be within a bound of e^{ϵ}. As shown in the studies introduced in future work [36] and also in [37], *differential privacy* has also been applied in the context of location privacy. Finally, *Transformation-based approaches* claim that the users' location should be completely invisible to the service provider. This is achieved by transforming all data to a different space, usually employing cryptographic primitives.

Chapter 3

Background

In this chapter we introduce the basic Android concepts necessary for understanding the following chapters of the thesis. In Section 3.1, we introduce the basic concepts of the Android OS software stack. Section 3.2 is entirely dedicated to introduce the user level.

3.1 Internals of the Android Operating System

Android is an operating system designed for hand-held systems such as smart-phones or tablet computers. It was set up by Andy Rubin in October 2003 and was later acquired by Google in 2005. The first version of Android, Android 1.0, was released in 2008 running on the HTC Dream device. As of May 2014, 1.5 million of Android phones were activated every day, up from 1.3 million in January 2013, 400 000 in June 2011 and 200 000 in August 2010. The most important feature of Android is its open source nature. That means its source code is available for free and can be modified. The difference with other open source projects such as The Linux Foundation, Debian or Python amongst others is that Android is developed by a closed group, without help from external groups or independent developers.

The basic structure of the Android software stack distinguishes between four different levels: (1) The Linux Kernel level, (2) the Libraries/Run-time level, (3) the Application Framework level and finally (4) the Applications level. This is however a coarse separation of the stack. Therefore, we will introduce the stack presented by Karim [33] shown in Figure 3.1. In blue, it represents all the Java-coded components of the stack while in green, it represents the C/C++-coded part. Finally, in yellow, it represents the applications included in the AOSP and all the other applications. In the following paragraphs we will introduce each level of the stack.

Although an extended knowledge of the Linux kernel is not required to understand this work, there is an important concept the reader should be aware of. Android developers modify the *vanilla* Linux kernel (kernels from The Linux Kernel Archives[1]) to meet Android devices requirements. The kernel version used in Android devices is not the latest version of the Linux kernel (mainline) but a long-term release

[1]Linux Kernel Archives website: `https://www.kernel.org/`

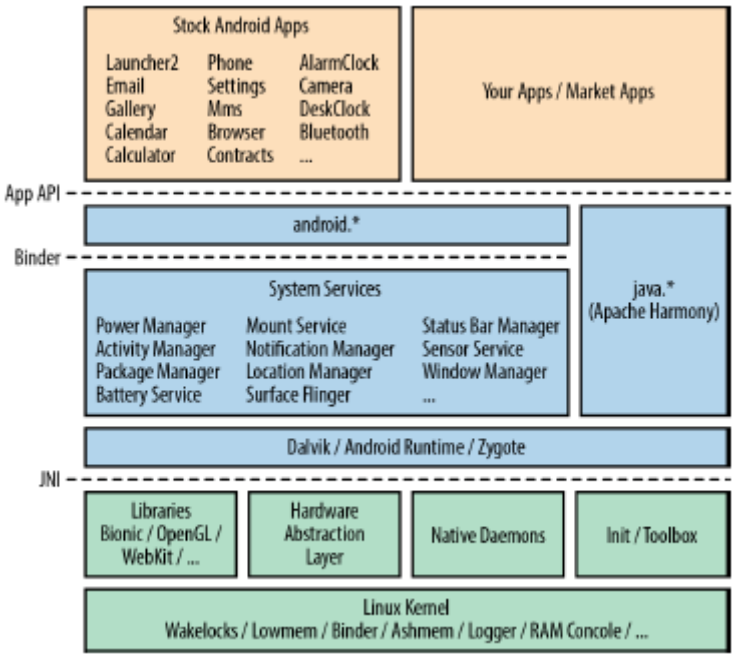


Figure 3.1: Complete Android Software Stack diagram as proposed by Yaghmour Karim.

(stable version). For instance, Android 4.4.2 is based on the kernel version 3.4.91 while the mainline as of May 2014 was the 3.15-rc5 . The higher levels of the stack wouldn't work unless they were running on an *"Androidized"* kernel. "Androidized" kernels typically contain several hundred patches over the standard kernel. There are six main *"androidizms"*: (1) Wakelocks, (2) Low-Memory Killer, (3) Binder, (4) Anonymous Shared Memory, (5) Alarms and (6) Logger.

Android relies on a Hardware Abstraction Layer (HAL) level. This layer contains what are called *hardware abstractions* which are sets of software routines that emulate some platform-specific details, providing applications with access to the hardware. The AOSP normally includes hardware support code for a few devices. These are generally those that were used by Google to develop new Android releases (e.g., Samsung Nexus S, Crespo, for Gingerbread or Galaxy Nexus, Maguro, for Jelly Bean). One of the main features of relying in a HAL layer is that the license under which the driver is distributed is up to the manufacturer. Hence, a device manufacturer can create a basic driver that implements the minimum mechanisms to control a certain component and make that driver available under the General Public License (GPL). The GPL is the most used free software license which guarantees users the freedom to use, modify and share the software. However, enhanced functionalities would be implemented within a proprietary shared library in the user space and will not be available publicly. This is because only the Linux kernel and the *vanilla* AOSP are available under the GPL. No matter which mechanism is used to load the drivers, a system service, included in the AOSP, corresponding to the type of hardware is typically responsible for loading and interacting with them. For instance,

the GPS device will be controlled by the Locations service. That system service will be responsible for interacting and coordinating with the other system services, such as the WiFi and the Telephony services, to make the hardware behave coherently with the rest of the system and the APIs provided to application developers.

The Native User space encompass all the user space components that run outside the Dalvik Virtual Machine (DVM, see next paragraph). This includes a few programs that run natively on the CPU such as the Init/Toolbox, the Native Daemons and the Native Libraries. Android relies on approximately 100 dynamically loaded libraries, all stored in the `/system/lib` directory. A great number of these libraries come from external projects. They were ported Android in order to make their functionality available within the Android software stack. For instance, BlueZ (support for the core Bluetooth layers and protocols), OpenSSL (toolkit for implementing Secure Socket Layer [SSL] and Transport Layer Security [TLS] protocols as well as a general purpose cryptography library), SQLite (for SQL databases support) database amongst others are included.

The DVM is a virtual machine specially designed for running on a (1) slow CPU, (2) little RAM, (3) no swap memory and (4) battery powered devices. In Android systems, each application is assigned his own DVM instance. Moreover, the DVM is also the interpreter of the Java Native Interface (JNI) which allows inter-programming-language communication. JNI is fundamentally a bridge between Java and C/C++. Internally, the AOSP depends on JNI to facilitate Java services and components to work with Android's lower levels that are written in C/C++. Java system services, for example, very often use JNI to communicate with matching native code that interfaces with a given service's corresponding hardware. A large part of the procedure that allows Java to communicate with other languages through JNI is done by the DVM.

Android comes with a number of system services that are constantly running and available for developers. These system services include the Location service, Sensor service, WiFi service, Telephony service, Bluetooth service and so on. System services are started at boot time and are guaranteed to be running by the time an application is launched.

3.2 Android Mobile Applications Fundamentals

An Android application is an installable unit which may or may not run independently of other Android applications. By this, we mean that Android applications can, in principle, interact with other applications, by sharing data for example, but usually run in an isolated way from other applications. Applications consists of different Android software components and resource files. Concretely, there are four different components that can be used to create an application: (1) *Activities*. (2) *Intents*, (3) *Broadcast Receivers* and (4) *Content Providers*.

Applications are usually written in Java using the proper SDK which includes the Android API. This SDK is freely available at the Android developers website and is can be loaded in several Integrated Development Environments (IDEs). Furthermore,

it includes the Android Development Tools (ADT). Applications can also be extended with C/C++ programs thanks to the Native Development Kit (NDK).

3.2.1 Development Environment

The Android IDE can be downloaded from the Android website and includes all the necessary tools to start developing Android applications. When installing it in our working environment (CentOS 64 bits machine), we had to install the 32 bit libraries as well. It can be done with the command `sudo apt-get install ia32-libs`.

The ADT basically includes the Android Virtual Device Manager (AVDM) and the Android SDK Manager. The former allows developers to create virtual devices which can be used to test applications in an emulator. The latter, is used to keep the SDK updated with the latest APIs. Furthermore, the ADT includes the Dalvik Debug Monitor Server (DDMS) perspective which includes several tools for application analysis and testing. For instance, developers can see the state of the heap at run-time with the Heap tool, or the network state through the Network Statistics tool. Further information that are beyond the scope of this work can be found in the developers official website[2].

3.2.2 Application basic components

In this section we will in detail elaborate on the four Android application components. Further, we will introduce the Graphic User Interface (GUI) as well as the permissions system of the applications.

Activities

An activity is an application component that represents a GUI which covers the whole screen. Applications can be formed by more than one activity but there must always be a *main* activity that will act as a start screen. For example, a game application can have one activity to log in a certain user account, another activity to choose the settings and another activity for playing. Each time a new activity starts, it is pushed onto a *back stack* (LIFO stack) and takes user focus. The previous activity is stopped, but the system preserves it in the back stack.

Each activity has its own life-cycle. Android developers, have no control over in which state their applications are. However, developers can be notified when the state is going to change thanks to a set of call-back methods. Figure 3.2 from Android developers website shows the complete life-cycle of an activity.

- **onCreate():** this method is called when the activity is initiated for the first time.
- **onStart():** is called just before the activity loads the GUI on the screen.

[2]Official Android developers website: `http://developer.android.com/`

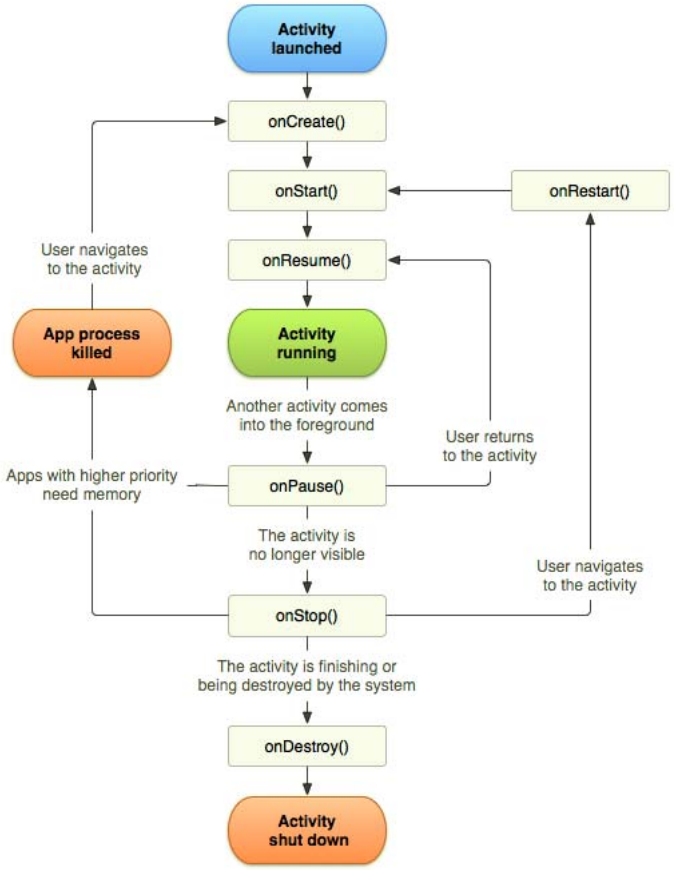


Figure 3.2: Activity's life-cycle from Android developers website.

- **onResume():** is called when the activity can start interacting with the user. Usually, sounds or animations are being started in this state.
- **onPause():** is used when the activity is going to be pushed to the background. Here is where the application state needs to be saved.
- **onStop():** used when the activity is not visible to the user anymore and is not going to be visible for a while.
- **onRestart():** called when the activity is no longer visible to the user, because another activity has been resumed and is covering this one.
- **onDestroyed():** Called before the activity is destroyed. This is the final call that the activity will receive.

Intents

An intent is a messaging object that developers can use to request an action from another application component. This other component can either belong to the same

application or to another one. There are three different cases in which is feasible to use intents. For the scope of this work only the one case is important. Developers can start a new activity by passing an intent to the `startActivity()` method. This intent can carry information in order to deliver it to the started activity. For instance, in a game-type application, when the user clicks the options button to open the options menu, an Intent object indicating which activity should be started, is passed to the `startActivity()` method as an argument.

There are two types of intents:

- Explicit intents name the component to which they are referring. They are normally used to start a component within the application such as a new activity.

- Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another application to handle it.

Broadcast Receivers

A broadcast receiver is an Android component which allows developers to register for system or application events. All registered receivers for an event are notified by the Android run-time once this event happens. For example, applications can register for the `ACTION_BOOT_COMPLETED` system event which is fired once the Android system has completed the boot process.

Content Providers

Content providers manage access to a structured set of data such as users' contacts list. They encapsulate the data, and provide mechanisms for securing it. In Chapter 5 we will explain how certain permissions can be queried by the content providers in order to access their content. Content providers allow Android applications to access data which belongs to other applications.

Content providers follow the client/server scheme. When developers want to access data in a content provider, they use the ContentResolver object to communicate with the provider as a client. The provider receives data requests from clients, performs the requested action, and returns the results.

Android itself includes several content providers that manage different data such as audio, video, images, and personal contact information.

Layout

A layout defines the visual structure for a GUI, such as the one for an activity or application *widgets* such as buttons. Layouts can be declared in two ways: statically or dynamically. Android offers the possibility to use either or both of them for declaring and managing application's GUIs.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.INTERNET" />
    ...
</manifest>
```

Figure 3.3: Permission section of the Android Manifest file from an application that requires access to the Internet.

- Statically: Declare GUI elements in XML. AOSP's View classes and sub-classes elements, such as those for widgets (buttons, images) and *layouts* (predefined structures that define the layout of an activity), can be referred in XML language.

- Dynamically: Instantiate layout elements at run-time. Applications can create View class objects and variate their properties at run-time.

Android applications are, however, composed of more than just code. They may require resources that are separate from the source code, such as images, audio files etc. One of the most important aspects of providing resources separate from the source code is the ability for developers to provide alternative resources for different device configurations. For instance, it is easy to develop an application in different languages as only the *string* resource, where all the text is written, needs to be provided in the different languages.

AndroidManifest and applications permissions system

The central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would impact other applications, the operating system, or the user. This includes reading or writing the user's *sensitive data* such as contacts or emails, reading or writing another application's files, performing network access, keeping the device awake, and so on. This approach is known in Android as *sandboxing*

Nevertheless, applications may need to share resources and data. They do this by declaring the permissions they need for additional capabilities not provided by the basic sandbox. Applications statically declare the permissions they require, and the Android system prompts the user for consent at the installation-time. Android has no mechanism for granting permissions at run-time because, as they claim, it complicates the user experience to the detriment of security.

A basic Android application has, by default, no additional permissions and thus it cannot access users' data or have control over some hardware devices. To make use of protected features of the device, developers need to include in their applications' AndroidManifest.xml file the `<uses-permission>` tags declaring the permissions the application needs. Figure 3.3 provides an example on how to access the `android.permission.INTERNET` permission.

At the installation-time, the permissions requested by the application are granted to it by the package installer. The package installer is the application that Android uses to install applications. This application, interactively asks the users to accept all the permissions declared in the application prior installing it. No checks with the user are done while an application is running; the application is either granted a particular permission when installed, and can use that feature as desired, or the permission is not granted and any attempt to use the feature fails without prompting the user.

3.2.3 Building an application

The building process of an application is shown in Figure 3.4 from the Android developers website. During the build process, Android projects are compiled and packaged into an Android Package (`.apk`) file, which contains the application binary. Further, it contains all of the information necessary to run the application on a device or emulator: compiled `.dex` files (`.class` files converted to Dalvik byte code), a binary version of the AndroidManifest.xml file, compiled resources (`resources.arsc`) and uncompiled resource files.

When developing Android applications with Eclipse, the ADT plug-in automatically builds the project as new changes are made to the source code. Eclipse creates an `.apk` file and signs it with a digital signature that identifies the developer. Android requires all the installed applications to be signed with a certificate whose private key is held by the application's developer. Android uses the certificate to identify the developers.

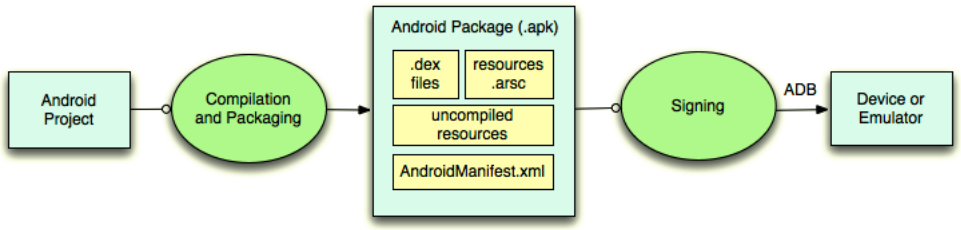


Figure 3.4: Building process of an Android application as shown in the Android developers website.

Chapter 4

3Ov: Identifying Over-privileges in Android Applications

In this chapter, we introduce the *Third Party Over-privileges* (*3Ov*) tool, a tool that performs a static analysis of Android applications by inspecting their source code in order to investigate its granted permissions. 3Ov identifies the privileges that are granted to Android applications due to the usage of third party libraries and not because the application requires them. 3Ov also evaluates the increase of permissions granted to the application due to the usage of third-party libraries. Furthermore, 3Ov identifies privileges that are actually not required by the application or third party libraries, e.g. falsely granted by the developer, and missing permissions.

3Ov is capable of crawling and downloading all currently available applications on the Google Play market. After downloading an application, 3Ov starts its inspection. It therefore firstly reverse engineers the application using the *apktool* and secondly, identifies the application for its AOSP permissions by parsing and analysing the source code. Therefore, our tool uses the PScout tool [15] to link the application's Android API calls to the privileges required by the application. Figure 4.1 shows the work flow of 3Ov.

Providing a formal definition of the term *over-privilege* is not an easy task as it depends on the individual's point of view. For instance, an advertising library can be considered as unnecessary by an application user but can be extremely important for the developer. Therefore, we will define five different heuristics and we will thoroughly seek for over-privileges in Android applications according to them. As a proof of work, we analyze a set $\mathcal{B}$ of 89 applications and identify over-privileged applications when considering all the five heuristics.

4.1 The Google Play Market

Android's store was launched in August 2008 as the Android Market. On March 2012, however, with the merging of the Android Market and Google Music, the store was renamed Google Play. As mentioned in the introduction, with more than 1 200 000 applications, it is the most popular applications market in the world.

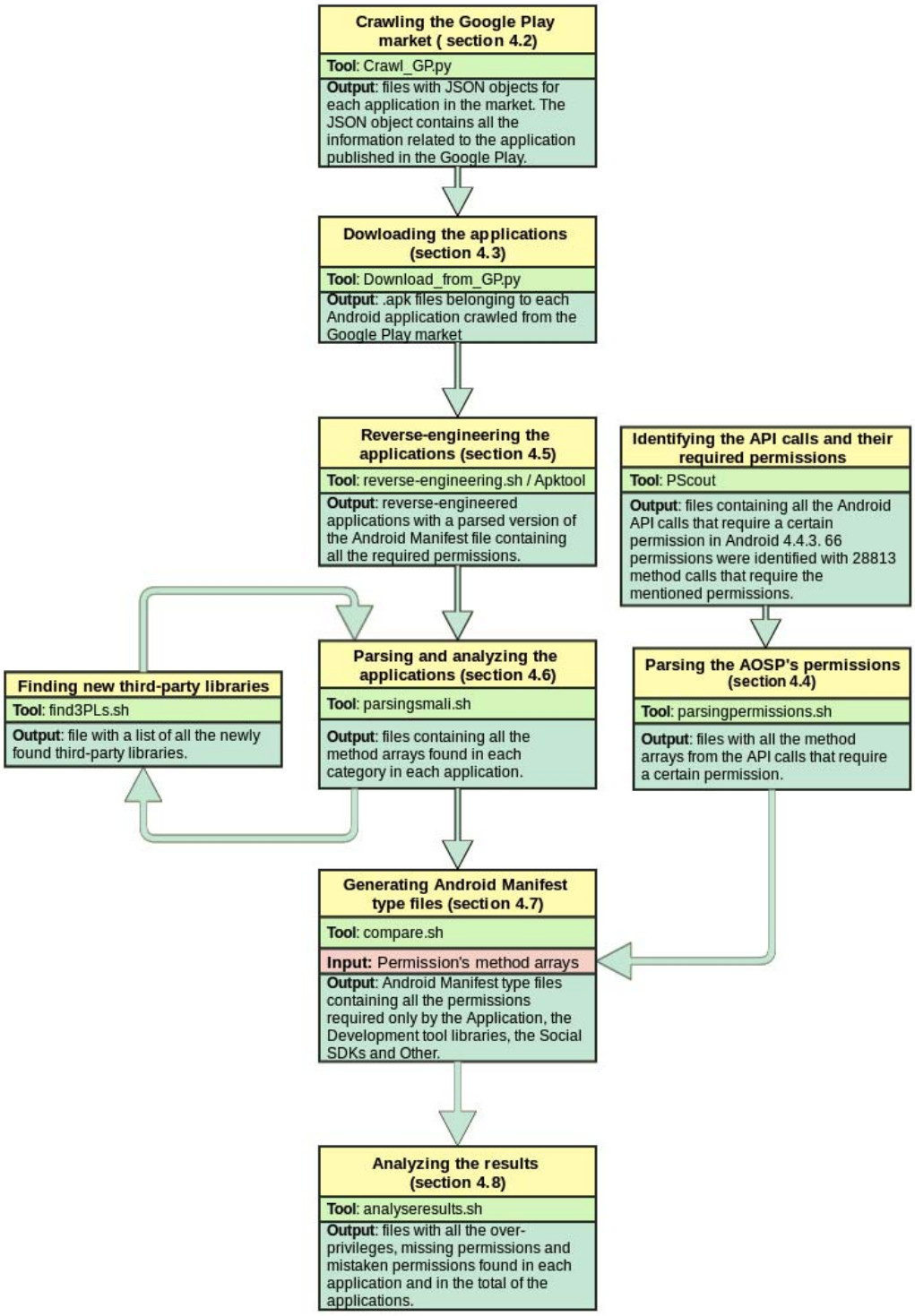


Figure 4.1: Flow chart of 3Ov.

As of May 2014, its catalogue was divided into six different categories: music, books, newsstand, movies and TV, games and applications. For our work, we are only interested in the last two categories. Android applications are divided into 26 categories such as Business, Education and Photography. Android games are divided into 20 categories such as Arcade, Casino and Sports. Please note that the Google Play is different in every country and, hence, some of the official categories may not exist in a certain country and/or additional categories may be found.

Once a certain application or game category is selected, consumers can choose between seven lists of products. Concretely, consumers can choose between the Top Free, Top Paid, Top Grossing Free, Top Grossing Paid, Top New Paid, Top New Free and Trending. Once an application from one of the lists is selected, several information about the mentioned application can be accessed. Concretely, users can access the URL, the rating, the minimum version of Android that is required, its name, the number of reviews etc.

4.2 Crawling the Google Play Market

In order to automatically download applications from the Android Play store we adjusted a Python program published by Anuvrat Singh [34]. Therefore, we firstly ported the program to the Python version 2.6.6, which is the version available in our working environment. Secondly, we removed unnecessary steps in the program, such as crawling for the meta-data of the application's price. Finally, we crawled the Google Play for free applications only. We provide the source code of the program in the Appendix A.1.

The crawling process downloaded 7.8 MB of data and took in total around 11.5 hours. We downloaded the information of 19 335 applications from the Google Play market belonging to 30 different application categories such as Business, Entertainment, Medical and Photography. Please note that the Google Play market may be different at every country. Therefore, the applications available in Belgium can be different from the applications that are available in another country.

Similarly to Singh's code, our modified version stores the crawled information from each application in separate files, depending on its category, in the JSON format as shown below. JSON stands for JavaScript Object Notation and is a standard format for transmitting readable data consisting of attribute-value pairs. For instance, in the example below, `"app_url"` is the attribute and `"https://play.google.com/store/apps/details?id=com.facebook.orca"` is its value.

```
1 {"app_url": "https://play.google.com/store/apps/details?id=com.facebook.orca", "rating": "
    4.22878", "operating_system": "Verschilt per apparaat", "title": "Facebook Messenger", "
    reviewers": "3343878", "downloads": "100.000.000 - 500.000.000", "content_rating": "
    Volwassen: medium", "date_published": "27 februari 2014", "developer_link": "/store/apps/
    developer?id=Facebook", "category": "Communicatie", "badge": "Topontwikkelaar", "
    developer": "Facebook"}
```

For our next step, to download the applications, we only required the URL attribute of an application. From all the created text files, we extracted the URL,

by applying the following bash command:

```
1 cat $permissionfile | grep http | grep -shoP 'http.*?[">]'
```

4.3 Downloading the Applications

The next step consisted in downloading the `.apk` packages. To this end, we firstly wanted to take advantage of the *Apk Downloader* website[1]. The website offers an online, unofficial tool where users can insert the URL of an application and it will automatically generate a download link from which the `.apk` file can be downloaded. However, our first approach of using the Apk Downloader turned out to be inefficient because: (1) every three consecutive downloads users are automatically redirected to an advertising website and (2) the amount of downloads per day is limited (not per user but in total). Please notice that the daily amount of downloads is not disclosed.

Therefore, we used the Chrome browser extension *Apk Downloader*[2] which allowed us to download the `.apk` files without limitations. The extension is automatically activated when a Google Play's site is accessed by showing an icon at the end of the browser's search bar. When the user clicks the icon, the download of the `.apk` file related to the application starts.

The problem with this extension was that we could not manually press the button each time we wanted to download an application. Hence, we modified the extension to automatically start the download process once the website was accessed. Google Chrome extensions are coded in Java Script, HTML or CCS languages. Once programmed, they are compressed into `.crx` files making their source code accessible by changing the extension to `.zip`. In the case of the Apk Downloader, the source code consisted of several Java Script files. We modified the `backgroud.js` file to automate the download process. We defined the function `autoDowloadApk()` which checks whether the URL in the search bar matches the scheme followed by the URLs of Google Play applications' and calls the `download()` function of the APK Downloader extension. Figure 4.2 and 4.3 show the source code of the modifications in the `backgroud.js` file.

Once the extension was properly modified, a program to automate the download process using the modified version of the Apk Downloader extension was written. The program was written in Python in order to use the Selenium library. Selenium is a testing framework for web applications that provides tools for automating websites. Appendix A.2 shows the source code of the program that automates the download process.

[1] Apk Downloader website: `http://apps.evozi.com/apk-downloader/`
[2] Apk Downloader Chrome extension website: `https://chrome.google.com/webstore/detail/apk-downloader/obhlfmheblhjhkmacldlhdnbgbaiigba`

```
function autoDowloadApk(details){
   var tabId = details.tabId;
    var tabUrl = details.url;
   console.log("this is a log"+ tabUrl);
   var match = /play\.google\.com\/store\/apps\/details\?(?:|.*&)id=([\w\d\.\_]+)/i.exec(
       tabUrl);
    if (match) {
       MarketSession.download(match[1], tabId);
    }
}
```

Figure 4.2: This function automatically initiates the download process.

