计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2017.20150928
54(2):313-327, 2017

Android 动态加载与反射机制的静态污点分析研究

乐洪舟[1] 张玉清[1,2] 王文杰[2,3] 刘奇旭[2,3]
[1](综合业务网理论及关键技术国家重点实验室(西安电子科技大学) 西安 710071)
[2](中国科学院大学国家计算机网络入侵防范中心 北京 101408)
[3](信息安全国家重点实验室(中国科学院信息工程研究所) 北京 100093)
(yuehz@nipc.org.cn)

Android Static Taint Analysis of Dynamic Loading and Reflection Mechanism

Yue Hongzhou[1], Zhang Yuqing[1,2], Wang Wenjie[2,3], and Liu Qixu[2,3]
[1](*State Key Laboratory of Integrated Services Networks* (*Xidian University*), *Xi'an* 710071)
[2](*National Computer Network Intrusion Protection Center*, *University of Chinese Academy of Sciences*, *Beijing* 101408)
[3](*State Key Laboratory of Information Security* (*Institute of Information Engineering*, *Chinese Academy of Sciences*), *Beijing* 100093)

Abstract Privacy leakage is one of the most important issues in the current Android security. The present most important method to detect privacy leakage is taint analysis. Because of its high code coverage and low false negative, the technique of static taint analysis is widely used in the detection of Android privacy leakage. However, the existing static taint analysis tools cannot do effective taint analysis for Android dynamic loading and reflection mechanism. Taking into account the present situation that Android dynamic loading and reflection mechanism are being used more and more widely, we focus on how to enable static taint analysis tools to effectively deal with Android dynamic loading and reflection mechanism. We modify the Android source code to enable the Android system to timely store the loaded dex files and reflection invocation information during the running process of an Android app. This information will be used to guide the static taint analysis process of the app and a policy that replacing the reflective method invocation with non-reflective method invocation is proposed. Based on these ideas, a taint analysis tool—DyLoadDroid is proposed, which has made some improvements of the state-of-the-art static taint analysis tool—FlowDroid and can do effective taint analysis for Android dynamic loading and reflection mechanism. Sufficient experimental results show that DyLoadDroid is very effective in tackling the problem of static taint analysis of Android dynamic loading and reflection mechanism.

Key words Android privacy leakage; taint analysis; dynamic loading; reflection

摘 要 隐私泄露是当前 Android 安全中最为重要的问题之一,目前检测隐私泄露的最主要方法是污点分析.Android 静态污点分析技术凭借其代码覆盖率高、漏报率低的特点而被广泛应用在 Android 应用隐私泄露的检测上.然而,现有的静态污点分析工具却不能对 Android 动态加载和反射机制进行有效

收稿日期:2015-10-26;**修回日期**:2016-03-22
基金项目:国家自然科学基金项目(61572460,61272481,61303239);信息安全国家重点实验室开放课题(2015-MS-06,2015-MS-04)
This work was supported by the National Natural Science Foundation of China (61572460, 61272481, 61303239) and the Open Project of the State Key Laboratory of Information Security (2015-MS-06, 2015-MS-04).

污点分析.鉴于当前 Android 动态加载和反射机制被越来越广泛地应用的现状,对如何使 Android 静态污点分析工具有效地处理 Android 应用的动态加载和反射机制的问题进行了研究.对 Android 源码进行了修改,使 Android 系统能够对 Android 应用实际运行中加载的 dex 文件和反射调用信息进行实时存储,并利用这些信息对 Android 静态污点分析过程进行引导.以当前领先的静态污点分析工具 FlowDroid 为基础,对其进行了改进,提出了使用非反射调用语句替换反射调用语句的策略,实现了一个能够对 Android 动态加载和反射机制进行有效污点分析的工具——DyLoadDroid,并通过实验验证了其在处理 Android 动态加载和反射机制的污点分析问题上的有效性.

关键词 安卓;隐私泄露;污点分析;动态加载;反射

中图法分类号 TP309

当前 Android 隐私泄露问题泛滥,不管是恶意应用还是非恶意应用,都存在隐私泄露问题[1].恶意应用常常以侵犯用户的隐私作为目的[2],为恶意应用制造者或传播者提供非法利益[3].非恶意应用为了实现特定功能或提供更好的服务,也存在侵犯用户隐私的问题,例如,手机地图应用为了向用户提供用户所在地的地图信息,会获取用户的地理位置等隐私信息[4];嵌入 Android 应用(以下简称 App)中的第三方广告类库可能会收集用户的一些隐私信息,以了解用户的喜好,提供更具针对性的广告[5-6].然而,不管 Android 应用是否恶意,隐私泄露都是当前 Android 手机用户面临的最主要安全威胁之一[4],对 Android 隐私泄露的检测依然是当前移动安全领域研究的热点.

对于隐私泄露的检测,目前最主要的方法是污点分析,包括静态污点分析和动态污点分析[7].动态污点分析方法是指在 Android 应用动态执行的过程中对手机中的隐私数据进行追踪,主要的动态污点分析工具有 TaintDroid[8], AppFence[9], KynoidA[10]等.静态污点分析是指在不运行 Android 应用的情况下直接采用静态分析的方法对其进行污点分析,比较典型的静态污点分析工具有 FlowDroid[11], LeakMiner[12], AndroidLeaks[13]等.使用动态污点分析的方法检测隐私泄露的优点是隐私数据追踪准确、误报率低,缺点是代码覆盖率低,对于实际运行中未被触发的隐私泄露,使用动态污点分析的方式将无法检测,因此,其漏报率较高.而静态污点分析则刚好相反,静态污点分析的缺点是代码覆盖率高,可以检测出比动态污点分析更多的隐私泄露问题;然而,其误报率也较高.因此,在实际的隐私泄露检测中,静态污点分析和动态污点分析方法是互补的.

就静态污点分析而言,当前静态污点分析工具面临的一个重要问题就是不能对 Android 动态加载和反射机制进行有效污点分析.而当前 Android 动态加载与反射机制被越来越广泛地应用在 Android 开发中,文献[14]对 Google Play 上的 14 283 个 App 进行统计发现,在这些 App 中有 18.5% 的 App 使用了动态加载技术,88% 的 App 使用了反射调用机制.文献[15]也指出,从 2010—2014 年,使用反射机制的 Android 恶意应用的比例从43.87%上升到 78%,非恶意应用也从 39.55% 上升到 93%.动态加载和反射调用技术的大量应用使当前的静态污点分析工具越来越难以对 Android 应用的污点分析问题进行有效检测[16].虽然动态污点分析工具作为静态污点分析工具的一个重要补充,可以在不受动态加载和反射机制影响的情况下对 Android 应用进行污点分析,但由于它在代码覆盖率上的劣势,使其无法取代静态污点分析工具在 Android 污点分析中的作用.

鉴于动态加载和反射机制被广泛使用的现状,为了充分发挥静态污点分析工具在 Android 隐私泄露检测中的作用,就必须采取措施解决静态污点分析工具处理 Android 动态加载和反射机制的缺陷.本文以此为出发点,对当前领先的静态污点分析工具 FlowDroid[11]进行改进,开发出能够对 Android 动态加载和反射机制进行有效污点分析的工具 DyLoadDroid,首先将被测 App 实际运行中被加载的 dex 文件下载到本地计算机,并将系统中所发生的反射调用信息记录下来;然后在静态分析的过程中,根据这些反射调用信息将相应的反射调用语句替换为非反射调用语句,再运行 FlowDroid 的静态污点分析流程,使其能够对使用 Android 动态加载和反射调用机制的 Android 应用进行有效的污点分析.

本文的主要贡献包括:

1) 首次提出了使用非反射机制代替反射机制的解决方案,解决了静态污点分析技术不能正确处理

Android 反射机制的问题.

2) 对当前领先的静态污点分析工具 FlowDroid 进行了改进,开发出能够对 Android 动态加载和反射机制进行有效污点分析的工具 DyLoadDroid,并通过实验验证了其有效性.

3) 改写了 Android 源码,使 Android 系统能够在系统中发生动态加载和反射调用行为时通过日志输出的形式输出相关信息,DyLoadDroid 便可以根据这些信息及时下载被加载的 dex 文件,并提取反射调用信息,从而指引 DyLoadDroid 的静态污点分析模块对 Android 应用进行污点分析.

1 相关工作

在 Android 应用动态加载和反射机制的研究方面, Poeplau 等人[17]对 Android 动态加载执行代码的安全问题进行了研究,研究了恶意应用利用动态加载机制绕过安全检查的问题,并对非恶意应用动态加载执行代码可能导致的安全漏洞进行了研究. Zhauniarovich 等人[14]提出了 StaDynA 工具,能够在 Android 应用运行过程中发生动态加载时,下载被动态加载的 dex 文件,并记录 Android 应用的反射调用信息,结合 AndroGuard 生成函数调用关系图. 然而 StaDynA 是一个比较初步的工具,只能为分析者提供一种辅助,不能对 Android 应用中存在的安全问题进行自动化分析.

