面向 Android 第三方库的共谋行为检测

张 婧 李瑞轩 汤俊伟 韩洪木 辜希武
(华中科技大学计算机科学与技术学院 武汉 430074)

摘 要 第三方库是安卓应用重要的组成部分,应用开发者往往会引入一些具有特定功能的第三方库进行快速开发。针对 Android 第三方库中存在的共谋风险,提出了面向 Android 第三方库的共谋行为检测的研究。Android 第三方库与应用属于不同的利益体,隐藏在第三方库中的通信行为可以视为应用共谋的一种特殊情况,同样会引发权限提升、组件劫持、性能消耗等恶意行为,这些行为可以引起过多的系统消耗,甚至是引发安全威胁。文中对近些年来国内外学者在该研究领域取得的成果进行了系统总结,给出了研究的共谋定义,并对 Android 第三方库共谋行为可能产生的风险威胁进行了分析。然后详细介绍了安卓第三方库共谋行为检测的设计方案。针对测试集中的 29 个第三方库的实验表明,所提设计方案的精确率达到了 100%,召回率为 89.66%,F-measure 值为 0.945;同时,本实验还对下载的 1207 个第三方库进行了分析,对 41 个国内著名的第三方库非敏感信息共谋行为导致的资源消耗情况进行了验证。最后,对工作进行了总结,并对未来研究进行了展望。

关键词 安卓第三方库,敏感路径,组件通信,应用共谋
中图法分类号 TP309 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.05.013

Collusion Behavior Detection Towards Android Third-party Libraries

ZHANG Jing LI Rui-xuan TANG Jun-wei HAN Hong-mu GU Xi-wu
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract Third-party library is an important part of Android applications. Application developers often introduce some third-party libraries with specific functions for rapid development. Concerning the risk of collusion in Android third-party libraries, this paper studied the collusion of Android third-party libraries. Android third-party libraries and applications belong to different interests. Communication behaviors hidden in third-party libraries can be considered as a special case of application collusion, and it will also lead to privilege escalation and component hijacking. Furthermore, these behaviors can cause excessive system consumption, and even trigger security threats. This paper presented a systematic survey of existing research achievements of the domestic and foreign researchers in recent years. First, this paper gave the definition of collusion, and analyzed the risks of the collusion behavior in Android third-party libraries. Then, it presented the design of the Android third-party library collusion behavior detection system in detail. For the 29 third-party libraries in the test set, the experiment shows that the accuracy of this design is 100%, the recall rate is 89.66%, and the F-measure value is 0.945. At the same time, the downloaded 1207 third-party libraries were analyzed. The experiments also verify the resource consumption caused by non-sensitive information collusion behavior of 41 domestic famous third-party libraries. Finally, this paper concluded the work and gave a perspective of the future work.

Keywords Android third-party library, Sensitive path, Inter-component communication, Application collusion

随着移动智能终端性能的不断提升以及功能的日益多样化,其用户规模不断增大,根据中国互联网络信息中心(CNNIC)近日发布的第 41 次《中国互联网络发展状况统计报告》[1]显示,截至 2017 年 12 月,我国手机网民规模达 7.53 亿,由 2016 年的 95.1%提升至 97.5%,较 2016 年底增加了 5734 万人,其中安卓系统的智能终端占主导地位,占比达到 59%。

Android 应用内普遍使用了第三方库来简化开发。研究显示,45%的应用项目引用了外部的项目源码,Google Play 中有超过 60%的应用引入了广告库,并且部分应用使用了超过 20 个第三方库[2-3],因此一般应用由主程序和第三方库组成,是多个利益体代码的集合。第三方库在为开发者带来好处的同时也带来了一定的安全风险——恶意的第三方库提供

到稿日期:2018-10-01 返修日期:2018-12-02 本文受国家重点研发计划(2016YFB0800402,2016QY01W0202),国家自然科学基金项目(U1836204,61572221,61433006,U1401258,61502185),国家社科基金重大项目(16ZDA0092),广西高等学校高水平创新团队-数字东盟云大数据安全与挖掘技术创新团队资助。

张 婧(1994-),女,硕士生,主要研究方向为移动安全,E-mail:zhangjing94@hust.edu.cn;李瑞轩(1974-),男,博士,教授,CCF 杰出会员,主要研究方向为移动安全、系统安全、区块链,E-mail:rxli@hust.edu.cn(通信作者);汤俊伟(1990-),男,博士生,主要研究方向为移动安全;韩洪木(1980-),男,博士生,主要研究方向为移动安全;辜希武(1967-),男,硕士生,主要研究方向为移动安全。

者以及一些恶意攻击者将恶意代码封装在第三方库中，当多种应用使用这些库时，恶意行为大范围传播，从而导致大规模的应用污染，并且 Android 安全模型是以应用为单位来进行权限管理[4-5]的。

应用市场对应用监管检查越来越严格，单个应用的危险程度得到了控制。于是，将功能分散到多个应用中，当这些应用同时在设备中使用时，实现恶意行为的共谋攻击成为热点。应用共谋，通俗来讲就是两个或多个应用有目的地串通，从而实施一些协作行为[6-8]。目前，对于共谋攻击的研究更多的是基于应用的研究，仅有文献[9]提出相同库之间的共谋将会给用户带来更大的风险，而且通过实验发现应用中的共谋攻击可能来自第三方库，并且现有的应用共谋检测技术并不能完全检测出存在于第三方库中的共谋行为。基于这种情况，本文以第三方库作为输入进行检测，基于 Flowdroid 实现一个能对第三方库进行敏感路径分析的工具，并对库进行关联分析，找出潜在的共谋行为。

本文主要的贡献有：

1)提出了一种多个安卓第三方库协作的攻击(Inter-Library Collusion)。为了扩大攻击的范围以及获取更多的信息，攻击者可能会将恶意代码分散在不同的第三方库中，通过功能分散实现信息窃取攻击。

2)首次提出对安卓第三方库的共谋行为进行检测，利用数据流分析找出第三方库中潜在的共谋通信路径，并且可以找出所有能够触发敏感路径的 API。应用开发者在使用第三方库时，更多的是关心自己所需要的功能，可能并不会检查库中代码的实现，而且现在越来越多的代码会进行混淆和加密，使得阅读存在一定的困难。敏感 API 的定位有助于开发者对第三方库进一步审查。