```
chrome.webNavigation.onHistoryStateUpdated.addListener(autoDowloadApk, urlFilter);
```

Figure 4.3: Once the application's website is loaded, we call the `autoDownloadApk()` function.

4.4 Parsing the AOSP's Permissions

Once we had a set $\mathcal{A}$ of 1126 applications downloaded from the market, we moved on to the next step as it was a fair number of applications to work with. Key to our evaluation of which applications' permissions are due to third-party libraries is to learn the function calls present in the application that require a certain permission. For example, we would like to know all the methods included in the Android APIs that require the `BLUETOOTH` permission, the `READ_MESSAGES` permission or the `ACCESS_FINE_LOCATION` etc. To this end, we used the PScout tool, developed by *Kathy et al.* [15]. It analyses the AOSP and maps all the API methods with their required permissions. We analysed the latest available version of Android, Android 4.4.3, using the PScout tool. The fact that PScout is not limited to documented APIs but it also considers the undocumented methods, such as the ones from hidden APIs, is particularly beneficial as it is not rare for application developers to use these methods. Although we explained in Chapter 3 that Android is an open source OS, there is a handful number of APIs which are not exposed to the SDK. For instance, there are two different classes for sending SMS messages (SmsManager and SmsMessage) but there are no public classes for receiving them. In order to develop an application capable of receiving SMSs, developers need to use a hidden API.

In the end of the analysis, PScout came up with 66 files (one for each permission) containing a total of 28 813 methods from the AOSP that require the mentioned permissions. Please notice that Android currently declares 146 permissions. However, on the one hand, some of these permissions are declared as deprecated and hence are not in use any more. On the other hand, some of these permissions are internal and not used by the applications. At this point, we needed to define a set of identifiers that uniquely identify the methods. This procedure was necessary to, later, uniquely identify such methods in the downloaded applications. Five different identifiers are necessary to have a unique identify a method: (1) Method name, (2) Class package to which the method belongs, (3) Return type package of the method, (4) Number of

arguments and (5) Arguments type package. By the term package, we are referring to the complete path to the class.

To extract these five identifiers from the PScout's output files we wrote a parse program that analyses each method from each file and creates a *method array* which contains all the unique identifiers. For instance, given a certain permission, the parser program analyses all its methods and creates an array for each one. This array contains the five identifiers in each one of its fields. Below, we provide the generated arrays for the first two methods that require the `INTERNET` permission:

```
hasUploadingPermission 0 com/android/inputmethod/research/Uploader boolean
isPossibleToUpload 0 com/android/inputmethod/research/Uploader boolean
onHandleIntent 1 com/android/inputmethod/research/UploaderService void andr-
oid/content/Intent
```

Once we parsed all the methods we were were able to start working on the downloaded files. Please consult Appendix A.3 to see the source code of the parser program.

4.5 Reverse-engineering the Applications

There are several tools available to reverse engineer Android applications. Some years ago, the only option to reverse engineer the `.apk` files was as follows: firstly, the `.apk` file extension needed to be changed to `.zip`. Secondly, the content was able to be extracted from the compressed file and, hence,the `classes.dex` file was able to be accessed. Using a tool such as `dex2jar`, it was possible to convert the `classes.dex` file into a compressed jar. Finally, by decompressing the file, the application content was accessible. A problem with that procedure was, however, that the content from the Android Manifest file was not readable as it was presented in a binary format (BXML). This format reduces the verbosity of XML documents but hinders the use of ordinary text editors to view and edit the document. Therefore, developers were forced to use other tools such as `devtcg` (which basically convert BXML files into XML files) to access its content.

From March 2010 onwards, developers can make use of the `Apktool`[3] tool to reverse engineer Android applications. This tool, reverse engineers the applications and presents their content in `.smali` files. Further, the content from the `AndroidManifest.xml` file is treated to convert it from a binary format (BXML) into a readable format (XML). The reverse-engineering process, followed by this tool, is slightly different to the one depicted before. It is based on what is known as the *baksmali* process ("baksmali" means disassembler in Icelandic). This process, introduced in June 2009, basically acts as a disassembler for `.dex` files and presents the outcome in so called `.smali` files. This format is basically the Dalvik byte-code representation of the application's `.dex` file. Please recall from Chapter 3 that

[3]Apktool official website: `https://code.google.com/p/android-apktool/`

when an Android application is compiled, the compiler converts the Java code into byte-code (the same way that `javac` converts Java to Java byte-code for a standard Java Virtual Machine (JVM) application) using the 256 dalvik *opcodes*[4]. An opcode (operation code) is the portion of the machine language instruction that specifies the operation to be performed.

The only problem that can be found with the baksmali process is that some developers may use obfuscation techniques to avoid the reverse engineering procedure. For the sake of simplicity, we distinguish between two levels of obfuscation. The first level obfuscates the entire source code, excluding the Android API calls, by making it completely unreadable. For instance, an obfuscation tool replaces all the characters by lower-case "l" and upper-case "i" as was observed in the `com.mt.airad` package. The second level obfuscates the methods', directories' and classes' names defined by the developer or by third-party libraries by replacing them with single characters. For instance, `a`, `a/a`, `a/b`, `a/b/a`, `a/c` directories and `a`, `b`, `c`, `d` files can be found in some reverse engineered applications that have used *ProGuard*. ProGuard is a free Java files optimizer and obfuscator. It detects and removes unused classes, fields, methods and attributes and renames the remaining classes, fields, and methods using short meaningless names. Even with obfuscated files, it is still possible to identify whether a file belongs to the core of the application or not. It can be checked whether the method calls are to the core application, meaning that they are likely to be from the developer as third-party libraries are not aware of the methods declared by the developer.

A bash script program to reverse-engineer the downloaded applications and extract the necessary permissions from their Android Manifest was written. We present the source code of the script in the Appendix A.4.

4.6 Parsing and Analyzing the Applications

To check whether an application requires a certain permission or not, we developed a bash script to analyse the source code from each application and check for the methods found with the PScout tool introduced in Section 4.3. Please recall that PScout classifies the methods depending on the permission they require to be called. Therefore, if we are able to identify one of these methods in an application, it will indicate that the mentioned application needs to declare the permission required by that method.

As it has been explained in the previous section, once an application has been reverse-engineered, its output is presented in the `.smali` format. To develop a parser that analyses the methods used in an application we needed first to understand the structure of the `.smali` files. Every `.smali` file can contain the following sections: class information, static fields and methods. Figure 4.4 shows an example of how it is organized.

[4]Website that lists all the dalvik opcodes and their meaning:
`http://pallergabor.uw.hu/androidblog/dalvik\_opcodes.html`

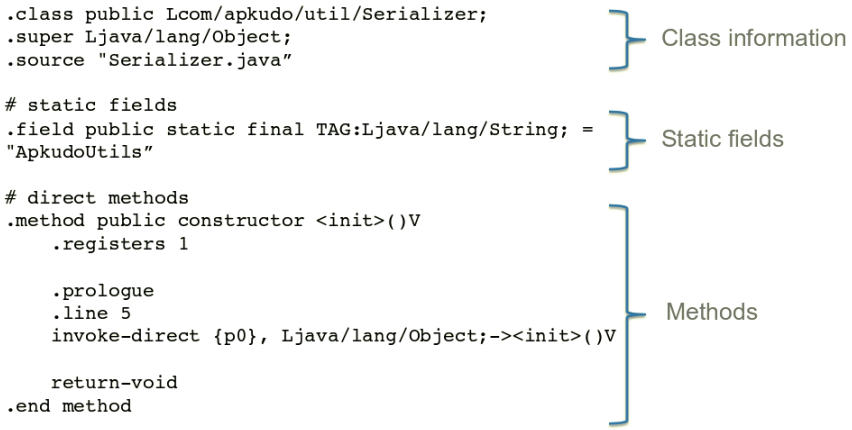


Figure 4.4: General structure of a smali file.

The only section that actually matters to our task is the methods section. Android developers can create their own methods or can call the ones from the Android APIs or other libraries' APIs. When a developer defines a method, this method will not be part of the Android API and therefore it will not be mapped to any permission. Hence, that method is not interesting. When a developer calls a method it will be reflected in the `.smali` file with a *method invocation* tag which begins with one of the following opcodes:

- **invoke-virtual**: is used to invoke a normal virtual method (a method that is not static or final, and is not a constructor).
- **invoke-super**: is used to invoke the closest superclass' virtual method.
- **invoke-direct**: is used to invoke a non-static direct method (a method that is by its nature non-overridable, namely either a private instance method or a constructor).
- **invoke-static**: is used to invoke a static method.
- **invoke-interface**: is used to invoke an interface method, that is, on an object whose concrete class is not known, using a `method_id` that refers to an interface.

Identifying method invocations allows us to identify method calls. Once the invocation of a method is identified, we can analyse it to extract all the necessary data that uniquely identifies it. Similar to the process in Section 4.3, we developed a bash script that extracts the five identifiers that uniquely identify a method. Subsequently, the parse program constructs method arrays containing these five identifiers in order to later compare them with the method arrays from the permission files. For instance, in Section 4.3 we constructed the method array `onHandleIntent 1 UploaderService void Intent`. Further, we know that this method requires a

`.smali` nomenclature	Java nomenclature
V	void
I	int
Z	boolean
B	byte
S	short
C	char
J	long
F	float
D	double

Table 4.1: Return and argument type nomenclature in `.smali` files and their meaning.

the permission `READ_PHONE_STATE` as shown by PScout. At this point, we analyze an application and create a list that contains all the methods, uniquely identified, used by the application. This will allow us to check whether a certain method array in the application is present or not in one of the permission files. For instance, if we identify the method `onHandleIntent 1 UploaderService void Intent` within the application, it will indicate that the application should declare the `READ_PHONE_STATE` permission in its manifest file.

Please note how the argument types and return types, can follow a different nomenclature in the `.smali` files as shown in Table 4.1. Our parsing program replaces the `.smali` nomenclature by the one used in Java.

Since we want to investigate the number of privileges due to the different third-party libraries, we noted that code files on an application can belong to five different categories: (1) Application itself, (2) Development tools, (3) Advertisement Libraries, (4) Social SDKs and (5) Obfuscated. The Application category contains all the files that belong to the application itself (flies within the package specified by the developer). The other categories are basically due to the presence of their respective third-party libraries. Finally, we include a category that encompasses all the obfuscated files, e.g. the derivative files due to the usage of obfuscation programs such as ProGuard. Please notice that there is no way, other than manually inspecting the obfuscated files, for classifying the obfuscated files in a certain third-party library or another. The only thing we can do is to distinguish whether it belongs to the Application itself or not.

At this point, we needed to differentiate the files belonging to each category. When developing an application, the developer defines a package for the application. Therefore, all the application-related files will be included within that package. Following this rule, it is easy to discern whether a file belongs to the application itself or not regarding its current directory. To distinguish whether an application belongs to a certain permission or to another one, we consulted three lists of the most popular Advertising Libraries, Development tools and Social SDKs. This initial lists were obtained from the Appbrain project. Once we knew how to correctly classify each directory, the script was executed. Its output consists in five different output directories: (1) MethodsApp, (2) MethodsDevelopmentTools, (3) MethodsAdLibraries, (4) MethodsSocialSDKs and (5) MethodsObfuscated. These directories

contain files with the method arrays formed with all the methods found in each `.smali` file belonging to each category. For further information about this program please check the Appendix A.6.

Please note that sticking to a fixed list of third-party libraries is a very limited approach as there can be other third-party libraries rather than the ones included in the Appbrain lists. Consequently, we wrote a bash script that finds new third-party libraries. The outcome of this script is a file which contains all the newly found third-party libraries which are not present in the Appbrain lists. Later, we can manually analyse and classify them into one category. Please note that the amount of third-party libraries is not fixed and, therefore, there is no way to find them out other than manually inspecting and categorizing them. The source code of the bash script can be found in A.5.

Once all the third-party libraries that were included within the set of applications $\mathcal{A}$ were identified, we determined that the rest of the directories had been obfuscated and we added them in the Obfuscated category. After the manual inspection an classification of the newly found third-party libraries, we had a list of 80 Advertising libraries, 212 Development tools libraries and 15 Social SDKs.

Before going on, the set $\mathcal{A}$ of 1126 applications from the Google Play market was analysed. In total, 76 new third-party libraries were discovered including Advertising Libraries and Development Tools libraries. Table A.1 in the Appendix A.5 shows the name of the found third-party libraries, how many applications used them, what are they used for and their category.

We noticed that 74 out of 76 of these libraries are development libraries and only two are advertising libraries. Taking advantage of the code wrote to find new third-party libraries, we wrote a bash script to get statistics about the usage of third-party libraries in the considered applications. By slightly modifying the script from Appendix A.5, we got a list with all the third-party libraries and the number of times they were found in the set of applications $\mathcal{A}$. Figure 4.5 shows the top ten most popular libraries from each category.

Image (a) shows that Google/ads is clearly the leading Advertising library. The reader needs to know that the Google/ads library is branded as AdMob. It is found in 761 applications (67.51%). Further, the difference with respect to the second most popular, Charboost, is of 543 applications. In this case, the second, third, forth and fifth place are clearly allocated: Charboost, Millennial Media, Inmobi and Tapjoy.

With respect to the Development libraries, graph (b) shows how the Android supporting tools is the top library. It is found in 1053 applications (93.52%). Please note that Android supporting tools encompass the Android Annotation, Android Support, Android Vending and Android GSM libraries. The difference with respect to the second most popular, Flurry, is rather big (present in 788 applications less). As we can see from graph (b), the difference between all the Development tools libraries is rather small (13 applications between the third most popular and the fifth most popular) and we can therefore conclude that the market is clearly competing for the third place.

Finally, Facebook is clearly the leading OSN SDK. In particular, it is present in 299 applications out of 1126 (26.55%), being more than six times as popular than

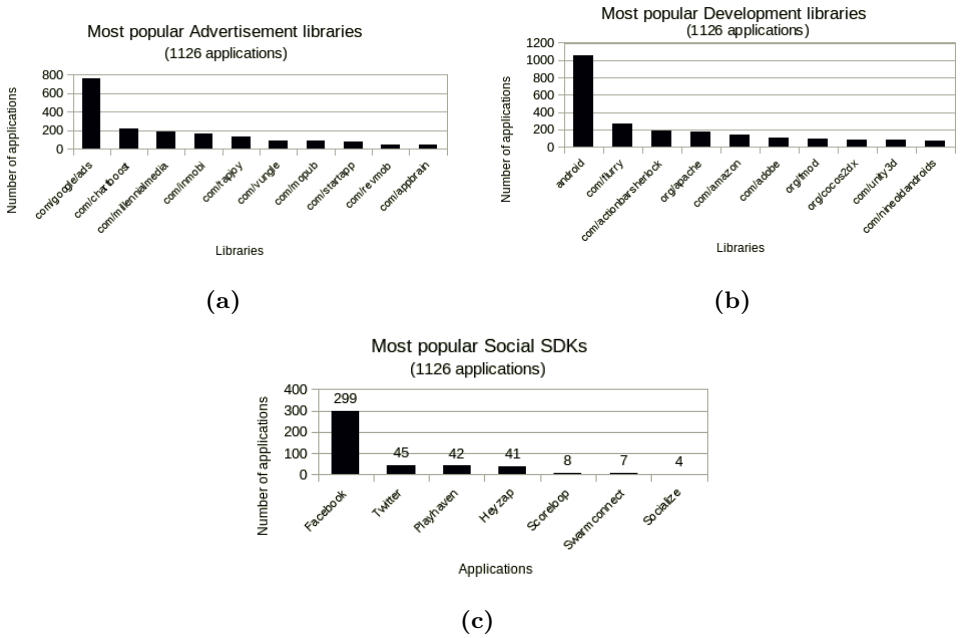
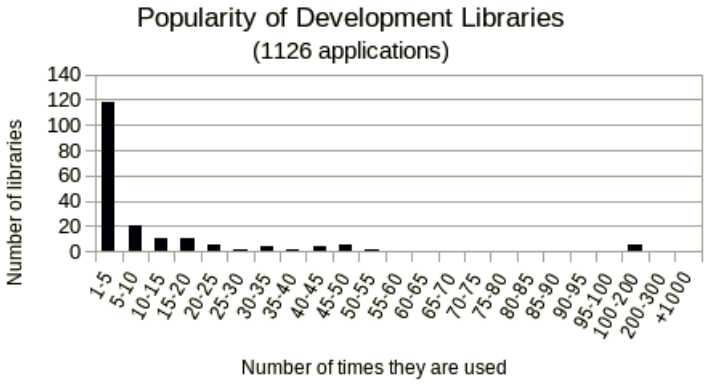


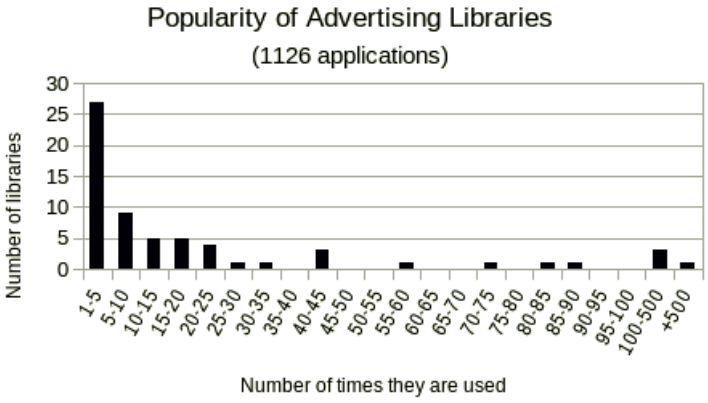
Figure 4.5: Most popular third-party libraries found in the 1126 analyzed applications. Graph (a) shows the most popular Advertising, graph (b) the most popular Development tools libraries and graph (c) the most popular Social SDKs.

the second in the list, Twitter, which is only present in the 3.99% of the applications. Interestingly, in the set $\mathbb{A}$ we only found seven different Social SDKs libraries, while we found 197 Development libraries and 63 Advertising libraries. Please notice that the updated versions of the Appbrain's lists contain 14 Social SDKs, 211 Development libraries and 80 Advertising libraries. Therefore, we only were able to identify the 50% of the Social SDKs, a 78.75% of the Advertising libraries and a 93.36% of the Development tools libraries in the set $\mathcal{A}$.

Figure 4.6 presents two graphs that illustrate the presence of Development and Advertising libraries in the applications. In the set of 1126 applications we identified the presence of 197 Development libraries and 63 Advertising libraries. The vast majority of them seem to be not common and are only used in between 1 to 5 applications. Concretely, 118 (almost 60%) Development tools libraries are only present in between one and five applications. That means, that the vast majority of these libraries are not common. Only six libraries are used in more than the 10% of the applications. With respect to the Advertising libraries, we find a similar situation. More than the 40% of the Advertising libraries are used in between 1 to 5 applications. Finally, only 4 libraries are used in more than the 10% of the applications.



(a)



(b)

Figure 4.6: Presence rate of Development tools (a) and Advertising libraries (b).

4.7 Generating Android Manifest Type Files

The last step, consisted in analysing the permissions required by the applications and compare them with the declared permissions in their Android Manifests. To define the permissions an application requires, we checked whether a certain method array created in Section 4.4 was present in the list containing all the method arrays, created in Section 4.6, from an application. Basically, we read all the methods from the permission files and sought for these methods in the application files. If a certain method m from the permission p is present in the application, that permission p should be declared in the Android Manifest. We developed a bash script that seeks the permission methods in the application files. In the end, it generates 5 manifest type files, one for each category, containing all the necessary permissions that should be included to use all the methods provided by the third-party libraries or called in the application itself.

Notice that some reverse-engineered applications occupy up to 90 MB and contain more than 8 000 files. This fact implies that generating the corresponding manifest-type file can require a lot of time as we have to seek for the presence of more than 28 000 methods in more than 8 000 files. For large applications (>50 MB), the entire procedure followed by 3Ov can last up to eight hours (in a 97.8 MB application). Please consult Appendix A.7 to see the source code of the bash script.

4.8 Limitations of 3Ov

Before discussing the outcome of our research, we state the limitations of our approach. 3Ov is a static analysis tool. This means that it only analyses the raw source code of the applications and not their behaviour at run-time. Android permissions, however, are only necessary at run-time. For instance we can build an application that uses the GPS without declaring the `ACCESS_FINE_LOCATION` permission. We will also be able to install it. Now, suppose the application retrieves the coordinates from the GPS only when a certain button *"get GPS data"* is pressed. The application will only crash if that button is pressed.

This fact rises a problem. Suppose an application with an embedded Advertising library and a Social SDK such as AdMob and Facebook. If a certain permission, present in the manifest, is not required by the Application itself but it is required by both the Advertising library and the Social SDK, it is not possible for 3Ov to distinguish whether the mentioned permission is present in the manifest due to the Advertising library, the Social SDK or both. In the next section, we will define five different heuristics that will study the presence of over-privileges in each category independently from the others. However, for what we have shown in the previous example, if a certain permission in granted in more than one third-party library, there is no way in which 3Ov can determine whether the mentioned permission has been granted to the application due to a certain third-party library, due to more than one or due to all of them. This issue leads us towards the necessity of defining a lower and an upper bound in the number of over-privileges. Consequently, for the lower bound, if a certain permission is only granted due to one category of third-party library it will be counted as an over-privilege while if it is present in more than one category it will be dismissed. For the upper bound, if a certain permission, which is not due to the Application itself, is present in a third-party library, it will be considered as an over-privilege independently to whether this permission is also present in other third-party libraries or not.

Finally we have to state the limitations due to the obfuscation tools. There is no way in which 3Ov can distinguish whether a certain obfuscated file belongs to a certain third-party library or to another one. Therefore, the only way in which we could classify the obfuscated libraries would be to manually analyse them. This issue will be addressed in Chapter 6.

Permission Name	Application	Development	Advertising	Social SDKs
ACCESS_COARSE_LOCATION	0.0562	0.0234	0.0674	0.0000
ACCESS_FINE_LOCATION	0.0225	0.0234	0.0674	0.0000
ACCESS_NETWORK_STATE	0.2247	0.1122	0.6517	0.3538
ACCESS_WIFI_STATE	0.0562	0.0210	0.0787	0.0000
BLUETOOTH	0.0225	0.0047	0.0000	0.0000
BLUETOOTH_ADMIN	0.0112	0.0023	0.0000	0.0000
DISABLE_KEYGUARD	0.0225	0.0000	0.0000	0.0000
INTERNET	0.2697	0.1192	1.0000	1.0000
KILL_BACKGROUND_PROCESSES	0.0112	0.0000	0.0000	0.0000
READ_PHONE_STATE	0.0899	0.0327	0.1236	0.0000
READ_PROFILE	0.2921	0.1449	0.6517	0.0000
READ_SYNC_SETTINGS	0.1910	0.0350	0.0000	0.0000
RESTART_PACKAGES	0.0112	0.0000	0.0000	0.0000
SET_WALLPAPER	0.0449	0.0000	0.0000	0.0000
WAKE_LOCK	0.3258	0.1122	0.6404	0.0000
WRITE_EXTERNAL_STORAGE	0.0000	0.0000	0.0000	0.0154

Table 4.2: Shows the probabilities of requiring the permissions from the first column by the Application itself, the Development tools, the Advertising libraries and the Social SDKs.

4.9 Proof of Work

At this point, we analysed a subset $\mathcal{B}$ of 89 applications seeking for the presence of over-privileges. Firstly, we studied the increase of permissions due to the usage of third-party libraries. Secondly, we analysed the presence of over-privileges. Within the subset $\mathcal{B}$ we identified 27 different Advertising libraries, 63 Development tools libraries and five Social SDKs. Please notice that 89 applications hardly represent a useful percentage out of the total number of crawled applications. Due to time constrains, we could not analyse more applications and therefore we introduce this section as a *Proof of Work* and not as a final result.

Analysing the likelihood of finding the different Android permissions granted in the manifest gives us a first idea about how different third-party libraries are more likely to declare certain permissions than others. If Android applications were shown to be over-privileged, the different permissions required by the different categories of third-party libraries, would lead towards the presence of different over-privileges. We started considering each category independently, e.g. only the Application itself, only Development tools libraries and so on. Then, we studied the pair combination of the Application with all the third-party libraries, e.g. Application with Development tools, Application with Advertising libraries and so on. With this analysis, we determined how certain permissions are more likely to be granted to the application when a certain category of third-party library is used to extend the applications' functionality. Please note that all the obfuscated directories an files cannot be classified in a certain category. Hence, the obfuscated libraries will be analysed later. Table 4.2 shows all the probabilities identified when analysing each category (Obfuscated category excluded) independently from each other. Please notice that the table only shows all those permissions whose probability was greater than zero. To see the source code of the bash script that calculates the probabilities please consult the Appendix A.8.

Notice from the previous table that only five permissions are likely to be granted to more than a 10% of the applications when considering only the Application itself: `ACCESS_NETWORK_STATE`, `INTERNET`, `READ_PROFILE`, `READ_SYNC_SETTINGS` and

WAKE_LOCK being that last one the likeliest (32.58% of the applications declare this permission in their manifest). The WAKE_LOCK permission grants the application with privileges to keep the CPU from sleeping or the screen from dimming. The likeliest permissions regarding the Development and Advertising libraries and the Social SDKs are READ_PROFILE (14.49%), INTERNET (100%) and INTERNET (100%) respectively. As it was expected, Advertising libraries and Social SDKs require the INTERNET permission in order to work. More interesting, the likeliest permission required by the Development tools libraries is the READ_PROFILE permission. This permission allows an application to read the user's personal profile data. User profile encompasses several device settings such as the Bluetooth, GPS, WiFi, Mobile Data and Volume states. A part from that, it can also be observed how a lower number of permissions are granted to the different third-party libraries. In this way, 16 different permissions are declared by the Application itself, eleven by the Development tools libraries, eight by the Advertising libraries and only three different permissions by the Social SDKs.

Following, we analysed the probabilities when grouping pairs of the Application itself with the rest of third-party libraries. Table 4.3 shows the mentioned probabilities. As before, we only present non-null probabilities. Before explaining the results, we want to state that the ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, READ_PHONE_STATE, READ_PROFILE and READ_SYNC_SETTINGS permissions provide different mechanisms to access users' sensitive data. When grouping the Application itself with the Development libraries, it can be observed how all the previous permissions decrease. For instance, the ACCESS_NETWORK_STATE permission decreases from a likelihood of 22.47% to a 9.86% or the READ_PROFILE permission from a 29.21% to a 12.77%. This means that the usual permissions required by the Development tools libraries are not meant to access users' sensitive data. However, when grouping the Application itself with the Advertising libraries, the likelihood of requiring all the previous permissions, excluding READ_PROFILE and READ_SYNC_SETTINGS, increases. That means that Advertising libraries require more often permissions to access users' sensitive data. Finally, when combining the Application itself with the Social SDKs, we see how all the permissions, except the ACCESS_NETWORK_STATE permission, decrease. This permission allows applications to determine whether the user has an internet connection or not, determine the type of connection (WiFi, 2G/3G/4G) and subscribe to changes in the connectivity. Therefore, we can conclude that Social SDKs are not meant to gather personal data from the user excluding the necessity of being aware of the Internet connection status as it is essential for their functionality.

Taking advantage of the tools included in 3Ov, we analysed the five most popular third party libraries from each category in order to provide further information about the permissions required by each of them. Please notice that, ideally, we would have analysed all the found third party libraries in the set $\mathcal{B}$. However, due to time constrains, we considered only the top five libraries which are present in a fair Android applications as shown in Table 4.4. Please notice that the rest of the third-party libraries from each category are used in less than the 1% of the current applications. Figure 4.7 illustrates the results. Notice that we do not provide any

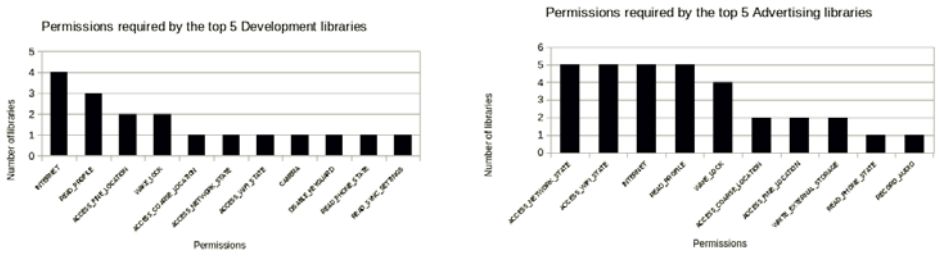
Permission Name	Application Development	Application Advertising	Application Social SDKs
ACCESS_COARSE_LOCATION	0.0251	0.0562	0.0327
ACCESS_FINE_LOCATION	0.0213	0.0449	0.0131
ACCESS_NETWORK_STATE	0.0986	0.3539	0.2843
ACCESS_WIFI_STATE	0.0213	0.0674	0.0327
BLUETOOTH	0.0077	0.0112	0.0131
BLUETOOTH_ADMIN	0.0039	0.0056	0.0065
DISABLE_KEYGUARD	0.0039	0.0112	0.0131
INTERNET	0.1364	.8764	0.4365
KILL_BACKGROUND_PROCESSES	0.0019	0.0056	0.0065
READ_PHONE_STATE	0.0329	0.0843	0.0523
READ_PROFILE	0.1277	0.3708	0.1830
READ_SYNC_SETTINGS	0.0368	0.0955	0.1111
RESTART_PACKAGES	0.0019	0.0056	0.0065
SET_WALLPAPER	0.0077	0.0225	0.0261
WAKE_LOCK	0.1122	0.3596	0.1895
WRITE_EXTERNAL_STORAGE	0.0000	0.0000	0.014

Table 4.3: Shows the probabilities of requiring the permissions from the first column by the Application along with the third-party libraries.

Name	Presence	Name	Presence	Name	Presence
Facebook	12.36%	Google/ads	37.4%	Android Support	43.56%
Twitter4j	1.63%	Millennial Media	3.63%	Flurry	7.4%
Playheaven	0.75%	InMobi	3.11%	Actionbar Sherlock	7.33%
Heyzap	0.58%	Chartboost	2.31%	Apache	5.75%
Scoreloop	0.4%	Tapjoy	1.59%	Amazon	1.63%

Table 4.4: shows the name and the presence percentage (out of the total number of applications) that use the mentioned third party libraries.

graph concerning the Social SDKs because only the `INTERNET` permission (required by the five of them) and the `ACCESS_NETWORK_STATE` (required by two) were found. Notice that the number of different permissions identified in both categories is very similar (11 for the Development libraries and 10 for the Advertising). Further, there are seven permissions which are required by both categories.



(a) **(b)**

Figure 4.7: Graph (a) shows the permissions required to use the top five Development tools libraries. Graph (b) shows the permissions required to use the top five Advertising libraries.

The fact that a certain permission is more likely to be granted to the application when a certain third-party library is used, does not necessary lead towards the presence of over-privileges in Android applications. A formal definition of the term over-privilege is difficult to provide as it is basically a *point of view*. For instance,

what the user could consider as an over-privilege, e.g. an advertising banner, for the developer could be the most important feature of her application. Therefore, we are going to consider five different heuristics which will be thoroughly studied. Please notice that for each heuristic, we will determine a lower and an upper bound for the number of over-privileges as it was explained in the previous section. The five heuristics are:

1. Heuristic 1: Only permissions granted due to the presence of third-party Advertising libraries can be considered as over-privileges. We could argue that a permission granted to the application only because of the usage of an Advertising library should be considered as an over-privilege as far as an embedded banner is not necessary for the proper functionality of the application.

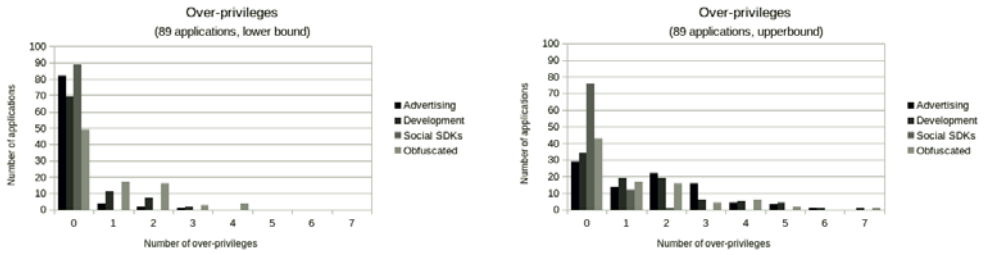
2. Heuristic 2: Only permissions granted due to the presence of third-party Development tools libraries can be considered as over-privileges. Only a manual thorough analysis would determiner whether these libraries are gathering sensitive data from the user and sending it to third-party servers or not.

3. Heuristic 3: Only permissions granted due to the presence of third-party Social SDKs libraries can be considered as over-privileges. Sharing the application's data with the OSNs is not necessary for the correct functionality of an application. For instance, sharing the score of a game-type application with users' friends on Facebook is not necessary to play the game.

4. Heuristic 4: Only permissions granted due to the presence of obfuscated libraries can be considered as over-privileges. In this case, we are considering that the Obfuscated category only encompass files belonging to the third party libraries and not to the Application itself. As explained in the previous section, this must be cautiously considered as files belonging to the Application itself could also have been classified within the Obfuscated category. However, all the files that belong to the Application itself should be, theoretically, included within the application's package. Therefore, we assume this heuristic as feasible.

5. Heuristic 5: Only permissions granted due to the presence of Advertising libraries, assuming that the Development tools libraries and the Social SDKs belong to the core of the application, can be considered as over-privileges. No obfuscated libraries are considered in this heuristic. We could argue that, in the end, advertising banners are not necessary for the proper functionality of the application. Basically, an application can offer the same possibilities to the users independently of using Advertising libraries or not. Hence, in this last situation, we consider the Development tools libraries and the Social SDKs as belonging to the core of the application.

We studied the number of over-privileged applications considering the first four heuristics. In Figure 4.8, both graphs show the number of applications that present a certain number of over-privileges according to the first four heuristics. Graph (a) considers the definition of the lower bound while graph (b) considers the upper.



(a) **(b)**

Figure 4.8: Over-privileges found according to the first four heuristics. Graph (a) considers the lower bound while graph (b) the upper.

On the one hand, graph (a) shows the number of over-privileged applications when considering the first four heuristics and the lower bound. Notice how the vast majority of applications are not over-privileged. When we analysed the applications from the subset $\mathcal{B}$, we only identified 22 different permissions (33.33% of the total of permissions identified by PScout) declared in the applications' manifest files. All together means that this set of 22 permissions are very common and, hence, the likelihood of finding these permissions declared in other libraries' manifest-type files is very high. Finally, notice that Development libraries leads towards the presence of a greater number of over-privileges than the Advertising libraries or the Social SKDs. This is due to their varied nature. However, the obfuscated libraries are the ones responsible for the greatest number of over-privileges. This situation is logical as we are assuming the Obfuscated category as a set of all the other categories of third-party libraries. Therefore, permissions such as `CAMERA`, `CHANGE_WIFI_STATE` and `RECORD_AUDIO` were only present in the obfuscated libraries. On the other hand, graph (b) considers the upper bound. In this graph it can be observed how Development tools and Advertising libraries can lead towards the presence of seven and six over-privileges (respectively), while the Social SDKs only up to two. Notice that the Development tools and the Advertising libraries normally lead towards the presence of one, two or three over-privileges (49.44% and 58.43% of the applications respectively). Reasonably, the number of over-privileges found due to the obfuscated libraries when considering the upper bound does not importantly increase. Once again, this is due to the presence of some unique permissions declared in the obfuscated libraries. Concretely five out of the 18 (27.78%) permissions found in the obfuscated libraries were unique. Therefore this five permissions were considered as over-privileges by both the upper and the lower bound.

Figure 4.9 presents the results of the fifth heuristic. Only the permissions granted to the application due to the usage of Advertising libraries are considered as over-privileges. Further, no obfuscated libraries are considered in this case as there is no way to distinguish between the different categories of third-party libraries. Please notice that there is no necessity to define an upper and a lower bound. We consider a permission as an over-privilege when it is declared in the manifest but it is not

needed by the set of the Application itself, Development tools libraries and Social SDKs. Notice how the most common number of over-privileges in this situation is two. Concretely 17 applications (almost 20%) presented two over-privileges.

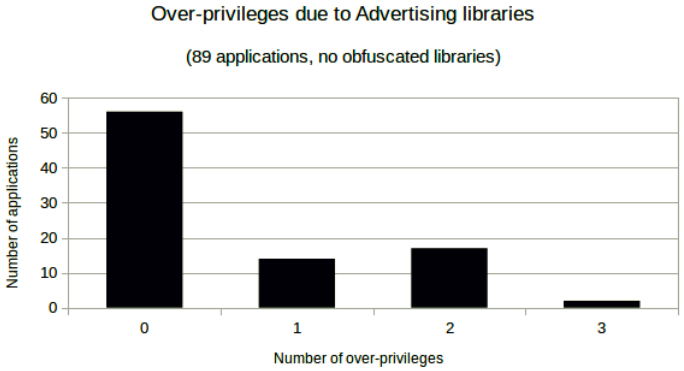
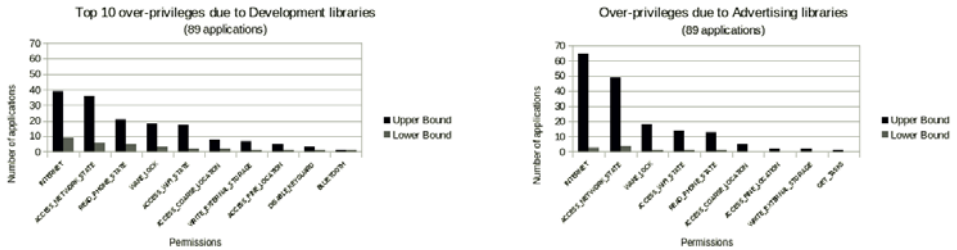


Figure 4.9: Over-privileges found according to the fifth heuristic.

Afterwards, we analysed the most popular over-privileges when considering the first three heuristics. In Figure 4.10, it can be seen the most popular over-privileges found according to the different categories of third-party libraries. Please notice that no over-privileges were found do to the Social SDKs when considering the lower bound and only 13 applications were over-privileged when considering the upper bound. Further, only the `INTERNET` (present in the 13 applications) and the `ACCESS_NETWORK_STATE` (present in one application) permissions were classified as over-privileges in this last case. Notice that the `INTERNET` permission is the most popular over-privilege when considering the upper bound. When considering the lower bound, the `ACCESS_NETWORK_STATE` permission becomes the most popular one. Moreover, notice from the Advertising libraries graph that only nine different permissions were found when analyzing the 27 found Advertising libraries.



(a) **(b)**

Figure 4.10: Most popular over-privileges due to the usage of the different categories of third party libraries. Graph (a) shows the most popular over-privileges due to Development libraries. Graph (b) shows the most popular over-privileges due to Advertising libraries.

Further, we studied the most popular over-privileges due to the fifth heuristic. Please recall from the definition of the heuristic that this is a very special case. Thus, we analyse it separately. Considering the fact that the Obfuscated category only contains files belonging to the different categories of third-party libraries, we analysed the most popular over-privileges due to the Obfuscated libraries. Notice from the Figure 4.11 that there is not a big difference with respect to the most popular over-privileges caused by the three categories of third-party libraries. This is a good reason to think that the Obfuscated category could only contain obfuscated third-party libraries.

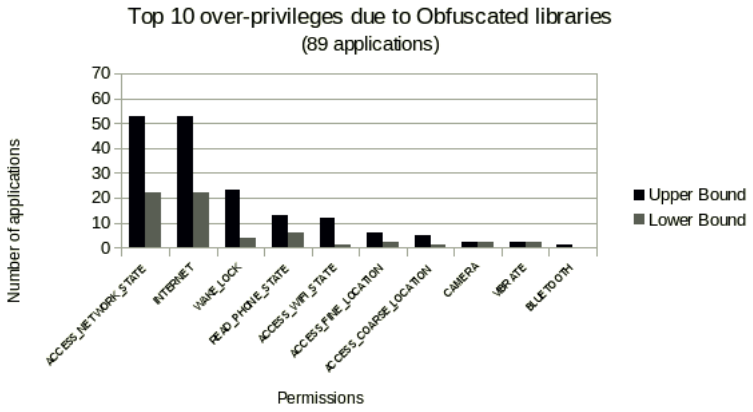
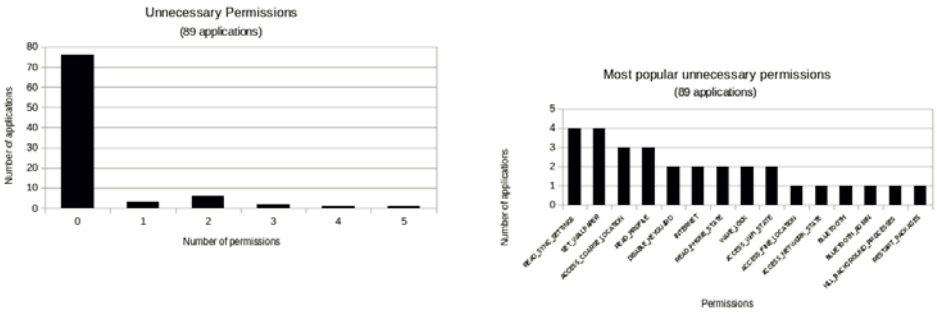


Figure 4.11: Over-privileges found according to the fourth heuristic.

We also analysed the presence of missing permissions. Out of the set of 89 applications we found two of them that were missing a permission. A permission was assumed as missing when it was required by the core of the application but it was not declared in the manifest file. Concretely, application `com.miniclip.plagueinc` was missing the `READ_PROFILE` permission and application `mobi.artibus.slotmaniacs` the `WAKE_LOCK` permission. We manually inspected both applications to check whether we had a false positive or the applications were really missing the mentioned permissions. We could not determine any cause for which these two permissions were missing. Therefore, we concluded that either both applications had dead code or the PScout tool was mistaking certain mappings.

Finally, we studied the unnecessary permissions granted to the applications. In Figure 4.12, it can be observed how 13 applications (14.61%) declared unnecessary permissions in their manifest file. A permission was assumed as unnecessary when it was declared in the manifest but it was not present in any of the manifest-type generated files. The `com.blocks.game.brick.puzzle.blockpuzzle` declared five unnecessary permissions. Further, the most popular unnecessary permissions are `READ_SYNC_SETTINGS` and `SET_WALLPAPER`. The former allows applications to access the online accounts managed by the mobile phone such as email accounts and OSNs accounts. The latter allows applications to set a wallpaper.



(a) **(b)**

Figure 4.12: Image (a) shows the number of applications that have declared one or more unnecessary permissions. Image (b) shows which are the most popular permissions unnecessarily granted to the applications.

4.10 Discussion

Firstly, by analysing the set $\mathcal{A}$, we found out 74 new Development tools libraries and 2 new Advertising libraries that were not included in the Appbrain lists. In this way we provide a final list with 211 Development tools libraries, 80 Advertising libraries and 14 Social SDKs. Furthermore, we showed how the vast majority of third party libraries are rarely used in more than five applications out of 1126 (0.45% of the applications).

So far we have shown that the term *over-privilege* is not easy to define. Therefore, we have introduced five different heuristics that can pave the way for future investigation. Whatever the reader decides that an over-privilege is, she can play with the provided data to figure out how many over-privileges she is likely to find according to her definition. As a proof of work, we have shown how Android applications are over-privileged independently of the heuristic. Only when considering the Social SDKs and the definition of the lower bound no over-privileges where found in the set $\mathcal{B}$. Further, we have shown how only a small set of Android permissions (around 30%) are commonly used. This implies that the most popular over-privileges found due to the different heuristics are similar. Only the Development tools libraries lead towards the presence of more varied over-privileges. Moreover, we have shown how Android applications can miss permissions in their manifest files, e.g. the presence of dead code generates false positives. Finally, we have illustrated how developers declare unnecessary permissions in their applications manifest.

A part from that, we have analysed the probabilities of declaring certain permissions in the manifest due to the usage of different third party libraries. Coherently, we showed how Advertising libraries and Social SDKs always require the `INTERNET` permission to be declared in order to use them. Development tools libraries are the ones that require greater number of different permissions while Social SDKs are the ones that require less (eleven and three respectively) Furthermore, we showed that Development tools libraries are more likely to require a greatest variety of permissions.

When combining the Application itself with the different third party libraries, it is specially interesting to pinpoint how the probability of declaring permissions which potentially can access users' sensitive data increases.

Chapter 5

LPDroid: Application Transparent Protection of Location Data

In Chapter 4, we presented how an important number of the applications from the set $\mathcal{B}$ are over-privileged. Concretely, we found over-privileges when considering all the five heuristics. Apart from that, previous work has shown that sensitive data can be transferred to third-party servers [10, 11] for user profiling and personalized advertisement targeting. This makes sensitive data protection mechanisms necessary. In this section, we provide an example on how to protect users' sensitive data. In particular, as explained in the Introduction, we focus on protecting users' location data.

In Section 5.1 we will elaborate on what Android applications do at the user level to protect users' location data. Since we will identify several shortcomings, we will discuss how user data can be protected in lower levels of the AOSP in Sections 5.2 and 5.3. Consequently, we will introduce LPDroid, an extension of TaintDroid that provides developers with a set of tools to pave the way for future implementation of LPPMs. As a proof of work, we implement three trivial protection mechanisms.

5.1 Protecting Users' Location at the User Level

5.1.1 Current Applications

There are several location privacy tools available in the Google Play market. The vast majority of them consist of location spoofers which allow users to fake their location coordinates. Table 5.1 illustrates the top ten location-spoofer type applications present in the Google Play market as of May 2014.

Their functionality is almost the same in all cases: firstly, users need to activate the *allow mock locations* option from the *settings menu*. Once it is done, users can proceed to fake their location. We manually tested all the applications listed in Table 5.1 and realized that all of them include the option to manually set fake coordinates

Name	Number of downloads	Minimum Android version	Drawbacks
com.lexa.fakegps	500,000 - 1,000,000	1.5	Wrong updating timing for faked coordinates
org.ajeje.fakelocation	100,000 - 500,000	1.6	Does not work with Facebook 4.0 nor with Youtube
com.fakegps.mock	100,000 - 500,000	2.3	Does not work with Facebook 4.0
com.my.fake.location	100,000 - 500,000	1.5	Manual configuration of the faked coordinates
kr.woot0pia.gps	100,000 - 500,000	1.5	Can't stop faking the coordinates unless you destroy the application
com.incorporateapps.fakegps.free	50,000 - 100,000	2.1	No missfunctionalities were identified
com.appandmobile.locationspooferfree	10,000 - 50,000	1.5	Does not work in Android 4.4.2
ait.com.locationfaker	1,000 - 5,000	2.2	No missfunctionalities were identified
com.hyunnyapp.fakemylocation	1,000 - 5,000	2.2	No missfunctionalities were identified
com.mjhdev.fakelocationfree	500 - 1,000	4.2	Does not work in Android 4.4.2

Table 5.1: Shows the top 10 location spoofer applications available in the Google Play market as of May 2014.

or to pick randomly generated coordinates. The `com.lexa.fakegps` application also allows to set an obfuscation radius (centered at the user's current location) and an obfuscation time interval. When setting an obfuscation radius and a time interval, the application generates fake coordinates within that radius at every time interval.

Besides location spoofer type applications, applications such as *Clueful Privacy Advisor* or *LBE Privacy Guard* read the Android Manifest file of the installed applications at run-time. Then, according to a certain criteria, they classify the applications in *low risk*, *medium risk* or *high risk* depending on the permissions declared in the Android Manifest file. The followed criteria to classify the analyzed applications is hard coded in the application and therefore it is not available publicly.

There is one application that includes all the mentioned features: *Place Mask*. It allows users to monitor the privacy level of their installed applications at run time and obfuscate the location coordinates. It offers the possibility to manually set the faked coordinates or to select an obfuscation radius and a generating time interval. Once these values are set, the application generates new random coordinates within the radius at every time interval. Please notice that all the protection mechanisms implemented in these applications are very easy to break.

Although it is not yet an official Android feature it is worth to mention the *App Ops* application. App Ops is a hidden menu present from Android 4.3 onwards that lets users manage the permissions used by the applications. It was found by the Android Police[1] team who created an application to show this hidden menu and let users administrate the privileges granted to their installed applications. Android Police is a web blog dedicated to the Android world. App Ops menu presents four permission groups: location, personal, messaging and device. Each group is shown in a tab which presents a list of applications sorted by the time when they last used the

[1]Android Police official website: `http://www.androidpolice.com/`

permissions of the current tab. For example, in the Location tab, all the applications that have queried access to an `ACCESS_LOCATION` type permission will be displayed in a list sorted by the time they queried the access to the mentioned permission. App Ops also shows the permissions used by each application. Users can disable permissions for each application they have installed, even for system applications. Sometimes, however, disabling permissions does not have any effect. For instance, when we tried to block Facebook's `ACCESS_LOCATION` type permissions, it still had access to location data.

Interesting enough, Google immediately modified the App Ops menu to avoid it being opened by the Apps Ops application developed by the Android Police team[2]. Consequently, in Android 4.4.2 versions it was not accessible anymore. Google claimed that the menu was experimental, and that it could break some of the applications policed by it. Currently, it is unclear what will the Android team do with App Ops.

In the following sections we present two Android applications to understand how LBAs work. Firstly, we will present an application that shows users' current location. Secondly, we will introduce an application that lists all the installed applications that require either the `ACCESS_COARSE_LOCATION` or the `ACCESS_FINE_LOCATION` permissions. Finally, we will analyze the location spoofer type applications. Developing the first application will allow us to later check whether the mechanisms that we will implement to protect users' location work or not. The second application will provide users with a tool to see how many of their installed applications are potentially dangerous concerning to their location data. Both applications will be part of LPDroid which will be introduced in Section 5.2.5.

5.1.2 MyLocation Application

In Android systems, users have access to 3 different Location Providers: *GPS*, *Network* and *Passive* providers. Firstly, the GPS provider determines users' location using satellites. It is the most accurate provider although it is the most expensive in terms of energy consumption and it is only available outdoors. It requires the `ACCESS_FINE_LOCATION` permission to be used. Secondly, the Network provider determines the location based on cell towers and WiFi access points. It requires the `ACCESS_COARSE_LOCATION` permission to be used. Finally, the Passive provider returns the locations generated by other providers. It is a combination of the GPS and the network providers and requires the `ACCESS_FINE_LOCATION` permission to be used.

Every LBA obtains an instance of the Location Manager class by calling the method `getSystemService(LOCATION_SERVICE)`. This is done in `onCreate()` as this class cannot be instantiated following the regular Java procedure for object invocation. After, LBAs create a LocationListener object from which the following callback methods can be overridden:

[2]Article that informs about the actions taken by Google to remove the App Ops menu:
`https://www.eff.org/deeplinks/2013/12/google-removes-vital-privacy-features-android-shortly-after-adding-them`

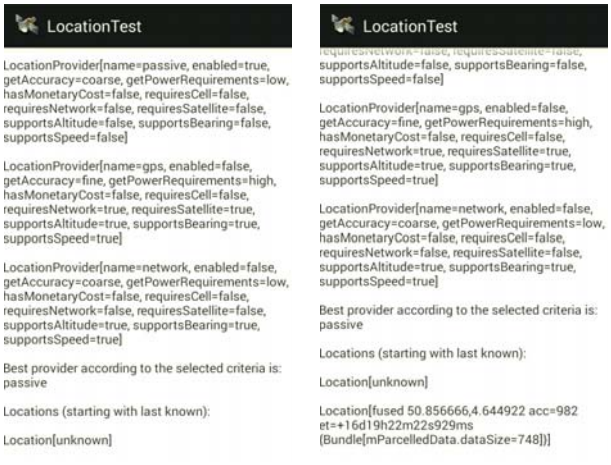
- **onLocationChanged()**: is called when new coordinates are received. During the registration time, developers can specify how often the location coordinates should be updated.
- **onProviderDisabled()**: is called when a provider is disabled, e.g. when a user is running out of battery and, therefore, decides to disable the GPS provider.
- **onProviderEnabled()**: is called when a provider is enabled, e.g. when a user that is currently using the Network provider decides to change to the GPS provider in order to receive more accurate coordinates.
- **onStatusChanged()**: is called when the status of a provider changes. There are three different status for a location provider: `OUT_OF_SERVICE` if the provider is out of service, and this is not expected to change in the near future; `TEMPORARILY_UNAVAILABLE` if the provider is temporarily unavailable but is expected to be available shortly; and `AVAILABLE` if the provider is currently available.

In `onResume()`, the method `requestLocationUpdates()` is called in order to subscribe the application to one of the providers. The desired provider can be specified through a hard coded Criteria object. For instance, we can specify the `FINE_ACCURACY` criteria in order to subscribe to the GPS provider. Further, it can be specified the time interval in which location updates should occur. For example, a developer may want to develop a battery-aware application that queries for location updates only when they are necessary and not "as soon as possible". Moreover, the distance interval in which location updates should occur can also be specified. For instance, if the application's accuracy is not an important parameter, developers can set a certain update distance rather than update the location every time new coordinates are available.

According to the procedure described above, we first developed *MyLocation*, an application that displays the location providers and their information as shown in Figure 5.1. It selects the best provider according to a hard coded criteria and finally shows the location coordinates of the user. The criteria used in our case was the `ACCURACY_FINE` criteria which only allows the usage of the GPS provider. As it will be addressed in Section 5.2.5, this application will belong to LPDroid. MyLocation will allow LPDroid users to check the functionality of the LPPMs implemented in LPDroid. Please refer to Appendix B.1.1 to see the source code of this application.

5.1.3 LocationApps Application

We developed *LocationApps*, an application that lists all those installed applications that require one of the two location related permissions: `ACCESS_FINE_LOCATION` or `ACCESS_COARSE_LOCATION`. This application can be useful to determine the number of installed applications potentially dangerous with respect to users' location privacy. Therefore, this application will also be part of LPDroid and it will allow users to have an overview on how many applications have access to their location data.



(a) **(b)**

Figure 5.1: Show MyLocation application running in a real device. Image (a) shows the information about the providers while image (b) shows the information embedded with the received coordinates.

LocationApps first retrieves a list containing all the installed applications by calling the `getInstalledPackages()` method. Later, for every application installed, it checks whether it requires a location based permission or not. The following portion of code is used to make this verification:

```
1 if (PackageManager.PERMISSION_GRANTED == packageManager.checkPermission(Manifest.permission.
      ACCESS_FINE_LOCATION,
2 pk.packageName) || PackageManager.PERMISSION_GRANTED == packageManager.checkPermission(
      Manifest.permission.ACCESS_COARSE_LOCATION,
3 pk.packageName))
4             results.add("" + pk.applicationInfo.loadLabel(packageManager));
```

Finally, it retrieves the names of those applications and displays them on a list. Figure 5.2 shows a screen-shot of the application.

To see the full code of the application, please consult the Appendix B.1.2.

5.1.4 Location Spoofer Type Applications

In this section, we introduce the basis of location spoofer type applications. If these applications correctly protected users' location data in all situations, our work would be done as far as users' location would be protected. Consequently, it was important to thoroughly analyse these applications and deeply understand how they work.

The first step to develop a location spoofer type application is to create a *mock provider*. This mock provider is not other than one of the 3 default providers with its coordinates faked. To fake its coordinates, developers must include the `android.permission.ALLOW_MOCK_LOCATIONS` permission in the Android Manifest file as well as activate the allow mock locations setting in the device. Then, the procedure is normally as follows:

Figure 5.2: Listpermissions application running in a real device.

1. Add a test provider:

```
mgr.addTestProvider(LocationManager.NETWORK_PROVIDER, false, false, false, false, false
    , false, false, 0, 10);
```

2. Enable the test provider:

```
mgr.setTestProviderEnabled(LocationManager.NETWORK_PROVIDER, true);
```

3. Clear all previous location data:

```
mgr.clearTestProviderEnabled(LocationManager.NETWORK_PROVIDER);
```

4. Set the status of the provider:

```
mgr.setTestProviderStatus(LocationManager.NETWORK_PROVIDER,
LocationProvider.AVAILABLE,null,System.currentTimeMillis());
```

5. Subscribe to location updates:

```
mgr.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);
```

6. Create a mock location. This is normally done by calling a external method that generates the fake coordinates:

```
updatedLocation = createLocation(20,20,0);
```

7. Set the mock location to the provider:

```
mgr.setTestProviderLocation(LocationManager.NETWORK_PROVIDER, updatedLocation);
```

LBAs will query for location updates at different time intervals depending on the arguments passed to the `getLocationUpdates()` method. If LBAs queried for location updates more frequently than the location coordinates are faked, LBA would access non-faked data. In order to avoid LBAsobtain with the real location, location spoofer applications need to push the faked location before the provider does it. Therefore, location spoofers periodically push the faked location to the provider. This procedure is normally done in an independent thread which updates the data at a certain rate.

Location spoofer applications present poor mechanisms for protecting users location data. The most complex mechanism used by these applications is the obfuscation radius. As explained before, location spoofers generate random coordinates within that interval at a certain rate. For an adversary trying to discern users' actual location, the fact that the obfuscation radius is always centred in the user's actual location would lead towards the disclosure of her most common locations. Further, the mechanisms implemented in the applications are easily detectable. Following, we will present four different ways in which the usage of a location spoofer application can be uncovered. Firstly, location spoofers require the `ACCESS_MOCK_LOCATION` permission and the allow mock locations setting activated. Due to this fact, a developer can easily detect whether the allow mock locations setting is activated or not and query the user to deactivate it previously using the LBA. It is an easy verification with control flow techniques. For instance, a developer can check whether mock locations are enabled with:

```
1 if(Settings.Secure.getString(getContentResolver(),Settings.Secure.ALLOW_MOCK_LOCATION).equals
      ("0")){
2    return false;}
3 else{
4    return true;}
```

Secondly, a more radical option is to check whether the `ALLOW_MOCK_LOCATION` permission is present in the applications' Android Manifest file or not and, consequently, query the users to uninstall those applications or at least, not to use them. Below, it is shown the necessary code to check whether the `ALLOW_MOCK_LOCATION` permission is present or not:

```
1 if (PackageManager.PERMISSION_GRANTED == packageManager.checkPermission(Manifest.permission.
      ALLOW_MOCK_LOCATION,pk.
2 packageName)
```

Thirdly, mock location providers do not send or receive National Marine Electronics Association (NMEA) data. NMEA is a standard for communicating or receiving GPS data. Therefore, developers can create a NMEA listener with `addNmeaListener()` method and if they do not get any NMEA update but location is still changed in `onLocationChanged()`, then they can determine the location has been faked.

One last option is to remove the test providers, with the LocationManager's method `removeTestProvider()`. In this way, the test provider created by the location spoofer to fake the location will be removed and, therefore, the location

coordinates will not be faked.

For all the four previous reasons, implementing applications to protect users' location is not the best solution. Instead, we should go deeper on the Android stack and check whether certain mechanisms are feasible to be implemented on the lower layers.

5.2 Protecting Users' Location at the Framework Level

As it has been seen in the previous section, location spoofer applications present several drawbacks. Therefore, we should inspect the Android stack to check whether implementing LPPMs in the lower levels is possible or not. Basically, two conditions need to be fulfilled to accept the idea of implementing LPPMs at a certain level. Firstly, they need to be accessible from the user level and, secondly, they need to be unbounded by the limitations explained before. In this section we will introduce LPDroid, a novel add-on to the AOSP. This modification allows that arbitrary LPPMs, if implemented, can be transparently used by every mobile application and therefore provide a strong level of protection. This is achieved by the implementation of a inter-level communication mechanism that allows LPDroid users to communicate with the Android location framework and control its behaviour. As a proof of work, we introduce three different basic location obfuscation strategies for protecting users' location data.

Although by the time this thesis was written the latest available version of Android was Android 4.4.3, there were no real devices in the market capable of dealing with that version of Android as no drivers were available. Therefore, LPDroid was tested directly over Android 4.4.2 in a LG Nexus 5 devices. Further, we ported LPDroid over TaintDroid 4.3 and check its functionality in an emulator.

5.2.1 Downloading and Building the AOSP

The first necessary step before modifying the lower levels of the Android stack is to download the AOSP. The steps followed to download the source code are well explained at the AOSP website[3]. We only needed to specify the version that we wanted to download:

```
1 repo init -u https://android.googlesource.com/platform/manifest -b android-4.2.2_r2
```

After we downloaded the AOSP, we downloaded the drivers for the device. The necessary drivers[4] for the LG Nexus 5 device are:

- NFC, Bluetooth, WiFi:
  `https://dl.google.com/dl/android/aosp/broadcom-hammerhead-kot49h-a670ed75.tgz`
- Camera, Sensors, Audio:
  `https://dl.google.com/dl/android/aosp/lge-hammerhead-kot49h-e6165a67.tgz`

[3]AOSP official website: `https://source.android.com/`
[4]`https://developers.google.com/android/nexus/drivers`

- Graphics, GSM, Camera, GPS, Sensors, Media, DSP, USB:
  `https://dl.google.com/dl/android/aosp/qcom-hammerhead-kot49h-518133bf.tgz`

We put the drivers in the root directory of the AOSP and extracted them with the following command:

```
tar -zxvf FILE_NAME
./FILE_NAME.sh # (view the license and then type "I ACCEPT")
```

At this point, we had everything we needed to start implementing LPDroid. Before doing that, however, we compiled the source code and run it in the device. The following procedure, will be also used to compile the modified AOSP and to port it to the real device. In order to compile the AOSP, the first step consists in initializing the building environment and choosing the target to build. To initialize the building environment, the command `. build/envsetup.sh` needs to be executed and the target, `aosp_hammerhead-userdebug` in our case, specified. The general structure of the target name is `BUILD_NAME-BUILD_TYPE`. In the `BUILD_NAME` field, the codename referring to the particular device is specified. For instance, `aosp_arm` is used to install the AOSP in an ARM emulator or `aosp_hammerhead` in a LG Nexus 5 device. In the `BUILD_TYPE` field, one of the three different build types can be specified: (1) `user` to have limited access to the device's internals (suited for mass production), (2) `userdebug` to have root access to the device's internals and (3) `eng` to have access to a development configuration with further debugging capabilities. Later, the source code needs to be compiled with the command `make`. The `make` command can handle parallel tasks with N arguments, and it is common to use a number of tasks N that is between 1 and 2 times the number of hardware threads on the computer being used for the build. As we were dealing with CentOS 6.5 Core 2 Quad machine (four cores) we executed the `make -j8` command. Once the building process is finished, the `boot.img`, `system.img` and `userdata.img` files are created.

To allow Linux machines to recognize external devices, certain `udev` rules need to be added in the `/etc/udev` directory. The `udev` rules allow developers to identify devices based on their properties, like vendor ID and device ID, dynamically in a Linux machine. The necessary rule to identify the LG Nexus 5 in the CentOs machine was:

```
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4ee1", ENV{ACL_MANAGE}=="1"
```

Once the device was correctly identified we unlocked the *bootloader*. Literally, a bootloader is the code that is executed before any OS starts. Every Android phone has a bootloader that instructs the Linux kernel to boot normally. Bootloaders are usually locked on Android devices because manufacturers want users to stick to their Android OS version specifically designed for the device. However, bootloaders can be unlocked by executing the `fastboot oem unlock` command and following the instructions showed on the device's screen. Note that unlocking the bootloader nullifies the guarantee.

Finally, we moved to the `out/target/product/hammerhead` directory and executed the 3 following commands to install the `.img` files in our device:

1. `sudo fastboot flash boot boot.img`
2. `sudo fastboot flash system system.img`
3. `sudo fastboot flash userdata userdata.img`

At this point, we were able to start implementing LPDroid. After a thorough analysis of the AOSP, we identified that the LocationManager class from the location framework needed to be modified prepare it for the future implementation of LPPMs. Related work such as the AppFence project [17] has also modified this class to implement different location obfuscation mechanisms. All the files we need to modify are in the `<ROOT>/frameworks/base/location/java/android/location` directory. Within this directory we find the GPS Manager, the Address Manager, the Country Manager, the Location and the class of interest: the Location Manager. The `android.location.LocationManager` class is the most important class in the location framework. It provides access to the system location services. These services allow applications to obtain periodic updates of the device's geographical location, or to fire an Intent object when the device enters the proximity of a given geographical area.

5.2.2 TaintDroid

As commented in the introduction of this chapter, our work was brought over TaintDroid in order to extend its functionality by preparing it for the future implementation of LPPMs. TaintDroid is an extension of Android capable of tracking sensitive data leakage developed by *Enck et al.* [16]. However, TaintDorid only covers the Java part of the Android OS (User level and Framework level) meaning that the C/C++ levels from the Android software stack are not monitored.

What TaintDroid basically does is to monitor how third-party applications handle users' data using dynamic taint analysis techniques [35]. Taint analysis is based on inspecting the source code as it executes being a commonly used technique in the malware analysis and vulnerabilities discovery. TaintDroid firstly identifies sensitive information at a `taint source` where a *taint marking* (taint tag), indicating the information type, is assigned as:

```
1 int tag = Taint.TAINT_NAME;
2 Taint.addTaintDouble(VARIABLE_TO_TRACK, tag)
```

Dynamic taint analysis tracks how labelled data impacts other data in a way that might leak the original sensitive information. This tracking is often performed at the instruction level and produces only a 14% performance overhead. Finally, the impacted data is identified before it leaves the system at a *taint sink* (usually the network interface, but can be others).

For downloading and building TaintDroid we followed the instructions in the Appanalysis website[5]. However, the necessary procedure to run TaintDroid in an emulator is not correctly described as one more step is required. Developers need to

[5]TaintDroid official website: `http://appanalysis.org/`

add an SD card to the emulator in order to allow taint propagation. Moreover, the SD card needs to have an EXT2 file system. The complete procedure is as follows:

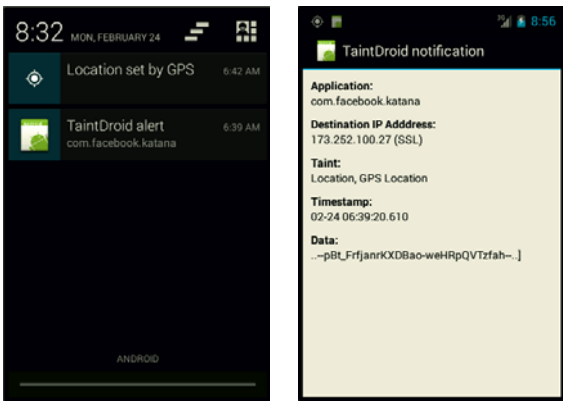
1. Create a SD Card Image:
   `mksdcard -l mySdCardFAT 32M mySdCardFATFile.img`

2. Change Filesystem to EXT2:
   `cp mySdCardFATFile.img myTdCardEXTFile.img`
   `sudo mke2fs myTdCardEXTFile.img`

Finally, we run TaintDorid on the emulator the emulator with the SD card by executing the following command:

```
emulator -kernel <PATH_TO_THE_KERNEL> -image <PATH_TO_THE_IMAGE> -sdcard <PATH_TO_THE_SDCARD>
```

As shown in Figure 5.3, TaintDroid correctly worked on the emulator. Image (b) shows the name of the application that is about to send your sensitive data (`com.facebook.katana`), where will it send it (@IP: 173.252.100.27) and the type of sensitive data (GPS location) amongst others.



(a) **(b)**

Figure 5.3: Image (a) shows how the Action bar of the emulator notifies a message from TaintDroid. Image (b) shows the information of the message.

5.2.3 AppFence

AppFence[6] [17] is an extension of TaintDroid 2.1 (based on Android 2.1). It implements two privacy controls that covertly substitute obfuscated data in place of data that the user wants to keep private and block network transmissions that contain that private data. Although Android 2.1 was already deprecated when we started

[6]AppFence official website: `http://appfence.com/`

the research, AppFence served as a starting point for implementing the LPPMs in the framework level. In this way, we first ported the mechanisms that allowed AppFence to obfuscate the location data to TaintDroid 4.3. Basically, in AppFence, the location data is replaced by certain fixed coordinates (-122.084026, 37.421265).

When a new location in received in AppFence, the regular work flow from the *vanilla* Android OS is interrupted by a call to the `fakeLocation()` method. This method is responsible for replacing the newly received coordinates by the fixed coordinates described in the previous paragraph. Figure 5.4 shows the described procedure in the *vanilla* Android OS and in AppFence.

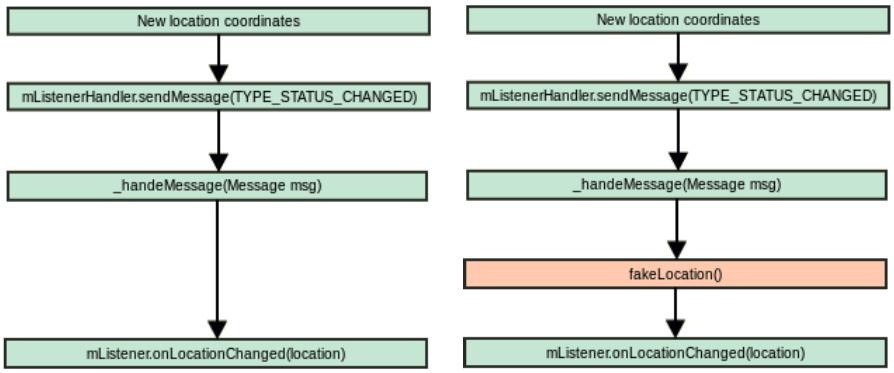
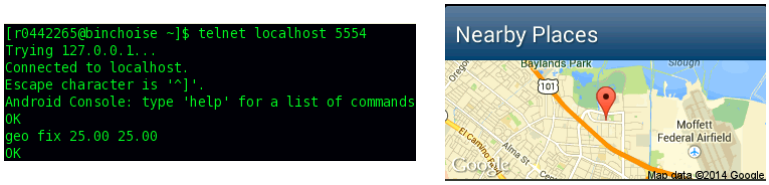


Figure 5.4: Image (a) shows the procedure followed every time new location coordinates are received. Image (b) shows how AppFence interrupts the work flow by calling the fakeLocation() method.

There is another problem the AppFence team had to deal with. When a new location is received, it is stored as the *last known location* before the callback method `onLocationChanged()` is triggered. Consequently, even though AppFence obfuscates the coordinates, if an application makes a call to the `getLastKnownLocation()` method to retrieve the last known coordinates, those will not be obfuscated. To fix this issue, AppFence calls the `fakeLocation()` method before retrieving the location data in the `getLastKnownLocation()` method.

Figure 5.5 shows the correct functionality of the AppFence code brought over TaintDroid 4.3. In image (a) we can see how the coordinates 25.0 25.0 are emulated through the `geo fix` command. Firstly, a `telnet` connection needs to be stablished. Normally, the port that connects with the device or the emulator is set to 5554. Hence, a `telnet` connection can be stablished with the `telnet localhost 5554` command. Later, with the `geo fix` command, which allows to set location coordinates emulating the GPS behaviour, we set the (25.0, 25.0) coordinates which play the role of the real coordinates. Image (b) shows how the interpreted coordinates by other LBAs are the faked locations (-122.084026, 37.421265).

Although we took the implementations of AppFence as a basis, we saw that the implemented mechanism to protect users' location data was not feasible as it turns LBAs into useless applications. Further, it is easy for a third-party server to



(a) **(b)**

Figure 5.5: Image (a) shows how the telnet session is established and how the (25.00, 25.00) coordinates are established with the `geo fix` command. Image (b) shows how Location Bases Applications interpret the (-122.084026, 37.421265) coordinates

infer that the user's location is being obfuscated. Therefore, in the next section, we implement three different LPPMs which allow users to protect their location data while being able to take advantage of LBAs.

5.2.4 LPDroid

So far, we have introduced TaintDroid and AppFence. Basically, we have illustrated how TaintDroid is capable of tracking users data and how AppFence provides several mechanisms for obfuscating this data and block it in case of leakage. However, AppFence is not available in the current version on Android. Furthermore, when focusing on a concrete type of data, e.g. location data, the obfuscation mechanisms implemented by AppFence are very limited.

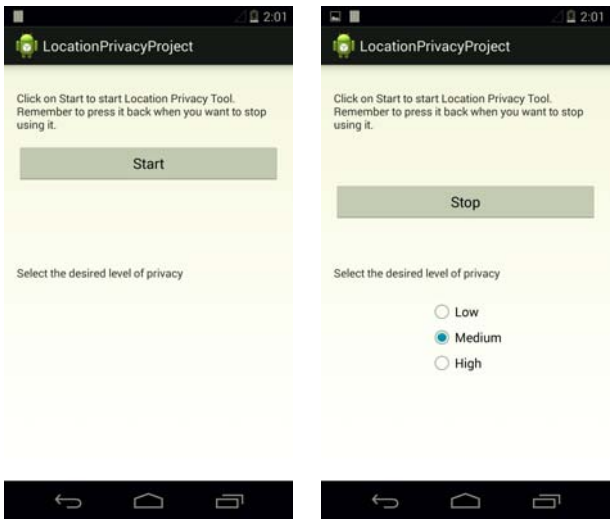
In this section we will introduce LPDroid, a novel add-on to the AOSP. This modification allows that arbitrary LPPMs, if implemented, can be transparently used by every mobile application and therefore provide a strong level of protection. This is achieved by the implementation of a inter-level communication mechanism that allows LPDroid users to communicate with the Android location framework and control its behaviour. As a proof of work, we introduce three different basic location obfuscation strategies for protecting users' location data.

User Level

To allow users communicate with the framework, we implemented an inter-level communication mechanism through an Android application. The layout of the application was designed to be clear and functional. It consists in a start/stop button and a RadioGroup object to select the desired level of privacy. Figure 5.6 shows the application's layout.

Please note that the three implemented obfuscation strategies are trivial. Our goal was to demonstrate that implementing LPPMs at the location framework was feasible. However, the implementation of more sophisticated LPPMs such as the ones described in Chapter 2 was not feasible in terms of time. We will address this issue in Chapter 6. Although AppFence demonstrated that it is feasible to implement LPPMs at the framework level, it was the first time an inter-level communication

mechanism was developed in order to allow LPDroid users to modify the behaviour of the location framework.



(a) **(b)**

Figure 5.6: In image (a) we see the state of the application when the LPPMs are stopped. Image (b) shows how, once the LPPMs are activated, the level of privacy can be selected.

Please remember from Chapter 3 that Android applications are sandboxed. However, Android provides three different mechanisms for allowing applications to communicate: intents, binders or messengers, and broadcast receivers. Further, a structured set of data can be shared amongst application using a content provider. Content providers enable developers to encapsulate different types of data, such as telephone numbers, and provide mechanisms for defining data security. For instance, a content provider can specify a permission which will be required to access its data.

Establishing a communication process was unnecessary as far as sharing data was a valid and an easier solution. Consequently, we defined a flag to determine whether the LPPMs should be activated or not, and another flag to determine the desired level of privacy. We decided to work with integer type flags. Consequently, `PRIVACY_ACTIVATED` = 0 would mean that the LPPMs should not be used while `PRIVACY_ACTIVATED` = 1 would activate them. Further, `PRIVACY_LEVEL` = 1, 2 or 3 would indicate the level of privacy low, medium or high respectively.

At this point we needed to decide whether we would create a new content provider or use an existent one. One of the characteristics this content provider should have is that it should be system-wide. As creating a new content provider is a rather complex task, we decided to modify the Settings Provider, a system-wide content provider, to include the mentioned flags. Modifying the Settings Provider to include the two integer-type flags, required modifying the framework level. Hence, we will address this issue in the Framework level section.

In the application, once the user selects an option, the corresponding flag value will be appropriately modified. This can be done by calling the `putInt()` method

from the Settings class:

```
1 android.provider.Settings.System.putInt(getApplicationContext().getContentResolver(), android.
      provider.Settings.System.PRIVACY_ACTIVATED, VALUE);
2 android.provider.Settings.System.putInt(getApplicationContext().getContentResolver(), android.
      provider.Settings.System.PRIVACY_LEVEL, VALUE);
```

This code shows how for dealing with content providers we need to create a ContentResolver object which, at the same time, requires to be aware of the application's context. Then, we just need to specify the flag to be modified and its new value.

Besides, our application stores the selected options even if the device is switched off. So for instance, if a user always wants to have the LPPMs activated at the highest level of privacy, the application will remember it. Moreover, the default level of privacy is set to medium when the application is launched for the first time. The medium level was set as a default level for being a fair privacy level.

At the next step, we added the application to the AOSP in order to include it as a system application. System applications are placed under `/system/app` folder in an Android device. As this folder is a read-only folder, system applications cannot be uninstalled. To include an application as a system application, we first define the `Android.mk` file of the application. Please note that the package name must be correctly specified as it will be used in the next step. The `.mk` needs to be defined as follows:

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES := $(call all-java-files-under, src)
LOCAL_PACKAGE_NAME := LocationPrivacyProject
include $(BUILD_PACKAGE)
```

Secondly, we need to create a link inside the `<ROOT>/packages/apps` directory to refer to our application folder. This link can be created through the `ln-s<PATH\_TO\_OUR\_APPLICATION\_FOLDER>` command. Finally, in `<ROOT>/build/target/product/core.mk` we added the package name of the application in the `PRODUCT_PACKAGES` list.

When the application was finished we started the modifications at the Framework level. The first necessary modification that we needed to do was to find a way to retrieve the values from the Settings provider into the Framework level. Then, once we knew whether the user wants to use the tool and at which level, we would use the corresponding obfuscation mechanisms. This will be addressed in the next section.

Framework Level

As explained in the previous sections, the only class from the Location framework that was modified to implement LPDroid was the LocationManager class. As every class, it contains the declaration of the necessary variables, the constructor and

certain methods. It is important to notice that the constructor includes a Context object of the LocationManager class. The Context class is an abstract class whose implementation is provided in the AOSP. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc. Although a Context is normally related to applications, it can also be defined for classes in the framework. This point was crucial by the time of developing an inter-level communication application. Without a Context, an instance of the ContentResolver class cannot be created and, therefore, the ContentProviders' content cannot be accessed.

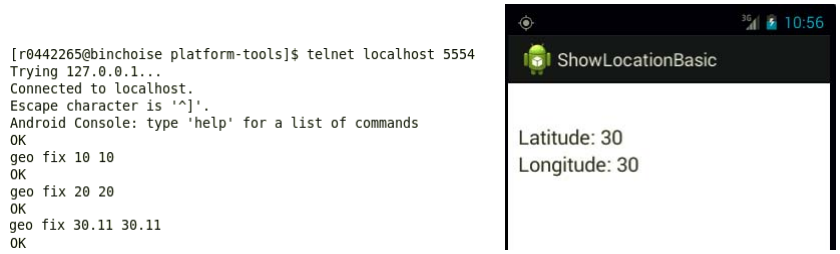
Before implementing the mechanisms to access the Settings provider, we modified it to incorporate the explained flags. The Settings provider is located in `<ROOT>/frameworks/base/core/java/android/provider` where we added the `PRIVACY_ACTIVATED` and the `PRIVACY_LEVEL` flags.

Then, we proceeded to implement three different obfuscation strategies. For the lowest level we implemented a strategy that replaces the actual location by rounded coordinates. For the medium level, the strategy replaces the coordinates by a fixed ones. Finally, for the highest level, the implemented strategy replaces the location data by randomly generated coordinates.

The different strategies were all implemented within the `fakeLocation()` method. This method reads the flags from the settings provider and determines whether it should use a LPPM, if implemented, and, if yes, which one. To implement the strategy in the lowest case, we simply cast the current location to convert it into an integer value. The strategy from the medium was implemented by setting a fixed longitude and latitude values. Concretely, the ESAT department's coordinates (50.862346, 4.686362) were selected. Implementing this tool required 2 different steps. Firstly, we declare the fixed coordinates. Secondly, we make a call to a method for setting the fixed coordinates as the actual coordinates. Finally, the obfuscation strategy that generates random coordinates makes use of the a method to generate random values for the latitude and the longitude. Please notice that to check whether the LPPMs, if implemented, should be activated and which LPPM should be used, the `checkPrivacy()` and `checkPrivacyLevel()` methods are used to read the content provider's flags set from the application. We provide the source code of the `fakeLocation()` method in the Appendix B.2.

We brought LPDroid firstly over TaintDroid and run it on an emulator. As a proof of work, Figure 5.7 shows how the lowest level strategy works: when coordinates (30.11, 30.11) are set, it only shows (30, 30).

As far as we wanted to test the obfuscation strategies in a real device, we brought LPDroid directly over the *vanilla* Android 4.4.2. We check the correct functionality of the three strategies using the *OpenPaths* application. Figure 5.8 shows the different effects regarding to each one of the implemented obfuscation strategies. Image (a) was taken as the reference path. Images (b), (c) and (d) show the coordinates that were captured by the OpenPaths application when going over the same path with the different strategies activated. With the lowest level there is only one coordinate shown as the path was not long enough to imply a change in the order of the units.



(a) **(b)**

Figure 5.7: Lowest level obfuscation strategy working on the emulator. Image (a) shows how the telnet connection is established and how the (30.11, 30.11) coordinates are set with the `geo fix` command. Image (b) shows the coordinates read by the LBA.

(a) **(b)** **(c)** **(d)**

Figure 5.8: Different obfuscation strategies working on the LG Nexus 5 device. Image (a) shows the followed path. Image (b) shows the captured coordinates when following the same path with the lowest level obfuscation strategy activated. Image (c) shows the captured coordinates when following the same path with the medium level obfuscation strategy activated. Image (d) shows the captured coordinates when following the same path with the high level obfuscation strategy activated

5.2.5 Conclusions

At this point of our research, we developed a novel add-on to the AOSP that paves the way for future developers to implement arbitrary LPPMs and transparently use them by any application. Although the obfuscation strategies implemented are trivial, it shows how location data can be obfuscated preventing it from all the mechanisms described in Section 5.4.1 to check whether the location has been faked or not. Once the LPPMs are implemented in LPDroid, it is not necessary to activate the *"Allow mock locations"* option in the Settings menu nor it is necessary to declare the `ALLOW_MOCK_LOCATIONS` in the LPDroid application. Moreover, we are not using Test providers and therefore the NMEA data remains as part of the location meta-data.

Our work extends AppFence in the way that it provides a inter-level communica-

tion mechanism which allows users communicate with the framework level and modify its behaviour. Users could make use of our extended framework when TaintDroid notifies them of a location data leakage. Suppose that TaintDroid notifies a user that her location data is going to be sent towards a third-party server. Hence, the user could decide to activate our LPPMs. Please notice that, in last instance, we are relaying on TaintDroid to know whether our sensitive data is being leaked or not.

5.3 Lower Levels of the AOSP

So far, we know that TaintDroid can monitor every data leakage that occurs at the Java level. This means that it can track not only third party applications but also the Java portion of the Android OS. This means that if any mechanisms to steal sensitive data were implemented in the lower levels (C/C++ coded levels), TaintDroid would not be aware of them. According to Enk et al. [16], the the Android OS can be trusted and therefore, no mechanisms that could steal sensitive data from the users could be found in its basis.

In this work, we considered this assumption as dangerous and we studied what happens at the lower levels. Hence, as described before, there remain two sections to be analyzed: the Linux Kernel and the Hardware Support.

5.3.1 The Kernel and the HAL Layers

At the lowest level, the Android platform provides the security of the Linux kernel. The Linux kernel itself has been in widespread use for years, and is used in millions of environments. Through its history of constantly being enhanced, attacked, and fixed by thousands of developers, Linux has become a stable and secure kernel trusted by many corporations and security professionals. Therefore, we can assume that the Kernel is free from code that steals sensitive data from the users even with the added "androidizms".

As explained in Chapter 3, the Android stack typically relies on drives provided by manufacturers to interact with the hardware. This fact seriously limits our research: a device manufacturer can create a basic driver that implements the minimum mechanisms to control a certain component and make that driver available under the General Public License (GPL). However, enhanced functionalities would be implemented within a proprietary driver in the user space and will not be available publicly. Consequently, sensitive data security threads could be implemented at this point by the vendor and we would remain unaware. Investigating this issue would require a lot of time as we should first find a way to access all the vendors' drivers and then analyse them. We will leave this analysis for the future work.

Chapter 6

Future Work

6.1 Improving 3Ov

3Ov is a static analysis tool capable of identifying over-privileges, studying the increase of permissions due to the presence of third-party libraries and identifying missing and mistaken permissions in Android applications. As seen in Chapter 4, 3Ov provides an upper and a lower bound for the number of over-privileges an application has. When a certain permission (not required by the Application but present in the manifest) is required by more than one third-party library and is assumed as an over-privilege, it defines the upper bound. When it is not assumed as an over-privilege, it defines the lower bound.

There is a way in which we could determine the exact number of over-privileges an application has. To this end, we should evaluate each third-party library separately and map its methods to certain Android permissions. In the end, we propose an extension of the PScout tool capable of mapping third-party libraries rather than only the AOSP. The implementation of this tool would require an extensive knowledge of the different third-party libraries. This fact rises several difficulties. Firstly, we should be aware of all the third-party libraries present on the applications she is about to analyse. This first issue can be solved with the bash script provided with 3Ov (`find3PLs.sh`). Secondly, we should define a parser that searched for method calls in the third-party libraries. Once a method call was identified, the parser program should identify the Android API calls within that method. Hence, the parser would be able to relate the method call with certain Android permissions. In the end, the parser would come up with a list containing all the methods found in the third-party library and their required permissions. Once we had a map between all the third-party libraries' methods and the Android permissions, we would create method arrays to later identify those arrays within the application. In this way, we would solve the problem of knowing whether an application will use one of the methods from the third-party libraries that require a certain permission or not.

Finally, we would like to improve 3Ov to avoid it being so limited when analysing obfuscated libraries. To this end, we should thoroughly study all the different obfuscation tools aimed to obfuscate Android applications. Therefore, we would

know what is really happening under the hood when application developers obfuscate their applications. Consequently, we would maybe be able to programmatically classify the obfuscated libraries into one third-party library category.