在 Android 静态污点分析方面, Yang 等人[12]提出了 LeakMiner 工具, LeakMiner 将 Android 的生命周期考虑在内,构建能够覆盖所有回调方法的函数调用图;但 LeakMiner 对上下文不敏感,且不支持隐式信息流泄露,因此其污点分析准确率不高. Gibler 等人[13]提出的 AndroidLeaks 工具具有上下文敏感的特性,但它不具有域敏感和对象敏感的特性,即当对象的某一个域存储了污点数据,则 AndroidLeaks 将整个对象视为污点,因此其检测准确性也不高. FlowDroid 是当前较为先进的 Android 静态污点分析工具,由 Arzt 等人[11]在 PLDI 2014 会议上提出,FlowDroid 能够构建准确的 Android 生命周期模型,并对 Android 的回调函数进行有效的处理. 它使用 IFDS 算法进行准确和高效的污点传播分析,具有上下文敏感、流敏感、域敏感和对象敏感的特性. 为了解决 FlowDroid 在组件间数据流处理问题上的缺陷,Octeau 等人[18]在 FlowDroid 的基础上,提出了组件间污点分析工具 Epicc,能够对组件间的数据传递进行较为准确的污点分析. 为了使 FlowDroid 能够在多个应用之间进行污点传播分析,Klieber 等人[19]在 FlowDroid 和 Epicc 的基础上提出 DidFail 工具,能够正确处理多个应用之间的相互调用和数据传输,进而追踪污点的传播.

在 Android 动态污点分析方面, TaintDroid[8]将所有的隐私数据标记为污点,它在内存中为系统中的每个对象多申请一倍的空间,用于存储污点标记,在 App 动态执行的过程中,根据污点标记追踪污点在内存中的传播情况. Hornyack 等人[9]提出的 AppFence 工具不仅可以检测到系统中发生的隐私泄露问题,而且可以阻止隐私信息的泄露. 它采取数据遮蔽和渗出阻塞技术,将隐私数据替换为影子数据传递到外部网络来防止隐私数据的泄露,但其严格的安全控制也容易阻碍 App 的正常运行. Schreckling 等人[10]提出的 Kynoid 工具是对 TaintDroid 的改进,能够监控更细粒度的污染源,并能够使用户以安全策略的方式定义污点传播内容,然而, Kynoid 运行时的内存消耗比 TaintDroid 更大,速度也更慢.

2 背景介绍

Android 动态加载技术是指 Android 应用在动态执行的过程中加载主应用以外的 dex 文件,并对该文件中的程序和资源进行访问的技术[14],被加载的 dex 文件包括“.apk”和“.jar”2 种后缀的文件形式,该 dex 文件可以在没有被安装的情况下被其他 Android 应用通过反射机制访问.

反射机制是 Java 语言的一个重要特性,它使程序在 Java 的运行状态中,对于任意一个类,都能够知道这个类的所有属性和方法;对于任意一个对象,都能够调用它的任意一个方法和属性,这种动态获取信息以及动态调用对象的方法的功能称为 Java 语言的反射机制[20]. Android 应用基于 Java 语言开发,因此可以充分利用 Java 的反射机制. 下面我们通过一个例子展示 Android 的动态加载和反射调用机制.

图 1 所示为在 Android 反射调用中充当调用者的例子,该段代码在第⑥行动态加载了位于 SD 卡中的 apk 文件 DynamicLoadClient.apk;第⑦行加载了该 apk 文件中的类 com.dynamicloadclient.SendMessage;在第⑨行对该类进行了实例化,并通过该类的构造方法传入参数“MessageMark”;第⑩行取出该类的名为 *sendMSG* 的方法;第⑪行对该方法

进行了反射调用,传入参数"123456"和 *deviceId*. 图 2 所示的代码展示了被调用者代码,该段代码的功能是向手机号为 *number* 的移动设备发送内容为 *content* 的短信信息. 结合调用者和被调用者的代码我们可以看出,调用者通过反射调用向手机号为"123456"的设备发送了内容为设备的 IMEI 号的短信.

```
① TelephonyManager tm=(TelephonyManager)this.
  getSystemService(Context.TELEPHONY_SERVICE);
② String deviceId=tm.getDeviceId();
③ try{
④ File file=new File("/sdcard/DynamicLoadClient.apk");
⑤ if (file.exists()){
⑥   DexClassLoader cl=new DexClassLoader(file.toString
    (),getFilesDir().getAbsolutePath(),null,ClassLoader.
    getSystemClassLoader().getParent());
⑦   Class myClass=cl.loadClass("com.dynamicloadclient.
    SendMessage");
⑧   Constructor constructor=myClass.getConstructor(new
    Class[]{String.class});
⑨   Object obj=constructor.newInstance(new Object[]{"
    MessageMark"});
⑩   Method action=myClass.getMethod("sendMSG",new
    Class[]{String.class,String.class});
⑪   String return_value=(String) action.invoke(obj,"
    123456",deviceId);
⑫   Log.i("return_value","return_value:"+return_value);
⑬   }} catch(Exception ex){ex.printStackTrace();}
```

Fig. 1 The caller in Java reflection invocation

图 1 Java 反射调用中的调用者

```
① public String sendMSG(String number,String content) {
②   SmsManager sManager=SmsManager.getDefault();
③   sManager.sendTextMessage(number,null,content,null,
    null);}
④   Log.i(markStr,"A message has been sent to "+number
    +",content:"+content);
⑤   return "OK"; }
```

Fig. 2 The callee in Java reflection invocation

图 2 Java 反射调用中的被调用者

图 1 和图 2 所示的例子演示了一个典型的隐私泄露问题,设备的 IMEI 号被以短信的形式发送给指定手机. 隐私泄露问题也是当前污点分析重点关注的问题,然而,当前的静态污点分析技术却难以对存在动态加载和反射机制的程序进行正确的处理,因而不能进行有效的静态污点分析. 例如,若将该例中的设备 IMEI 号作为污染源(source),将发送短信的 API 调用作为陷入点(sink),则静态污点分析不能构建从污染源到陷入点的有效路径,因而不能检测到这个隐私泄露问题. 原因是静态分析不能确定通过 *newInstance* 方法实例化的对象和通过 *invoke* 方法反射调用的目标方法. 在该例子中,被反射调用的方法存在于被加载的 dex 文件中,实际上 App 也可以通过反射调用的方式调用 Android 系统中任何可达的 Java 方法,包括 App 本身包含的或 Android 系统类库中包含的 Java 方法,这些反射调用都不能被当前的静态污点分析工具正确的处理.

3 总体设计

本节开始将系统地介绍我们对 FlowDroid 进行改进后的工具——DyLoadDroid.

DyLoadDroid 的总体结构和流程图如图 3 所示,为了方便下文描述,我们做出如下名词定义.

1) 源方法. 调用者所在的方法,用 S_m 表示,如图 1 中所示代码所在的 Java 方法.

2) 源类. 源方法所在的类,用 S_c 表示.

3) 目标方法. 被调用者所在的方法,用 T_m 表示,如图 2 所示代码所在的 Java 方法.

4) 目标类. 被调用者所在的类,用 T_c 表示.

5) 反射方法. 调用者实施反射机制所借助的方法,包括 *newInstance* 和 *invoke* 方法,用 R_m 表示,如图 1 所示代码第⑨行被调用的 *newInstance* 方法和第⑪行被调用的 *invoke* 方法.

6) 反射类. 反射方法所在的类,用 R_c 表示.

DyLoadDroid 主要由 2 部分组成:动态执行模块和静态分析模块,为了便于下文的描述,我们把它们分别简称为 DyLoadDroid-D 和 DyLoadDroid-S. DyLoadDroid 首先运行 DyLoadDroid-D,将待测 App 安装进 Android 设备中,该 Android 设备中安装的 Android 系统是由我们对 Android 源码进行特定修改并重新编译后生成的,它能够在日志中输出系统中发生的动态加载和反射调用的相关信息,针对 Android 源码所做的修改我们将在 4.1 节中介绍. 安装完该 App 后,DyLoadDroid-D 运行该 App 文件,并循环读取系统的日志输出,获取 App 的动态加载和反射调用的相关信息. 当捕获到该 App 的动态加载行为时,DyLoadDroid-D 便向 Android 设备发出 adb 下载命令,将被加载的 dex 文件下载到本地计算机. 当捕获到该 App 的反射调用行为时,DyLoadDroid-D 将从日志中提取该反射调用所对应的源方法 S_m、反射方法 R_m 和目标方法 T_m 等信息,并将这些信息以三元组$\langle S_m, R_m, T_m\rangle$的形式存储在