3)提升了第三方库使用的安全性，对风险实现更精准的追责。通过对应用进行共谋分析只能分析出应用中是否存在风险，但是没有区分风险产生的来源。对第三方库进行分析，能够让开发者在使用前审查库行，出现问题时也能更准确地追责。

1 相关工作

目前对共谋攻击的研究主要集中在对威胁的检测发现上，从分析方法的类别上看主要有静态检测和动态分析两种。静态检测方法包含基于应用组件间通信信息的静态分析、基于信息流图的静态分析等；动态分析方法则主要是通过访问控制策略实现对应用间通信的实时监测与控制。

文献[10]将应用间的通信分析转化为应用内的组件间的通信分析。首先通过反编译得到应用的 manifest 文件和 smali 文件，解决应用之间的命名冲突后将两个应用合并成一个应用，然后使用已有的方法检测组件间通信以实现对应用间共谋的检测。这种方法使得一些现有的分析单个应用隐私泄漏的方法都能应用到应用间隐私泄漏的检测中，但其缺点是存在较高的误报率。

FUSE[11]首先对单个应用进行分析，生成对应的内部信息流图，接着结合所有应用的分析信息，生成多个应用信息流图，然后利用共谋应用的特点，制定特殊的安全策略，对多个应用信息流图进行分析，检测违反安全策略的潜在风险。

文献[12]按照一些规则自动化构造测试用例，将测试用例发送给待检测应用程序，通过分析测试过程中的交互情况以及输出结果，实现对 Android 应用组件间通信的鲁棒性检测。通过对测试数据的分析发现发送特殊 Intent 可以导致应用程序的崩溃，甚至引发系统服务的级联崩溃。

Asavoae 等[13]分析了共谋应用对之间的通信组件，对通信组件进行通用规律总结，生成模板代码段，随后通过注入模板代码生成共谋应用对。利用自动化生成共谋应用对的技术，该团队在文献[14]中进一步提出了将机器学习的方法应用于共谋检测，并设计了两种检测方法：一种是基于 Prolog 的规则识别共谋检测，另一种是基于数据分析的共谋路径检测。实验应用机器学习方法进行训练、验证、测试，分析的数据集中包含 9000 个恶意应用和 9000 个良性应用，增加了已有研究分析共谋攻击的数据量。

Amandroid[15]通过构建组件间的数据流图实现对组件间通信的检测，通过与 Flowdroid 和 Epicc 的对比得出，Amandroid 可以解决这些现有工具无法处理的组件间通信而导致的各种安全问题。但是该工具是针对单个应用的分析，本质上只能分析应用内组件间通信的问题。

大多数应用共谋的分析都是将应用两两组队进行分析，复杂度为 $O(n^2)$。为了降低分析复杂度，文献[16]利用改进后的 IC3[17]分析单个应用以提取 Intent 信息，利用 Flowdroid 分析应用中的敏感路径，最终通过关系型数据库存储所有的信息，并利用 SQL 语句进行应用间的关联分析。该项研究分析了 110150 个 APP，提高了已有研究的分析量级，并且将分析应用间共谋的复杂度降低至 $O(mN)$。

由于应用间共谋行为是由多个应用产生的，在运行时检测方面的研究主要集中在修改安卓系统以实现对共谋行为的实时监控以及防控。XmanDroid[18]采用基于运行时系统策略的动态分析方法对 Android 系统进行拓展，通过运行监控器监控应用运行状态并根据策略检查判断是否可以建立组件间通信。XmanDroid 实现了应用级别权限提升攻击的监测机制，能够有效地防止运行时权限的提升。更多地，应用开发者需要谨慎使用“exported”属性和隐式 Intent，防止被恶意开发者利用。

LinkDroid[19]提出了一种动态可链接图的概念(Dynamic Linkability Graph，DLG)。首先监控应用运行时的应用级可连接性，然后动态更新 DLG，基于这种连接图的概念，分析应用间的通信、可连接情况，为用户实时分析潜在共谋风险的情况。

无论是静态检测还是动态分析 ，目前大多数共谋研究都是针对应用进行的，但是 Android 第三方库作为应用的重要组成部分，来自于不同的利益体，针对应用检测的现有公开技术都不能很好地检测第三方库中的共谋行为，因此设计一种可以高效检测 Android 第三方库中的共谋行为的方法，提高第三方库的安全性进而提高应用的安全性，具有重要意义。

2 风险模型

由于安卓的开源性，网络上存在大量开源的第三方库，开发者为了简化开发以实现一些复杂的功能，会引用多个第三方库，直接利用库实现相关功能。恶意的开发者为了获得用户的隐私信息，将获取敏感信息的恶意代码注入到流行的第

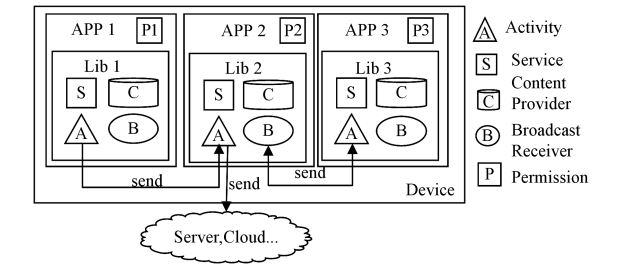
三方库中,调用这些风险库中的 API 时触发风险,导致用户隐私泄漏,而当这些库被一些流行应用广泛使用后,将会造成非常大的影响。例如,理论上存在一类共谋库通过通信达成权限互补,应用 A 可以获得位置信息,应用 B 可以联网,A 中的库便可以获取设备的位置信息并将其发送给 B 中的协作库,利用 B 的联网权限上传至服务器。

