基于静态污点分析的 Android 隐私泄露检测方法研究

胡英杰[1], 张琳琳[2], 赵 楷[2], 方文波[1], 于媛尔[2]

[1]新疆大学软件学院 乌鲁木齐 中国 830091
[2]新疆大学信息科学与工程学院 乌鲁木齐 中国 830046

摘要 Android 移动设备中存储了大量的敏感信息, 如通话记录、联系人等, 容易成为恶意攻击者的目标。基于静态污点分析技术, 提出了一种面向 Android 平台的隐私泄露检测方法。通过提取 Android 敏感权限与 API, 创建两者之间的映射关系, 生成 Android 应用程序的函数调用图, 实现了对于大规模应用程序中潜在隐私数据泄露行为的检测。实验结果表明, 本文所提出方法的准确率较高, 且运行耗时较短, 适合于大规模应用程序的检测。

关键词 Android; 敏感数据; 隐私泄露; 函数调用图; 污点分析
中图法分类号 TP309 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2020.09.10

Android Privacy Leak Detection Method Based on Static Taint Analysis

HU Yingjie[1], ZHANG Linlin[2], ZHAO Kai[2], FANG Wenbo[1], YU Yuaner[2]

[1] College of Software, Xinjiang University, Urumqi 830091, China
[2] College of Information Science and Engineering, Xinjiang University, Urumqi 830046, China

Abstract Android mobile devices store a large amount of sensitive information, such as call records, contacts, and so on, which is easy to be target of malicious attackers. A privacy leakage detection method based on static taint analysis is proposed. A function call graph of the Android application is generated by extracting Android sensitive permissions and API to create a mapping relationship between them, and to detect potential privacy data leakage behavior in large-scale applications. The experimental results show that the accuracy of the proposed method is higher with shorter running time, which is suitable for the detection of large-scale applications.

Key words Android; sensitive information; privacy leakage; call graph; taint analysis

1 前言

随着互联网时代的发展, 以智能手机为代表的移动智能终端迅速普及。据 IDC 发布的智能手机市场最新预测报告统计, Android 手机销量在全球市场占据 87.0%的份额[1], 且在未来四年内将呈稳定增长趋势。由于 Android 的开源特性, 来自各大手机应用市场的应用程序(Application, 简称 App)数量逐年增长, 截至 2018 年 12 月, 我国市场上 App 的数量已高达 449 万款[2]。App 在为用户提供越来越丰富功能的同时, 由于包含了大量的敏感数据, 如用户的 IMEI ID、位置信息、联系人、通话记录等, 更易成为恶意攻击者的目标。由于 Android 市场内的大量 App 在上传时没有经过严格的安全审查, 导致 Android App 存在较大的安全隐患。根据 App 隐私权限测评权威报告[3], 63.3%的用户没有认真阅读过隐私条款, 75.7%的用户未进行过明确授权, 对于 App 的功能不甚了解。

如何防止 Android 平台下敏感信息的泄露成为人们普遍关注的话题。一方面, Android 系统在

通讯作者: 张琳琳, 博士, 副教授, Email: zllnadasha@126.com。
本课题得到国家自然科学基金项目(No. 61867006); 新疆维吾尔自治区科技厅创新环境建设专项(PT1811); 新疆维吾尔自治区创新环境建设专项(自然科学基金)联合基金项目(No. 2019D01C062, 2019D01C041); 新疆维吾尔自治区高校科研计划项目(No. XJEDU2017M005); 国家级大学生创新创业训练计划项目(No. 201910755047)资助。
收稿日期: 2019-08-31; 修改日期: 2020-03-09; 定稿日期: 2020-08-21

Manifest 文件中对权限进行了声明[4], App 开发人员必须根据此文件来申请所使用的资源。另一方面, 当用户在安装 App 时, 安装程序会提醒用户授权。如果用户选择了“拒绝”, 则 App 无法获取此权限相关的资源。然而, 实际生活中, 由于用户的安全意识淡薄, 多数情况下用户会直接略过隐私条款选择“同意”, 无意中造成了“允许”App 访问系统中的数据, 从而增大隐私泄露的风险。

App 在为用户提供更加便利服务的同时, 难免会带来用户隐私泄露问题。例如, 天气预报类 App 需要请求系统的 INTERNET、ACCESS_FINE_LOCATION 以及 ACCESS_COARSE_LOCATION 等权限, 在与互联网连接的情况下, 获取用户当前的位置情况, 以便告知用户当地天气情况。然而, 天气预报 App 在为用户提供服务的同时, 存在将用户的位置信息秘密泄露给第三方的风险, 从而造成数据泄露。

