一种基于 FlowDroid 的 Android 隐私保护方法

马绍菊 万 良* 杨 婷 马林进
(贵州大学计算机科学与技术学院 贵州 贵阳 550025)

摘 要 随着移动设备市场的扩大,Android 智能系统占据了手机市场的很大份额,手机设备是承载用户隐私数据较多的移动设备。由于 Android 系统的开源特性,其存在很多安全隐患。提出一种基于 FlowDroid 的 Android 增强型隐私保护方法。对 Android 应用进行静态污点分析,判断其是否存在隐私泄露,并基于 FlowDroid 静态污点分析工具实现与验证。通过验证表明提出的方法是有效的。

关键词 静态污点 隐私泄露 程序间控制流图 Android 操作系统

中图分类号 TP309 **文献标识码** A DOI: 10.3969/j.issn.1000-386x.2017.05.055

AN ANDROID PRIVACY PROTECTION METHOD BASED ON FLOWDROID

Ma Shaoju Wan liang* Yang Ting Ma Linjin
(*College of Computer Science and Technology, Guizhou University, Guiyang 550025, Guizhou, China*)

Abstract With the expansion of the mobile device market, Android operating system occupies a large share of the mobile phone market. Mobile phone equipment is carrying more user privacy data. As the open source features of Android system, there are many security risks. In this paper, we propose an Android-enhanced privacy protection method based on FlowDroid, static taint analysis method of Android applications, and whether there is privacy leak, and based on FlowDroid static stain analysis tool to achieve and verify. Verification shows that the proposed method is valid.

Keywords Static taint Privacy leak ICFG (Inter-procedural Control-Flow Graph) Android operating system

0 引 言

Android 隐私保护方法主要包括两类: 静态检测和动态检测[1]。静态检测方法是指在不需要运行应用的情况下,对 apk 文件进行分析的一种方法。动态检测方法是指在程序运行过程中,通过提取应用的行为或动态检测应用的操作,分析出应用的恶意行为。Android 隐私保护研究方面,Lu 等提出了 CHEX[2],主要用于 Android 应用的漏洞劫持和静态分析,静态分析中,CHEX 只能最多考虑单个对象敏感,存在一定的局限性。Yang 等提出了 LeakMiner[3] 静态分析方法,该方法对上下文不敏感,检测的精确性不高。Gibler 等提出的 AndroidLeaks[4] 方案对于字段和对象都不敏感,大大降低了检测的精确性。Reina 等提出的 CopperDroid[5] 动态检测方法在 Android 底层 Linux 操作系统的基础上重建 Android 应用组件的行为,从而找出恶意的活动。该方法需要在应用运行时进行检测,占用系统资源高,降低应用运行效率。

随着 Android 手机用户的不断增长,用户隐私泄露问题也开始泛滥。Android 手机作为移动设备的一种势头,对其隐私保护的研究具有重要意义。本文提出一种增强型静态污点分析方法 DFlowDroid,在 FlowDroid 工具的基础上进行一定的改进,提高了静态污点分析的准确性,具有实际意义。

1 背景知识

1.1 Android 基础

Android[6] 应用使用 Java 语言编写,同时也支持本地语言 C。

收稿日期: 2016-09-02。马绍菊,硕士生,主研领域: 信息安全,Android 隐私保护。万良,教授。杨婷,硕士生。马林进,硕士生。

Android 应用程序被打包成一个 apk 文件，apk 本身就是一个压缩文件，可以将其直接解压，解压后一般包括六个部分。每部分的含义如表 1 所示。

表 1 apk 文件结构

res	存放资源文件的目录，包括应用的图片、布局、字符串等资源文件
META-INF	存放应用签名信息，用来保证 apk 的完整性和系统的安全
lib	存放 ndk 编译出来的 so 库，即动态链接库
resources. arsc	编译后的二进制资源文件的索引，即 apk 文件的资源表
classes. dex	最终生成的 dalvik 字节码，包含了应用的可执行代码
AndroidManifest. xml	应用的配置文件，包含组件、应用权限、sdk 版本等信息