本节主要介绍了 Android 第三方库共谋行为的概念,分析了本研究对共谋行为的定义以及 Android 第三方库共谋行为可能产生的风险威胁。

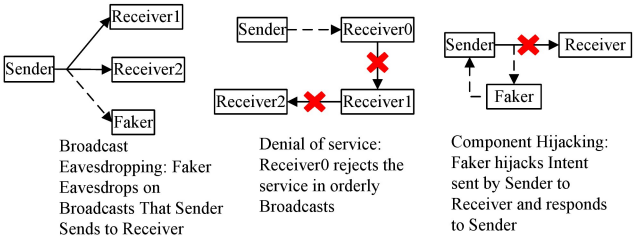
2.1　共谋风险

第三方库共谋可能主要带来以下几种风险:

1)权限提升。APP 1,APP 2 分别拥有权限 P1 和 P2,通过应用间的通信,APP 1 获得了只有拥有权限 P2 才能得到的数据,此时 APP1 的权限就得到了提升,具体的攻击场景如图 1(a)所示。图中包含两种场景:①对于拥有相同签名的应用,根据 Android 的沙箱机制,它们会运行在同一个沙箱中,可以共享权限和一些其他的资源;②拥有不同签名的应用通过组件间通信,Lib 1 将只能被拥有权限 P1 的应用获取的信息传递给 Lib 2。



(a)Privilege escalation and privacy leaks



(b)Component hijacking

图 1　共谋风险示例

Fig. 1　Example of collusion attacks

2)隐私泄漏。应用可以利用组件间的通信传递获取隐私信息,并最终将其泄漏到设备之外。例如 APP 1 拥有读取联系人的权限,APP 2 拥有联网的权限,Lib 1 将获取到的信息传递给 Lib 2,Lib 2 利用联网功能将信息传递到设备之外。

3)Intent 欺诈。Android 应用的组件通过设置“exported”属性分为私有和公有两类。设置为 false 的组件为私有组件,只允许具有相同 uid 的应用调用。而设置为 true 的共有组件,允许接收其他应用发送的 Intent,一般称之为暴露组件。恶意应用发送 Intent 给暴露组件就可能导致 Intent 欺诈,Activirty,Service 都有可能导致此类攻击。例如,对于暴露的 BroadcastReceiver,假如其盲目地接收任何广播,它有可能会得到广播中的恶意数据,并在应用内扩散;恶意暴露的 Activity 可以被任意应用调用,存在用户调用恶意应用的可能性,如图 1(b)所示。

4)组件劫持。恶意应用可以拦截源应用和目标应用间的通信,并且有可能代替目标应用进行响应。

5)性能损耗。类似 BroadcastReceiver,Service 的组件一般在后台运行,一般情况下用户无法感知或管理,此种情况虽然不会有敏感信息泄漏等危害,但设备中若存在此类组件,大量的后台进程会影响设备的性能,进而影响用户的使用感受,也会产生一定危害。

2.2　相关定义

狭义上的共谋攻击将应用间的合谋定义为开发者有意的行为,恶意开发者开发多个应用并利用这些应用;广义上的共谋攻击则包含了恶意开发者利用其他应用存在的一些缺陷进行共谋攻击。现有研究更多的是关注于广义的共谋攻击,因为分析的过程中很难区分合作行为是否是双方协同。

本研究主要考虑安卓特有的组件间通信构成的共谋,为了更加准确地检测 Android 第三方库中的共谋行为,对库共谋行为的定义主要考虑敏感数据使用的共谋和非敏感数据共谋两类。

在进行共谋分析时,主要考虑的风险是对敏感数据的使用,具体定义如下:

1)存在一个 Android 第三方库的集合 L,A 和 B 是两个库,$A,B\in L$;

2)在 A 中存在 $flowOut=\{\text{Sender}\leftarrow\text{Source}\}$,Source 是敏感权限的调用 API,获取到敏感数据后,将数据传递到 Sender,Sender 是启动组件的函数,也是信息发送点,可以通过参数传递包含的信息,例如 startBroadcast(Intent i)就是一个携带敏感信息的 Sender;

3)在接收方中存在 $flowIn=\{\text{Sink}\leftarrow\text{Receiver},\text{component}\}$,component 是可以响应 A 的调用的组件,Receiver 表示 component 中可以获取 Sender 传进来的值的函数,例如 getIntent(),然后 Sink 可以使用获得的数据实现隐私泄漏。

这种情况下,A 和 B 就是共谋的,如图 2 所示。

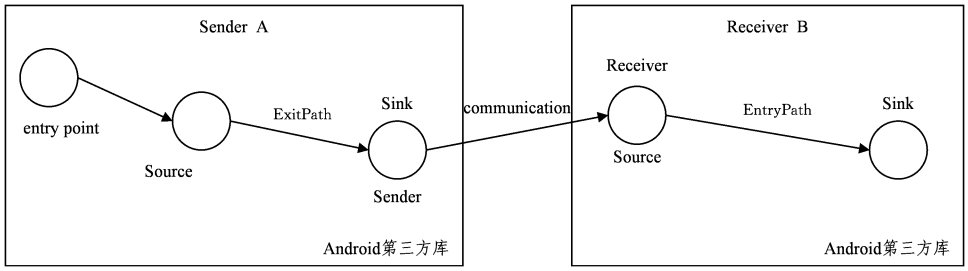


图 2　共谋行为示例

Fig. 2　Example of collusion behaviors in libraries

图 2 中，entry point 表示触发 ExitPath 的 API，对库而言，其可能是一个 Android 组件类，也可能是普通的类中的方法；发送方中的 Source 点指获取敏感信息的点，主要是敏感权限类的 API，例如获取位置信息的 getLastKnownLocation()、获取手机号的 getLine1Number()等；发送方中的 Sink 点指敏感信息的发送点 Sender，使用 sendBroadcast()等函数将 Source 点获得的敏感信息发送出去，有时候也叫作 exit point；EexitPath 表示发送方中获取敏感数据并发送的输出路径；接收方中的 Source 点指接收 exit point 发送的消息的 API，有时也称作 entry point(并不是数据流分析的入口)；接收方中的 Sink 点指从 entry point 接收到的消息的使用点，例如写入文件 write()、发送消息 sendTextMessage()等；EntryPath 表示接收发送方的数据并使用数据的输入路径。

除了敏感信息的传输可以造成共谋，基于组件的非敏感信息通信也会带来一定的危害，例如在用户不可见的情况下后台进行大规模频繁的组件通信可以实现对设备的高损耗，影响用户的使用，达到攻击的目的。因此，本文也对此类情况进行了检测，有如下具体定义：