针对上述问题, 如何有效地检测用户手机中存在的隐私泄露行为变得十分重要。本文提出一种基于静态污点分析的隐私泄露检测方法, 用于大规模检测 Android App 中存在的个人隐私信息泄露情况, 提高对于用户隐私信息的保护能力。

2 相关工作

Android 平台的隐私数据包括位置、电话状态、录音等各类信息, 这些敏感数据总是流向集中固定类型的数据泄露点[5], 如短信和网络。应用市场通常仅采用较为简单的检测方法来识别 App 功能, 导致开放平台的监管力度有所不足[6], 而污点分析作为信息流分析的一种实践技术, 常被研究人员用于对隐私数据的传输过程进行分析[7], 主要包含动态和静态污点分析两种技术。

动态污点分析是指通过实时监控已标记的敏感数据, 即污点数据在程序中的传播过程, 来监测数据能否从污点数据的来源传播到污点数据发生泄露的位置[8]。Intel 实验室[9]设计了一款系统级高效的动态污点分析和追踪系统 TaintDroid, 用于在 Android 虚拟执行环境中对 App 的运行时数据状态进行分析。北京交通大学的何永忠等人[10]通过提出了一种细粒度的动态隐私泄露分析工具, 能够实时分析由第三方库造成的隐私泄露行为; 中南大学的王伟平教授等人[11]提出的 BridgeTaint, 是一种双向动态污点跟踪方法, 能够在 App 执行期间动态跟踪受污染的数据, 实现了 BridgeInspector 工具, 用于检测使用 JavaScript 混合应用程序中的跨语言隐私泄露和代码注入攻击问题。然而, 尽管动态污点分析技术在一定程度上可以解决隐私泄露问题, 但仍存在局限性, 例如运行程序需要占用较大的空间, 消耗大量的计算资源, 且在运行时可能无法覆盖应用程序的所有传输路径, 导致检测准确率会有所下降。

静态污点分析能够在不执行程序的前提下, 通过分析程序变量间的数据依赖关系, 检测数据能否从污点源传播到污点汇聚点[8]。FlowDroid[12]通过创建虚拟 main 函数来模拟组件的生命周期, 检测敏感数据传输路径, 以发现其中的隐私泄露问题, 实验结果精确度达到了 86%; Zohreh 等人[13]设计了 IIFDroid, 是一种组件间信息流控制静态分析工具, 用于检测 App 组件间通信过程中的显示和隐式流信息泄露; 西安电子科技大学的田聪教授等人[14]致力于组件间通信(ICC)重用和修订 Intent 的分析, 通过构建 ICC 图, 来跟踪 ICC 泄露的组件, 检测到许多潜在的泄露; 北京信息科技大学的曹宏盛等人[15]针对隐私信息泄露过程中的隐式信息泄露问题, 利用结构相关流模型 SRFM 来描述隐式流和控制结构之间的关系, 开发了原型系统 TSDroid, 用于更加高效准确地检测数据传输过程中造成的隐式信息泄露; 中国科学院计算技术研究所的王蕾等人[16]提出了一种多源绑定发生的污点分析技术, 分析多组源的绑定发生情况, 设计了原型系统 MultiFlow, 验证了方法的有效性; 西安电子科技大学的 ICTT 与 ISN 实验室[17]提出了 FastDroid, 通过构造污点值图(taint value graph, TVG), 对传播过程进行分析以获得更高的精度和召回率, 效率显著提升; 王蕾等人[18]针对当前检测分析中存在的开销过高的问题, 提出了基于稀疏优化的污点分析方法, 实现了工具 FlowDroidSP, 消除静态污点分析中无关联的传播, 降低了运行时间, 减少了使用内存。

以上研究方法在隐私泄露检测方面已取得较好的进展, 通过总结分析可以得知, 检测敏感权限所对应的 API 调用, 有助于更为准确地发现数据泄露的污点源, 从而获得更高的检测准确率。然而, 上述研究侧重于对单一应用程序的全面分析, 对于大规模 App 的分析能力略显不足。因此, 本文通过对 Android 源码进行分析, 在敏感权限及其相关的 API 之间建立映射关系, 并将其应用于大规模的 App, 用以分析其中可能调用敏感信息的权限及 API 的执行路径。继而, 对找到的方法路径执行静态污点分析, 以确定其中是否存在隐私数据泄露的情况。