Android [6] 应用开发中通常涉及四个组件：活动(Activity)、服务(Service)、广播接收者(BroadCastReceiver)、内容提供者(ContentProvider)。这四个组件互相独立，但可以相互调用，实现特定的功能。Android 应用开发中，页面跳转及数据传递使用 Intent 意图来实现。意图是 Android 数据流分析的一个重要部分。

1.2 Soot 框架

Soot [7] 是 McGill 大学的 Sable 研究小组所研发的一种 Java 优化框架，现在被世界各地的研究人员用来对 Java 工程和 Android 应用进行分析。为了达到分析的目的，Soot 提供了四种不同抽象层次的中间表示。这些中间表示在代码分析过程中有着各自的用途。同时，Soot 为中间表示定义了五种数据结构，分别是 Scene、SootClass、SootMethod、SootField 和 Body。这些数据结构使用面向对象技术来实现，作为一个通用类，需要时很容易被使用。

Soot [7] 可以处理格式有 Java 字节码和源码、Android 字节码、jimple 中间表示、Jasmin。处理后输出格式可以为 Java 字节码、Android 字节码、jimple 中间表示和 Jasmin。Soot 可以从任何输入格式到任何输出格式，比如可以从 Android 字节码到 Java 源码，Java 源码到 Jimple 中间输出。Soot 提供了多种类型的分析，主要包括：生成调用结构图、指向分析、内部数据流分析、与 FlowDroid 结合的静态污点分析等。Soot 的工作原理是：将程序转换为任意一种中间表示，对这些中间表示进行分析，可以对代码进行优化、添加标签等操作。

Jimple [7] 是 Soot 中最重要的一种中间表示，很多分析都是在 Jimple 级别上实现的。它是一种类型化的、3 - 地址、基于语句的中间表示。在 JDK1. 4 及更高版本中，Jimple 可以直接由 Java 源码生成，JDK1. 5 及更高版本中，Jimple 可以直接由 Java 字节码或 Java 类生成。在 Java 字节码到 Jimple 的转换过程中，为了简化栈地址、子程序操作及移除 Java 规范 jsr 指令，Jimple 引入局部变量。同时，Jimple 会移除冗余的代码，如没有使用的变量和未使用的内存分配。该转换过程中最重要的一步是线性化操作，保证所有的语句只涉及到最多三个变量或常量，也就是所谓的 3 - 地址，这使得分析过程中更加简便和高效，例如：

$$\text{int}x = f(f.\,bar(12) + a) \times b \rightarrow \begin{cases} \$i4 = \$i3 + i0 \\ i2 = \$i4 \times i1 \end{cases}$$

Jimple 表示中只需要处理 15 种语句，而在 Java 字节码中需要处理 200 多种指令，因此，Jimple 大大简化了 Java 字节码。可以对 Jimple 进行优化，优化后将其重新转换为 Java 字节码，这样可以减小系统的开销，使得代码运行更有效率。

Soot [7] 框架为 Android 静态分析提供了重要的预备知识，FlowDroid 继承了 Soot 框架，特别是 Soot 中的三地址中间表示 Jimple 和精确的数据流调用分析框架 Spark [8]。Dexpler 插件允许 FlowDroid 将 Android 的 Dalvik 字节码转换为 Jimple [9]。

1.3 FlowDroid 框架

FlowDroid [10] 对 Android 应用的分析中，其输入为 Android 应用包 apk 文件，输出为程序的数据流和流路径。

Android 应用程序不像 Java 应用那样，有入口 Main 函数。Android 应用包含了许多入口节点，它的方法隐含的被 Android 框架调用。Android 操作系统对应用中每一个组件定义了完整的生命周期，这些组件主要包括 Activity、Service、BroadCastReceiver 和 ContentProvider，所有的这些组件都需要在应用配置文件 AndroidManifest. xml 中进行配置。因此，当构建一个调用图时，apk 分析不能简单地从检测预定义的 main 方法开始，但 Android 应用中每个组件都有函数来反映此组件的生命周期，因此，FlowDroid 构建一个通用的 dummyMainMethod( ) 方法来模拟生命周期 [10]。