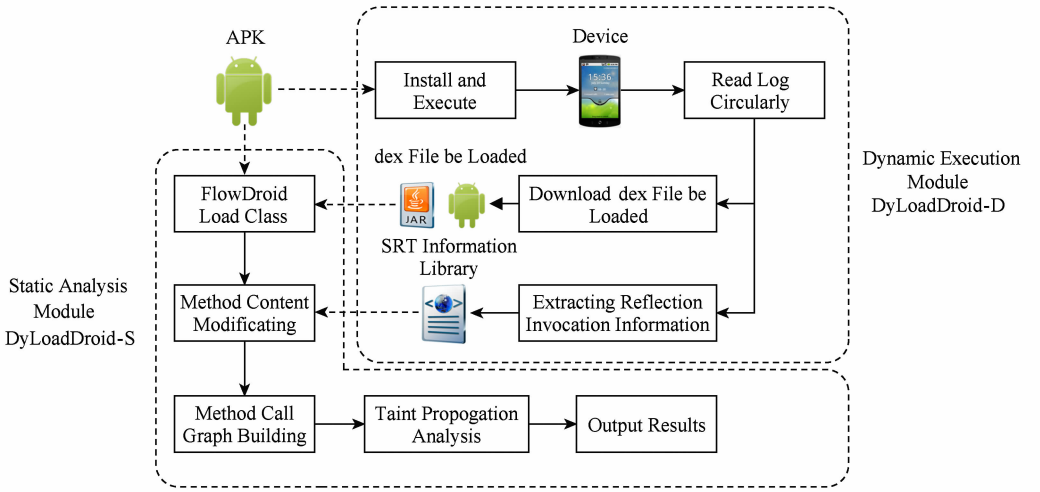


Fig. 3 Overall flow chart of DyLoadDroid

图 3 DyLoadDroid 总体流程图

本地计算机的文件中,我们称该文件为 SRT 信息库.这些被下载的 dex 文件和 SRT 信息库将用于后续的静态污点分析.

当动态分析模块执行结束以后,DyLoadDroid 运行 DyLoadDroid-S,DyLoadDroid-S 实际上是对 FlowDroid 的改进,我们将动态分析模块输出的 dex 文件和 SRT 信息库补充进 FlowDroid 的静态污点分析流程,弥补 FlowDroid 不能处理动态加载和反射调用的缺陷.FlowDroid 首先将待测 APK 和被加载的 dex 文件中的一些必需的 Java Class 文件加载进内存,并转换为 Soot[21] 的三地址中间语言 Jimple;然后 FlowDroid 根据 SRT 信息库对源方法的相应反射方法调用进行转换,使 FlowDroid 能够对反射方法调用构建正确的函数调用图,指引 FlowDroid 进行静态污点分析,最终输出结果.

4 DyLoadDroid-D 原理介绍

4.1 Android 源码修改

我们针对 Android 源码的修改参考了文献[14]的方法,所修改的 Android 版本为 4.1.2,主要修改包括:

1) Android 系统方法调用堆栈所存储元素的修改.Android 系统方法调用堆栈存储着系统中发生的方法调用序列.源码中方法调用堆栈只存储类名和方法名,而不存储方法的参数.我们修改了 Android 源码,使其能够存储方法的参数,这样便可以通过方法调用堆栈获得方法的完整信息.

2) 增加动态加载的日志输出.为了监控 Android 应用的动态加载行为,我们对 Android 源码的 DexFile.java 文件的 *openDexFile* 方法进行了相应的修改,使当系统发生动态加载行为后,通过日志输出被加载的 dex 文件的路径,以便于 DyLoadDroid-D 能够及时下载被加载的 dex 文件.

3) 增加 *newInstance* 方法调用的日志输出.为了监控 Android 应用通过 *newInstance* 方法进行对象实例化的行为,我们对 Android 源码的 Constructor.java 和 Method.java 文件的 *newInstance* 方法进行了修改,使系统中发生了对 *newInstance* 方法的调用时,输出被实例化的目标类的类名、初始化方法名和初始化参数.

4) 增加 *invoke* 方法调用的日志输出.为了监控 Android 应用通过 *invoke* 方法进行反射调用的行为,我们对 Method.java 方法中的 *invoke* 方法进行了修改,使系统发生了 *invoke* 方法调用时,输出目标类的类名、目标方法的方法名和参数.

以上对源码的修改的 2~4 这 3 项都对应日志输出,除了其中所描述的日志输出以外,还需要额外输出日志标识、uid 号、操作号、调用堆栈信息等.日志标识用于标识哪些日志信息是需要被 DyLoadDroid-D 解析的;uid 号使 DyLoadDroid-D 能够辨别哪些 log 输出是由待测 App 产生的;操作号用于标识所监控的程序行为的类型,根据被调用的方法的不同,我们设置 *newInstance*,*invoke*,*openDexFile* 方法所对应的操作号分别为 1,2,3.此外,对于修改 3,4 这 2 项的日志输出,我们还额外输出当前线程在 Android 方法调用堆栈中的信息,这些信息将辅助 DyLoadDroid-D 获得源方法信息和反射方法信息,

具体内容我们将在 4.2 节介绍.

4.2 日志解析

DyLoadDroid-D 安装并运行 App 后会取出该 App 所对应的 uid 号,App 的运行需要实验者对其进行实际使用,实验者在使用 App 的过程中触发尽可能多的动态加载和反射调用行为. DyLoadDroid-D 会循环读取手机的日志信息,提取 uid 号为待测 App 的日志记录. 当读取到动态加载 dex 文件的记录时,DyLoadDroid-D 会根据日志中记录的文件信息将该 dex 文件下载到本地电脑的指定文件夹,用于为 DyLoadDroid-S 的静态污点分析提供类加载源. 当捕获到 *newInstance* 方法的输出信息或 *invoke* 方法输出的反射调用信息时,DyLoadDroid 会从中提取出对应的源方法 S_m、反射方法 R_m 和目标方法 T_m 的信息,并将这些信息以三元组$\langle S_m, R_m, T_m\rangle$的形式存入 SRT 信息库,用于为 DyLoadDroid-S 的静态污点分析过程提供反射调用信息,图 4 描述了该信息的提取原理.

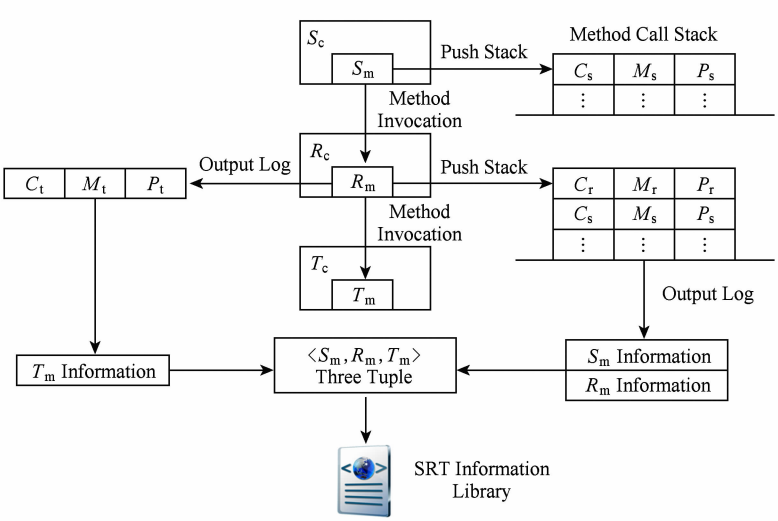


Fig. 4 Schematic diagram of log analysis

图 4 日志解析原理图

我们以 C_x,M_x,P_x 分别表示类名、方法名和方法的参数名,x 取 s,r,t 分别表示源、反射和目标,例如 C_s 表示源类的类名,依此类推. 根据 4.1 节中关于 Android 源码的修改,当待测 Android 系统中发生 *newInstance* 或 *invoke* 反射方法调用时,我们可以从日志中直接获取到 T_m 的信息 C_t,M_t,P_t. 对于 S_m 和 R_m 的信息,我们可以通过当前线程的 Android 方法调用堆栈获得. Android 方法调用堆栈存储着从当前方法调用开始,按时间顺序排列的方法调用序列,当前正在调用的方法信息存储在堆栈的栈顶,距离当前方法调用最近的一次方法调用存储在堆栈的第 2 个单元. 因此,若发生反射方法调用时,处在堆栈最顶部的方法通常是反射方法 R_m 的信息,而处在堆栈中第 2 位的便是进行反射调用的源方法 S_m 的信息. 根据 4.1 节中对 Android 系统方法调用堆栈所存储元素的修改,我们便可以在反射方法调用发生时,通过输出的堆栈信息,获得 S_m 的信息 C_s,M_s,P_s 和 R_m 的信息 C_r,M_r,P_r. DyLoadDroid-D 会将解析产生的 S_m,R_m,T_m 以三元组的方式存储在 SRT 信息库中.