6.2 Improving LPDroid

As mentioned in Chapter 5, the LPPMs implemented in the LPDroid framework are trivial. Due to time constrains, we limited our research to demonstrate that the implementation of LPPMs at the framework level is feasible. In future work, we would like to enhance the LPPMs implemented in LPDroid. As described in Chapter 2, obfuscation-based LPPMs lower the quality of the location information. In order to avoid that, LPPMs based on cryptographic primitives have been proposed in the literature. However, we cannot rely on these crypto LPPMs as their require a trusted service provider. Therefore, we would like to extend LPDroid with the formal implementation of a *differential privacy* based LPPM. Such LPPMs appears appear to be particularly suited, because they impose a rather small computational overhead.

At this point, we would like to provide LPDroid with a former implementation of a mechanism κ to achieve Geo-indistinguishability. In [36], Andres et al. introduce this concept for the first time. They propose a model where χ is a set of Points Of Interests (POIs) and $\mathcal{Z}$ a set of possible *reported values* (for instance, values sent through the network). Also, they assume an operational scenario of a user located at $x \in \chi$ and communicating to the attacker a randomly selected location $z \in \mathcal{Z}$. The attacker's side information can be modelled by a *prior* distribution π on χ, where $\pi(\chi)$ is the probability assigned to the location x. Our mechanism κ should then be a function for assigning to each location $x \in \chi$ a probability distribution on $\mathcal{Z}$. This mechanism κ satisfies ϵ-geo-indistinguishability iff for all x, x':

$$d_P(\kappa(x), \kappa(x')) \leq \epsilon d(x, x') \tag{6.1}$$

Chapter 7

Conclusion

As of May 2014, Android is the most popular smart-phone OS in the world being present in the 79% of them. With Android an ever growing number of Android applications have become essential for individuals' daily life. These applications rely on the usage of third-party libraries that extend the basic functionalities included in the Android APIs. The usage of these third-party libraries leads Android developers towards the necessity of declaring further permissions in the application' manifest file. Some of these permissions grant the applications with certain permissions to access users' sensitive data. The aim of this work was to show that Android applications are over-privileged and, furthermore, illustrate how the usage of different third-party libraries leads towards the necessity of declaring different permissions which can be likelier to become an over-privilege under different circumstances. If applications were shown to be over-privileged, we needed to implement further mechanisms that would allow Android users to protect their sensitive data.

In this work we have introduced 3Ov and LPDroid. The former is a static analysis tool that inspects the increase of permissions due to the usage of third-party libraries and the tendency to over-privilege applications. Further, it identifies mistaken permissions in Android applications and missing permissions in the manifest file. The latter is a novel add-on to the AOSP which paves the way for future implementations of LPPMs in Android systems.

We analysed a set $\mathcal{A}$ of 1126 Android applications and a subset $\mathcal{B}$ of 89 applications with 3Ov. By inspecting the set $\mathcal{A}$ we identified the presence of 76 third-party libraries not included in the Appbrain lists. Two libraries out of the 76 were Advertising libraries while the other 74 were Development tools libraries. Further, we provided an overview of the usage of third-party libraries in Android applications as of May 2014 observing that Facebook, Android supporting libraries and Goolge/Ads were the most popular Social SDK, Development tools and Advertising libraries respectively. By analysing set $\mathcal{B}$ we considered the definition of five different heuristics to define the term *over-privilege*. In all the situations, 3Ov identifies the presence of over-privileges in the analyzed applications.

In front of the results showed by 3Ov and the previous work stating the leakage of data [10, 11], we implemented LPDroid. It is a novel add-on to the AOSP which

allows arbitrary LPPMs, if implemented, to be transparently used by every mobile application and therefore provide a strong level of protection. This is achieved by the implementation of a inter-level communication mechanism that allows LPDroid users to communicate with the Android location framework and control its behaviour. As a proof of work, we introduce three different basic location obfuscation strategies for protecting users' location data.

With this work we have shown the existence of over-privileges in Android applications that guarantee third-party libraries with access to users' sensitive data. To the best of our knowledge, it has been the first time that the presence of over-privileges has been analysed regarding the three different third-party libraries categories. Further, we have provided an overview of the most popular third-party libraries as of May 2014. Also, we have studied the implications the usage of these libraries have in terms of the declared permissions in the manifest file. Finally, we have shown that the correct place to implement LPPMs is the location framework rather than with applications and we have provided an add-on to the AOSP that sets the necessary tools for doing that.

Appendices

Appendix A

Source Code of 3Ov

A.1 Crawling the applications

Following, we provide the modified version of Singh's code for crawling the Google Play market.

```
'''
Created on Aug 28, 2013
Modified on Feb 23, 2014

@author: anuvrat
@adaptation: roger_ribas

This script has been modified to only download information
for the current free applications in the Google Play market.

A part from that, it has been modified in order to work
with current applications information format.

'''

from bs4 import BeautifulSoup
import urllib.request
import urllib.parse
import codecs
import json
import pickle
from datetime import datetime

def loadState():
    try:
        state_file = open( "state_dump", "rb" )
        apps_discovered = pickle.load( state_file )
        apps_pending = pickle.load( state_file )
        #apps_count = pickle.load( state_file )
        state_file.close()
        print( "Pending = ", len( apps_pending ), " Discovered = ", len( apps_discovered ) )
        return apps_discovered, apps_pending
    except IOError:
        print( "A fresh start ..." )
        return [], []

character_encoding = 'utf-8'
apps_discovered, apps_pending = loadState()
```

```
39 count_offset = len( apps_discovered )
40
41 start_time = datetime.now()
42
43 def getPageAsSoup( url, post_values ):
44     if post_values:
45         data = urllib.parse.urlencode( post_values )
46         data = data.encode( character_encoding )
47         req = urllib.request.Request( url, data )
48     else:
49         req = url
50     try:
51         response = urllib.request.urlopen( req )
52     except urllib.error.HTTPError as e:
53         print( "HTTPError with: ", url, "\t", e )
54         return None
55     the_page = response.read()
56     soup = BeautifulSoup( the_page )
57
58     return soup
59
60 def openResultFiles(all_categories):
61     fileHandlers = {}
62     for category in all_categories:
63         fileHandler = codecs.open( '_'.join( ["apps", category.lower()] ), 'ab',
               character_encoding, buffering = 0 )
64         fileHandlers[category] = fileHandler
65     return fileHandlers
66
67 def closeResultFiles(fileHandlers):
68     for v in fileHandlers.values():
69         v.close()
70
71 def reportProgress():
72     current_time = datetime.now()
73     elapsed = current_time - start_time
74     v = ( ( len( apps_discovered ) - count_offset ) / elapsed.seconds ) * 60
75     t = len( apps_pending ) / v if v > 0 else 0
76     print( "Pending = ", len( apps_pending ), " Discovered = ", len( apps_discovered ), "
           Velocity = ", str( v ), " apps per minute and time remaining in minutes = ", str( t )
            )
77
78 def saveState():
79     state_file = open( "state_dump", "wb" )
80     pickle.dump( apps_discovered, state_file )
81     pickle.dump( apps_pending, state_file )
82     state_file.close()
83     reportProgress()
84
85 def getAppDetails(app_url):
86     if app_url in apps_discovered:
87         print( "Skip because already parsed: ", app_url )
88         return None
89
90     g_app_url = 'https://play.google.com' + app_url
91     app_details = {}
92
93     app_details['app_url'] = g_app_url
94
95     soup = getPageAsSoup( g_app_url, None )
96     if not soup: return None
97
98     apps_discovered.append( app_url )
```

```
99
100     print( g_app_url )
101     print( "HOLA ESTIC A 3" )
102     title_div = soup.find( 'div', {'class':'document-title'} )
103     app_details['title'] = title_div.find( 'div' ).get_text().strip()
104
105     subtitle = soup.find( 'a', {'class' : 'document-subtitle primary'} )
106     app_details['developer'] = subtitle.get_text().strip()
107     app_details['developer_link'] = subtitle.get( 'href' ).strip()
108
109     ####################MODIFICATION#################
110
111     # The code used to get stuck at this point, after removing it the code works
112
113     #price_buy_span = soup.find( 'span', {'class' : 'price buy'} )
114     #price = price_buy_span.find_all( 'span' )[-1].get_text().strip()
115     #price = price[:-4].strip() if price != 'Install' else 'Free'
116     #app_details['price'] = price
117
118     rating_value_meta = soup.find( 'meta', {'itemprop' : 'ratingValue'} )
119     app_details['rating'] = rating_value_meta.get( 'content' ).strip()
120
121     reviewers_count_meta = soup.find( 'meta', {'itemprop' : 'ratingCount'} )
122     app_details['reviewers'] = reviewers_count_meta.get( 'content' ).strip()
123
124     num_downloads_div = soup.find( 'div', {'itemprop' : 'numDownloads'} )
125     if num_downloads_div: app_details['downloads'] = num_downloads_div.get_text().strip()
126
127     date_published_div = soup.find( 'div', {'itemprop' : 'datePublished'} )
128     app_details['date_published'] = date_published_div.get_text().strip()
129
130     operating_systems_div = soup.find( 'div', {'itemprop' : 'operatingSystems'} )
131     app_details['operating_system'] = operating_systems_div.get_text().strip()
132
133     content_rating_div = soup.find( 'div', {'itemprop' : 'contentRating'} )
134     app_details['content_rating'] = content_rating_div.get_text().strip()
135
136     category_span = soup.find( 'span', {'itemprop' : 'genre'} )
137     app_details['category'] = category_span.get_text()
138
139     for dev_link in soup.find_all( 'a', {'class' : 'dev-link'} ):
140         if dev_link.get_text().strip() == "Email Developer":
141             app_details['email'] = dev_link.get( 'href' ).strip()[7:]
142         elif dev_link.get_text().strip() == "Visit Developer's Website":
143             app_details['dev_website'] = dev_link.get( 'href' ).strip()
144
145     badge_span = soup.find( 'span', {'class' : 'badge-title'} )
146     if badge_span: app_details['badge'] = badge_span.get_text().strip()
147
148     for more_apps in soup.find_all( 'div', {'data-short-classes' : 'card apps square-cover
          tiny no-rationale'} ):
149         more_app_url = more_apps.find( 'a', {'class' : 'card-click-target'} ).get( 'href' )
150         if more_app_url not in apps_discovered and more_app_url not in apps_pending:
              apps_pending.append( more_app_url )
151     print( "DONE IT" )
152     return app_details
153
154 def getTopAppsData( url, start, num, app_type ):
155     values = {'start' : start,
156               'num': num,
157               'numChildren':'0',
158               'ipf': '1',
159               'xhr': '1'}
```

```
160     soup = getPageAsSoup( url, values )
161     if not soup: return [], []
162 
163     apps = []
164     skipped_apps = []
165 
166     ###################MODIFICATION#################
167 
168     # If we don't reference app_details in here we would get an error
169     app_details = {}
170     for div in soup.findAll( 'div', {'class' : 'details'} ):
171         title = div.find( 'a', {'class':'title'} )
172         try:
173             app_details = getAppDetails( title.get( 'href' ) )
174         except AttributeError:
175             pass
176         if app_details: apps.append( app_details )
177         else: skipped_apps.append( title.get( 'href' ) )
178 
179     return apps, skipped_apps
180 
181 def getApps( url ):
182     previous_apps = []
183     previous_skipped_apps = []
184     start_idx = 0
185     size = 100
186     while(True):
187         apps, skipped_apps = getTopAppsData( url, start_idx, size, app_type )
188         if apps == previous_apps and skipped_apps == previous_skipped_apps: break
189         for app in apps:
190             if app['category'].upper() not in fileHandlers:
191                 fileHandlers[app['category'].upper()] = codecs.open( '_'.join( ["apps", app['
                        category'].lower()] ), 'ab', character_encoding, buffering = 0 )
192             fileHandler = fileHandlers[app['category'].upper()]
193             try:
194                 fileHandler.write( json.dumps( app ) + "\n" )
195             except Exception as e:
196                 print( e )
197         previous_apps = apps
198         previous_skipped_apps = skipped_apps
199         start_idx += size
200         saveState()
201 
202 categories = ['BOOKS_AND_REFERENCE', 'BUSINESS', 'COMICS', 'COMMUNICATION', 'EDUCATION', '
        ENTERTAINMENT', 'FINANCE', 'HEALTH_AND_FITNESS', 'LIBRARIES_AND_DEMO', 'LIFESTYLE', '
        APP_WALLPAPER', 'MEDIA_AND_VIDEO', 'MEDICAL', 'MUSIC_AND_AUDIO', 'NEWS_AND_MAGAZINES', '
        PERSONALIZATION', 'PHOTOGRAPHY', 'PRODUCTIVITY', 'SHOPPING', 'SOCIAL', 'SPORTS', 'TOOLS',
         'TRANSPORTATION', 'TRAVEL_AND_LOCAL', 'WEATHER', 'ARCADE', 'BRAIN', 'CARDS', 'CASUAL', '
        GAME_WALLPAPER', 'RACING', 'SPORTS_GAMES', 'GAME_WIDGETS']
203 
204 ###################MODIFICATION#################
205 
206 #app_types = ['free', 'paid']
207 
208 # We just want to take into account the free ones
209 app_types = ['free']
210 
211 fileHandlers = openResultFiles( categories )
212 
213 for category, app_type in [( x, y ) for x in categories for y in app_types]:
214     print( "Type = ", app_type, " Cateory = ", category )
215     url = 'https://play.google.com/store/apps/category/' + category + '/collection/topselling_
          ' + app_type
```

```
216     getApps( url )
217
218 ###################MODIFICATION#################
219
220 #top_urls = ['https://play.google.com/store/apps/collection/topselling_paid_game',
221  #          'https://play.google.com/store/apps/collection/topselling_free',
222  #          'https://play.google.com/store/apps/collection/topselling_paid',
223  #          'https://play.google.com/store/apps/collection/topgrossing',
224  #          'https://play.google.com/store/apps/collection/topselling_new_paid_game',
225  #          'https://play.google.com/store/apps/collection/topselling_new_free',
226  #          'https://play.google.com/store/apps/collection/topselling_new_paid']
227
228 # We just one to take into account the free ones
229 top_urls = ['https://play.google.com/store/apps/collection/topselling_free',
230            'https://play.google.com/store/apps/collection/topselling_new_free',
231            'https://play.google.com/store/apps/collection/topgrossing']
232
233 for url in top_urls:
234     print( "Crawl collection - ", url )
235     getApps( url )
236
237 errorUrls = codecs.open('_'.join(["apps", "error"]), 'ab', character_encoding, buffering = 0)
238 count = 100
239 while apps_pending:
240     if count == 0:
241         saveState()
242         count = 100
243     count = count - 1
244
245     app = apps_pending.pop()
246     try:
247         app_data = getAppDetails( app )
248     except AttributeError:
249         pass
250     if not app_data:
251       continue
252     if app_data['category'].upper() not in fileHandlers:
253         fileHandlers[app_data['category'].upper()] = codecs.open( '_'.join( ["apps", app_data[
              'category'].lower()] ), 'ab', character_encoding, buffering = 0 )
254     fileHandler = fileHandlers[app_data['category'].upper()]
255     try:
256         fileHandler.write( json.dumps( app_data ) + "\n" )
257     except Exception as e:
258         print( e )
259
260 errorUrls.close()
261 print( 'Complete' )
262 closeResultFiles( fileHandlers )
```

A.2 Downloading the applications

Once we had the list containing the 19 335 urls from the applications, we developed the following Python-coded program to download the `.apk` files using the modified `Apkdownloader.crx` extension. Notice that an email address (Gmail) and its password must be added. Further, the phone's ID number must also be specified. Applications such as `Device ID` can be used to this end.

The code has been tested in a 10 Mbps internet connection. Hence, we set an idle time of 1 minute every 5 downloads to avoid overtaking the limit number of

simultaneous downloads. The idle time was empirically tested and set.

```
'''
Roger Ribas
roger.ribas90@gmail.com
Mar 27, 2014

This program automates the download of the 19335 application whose URLs
are stored in the file urls.txt

'''

from selenium import webdriver
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.support.ui import WebDriverWait # available since 2.4.0
from selenium.webdriver.chrome.options import Options
import time

# Create an options object
chrome_options = Options()
# Create an option with our modified version of the APK downloader extension
chrome_options.add_extension("./apk-downloader-2 .0-test.crx")
# Create a new instance of the Chromium driver
driver = webdriver.Chrome(executable_path="./chromedriver",chrome_options=chrome_options)

# Setting the preferences of the APK downloader extension
print("Setting the preferences...")
driver.get("chrome-extension://kdlcoedfnbhajfpbiafnnkfojgipmlje/options.html")
inputElement1 = driver.find_element_by_id("user_email")
inputElement1.send_keys("YOUR_EMAIL")
inputElement2 = driver.find_element_by_id("user_password")
inputElement2.send_keys("YOUR_EMAIL_PASSWORD")
inputElement3 = driver.find_element_by_id("user_device_id")
inputElement3.send_keys("YOUR_PHONE_ID")

buttonElement = driver.find_element_by_id("btn_login")
buttonElement.click()
time.sleep(5)

#Bucle that iterates over the URLs file
print("Starting the downloading process")
downloadsCounter = 0
total = 17231
i = 0
f = open('urls.txt', 'rU')
for line in f.readlines():
   driver.get(line)
   downloadsCounter = downloadsCounter + 1
   i = i + 1
   #we have to give some time to start the download
   time.sleep(5)
   #as the number of simultaneous downloads is limited we give 1 minute of time every 5
        simultaneous downloads
   if(i > 4):
      print("Downloads: " + str(downloadsCounter) + ", Remaining: " + str(total -
           downloadsCounter) + ", Percentage: " + str(downloadsCounter/total*100) + "%" )
      time.sleep(60)
      i = 0

if(downloadsCounter == total):
   print( "All the apks have been downloaded: " + str(downloadsCounter))
else:
   print( "Download finished with some missing applications. There should be: " + str(total) +
```

```
        ", and there are: " + str(downloadsCounter))
driver.quit()
```

A.3 Parsing AOSP's permissions

The following bash script parses the AOSP's permissions files generated with PScout. It generates a text file for each permission in which we can find all the method arrays from the methods that require the mentioned permission. Please notice how the script includes the partial results within the `methodpermissions` (method names), `commaspermissions` (number of variables), `classpermissions` (class names), `rtypepermissions` (return type) and `vtypepermissions` (variables type) directories. It displays the first two method arrays for the first two entries from each permission.

```
#!/bin/bash

#
# Roger Ribas
# roger.ribas90@gmail.com
# 15th of April 2014
#
# In this script we are going to parse the output files from the
# PScout tool. Hence, we will create arrays containing 5 fields
# which will work as unique identifiers of a method.
#
# We are going to consider 5 heuristics:
#  1) The name of the method
#  2) The name of the class to which the method belongs
#  3) The number of arguments that are passed to the method
#  4) The type of variables that are passed to the method
#  5) The return type of the method
#

i=0
z=0
limit=0
methodcounter=0

mkdir ./permissions

for permission in `find /volume1/scratch/r0442265/pscout4.4/pscout_android_4_4_output/
    Permissions_slashed -type f -name '*.txt'`
do
   echo "Parsing $permission ..."
   permissionname=`echo $permission | awk -F"/" '{print $NF}'`
   permissionnamewithouttxt=`echo $permissionname | awk -F"." '{print $1}' `

   mkdir ./permissions/$permissionnamewithouttxt
   cd ./permissions/$permissionnamewithouttxt
   mkdir methodspermissions
   mkdir commaspermissions
   mkdir classpermissions
   mkdir rtypepermissions
   mkdir vtypepermission
   mkdir resultspermission


```

```
43    # Gets the method name from the Permissions file
44    for line in `cat $permission | awk '{print $3}' | awk -F"(" '{print $1}'`
45    do
46       method=$line
47       methodsv[i]=$line
48       i=`expr $i + 1`
49       methodcounter=`expr $methodcounter + 1`
50    done
51    echo ${methodsv[@]} >> methodspermissions/methodspermissions.${permissionname}
52    i=0
53
54    # Gets the number of variables per method and gets the type of variables the longest
          method has 16 variables so we we'll check them all
55    IFS=$'\n'
56    for line in `cat $permission`
57    do
58       wordcount=`echo $line | awk -F"(" '{print $2}' | awk -F")" '{print $1}' | wc -w`
59       if [ "$wordcount" -gt "$limit" ]; then
60          commas=`echo $line | tr -cd "," | wc -c`
61          commas=`expr $commas + 1`
62
63          arr=`echo $line | awk -F"(" '{print $2}' | awk -F")" '{print $1}' | tr "," "\n"`
64          for z in $arr
65          do
66             z=`echo $z | awk -F"/" '{print $NF}'`
67             vtypev[i]+=" $z"
68          done
69       else
70          commas=0
71       fi
72       commasv[i]=$commas
73       i=`expr $i + 1`
74    done
75    echo ${commasv[@]} >> commaspermissions/commaspermissions.${permissionname}
76    echo ${vtypev[@]} >> vtypepermission/vtypetypespermission.${permissionname}
77    i=0
78    i=0
79
80    # Gets the class name where the method belongs
81    for line in `cat $permission | awk '{print $1}' | awk -F":" '{print $1}' | awk -F"<" '{
          print $2}'`
82    do
83       class=$line
84       classv[i]=$class
85       i=`expr $i + 1`
86    done
87    echo ${classv[@]} >> classpermissions/classpermissions.${permissionname}
88    i=0
89
90    # Gets the return type of the method
91    for line in `cat $permission | awk '{print $2}' | awk -F"/" '{print $NF}'`
92    do
93       rtype=$line
94       rtypev[i]=$rtype
95       i=`expr $i + 1`
96    done
97    echo ${rtypev[@]} >> rtypepermissions/rtypepermissions.${permissionname}
98    i=0
99
100   methodcounter=`expr $methodcounter - 1`
101
102   while [[ $i -lt $methodcounter ]]
103   do
```

```
104       echo "${methodsv[$i]} ${commasv[$i]} ${classv[$i]} ${rtypev[$i]} ${vtypev[$i]}" >>
              resultspermission/resultspermission.${permissionname}
105       i=‘expr $i + 1‘
106     done
107     i=0
108     methodcounter=0
109
110     cd ..
111     cd ..
112
113     echo "
            ------------------------------------------------------------------------------------------------------
            "
114     echo "${methodsv[0]} ${commasv[0]} ${classv[0]} ${rtypev[0]} ${vtypev[0]}"
115     echo "${methodsv[1]} ${commasv[1]} ${classv[1]} ${rtypev[1]} ${vtypev[1]}"
116     echo "
            ------------------------------------------------------------------------------------------------------
            "
117
118     methodsv=()
119     commasv=()
120     classv=()
121     rtypev=()
122     vtypev=()
123
124     echo "done"
125 done
```

A.4 Reverse-engineering the applications

We developed a bash scrip program to automate the reverse-engineering process. Further, it parses the permissions declared en each application's Android Manifest file. The scrip uses the `Apktool` to reverse-engineer the applications and then parses the `AndroidManifest.xml` file looking for the `<uses-permission>` tag in order to identify the permissions.

```
1  #!/bin/sh
2
3  #
4  # Roger Ribas
5  # roger.ribas90@gmail.com
6  # 7th of April 2014
7  #
8  # This script proceeds to reverse engineer each apk file found
9  # in the current directory. In order to do that we use the apktool
10 # which output the java files of the application in a readable smali
11 # format.
12 #
13 # Further, we analyze each app's AndroidManifest.xml file to extract
14 # the permission this app requires at the installation time. The
15 # permissions shown in the output file are only those permissions
16 # defined in the Android API. Hence, we do not consider any permission
17 # defined by the user or by 3rdPLs.
18 #
19
20 counter1=0
21 counter2=0
22
```

```
23 mkdir ./apktool/output
24
25 for file in *.apk
26 do
27     echo "Reverse engineering $file"
28     filename=‘echo $file | awk -F".apk" ’{print $1}’‘
29     java -jar /volume1/scratch/r0442265/Analysis/apktool.jar decode "$file" apktool/output/"
           $filename"
30     counter1=‘expr $counter1 + 1‘
31 done
32 echo "Total number of apps reversed: $counter1"
33
34 cd ./apktool/output
35
36 for dir in ‘find -maxdepth 1 -type d‘
37 do
38    if [ "$dir" = "." ] || [ "$dir" = ./manifest ];then
39       continue
40    fi
41
42    filename=‘echo $dir | awk -F"/" ’{print $2}’‘
43
44    cd $dir
45
46    cat "AndroidManifest.xml" | grep "uses-permission" | grep "android.permission" | awk -F’"’
          ’{print $2}’ > AndroidManifest.txt
47    sort AndroidManifest.txt -o AndroidManifest.txt
48    counter2=‘expr $counter2 + 1‘
49
50    cd ..
51 done
52
53 echo "Total number of manifests analyzed: $counter2"
54
55 if [ "$counter1" -eq "$counter2" ];then
56     echo "All manifests parsed correctly"
57 else
58     echo "Not all the manifests were parsed correctly"
59 fi
```

A.5 Find third-party libraries

We analyzed the applications looking for third-party libraries that were not included in the Appbrain lists. The following table shows the libraries we found. In some cases, no further information about their purpose was found. Although they were seldom used by the applications, we found that the Air library was used by 17 applications.

Name	#	Purpose	Category
Acme/Serve	1	Java web server	Development tools
afzkl/development	2	Color picker component for Android	Development tools
air	17	In-app billing Adobe extension	Development tools
amerisoft	1	Mobile applications development company	Development tools
anywheresoftware/b4a	1	Development environment for Android	Development tools
balofo	1	Mobile applications development company	Development tools
biz/source_code	1	Base64 encoder/decoder class. There is no Base64 encoder/decoder in the standard Java SDK	Development tools

BusinessObjects	1	Mobile applications development company	Development tools
com/ansca	1	Cross-platform development environment	Development tools
com/arthisoft	1	Mobile applications development company	Development tools
com/chenlb	1	General development tools	Development tools
com/dexati	2	Mobile applications development company	Development tools
com/directtap	1		Development tools
com/doitflash	1	Android application builder	Development tools
com/ffmpeg	1	FFmpeg support for Android	Development tools
com/freshplanet	5	In-app purchase	Development tools
com/github	1	Github support for Android	Development tools
com/handlerexploit	1	Mobile applications development company	Development tools
com/ice	1	Ice for Android offers support for designing networked applications	Development tools
com/ironsource	2	Development environment for Android	Development tools
com/isgmqlqdubhodykn	1		Development tools
com/jirbo	1	Mobile applications development company	Development tools
com/koushikdutta	1	Cyanogen's developer own library	Development tools
com/lsgvgames	1	Mobile applications development company	Development tools
com/madelephantstudios	1	Mobile applications development company	Development tools
com/magnetjoy	1	Mobile applications development company	Development tools
com/millennialmedia	1	Development environment for Android	Development tools
com/mysql	1	SQLite support for Android	Development tools
com/omniture	1	Adobe applications analysis and marketing platform	Development tools
com/ormma	1	Open Rich Media Mobile Advertising is an advertising library	Advertisement library
com/poqop	1	Unknown Android developer own library	Development tools
com/purplebrain	1	Mobile applications development company	Development tools
com/rabbit	3	Mobile applications development company	Development tools
com/rootsoft	1	Mobile applications development company	Development tools
com/samsungapps	1	Mobile applications development company	Development tools
com/smilerlee	1	Mobile applications development company	Development tools
com/soundtouch	2	SoundTouch source code package compiles SoundTouch source codes into Android native library	Development tools
com/tinidream	1	Mobile applications development company	
com/vtvcj	1		Development tools
com/wenbin	1	Unknown Android developer own library	Development tools
com/ylvwpqe	4		Development tools
com/zemariamm	1	Mobile applications development company	Development tools
com/pcsoft	1	Development environment for Android	Development tools
com/mobi	1	Cross platform push messaging, application promotion, in-application purchasing and integrated analytics tools	Development tools
org/anddev	1	Android development community	

org/haxe	1	Cross-platform UI components	Development tools
org/teleal	1	UPnP-compatible software stack for Java environments	Development tools
ramadan	1		Development tools
androi	1	Mobile applications development company	Development tools
AndroView	1	Mobile applications development company	Development tools
aurelienribon	3	Aurelien Ribon's own library	Development tools
cmn	8	Mobile applications development company	Development tools
com/appboy	1	Development environment for Android	Development tools
com/appflood	3	Development environment for Android	Development tools
com/badlogic	4	Mobile applications development company	Development tools
com/coffrsra	2		
com/eightbitmage	5	Development environment for Android	Development tools
com/gamexc	1	Virus!	
com/gblzllib	1	C/C++ support library	Development tools
com/googlecode	1	It is a project to port Android open source project to x86 platforms	Development tools
com/ivasionsoft	1	Mobile applications development company	Development tools
com/lyx	1	Mobile applications development company	Development tools
com/MASTAdView	1	Advertising library embedded in the mOcean SDK for Android	Advertising libraries
com/planarsoft	1	Mobile applications development company	Development tools
com/playnomics	1	Development environment for Android	Development tools
com/pleaserateus	1	Support library for rate-me-type dialog	Development tools
com/squareup	1	Support libraries for Square Up devices	Development tools
com/sun	1	Sun/Oracle support libraries	Development tools
com/tremorvideo	1	Mobile applications development company	Development tools
imoblife	1	Mobile applications development company	Development tools
javax	1	JavaX support for Android devices	Development tools
nanoxml	1	NanoXML support library for Android devices	Development tools
net/rbgrn	1	Robert Green's own library	Development tools
org/libsdl	1	SDL support for Android devices	Development tools

Table A.1: Shows the name of the found third-party libraries, the number of applications which made use of these libraries, their aim and their category.