FlowDroid [10] 是 Android 静态污点分析工具，其检测效率高，检测流程如图 1 所示。首先，FlowDroid 以 apk 文件作为输入，将其反编译之后，分别分析 AndroidManifest. xml 文件、classes. dex 文件和 layout 中的布局文件。通过这些文件的解析来获得应用对应的 source、sink 以及入口节点，得到生命周期及回调函数列表。接着，FlowDroid 通过生命周期及回调函数列表

来产生 dummyMainMethod( ) 函数,FlowDroid 中的 Dexpler 插件将 apk 中的. dex 文件转为 soot 框架的 Jimple 中间表示,被用来产生调用图 CG(Call Graph) 和程序间控制流图 ICFG(Inter-procedural Control-Flow Graph) [9] 。在代码检测开始时,静态分析方法将通过遍历 ICFG 来追踪污点,FlowDroid 使用 Heros [10] 框架来实现 IFDS 中的污点追踪算法,生成数据流和流路径 [6] 。

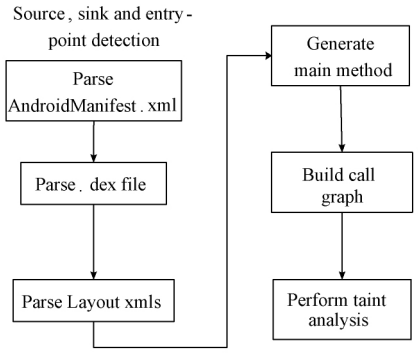

图 1　FlowDroid 分析流程

Flowdroid 基于 IFDS [11] 框架实现了一套 Android 静态污点分析原型,它对 Android 框架和应用特性做了大量的建模工作。FlowDroid 准确性高,但是比较臃肿,在资源和时间效率上存在一些问题。同时,由于 apk 的一些正常应用需求,该方法存在一定的误报率。

2　基于 FlowDroid 的 Android 隐私保护方法

2.1　方法分析

本文提出的方法是 DFlowDroid,该方法方法借助 FlowDroid 中的 Dexpler 插件,将 Android 应用的源码转换为 Soot 框架的 jimple 中间表示,由 jimple 中间表示来产生程序间控制流图。借助 Soot 框架,DFlowDroid 方法在 FlowDroid 的基础上,增加了一定的判定条件,降低了误报率。

为了更清楚地说明分析方法,这里先定义一些符号变量。

Source: apk 应用中的 source 集合,即污染源集合。该集合来源于 Android 系统中某些涉及敏感信息的接口。

Sink: apk 应用中的 Sink 集合,即污点接收端的集合。该集合表示的是 Android 系统中所有可能导致用户隐私泄露的接口。

Source [*i*]: Source 集合中的某个元素。

Sink [*j*]: Sink 集合中的某个元素。

NextNode: 程序间控制流图中的节点。程序间控制流图,即 ICFG,是由 Soot 中间表示 Jimple 而产生的一种基于语句的程序间调用流图。

T [*Source* [*i*] → *Sink* [*j*]]: apk 中 Source 到 Sink 的污染路径。

T [*SafeSource* [*k*] → *SafeSink* [*p*],*Info*]: 给定的 source 到 sink 的安全路径集合。

Info 表示 *Source* [*k*] → *Sink* [*p*] 传输路径中携带的一些辅助数据信息,可以为字符串类型,且 *Info* ∈ *Info* [*t*] 。

Info [*t*]: 白名单数据的集合,即如果安全路径中携带的辅助数据信息 *Info* 属于该集合,则表明该路径不是隐私泄露路径。

Source 和 Sink 来源于 Android 系统源码提供的接口方法,这些 Source 和 Sink 点在处理的时候存放在同一个文件中,通过特殊符号来标记。由于涉及到的 Source 和 Sink 很多,这里列举部分来说明问题。Source 使用"_SOURCE_"来标记。