1)存在一个 Android 第三方库的集合 L，A 和 B 是两个库，$A,B\in L$；

2)在 A 中存在 $flowOut=\{\text{Sink}\}$，Sink 是启动组件的函数，例如 startService(Intent i)；

3)在 B 中存在 $flowIn=\{\text{component}\}$，component 是可以响应 A 调用的组件。

这种情况下，A 可以将 B 中的组件唤醒运行，也是共谋的一种情况，过多的这种操作会对设备内存等性能产生损耗，虽然没有敏感数据的泄漏，但也会造成一定程度的危害。

需要注意的是，不同的应用中可能会引用相同的第三方库，因此会存在不同应用间相同第三方库共谋以及不同应用间不同第三方库共谋两种情况，如图 3 所示。

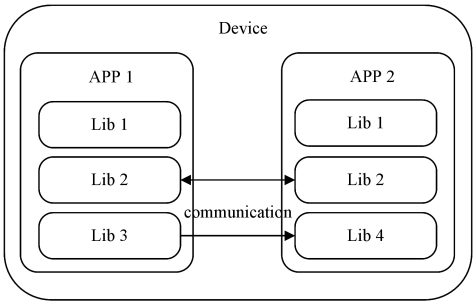


图 3 第三方库共谋示例

Fig. 3 Example of Android third-party libraries collusion

3 风险检测模型

本方案主要为实现对安卓第三方库共谋行为的检测。为了检测出完整的共谋通道，需要检测出每个第三方库中的 EntryPath、ExitPath、Intent 信息、IntentFilter 信息、URI 信息，然后根据不同组件的通信规则进行匹配，最终检测出完整的共谋通道。该方案设计了一个面向 Android 第三方库的共谋行为检测系统，主要包括预处理、静态检测、策略匹配 3 个模块，其系统架构如图 4 所示。

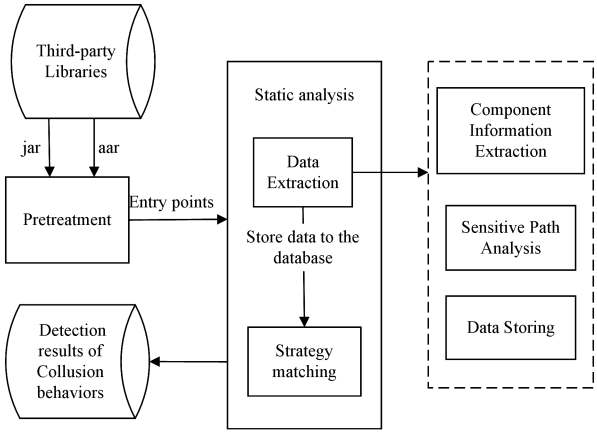


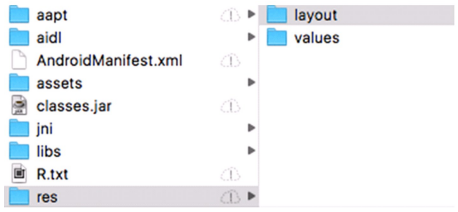
图 4 系统架构

Fig. 4 Structure of detection system

3.1 预处理模块

预处理模块主要对输入进行格式化处理，并且分析 Android 第三方库中可以作为数据流分析的 entry points。Android 第三方库主要有两种存在形式：".jar"文件和".aar"文件，两类文件在内容和格式上都有所不同，如图 5 所示。

(a)File structure of decompressed ".jar"



(b)File structure of decompressed ".aar"

图 5 ".jar"文件和".aar"文件的结构对比

Fig. 5 Structure comparison between ".jar" and ".aar"

".aar"文件经过解压后含有资源文件和 classes.jar 文件，而 classes.jar 其实就是代码编译后的文件集合，当相同的库项目分别打包成".jar"文件和".aar"文件时，aar 文件解压后的 classes.jar 与".jar"文件的内容应该是相同的。因此，针对两种不同的输入，需要进行不同的处理，预处理模块主要有以下步骤：

1)根据输入文件的路径、文件名信息，提取第三方库的名称、类型、版本信息，这些信息可以使分析结果更加详细。

2)如果待检测文件为 aar 库，首先解压文件到目标文件夹，得到 classes.jar，AndroidManifest.xml，res 等文件/文件夹。由于需要从 resources.arsc 文件获取数据分析模块使用的回调函数，因此会使用 aapt 工具，利用命令将 res 文件夹打包成 resources.arsc 文件；如果待检测文件为 jar 库，直接跳转至步骤 3)。需要注意的是，下载的数据集中存在既有 aar 又有 jar 的情况，此时会优先分析".aar"文件，因为 aar 中的信息更加详细，分析结果会更准确。

3)对".jar"文件进行 entry points 解析。采用 Soot 对 jar 进行分析,首先将 class 转化为 jimple,然后通过分析类的继承和权限信息、方法的权限信息等对 entry points 进行提取。输出 Android 组件类的类名集合和普通 java 类可调用方法的方法签名(类名:方法名)集合。

3.2　数据提取模块

数据提取模块是本系统的核心,主要功能是从 Manifest 和代码中提取 Intent 和 IntentFilter 信息,以及进行数据流分析以获取敏感路径。整个数据提取模块主要包含组件信息提取和敏感路径分析两部分内容。

组件信息提取主要是从 manifest 配置文件和代码中提取组件相关的 Intent,IntentFilter,URI 信息。Android 组件分为静态注册和动态注册两种,在 AndroidManifest. xml 文件中配置的所有组件都属于静态注册,动态注册是通过相关 API 在代码中运行时注册,目前只有广播接收器可以进行动态注册。本研究利用改进后的 IC3 对每个库进行组件信息分析,主要利用 AXMLPrinter2 解码 AndroidManifest. xml 配置文件,获取配置文件中注册的组件列表、组件的 intent-filter 标签值、组件的 exported 等属性值、申请的 permission 等信息、利用 IC3 的核心算法获取程序 exit points 中的 Intent 参数值(例如 sendBroadcast()和 startService()等关联的 Intent)、动态注册的 BroadcastReceiver 的 IntentFilter 信息、及 ContentProvider 使用的 URI 信息。

