Building virtual reference monitor to assure Android Security

A R&D Report

Submitted in partial fulfillment of requirements for degree of

Masters of Technology

by

Asif Ali
Roll No : 143059009

under the guidance of

Prof. R.K. Shyamasundar

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

May, 2016

Contents

Abstract

Smartphone are become integral part of our life.This is possible because the smart-phone operating system allows third party application to use system API for completing its job. the major player in the smart-phone segment is ANDROID.which cover over 80 % of total smart-phone market share and got millions of application in its play store .However it also has its Dark side. There are thousands of malicious application in play store which can steal your credential and exploit the weakness of security policies in the ANDROID O.S. So in order to solve this problem we are using the concept of **"virtual monitor"** to assure the android security .

Chapter 1

Introduction

Most of the android developers are using third party libraries in their applications. For example 50% of the free apps embed third party libraries (like advertisement libraries etc.). They may used by developers to provide various features at significant reduced time and cost. For example in app purchases, PDF view, cloud computing etc.

Many third party libraries access privacy-sensitive information even without notification to users or application developers. For example over the analysis of 1,00,000 android apps reveals that third party libraries covertly utilize Andrid APIs to access privacy-sensitive resources such as GET_ACCOUNTS, READ_PHONE_STATE or READ_CALENDAR without mentioning them properly in their Developer's Guide. The current Android platform provides coarse-grained controls for regulating whether third party libraries access private information, allowing them to operate with the same permission as their host apps.

One of the technique here we presents FlexDroid, an extension to the Android permission system that allows app developers to control access to user's private information by third party libraries. Its main goal is to separate in-app privilege between third party libraries and host application while running all in the same process, the same UID privilege. FlexDroid provide an interface that enables developer to specify different permissions to each third party libraries using app manifest file. FlexDroid uses a new security mechainis called inter-process stack inspection to find the principal of the currently running code and according to identified principal, FlexDroid allows or denied the request made by the principal by dynamically adjusting the apps permission according to pre-specified permissions in the application manifest file.

Chapter 2

Motivation

Each Android app run in its own sandbox with UID, which is application-specific allocated at the time of installation. In order to use the resource outside the app sandbox it has to request to the main thread then the main thread check the permissions given in AndroidManifest.xml file. Android permission model offers either to accept all the requested permission or not to install the app. So the app can be granted to take resource any time, no matter what it needs or not. Third party libraries which are there in the android app to provide additional funcionality have exactly the same permissions results in more privilege than actually needed. As a result, third party libraries may be malicious and can exploit the app's SEND_SMS permission to send premium-rate sms.

2.1 Potential Attack Senarios

Below are the three attack scenarios due to overprivileged third party libraries regardless of the developers intention.

2.1.1 Libraries abusing undocumented permissions

App developers knows what third party library will use based on its permissions required in the documentation, specifying its must-have permissions and optionally required permissions. Due to lack in in-app privilege separation app can take permissions(as its host app) which were not mention in their documentation. If third party library dynamically checks whether it has certain permission it can abuse those permission without notifying users.

2.1.2 Contaminated libraries

Sometimes the libraries are legitimate but adversary is able to rewrite its binary or source code and redistribute it. App developer who uses these contaminated library code to develop the app are more vulnerable to privacy leaks, and may suffer monetary damage.