1) <android. location. Location: double getLatitude( ) > -> _SOURCE_　　//获取用户所在位置纬度信息

2) <android. location. Location: double getLongitude( ) > -> _SOURCE_ //获取用户所在位置经度信息

3) <android. telephony. TelephonyManager: java. lang. String getDeviceId( ) > android. permission. READ_PHONE_STATE -> _SOURCE_　　//获取用户设备 ID

4) <android. content. Intent: android. net. Uri getData( ) > -> _SOURCE_　　//获取 Intent 中携带的数据

5) <android. os. Handler: android. os. Message obtainMessage( ) > -> _SOURCE_　　//读取用户短信

上述表示中,冒号前面的部分表示该方法所在的类,冒号以后的部分表示该 source 方法及其返回类型。所有的 source 都涉及到获取数据的过程。source 一般包含 get、write、obtain 等获取数据的关键字。

Sink 使用"_SINK_"来标记:

1) <android. os. Bundle: void putFloat(java. lang. String, float) > -> _SINK_　　//将数据存入 Bundle 中

2) <android. content. Intent: android. content. Intent putExtra (java. lang. String,int) > -> _SINK_ //添加 Intent 携带的数据

3) <android. content. Context: void sendBroadcast(android. content. Intent) > -> _SINK_　　//发送广播

4) <android. content. SharedPreferences$ Editor: android. content. SharedPreferences$ Editor putBoolean(java. lang. String, boolean) > -> _SINK_　　// 添加 SharedPreferences 数据

5) <android. telephony. SmsManager: void sendTextMessage (java. lang. String, java. lang. String, java. lang. String, android. app. PendingIntent,android. app. PendingIntent) > android. permis-

sion. SEND_SMS - > _SINK_ //发送短信

同样,整条数据反应出该 Sink 所在的类、方法、方法参数及其返回值。Sink 中的方法都涉及到写或发送数据的过程。一般包含 put、read、send 等关键字。

检测过程中,判断待检测的 apk 应用中涉及到隐私数据的接口是否属于 *Source* 集合,如果属于,则追踪。在数据传输过程中,检测该数据的流向是否涉及到 *Sink* 集合中的方法,如果有,则表示可能存在隐私泄露。同时,获取该数据的附加信息,判断其是否属于白名单,如果属于,则表示应用是正常的,否则表示该应用存在泄露用户隐私的行为。

根据给定符号定义,检测方法可以形式化的描述为:

$$T(Source_i \rightarrow Sink_j) = \cup \begin{cases} T(Source_i \rightarrow NextNode) & NextNode \in Sink \\ T(Source_i \rightarrow Sink_j) \setminus T & T \in T(SafeSource_k \rightarrow SafeSink_p, Info) \\ T(Source_i \rightarrow Sink_j) & \text{otherwise} \end{cases}$$

对于以上等式,分析如下:

$T(Source_i \rightarrow NextNode)$,$NextNode \in Sink$ 表示在经过 ICFG 进行分析的过程中,以每一个 apk 中的 Source 源作为起点,根据 ICFG 找到下一个节点,判断该节点是否属于 apk 中的 *Sink* 集合中的元素。如果属于,则表示该 Source 和 Sink 之间存在污染路径,将其保存到对应集合中。

$$T(Source_i \rightarrow Sink_j) \setminus T, T \in T(SafeSource_k \rightarrow SafeSink_p, Info)$$

表示在分析过程中,某些特定 Source 到 Sink 的路径是安全的路径,这些路径用于正常的功能。例如,某导航应用,为了达到导航的目的,需要获取用户的精确位置,然后发送到给定服务器,服务器再制定出最佳路线发送给用户,而 *Info* 就表示接收端的附加信息。该过程存在隐私窃取,但并不存在隐私泄露。分析过程中将该类安全路径保存到集合 $T[SafeSource_k \rightarrow SafeSink_p]$ 中,通过判断传输过程中的特定数据 *Info*,来决定是否将其从路径集合中去除。如果检测结果路径属于该集合的元素,并且 *Info* 满足白名单集合,则将其从污染路径 $T[Source_i \rightarrow Sink_j]$ 中去除。