5 DyLoadDroid-S 原理介绍

DyLoadDroid-D 的结束由人工进行控制,当实验者充分使用 App 后,可以对 DyLoadDroid 输入命令,终结 DyLoadDroid-D 的运行,并启动 DyLoadDroid-S 对 App 进行静态污点分析. DyLoadDroid-S 实际上是一个改进版的 FlowDroid,它将 DyLoadDroid-D 输出的 dex 文件和 SRT 信息库补充进 FlowDroid 的分析流程,使 FlowDroid 能够正确处理 App 的动态加载和反射机制的程序代码,进行更全面的污点分析.

5.1 类的加载

FlowDroid 的静态分析方法建立在 Soot 的 Jimple 语言基础上,FlowDroid 会将 apk 文件中包含的所有类加载进内存,然后构建主方法,并构建函数调用图. 为了减小内存的负担,DyLoadDroid-S 采取按需加载的方式对被加载的 dex 文件中相应的类进行加载. 即 DyLoadDroid-S 只取 SRT 信息库中的源类和目标类进行加载,在构建函数调用图时,根据函数调用图的扩展,Soot 会自动按需加载额外需要

的类.目标方法既可能存在于被加载的 dex 文件中,也可能存在于 App 本身具有的类中或 Android 系统库 android.jar 中,这些可能存在的文件都将作为 Soot 的基类.

5.2 模糊匹配原则和条件转移原则

DyLoadDroid-S 对源方法的修改实际上是将源方法中相应的反射方法调用修改为 FlowDroid 能够处理的目标方法调用,使 FlowDroid 的污点分析模块能够对反射方法调用进行正确的处理.在介绍源方法修改算法之前,我们先介绍源方法修改的 2 个原则——模糊匹配原则和条件转移原则.

模糊匹配原则是指在对源方法进行修改时,DyLoadDroid-S 根据 SRT 信息库,搜索源方法中的反射方法调用,并根据反射方法的参数个数、参数类型、返回值类型近似确定反射方法调用的位置.例如,图 5 是图 2 所示源码的第⑨行和第⑪行代码对应的 Jimple 代码,DyLoadDroid-S 从 SRT 信息库中取出该段代码所在源方法 S_m 对应的反射调用信息三元组$\langle S_m, R_m, T_m\rangle$,若 R_m 为 *newInstance* 方法,且 T_m 具有的参数个数、参数类型与图 5 第③行的 $r14 数组元素个数和元素类型对应,DyLoadDroid-S 便认为图 5 第③行的反射调用语句对应于这个反射调用信息三元组$\langle S_m, R_m, T_m\rangle$.同理,若 R_m 为 *invoke* 方法,且 T_m 具有的参数个数、参数类型与图 5 第⑧行的 $r14 数组元素个数和元素类型对应,且 T_m 的返回值类型与 $r15 对应,DyLoadDroid-S 便认为图 5 第⑧行的反射调用语句对应于反射调用信息三元组$\langle S_m, R_m, T_m\rangle$.

```
① $r14 = newarray (java.lang.Object)[1];
② $r14[0] = "MessageMark";
③ $r5 = virtualinvoke $r13.<java.lang.reflect.Constructor: java.lang.Object newInstance(java.lang.Object[])>($r14);
④ ⋮
⑤ $r14 = newarray (java.lang.Object)[2];
⑥ $r14[0] = "123456";
⑦ $r14[1] = $r7;
⑧ $r5 = virtualinvoke $r15.<java.lang.reflect.Method: java.lang.Object invoke(java.lang.Object, java.lang.Object[])>($r5, $r14);
```

Fig. 5 The Jimple code corresponding to *newInstance* and *invoke*

图 5 *newInstance* 和 *invoke* 调用对应的 Jimple 代码

条件转移原则是根据 FlowDroid 的污点分析特性设置的,由于 FlowDroid 不具有符号执行的功能,因此对于“if…else…”的处理不会根据判断标识的具体取值选择程序流程,因此,DyLoadDroid-S 对反射方法的修改采取增加“if…else…”语法的方法,首先设置一个判断标识,当标识为某一数值时,DyLoadDroid 根据 SRT 信息库构造目标方法的调用,否则执行原始反射方法调用.之所以构建“if…else…”而不是直接替换反射方法,主要原因有 2 点:

1) 防止同一源方法通过同一反射方法调用多种目标方法,这样,DyLoadDroid-S 可以方便地在反射方法调用上进行“if…else…”的嵌套,而不至于使 FlowDroid 发生漏报.

2) 防止模糊匹配的不准确性导致的污点分析错误,根据 FlowDroid 对“if…else…”不选择执行的特性,若发生匹配错误,FlowDroid 也至少可以在原程序方向上正确地运行.此外,若同一源方法出现多个匹配,即对于某一三元组$\langle S_m, R_m, T_m\rangle$,在 S_m 中存在多个反射调用与该三元组匹配,则“if…else…”原则能够保证至少有一个匹配是正确的,而其他错误的匹配不会对 FlowDroid 的污点传播结果造成影响.

5.3 源方法修改算法

为了方便算法的描述,我们补充如下定义.

1) 目标反射对象.使用反射机制对目标类进行实例化的对象,用 F_o 表示,如图 1 所示代码第⑨行的 *obj* 对象.

2) 目标对象.直接对目标类进行实例化后的对象,用 T_o 表示.

3) 目标反射参数.使用反射机制向目标方法传递的参数,用 F_p 表示,如图 5 所示代码第③行和第⑧行对应的 $r14 数组.

4) 目标参数.向目标方法传递的实际参数,用 T_p 表示,如图 5 中第②行传给目标类构造方法的参数“MessageMark”和第⑥,⑦行传给目标方法的参数“123456”和 $r7.

5) 反射对象.反射类被实例化后的对象,用 R_o 表示,如图 1 所示代码第⑨行的 *constructor* 对象和第⑪行的 *action* 对象.

具体的源方法修改算法的算法流程图如图 6 所示,对于一条 SRT 映射,DyLoadDroid-S 首先判断反射方法为 *newInstance* 还是 *invoke* 方法.对于 *invoke* 方法,DyLoadDroid-S 首先将目标反射参数 F_p 转化为目标参数 T_p;然后判断目标反射对象是否为空,若为空,则说明目标方法为静态方法,构建静态目标方法调用;若不为空,则说明目标方法为非静态方法,

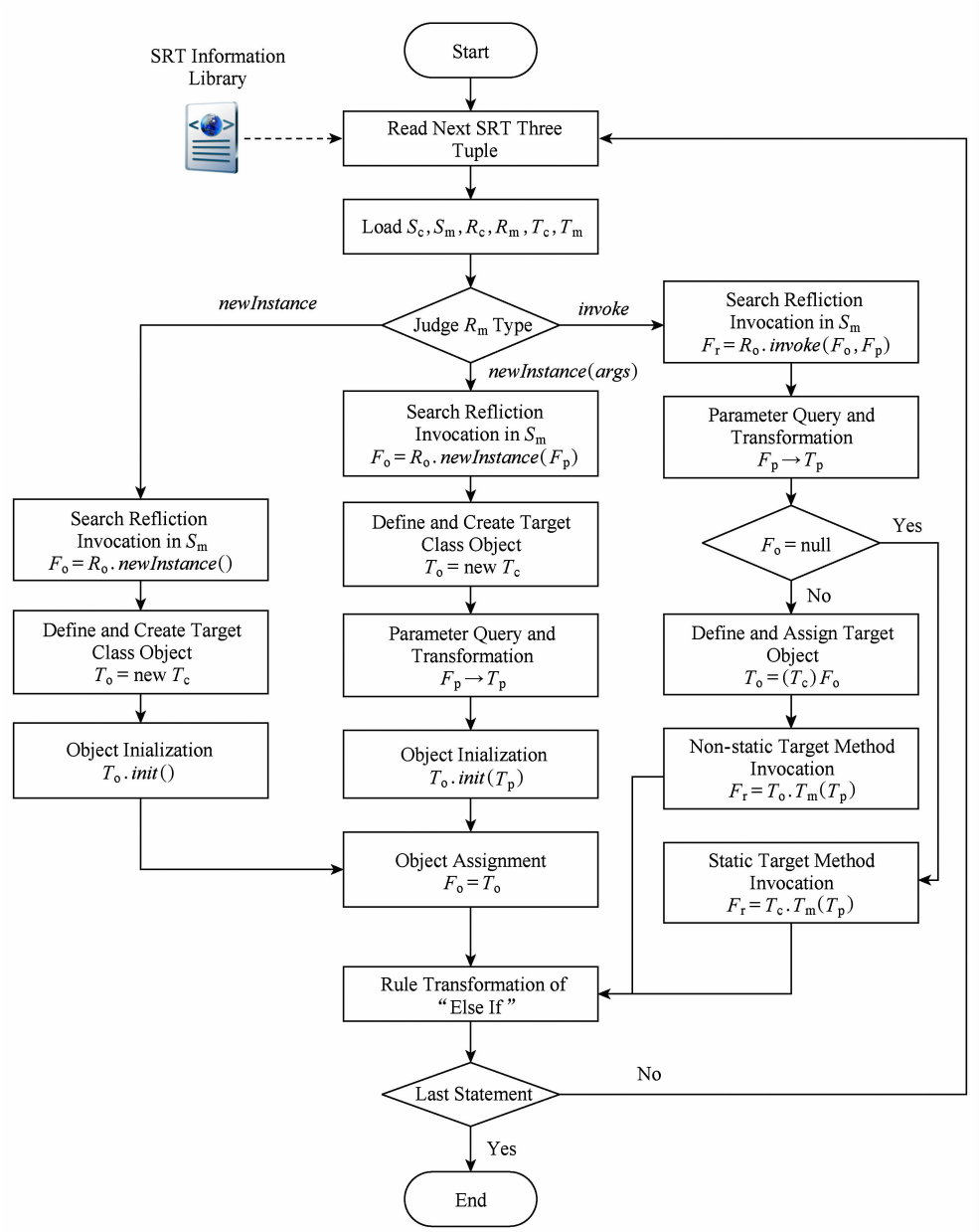


Fig. 6 Algorithm flow chart of source method modifying

图 6 源方法修改算法流程图

DyLoadDroid-S 会定义目标类类型的对象 T_o，并将目标反射对象 F_o 强制类型转换并赋值给 T_o，然后执行非静态方法调用.