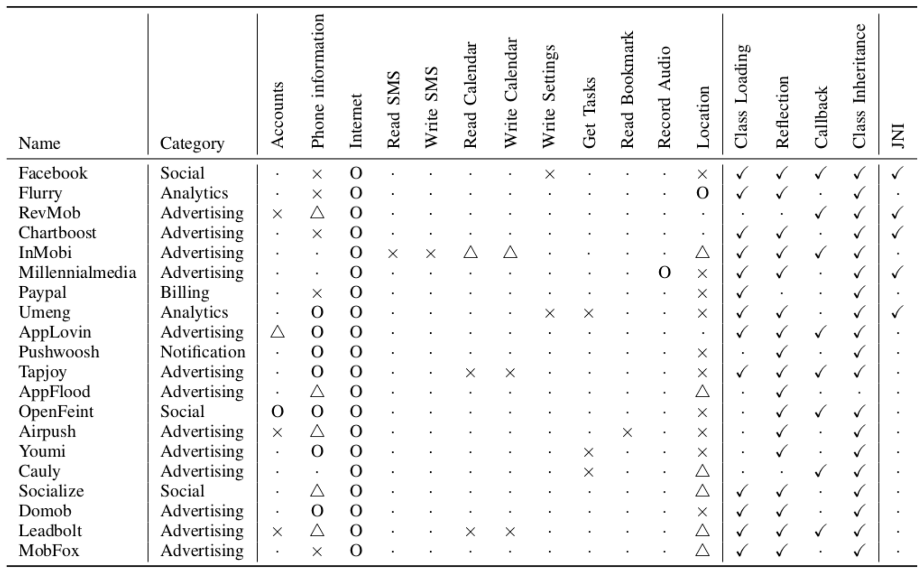
2.1.3 Vulnerable libraries

A third party may execute a class or javascript code downloaded from internet at runtime. If the mobile is connected to unsafe network and file is not encrypted the malicious code can be replaced by the attacker. Such a malicious code can exploit host app permissions to leak personal informations.

2.2 Real World Findings

We investigate different characteristics of several third party libraries berief summary of which is given below:

- 17 of 20 popular apps uses undocumented permissions.
- 72% of 295 third party libraries rely on dynamic code executions.
- 17% of 295 third party libraries uses JNI(Java Native Interface).



Name	Category	Accounts	Phone information	Internet	Read SMS	Write SMS	Read Calendar	Write Calendar	Write Settings	Get Tasks	Read Bookmark	Record Audio	Location	Class Loading	Reflection	Callback	Class Inheritance	JNI
Facebook	Social	·	×	O	·	·	·	·	×	·	·	·	×	✓	✓	✓	✓	✓
Flurry	Analytics	·	×	O	·	·	·	·	·	·	·	·	O	✓	✓	·	✓	·
RevMob	Advertising	×	△	O	·	·	·	·	·	·	·	·	·	·	·	✓	✓	✓
Chartboost	Advertising	·	×	O	·	·	·	·	·	·	·	·	·	✓	✓	·	✓	✓
InMobi	Advertising	·	·	O	×	×	△	△	·	·	·	·	△	✓	✓	✓	✓	·
Millennialmedia	Advertising	·	·	O	·	·	·	·	·	·	·	O	×	✓	✓	·	✓	✓
Paypal	Billing	·	×	O	·	·	·	·	·	·	·	·	×	✓	·	·	✓	·
Umeng	Analytics	·	O	O	·	·	·	·	×	×	·	·	×	✓	✓	·	✓	✓
AppLovin	Advertising	△	O	O	·	·	·	·	·	·	·	·	·	✓	✓	✓	✓	·
Pushwoosh	Notification	·	O	O	·	·	·	·	·	·	·	·	×	·	✓	·	✓	·
Tapjoy	Advertising	·	O	O	·	·	×	×	·	·	·	·	×	✓	✓	✓	✓	·
AppFlood	Advertising	·	△	O	·	·	·	·	·	·	·	·	△	·	✓	·	·	·
OpenFeint	Social	O	O	O	·	·	·	·	·	·	·	·	×	·	✓	✓	✓	·
Airpush	Advertising	×	△	O	·	·	·	·	·	·	×	·	×	·	✓	·	✓	·
Youmi	Advertising	·	O	O	·	·	·	·	·	×	·	·	×	·	✓	·	✓	·
Cauly	Advertising	·	·	O	·	·	·	·	·	×	·	·	△	·	·	✓	✓	·
Socialize	Social	·	△	O	·	·	·	·	·	·	·	·	△	✓	✓	·	✓	·
Domob	Advertising	·	O	O	·	·	·	·	·	·	·	·	×	✓	✓	·	✓	·
Leadbolt	Advertising	×	△	O	·	·	×	×	·	·	·	·	△	✓	✓	✓	✓	·
MobFox	Advertising	·	×	O	·	·	·	·	·	·	·	·	△	✓	✓	·	✓	·

TABLE I: Characteristics of third-party libraries. Columns 3-14 show the permissions potentially used by apps (O: required permission, △: optional permission, ×: undocumented permission). The rest of columns are related to runtime behavior and dependency with host apps.