The following bash script analyzes the applications seeking for new third-party libraries which are not present in the Appbrain lists.

```
1  #!/bin/sh
2
3  #
4  # Roger Ribas
5  # roger.ribas90@gmail.com
6  # 5th of May 2014
7  #
8  # This script finds all the 3rdPLs present in all the downloaded
9  # applications from the Google Play.
10 #
11
12 i=0
13 j=0
14 limit=0
```

```
15 methodcounter=0
16 counter=0
17
18 IFS=$'\n'
19
20 for app in `find * -maxdepth 0 -type d`
21 do
22    if [ "$app" = "manifest" ];then
23       continue
24    fi
25
26    cd $app
27
28    nameapp=`pwd | awk -F"/" '{print $NF}'`
29    tmp1=`echo $nameapp | awk -F"." '{print $1,"/",$2}'`
30    dirapp=`echo $tmp1 | tr -d " "`
31    echo $nameapp
32
33    cd smali
34
35    for dir in `find * -maxdepth 1 -type d`
36    do
37       if [ "$dir" = "$dirapp" ];then
38          continue
39       fi
40
41       dirname="Directory: $dir"
42       echo $dirname >> /volume1/scratch/r0442265/Analysis/apktool/output/$nameapp/results3pls.
             txt
43
44       for pl in `cat /volume1/scratch/r0442265/Analysis/apktool/adlibraries.txt`
45       do
46          tmp=`echo "$dir" | grep -c -i -F "$pl"`
47          counterpartial1=`expr $counterpartial1 + $tmp`
48          counter1=`expr $counter1 + $tmp`
49          tmp=0
50       done
51
52       for pl in `cat /volume1/scratch/r0442265/Analysis/apktool/developmenttools.txt`
53       do
54          tmp=`echo "$dir" | grep -c -i -F "$pl"`
55          counterpartial2=`expr $counterpartial2 + $tmp`
56          counter2=`expr $counter2 + $tmp`
57          tmp=0
58       done
59
60       for pl in `cat /volume1/scratch/r0442265/Analysis/apktool/socialsdks.txt`
61       do
62          tmp=`echo "$dir" | grep -c -i -F "$pl"`
63          counterpartial3=`expr $counterpartial3 + $tmp`
64          counter3=`expr $counter3 + $tmp`
65          tmp=0
66       done
67
68       if [ $counterpartial1 -eq 0 ] || [ $counterpartial2 -eq 0 ] || [ $counterpartial3 -eq 0
             ];then
69          echo $dirname >> /volume1/scratch/r0442265/Analysis/apktool/output/new3PLs.txt
70       fi
71    done
72
73    echo "Number of AdLibraries found: $counterpartial1" >> /volume1/scratch/r0442265/Analysis/
          apktool/output/$nameapp/results3pls.txt
74    echo "Number of Support Libraries found: $counterpartial2" >> /volume1/scratch/r0442265/
```

```
        Analysis/apktool/output/$nameapp/results3pls.txt
75    echo "Number of Social sdks found: $counterpartial3" >> /volume1/scratch/r0442265/Analysis/
        apktool/output/$nameapp/results3pls.txt
76
77    counterpartial1=0
78    counterpartial2=0
79    counterpartial3=0
80
81    cd ..
82    cd ..
83  done
84
85    echo "Number of AdLibraries found: $counter1" >> /volume1/scratch/r0442265/Analysis/
        apktool/output/results3pls.txt
86    echo "Number of Support Libraries found: $counter2" >> /volume1/scratch/r0442265/Analysis/
        apktool/output/results3pls.txt
87    echo "Number of Social sdks found: $counter3" >> /volume1/scratch/r0442265/Analysis/
        apktool/output/results3pls.txt
```

A.6 Parsing and analyzing the applications

The following bash script parses the `.smali` files from the reverse-engineered applications and creates the method arrays. This method arrays will be stored in text files which will be classified depending on their category. Notice how the translation of the return variables type and the argument variables type into the standard format is done here.

```
1   #!/bin/bash
2
3   #
4   # Roger Ribas
5   # roger.ribas90@gmail.com
6   # 15th of April 2014
7   #
8   # This script parses the smali files of each application and
9   # creates vectors containing the 5 heuristics. Then, we will
10  # try to find for equivalences between the vectors from the
11  # permission files and the ones created here. If there is a
12  # matching, it will mean that this application needs that
13  # permission.
14  #
15  # We are going to consider 5 heuristics:
16  #  1) The name of the method
17  #  2) The name of the class to which the method belongs
18  #  3) The number of arguments that are passed to the method
19  #  4) The type of variables that are passed to the method
20  #  5) The return type of the method
21  #
22
23  i=0
24  j=0
25  limit=0
26  methodcounter=0
27  counter=0
28
29  IFS=$'\n'
30
31  for app in `find ./output -maxdepth 1 -type d`
```

```
32 do
33    if [ "$app" = "./output/manifest" ] || [ "$app" = "./output" ];then
34       continue
35    fi
36
37    cd $app
38
39    nameapp=‘pwd | awk -F"/" ’{print $NF}’‘
40    tmp1=‘echo $nameapp | awk -F"." ’{print $1,"/",$2}’‘
41    dirapp=‘echo $tmp1 | tr -d " "‘
42
43    mkdir resultssmaliapp
44    mkdir resultssmaliadlibraries
45    mkdir resultssmalidevelopmenttools
46    mkdir resultssmalisocialsdks
47    mkdir resultssmaliothers
48
49    for smali in ‘find * -type f -name ’*.smali’‘
50    do
51       tmp2=‘echo $smali | awk -F"smali/" ’{print $2}’ | awk -F"/" ’{print $1,"/",$2}’‘
52       currentdir=‘echo $tmp2 | tr -d " "‘
53       echo $currentdir
54       echo $dirapp
55       echo "Parsing $smali ..."
56       smaliname=‘echo $smali | awk -F"/" ’{print $NF}’‘
57       smalinamewithoutsmali=‘echo $smaliname | awk -F"." ’{print $1}’ ‘
58
59       # Gets the method name from the Smali file
60       for line in ‘cat $smali | grep "invoke" | awk -F"->" ’{print $2}’ |awk -F"(" ’{print $1
             }’‘
61       do
62          name=$line
63          methodsv[i]=$line
64          i=‘expr $i + 1‘
65          methodcounter=‘expr $methodcounter + 1‘
66       done
67       i=0
68
69       # Gets the number of variables per method and the package type of variables
70       for line in ‘cat $smali | grep "invoke"‘
71       do
72          wordcount=‘echo $line | awk -F"(" ’{print $2}’ | awk -F")" ’{print $1}’ | wc -w‘
73          if [ "$wordcount" -gt "$limit" ];then
74             semicolumn=‘echo $line | awk -F"(" ’{print $2}’ | awk -F")" ’{print $1}’ | tr -cd
                   ";" | wc -c‘
75             if [ "$semicolumn" -eq "$limit" ];then
76                uppercaseletters=‘echo $line | awk -F"(" ’{print $2}’ | awk -F")" ’{print $1}’ |
                        tr -cd "[:upper:]" | wc -c‘
77                semicolumn=$uppercaseletters
78             fi
79
80             arr=‘echo $line | awk -F"(" ’{print $2}’ | awk -F")" ’{print $1}’ | tr ";" "\n"‘
81             for z in $arr
82             do
83                if [ "$z" = "V" ];then
84                   z="void"
85                   semicolumn=‘expr $semicolumn + 1‘
86                elif [ "$z" = "I" ];then
87                   z="int"
88                   semicolumn=‘expr $semicolumn + 1‘
89                elif [ "$z" = "C" ];then
90                   z="char"
91                   semicolumn=‘expr $semicolumn + 1‘
```

```
92              elif [ "$z" = "Z" ];then
93                 z="boolean"
94                 semicolumn=`expr $semicolumn + 1`
95              elif [ "$z" = "B" ];then
96                 z="byte"
97                 semicolumn=`expr $semicolumn + 1`
98              elif [ "$z" = "S" ];then
99                 z="short"
100                semicolumn=`expr $semicolumn + 1`
101             elif [ "$z" = "J" ];then
102                z="long"
103                semicolumn=`expr $semicolumn + 1`
104             elif [ "$z" = "F" ];then
105                z="float"
106                semicolumn=`expr $semicolumn + 1`
107             elif [ "$z" = "D" ];then
108                z="double"
109                semicolumn=`expr $semicolumn + 1`
110             elif [ "$z" = "VV" ];then
111                z="void void"
112                semicolumn=`expr $semicolumn + 1`
113             elif [ "$z" = "II" ];then
114                z="int int"
115             elif [ "$z" = "CC" ];then
116                z="char char"
117             elif [ "$z" = "ZZ" ];then
118                z="boolean boolean"
119             elif [ "$z" = "BB" ];then
120                z="byte byte"
121             elif [ "$z" = "SS" ];then
122                z="short short"
123             elif [ "$z" = "JJ" ];then
124                z="long long"
125             elif [ "$z" = "FF" ];then
126                z="float float"
127             elif [ "$z" = "DD" ];then
128                z="double double"
129             fi
130             z=`echo $z | awk -F"/" '{print $NF}'`
131             vtypev[i]+=" $z"
132          done
133
134       else
135          semicolumn=0
136       fi
137       commasv[i]=$semicolumn
138       i=`expr $i + 1`
139    done
140    i=0
141
142    # Gets the class package name where the method belongs
143    for line in `cat $smali | grep "invoke" | awk -F"," '{print $NF}' | awk -F";" '{print $1
           }' | awk -F" L" '{print $2}'`
144    do
145       classes=$line
146       classv[i]=$classes
147       i=`expr $i + 1`
148    done
149    i=0
150
151    # Gets the return type package of the method
152    for line in `cat $smali | grep "invoke" | awk -F")" '{print $2}' | awk -F";" '{print $1
           }'`
```

```
153       do
154          types=$line
155          if [ "$types" = "V" ];then
156             types="void"
157          elif [ "$types" = "I" ];then
158             types="int"
159          elif [ "$types" = "C" ];then
160             types="char"
161          elif [ "$types" = "Z" ];then
162             types="boolean"
163          elif [ "$types" = "B" ];then
164             types="byte"
165          elif [ "$types" = "S" ];then
166             types="short"
167          elif [ "$types" = "J" ];then
168             types="long"
169          elif [ "$types" = "F" ];then
170             types="float"
171          elif [ "$types" = "D" ];then
172             types="double"
173          else
174             types=`echo $line | awk -F"/" '{print $NF}'`
175          fi
176          rtypev[i]=$types
177          i=`expr $i + 1`
178       done
179       i=0
180
181       # Decides whether it is a support library file, a social network library file or an
              AdLibrary file
182       filename="adlibraries"
183       for pl in `cat /volume1/scratch/r0442265/Analysis/apktool/adlibraries.txt`
184       do
185          tmp=`echo "$currentdir" | grep -c -i -F "$pl"`
186          counter1=`expr $counter1 + $tmp`
187          tmp=0
188       done
189       if [ $counter1 -gt 0 ];then
190          outdir=$filename
191       fi
192
193       filename="developmenttools"
194       for pl in `cat /volume1/scratch/r0442265/Analysis/apktool/developmenttools.txt`
195       do
196          tmp=`echo "$currentdir" | grep -c -i -F "$pl"`
197          counter2=`expr $counter2 + $tmp`
198          tmp=0
199       done
200       if [ $counter2 -gt 0 ];then
201          outdir=$filename
202       fi
203
204       filename="socialsdks"
205       for pl in `cat /volume1/scratch/r0442265/Analysis/apktool/socialsdks.txt`
206       do
207          tmp=`echo "$currentdir" | grep -c -i -F "$pl"`
208          counter3=`expr $counter3 + $tmp`
209          tmp=0
210       done
211       if [ $counter3 -gt 0 ];then
212          outdir=$filename
213       fi
214
```

```
215       if [ $counter1 -eq 0 ] && [ $counter2 -eq 0 ] && [ $counter3 -eq 0 ];then
216          outdir="others"
217       fi
218       counter1=0
219       counter2=0
220       counter3=0
221
222       while [[ $i -lt $methodcounter ]]
223       do
224          if [ "$currentdir" = "$dirapp" ];then
225             echo "${methodsv[$i]} ${commasv[$i]} ${classv[$i]} ${rtypev[$i]} ${vtypev[$i]}" >>
                    ./resultssmaliapp/resultssmaliapp.${smalinamewithoutsmali}.txt
226          else
227             echo "${methodsv[$i]} ${commasv[$i]} ${classv[$i]} ${rtypev[$i]} ${vtypev[$i]}" >>
                    ./resultssmali$outdir/resultssmali3rd.${smalinamewithoutsmali}.txt
228          fi
229          i=‘expr $i + 1‘
230       done
231       i=0
232       methodcounter=0
233
234       echo "
              ----------------------------------------------------------------------------------------------------
              "
235       echo "${methodsv[0]} ${commasv[0]} ${classv[0]} ${rtypev[0]} ${vtypev[0]}"
236       echo "${methodsv[1]} ${commasv[1]} ${classv[1]} ${rtypev[1]} ${vtypev[1]}"
237       echo "
              ----------------------------------------------------------------------------------------------------
              "
238
239       methodsv=()
240       commasv=()
241       classv=()
242       rtypev=()
243       vtypev=()
244
245       echo "done"
246       echo ""
247    done
248
249    cd ..
250    cd ..
251 done
```

A.7 Generating the Android Manifest files

Next, we introduce the script used to compare the method arrays from the applications with the method arrays from the permissions. Notice how we excluded the R-type files from the applications to avoid analyzing them. R-type files, such as `R$layout` or `R$menu`, are automatically generated by the builder and they do not belong to the application.

```
1  #!/bin/bash
2
3  #
4  # Roger Ribas
5  # roger.ribas@gmail.com
6  # 28th April 2014
```

```
7  #
8  # This script tries to find each method in the permission
9  # files in the application.
10 #
11 # We will distinguish between the code which belongs to the application
12 # itself and the code that belongs to the added 3rdPLs: AdLibraries,
13 # Development Tools, Social SDKs and Others.
14 #
15 # Finally it generates a GeneratedManifest file for each category: The
16 # 3rdPLs and the Application.
17 #
18
19 counter1=0
20 counter2=0
21 counter3=0
22 counter4=0
23 counter5=0
24
25
26 IFS=$'\n'
27
28 for app in `find * -maxdepth 0 -type d`
29 do
30    echo "Application: $app"
31    if [ "$app" = "manifest" ];then
32       continue
33    fi
34
35    for permission in `find /dev/shm/Analysis/apktool/permissions -type f -name '
          resultspermission.*.txt'`
36    do
37       echo "Checking $permission ..."
38       permissionname=`echo $permission | awk -F"/" '{print $NF}'`
39       permissionnamewithouttxt=`echo $permissionname | awk -F"." '{print $2}'`
40
41       IFS=$'\n'
42       for line in `cat $permission`
43       do
44          echo "Looking for $line in App files from $app"
45          for smali in `find /dev/shm/Analysis/apktool/output/$app/resultssmaliapp -type f -
               name "*.txt"`
46          do
47             smaliname=`echo $smali | awk -F".txt" '{print $1}' | awk -F"." '{print $NF}'`
48             if [ $smaliname = 'R$attr' ] || [ $smaliname = 'R$bool' ] || [ $smaliname = '
                    R$dimen' ] || [ $smaliname = 'R$drawable' ] || [ $smaliname = 'R$id' ] || [
                    $smaliname = 'R$layout' ] || [ $smaliname = 'R$menu' ] || [ $smaliname = '
                    R$string' ] || [ $smaliname = 'R$style' ];then
49                continue
50             fi
51             tmp1=`grep -i -c -F "$line" $smali`
52             counter1=`expr $counter1 + $tmp1`
53             tmp1=0
54             grep -i -F "$line" $smali >> ./$app/methodsresultapp.txt
55          done
56
57          echo "Looking for $line in AdLibraries files from $app"
58          for smali in `find /dev/shm/Analysis/apktool/output/$app/resultssmaliadlibraries -
               type f -name "*.txt"`
59          do
60             tmp2=`grep -i -c -F "$line" $smali`
61             counter2=`expr $counter2 + $tmp2`
62             tmp2=0
63             grep -i -F "$line" $smali >> ./$app/methodsresultadlibraries.txt
```

```
64          done
65
66          echo "Looking for $line in SupportLibraries files from $app"
67          for smali in `find /dev/shm/Analysis/apktool/output/$app/resultssmalidevelopmenttools
                -type f -name "*.txt"`
68          do
69             tmp3=`grep -i -c -F "$line" $smali`
70             counter3=`expr $counter3 + $tmp3`
71             tmp3=0
72             grep -i -F "$line" $smali >> ./$app/methodsresultdevelopmenttools.txt
73          done
74
75          echo "Looking for $line in SocialLibraries files from $app"
76          for smali in `find /dev/shm/Analysis/apktool/output/$app/resultssmalisocialsdks -type
                f -name "*.txt"`
77          do
78             tmp4=`grep -i -c -F "$line" $smali`
79             counter4=`expr $counter4 + $tmp4`
80             tmp4=0
81             grep -i -F "$line" $smali >> ./$app/methodsresultsocialsdks.txt
82          done
83
84          echo "Looking for $line in Others files from $app"
85          for smali in `find /dev/shm/Analysis/apktool/output/$app/resultssmaliothers -type f -
                name "*.txt"`
86          do
87             tmp5=`grep -i -c -F "$line" $smali`
88             counter5=`expr $counter5 + $tmp5`
89             tmp5=0
90             grep -i -F "$line" $smali >> ./$app/methodsresultothers.txt
91          done
92       done
93
94       echo $counter1
95       echo $counter2
96       echo $counter3
97       echo $counter4
98       echo $counter5
99
100      echo "" >> ./$app/GeneratedManifestApp.txt
101      echo "" >> ./$app/GeneratedManifestadlibraries.txt
102      echo "" >> ./$app/GeneratedManifestdevelopmenttools.txt
103      echo "" >> ./$app/GeneratedManifestsocialsdks.txt
104      echo "" >> ./$app/GeneratedManifestothers.txt
105
106      if [ $counter1 -gt 0 ];then
107         echo "android.permission.$permissionnamewithouttxt" >> ./$app/GeneratedManifestApp.
                txt
108      fi
109      counter1=0
110
111      if [ $counter2 -gt 0 ];then
112         echo "android.permission.$permissionnamewithouttxt" >> ./$app/
                GeneratedManifestadlibraries.txt
113      fi
114      counter2=0
115
116      if [ $counter3 -gt 0 ];then
117         echo "android.permission.$permissionnamewithouttxt" >> ./$app/
                GeneratedManifestdevelopmenttools.txt
118      fi
119      counter3=0
120
```

```
121       if [ $counter4 -gt 0 ];then
122          echo "android.permission.$permissionnamewithouttxt" >> ./$app/
                 GeneratedManifestsocialsdks.txt
123       fi
124       counter4=0
125       if [ $counter5 -gt 0 ];then
126          echo "android.permission.$permissionnamewithouttxt" >> ./$app/GeneratedManifestothers.
                 txt
127       fi
128       counter5=0
129    done
130
131    echo "All permissions correctly analyzed"
132    echo "-------------------------------------"
133 done
```