newInstance 方法包括带参数和不带参数 2 种，对于带参数的 *newInstance* 方法，DyLoadDroid-S 定义并创建一个目标对象 T_o，并将目标反射参数 F_p 转化为目标参数 T_p，并调用目标方法 T_m（只会是 *init*()）对其进行初始化，然后 DyLoadDroid-S 将该 T_o 对象重新赋值给原目标反射对象 F_o，以保证 F_o 在程序中向下正常传递.

图 7 和图 8 所示分别为对反射方法为有参 *newInstance* 方法和反射方法为 *invoke* 方法且目标方法为非静态方法的反射调用进行源方法修改后的结果，其中左边的 S_c 和 S_m 代表修改之前的源类和源方法，右边的 S_c 和 S_m 代表修改之后的源类和源方法. 图 9 所示为对图 5 中的 Jimple 代码进行源方法修改后的结果.

5.4 污点传播分析

在源方法修改之前，由于 FlowDroid 不能正确处理通过 *newInstance* 方法调用进行的对象实例化和通过 *invoke* 方法调用进行的反射调用，因此，在进行污点分析时，FlowDroid 不能正确地分析出污点在源方法和目标方法之间的传播. 如图 7 和图 8 所示，修改前 FlowDroid 无法将 S_m 与 T_c，T_m 建立关系.

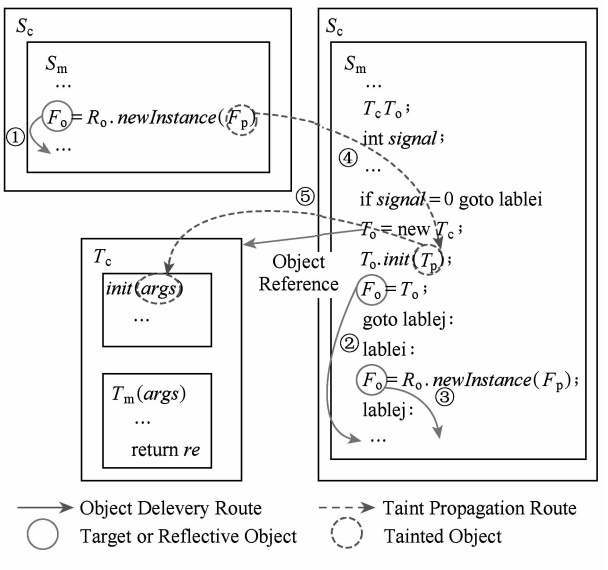


Fig. 7 Taint propagation analysis for *newInstance*

图 7 *newInstance* 方法的污点传播分析

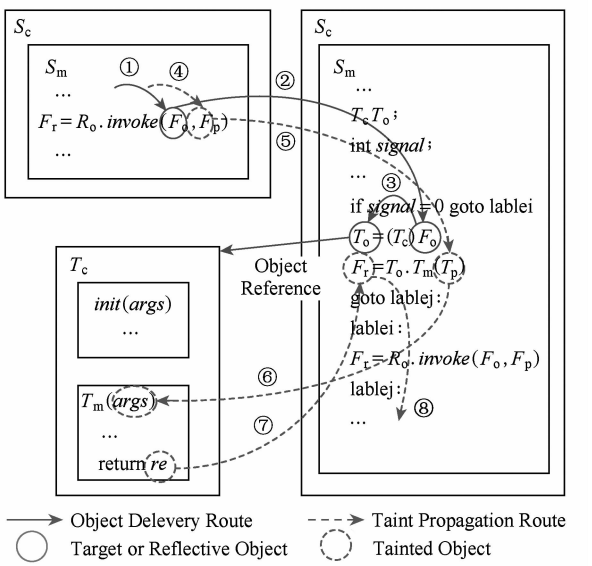


Fig. 8 Taint propagation analysis for *invoke*

图 8 *invoke* 方法的污点传播分析

在源方法修改之后,对于 *newInstance* 方法调用,DyLoadDroid-S 使用 new 关键字直接创建目标对象,并且将目标反射参数转化为目标参数,所生成的目标对象 T_o 又赋值给目标反射对象 F_o,F_o 能够被 FloDroid 正确处理,并在源方法中向下传递. 图 7 和图 8 中的实线椭圆和箭头分别代表目标(反射)对象和目标(反射)对象的传递路线,从中可以看出,图 7 中源方法修改后新增了一条目标反射对象 F_o 的向下传递路线,并且这个新增的路线能够被 FlowDroid 正确处理. 对于 *invoke* 方法调用,源方法修改过后的结果是,在源方法 S_m 和目标方法 T_m 之间建立了能够被 FlowDroid 处理的调用关系,S_m 向 T_m 传递的参数和接收的 T_m 的返回结果能够被 FlowDroid 正确追踪. 目标对象的传递也具有连续性,例如图 8 中的目标反射对象 F_o 被强制类型转换后赋值给了目标对象 T_o,接下来,对 T_o 的目标方法 T_m 进行了调用.

```
① $r14 = newarray (java.lang.Object)[1];
② $r14[0] = "SendMessage";
③ $i1 = 0;
④ if $i1 == 0 goto label08;
⑤ $r18 = new com.dynamicloadclient.SendMessage;
⑥ specialinvoke $r18.<com.dynamicloadclient.SendMessage:
   void <init>(java.lang.String)>("SendMessage");
⑦ $r5 = $r18;
⑧ goto label09;
⑨ label08:
⑩ $r5 = virtualinvoke $r13.<java.lang.reflect.Constructor:
   java.lang.Object newInstance(java.lang.Object[])>($r14);
⑪ label09:
⑫ ...
⑬ $r14 = newarray (java.lang.Object)[2];
⑭ $r14[0] = "123456";
⑮ $r14[1] = $r7;
⑯ $i0 = 0;
⑰ if $i0 == 0 goto label14;
⑱ $r17 = (com.dynamicloadclient.SendMessage) $r5;
⑲ $r5 = virtualinvoke $r17.<com.dynamicloadclient.
   SendMessage:java.lang.String sendMSG(java.lang.String,
   java.lang.String)>("123456", $r7);
⑳ goto label15;
㉑ label14:
㉒ $r5 = virtualinvoke $r15.<java.lang.reflect.Method:java.
   lang.Object invoke(java.lang.Object,java.lang.Object[])>
   ($r5, $r14);
㉓ label15:
```

Fig. 9 The Jimple code after the transformation of *newInstance* and *invoke*

图 9 *newInstance* 和 *invoke* 被转换后的 Jimple 代码

为了便于分析,我们不妨假设图 7 中向下传递的目标反射对象 F_o 最终流向图 8 中的 F_o,那么从图 7 到图 8,DyLoadDroid-S 便形成了一条能被 FlowDroid 正确处理的目标对象传递路径,即图 7 的②到图 8 的①②③.

图 7 和图 8 中的虚线椭圆和箭头代表可能的污点传播线路,在图 7 中,若目标反射参数 F_p 携带有

污点数据,则会沿着④转换为目标参数 T_p,并在对象初始化时传给 T_c 的初始化方法 *init*(). 同理,在图 8 中,若目标反射参数 F_p 携带有污点数据,则会沿着路径④⑤⑥进行污点传播. 若 T_m 的返回值 *re* 携带有污点数据,则会沿着路径⑦传回源方法 S_m,S_m 中接收返回值的 F_r 对象将在 S_m 中向下传播污点数据.

6 实验与分析

本节我们将通过实验验证 DyLoadDroid 对使用 Android 动态加载和反射机制的 App 进行污点分析的有效性,并与 FlowDroid 的检测效果进行对比,同时对一些典型 App 进行实例分析,最后,将 DyLoadDroid 与其他动态污点分析工具进行定性的对比.