Table -1 summarizes the results of our findings for third party libraries used in 1,00,000 Android apps. 20 of the top popular which use at least one permissions out of 16 permissions mentioned above.

2.3 Thread Model

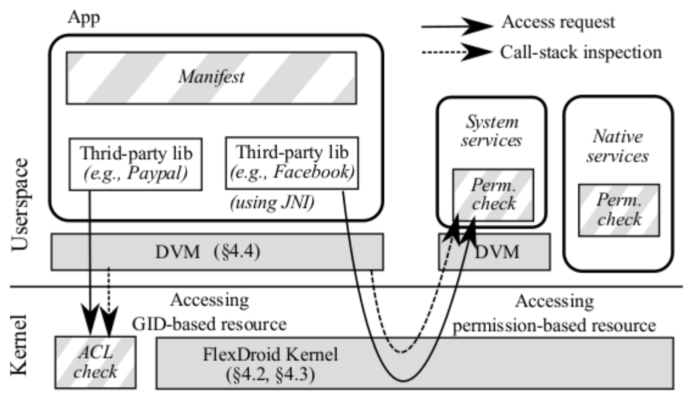
FlexDroid assumes all third party libraries are malicious and their code and logic are not visible to developers and they might use dynamic features of java language.

Chapter 3

FlexDroid Design

3.1 Overview

FlexDroid is a type of new permission system that adjusts the permissions of Android apps dynamically so as to enforce fine-grained controls of untrusted application modules.



App
Access request
Call-stack inspection
Manifest
Thrid-party lib (e.g., Paypal)
Third-party lib (e.g., Facebook)
(using JNI)
System services
Perm. check
Native services
Perm. check
DVM (§4.4)
DVM
Userspace
Kernel
Accessing GID-based resource
Accessing permission-based resource
ACL check
FlexDroid Kernel (§4.2, §4.3)

Key features of FlexDroid are as follows(as given in figure-1)

- App developers specify a set of permissions for each individual module in android manifes file.
- Upon request of resource access, FlexDroid identifies the context of execution by inspecting the Dalvik call stack.
- FlexDroid then determines whether to accept or decline the request according to permissions commonly granted to the modules.

```
<flexdroid android:name="com.third.party.library"
           android:mockOnException="true">
  <allow android:permission="android.permission.READ_CONTACTS" />
  <allow android:permission="android.permission.READ_EXTERNAL_STORAGE" />
  <allow android:permission="android.permission.INTERNET" />
</flexdroid>
```

FlexDroid provide programming interface to developers in the form of simple XML manifes rules, to restrict third party library privileges. Figure 2 shows an example rule in FlexDroid's policy. FlexDroid provides a *mockOnException* attribute to enable developers to choose whether FlexDroid should offer mock data(ex. fake IMEI code) upon a request for an unauthorised resource.

3.2 Challenges

Achieving the above feature for a wide range of Android Apps presents several challenges:

3.2.1 Secure Inter-Process Stack Inspection

Android performs permission checking in a separate address space to protect memory tampering or in the kernel to secure the use of low level system calls and FlexDroid does so too. FlexDroid require extra information to understand the context for fine-grained access control. To do that FlexDroid creates extra thread called *stack tracer*, that collects Dalvik call trace and send it to the permission checker upon request. It is also possible that malicious third-party library may also create the same name thread and send fake call data trace. To protect from that FlexDroid uses secure transmission channel between individual stack tracer thread and permission checker and ignores all attempts to use stack transmission channel except those by the stack tracer.

The detailed procedure of the inter-process stack inspection differs according to the type of requested resource. Android resource broadly fall into two categories: user-space and kernel spcace depending upon which component they are responsible for access control.

User-space Resources: Apps access userspace resources through interfaces provided by the Android system services. Such resources include system resources(GPS,camera etc.) and app components (activity, service, content provider, and broadcast reciever).

In Android, when an app requests to access a resource to a corresponding system service (eg. Location Manager), the system service process queries the PackageManager(PM) to see whether the app has proper permission. FlexDroid provides inter-process stack inspection to conduct access control at the granularity of a module. Upon request from PM, the stack tracer of the app passes Dalvik call trace to PM via secure stack transmission channel mentioned above. PM then looks through all the modules involved in the current access request to find out the granted permission among them.