3 隐私泄露检测方法

本节在深入研究 Android 理论的基础上, 结合已有数据集实际情况, 提出本文研究方法, 整体框架如图 1 所示。该框架主要包括三个主要检测过程: (1)逆向工程提取权限和 API; (2)生成函数调用图; (3)进行静态污点分析。目前 Android 包含 Google 官方声明的权限以及大量的已声明和未声明的 API, 通过查找其中敏感的权限以及 API, 实现函数调用图的创建。另外, 根据 Android 开发相关知识, 只有 7%的 Android 应用程序中包含本地代码, 对检测结果影响较小, 因此暂不考虑本地代码调用带来的影响。本文的实验数据集来源于 Google Play 和安智网, 收集了不同类别的应用程序, 共计包含 15 个类别中排名前 100 的 1500 个 App。

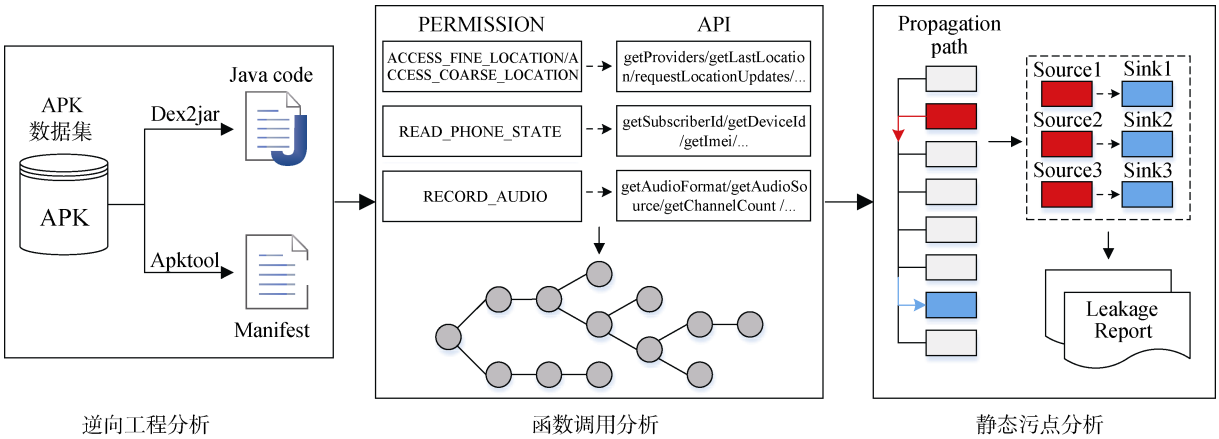

图 1 隐私泄露检测整体框架

Figure 1 Privacy leakage detection framework

3.1 Android 理论基础

Android App 使用 Java 语言编写, 被打包成 apk 文件[19]。Apk 是一个压缩文件, 可以将其直接进行解压, 得到 Android App 包含的六个部分, 每个部分的含义如表 1 所示:

表 1 Apk 文件结构

Table 1 Apk file structure

资源文件名	说明
res	存储资源文件
META-INF	存放应用程序签名信息
lib	存放应用程序依赖的 native 库文件
assets	与 res 类似, 但可直接通过访问文件地址来访问类
classes.dex	Java 源码编译后生成的 Java 字节码文件
resources.arsc	编译后的二进制资源文件的索引
AndroidManifest.xml	应用程序的配置文件, 包含组件、应用权限、sdk 版本等信息

Android App 的开发包含四大基本组件, 分别是活动(Activity)、服务(Service)、广播接收者(BroadcastReceiver)和内容提供者(ContentProvider)。四个组件虽然相互独立, 但可以相互调用, 从而实现 App 的正常运行。

Android 还提供了 Intent 机制, 一种运行时绑定机制, 在程序运行的过程中连接两个不同的组件。使用 Intent 不仅可以用于程序间的交互以及通讯, 还可用于 App 内部的 Activity、Service 和 BroadcastReceiver 之间的交互, 因而 Intent 是 Android 进行通信的重要组成部分。

3.2 Android 逆向工程分析

对于.apk 文件, 首先使用 apktool 对其进行逆向工程分析(又称反编译), 得到存放代码的 Smali 文件和 AndroidManifest.xml 文件。前者是一种基于寄存器的语言, 每一个 Smali 文件都对应着一个 Java 文件。后者存放了程序配置信息, 定义了软件所需要的权限信息以及 Intent 内容。其次, 借助 dex2jar 和 JD-GUI 工具获得可读的 Java 源码。dex2jar 工具能够将 dex 格式的文件转换成 jar 文件, dex 文件是在 Android 虚拟机上的可执行文件, 而 jar 文件则是 Java 的 class 文件。JD-GUI 是一个图形化工具, 用于查看由 dex2jar 得到的 jar 包, 即查看 Java 源代码。最后, 通过逆向工程分析得到 AndroidManifest.xml、res、Java 代码等资源文件进行整理, 便于下一步从中获取如敏感权限、API 等需要进行处理的信息。