6.1 实验环境和实验样本

实验的主要硬件设备为一个三星 I9300 手机(4 核处理器)和一台 ThinkCentre 计算机(8 核 CPU、8 GB 内存),我们将 4.1 节介绍的修改后的 Android 4.1.2 系统制作成 ROM 并刷入三星 I9300 手机,DyLoadDroid-D 所测试的所有 App 将在这个修改后的 Android 系统中运行. 而 ThinkCentre 计算机则主要用来运行 DyLoadDroid 主程序.

实验样本包括:1)从 Android 应用市场豌豆荚[22]下载的 11 428 个 App,我们把它们作为非恶意样本(虽然可能有少数恶意 App,但对实验结果不会造成大的影响);2)从恶意应用共享网站 VirusShare[23]下载的 1 132 个 App,我们将它们作为恶意样本. 实验样本的下载时间均为 2015 年 6 月,我们使用静态分析的方法对它们使用动态加载和反射机制的情况进行统计,统计结果如表 1 所示:

Table 1 Number Statistics of Dynamic Loading and Reflection Invocation in Experiment Samples

表 1 实验样本中动态加载与反射方法调用的数目统计

Type	Total	With Dynamic Loading Mechanism		With Method "*newInstance*"		With Method "*invoke*"	
		App Number	Proportion	App Number	Proportion	App Number	Proportion
Non-malicious App	11 428	3 760	0.329 0	8 715	0.762 6	9 123	0.798 3
Malicious App	1 132	180	0.159 0	493	0.435 5	521	0.460 2

从表 1 可以看出,动态加载和反射机制在 Android 应用中是普遍存在的,并且在非恶意应用中存在的比例高于恶意应用,这可能是由于恶意应用在功能上通常没有非恶意应用丰富,代码量小于非恶意应用.

6.2 实验结果

我们根据 6.1 节中对样本 App 使用动态加载和反射机制情况的统计,选取了动态加载和反射方法调用数量较多的 App 进行分析,表 2 所示的 10 个 App 为其中比较有代表性的 Android 应用,其中

Table 2 Information of Dynamic Loading and Reflection Invocation in 10 Samples

表 2 10 个样本的动态加载与反射方法调用情况

App ID	App Package Name	Loaded dex Files	Method "*newInstance*"		Method "*invoke*"	
			Total	Triggered	Total	Triggered
1	com. jtb. gosms	3	29	18	528	356
2	com. edaxi. acmove	2	40	27	161	102
3	com. nyd. deskcontacts	5	67	45	139	98
4	com. yibajie. weather	4	13	10	125	86
5	com. shantin. dial	3	46	33	108	70
6	it. medieval. blueftp	3	8	8	85	71
7	com. qLw. uzoFvDQB	1	13	10	67	45
8	njpdd. pkrk. dhqa	6	9	7	34	21
9	tk. bohe. Lock	0	100	68	460	296
10	com. adr. yykbplayer	3	33	26	146	115

App 编号为 1～5 的 App 为非恶意样本,编号为 6～10 的 App 为恶意样本,表 2 对 App 运行中下载的 dex 文件数、存在的 *newInstance* 反射方法调用数和运行中被触发的反射方法调用数、存在的 *invoke* 反射方法调用数和运行中被触发调用的反射方法调用数进行了统计. 由于实验中对 App 的具体操作依赖于实验者,因此 DyLoadDroid 并不能保证反射方法被触发的概率为 100%.

我们分别使用 DyLoadDroid 和 FlowDroid 对这 10 个 App 进行了污点分析,并对它们的检测效果进行了对比,分析结果如表 3 所示(表 3 中的 App 编号与表 2 的 App 编号对应). 我们依据文献[24]中的方法选择污染源(source)和陷入点(sink). 实验中的污染源即手机中的隐私数据,我们主要考虑的隐私数据类型包括 10 种:1)手机标识,包括 IMEI,IMSI,ICCID,MSISDN,ANDROID_ID 等;2)地理位置信息;3)短信记录;4)通讯录;5)通话记录;6)Google 账号;7)wifi 信息;8)蓝牙信息;9)基站信息,如基站编号、位置区域码等;10)浏览器信息,如浏览器书签、浏览历史记录等. 实验中的陷入点即可能的隐私泄露途径,我们主要考虑 4 种隐私泄露途径:1)网络;2)短信;3)日志;4)文件. 它们分别表示通过网络或短信发送隐私信息,或通过日志输出或写文件的方式将隐私信息暴露在系统中. 表 3 中的中括号中的内容代表 FlowDroid 和 DyLoadDroid 共同检测出的隐私泄露内容或共同检测出的隐私泄露路径,表 3 中的大括号中的内容代表 DyLoadDroid 检测出、但 FlowDroid 未检测出的隐私泄露内容或隐私泄露途径,从另一角度来看,大括号中的内容实际上是 App 通过动态加载和反射机制所泄露的隐私内容.

Table 3 Results Comparison of Taint Analysis

表 3 污点分析结果对比

App ID	FlowDroid			DyLoadDroid			Content of Privacy	Ways of Privacy Leakage
	Source Number	Sink Number	Taint Propogation Path Number	Source Number	Sink Number	Taint Propogation Path Number		
1	36	53	0	45	79	5	{1,2}	{①}
2	108	75	16	112	10	20	[1,2],{6,7}	[①,④]
3	73	54	5	85	67	19	[1,7],{2,9}	[①]
4	103	88	24	124	108	38	[1,2,7],{8,9}	[①],{③}
5	87	53	14	91	67	22	[1,2],{7,9,10}	[①,③],{④}
6	13	30	4	83	65	15	[1],{2,3,4,9,10}	[①],{②}
7	30	27	10	41	33	18	[1,2],{3,4}	[①],{②,③}
8	0	5	0	15	8	7	{4,5}	{①}
9	13	26	2	54	70	9	[1,2],{2,3,4,6}	[①],{②}
10	9	23	0	20	28	7	{2,4,5}	{①,③}

Note: The numbers in column "Content of Privacy" represent the type of the private data that be leaked: 1—mobile phone identification, 2—geolocation information, 3—message records, 4—contacts, 5—call records, 6—google accounts, 7—wifi information, 8—bluetooth information, 9—base station information, 10—browser information. The marks in column "Ways of Privacy Leakage" represent the ways through which the private data be leaked: ①—network, ②—short message, ③—log, ④—file. The contents in the brackets mean the private data or ways of privacy leakage detected by FlowDroid and DyLoadDroid together, while contents in the braces mean the private data or ways of privacy leakage detected by DyLoadDroid, but not by FlowDroid.

6.3 实验结果分析

从表 3 可以看出,由于 DyLoadDroid 能够对 Android 动态加载和反射机制进行有效的污点分析,因此它能够比 FlowDroid 检测出更多的 source、sink 和污点传播路径数,检测出更多的隐私泄露内容和隐私泄露途径. 并且我们也能从表 3 中看出,非恶意应用通过动态加载和反射调用泄露的用户隐私的恶意性并不大,通常不会泄露短信记录、通讯录、通话记录等高度敏感内容,而恶意应用泄露用户隐私的行为则通常具有较高的恶意性,这可能是由于非恶意应用获取用户隐私信息的目的只是为了提供更好的服务,而并不是为了侵犯用户的隐私,而恶意应用则具有明显的窃取用户隐私的目的. 为了更好地了解它们的动态加载和反射调用行为,我们采用逆向分析的方法对其进行了实例分析.

6.3.1 对非恶意应用的实例分析

通过对非恶意应用的逆向分析,我们发现,一些非恶意应用利用动态加载和反射调用机制实现插件

式管理,例如,App1(表 3 中编号为 1 的 App,以后我们均这样简称)使用大量的反射调用对被加载 dex 文件中组件的生命周期方法进行调用,并通过反射调用,对被加载 dex 文件中的资源进行加载和访问,被加载的 dex 文件实际上是以插件的方式为 App1 提供界面和功能的扩展.此外,一些非恶意应用利用动态加载和反射机制进行动态升级或模块更新,例如,App3 启动运行时会弹出对话框,提醒用户是否更新应用,当点击"是"时,App3 会从网上下载一个名为 component.jar 的 jar 文件,并替换掉应用的数据目录中同名的文件,App3 在运行的过程中会使用反射机制对该 jar 文件进行访问.我们也对非恶意应用获取用户隐私相关信息的原因进行了分析,发现它们主要还是正当的目的,例如,App4 是一款天气预报软件,它获取位置信息是为了向用户提供用户所在地的天气情况.App5 具有展示广告的功能,其内部嵌入了广告类库,该广告类库具有阅读浏览器浏览记录并传给网络服务器的功能,该行为或许是为了根据用户的喜好,提供更具针对性的广告.