Kernel-space Resources: Apps access kernel resources (Internal ,external strage,Bluetooth etc) via system calls. In Android, unlike the user-space resource case, PM does not conduct permission checking for kernel level resources. Instead at the initialisation of an app, PM passes a set of permissions checking itslef, using Linux's Access Control List(ACL). FlexDroid enforces the kernel to conduct inter-process inspection through the stack transmission channel during permission checking. Additionally , upon installation, PM sends to the kernel a set of granted permissions to each module in the app so as to avoid expensive user-kernel communications later on.

3.2.2 Ensuring Dalvik Stack Integrity against Native Code

Android supports the Java Native Interface and allows developers to parts of an app or library to incorporate native libraries.

Despite its advantages, it renders memory safety of Java programming language obselete, which results in security threats in Android. Some of the possible attacks by malicious JNI code are as follows:

Compromising the stack trace An attacker can effectively guess the address of a memory region used by stack tracer and manipulate the content.

Manipulating Dalvik stack An attacker can directly manipulate Dalvik stacks, thereby corrupting the integrity of the call stacks used for inspection.

Defenses

Process Separation: Process separation naturally supports sandboxing. Since it lets JNI thread run in a separate process, JNI code is not able to access the region of Java code is not able to access the memory region of java code directly.

Software fault Isolation(SFI): SFI restricts memory accessible by JNI through masking the operands of store and jump instructions used in JNI.

Hardware fault Isolation(HFI): FlexDroid uses HFI approach to implement JNI sandboxing, as the process separation and SFI approaches significant overheads compared to HFI. We introduced two memory domains, JNI and Java which represent restricted memory regions assigned for JNI and Java code respectively.

But there are many challeges in applying this design of FlexDroid. In the default setting, when JNI attempts to access memory regions for the stack, heap, and shared libraries at the java domain, JNI domain fault occurs since only JNI can access JNI domain. To overcome this, FlexDroid provides the JNI domain with separate stack and heap and loads a set of necessary shared libraries at the JNI domain.

Similary FlexDroid divides the pointer variables as valid or potentially malicious pointers and disallows latter ones to be passed to java code via Java API. FlexDroid manages a table called Valid Address Table (VAT) that maintains a list of memory addresses which have been returned from Java code via Java API calls.

3.2.3 Dynamic Permission Management

Java Reflection is a feature in Java which allows programmers to inspect or modify any code at run time across Java classes for various benefits. At the same time it creates a lot of problem for creating the mapping between modules and permissions accurately. For example, a third party library may create and run the code dynamically using reflection as host module.

Chapter 4

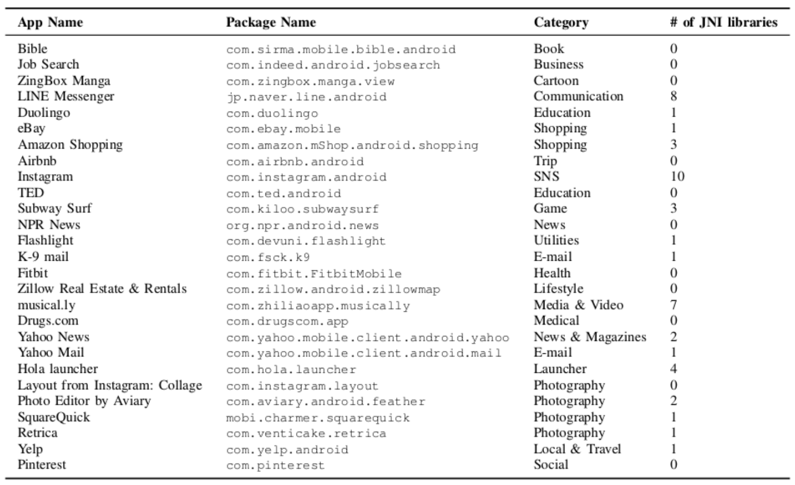
Experimental Setup

4.0.1 System Configurations

- Nexus 5 (2.265 Ghz quad core CPU 2GB RAM)
- FlexDroid
- Android 4.4.4(KitKat)

4.0.2 Experiment

We searched out 32 top apps from App Annie and downloaded it, from Google Play store. Then we ran out these apps in Android with and without FlexDroid, and checked to see if an app crashes during the execution.