3.3 函数调用分析

权限为 Android 平台 App 的开发, 提供了对于隐私敏感数据的访问控制。截至 Android API level 29, Android 共包含 164 个权限。自 Android 6.0 起, Google 官方将权限分成了普通权限和危险权限两类, 其中

危险权限一共有 9 组共 24 个, 分别是日历、相机、联系人、定位、麦克风、电话、传感器、短信和存储。大多数时候, 危险权限都会涉及用户的隐私, 在使用时需要用户进行授权, 恶意行为将通过调用这些危险权限以获取敏感信息, 从而进行敏感数据的传输与发送, 例如读取用户当前位置、访问通讯录和后台拨打电话等。

相对于其他的普通权限, Android 的 24 个危险权限, 在实现 App 正常功能的同时, 也更加容易被恶意攻击者利用。恶意攻击者通过在程序中插入恶意代码段, 私自调用危险权限, 将用户的敏感信息向外界传输, 从而窃取用户的个人敏感信息。因此, 本文将 Android 的 24 个危险权限定义为敏感权限, 重点分析敏感权限与 API 之间的映射关系, 检测隐私泄露情况。继而借助 PScout[20]工具和 Soot 框架[21]来进行静态分析, 其中, PScout 用来获取 Android 框架中敏感权限与 API 之间的映射关系, Soot 框架则进行 Android 的静态分析。

对于 Android App, 有三种类型的操作与权限相关: (1)通过 *checkPermission*()方法调用 API; (2)利用 Intent 进行通信的方法; (3)利用 ContentProvider 进行通信的方法, 如图 2 所示:

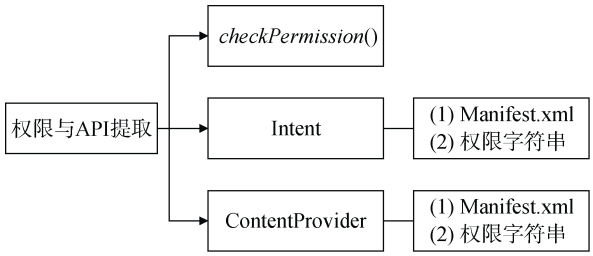

图 2 权限与 API 提取

Figure 2 Permissions and API extraction

(1) 通过 *checkPermission*()调用 API。App 调用的敏感权限在 Android 源码中显示为字符串, 大多数情况下敏感权限会与用户的信息一起传递给 *checkPermission*()函数。因而在检查敏感权限时, 可先定位至 *checkPermission*()函数, 而后对权限使用情况进行识别和判断。例如, 对于访问位置信息的敏感权限, 将通过定位 *context.checkPermission(ACCESS_FINE_LOCATION)*语句来进行确认, 如果存在位置权限调用, 返回值将为 *PackageManager.PERMISSION_GRANTED*, 根据该语句遍历与位置这一敏感权限相关的全部 API, 把这些 API 汇总为一个集合, 从而可得该敏感权限与 API 之间的映射关系。

(2) 利用 Intent 进行通信的方法。Android 的组件之间进行通信时通常会利用 Intent 发送和接收数据, 而与 Intent 操作字符串相关联的权限信息也会写入 Manifest 文件中。例如, 当访问位置信息时, 需要 Intent 进行通信, 在 Manifest 文件中会存在 *<uses-permission android:name = "android.permission.ACCESS_FINE_LOCATION" />*语句。因此, 可以通过对 Android 源码进行分析, 检查权限字符串 *sendBroadcast(intent)* 和 *registerReceiver(intent)* 来识别 Intent 的调用, 并构建敏感权限与 Intent 之间的映射关系。

(3) 利用 ContentProvider 进行通信的方法。提取 ContentProvider 的方法与 Intent 类似, ContentProvider 需要读取和写入的权限在 Manifest 中进行声明, URI 会将指定的 ContentProvider 传递给 ContentResolver 类。通过检查与 ContentProvider 相关联的 URI, 来识别对应的权限调用, 在敏感权限与 ContentProvider 之间构建映射。

在以上三种类型的敏感权限与 API 之间映射关系的基础上, 对 App 源码进行分析, 查找源码中与敏感权限及 API 相关的方法函数调用, 并根据函数之间的调用关系, 为每个 App 源码生成函数调用图。图 3 为与位置这一敏感权限相关的部分函数调用图示例, 图中的每个节点表示一个函数, 节点之间的边表示函数之间的调用关系, 如 *onLocationChanged*()将调用 *Android.Location*(), 以实现其功能。此外在函数的实现过程中需要设置异常检测, 了解函数运行情况, 通过 *RuntimeException*()来实现对于函数异常的抛出。