6.3.2 对恶意应用的实例分析

恶意应用泄露用户隐私的行为通常具有较高的恶意性,通过对恶意应用的逆向分析,我们发现恶意应用的动态加载主要是为了隐藏自身和动态下载执行一些恶意程序,反射机制的运用也能在一定程度上帮助恶意应用绕过一些安全检测.例如,App6 将恶意的 dex 文件隐藏在资源文件中,并伪装其名称为"logos.png",实际上该文件并不是 png 文件,而是 zip 文件,程序运行时,通过"XOR"解密和其他一些处理将其转换为 dex 文件,并通过反射调用对其进行调用.DyLoadDroid 截获了其加载命令,将这个转换后的 dex 文件下载到本地电脑,并通过污点分析,成功发现它读取短信记录和通信录并发送给外界的隐私泄露路径.

App8 安装后会在后台默默启动一个服务,这个服务会获取本应用在手机中被授予的权限信息,并将信息传给一个 HTTP 服务器,该 HTTP 服务器收到信息时会返回一个 URL,该 URL 是一个通往网络上的 jar 包的路径.然后 App8 会下载并加载和调用该 jar 文件,DyLoadDroid 通过污点分析成功分析出该 jar 文件所具有的读取通信录、通话记录并传给 HTTP 服务器的行为.

App9 将 2 个隐藏的 App 应用置于其 assets 文件夹,取名为"anservera.db"和"anserverb.db".我们简称其为 pa 和 pb,当 App9 运行时,会弹出一个对话框,诱惑用户安装 pa,pa 安装完成后不显示任何图标,在后台运行(这样一来,即使 App9 被卸载了,pa 也可以继续执行恶意操作).App9 和 pa 都可以通过动态加载的方式执行 pb,pb 会连接网络服务器,向该服务器发送用户的隐私数据,并接收网络服务器的命令,DyLoadDroid 成功追踪到 pb 获取短信记录和通信录并传给网络服务器的污点传播路径.

App10 在运行时,会注册 2 个服务,第一个服务动态地注册一个具有最大权限的短信监听器,另一个服务会使用 DES 算法解密 app_TYPE_JAR 文件夹下的 rt2.jar 文件,当解密完成后,它能够在 app_sim_index 文件夹下生成一个真正的 jar 格式文件 appmgr.jar,然后 App10 动态加载并运行 appmgr.jar,并使用反射机制调用其中的方法,DyLoadDroid 成功追踪到它泄露用户通讯录和通话记录的程序路径.

6.4 与其他工具的对比

由于 FlowDroid 是当前领先的 Android 静态污点分析工具,而本文又是在 FlowDroid 的基础上进行的改进,通过前面几节的对比,DyLoadDroid 已经显示出相对于 FlowDroid 的优势,因此,我们不需要将 DyLoadDroid 与其他类型的静态污点分析工具进行对比.Epicc[18] 和 DidFail[19] 虽然是对 FlowDroid 的扩展,但它们只是扩展了 FlowDroid 所能够分析的数据传播途径,工具的主体还是 FlowDroid,在污点分析效果上并没有明显改善,因此我们也没有必要将 DyLoadDroid 与它们进行对比,而将它们作为 DyLoadDroid 的有效补充.本节内容主要是将 DyLoadDroid 与一些动态污点分析工具进行对比,因为动态污点分析工具通常不会受到动态加载和反射机制的影响,能够对其进行正常的污点分析.又由于静态污点分析工具与动态污点分析工具各具优势和劣势,难以进行定量比较,因此,我们不对静态污点分析工具和动态污点分析工具进行定量比较,而是对他们进行定性比较.

1) 与 TaintDroid 的比较

TaintDroid 是当前较为领先的 Android 动态污点分析工具,TaintDroid[8] 将所有的隐私数据标记为污点,它在内存中为系统中的每个对象多申请一倍的空间,用于存储污点标记,在 App 动态执行的过程中,根据污点标记追踪污点在内存中的传播情况.当 TaintDroid 监测到隐私数据被泄露出去时,会弹出提示框提醒用户.TaintDroid 可以对 Android

动态加载和反射机制进行正确的污点分析,并且具有污点分析准确、误报率低的优点.就准确性而言,FlowDroid 和 DyLoadDroid 都无法和 TaintDroid 相比,然而,从程序覆盖率方面来说,DyLoadDroid 却要高于 TaintDroid,可以发现 TaintDroid 不能发现的污点传播路径.虽然在处理 Android 动态加载和反射机制的问题上,两者都依赖 App 的实际运行,但 DyLoadDroid 依赖 App 的实际运行只是为了获得 App 动态加载的 dex 文件和反射调用信息,以辅助静态污点分析的过程,弥补静态污点分析工具对动态加载和反射机制不能正确处理的缺陷.DyLoadDroid 只需要监测到 Android 的动态加载和反射调用的行为便可以对 App 进行更为全面的污点分析,而 TaintDroid 则需要 App 在动态运行的过程中走完从污染源到陷入点的完整的路径才能确定是否发生隐私泄露,而若某条路径上出现程序阻断(例如网络阻塞、用户认证失败等)时,TaintDroid 便不能追踪到这条路径上存在的隐私泄露问题.此外,TainDroid 只能在发生隐私泄露时向用户弹窗提醒,而不能输出从污染源到陷入点的完整的路径,DyLoadDroid 则可以输出从污染源到陷入点的完整的路径.

2) 与 AppFence 的比较

AppFence 是一个隐私保护工具,不仅可以检测到系统中发生的隐私泄露问题,而且可以阻止隐私信息的泄露.它采取数据遮蔽和渗出阻塞技术限制 App 泄露隐私数据,核心思想是将隐私数据替换为影子数据传递到外部网络来防止隐私数据的泄露.与 TaintDroid 一样,AppFence 能准确地检测出 Android 系统中发生的隐私数据泄露,误报率低.然而,它在代码覆盖率和漏报率方面都存在和 TaintDroid 同样的缺点,比 DyLoadDroid 弱.并且 AppFence 容易阻碍 App 的正常运行,因为 App 的正常运行可能需要向外部网络发送一些隐私信息(如 IMEI、位置信息等),而 AppFence 的数据遮蔽策略可能导致 App 不能正常地向外发送数据,阻碍 App 的正常运行,因此无法顺利地监控到 App 中存在的其他隐私泄露问题.

7 讨 论

虽然 DyLoadDroid 可以对 Android 的动态加载和反射机制进行有效的污点分析,但 DyLoadDroid 仍然有其自身的缺陷,主要包括:

1) 污点分析的准确性依赖于 App 实际运行中被加载的 dex 文件数和被触发的反射调用的比例.通过表 2 可以看出,在 App 实际运行过程中,DyLoadDroid-D 并不能保证能够触发到 App 中所有的反射调用行为,并且也不能保证能够下载到所有能够被加载的 dex 文件.因此,DyLoadDroid 并不能保证对 App 的动态加载和反射调用机制进行完整的分析,它只能保证以最大的可能弥补静态分析方法在处理动态加载和反射调用机制方面的缺陷.

2) 不能有效分析组件间和应用间的污点传播.在第 1 节中我们提到过,Epicc 和 DidFail 扩展了 FlowDroid 所能够分析的数据传播途径,使 FlowDroid 能够追踪到组件间和应用间的污点传播.然而 DyLoadDroid 并没有将这 2 个工具对 FlowDroid 的扩展功能集成进来,因此 DyLoadDroid 并不能对组件间和应用间的污点传播进行有效处理.原因是 Epicc 目前没有开放源代码,而 DidFail 目前还不完善,实用性较差,并且 DidFail 依赖于 Epicc.因此,目前将 Epicc 和 DidFail 对 FlowDroid 的扩展功能集成进 DyLoadDroid 不易于实现.可以认为,Epicc,DidFail 与 DyLoadDroid 互为补充,三者对 FlowDroid 进行了不同方面的改进.我们也正在联系 Epicc 和 DidFail 的作者,将 Epicc、DidFail 与 DyLoadDroid 进行有效集成,实现更为强大的静态污点分析工具.

在以后的研究工作中,我们将重点研究如何使 App 触发更多的动态加载行为和更多的反射调用行为,以提高 DyLoadDroid 的污点分析的准确性.同时,我们将对 DyLoadDroid 进行扩展,实现组件间和应用间的正确的污点分析.

8 结束语

本文针对当前 Android 静态污点分析工具无法对包含 Android 动态加载和反射机制的 Android 应用进行正常污点分析的问题进行了研究,对当前领先的静态污点分析工具 FlowDroid 进行了改进,提出了 DyLoadDroid 工具,并介绍了 DyLoadDroid 的 2 个主要模块 DyLoadDroid-D 和 DyLoadDroid-S 的设计原理,提出了根据 App 在实际运行过程中输出的动态加载和反射调用信息,下载被加载的 dex 文件,并将相应的反射方法调用修改为 FlowDroid 能够处理的方法调用的解决方案,使 FlowDroid 能够对 Android 动态加载和反射机制进行正常的污点分析.我们通过实验验证了 DyLoadDroid 的有效性,

并和 FlowDroid 进行了对比,并对一些实验样本进行了实例分析,我们也将 DyLoadDroid 与其他动态污点分析工具进行了定性对比,说明了 DyLoadDroid 所具有的优势.