$T(Source_i \rightarrow Sink_j)$,otherwise 表示其他情况。

通过以上分析可以看出,某些恶意应用的行为与正常应用行为存在相似性,甚至相同。为了找出这些隐藏的恶意应用,在静态污点分析过程中,加入部分关键判定信息,可以提高检测的准确性。

2.2 实验结果分析

Fowdroid 提供了一个 Android 测试集 DroidBench[13],1.0 版本中包含了 39 个不同的 Android 应用。所有的这些应用单独包含某个需要检测的模块,如别名、生命周期、数组等,而对于某些原本就不存在隐私泄露的应用,Flowdroid 很可能出现误报。因此,在测试集中,我们添加两个不存在隐私泄露的 Android 应用作为检测对象,这些应用描述如表 2 所示。

表 2 Droidbench 添加的 apk

应用名称	描述	*Info*	是否存在隐私泄露
PhoneLocation.apk	将手机当前位置获取,发给合法的应用进行定位,此处假定合法定位包名为"com. guida. location",合法类名为"com. guida. location. Location"	"com. guida. location: com. guida. location. Location"	否
PhoneSafe. apk	当 sim 卡发生变化时,向固定号码发送警告提示。此处假定固定号码及短信内容为"13885849901: sim card changed!"	"13885849901: sim card changed!"	否

通过对 Droidbench 中 41 个 apk 做静态分析,这些 apk 总体情如表 3 所示。

表 3 检测对象情况

检测对象总数	存在隐私泄露总数	不存在隐私泄露总数
41	28	13

FlowDroid 与 DFlowDroid 实验结果对比如表 4 所示。

表 4 实验结果对比

	FlowDroid	DFlowDroid
正确数	26	28
误报数	6	4
漏报数	9	9
准确率(正确数/总数)	63.41%	68.29%

实验结果可以看到,DFlowDroid 对某些特定的应用具有识别作用,它能够根据所给安全路径中携带的安全数据进行判断,从而降低误报数,提高检测的准确性。

3 结 语

本文基于 FlowDroid 完成了 Android 恶意应用静态检测增强型方法 DFlowDroid。该方案通过对 FlowDroid 进行改进,在一定程度上提高了准确率,但还是

存在一定的局限性。首先,FlowDroid 本身的冗余、效率问题没有得到更完美的解决。其次,DFlowDroid 在检测过程中,需要从多方面收集安全路径集合中携带的安全数据信息,这需要对大量数据进行分析挖掘。而本文通过两个简单的 apk 作为实验对象进行检测,存在局限性。

Android 手机日益普遍,很多手机厂商对 Android 原生系统进行修改后应用于自己的手机品牌,这就带来了各种各样的问题。因此,对于手机隐私泄露的研究具有实际意义。在后续研究中,将在该工具的基础上不断完善,争取达到在提高效率的同时,能够结合动态检测相关技术,实现一个高效率、高准确性的 Android 恶意应用检测工具。

参 考 文 献

[1] 张玉清, 王凯, 杨欢,等. Android 安全综述 [J]. 计算机研究与发展, 2014, 51(7): 1385-1396.

[2] Lu L, Li Z, Wu Z, et al. CHEX: statically vetting Android apps for component hijacking vulnerabilities [C] // ACM Conference on Computer and Communications Security. ACM, 2012: 229-240.

[3] Yang Z, Yang M. LeakMiner: Detect Information Leakage on Android with Static Taint Analysis [C] // Software Engineering. IEEE, 2012: 101-104.

[4] Gibler C, Crussell J, Erickson J, et al. AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale [C] // International Conference on Trust and Trustworthy Computing. Springer-Verlag, 2012: 291-307.