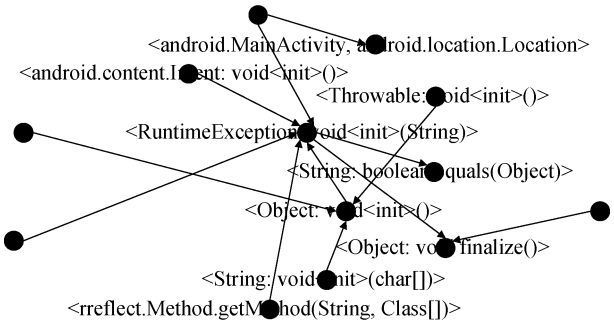

图 3 函数调用图示例

Figure 3 An example of call graph

3.4 静态污点分析

在本节中, 将采用静态污点分析技术对 Android App 中的数据传输路径进行检测, 以确定是否有隐私数据流发生隐私泄露。为了方便理解此分析过程, 首先对污点分析技术进行简单介绍。

3.4.1 污点分析基础知识

(1) **污点源 Source**。Source 是那些直接引入不受信任数据的接口。App 访问危险权限时调用的 API 被标记为污点源。

(2) **污点汇聚点 Sink**。Sink 是将敏感数据传送到外界, 从而导致隐私泄露的接口。App 中从污点源传递过来, 并通过 Internet 对外发送数据的 API 被标记为污点汇聚点。

(3) **污点传播**。污点传播是指从 Source 处开始, 沿方法调用顺序向 Sink 进行传输的过程及所经过的路径。图 4 展示了污点源 a 的值通过显示和隐式流结合的方式传递给污点汇聚点 b 的过程和路径。污点传播的类型分为组件内传播、组件间传播、组件与函数之间传播三种[8], 如图 5 所示。

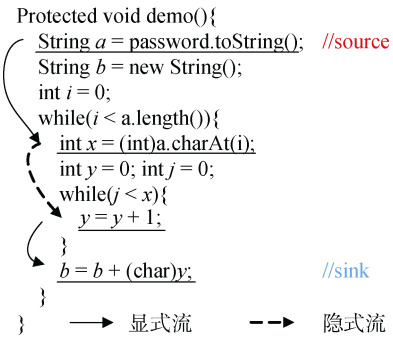

图 4 污点信息流分析

Figure 4 Taint information flow analysis

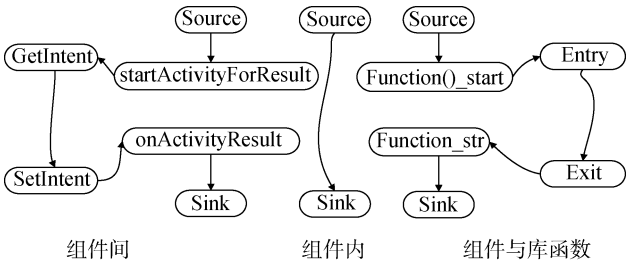

图 5 污点传播类型

Figure 5 Taint propagation type

3.4.2 污点分析技术在本文的应用

根据生成的与敏感权限相关的函数调用图, 可以确定隐私数据传输的污点源和污点汇聚点, 以及由两者相互之间组成的边(如{*Perm: Source*$_i$→S_{i1}…S_{in}→*Sink*$_i$})的集合。

(1) **Source 确定**。根据敏感权限与 API 之间的映射关系, 以及生成的函数调用图来确定 Source。以位置信息的传输为例, 选择 *getLatitude*()、*getLongitude*()、*getLastKnownLocation*() 、 *getCid/getLac*() 等作为 Source。

(2) **Sink 确定**。目前, App 造成的敏感信息泄露主要归因于恶意攻击者进行远程信息窃取, 并通过网络接口向外界传输信息。因此, 根据函数调用图, 从中提取出所有与 Internet 有关的敏感 API(如(*java.net.URL*)*set/init*() 、 *java.net.URLConnection*() 、(*HttpClient*)*execute*())作为 Sink。