敏感路径分析部分通过改进 Flowdroid,使其可以对第三方库进行分析。由图 2 可知,本方案会针对 ExitPath 和 EntryPath 分别配置不同的 Source 和 Sink 列表。Susi 是一个通过分析 Android 系统源码,然后利用机器学习自动生成 Source 和 Sink 列表的工具,其准确率和覆盖率超过 92%。根据现有的研究以及实验,本研究拟采用 Susi 生成的 Source 和 Sink 列表,总结可以形成共谋行为的 Source 点(有 255 个),包括设备号、浏览器书签、网络信息、位置等信息相关函数;Sink 点(exit point)有 94 个;而对于接收方,Source 点(entry point)有 52 个;Sink 有 175 个。部分函数列表如具体的 Source 和 Sink 参考表 1。

表 1　Source 和 Sink 点示例

Table 1　Sources and Sinks of ExitPath and EntryPath

	Source	Sink
ExitPath	〈android. telephony. TelephonyManager: String getSimSerialNumber()〉	〈android. content. ContextWrapper: void sendBroadcast (Intent)〉
	〈android. telephony. TelephonyManager:String getDeviceId()〉	〈android. content. Context:void sendBroadcast(Intent,String)〉
	〈android. location. Location:double getLongitude()〉	〈android. content. Context:void sendBroadcast(Intent)〉
	〈android. accounts. AccountManager: Account[] getAccounts()〉	〈android. content. ContextThemeWrapper: void sendBroadcast (Intent)〉
	〈android. provider. Browser: android. database. Cursor getAllBookmarks()〉	〈android. content. Context:void startActivities(Intent[])〉
	…	…
EntryPath	〈android. content. Context:Intent getIntent()〉	〈java. net. Socket:void connect(SocketAddress)〉
	〈android. content. ContextWrapper: void onActivityResult(int, int,Intent)〉	〈org. apache. http. client. HttpClient:org. apache. http. HttpResponse execute(HttpUriRequest)〉
	〈android. content. Intent:Bundle getBundleExtra (String)〉	〈com. android. server. WifiService:void setCountryCode(String, boolean)〉
	〈android. content. Intent:android. net. Uri getData()〉	〈android. app. Activity:void setResult(int,Intent)〉
	…	…

Android 应用由不同的组件构成,Android 系统为这些组件定义了完整的从创建到销毁的生命周期,所有的组件类继承于系统类(ContextWrapper 等),开发者在实现某个组件时,需要重写组件的生命周期回调函数。以 Activity 的生命周期为例,当调用 onCreate 时会创建 Activity;当调用了 onStart,Activity 才处于前台用户可见状态;当调用 onStop 时,Activity 变成后台运行;当调用 onRestart 时,Activity 会再次回到前台可见。而这些调用都是在 Android 底层实现的,并不需要开发者主动调用这些函数,因此在对 Android 组件类进行数据流分析之前,会分别对所有组件的生命周期进行细致的建模,在 dummyMain 函数中模拟各个生命周期函数的调用。

在本研究中,由于分析对象是 Android 第三方库,数据流入口可能是 Android 组件也可能是普通的 Java 函数,因此不仅需要对 Android 组件进行建模,还需要对普通的函数调用进行建模。对 Java 函数构造 dummyMain 不需要模拟回调函数的调用,直接在 dummyMain 中插入对相关函数的调用语句即可。

根据第三方库的特性,Android 组件类和普通类都可能是调用入口,例如图 6 所示的情况,存在多个可以触发敏感通信的入口 API。本研究所提方案不仅能获取 Source-Sink 路径,也能够得到触发敏感路径的 API 信息。

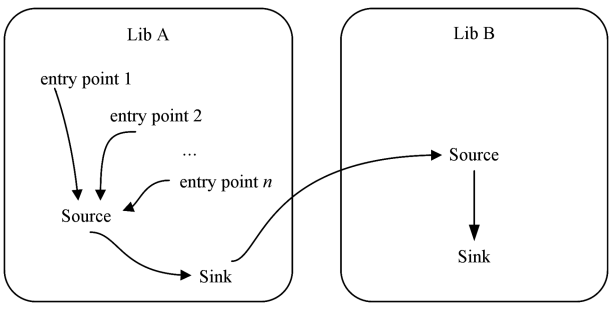


图 6　第三方库的数据流示意图

Fig. 6　Data-flow example of third-party library

3.3 数据分析模块

因此，对于策略匹配模块，只需要通过分析组件的 Intent，IntentFilter，URI 等信息的通信匹配策略，并利用这些特征分析数据就可以得到最终的第三方库共谋分析结果。下面将具体分析不同组件的共谋行为匹配策略。

1）Activity，Service，BroadcastReceiver 组件通信

Activity，Service，BroadcastReceiver 3 种组件都是通过发送方 Intent 信息来启动符合条件的目标组件，主要有两种实现方式：利用 Intent 的 Component 属性进行显式调用，通过 Action，Data，Category 进行隐式调用，这两种方式都是同接收方 IntentFilter 信息进行匹配来决定要启动的接收组件。Intent 一般在调用发送函数时在代码中配置，IntentFilter 一般在 manifest 文件中声明组件时进行配置。下面将详细介绍共谋行为的匹配规则。

①显式调用组件通信。通过 Component 属性则意味着进行显式调用，直接配置 Intent 要启动的目标组件的类名。在所有属性中，Component 属性具有最高优先级，即如果 Component 已经配置了，会直接查找符合条件的组件，而不再查找符合 Action，Data 等属性值条件的组件。以图 2 为例进行说明，显式 Intent 通信的共谋行为条件如下：a）库 A 中存在 ExitPath，并且 ExitPath 的 Sender 点携带的 Intent 配置了 Component 属性；b）库 B 中存在 EntryPath，并且 EntryPath 的 Receiver 点所在类为组件类，其类名信息与 a）中库 A 的 Component 属性值相等。此时，可以得到库 A 与库 B 之间有一条敏感信息传递通路，存在共谋行为。