参 考 文 献

[1] Wei T E, Jeng A B, Lee H M, et al. Android privacy [C] // Proc of 2012 Int Conf on Machine Learning and Cybernetics. Piscataway, NJ: IEEE, 2012: 1830-1837

[2] Felt A P, Finifter M, Chin E, et al. A survey of mobile malware in the wild [C] //Proc of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. New York: ACM, 2011: 3-14

[3] Zhou Y, Jiang X. Dissecting Android malware: Characterization and evolution [C] //Proc of 2012 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2012: 95-109

[4] Shu X, Du Z, Chen R. Research on mobile location service design based on Android [C] //Proc of the 5th Int Conf on Wireless Communications. Piscataway, NJ: IEEE, 2009: 1-4

[5] Grace M C, Zhou W, Jiang X, et al. Unsafe exposure analysis of mobile in-app advertisements [C] //Proc of the 5th ACM Conf on Security and Privacy in Wireless and Mobile Networks. New York: ACM, 2012: 101-112

[6] Book T, Wallach D S. A case of collusion: A study of the interface between ad libraries and their apps [C] //Proc of the 3rd ACM Workshop on Security and Privacy in Smartphones & Mobile Devices. New York: ACM, 2013: 79-86

[7] Zhang Yuqing, Fang Zhejun, Wang Kai, et al. Survey of Android vulnerability detection [J]. Journal of Computer Research and Development, 2015, 52(10): 2167-2177 (in Chinese)
(张玉清, 方喆君, 王凯, 等. Android 安全漏洞挖掘技术综述[J]. 计算机研究与发展, 2015, 52(10): 2167-2177)

[8] Enck W, Gilbert P, Han S, et al. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones [J]. ACM Trans on Computer Systems, 2014, 32(2): No. 5

[9] Hornyack P, Han S, Jung J, et al. These aren't the droids you're looking for: Retrofitting Android to protect data from imperious applications [C] //Proc of the 18th ACM Conf on Computer and Communications Security. New York: ACM, 2011: 639-652

[10] Schreckling D, Köstler J, Schaff M. Kynoid: Real-time enforcement of fine-grained, user-defined, and data-centric security policies for Android [J]. Information Security Technical Report, 2013, 17(3): 71-80

[11] 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

[12] Yang Z, Yang M. Leakminer: Detect information leakage on Android with static taint analysis [C] //Proc of the 3rd World Congress on Software Engineering. Piscataway, NJ: IEEE, 2012: 101-104

[13] Gibler C, Crussell J, Erickson J, et al. Androidleaks: Automatically detecting potential privacy leaks in Android applications on a large scale [J]. Trust and Trustworthy Computing, 2012, 73: 291-307

[14] Zhauniarovich Y, Ahmad M, Gadyatskaya O, et al. StaDynA: Addressing the problem of dynamic code updates in the security analysis of Android applications [C] //Proc of the 5th ACM Conf on Data and Application Security and Privacy. New York: ACM, 2015: 37-48

[15] Lindorfer M, Neugschwandtner M, Weichselbaum L, et al. ANDRUBIS-1000000 Apps later: A view on current Android malware behaviors [C/OL] //Proc of the 3rd Int Workshop on Building Analysis Datasets and Gathering Experience Returns for Security. 2014 [2015-09-20]. http://cs. ucsb. edu/~yanick/publications/2014_badgers_andrubis. pdf

[16] Rastogi V, Chen Y, Jiang X. Droidchameleon: Evaluating Android anti-malware against transformation attacks [C] // Proc of the 8th ACM SIGSAC Symp on Information, Computer and Communications Security. New York: ACM, 2013: 329-334

[17] Poeplau S, Fratantonio Y, Bianchi A, et al. Execute this! Analyzing unsafe and malicious dynamic code loading in Android applications [C/OL] //Proc of the 20th Annual Network & Distributed System Security Symp. 2014 [2015-09-20]. https://cs. ucsb. edu/~ vigna/publications/2014 _ NDSS_ExecuteThis. pdf

[18] Octeau D, McDaniel P, Jha S, et al. Effective inter-component communication mapping in Android with epicc: An essential step towards holistic security analysis [C] //Proc of the 22nd USENIX Security Symp. Berkeley: USENIX Association, 2013: 543-558

[19] Klieber W, Flynn L, Bhosale A, et al. Android taint flow analysis for app sets [C] //Proc of the 3rd ACM SIGPLAN Int Workshop on the State of the Art in Java Program Analysis. New York: ACM, 2014: 1-6

[20] Chiba S. ECOOP 2000—Object-Oriented Programming [M]. Berlin: Springer, 2000: 313-336

[21] GitHub. Soot [EB/OL]. [2015-09-20]. http://sable. github. io/soot

[22] Wandou Lab. wandoujia [EB/OL]. [2015-09-20]. https://www. wandoujia. com

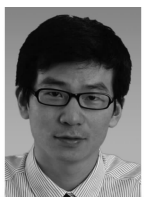
[23] VirusShare. Latest virus samples [EB/OL]. [2015-09-20]. http://virusshare. com

[24] Rasthofer S, Arzt S, Bodden E. A machine-learning approach for classifying and categorizing Android sources and sinks [C/OL] //Proc of the 20th Annual Network & Distributed System Security Symp. 2014 [2015-09-20]. http://www. bodden. de/pubs/rab14classifying. pdf

Yue Hongzhou, born in 1987. PhD candidate in the Department of Communication Engineering, Xidian University. His main research interests include Android security and Web security.

Zhang Yuqing, born in 1966. Professor in the Department of Computer and Control Engineering, University of Chinese Academy of Sciences. His main research interests include network and information system security.

Wang Wenjie, born in 1964. Associate professor in the Department of Computer and Control Engineering, University of Chinese Academy of Sciences. His main research interests include intelligent information processing and software reliability measurement.



Liu Qixu, born in 1984. Lecturer in the Department of Computer and Control Engineering, University of Chinese Academy of Sciences. His main research interests include system security, vulnerability assessment and Web security.

2017 年《计算机研究与发展》专题(正刊)征文通知

——人工智能前沿进展

智能化是信息技术发展的主流趋势,人工智能技术已广泛渗透于科学发现、经济建设、社会生活等各个领域.国务院 2016 年 11 月印发的《"十三五"国家战略性新兴产业发展规划》中将发展人工智能作为推动信息技术产业跨越发展的基础之一,美国政府同年 10 月发布的《国家人工智能研究与发展战略规划》将人工智能研发提升至国家战略层面.随着大数据、云计算、物联网等信息技术的不断发展,人工智能研究在理论、方法、应用等多个层面均面临新的挑战.

为及时反映国内同行在人工智能前沿的最新研究成果,进一步推动我国人工智能领域的创新发展,《计算机研究与发展》将于 2017 年 8 月出版"人工智能前沿进展"专题.欢迎人工智能及相关研究领域的专家学者、科研人员踊跃投稿.此外,专题组稿与 2017 中国计算机学会人工智能会议(CCFAI 2017)合作,遴选若干高质量会议投稿推荐至本专题.

征文范围(但不限于)

· 人工智能基础理论

知识表示与推理、多智能体系统、认知建模、计算学习理论、博弈论等.

· 人工智能方法技术

机器学习算法、数据挖掘与知识发现、规划(Planning)、搜索与优化、计算智能等.

· 人工智能应用

自然语言处理、模式识别、计算机视觉、信息检索、社交网络等.

征文要求

1) 论文应属于作者的科研成果,数据真实可靠,具有重要的学术价值与推广应用价值,且未在国内外公开发行的刊物或会议上发表,不存在一稿多投问题.作者在投稿时,需向编辑部提交版权转让协议.
2) 论文一律用 Word 格式排版,论文格式体例参考近期出版的《计算机研究与发展》的要求(http://crad.ict.ac.cn/).
3) 论文须通过期刊网站(http://crad.ict.ac.cn)投稿,投稿时提供作者的联系方式,留言中务必注明"人工智能前沿进展 2017 专题"(否则按自由来稿处理).

重要日期

论文截稿日期:2017 年 3 月 20 日
录用通知日期:2017 年 5 月 10 日
最终稿提交日期:2017 年 5 月 17 日
出版日期:2017 年 8 月

特邀编委

于 剑 教授 北京交通大学 jianyu@bjtu.edu.cn
张敏灵 教授 东南大学 zhangml@seu.edu.cn
余正涛 教授 昆明理工大学 ztyu@hotmail.com
尹义龙 教授 山东大学 ylyin@sdu.edu.cn

联系方式

编辑部:crad@ict.ac.cn 010-62620696, 010-62600350
通信地址:北京 2704 信箱《计算机研究与发展》编辑部
邮政编码:100190