(3) **污点传播**。污点传播的流程如图 6 所示, 以位置信息的传输为例, 首先选取一个 Source(如 *getLatitude*()), 进行前向切片分析, 得到与它有关的后续调用方法; 将新得到的方法作为起点, 并标记为已访问节点, 判断是否为污染数据, 而且如果传输敏感数据, 记录当前路径, 如果没有, 则访问下一个方法, 然后继续计算它的前向切片; 以此迭代, 遍历得到与该 Source 有关的函数调用方法以及数据流传输路径, 作为被污染的数据; 对于其他的 Source, 如 *getLongitude*()、*getLastKnownLocation*()等也执行同样的操作; 最后, 将 App 中从 Source 开始进行传播的所有污染数据流进行汇总, 以得到 App 中与敏感权限调用相关的数据传输路径。

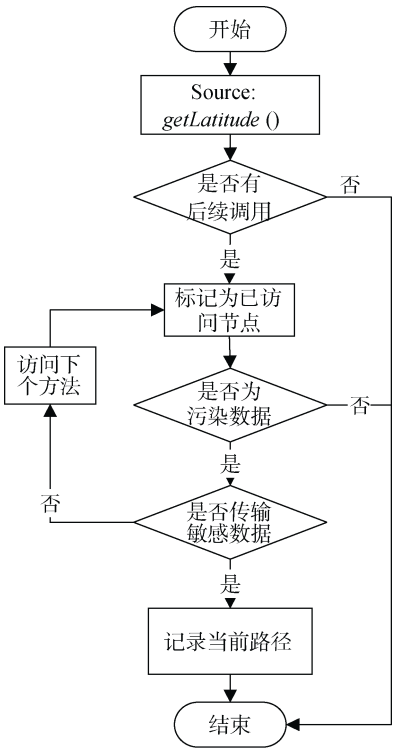

图 6 污点传播

Figure 6 Taint propagation

根据 App 运行得到的污染数据流, 查看通过网络输出的方法是否存在与 Sink 相匹配的 API 方法接收参数, 即是否有从 Source 到 Sink 的数据流存在。如果存在, 再进行人工操作对路径进行进一步的确

认和筛选, 以确认是否真正发生了隐私数据泄露。

4 实验结果与分析

本文对来自Google Play和安智网共15个类别的1500 个 App 进行了实验分析。本文的实验环境为: Intel Core i5 2.5GHz CPU, 8GB RAM, Ubuntu-16.04 版本的操作系统, 编程语言为 Java, 实验平台为 Eclipse。

实验选择了位置信息、电话状态以及录音三种敏感权限作为待检测的对象。就位置访问这一信息而言, 重点跟踪了 ACCESS_FINE_LOCATION 以及 ACCESS_COARSE_LOCATION 对 API 接口的访问; 电话状态则选取 READ_PHONE_STATE; 录音信息选择了 RECORD_AUDIO。这三类权限与 API 之间的部分对应关系如表 2 所示。

表 2 目标权限与 API 对应情况

Table 2 Target permissions correspond to the API

权限	API
ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION	getProviderProperties 获取 Provider 属性 addGpsStatusListener 获取 GPS 状态 addGpsMeasurementsListener 添加 Gps 测量监听器 requestLocationUpdates 请求位置更新 getLastLocation 获取当前位置 getProviders 获取 Provider ……
READ_PHONE_STATE	getNetworkPolicies 获取网络策略 getSubscriberId 获取 SIM 卡唯一标识 getDeviceId 获取设备号 getImei 获取 IMEI clearSubInfo 清除 sub 信息 ……
ACCESS_RECORD_AUDIO	getAudioFormat 获取音频格式 getAudioSource 获取音频资源 getChannelConfiguration 获取 Channel 配置 getChannelCount 获取 Channel 账户 getMinBufferSize 获取最小 Buffer ……

作为对照, 本文同时运行了FlowDroid污点分析工具。选择敏感权限对应的 API(如 *getLatitude*()、*getLastKnownLocation*()、*getDeviceId*()等)作为确定的 Source, 将通过 Internet 发送信息的调用方法 *java.net.URLConnection*() 、 (*java.net.URL*)*set/init*() 、(*HttpClient*)*execute*()等作为 Sink。

4.1 隐私泄露情况统计分析

经过实验分析, 本文方法检测出了 365 个存在潜在信息泄露的 App, 而 FlowDroid 报告了 324 个泄露情况。表 3 展示了本文方法与 FlowDroid 方法检测实验的对比结果。

表 3 信息泄露情况统计

Table 3 Information disclosure statistics

泄露资源	静态污点分析方法		FlowDroid	
	真正泄露	误报	真正泄露	误报
位置	191	32	165	30
电话状态	87	11	66	11
录音	39	5	49	3
总数	317	48	280	44

(1) 位置和电话状态的检测准确率优于 FlowDroid。由于本文方法在检测之前梳理了这两种敏感权限及其对应 API 之间的映射关系, 建立了函数调用图, 因而 Source 和 Sink 的选择更具针对性。FlowDroid 是对 App 的数据路径进行全面覆盖检测, 其中包含无关路径, 导致检测精度有所降低。