A.8 Discussion

The following bash script was used to analyze the Generated Manifest files. It outputs the unnecessary permissions present in each application as well as its over-privileges. Finally, it presents the total number of unnecessary permissions and over-privileges found in all the applications.

```
1  #!/bin/bash
2  #
3  # Roger Ribas
4  # roger.ribas90@gmail.com
5  # 6th of May 2014
6  #
7  # This script analyses the output results of the Compare script.
8  # It counts the number of Over-privileges as well as the number
9  # of Unecessary permissions.
10 #
11
12 IFS=$'\n'
13
14 counter=0
15 counter2=0
16 counter3=0
17 counter4=0
18 tmp=0
19
20 echo "" > ResultsNameUnecessary.txt
21 echo "" > ResultsNumbersUnecessary.txt
22
23 echo "" > ResultsNameMissing.txt
24 echo "" > ResultsNumbersMissing.txt
25
26 echo "" > ResultsNameOverLowDev.txt
27 echo "" > ResultsNameOverUpDev.txt
28 echo "" > ResultsNumbersOverLowDev.txt
29 echo "" > ResultsNumbersOverUpDev.txt
30
31 echo "" > ResultsNameOverLowAd.txt
32 echo "" > ResultsNameOverUpAd.txt
33 echo "" > ResultsNumbersOverLowAd.txt
34 echo "" > ResultsNumbersOverUpAd.txt
35
36 echo "" > ResultsNameOverLowSoc.txt
```

```
37 echo "" > ResultsNameOverUpSoc.txt
38 echo "" > ResultsNumbersOverLowSoc.txt
39 echo "" > ResultsNumbersOverUpSoc.txt
40
41 echo "" > ResultsNameAllThirdParty.txt
42 echo "" > ResultsNumbersAllThirdParty.txt
43
44 for dir in `find -maxdepth 1 -type d`
45 do
46    if [ "$dir" = "." ] || [ "$dir" = ./manifest ];then
47    continue
48    fi
49
50    cd $dir
51
52    # Joins the AdLibraries' permissions with the Social SDKs
53    # permissions.
54    cat GeneratedManifestadlibraries.txt >> tmp.txt
55    cat GeneratedManifestsocialsdks.txt >> tmp.txt
56    awk '!x[$0]++' tmp.txt > Ad_SocialSdks.txt
57    rm tmp.txt
58    sort Ad_SocialSdks.txt -o Ad_SocialSdks.txt
59    sed '/^$/d' Ad_SocialSdks.txt > Ad_SocialSdksDef.txt
60    rm Ad_SocialSdks.txt
61
62    # Joins the DevLibraries' permissions with the Social SDKs
63    # permissions.
64    cat GeneratedManifestdevelopmenttools.txt >> tmp.txt
65    cat GeneratedManifestsocialsdks.txt >> tmp.txt
66    awk '!x[$0]++' tmp.txt > Dev_SocialSdks.txt
67    rm tmp.txt
68    sort Dev_SocialSdks.txt -o Dev_SocialSdks.txt
69    sed '/^$/d' Dev_SocialSdks.txt > Dev_SocialSdksDef.txt
70    rm Dev_SocialSdks.txt
71
72    # Joins the DevLibraries' permissions with the AdLibraries's
73    # permissions.
74    cat GeneratedManifestdevelopmenttools.txt >> tmp.txt
75    cat GeneratedManifestadlibraries.txt >> tmp.txt
76    awk '!x[$0]++' tmp.txt > Dev_Ad.txt
77    rm tmp.txt
78    sort Dev_Ad.txt -o Dev_Ad.txt
79    sed '/^$/d' Dev_Ad.txt > Dev_AdDef.txt
80    rm Dev_Ad.txt
81
82    # Joins all the third party libraries
83    cat GeneratedManifestdevelopmenttools.txt >> tmp.txt
84    cat GeneratedManifestadlibraries.txt >> tmp.txt
85    cat GeneratedManifestsocialsdks.txt >> tmp.txt
86    cat GeneratedManifestothers.txt >> tmp.txt
87    awk '!x[$0]++' tmp.txt > allthird.txt
88    rm tmp.txt
89    sort allthird.txt -o allthird.txt
90    sed '/^$/d' allthird.txt > allthirdDef.txt
91    rm allthird.txt
92
93    echo "" > ResultsUnecessary.txt
94    echo "" > ResultsMissing.txt
95    echo "" > ResultsOverLowDev.txt
96    echo "" > ResultsOverUpDev.txt
97    echo "" > ResultsOverLowAd.txt
98    echo "" > ResultsOverUpAd.txt
99    echo "" > ResultsOverLowSoc.txt
```

```
100     echo "" > ResultsOverUpSoc.txt
101     echo "" > ResultsAllThirdParty.txt
102
103     # If a permission required by the application is not present
104     # in the manifest we have a missing permission.
105     for line in `cat GeneratedManifestApp.txt`
106     do
107        counter1=`cat AndroidManifest.txt | grep -i -c "$line"`
108
109        if [ $counter1 -eq 0 ];then
110           echo "Missing permission: $line" >> ResultsMissing.txt
111        fi
112     done
113
114     counter1=0
115
116     # A permission present in Dev
117     for line in `cat GeneratedManifestdevelopmenttools.txt`
118     do
119        counter2=`cat AndroidManifest.txt | grep -i -c "$line"`
120        if [ $counter2 -gt 0 ];then
121           counter3=`cat GeneratedManifestApp.txt | grep -i -c "$line"`
122           if [ $counter3 -eq 0 ];then
123              counter4=`cat Ad_SocialSdksDef.txt | grep -i -c "$line"`
124              if [ $counter4 -eq 0 ];then
125                 echo "OverLow: $line" >> ResultsOverLowDev.txt
126              else
127                 echo "OverUp: $line" >> ResultsOverUpDev.txt
128              fi
129           fi
130        fi
131     done
132     counter2=0
133     counter3=0
134     counter4=0
135
136     # A permission present in Ad
137     for line in `cat GeneratedManifestadlibraries.txt`
138     do
139        counter2=`cat AndroidManifest.txt | grep -i -c "$line"`
140        if [ $counter2 -gt 0 ];then
141           counter3=`cat GeneratedManifestApp.txt | grep -i -c "$line"`
142           if [ $counter3 -eq 0 ];then
143              counter4=`cat Dev_SocialSdksDef.txt | grep -i -c "$line"`
144              if [ $counter4 -eq 0 ];then
145                 echo "OverLow: $line" >> ResultsOverLowAd.txt
146              else
147                 echo "OverUp: $line" >> ResultsOverUpAd.txt
148              fi
149           fi
150        fi
151     done
152     counter2=0
153     counter3=0
154     counter4=0
155
156     # A permission present in Soc
157     for line in `cat GeneratedManifestsocialsdks.txt`
158     do
159        counter2=`cat AndroidManifest.txt | grep -i -c "$line"`
160        if [ $counter2 -gt 0 ];then
161           counter3=`cat GeneratedManifestApp.txt | grep -i -c "$line"`
162           if [ $counter3 -eq 0 ];then
```

```
163             counter4=`cat Dev_AdDef.txt | grep -i -c "$line"`
164             if [ $counter4 -eq 0 ];then
165                echo "OverLow: $line" >> ResultsOverLowSoc.txt
166             else
167                echo "OverUp: $line" >> ResultsOverLowSoc.txt
168             fi
169          fi
170       fi
171    done
172    counter2=0
173    counter3=0
174    counter4=0
175
176    # A permission present in total
177    for line in `cat allthirdDef.txt`
178    do
179       counter2=`cat AndroidManifest.txt | grep -i -c "$line"`
180       if [ $counter2 -gt 0 ];then
181          counter3=`cat GeneratedManifestApp.txt | grep -i -c "$line"`
182          if [ $counter3 -eq 0 ];then
183             echo "Over: $line" >> ResultsAllThirdParty.txt
184          fi
185       fi
186    done
187    counter2=0
188    counter3=0
189    counter4=0
190
191    sort ResultsUnecessary.txt -o ResultsUnecessary.txt
192    sort ResultsMissing.txt -o ResultsMissing.txt
193
194    sort ResultsAllThirdParty.txt -o ResultsAllThirdParty.txt
195
196    sort ResultsOverLowDev.txt -o ResultsOverLowDev.txt
197    cat ResultsOverLowDev.txt >> ResultsOverUpDev.txt
198    sort ResultsOverUpDev.txt -o ResultsOverUpDev.txt
199
200    sort ResultsOverLowAd.txt -o ResultsOverLowAd.txt
201    cat ResultsOverLowAd.txt >> ResultsOverUpAd.txt
202    sort ResultsOverUpAd.txt -o ResultsOverUpAd.txt
203
204    sort ResultsOverLowSoc.txt -o ResultsOverLowSoc.txt
205    cat ResultsOverLowSoc.txt >> ResultsOverUpSoc.txt
206    sort ResultsOverUpSoc.txt -o ResultsOverUpSoc.txt
207
208
209    for ln in `cat ResultsUnecessary.txt`
210    do
211       echo $ln >> ../ResultsNameUnecessary.txt
212       tmp=`echo $ln | grep -c "Unecessary"`
213       counter5=`expr $counter5 + $tmp`
214       tmp=0
215    done
216
217    for ln in `cat ResultsMissing.txt`
218    do
219       echo $ln >> ../ResultsNameMissing.txt
220       tmp=`echo $ln | grep -c "Missing"`
221       counter6=`expr $counter6 + $tmp`
222       tmp=0
223    done
224
225    for ln in `cat ResultsOverLowDev.txt`
```

```
226     do
227        echo $ln >> ../ResultsNameOverLowDev.txt
228        tmp=‘echo $ln | grep -c "OverLow"‘
229        counter7=‘expr $counter7 + $tmp‘
230        tmp=0
231     done
232
233     for ln in ‘cat ResultsOverUpDev.txt‘
234     do
235        echo $ln >> ../ResultsNameOverUpDev.txt
236        tmp=‘echo $ln | grep -c "OverUp"‘
237        counter8=‘expr $counter8 + $tmp‘
238        tmp=0
239     done
240
241     for ln in ‘cat ResultsOverLowAd.txt‘
242     do
243        echo $ln >> ../ResultsNameOverLowAd.txt
244        tmp=‘echo $ln | grep -c "OverLow"‘
245        counter9=‘expr $counter9 + $tmp‘
246        tmp=0
247     done
248
249     for ln in ‘cat ResultsOverUpAd.txt‘
250     do
251        echo $ln >> ../ResultsNameOverUpAd.txt
252        tmp=‘echo $ln | grep -c "OverUp"‘
253        counter10=‘expr $counter10 + $tmp‘
254        tmp=0
255     done
256
257     for ln in ‘cat ResultsOverLowSoc.txt‘
258     do
259        echo $ln >> ../ResultsNameOverLowSoc.txt
260        tmp=‘echo $ln | grep -c "OverLow"‘
261        counter11=‘expr $counter11 + $tmp‘
262        tmp=0
263     done
264
265     for ln in ‘cat ResultsOverUpSoc.txt‘
266     do
267        echo $ln >> ../ResultsNameOverUpSoc.txt
268        tmp=‘echo $ln | grep -c "OverUp"‘
269        counter12=‘expr $counter12 + $tmp‘
270        tmp=0
271     done
272
273     for ln in ‘cat ResultsAllThirdParty.txt‘
274     do
275        echo $ln >> ../ResultsNameAllThirdParty.txt
276        tmp=‘echo $ln | grep -c "Over"‘
277        counter13=‘expr $counter13 + $tmp‘
278        tmp=0
279     done
280
281     #Total number of unecessary permissions.
282     echo "$counter5" >> ../ResultsNumbersUnecessary.txt
283
284     #Total number of missing permissions.
285     echo "$counter6" >> ../ResultsNumbersMissing.txt
286
287     #Total number of over-privilege lower bound permissions due to Dev.
288     echo "$counter7" >> ../ResultsNumbersOverLowDev.txt
```

```
289     #Total number of over-privilege upper bound permissions due to Dev.
290     echo "$counter8" >> ../ResultsNumbersOverUpDev.txt
291
292     #Total number of over-privilege lower bound permissions due to Ad.
293     echo "$counter9" >> ../ResultsNumbersOverLowAd.txt
294     #Total number of over-privilege upper bound permissions due to Ad.
295     echo "$counter10" >> ../ResultsNumbersOverUpAd.txt
296
297     #Total number of over-privilege lower bound permissions due to Soc.
298     echo "$counter11" >> ../ResultsNumbersOverLowSoc.txt
299     #Total number of over-privilege upper bound permissions due to Soc.
300     echo "$counter12" >> ../ResultsNumbersOverUpSoc.txt
301
302     #Total number of over-privileges due to all third party.
303     echo "$counter13" >> ../ResultsNumbersAllThirdParty.txt
304
305     counter5=0
306     counter6=0
307     counter7=0
308     counter8=0
309     counter9=0
310     counter10=0
311     counter11=0
312     counter12=0
313     counter13=0
314
315     cd ..
316
317 done
318
319 sort ResultsNameUnecessary.txt -o ResultsNameUnecessary.txt
320 sed '/^$/d' ResultsNameUnecessary.txt > ResultsNameUnecessaryDef.txt
321 rm ResultsNameUnecessary.txt
322
323 sort ResultsNameMissing.txt -o ResultsNameMissing.txt
324 sed '/^$/d' ResultsNameMissing.txt > ResultsNameMissingDef.txt
325 rm ResultsNameMissing.txt
326
327 sort ResultsNameOverLowDev.txt -o ResultsNameOverLowDev.txt
328 sed '/^$/d' ResultsNameOverLowDev.txt > ResultsNameOverLowDevDef.txt
329 rm ResultsNameOverLowDev.txt
330
331 sort ResultsNameOverUpDev.txt -o ResultsNameOverUpDev.txt
332 sed '/^$/d' ResultsNameOverUpDev.txt > ResultsNameOverUpDevDef.txt
333 rm ResultsNameOverUpDev.txt
334
335 sort ResultsNameOverLowAd.txt -o ResultsNameOverLowAd.txt
336 sed '/^$/d' ResultsNameOverLowAd.txt > ResultsNameOverLowAdDef.txt
337 rm ResultsNameOverLowAd.txt
338
339 sort ResultsNameOverUpAd.txt -o ResultsNameOverUpAd.txt
340 sed '/^$/d' ResultsNameOverUpAd.txt > ResultsNameOverUpAdDef.txt
341 rm ResultsNameOverUpAd.txt
342
343 sort ResultsNameOverLowSoc.txt -o ResultsNameOverLowSoc.txt
344 sed '/^$/d' ResultsNameOverLowSoc.txt > ResultsNameOverLowSocDef.txt
345 rm ResultsNameOverLowSoc.txt
346
347 sort ResultsNameOverUpSoc.txt -o ResultsNameOverUpSoc.txt
348 sed '/^$/d' ResultsNameOverUpSoc.txt > ResultsNameOverUpSocDef.txt
349 rm ResultsNameOverUpSoc.txt
350
351 sort ResultsNameAllThirdParty.txt -o ResultsNameAllThirdParty.txt
```

```
sed '/^$/d' ResultsNameAllThirdParty.txt > ResultsNameAllThirdPartyDef.txt
rm ResultsNameAllThirdParty.txt

# Counts the number of Over-privileges as well as the number of Unecessary permissions.
awk '{ sum += $1 } END { print sum }' ResultsNumbersUnecessary.txt >> totalUnecessary.txt
awk '{ sum += $1 } END { print sum }' ResultsNumbersMissing.txt >> totalMissing.txt
awk '{ sum += $1 } END { print sum }' ResultsNumbersOverLowDev.txt >> totalOverLowDev.txt
awk '{ sum += $1 } END { print sum }' ResultsNumbersOverUpDev.txt >> totalOverUpDev.txt
awk '{ sum += $1 } END { print sum }' ResultsNumbersOverLowAd.txt >> totalOverLowAd.txt
awk '{ sum += $1 } END { print sum }' ResultsNumbersOverUpAd.txt >> totalOverUpAd.txt
awk '{ sum += $1 } END { print sum }' ResultsNumbersOverLowSoc.txt >> totalOverLowSoc.txt
awk '{ sum += $1 } END { print sum }' ResultsNumbersOverUpSoc.txt >> totalOverUpSoc.txt
awk '{ sum += $1 } END { print sum }' ResultsNumbersAllThirdParty.txt >> totalAllThirdParty.
    txt
```

Further, we developed a bash script to inspect the increase of granted permissions due to the presence of third-party libraries.

```
#!/bin/bash

#
# Roger Ribas
# roger.ribas90@gmail.com
# 6th of May 2014
#
# This script analyses the output results of the Compare script.
# It counts the number of Over-privileges as well as the number
# of Unecessary permissions.
#

IFS=$'\n'

counter=0
counter2=0
counter3=0
counter4=0
tmp=0

echo "" > ResultsNameUnecessary.txt
echo "" > ResultsNumbersUnecessary.txt
echo "" > ResultsNameMissing.txt
echo "" > ResultsNameUnecessary.txt
for dir in `find -maxdepth 1 -type d`
do
   if [ "$dir" = "." ] || [ "$dir" = ./manifest ];then
      continue
   else
      counter=`expr $counter + 1`
   fi
done

echo "I: $counter applications are going to be analyzed..."

for dir in `find -maxdepth 1 -type d`
do
   if [ "$dir" = "." ] || [ "$dir" = ./manifest ];then
      continue
   fi

   cd $dir

   echo "I: $dir"
```

```
45
46    echo "I: joining files..."
47
48    # Joins the Development libraries' permissions with the
49    # app permissions.
50    cat GeneratedManifestApp.txt >> tmp.txt
51    cat GeneratedManifestdevelopmenttools.txt >> tmp.txt
52    awk '!x[$0]++' tmp.txt > AppDev.txt
53    rm tmp.txt
54    sort AppDev.txt -o AppDev.txt
55    sed '/^$/d' AppDev.txt > AppDevDef.txt
56    rm AppDev.txt
57
58    # Joins the AdLibraries' permissions with the app
59    # permissions.
60    cat GeneratedManifestadlibraries.txt >> tmp.txt
61    cat GeneratedManifestApp.txt >> tmp.txt
62    awk '!x[$0]++' tmp.txt > AppAd.txt
63    rm tmp.txt
64    sort AppAd.txt -o AppAd.txt
65    sed '/^$/d' AppAd.txt > AppAdDef.txt
66    rm AppAd.txt
67
68    # Joins the Social SDKs' permissions with the app
69    # permissions.
70    cat GeneratedManifestsocialsdks.txt >> tmp.txt
71    cat GeneratedManifestApp.txt >> tmp.txt
72    awk '!x[$0]++' tmp.txt > AppSoc.txt
73    rm tmp.txt
74    sort AppSoc.txt -o AppSoc.txt
75    sed '/^$/d' AppSoc.txt > AppSocDef.txt
76    rm AppSoc.txt
77
78    # Joins the Obfuscated' permissions with the app
79    # permissions.
80    cat GeneratedManifestothers.txt >> tmp.txt
81    cat GeneratedManifestApp.txt >> tmp.txt
82    awk '!x[$0]++' tmp.txt > AppOb.txt
83    rm tmp.txt
84    sort AppOb.txt -o AppOb.txt
85    sed '/^$/d' AppOb.txt > AppObDef.txt
86    rm AppOb.txt
87
88    # Joins all the files together
89    cat GeneratedManifestdevelopmenttools.txt >> tmp.txt
90    cat GeneratedManifestadlibraries.txt >> tmp.txt
91    cat GeneratedManifestsocialsdks.txt >> tmp.txt
92    cat GeneratedManifestothers.txt >> tmp.txt
93    cat GeneratedManifestApp.txt >> tmp.txt
94    awk '!x[$0]++' tmp.txt > TotalPermissions.txt
95    rm tmp.txt
96    sort TotalPermissions.txt -o TotalPermissions.txt
97    sed '/^$/d' TotalPermissions.txt > TotalPermissionsDef.txt
98    rm TotalPermissions.txt
99
100   echo "" > ResultsUnecessary.txt
101   echo "" > ResultsMissing.txt
102
103   echo "I: Analysing..."
104
105   # Counts the number of permissions granted to the application
106   for line in `cat AndroidManifest.txt | uniq`
107   do
```

```
108       counter1=`expr $counter1 + 1`
109    done
110
111    echo "$counter1" >> ../TotalNumberOfPermissions.txt
112
113    # If a permission required by the application itself is not present
114    # in the manifest we have a missing permission.
115    for line in `cat GeneratedManifestApp.txt`
116    do
117       counter2=`cat AndroidManifest.txt | grep -i -c "$line"`
118
119       if [ $counter2 -eq 0 ];then
120          echo "Missing permission: $line" >> ResultsMissing.txt
121       fi
122    done
123
124    # If a line from the manifest is not present in the TotalPermissions
125    # it is an unecessary permission.
126    for line in `cat AndroidManifest.txt`
127    do
128       counter3=`cat TotalPermissionsDef.txt | grep -i -c "$line"`
129
130       if [ $counter3 -eq 0 ];then
131          echo "Unecessary permission: $line" >> ResultsUnecessary.txt
132       fi
133    done
134
135    for permission in `cat /volume1/scratch/r0442265/Analysis/apktool/permissions.txt`
136    do
137       #echo $permission
138
139       counter4=`cat GeneratedManifestApp.txt | grep -i -c "$permission"`
140       counter5=`cat GeneratedManifestdevelopmenttools.txt | grep -i -c "$permission"`
141       counter6=`cat GeneratedManifestadlibraries.txt | grep -i -c "$permission"`
142       counter7=`cat GeneratedManifestsocialsdks.txt | grep -i -c "$permission"`
143       counter8=`cat GeneratedManifestothers.txt | grep -i -c "$permission"`
144       counter9=`cat AppDevDef.txt | grep -i -c "$permission"`
145       counter10=`cat AppAdDef.txt | grep -i -c "$permission"`
146       counter11=`cat AppSocDef.txt | grep -i -c "$permission"`
147       counter12=`cat AppObDef.txt | grep -i -c "$permission"`
148       counter13=`cat AndroidManifest.txt | grep -i -c "$permission"`
149       echo $counter4 >> GeneratedManifestAppPermissions.txt
150       echo $counter5 >> GeneratedManifestdevelopmenttoolsPermissions1.txt
151       echo $counter6 >> GeneratedManifestadlibrariesPermissions1.txt
152       echo $counter7 >> GeneratedManifestsocialsdksPermissions1.txt
153       echo $counter8 >> GeneratedManifestothersPermissions1.txt
154       echo $counter9 >> AppDevPermissions1.txt
155       echo $counter10 >> AppAdPermissions1.txt
156       echo $counter11 >> AppSocPermissions1.txt
157       echo $counter12 >> AppObPermissions1.txt
158       echo $counter12 >> AndroidManifestPermissions.txt
159    done
160
161    paste GeneratedManifestdevelopmenttoolsPermissions1.txt AndroidManifestPermissions.txt >>
           GeneratedManifestdevelopmenttoolsPermissionsc.txt
162    awk '{ if ($2 != 0) print $1; else print 0 }'
           GeneratedManifestdevelopmenttoolsPermissionsc.txt >>
           GeneratedManifestdevelopmenttoolsPermissions.txt
163    rm GeneratedManifestdevelopmenttoolsPermissionsc.txt
164    rm GeneratedManifestdevelopmenttoolsPermissions1.txt
165
166    paste GeneratedManifestadlibrariesPermissions1.txt AndroidManifestPermissions.txt >>
           GeneratedManifestadlibrariesPermissionsc.txt
```

```
167   awk '{ if ($2 != 0) print $1; else print 0 }' GeneratedManifestadlibrariesPermissionsc.txt
          >> GeneratedManifestadlibrariesPermissions.txt
168   rm GeneratedManifestadlibrariesPermissionsc.txt
169   rm GeneratedManifestadlibrariesPermissions1.txt
170
171   paste GeneratedManifestsocialsdksPermissions1.txt AndroidManifestPermissions.txt >>
          GeneratedManifestsocialsdksPermissionsc.txt
172   awk '{ if ($2 != 0) print $1; else print 0 }' GeneratedManifestsocialsdksPermissionsc.txt
          >> GeneratedManifestsocialsdksPermissions.txt
173   rm GeneratedManifestsocialsdksPermissionsc.txt
174   rm GeneratedManifestsocialsdksPermissions1.txt
175
176   paste GeneratedManifestothersPermissions1.txt AndroidManifestPermissions.txt >>
          GeneratedManifestothersPermissionsc.txt
177   awk '{ if ($2 != 0) print $1; else print 0 }' GeneratedManifestothersPermissionsc.txt >>
          GeneratedManifestothersPermissions.txt
178   rm GeneratedManifestothersPermissionsc.txt
179   rm GeneratedManifestothersPermissions1.txt
180
181   paste AppDevPermissions1.txt AndroidManifestPermissions.txt >> AppDevPermissionsc.txt
182   awk '{ if ($2 != 0) print $1; else print 0 }' AppDevPermissionsc.txt >> AppDevPermissions.
          txt
183   rm AppDevPermissionsc.txt
184   rm AppDevPermissions1.txt
185
186   paste AppAdPermissions1.txt AndroidManifestPermissions.txt >> AppAdPermissionsc.txt
187   awk '{ if ($2 != 0) print $1; else print 0 }' AppAdPermissionsc.txt >> AppAdPermissions.
          txt
188   rm AppAdPermissionsc.txt
189   rm AppAdPermissions1.txt
190
191   paste AppSocPermissions1.txt AndroidManifestPermissions.txt >> AppSocPermissionsc.txt
192   awk '{ if ($2 != 0) print $1; else print 0 }' AppSocPermissionsc.txt >> AppSocPermissions.
          txt
193   rm AppSocPermissionsc.txt
194   rm AppSocPermissions1.txt
195
196   paste AppObPermissions1.txt AndroidManifestPermissions.txt >> AppObPermissionsc.txt
197   awk '{ if ($2 != 0) print $1; else print 0 }' AppObPermissionsc.txt >> AppObPermissions.
          txt
198   rm AppObPermissionsc.txt
199   rm AppObPermissions1.txt
200
201   sort ResultsUnecessary.txt -o ResultsUnecessary.txt
202   sort ResultsMissing.txt -o ResultsMissing.txt
203
204   for ln in `cat ResultsUnecessary.txt`
205   do
206      echo $ln >> ../ResultsNameUnecessary.txt
207
208      tmp=`echo $ln | grep -c "Unecessary"`
209      counter3=`expr $counter3 + $tmp`
210      tmp=0
211   done
212
213   for ln in `cat ResultsMissing.txt`
214   do
215      echo $ln >> ../ResultsNameMissing.txt
216
217      tmp=`echo $ln | grep -c "Missing"`
218      counter4=`expr $counter4 + $tmp`
219      tmp=0
220   done
```

```
221
222   #Total number of unecessary permissions.
223   echo "$counter3" >> ../ResultsNumbersUnecessary.txt
224   echo "$counter4" >> ../ResultsNumbersMissing.txt
225
226   cd smali
227
228   #Calculates the number of Development tools libraries
229   for library in `cat /volume1/scratch/r0442265/Analysis/apktool/adlibraries.txt`
230   do
231      for dir in `find * -maxdepth 1 -type d`
232      do
233         tmp=`echo $dir | grep -c "$library"`
234         counter13=`expr $counter13 + $tmp`
235         tmp=0
236      done
237      echo "$counter13" >> ../NumberAd.txt
238      counter13=0
239   done
240
241   #Calculates the number of AdLibraries libraries
242   for library in `cat /volume1/scratch/r0442265/Analysis/apktool/developmenttools.txt`
243   do
244      for dir in `find * -maxdepth 1 -type d`
245      do
246         tmp=`echo $dir | grep -c "$library"`
247         counter14=`expr $counter14 + $tmp`
248         tmp=0
249      done
250      echo "$counter14" >> ../NumberDev.txt
251      counter14=0
252   done
253
254   #Calculates the number of Social SDKs libraries
255   for library in `cat /volume1/scratch/r0442265/Analysis/apktool/socialsdks.txt`
256   do
257      for dir in `find * -maxdepth 1 -type d`
258      do
259         tmp=`echo $dir | grep -c "$library"`
260         counter15=`expr $counter15 + $tmp`
261         tmp=0
262      done
263      echo "$counter15" >> ../NumberSoc.txt
264      counter15=0
265   done
266
267   #calculates the number of directories
268   for dir in `find * -maxdepth 1 -type d`
269   do
270      counter16=`expr $counter16 + 1`
271   done
272   echo "$counter16" >> ../NumberDir.txt
273   counter16=0
274
275   cd ..
276   cd ..
277
278   counter1=0
279   counter2=0
280   counter3=0
281   counter4=0
282   counter5=0
283   counter6=0
```

```
284     counter7=0
285     counter8=0
286     counter9=0
287     counter10=0
288     counter11=0
289     counter12=0
290 done
291
292 for file in `find * -type f -name "NumberAd.txt"`
293 do
294     cat $file >> ListNumberAd.txt
295 done
296 for file in `find * -type f -name "NumberDev.txt"`
297 do
298     cat $file >> ListNumberDev.txt
299 done
300 for file in `find * -type f -name "NumberSoc.txt"`
301 do
302     cat $file >> ListNumberSoc.txt
303 done
304 for file in `find * -type f -name "NumberDir.txt"`
305 do
306     cat $file >> ListNumberDir.txt
307 done
308
309 #Counts the number of third-party libraries found of each category
310 awk '{ sum += $1 } END { print sum }' ListNumberAd.txt >> totalNumberAd.txt
311 awk '{ sum += $1 } END { print sum }' ListNumberDev.txt >> totalNumberDev.txt
312 awk '{ sum += $1 } END { print sum }' ListNumberSoc.txt >> totalNumberSoc.txt
313 awk '{ sum += $1 } END { print sum }' ListNumberDir.txt >> totalNumberDir.txt
314
315 rm  ListNumberAd.txt
316 rm  ListNumberDev.txt
317 rm  ListNumberSoc.txt
318 rm  ListNumberDir.txt
319
320 totalad=`cat totalNumberAd.txt`
321 totaldev=`cat totalNumberDev.txt`
322 totalsoc=`cat totalNumberSoc.txt`
323 totaldir=`cat totalNumberDir.txt`
324 totalobfuscated=$((totaldir-totalad-totaldev-totalsoc))
325
326 echo $totalobfuscated
327
328 sort ResultsNameUnecessary.txt -o ResultsNameUnecessary.txt
329 sed '/^$/d' ResultsNameUnecessary.txt > ResultsNameUnecessaryDef.txt
330 rm ResultsNameUnecessary.txt
331
332 sort ResultsNameMissing.txt -o ResultsNameMissing.txt
333 sed '/^$/d' ResultsNameMissing.txt > ResultsNameMissingDef.txt
334 rm ResultsNameMissing.txt
335
336 # Counts the number of Over-privileges as well as the number of Unecessary permissions.
337 awk '{ sum += $1 } END { print sum }' ResultsNumbersUnecessary.txt >> totalUnecessary.txt
338 awk '{ sum += $1 } END { print sum }' ResultsNumbersMissing.txt >> totalMissing.txt
339
340 echo "" >> GeneratedManifestAppPermissionsTotal.txt
341 for file in `find * -type f -name "GeneratedManifestAppPermissions.txt"`
342 do
343     if [ $file = "GeneratedManifestAppPermissions.txt" ];then
344         continue
345     fi
346     paste GeneratedManifestAppPermissionsTotal.txt $file > tmp
```

```
347     mv tmp GeneratedManifestAppPermissionsTotal.txt
348 done
349 awk '{ for(i=1; i<=NF;i++) j+=$i; print j; j=0 }' GeneratedManifestAppPermissionsTotal.txt >>
        GeneratedManifestAppPermissionsTotalDef.txt
350 rm GeneratedManifestAppPermissionsTotal.txt
351 awk '{ if ($counter != 0) print $1/'$counter'; else print 0 }'
        GeneratedManifestAppPermissionsTotalDef.txt >> AppProb.txt
352 rm GeneratedManifestAppPermissionsTotalDef.txt
353
354 echo "" >> GeneratedManifestdevelopmenttoolsPermissions.txt
355 for file in `find * -type f -name "GeneratedManifestdevelopmenttoolsPermissions.txt"`
356 do
357     if [ $file = "GeneratedManifestdevelopmenttoolsPermissions.txt" ];then
358         continue
359     fi
360     paste GeneratedManifestdevelopmenttoolsPermissions.txt $file > tmp
361     mv tmp GeneratedManifestdevelopmenttoolsPermissions.txt
362 done
363 awk '{ for(i=1; i<=NF;i++) j+=$i; print j; j=0 }'
        GeneratedManifestdevelopmenttoolsPermissions.txt >>
        GeneratedManifestdevelopmenttoolsPermissionsDef.txt
364 rm GeneratedManifestdevelopmenttoolsPermissions.txt
365 awk '{ if ($totaldev != 0) print $1/'$totaldev'; else print 0 }'
        GeneratedManifestdevelopmenttoolsPermissionsDef.txt >> DevProb.txt
366 rm GeneratedManifestdevelopmenttoolsPermissionsDef.txt
367
368 echo "" >> GeneratedManifestadlibrariesPermissions.txt
369 for file in `find * -type f -name "GeneratedManifestadlibrariesPermissions.txt"`
370 do
371     if [ $file = "GeneratedManifestadlibrariesPermissions.txt" ];then
372         continue
373     fi
374     paste GeneratedManifestadlibrariesPermissions.txt $file > tmp
375     mv tmp GeneratedManifestadlibrariesPermissions.txt
376 done
377 awk '{ for(i=1; i<=NF;i++) j+=$i; print j; j=0 }' GeneratedManifestadlibrariesPermissions.txt
        >> GeneratedManifestadlibrariesPermissionsDef.txt
378 rm GeneratedManifestadlibrariesPermissions.txt
379 awk '{ if ($totalad != 0) print $1/'$totalad'; else print 0 }'
        GeneratedManifestadlibrariesPermissionsDef.txt >> AdProb.txt
380 rm GeneratedManifestadlibrariesPermissionsDef.txt
381
382 echo "" >> GeneratedManifestsocialsdksPermissions.txt
383 for file in `find * -type f -name "GeneratedManifestsocialsdksPermissions.txt"`
384 do
385     if [ $file = "GeneratedManifestsocialsdksPermissions.txt" ];then
386         continue
387     fi
388     paste GeneratedManifestsocialsdksPermissions.txt $file > tmp
389     mv tmp GeneratedManifestsocialsdksPermissions.txt
390 done
391 awk '{ for(i=1; i<=NF;i++) j+=$i; print j; j=0 }' GeneratedManifestsocialsdksPermissions.txt
        >> GeneratedManifestsocialsdksPermissionsDef.txt
392 rm GeneratedManifestsocialsdksPermissions.txt
393 awk '{ if ($totalsoc != 0) print $1/'$totalsoc'; else print 0 }'
        GeneratedManifestsocialsdksPermissionsDef.txt >> SocProb.txt
394 rm GeneratedManifestsocialsdksPermissionsDef.txt
395
396 echo "" >> GeneratedManifestothersPermissions.txt
397 for file in `find * -type f -name "GeneratedManifestothersPermissions.txt"`
398 do
399     if [ $file = "GeneratedManifestothersPermissions.txt" ];then
400         continue
```

```
401     fi
402     paste GeneratedManifestothersPermissions.txt $file > tmp
403     mv tmp GeneratedManifestothersPermissions.txt
404   done
405   awk '{ for(i=1; i<=NF;i++) j+=$i; print j; j=0 }' GeneratedManifestothersPermissions.txt >>
        GeneratedManifestothersPermissionsDef.txt
406   rm GeneratedManifestothersPermissions.txt
407   awk '{ if ($totalobfuscated != 0) print $1/'$totalobfuscated'; else print 0 }'
        GeneratedManifestothersPermissionsDef.txt >> ObfProb.txt
408   rm GeneratedManifestothersPermissionsDef.txt
409
410   echo "" >> AppDevPermissions.txt
411   for file in `find * -type f -name "AppDevPermissions.txt"`
412   do
413     if [ $file = "AppDevPermissions.txt" ];then
414       continue
415     fi
416     paste AppDevPermissions.txt $file > tmp
417     mv tmp AppDevPermissions.txt
418   done
419   awk '{ for(i=1; i<=NF;i++) j+=$i; print j; j=0 }' AppDevPermissions.txt >>
        AppDevPermissionsDef.txt
420   rm AppDevPermissions.txt
421   appdev=`expr $counter + $totaldev`
422   awk '{ if ($appdev != 0) print $1/'$appdev'; else print 0 }' AppDevPermissionsDef.txt >>
        AppDevProb.txt
423   rm AppDevPermissionsDef.txt
424
425   echo "" >> AppAdPermissions.txt
426   for file in `find * -type f -name "AppAdPermissions.txt"`
427   do
428     if [ $file = "AppAdPermissions.txt" ];then
429       continue
430     fi
431     paste AppAdPermissions.txt $file > tmp
432     mv tmp AppAdPermissions.txt
433   done
434   awk '{ for(i=1; i<=NF;i++) j+=$i; print j; j=0 }' AppAdPermissions.txt >> AppAdPermissionsDef.
        txt
435   rm AppAdPermissions.txt
436   appad=`expr $counter + $totalad`
437   awk '{ if ($appad != 0) print $1/'$appad'; else print 0 }' AppAdPermissionsDef.txt >>
        AppAdProb.txt
438   rm AppAdPermissionsDef.txt
439
440   echo "" >> AppSocPermissions.txt
441   for file in `find * -type f -name "AppSocPermissions.txt"`
442   do
443     if [ $file = "AppSocPermissions.txt" ];then
444       continue
445     fi
446     paste AppSocPermissions.txt $file > tmp
447     mv tmp AppSocPermissions.txt
448   done
449   awk '{ for(i=1; i<=NF;i++) j+=$i; print j; j=0 }' AppSocPermissions.txt >>
        AppSocPermissionsDef.txt
450   rm AppSocPermissions.txt
451   appsoc=`expr $counter + $totalsoc`
452   awk '{ if ($appsoc != 0) print $1/'$appsoc'; else print 0 }' AppSocPermissionsDef.txt >>
        AppSocProb.txt
453   rm AppSocPermissionsDef.txt
454
455   echo "" >> AppObPermissions.txt
```

```
456 for file in ‘find * -type f -name "AppObPermissions.txt"‘
457 do
458    if [ $file = "AppObPermissions.txt" ];then
459       continue
460    fi
461    paste AppObPermissions.txt $file > tmp
462    mv tmp AppObPermissions.txt
463 done
464 awk ’{ for(i=1; i<=NF;i++) j+=$i; print j; j=0 }’ AppObPermissions.txt >> AppObPermissionsDef.
      txt
465 rm AppObPermissions.txt
466 appobf=‘expr $counter + $totalobfuscated‘
467 awk ’{ if ($appobf != 0) print $1/’$appobf’; else print 0 }’ AppObPermissionsDef.txt >>
      AppObfProb.txt
468 rm AppObPermissionsDef.txt
```