②隐式调用组件通信。当组件通过 action 等属性进行匹配通信时，共谋条件如下：a）库 A 中存在 ExitPath，并且 ExitPath 的 Sender 点携带的 Intent 配置了 Action 等属性；b）库 B 中存在 EntryPath，并且 EntryPath 的 Receiver 点所在类为组件类，该组件类配置了 IntentFilter 信息；c）在 a）中 Intent 的 Action，Category 等信息与 b）中 IntentFilter 对应属性的值完全匹配。同样，此时得到库 A 与库 B 之间的共谋信息。需要特别说明的是，由于库是应用中的一部分，一个应用可以引用多个第三方库，因此对库做共谋分析没有关注是应用内还是应用间，两种情况都有可能存在，本文所提方法既可以检测应用间的库共谋也可以检测应用内的库共谋，并且共谋行为可以来源于不同的库之间，也可以来源于不同应用相同的库之间。

2）ContentProvider 组件通信

ContentProvider 是对文件进行操作的组件，用 URI 代表操作的文件，因此只要 URI 相同，就可以访问相同的数据。a）库 A 中存在 ExitPath，并且 ExitPath 的 Sender 点是通过 ContentProvider 操作指定 URI 代表的数据；b）库 B 中存在 EntryPath，并且 EntryPath 的 Receiver 点是通过 ContentProvider 操作指定 URI 代表的数据获取信息；c）步骤 a）和步骤 b）中操作的是相同的 URI。URI 表明 ContentProvider 组件操作的数据，当 a）和 b）中操作的是相同的 URI 时，表明两个 ContentProvider 操作的是相同的文件。此时可以得到库 A 与库 B 之间有一条敏感信息传递通路，存在共谋行为。

4 实验设计与结果分析

4.1 测试数据集的可靠性验证

由于本研究是第一个提出对 Android 第三方库进行共谋行为研究的，因此并没有公认的数据集可用于检测系统的效果。为了使数据集具有说服力，研究的数据集将由两部分构成，一部分将通过分析已存在的应用共谋数据集的源码（主要是 DroidBench[1] 和 ICC-Bench[2] 2 个数据集），然后逐一对照这两个数据集构造库共谋的数据集，最终分别构造 17 个库和 7 个库；另一部分数据集是分析共谋过程中自己构建的一部分，一共实现了 8 个库。

本研究主要是针对利用安卓组件进行通信的风险检测，上述数据集已包含所有可行的通信用例，具体组件通信 API 如表 2 所列。

表 2 组件通信关键 API 列表

Table 2 API list for component communication

关键 API	接收方
sendBroadcast(Intent i)； sendBroadcast(Intent i,String recePermission) sendOrderedBroadcast(Intent i,String recePermission,…)； sendOrderedBroadcast(Intent i,String recePermission)； sendStickyBroadcast(Intent i)； sendStickyBroadcast(Intent i,BroadcastReceiver receiver,…)；	Broadcast Receiver
startActivity(Intent i)； startActivityForResult(Intent i,int requestCode)；	Activity
startService(Intent i) bindService (Intent i,ServiceConnection coon,int flag)；	Service
insert(Uri uri,ContentValues values)； query(Uri uri,String[] projection,String selection,…)； update(Uri uri,ContentValues values,String selection)；	Content Provider

Android 第三方库并不可以直接运行在移动设备中，它只是作为一种资源服务在应用中被应用，因此只是单纯地构造出测试数据集是不够的，我们还需要构造并使用这些库的应用，通过应用的运行结果检测测试数据集的有效性。本文为了获取详细的调用信息，拟采用 hook 技术，通过被 hook 函数的调用栈信息以及参数信息来分析通信连接情况。以 startService 为例，需要得到的是调用 startService 后共谋确实产生了的信息（证据）。

通过分析 startService 的调用时序可以发现，为了获取具体的调用栈以及 Intent、IntentFilter 参数值，需要 hook

[1] http://github.com/secure-software-engineering/DroidBench/tree/develop

[2] http://github.con/fgwei/ICC-Bench

ContextWrapper. startService 以获取发送函数的 Intent 值,采用 hook 组件类中的 onStartCommand 方法获得接收函数的 IntentFilter 值。类似地,本研究会逐个分析各种组件通信底层的调用流程,然后 hook 住关键方法,得到通信的关键联系,利用 hook 技术,验证 Android 第三方库测试集的可靠性。

4.2　实验设计

实验 1　针对 29 个测试数据集进行检测,记录系统分析每个库的时间和结果。对测试数据集的分析结果在一定程度上反映了本检测系统的效果。aar 是 Android 专有的第三方库,包含了资源文件,并且现在 Android Studio 已经支持自动合并应用的 AndroidManifest 和 aar 中的 AndroidManifest 文件,因此实验还会针对测试数据集的".aar"文件和".jar"文件分别进行检测,并对比两种情况下的检测效果。

实验 2　针对收集的国内外流行的 1 207 个 Android 第三方库进行检测,针对库的不同类别,分别从使用的敏感 API、敏感权限、ExitPath、EntryPath、共谋通信路径对结果进行统计分析。对真实库的检测能够验证本系统的运行效率、可行性以及兼容性。