App Name	**Package Name**	**Category**	**# of JNI libraries**
Bible	`com.sirma.mobile.bible.android`	Book	0
Job Search	`com.indeed.android.jobsearch`	Business	0
ZingBox Manga	`com.zingbox.manga.view`	Cartoon	0
LINE Messenger	`jp.naver.line.android`	Communication	8
Duolingo	`com.duolingo`	Education	1
eBay	`com.ebay.mobile`	Shopping	1
Amazon Shopping	`com.amazon.mShop.android.shopping`	Shopping	3
Airbnb	`com.airbnb.android`	Trip	0
Instagram	`com.instagram.android`	SNS	10
TED	`com.ted.android`	Education	0
Subway Surf	`com.kiloo.subwaysurf`	Game	3
NPR News	`org.npr.android.news`	News	0
Flashlight	`com.devuni.flashlight`	Utilities	1
K-9 mail	`com.fsck.k9`	E-mail	1
Fitbit	`com.fitbit.FitbitMobile`	Health	0
Zillow Real Estate & Rentals	`com.zillow.android.zillowmap`	Lifestyle	0
musical.ly	`com.zhiliaoapp.musically`	Media & Video	7
Drugs.com	`com.drugscom.app`	Medical	0
Yahoo News	`com.yahoo.mobile.client.android.yahoo`	News & Magazines	2
Yahoo Mail	`com.yahoo.mobile.client.android.mail`	E-mail	1
Hola launcher	`com.hola.launcher`	Launcher	4
Layout from Instagram: Collage	`com.instagram.layout`	Photography	0
Photo Editor by Aviary	`com.aviary.android.feather`	Photography	2
SquareQuick	`mobi.charmer.squarequick`	Photography	1
Retrica	`com.venticake.retrica`	Photography	1
Yelp	`com.yelp.android`	Local & Travel	1
Pinterest	`com.pinterest`	Social	0

TABLE V: Compatibility test. Running popular apps on FlexDroid without applying FlexDroid policy.

Table-5 shows a list of apps run as normal in presence of FlexDroid. But some apps are there which doesn't run with FlexDroid app. Those apps were Waze Social GPS Map and Travel(com.waze), Uber(com.ubercab),Adobe Acrobat Reader(com.adobe.reader), Facebook(com.facebook.kata UC Browser(com.UCMobile.intl).The problem we searched out was coming from JNI sandbox.Then we found that root of the fault was PthreadID,mmap() and free(). JNI code of Waze fails because in the safety check, it compares the Pthread ID obtained from Java to the thread id coming from JNI code. Since both are not the same it came to crash. Separating Pthread Id from thead structure was a way to avoid this problem, although it requires modification of JNI code.

Uber employs a JNI library SnappyDB which is a key-value database for Android. It maps data stored in a file to memory pages using mmap(). Since memory pages are out of JNI domain, it generates domain faults. Allocating memory to each separate region depending upon the caller of mmap() is our future work.

When executing JNI code of Adobe Acrobat Reader, free() was creating problem. We haven't figure what creating this fault but we beleive that we will solve it through engineering.

Bibliography

[1] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: Analyzing the android permission specification. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, 2012.

[2] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. Mockdroid: Trading privacy for application functionality on smartphones. In Proceedings of the 12th Workshop on Mobile Computing Systems and Applications,2011.

[3] R. Bhoraskar, S. Han, J. Jeon, T. Azim, S. Chen, J. Jung, S. Nath,R. Wang, and D. Wetherall. Brahmastra: Driving apps to test the security of third-party components. In 23rd USENIX Security Symposium, Aug.2014.

[4] acl Linux man page. `http://linux.die.net/man/5/acl`.

[5] Actionbarsherlock Android SDK.`http://actionbarsherlock.com/`.

[6] Adobe Pdf Library SDK. `http://www.adobe.com/devnet/pdf/library.html`.

[7] android-apktool: A tool for reverse engineering Android apk files. `https://code.google.com/p/android-apktool/`.