(2) 录音权限的检测准确率略低于 FlowDroid。分析原因是, 在确定 Source 时有所遗漏, 部分方法路径没有覆盖, 导致污点分析结果的准确率有所降低, 在后续的研究中将继续改进。

(3) 误报分析。对本文方法中的 48 个误报案例进行了人工分析, 产生误报的原因为: (1)污点分析过程中计算前向切片时覆盖的上下文信息过多, 导致识别内容错误; (2)部分合法的网络访问数据在检测过程中被误判为泄露, 如连接网络获取必要数据的方法调用。

4.2 检测时间对比分析

FlowDroid 在检测过程中, 由于需要构建 App 的生命周期模型, 从而会占用一定时间。此外, 样本的大小也会影响 FlowDroid 的检测时间, 一般情况下, 使用 FlowDroid 检测的最短运行时间约为 5 秒, 最长约为 71 秒, 平均耗时 16 秒。本文所提方法在检测 1500 个 App 时平均耗时 4.8 秒, 远小于 FlowDroid 的检测时间, 因此本文方法可用于对大规模 App 进行隐私泄露检测。

综上所述, 本文方法利用权限和 API 之间的映射关系检测出了更多的隐私信息泄露行为。与 FlowDroid 方法相比, 在检测准确率略有提高的同时, 检测耗时缩短, 效率明显提升。

5 总结

作为目前最大的移动平台, Android 更易成为恶意攻击者的目标。隐私泄露检测能够有效地保护用户的隐私, 增强 Android 系统的安全性, 因而具有较好的实用价值。本文提出了一种基于静态污点分析的 Android 隐私泄露检测方法, 结合位置、电话状态以及录音三种权限信息, 通过分析权限与 API 之间的映射关系, 来创建 App 与敏感权限相关的函数调用图, 从中寻找可疑的路径, 并采用静态污点分析的方法检测隐私泄露行为。通过实验, 验证了本文方法的有效性。本文在 Sink 选取时, 仅考虑了网络接口, 尚未涉及字符流输出((*OutputStream*)*write*()、(*java.io.Writer*)*write*() 等) 、 文 件 流 输 出((*FileOutputStream*)*write*()等)、配置数据等其他也会产生信息泄露行为的输出方式。因此, 下一步工作将在本文检测方法的基础上, 选取更多发送敏感信息的函数调用方法, 检测和挖掘 App 中更多潜在的隐私信息泄露行为, 进一步提高检测的效率。

参考文献

[1] IDC Media Center. 智能手机市场最新预测报告[R/OL]. *IDC*, 2019-09-09. https://www.idc.com/getdoc.jsp?containerId=prUS45487719.

[2] 中国互联网络信息中心. 第 43 次《中国互联网络发展状况统计报告》[R/OL]. *中国网信网*, 2019-02-28. http://www.cac.gov.cn/2019-02/28/c_1124175677.

[3] 艾媒前沿科技产业研究中心. 2020 中国手机 App 隐私权限测评报告[R/OL]. *艾媒咨询*, 2020-02-25. https://www.iimedia.cn/c400/69301.html.

[4] Sadeghi A, Bagheri H, Garcia J, et al. A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software[J]. *IEEE Transactions on Software Engineering*, 2017, 43(6): 492-530.

[5] Zhu Dali, Jin Hao, Wu Di, et al. Android malware detection method based on data-flow deep learning algorithm[J]. *Journal of Cyber Security*, 2019, 4(2): 53-68.
(朱大立, 金昊, 吴荻, 等. 基于数据流深度学习算法的 Android 恶意应用检测方法[J]. *信息安全学报*, 2019, 4(2): 53-68.)

[6] Zhang L, Zhu D L, Yang Z M, et al. A Survey of Privacy Protection Techniques for Mobile Devices[J]. *Journal of Communications and Information Networks*, 2016, 1(4): 86-92.

[7] Yue H Z, Zhang Y Q, Wang W J, et al. Android Static Taint Analysis of Dynamic Loading and Reflection Mechanism[J]. *Journal of Computer Research and Development*, 2017, 54(2): 313-327.
(乐洪舟, 张玉清, 王文杰, 等. Android 动态加载与反射机制的静态污点分析研究[J]. 计算机研究与发展, 2017, 54(2): 313-327.)

[8] Wang L, Li F, Li L, et al. Principle and Practice of Taint Analysis[J]. *Journal of Software*, 2017, 28(4): 860-882.
(王蕾, 李丰, 李炼, 等. 污点分析技术的原理和实践应用[J]. 软件学报, 2017, 28(4): 860-882.)