4.3　实验结果与分析

表 3 展示了测试数据集中的敏感路径情况,并且已经通过动态验证确认确实存在共谋情况。

表 3　测试数据集中的共谋行为

Table 3　Collusion behaviors in test data set

发送方	接收方	发送路径数量	接收路径数量	共谋路径数量
droibench_deviceid_broadcast1. arr	droibench_collector. aar	2	1	1
droibench_deviceid_contentprovider1. arr	droibench_collector. aar	3	1	1
droibench_deviceid_orderedintent1. arr	droibench_collector. aar	2	1	1
droibench_deviceid_service. arr	droibench_collector. aar	1	1	1
droibench_location. arr	droibench_collector. aar	2	1	2
droibench_location_ broadcast. arr	droibench_collector. aar	3	1	2
droibench_location_ service. aar	droibench_collector. aar	2	1	1
droibench_sendsms. aar	droibench_echor. aar	1	2	1
droibench_startactivityforresult1. aar	droibench_echor. aar	2	2	2
droibench_sendbroadcast1_source. arr	droibench_sendbroadcast1_sink. arr	1	2	1
droibench_startactivity1_source. arr	droibench_startactivity1_sink. arr	1	2	1
droibench_startservice1_source. arr	droibench_startservice1_sink. arr	1	2	1
iccbench_implicit_src_nosink. aar	iccbench_implicit_src_sink. aar	1	1	1
iccbench_implicit_src_nosink. aar	iccbench_implicit_nosrc_sink. aar	1	1	1
iccbench_implicit_src_nosink. aar	iccbench_implicit_action. aar	1	1	1
iccbench_implicit_src_sink. aar	iccbench_implicit_action. aar	1	1	1
iccbench_implicit_src_sink. aar	iccbench_implicit_nosrc_sink. aar	1	1	1
iccbench_implicit_action. aar	iccbench_implicit_src_sink. aar	1	1	1
iccbench_implicit_action. aar	iccbench_implicit_nosrc_sink. aar	1	1	1
iccbench_implicit_mix1. aar	iccbench_implicit_mix2. aar	1	1	1
iccbench_implicit_mix2. aar	iccbench_implicit_mix1. aar	1	2	1
dynamicbraodcastsendjar. aar	dynamicbraodcastreceivejar. aar	1	1	1
staticbraodcastsendjar. aar	staticbraodcastreceivejar. aar	1	1	1
startservice. arr	masterclient. aar	1	1	1
startactivity. arr	masterclient. aar	1	1	1
contentprovider. aar	masterclient. aar	1	1	1

系统的检测效果评估结果如表 4 所列,计算后可以得到,针对测试数据集(测试了 29 个库),系统的精确率(precision)达到 100%,召回率(recall)为 89.66%,*F-measure* 值为 0.945。分析以上结果,精确率说明系统检测出来的结果都是正确的,召回率说明了系统检测效果的可靠性,F 值综合评价了系统的检测效果,充分说明了本检测系统的可靠性。

表 4　29 个测试 Android 第三方库的检测效果评估

Table 4　Detection effectiveness evaluation of 29 test Android third-party libraries

混淆矩阵		预测	
		True	False
真实	Positive	26	0
	Negative	0	3

检测时间方面,主要时间消耗在进行静态分析部分,尤其敏感路径的分析时间会受到 Source 和 Sink 数目的影响,待测文件的大小也是影响因素之一。根据统计,测试数据集的最长检测时间为 25 s 左右,平均检测时间为 12.7 s,检测速度较快。

需要强调的是,上述是针对 aar 格式的库进行检测的结果。而当这些库都是 jar 形式时,只能检测出动态注册广播通信,因为它是在运行时注册广播,而其他通信方式都需要在 manifest 文件中注册,jar 库是没有打包资源文件的,因此无法进行确切匹配。

实验 2　利用 python 脚本从 maven 上下载了 1 162 个 Android 第三方库[20],这些库被分为 Analytics,Android,Cloud,SocialMedia,Utilities,Advertising 6 类,其组成个数如表 5 所列。需要说明的是,存在一个库有多个版本的情况,因此不考

虑同一个库不同的版本的情况，只有 41 个不同的库。另外一部分库是国内一些推送服务库、广告库，主要从各个官网下载，一共 17 个库，这些库的不同版本数目合计 45 个。

表 5 1162 个库类别的分布情况

Table 5 Distribution of 1162 libraries

类别	个数
Analytics	100
Android	131
Cloud	170
SocialMedia	124
Utilities	607
Advertising	30
总计	1162

其中，".aar"文件约占总分析文件数目的 20%，这说明大多数被分析的第三方库都没有 manifest 文件，由前文分析可知，这将影响最终的分析结果。但是存在一种明显的趋势，低版本向高版本发展时".aar"文件版本库明显增多，在共 61 种库中，低版本是 jar 类型的库且高版本是 aar 类型的库占比达 21.31%，这种变化趋势说明了 aar 逐渐被接受，本文研究的检测结果将随着这种改变呈现出更好的效果。

对 1207(1162 个库加上 45 个库)个第三方库进行分析，检测结果如表 6 所列，分析得到的 ExitPath 有 292 条，EntryPath 有 3719 条，但是并没有发现敏感信息传输的通信。一方面原因是，缺少 manifest 文件，导致没有组件的 IntentFilter 信息；另一方面原因是，在部分有 manifest 文件的库中，配置文件的信息基本都是不完整的，没有对组件做完整的定义，同样也缺乏 IntentFilter 信息，因此导致最终无法将路径关联起来。

表 6 检测结果统计

Table 6 Statistics of retection results

ExitPath	EntryPath
292	3719

在获取到的 ExitPath 中，所传输的敏感 API 如表 7 所列，主要涉及的敏感信息有网络情况、设备 ID、地理位置、账号信息等。

表 7 第三方库使用的敏感 API 分布情况

Table 7 Distribution of sensitive APIs used by third-party libraries

(单位：%)

敏感 API	占比
getActiveNetworkInfo	46.6
getDeviceId	21.4
getLatitude	9.2
getLongitude	9.2
getLastKnownLocation	5.4
getSimOperatorName	5.1
getAccounts	2.0
getRunningTask	1.7

文献[21]对非敏感信息通信的应用共谋进行设备性能损耗研究，结果发现这类行为会造成 2 倍左右的电量消耗、2 倍左右的延迟以及 200M 左右的内存损耗。基于该信息，本文也对这类非敏感信息共谋进行了分析。实验下载了国内著名的第三方库共 41 个，以及对应的官方模板应用，通过分析发现了一些库中存在唤醒组件，例如在极光推送库 jpush-android 中存在 service 组件 cn.jpush.android.service.DaemonService，该组件的 action 值为"cn.jpush.android.Intent.DaemonService"，能够唤醒相同设备中所有使用了 jpush 推送库的应用；Umeng-push 库和 alicloud-android-push 库中存在名为 com.taobao.accs.ChannelService 的组件，该组件的存在使得只要是使用了这两种第三方库的 ChannelService 组件都能够相互唤醒；在 Umeng-push 中注册了 Mipush 和 Huawei push 相关的组件，使得 Umeng-push 被唤醒的可能性增大，并且目前在应用中接入多个推送库、广告库的情况十分常见。实验中发现的另一个情况是，大部分库为了增加保活率会监听开机、电量变化、网络切换等系统广播，考虑到设备中使用的应用个数以及应用使用第三方库个数的情况可以发现，设备中大量存在此类广播，导致设备的性能受到影响。