Appendix B

Source Code of LPDroid

B.1 User level source code

B.1.1 MyLocation application

Next, the source code of the Show User Location application can be found. Notice how all the necessary steps are included in the `onCreate()` and `onResult()` methods.

```
1  package org.example.locationtest;
2
3  import java.util.List;
4
5  import android.app.Activity;
6  import android.content.Intent;
7  import android.location.Criteria;
8  import android.location.Location;
9  import android.location.LocationListener;
10 import android.location.LocationManager;
11 import android.location.LocationProvider;
12 import android.os.Bundle;
13 import android.view.Gravity;
14 import android.widget.TextView;
15 import android.widget.Toast;
16
17 public class LocationTest extends Activity{
18
19    private LocationManager mgr;
20    private LocationListener locationListener;
21    private Location location;
22    private Criteria criteria;
23    private LocationProvider info;
24    private TextView output;
25    private String best;
26
27    private static final String[] A = {"invalid" , "n/a", "fine", "coarse"};
28    private static final String[] P = {"invalid" , "n/a", "low", "medium", "high"};
29    private static final String[] S = {"out of service" , "temporally unavailabe"};
30
31    @Override
32    protected void onCreate(Bundle savedInstanceState) {
33       super.onCreate(savedInstanceState);
34       setContentView(R.layout.main);
35 /*LOCATION_SERVICE: use with getSystemService(String) to retrieve a LocationManager for
       controlling location updates. Return the handle to a system-level service by name. The
```

```
     class of the returned object varies by the requested name. */
36       mgr = (LocationManager) getSystemService(LOCATION_SERVICE);
37       output = (TextView) findViewById(R.id.output);
38
39       locationListener = new LocationListener(){
40          @Override
41          public void onLocationChanged(Location location) {
42             dumpLocation(location);
43          }
44
45          @Override
46          public void onProviderDisabled(String provider) {
47             log("\nProvider disabled: " + provider);
48          }
49
50          @Override
51          public void onProviderEnabled(String provider) {
52             log("\nProvider enabled: " + provider);
53          }
54
55          @Override
56  public void onStatusChanged(String provider, int status, Bundle extras) {
57  log("\nProvider status changed: " + provider + ", status = " + S[status] + ", extras = " +
     extras);
58          }
59       };
60
61  //Sets the criteria for getting the best provider and figures out which one is it
62       criteria = new Criteria();
63       criteria.setAccuracy(Criteria.ACCURACY_FINE);
64       criteria.setAltitudeRequired(true);
65
66       if (!mgr.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
67  Toast toast1 = Toast.makeText(this, "GPS not enabled", Toast.LENGTH_LONG);
68          toast1.setGravity(Gravity.CENTER, 0, 0);
69          toast1.show();
70  Toast toast2 = Toast.makeText(this, "Please activate GPS and press back", Toast.LENGTH_LONG);
71          toast2.setGravity(Gravity.CENTER, 0, 0);
72          toast2.show();
73          /*Opens the location settings*/
74  Intent gpsOptionsIntent = new Intent(android.provider.Settings.
     ACTION_LOCATION_SOURCE_SETTINGS);
75          startActivity(gpsOptionsIntent);
76       }
77
78  /*String getBestProvider(Criteria criteria, boolean enabledOnly) returns the name of the
     provider that best meets the given criteria. We could also use List<String> getProviders
     (Criteria criteria, boolean enabledOnly) which returns a list with the names of the
     providers that fulfill the requirements */
79       best = mgr.getBestProvider(criteria, true);
80
81       log("Location providers: ");
82       dumpProviders();
83
84  log("\nBest provider according to the selected criteria is: " + best);
85
86       log("\nLocations (starting with last known): ");
87       location = mgr.getLastKnownLocation(best);
88       dumpLocation(location);
89    }
90
91    @Override
92    protected void onResume() {
```

```
93          super.onResume();
94      //Initializes the updates
95      //requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener
            listener)
96      //listener: must be someone who contains the method onLocationChange() so we pass this.
97          mgr.requestLocationUpdates(best, 15000, 1, locationListener);
98       }
99
100      @Override
101      protected void onPause() {
102         super.onPause();
103     /*Stops the updates in order to save battery removeUpdates(LocationListener listener).
            Removes all location updates for the specified LocationListener*/
104         mgr.removeUpdates(locationListener);
105      }
106
107      //Writes the string in the TextView widget
108      private void log(String string){
109         output.append(string + "\n");
110      }
111
112      //Writes the information of all the Location Providers
113      private void dumpProviders(){
114         List<String> providers = mgr.getAllProviders();
115         for (String provider : providers){
116            dumpProvider(provider);
117         }
118      }
119
120      //Writes the information of only one provider
121      private void dumpProvider(String provider){
122         info = mgr.getProvider(provider);
123         StringBuilder builder = new StringBuilder();
124
125         builder.append("LocationProvider[")
126            .append("name=")
127            .append(info.getName())
128            .append(", enabled=")
129            .append(mgr.isProviderEnabled(provider))
130            .append(", getAccuracy=")
131            .append(A[info.getAccuracy() + 1])
132            .append(", getPowerRequirements=")
133            .append(P[info.getPowerRequirement() + 1])
134            .append(", hasMonetaryCost=")
135            .append(info.hasMonetaryCost())
136            .append(", requiresCell=")
137            .append(info.requiresCell())
138            .append(", requiresNetwork=")
139            .append(info.requiresNetwork())
140            .append(", requiresSatellite=")
141            .append(info.requiresSatellite())
142            .append(", supportsAltitude=")
143            .append(info.supportsAltitude())
144            .append(", supportsBearing=")
145            .append(info.supportsBearing())
146            .append(", supportsSpeed=")
147            .append(info.supportsSpeed())
148            .append("]");
149         log(builder.toString());
150      }
151
152      //Prints the current location
153      private void dumpLocation(Location location){
```

```
154       if(location == null)log("\nLocation[unknown]");
155       else log("\n" + location.toString());
156    }
157 }
```

B.1.2 LocationApps application

In this section, the source code of the List Permissions application can be found.

```
1  public class RequireLocationApps extends Activity {
2
3  /*ArrayList is an implementation of List, backed by an array. All optional operations
       including, removing, and replacing elements are supported. This class is a good choice
       as your default List implementation. All elements are permitted, including null.*/
4     private ArrayList<String> lista;
5  /*A List is a collection which maintains an ordering for its elements. Every element in the
       List has an index. Each element can thus be accessed by its index, with the first index
       being zero. Normally, Lists allow duplicate elements, as compared to Sets, where
       elements have to be unique.*/
6     private List<PackageInfo> applist;
7     private ListView listView;
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11       super.onCreate(savedInstanceState);
12       setContentView(R.layout.main);
13
14       listView = (ListView) findViewById(R.id.listView1);
15 // Instanciating an array
16          lista = new ArrayList<String>();
17             lista = getInstalledApps(this);
18
19 /*An adapter is used for managing the items in the list. Every line in the list view consists
        of a layout which can be as complex as you want. The adapter would inflate the layout
       for each row in its getView() method and assign the data to the individual views in the
       row.
20 It takes the context of the activity as a first parameter, the type of list view as a second
       parameter and your array as a third parameter.*/
21
22          /** The simple_list_item_1 is a built-in layout:
23 <TextView xmlns:android="http://schemas.android.com/apk/res/android"
24       android:id="@android:id/text1"
25          style="?android:attr/listItemFirstLineStyle"
26          android:paddingTop="2dip"
27          android:paddingBottom="3dip"
28          android:layout_width="fill_parent"
29          android:layout_height="wrap_content" />**/
30
31
32          ArrayAdapter<String> arrayAdapter =
33          new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1, lista);
34          listView.setAdapter(arrayAdapter);
35    }
36
37    private ArrayList<String> getInstalledApps(Context context) {
38          ArrayList<String> results = new ArrayList<String>();
39 /*Class for retrieving various kinds of information related to the application packages that
       are currently installed on the device. You can find this class through getPackageManager
       ()*/
40          PackageManager packageManager = context.getPackageManager();
41 //abstract List<PackageInfo> getInstalledPackages(int flags)
```

```
42        applist = packageManager.getInstalledPackages(0);
43
44  //An iterator over a sequence of objects, such as a collection.
45        Iterator<PackageInfo> iterator = applist.iterator();
46  /*abstract boolean hasNext()
47  Returns true if there is at least one more element, false otherwise.*/
48        while (iterator.hasNext()) {
49  /*Overall information about the contents of a package. This corresponds to all of the
        information collected from AndroidManifest.xml.*/
50              PackageInfo pk = (PackageInfo) iterator.next();
51  if ((pk.applicationInfo.flags & ApplicationInfo.FLAG_SYSTEM) != 0) {
52  /*Information you can retrieve about a particular application. This corresponds to
        information collected from the AndroidManifest.xml's <application> tag.
53  CharSequence loadLabel(PackageManager pm)
54  Retrieve the current textual label associated with this item.*/
55  Log.v("system app = ", ""+pk.applicationInfo.loadLabel(packageManager));
56                continue;
57             }
58  /*public abstract int checkPermission (String permName, String pkgName)
59  Check whether a particular package has been granted a particular permission*/
60  if (PackageManager.PERMISSION_GRANTED == packageManager.checkPermission(Manifest.permission.
        ACCESS_FINE_LOCATION,pk.packageName) || PackageManager.PERMISSION_GRANTED ==
        packageManager.checkPermission(Manifest.permission.ACCESS_COARSE_LOCATION,pk.packageName)
        )
61  results.add("" + pk.applicationInfo.loadLabel(packageManager));
62        }
63
64        Log.v("app using location = ", results.toString());
65
66        return results;
67    }
68  }
```

Finally, the source code of the LPDroid user application can be observed:

```
1   package com.example.locationprivacyproject;
2
3   import android.app.Activity;
4   import android.os.Bundle;
5   import android.widget.CompoundButton;
6   import android.widget.RadioButton;
7   import android.widget.RadioGroup;
8   import android.widget.RadioGroup.OnCheckedChangeListener;
9   import android.widget.Toast;
10  import android.widget.ToggleButton;
11
12  public class LocationPrivacy extends Activity {
13
14      private boolean isStarted = false;
15      private int selectedLevel = 2;
16
17      private ToggleButton start_button;
18      private RadioGroup radioLevelGroup;
19      private RadioButton radioLevelButton;
20
21      @Override
22      protected void onCreate(Bundle savedInstanceState) {
23          super.onCreate(savedInstanceState);
24          setContentView(R.layout.main);
25
26          addListenerOnButton();
27          addListenerOnRadioGroup();
28      }
```

```
29
30      private void addListenerOnButton(){
31          start_button = (ToggleButton) findViewById(R.id.toggleButton1);
32
33          start_button.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
34              @Override
35              public void onCheckedChanged(CompoundButton buttonView,boolean isChecked) {
36                  if(isChecked) isStarted = true;
37                  else isStarted = false;
38              }
39          });
40      }
41
42      private void addListenerOnRadioGroup() {
43          radioLevelGroup = (RadioGroup) findViewById(R.id.radioLevel);
44
45          radioLevelGroup.setOnCheckedChangeListener(new OnCheckedChangeListener()
46          {
47              public void onCheckedChanged(RadioGroup group, int checkedId) {
48
49                  // find the radio button by returned id
50                  radioLevelButton = (RadioButton) findViewById(checkedId);
51
52                  //Toast
53                  if(isStarted){
54                  mapValues();
55                      Toast.makeText(LocationPrivacy.this,radioLevelButton.getText(), Toast.
                            LENGTH_SHORT).show();
56                      }
57              }
58          });
59      }
60
61     private void mapValues() {
62        if(radioLevelButton.getText() == "low") selectedLevel = 1;
63        else if (radioLevelButton.getText() == "medium") selectedLevel = 2;
64        else selectedLevel = 3;
65     }
66  }
```

B.2 Framework level source code

In Figure B.1 we can see the modifications done in the LocationManager class to implement the lowest privacy level obfuscation strategy. The `switch` cases have been deleted to solely leave the lowest level case. The core of this LPPM resides in lines 18 and 19 where the casting is executed.

In Figure B.2 we can see the modifications done in the LocationManager class to implement the medium privacy level obfuscation strategy. As before, we have deleted the rest of the cases. It is important to notice the call in line 29 to the `setTaggedLocationCoordinates()` method. This method returns a Location object created with the given arguments, which are the fixed coordinates of the ESAT department.

In Figure B.3 we can see the modifications done in the LocationManager class to implement the medium privacy level obfuscation strategy. As before, we have omitted the rest of the cases. In lines 19 and 20 we make a call to `generateRandomCoordinates()`

```
1  public Location fakeLocation(Location current, Context context){
2     /**
3     * Depending on the selected preferences we'll implement a certain LLPM or another one.
4     */
5     Log.i(TAG, "Location Privacy: New location coordinates have been received, checking
          whether we should obfuscate them or not");
6
7     if (checkPrivacy(context) == 1){
8
9        privacyLevel = checkPrivacyLevel(context);
10
11       switch(privacyLevel){
12          case 1:
13             /**
14              * Rounds the current location.
15              */
16             Log.i(TAG, "Location Privacy: Tool activated with level 1, proceding to obfuscate
                   the new coordinates");
17
18             randomRoundFakeLatitude = (int) current.getLatitude();
19             randomRoundFakeLongitude = (int) current.getLongitude();
20
21             current = setTaggedLocationCoordinates(randomRoundFakeLatitude,
                   randomRoundFakeLongitude, current);
22             break;
23          default:
24             Log.i(TAG, "The tool is not activated");
25             break;
26       }
27    }
28    else{
29       Log.i(TAG, "The tool is not activated");
30       return current;
31    }
32    return current;
33 }
```

Figure B.1: Lowest level obfuscation strategy source code.

method. This method returns an integer number randomly generated within the interval (-90, 90). Once we have a longitude and a latitude coordinates we call the method `setTaggedLocationCoordinates()` to create the Location object with the newly generated coordinates.

Figure B.4 shows how the `getLasLocation()` and the `getLastKnownLocation()` methods make a call to the `fakeLocation()` method before returning the last known location.

Finally, Figure B.5 shows the code that implements the `checkPrivacy()` and `checkPrivacyLevel()` methods. The former, retrieves the value stored in the `PRIVACY_ACTIVATED` flag from the Settings provider. The latter, retrieves the value stored in the `PRIVACY_LEVEL` flag from the Settings provider.

```
1  //Part 1
2  private static final String fixedFakeLocation = "Fake Location";
3  private static final double fakeLatitude = 50.862346;
4  private static final double fakeLongitude = 4.686362;
5
6  //Part 2
7  public Location fakeLocation(Location current, Context context){
8     /**
9     * Depending on the selected preferences we'll implement a certain LLPM or another one.
10    */
11    Log.i(TAG, "Location Privacy: New location coordinates have been received, checking
          whether we should obfuscate them or not");
12
13    if (checkPrivacy(context) == 1){
14       /**
15       * Now, we should check the selected level of privacy.
16       * This level should be 2 (medium) by default.
17       * In this configuration, location coordinates will be replaced by randomly generated
18       * coordiantes.
19       */
20       privacyLevel = checkPrivacyLevel(context);
21
22       switch(privacyLevel){
23          case 2:
24             /**
25              * This LPPM replaces the Location coordinates by some fixed coordinates.
26              */
27             Log.i(TAG, "Location Privacy: Tool activated with level 1, proceding to obfuscate
                   the new coordinates");
28
29             current = setTaggedLocationCoordinates(fakeLatitude, fakeLongitude, current);
30
31             Log.i(TAG, "Location faked with fixed coordinates");
32             break;
33          default:
34             Log.i(TAG, "The tool is not activated");
35             break;
36       }
37    }
38    else{
39       Log.i(TAG, "The tool is not activated");
40       return current;
41    }
42    return current;
43     }
44
45 private Location setTaggedLocationCoordinates(double latitude, double longitude, Location
       current){
46
47    current.setLatitude(latitude);
48    current.setLongitude(longitude);
49
50    Date date = new Date();
51    current.setTime(date.getTime());
52
53    return current;
54 }
```

Figure B.2: Medium level obfuscation strategy source code.

```
//Part 1
public Location fakeLocation(Location current, Context context){
   /**
   * Depending on the selected preferences we'll implement a certain LLPM or another one.
   */
   Log.i(TAG, "Location Privacy: New location coordinates have been received, checking
        whether we should obfuscate them or not");

   if (checkPrivacy(context) == 1){

      privacyLevel = checkPrivacyLevel(context);

      switch(privacyLevel){
         case 3:
            /**
            * This LPPM replaces the Location coordinates by some randomly generated
                 coordinates
            */
            Log.i(TAG, "Location Privacy: Tool activated with level 3, proceding to obfuscate
                 the new coordinates");

            randomFakeLatitude = generateRandomCoordinates();
            randomFakeLongitude = generateRandomCoordinates();

            current = setTaggedLocationCoordinates(randomFakeLatitude, randomFakeLongitude,
                 current);

            Log.i(TAG, "Location faked with random coordinates");
            break;
         default:
            Log.i(TAG, "The tool is not activated");
            break;
      }
   }
   else{
      Log.i(TAG, "The tool is not activated");
      return current;
   }
   return current;
}

//Part 2
private int generateRandomCoordinates(){
   int min = -90;
   int max = 90;
   int tmp;

   Random r = new Random();
   tmp = r.nextInt(max - min + 1) + min;

   return tmp;
}
```

Figure B.3: Highest level obfuscation strategy source code.

```
1  public Location getLastLocation() {
2         String packageName = mContext.getPackageName();
3
4     /**
5      * If the privacy too has been activated by the user, we will not provide the last known
           location
6      * to the location based applications. Instead, we will return null and we will obfuscate
           any new
7           * data.
8      *
9      * However, if the Privacy Tool is not activated, we will retrieve the Last Known
10     * location to the location based applications. Moreover, we will be able to retrieve
11     * a non-obfuscated version of the last known coordinates, even if they were obfuscated
12     * in the past. This is a good point in terms of power cosumption.
13     */
14
15    try{
16       Log.i(TAG, "Location Privacy: Tool desactivated, returning las known coordinates");
17       Location loc = mService.getLastLocation(null, packageName);
18       loc = fakeLocation(loc, mContext);
19       return loc;
20          }catch (RemoteException e) {
21                Log.e(TAG, "RemoteException", e);
22             return null;
23    }
24 }
25
26 public Location getLastKnownLocation(String provider) {
27         checkProvider(provider);
28         String packageName = mContext.getPackageName();
29         LocationRequest request = LocationRequest.createFromDeprecatedProvider(provider, 0, 0,
               true);
30
31
32    try{
33       /**
34        * Vid supra.
35        */
36
37       Log.i(TAG, "Location Privacy: Returning last known coordinates");
38       Location loc = mService.getLastLocation(request, packageName);
39       loc = fakeLocation(loc, mContext);
40       return loc;
41
42         }catch (RemoteException e) {
43               Log.e(TAG, "RemoteException", e);
44            return null;
45    }
46 }
```

Figure B.4: Modifications done in the `getLastKnownLocation()` and `getLastLocation()` methods.

```
1  private int checkPrivacy(Context context){
2     try{
3         privacy = android.provider.Settings.System.getInt(context.getContentResolver(), android.
              provider.Settings.System.PRIVACY_ACTIVATED);
4     }catch (android.provider.Settings.SettingNotFoundException e) {
5         privacy = 0;
6         e.printStackTrace();
7          }
8     return privacy;
9  }
10
11 private int checkPrivacyLevel(Context context){
12    try {
13       privacyLevel = android.provider.Settings.System.getInt(context.getContentResolver(),
             android.provider.Settings.System.PRIVACY_LEVEL);
14        }catch (android.provider.Settings.SettingNotFoundException e) {
15       privacyLevel = 2;
16          e.printStackTrace();
17        }
18    return privacyLevel;
19 }
```

Figure B.5: `checkPrivacy()` and `checkPrivacyLevel()` methods.

Bibliography

[1] G.J. Spriensma. *Do free Apps Really Account For 89% Of All Downloads?* Distimo, Sept.2012. http://www.distimo.com/blog/2012_09_do-free-apps-really-account-for-89-of-all-downloads/

[2] Stevens, Ryan, et al. *"Investigating user privacy in Android ad libraries."* Workshop on Mobile Security Technologies (MoST). 2012.

[3] Andrews, Lori. "Facebook is using you". *The New York Times.* http://www.nytimes.com/2012/02/05/opinion/sunday/facebook-is-using-you.html?pagewanted=all&_r=0

[4] Dwoskin, Elisabeth. "Twitter's Data Business Proves Lucrative". *The New York Times.* http://online.wsj.com/news/articles/SB10001424052702304441404579118531954483974

[5] Adrienne Porter, et al. "Android permissions demystified." *Proceedings of the 18th ACM conference on Computer and communications security.* ACM, 2011.

[6] Book, Theodore, and Dan S. Wallach. "A case of collusion: A study of the interface between ad libraries and their apps." *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices.* ACM, 2013.

[7] Felt, Adrienne Porter, Kate Greenwood, and David Wagner. "The effectiveness of application permissions." *Proceedings of the 2nd USENIX conference on Web application development.* USENIX Association, 2011.

[8] Vidas, Timothy, Nicolas Christin, and Lorrie Cranor. "Curbing android permission creep." *Proceedings of the Web.* Vol. 2. 2011.

[9] Grace, Michael C., et al. "Unsafe exposure analysis of mobile in-app advertisements." *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks.* ACM, 2012.

[10] Enck, William, et al. "A Study of Android Application Security." *USENIX Security Symposium.* 2011.

[11] Book, Theodore, Adam Pridgen, and Dan S. Wallach. "Longitudinal analysis of Android ad library permissions." *arXiv preprint arXiv:1303.0857* (2013).

[12] Holavanalli, Shashank, et al. "Flow permissions for android." *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on.* IEEE, 2013.

[13] Leontiadis, Ilias, et al. "Don't kill my ads!: balancing privacy in an ad-supported mobile application market." *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications.* ACM, 2012.

[14] Shekhar, Shashi, Michael Dietz, and Dan S. Wallach. "Adsplit: Separating smartphone advertising from applications." *CoRR, abs/1202.4030* (2012).

[15] Au, Kathy Wain Yee, et al. "Pscout: analyzing the android permission specification."*Proceedings of the 2012 ACM conference on Computer and communications security.* ACM, 2012.

[16] Enck, William, et al. "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones."*OSDI.* Vol. 10. 2010.

[17] Hornyack, Peter, et al. "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications." *Proceedings of the 18th ACM conference on Computer and communications security.* ACM, 2011.2).

[18] Shokri, Reza, et al. "Quantifying location privacy." *Security and Privacy (SP), 2011 IEEE Symposium on.* IEEE, 2011.

[19] Shokri, Reza, et al. "Protecting location privacy: optimal strategy against localization attacks." *Proceedings of the 2012 ACM conference on Computer and communications security.* ACM, 2012.

[20] Herrmann, Michael, et al. "Optimal sporadic location privacy preserving systems in presence of bandwidth constraints." *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society.* ACM, 2013.

[21] Huang, Leping, et al. "Towards modeling wireless location privacy." *Privacy Enhancing Technologies.* Springer Berlin Heidelberg, 2006.

[22] Li, Mingyan, et al. "Swing & swap: user-centric approaches towards maximizing location privacy." *Proceedings of the 5th ACM workshop on Privacy in electronic society.* ACM, 2006.

[23] Beresford, Alastair R., and Frank Stajano. "Mix Zones: User Privacy in Location-aware Services." *PerCom Workshops.* 2004.

[24] Krumm, John. "Inference attacks on location tracks." *Pervasive Computing.* Springer Berlin Heidelberg, 2007. 127-143.

[25] Krumm, John. "Realistic driving trips for location privacy." *Pervasive Computing.* Springer Berlin Heidelberg, 2009. 25-41.

[26] Gruteser, Marco, and Dirk Grunwald. "Anonymous usage of location-based services through spatial and temporal cloaking." *Proceedings of the 1st international conference on Mobile systems, applications and services.* ACM, 2003.

[27] Chow, Chi-Yin, Mohamed F. Mokbel, and Xuan Liu. "A peer-to-peer spatial cloaking algorithm for anonymous location-based service." *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems.* ACM, 2006.

[28] Dong, Changyu, and Naranker Dulay. "Longitude: a privacy-preserving location sharing protocol for mobile applications." *Trust Management V.* Springer Berlin Heidelberg, 2011. 133-148.

[29] Zhong, Ge, Ian Goldberg, and Urs Hengartner. "Louis, lester and pierre: Three protocols for location privacy." *Privacy Enhancing Technologies.* Springer Berlin Heidelberg, 2007.

[30] Bilogrevic, Igor, et al. "Privacy in mobile computing for location-sharing-based services." *Privacy Enhancing Technologies.* Springer Berlin Heidelberg, 2011.

[31] Ghinita, Gabriel, et al. "Private queries in location based services: anonymizers are not necessary." *Proceedings of the 2008 ACM SIGMOD international conference on Management of data.* ACM, 2008.

[32] Olumofin, Femi, et al. "Achieving efficient query privacy for location based services." *Privacy Enhancing Technologies.* Springer Berlin Heidelberg, 2010.

[33] Yaghmour, Karim. Embedded Android: Porting, Extending, and Customizing. O'Reilly Media, Inc., 2013.

[34] Singh, A. (2013). *Diabolical or Smart.* Available at: `http://blog.singhanuvrat.com/`

[35] Schwartz, Edward J., Thanassis Avgerinos, and David Brumley. "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)." *Security and Privacy (SP), 2010 IEEE Symposium on.* IEEE, 2010.

[36] Andres, Miguel E., et al. "Geo-indistinguishability: Differential privacy for location-based systems." *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.* ACM, 2013.

[37] Machanavajjhala, Ashwin, et al. "Privacy: Theory meets practice on the map." *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on.* IEEE, 2008.