[9] Enck W, Gilbert P, Han S, et al. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones[J]. *ACM Transactions on Computer Systems*, 2014, 32(02): 1-29.

[10] He Yongzhong, Hu Binghui, Han Zhen. Dynamic Privacy Leakage Analysis of Android Third-Party Libraries[C]. *IEEE 1st International Conference on Data Intelligence and Security*, 2018: 275-280.

[11] Bai J Y, Wang W P, Qin Y, et al. BridgeTaint: A Bi-Directional Dynamic Taint Tracking Method for JavaScript Bridges in Android Hybrid Applications[J]. *IEEE Transactions on Information Forensics and Security*, 2019, 14(3): 677-692.

[12] Arzt S, Rasthofer S, Fritz C, et al. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps[C]. *the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2013: 259-269.

[13] Bohluli Z, Shahriari H.R. Detecting Privacy Leaks in Android Apps using Inter-Component Information Flow Control Analysis[C]. *IEEE 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology*, 2018: 1-6.

[14] Tian C, Xia C L, Duan Z H. Android Inter-component Communication Analysis with Intent Revision[C]. *the 40th International Conference on Software Engineering: Companion*, 2018: 254-255.

[15] Cao Hongsheng, Jiao Jian, Li Denghui. A Static Analysis Model for Implicit Information Leakage in Android Application[C]. *IEEE 18th International Conference on Communication Technology*, 2018: 1-8.

[16] Wang L, Zhou Q, He D J, et al. Multi-source Taint Analysis Technique for Privacy Leak Detection of Android Apps[J]. *Journal of Software*, 2019, 30(2): 211-230.
(王蕾, 周卿, 何冬杰, 等. 面向 Android 应用隐私泄露检测的多源污点分析技术[J]. 软件学报, 2019, 30(2): 211-230.)

[17] Zhang Jie, Tian Cong, Duan Zhenhua. FastDroid: Efficient Taint Analysis for Android Applications[C]. *ACM 41th International Conference on Software Engineering: Companion Proceedings*, 2019: 236-237.

[18] Wang L, He D J, Li L, et al. Sparse Framework Based Static Taint Analysis Optimization[J]. *Journal of Computer Research and Development*, 2019, 56(3): 480-495.

(王蕾, 何冬杰, 李炼, 等. 基于稀疏框架的静态污点分析优化技术[J]. 计算机研究与发展, 2019, 56(3): 480-495.)

[19] Qing Sihan. Research Progress on Android Security[J]. *Journal of Software*, 2016, 27(1): 45-71.
(卿斯汉. Android 安全研究进展[J]. *软件学报*, 2016, 27(1): 45-71.)

[20] Au K W Y, Zhou Y F, Huang Z, et al. PScout: Analyzing the Android Permission Specification[C]. *the 2012 ACM conference on Computer and communications security*, 2012: 217-228.

[21] Lam P, Bodden E, Lhotak O, et al. The Soot Framework for Java Program Analysis: A Retrospective[J]. *In Cetus Users and Compiler Infastructure Workshop*, 2011, 15: 1-8.

胡英杰 于 2017 年在湘潭大学软件工程专业获得学士学位。现在新疆大学软件工程专业攻读硕士学位。研究领域为信息安全、隐私保护。研究兴趣包括: 隐私泄露检测等。Email: 623547519@qq.com

赵楷 于 2011 年在武汉大学计算机软件与理论专业获得博士学位。现任新疆大学信息科学与工程学院计算机系主任。研究领域为语义软件开发、容错计算等。研究兴趣包括: 语义 Web、Android 安全等。Email: zhawkk@xju.edu.cn

于媛尔 于 2017 年在新疆大学食品科学与工程专业获得学士学位。现在新疆大学计算机技术专业攻读硕士学位。研究领域为信息安全、Android 平台安全。研究兴趣包括 Android 恶意软件检测与家族分类。Email: 479133518@qq.com

张琳琳 于 2009 年在武汉大学计算机与软件理论专业获得博士学位。现任新疆大学信息科学与工程学院信息安全系主任。研究领域为移动应用安全、软件安全。研究兴趣包括: 恶意软件检测, 隐私保护等。Email: zllnadasha@126.com

方文波 于 2016 年在商洛学院计算机科学与技术专业获得学士学位。现在新疆大学软件工程专业攻读硕士学位。研究领域为恶意软件检测。研究兴趣包括: 机器学习、深度学习、人工智能。Email: 1210314799@qq.com