结束语 应用分析技术的成熟以及 Android 隐私数据的价值的提升使协作攻击问题逐渐得到重视。目前很多研究都是针对应用的共谋研究，但是 Android 第三方库与应用属于不同的利益体，也可能引发这种风险。为了解决这类问题，文中主要基于 Android 第三方库进行共谋行为研究，充分考虑第三方库的特点，对 Android 第三方库可能引起的共谋风险进行了研究，考虑 Android 安全机制和不同利益体的目标，分析这种问题的成因和危害，提出一种 Android 第三方库共谋行为检测方法，并实现了原型系统，验证了所提方案的有效性和实用性。

本文主要是对 Android 第三方库的共谋行为进行了基础性的研究和探索，但主要是针对 Android 组件间通信构成的共谋行为进行检测，事实上 Android 还可以通过文件读写、Socket 等方式进行通信，并构成共谋，未来可进一步扩充对 Socket 通信等方式的检测。

参 考 文 献

[1] China Internet Development Statistics Report [OL]. [2018-01-31]. http://www.cac.gov.cn/2018-01/31/c_1122347026.htm.

[2] VIENNOT N, GARCIA E, NIEH J. A measurement study of google play[C]// Proceedings of the 2014 ACM International Conference on Measurement and Modeling of Computer Systems. New York: ACM, 2014: 221-233.

[3] SEO J, KIM D, CHO D, et al. FLEXDROID: Enforcing In-App Privilege Separation in Android[C]// Proceedings of the 23th Annual Network & Distributed System Security Symposium. Reston, Virginia: ISOC, 2016: 1-15.

[4] LI Q, CLARK G. Mobile Security: A Look Ahead[J]. IEEE Security & Privacy, 2013, 11(1): 78-81.

[5] ZHANG Z W, LEI L G, WANG Y W. Studying the Implementa-

tion and Security of the Permission Mechanism in Android[J]. Netinfo Security,2012(8):3-6. (in Chinese)
张中文,雷灵光,王跃武. Android Permission 机制的实现与安全分析[J]. 信息网络安全,2012(8):3-6.

[6] BHANDARI S,JABALLAH W B,JAIN V,et al. Android App Collusion Threat and Mitigation Techniques[OL]. [2018-05-27]. https://arxiv.org/pdf/1611.10076.

[7] LIU B,JIN H X,GOVINDAN R. Efficient Privilege De-Escalation for Ad Libraries in Mobile Apps[C]// Proceedings of the 13th International Conference on Mobile System,Applications,and Services. New York:ACM,2015:89-103.

[8] WANG J,WU H. Android Inter-App Communication Threats,Solutions,and Challenges[OL]. [2018-05-27]. https://arxiv.org/pdf/1803.05039.

[9] TAYLOR V F,BERESFORD A R,MARTINOVIC I. Intra-Library Collusion:A Potential Privacy Nightmare on Smartphones [OL]. [2018-05-27]. https://arxiv.org/pdf/1708.03520.

[10] LI L,ALEXANDRE B,TEGAWENDÉ F,et al. Apkcombiner:Combining multiple android apps to support inter-app analysis [C]// Proceedings of the 30th ICT Systems Security and Privacy Protection. Berlin:Springer,2015:513-527.

[11] RAVITCH T,CRESWICKE R,TOMB A,et al. Multi-App Security Analysis with FUSE:Statically Detecting Android App Collusion[C]// Proceedings of the 4th Program Protection and Reverse Engineering Workshop. New York:ACM,2014:1-10.

[12] ZHANG M,YANG L,ZHANG J W. FuzzerAPP:The Robustness Test of Application Component Communication in Android [J]. Journal of Computer Research and Development,2017,54(2):338-347. (in Chinese)
张密,杨力,张俊伟. FuzzerAPP:Android 应用程序组件通信鲁棒性测试[J]. 计算机研究与发展,2017,54(2):338-347.

[13] BLASCO J,CHEN T M. Automated generation of colluding apps for experimental research[J]. Journal of Computer Virology & Hacking Techniques,2018,14(2):127-138.

[14] ASAVOAE I M,BLASCO J,CHEN T M,et al. Towards Automated Android App Collusion Detection[C]// Proceedings of the 1st International Workshop on Innovations in Mobile Privacy and Security. 2016.

[15] WEI F G,ROY S,OU X M,et al. Amandroid:A Precise and General Inter-Component Data Flow Analysis Framework for Security Vetting of Android Apps[C]// Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. New York:ACM,2014:1329-1341.

[16] BOSU A,LIU F,YAO D F,et al. Collusive Data Leak and More:Large-scale Threat Analysis of Inter-app Communications [C]// Proceedings of the 12th ACM on Asia Conference on Computer and Communications Security. New York:ACM,2017:71-85.

[17] OCTEAU D,LUCHAUPD,DERINGM,et al. Composite constant propagation:Application to android intercomponent communication analysis[C]// Proceedings of the 37th International Confe-rence on Software Engineering (ICSE),2015.

[18] BUGIELS,DAVI L,DMITRIENKO A,et al. Xmandroid:A new android evolution to mitigate privilege escalation attacks:Technical Report TR-2011-04[R]. Technische Universitadt Darmstadt,2011.

[19] FENG H,FAWAZ K,SHIN K G. LinkDroid:Reducing Unregulated Aggregation of App Usage Behaviors[C]// Proceedings of the 24th USENIX Security Symposium. Berkely,CA:USENIX,2015:769-783.

[20] BACKES M,BUGIEL S,DERR E. Reliable Third-Party Library Detection in Android and its Security Applications[C]// Proceedings of the 23th ACM Conference on Computer and Communications Security. New York:ACM,2016:356-367.

[21] XU M W,MA Y,LIU X Z,et al. AppHolmes:Detecting and Characterizing App Collusion among Third-Party Android Markets[C]// Proceedings of the 16th International Conference on World Wide Web. Holland:Elsevier,2017.