[5] Reina A, Fattori A, Cavallaro L. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors [C] // EUROSEC, Prague, Czech Republic, 2013: 5-11.

[6] Reps T, Horwitz S, Sagiv M. Precise interprocedural dataflow analysis via graph reachability [J]. Lecture Notes in Computer Science, 1995, 167(96): 49-61.

[7] Einarsson A, Nielsen J D. A survivor's guide to Java program analysis with soot [D]. BRICS, Department of Computer Science, University of Aarhus, Denmark, 2008: 16-63.

[8] Lhoták O, Hendren L J. Scaling Java Points-to Analysis Using SPARK [C] // Compiler Construction, International Conference, Cc 2003, Held As. DBLP, 2003: 153-169.

[9] Bartel A, Klein J, Traon Y L, et al. Dexpler: converting Android Dalvik bytecode to Jimple for static analysis with Soot [C] // ACM Sigplan International Workshop on State of the Art in Java Program Analysis. ACM, 2013: 27-38.

[10] Arzt S, Rasthofer S, Fritz C, et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps [J]. ACM SIGPLAN Notices, 2014, 49(6): 259-269.

[11] Bodden E. Inter-procedural data-flow analysis with IFDS/IDE and Soot [C] // ACM Sigplan International Workshop on the State of the Art in Java Program Analysis. ACM, 2012: 3-8.

[12] Fritz C. Secure Software Engineering/DroidBench [EB/OL]. https://github.com/secure-software-engineering/DroidBench/.

(上接第 222 页)

[3] 刘姝威,陈伟忠,王爽,等. 提高我国科技成果转化率的三要素 [J]. 中国软科学,2006(4): 55-58,123.

[4] 李霞. 高校创新型科研团队知识共享行为、学习行为及团队绩效研究 [J]. 软科学,2012,26(6): 83-87,91.

[5] 李纲,刘先红. 知识缺口视域下科研团队成员选择研究 [J]. 科技进步与对策,2015,32(12): 139-143.

[6] 吴卫,陈雷霆. 谈高校科研团队的组建与管理 [J]. 科技管理研究,2006,26(11): 140-141,155.

[7] 邢一亭,孙晓琳,王刊良. 科研团队合作效果研究——一个高校科研团队合作状况的调查分析 [J]. 科学学与科学技术管理,2009,30(1): 181-184.

[8] 肖伟,赵嵩正,魏庆琦. 基于 AHP 模糊优先规划的虚拟团队成员选择方法 [J]. 统计与决策,2009(4): 151-153.

[9] 晋琳琳. 基于知识溢出的科研团队成员选择模型研究 [J]. 科技管理研究,2010,30(12): 185-188.

[10] 许正权,王华清,王红军,等. 基于模糊相似优先比法的高校教学团队成员选择 [J]. 价值工程,2014(26): 1-2.

[11] 李浩君,项静,华燕燕. 基于 KNN 算法的 mCSCL 学习伙伴分组策略研究 [J]. 现代教育技术,2014,24(3): 86-93.

[12] Merigó J M, López-Jurado P, Gracia M C, et al. A method under uncertain information for the selection of students in interdisciplinary studies [J]. International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering, 2009, 3(7): 1383-1390.

[13] Irving R W, Leather P. The complexity of counting stable marriages [J]. SIAM Journal of Computing, 1986, 15(3): 656-667.

[14] Jung J J, Jo G S. Brokerage between buyer and seller agents using constraint satisfaction problem models [J]. Decision Support Systems, 2000, 28(4): 293-304.

[15] 李铭洋,樊治平,乐琦. 考虑稳定匹配条件的一对多双边匹配决策方法 [J]. 系统工程学报,2013,28(4): 454-463.

[16] Martínez R, Massó J, Neme A, et al. The blocking lemma for a many-to-one matching model [J]. Journal of Mathematical Economics, 2010, 46(5): 937-949.

[17] 沈国军. 基于多目标混合遗传算法的人力资源配置模型构建 [J]. 统计与决策,2013(21): 60-63.