计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20180548
56(6):1252-1262,2019

一种跨 APP 组件间隐私泄露自动检测方法

李 振[1,2] 汤战勇[1,2] 李政桥[1,2] 王 海[1] 龚晓庆[1] 陈 峰[1] 陈晓江[1,2] 房鼎益[1,2]
[1](西北大学信息科学与技术学院 西安 710075)
[2](陕西省无源物联网国际联合研究中心 西安 710075)
(rongzhenl@stumail.nwu.edu.cn)

An Automatic Detection Method for Privacy Leakage Across Application Components

Li Zhen[1,2], Tang Zhanyong[1,2], Li Zhengqiao[1,2], Wang Hai[1], Gong Xiaoqing[1], Chen Feng[1], Chen Xiaojiang[1,2], and Fang Dingyi[1,2]
[1](*School of Computer Science and Technology, Northwest University, Xi'an* 710075)
[2](*Shaanxi International Joint Research Centre for the Battery-free Internet of Things, Xi'an* 710075)

Abstract In recent years, Android operating system has developed rapidly. A large number of mobile users use Android smart devices as tools for personal communication and work. The privacy information of Android mobile users has become one of the main targets of black industry practitioners. Existing privacy detection research mainly focuses on addressing privacy leakage risk within Android applications, including the detection of privacy leakage within program components, the detection of privacy leakage between components, and the detection of ICC vulnerability. Actually, the behavior of sharing users' privacy through collaboration among application components is widespread, which causes a large number of users' privacy information to be leaked. How to effectively detect and prevent privacy leakage between application components is an urgent problem. However, the number of components in Android applications is huge and there are plenty of components unrelated to privacy leaks between applications. Therefore, how to detect possible privacy leaks between applications meets a serious challenge. Aiming at this problem, this paper presents a method to construct a component sequence with potential privacy leaks, and the method uses data flow analysis technology to realize a detection system for privacy leakage between application components, named PLDetect. PLDetect solves the problem of incomplete coverage of code and lagging detection results in the existing technology. Finally, based on the privacy leak path, PLDetect utilizes an encryption-based privacy leak protection method to encrypt privacy information, ensuring that information is effectively prevented from being maliciously transmitted without affecting application runtime performance. The final experiment shows that PLDetect detects 5 groups of applications with privacy leaks across application components in 81 applications and effectively blocks privacy data leaks.

Key words Android security; privacy leakage; static analysis; data-flow analysis; taint analysis

收稿日期:2018-08-02;**修回日期**:2018-11-13

基金项目:国家自然科学基金项目(61672427);陕西省国际合作项目(2017KW-008);陕西省国际合作计划 (2019KW-009);陕西省重点研发计划 (2017GY-191);陕西省创新团队(2018SD0011)
This work was supported by the National Natural Science Foundation of China (61672427), the International Cooperation Program of Shaanxi Province (2017KW-008), the International Cooperation Program of Shaanxi Province(2019KW-009), the Key R&D Project of Shaanxi Province (2017GY-191), and the Innovation Research Team of Shaanxi Province (2018SD0011).

通信作者:汤战勇(zytang@nwu.edu.cn)

摘　要　近年来,Android 操作系统发展迅猛,大量的移动用户使用 Android 智能设备作为私人通信和工作的工具.Android 移动用户的隐私信息随之成为黑色产业从业者的主要攻击目标之一.现有的隐私检测研究主要集中于解决 Android 应用程序内部的隐私泄露风险,包括程序组件内隐私泄露、组件间隐私泄露以及组件间通信(inter-component communication, ICC)漏洞的检测.然而在实际环境中,不同应用程序间通过协作获取用户隐私的行为广泛存在,这造成大量用户隐私信息被泄露的风险.如何有效检测和防止跨 APP 组件间隐私泄露是亟待解决的问题.然而 Android 应用程序中组件数量庞大并且存在大量与跨 APP 间隐私泄露无关的组件.因此在应用程序之间如何检测可能存在的隐私泄露路径面临严峻的挑战.针对该问题,提出一种构建潜在泄露隐私的组件序列的方法,并利用数据流分析技术实现一个跨 APP 组件间隐私泄露的检测系统 PLDetect.PLDetect 解决了现有技术存在的检测结果滞后的问题以及代码覆盖率不全的问题.最后,PLDetect 在隐私泄露路径的基础上,使用一种基于加密的隐私泄露防护方法对隐私信息进行加密,保证在不影响应用程序运行时性能的情况下有效阻止隐私数据被恶意传送.最终实验表明,PLDetect 在 81 个应用程序中监测出 5 组应用程序存在跨 APP 组件间隐私泄露问题并有效阻断了隐私数据的泄露.

关键词　Android 安全;隐私泄露;静态分析;数据流分析;污点分析

中图法分类号　TP393.0

随着 Android 移动市场份额的不断扩大,越来越多的恶意攻击者将目标瞄向 Android 设备.腾讯社会研究中心最新一期发布的互联网安全报告[1]显示:在所选取的 852 个 Android APP(Android application)中,约有 98.5%的 APP 都需要获取用户隐私权限.Felt 等人[2]研究发现,以获取并泄露用户隐私信息为目标的恶意软件占据很大的比例,这一结果很好地解释了近年来国内外个人信息泄露事件频发的问题.

学术界和工业界一般将隐私权限等级划分为高危隐私权限、核心隐私权限、重要隐私权限和普通隐私权限[3-4].不同层面的权限滥用会对用户造成不同程度的影响,如何在权限安全控制和用户的体验之间抉择是一个相互制约的话题,使用权限定义复杂的控制手段能够在一定程度上保护用户隐私数据,但却以牺牲用户体验为代价.为了娱乐性和实用性,目前移动设备仅使用粗粒度的访问权限控制手段.Android 系统自身实现了一套用于保护敏感信息资源安全性的应用程序编程接口(application programming interface, API)访问权限控制机制,用于保护敏感信息资源的安全性.其要求在程序安装时向用户显式地申请使用权限.然而中国互联网数据中心[4](Data Center of China Internet, DCCI)调研发现,有 72%的用户不清楚 APP 获取隐私权限的用途.所以大多数用户为了追求良好的用户体验,不惜将该程序自身功能不必需的隐私权限授权给该 APP,因此这种 API 权限控制机制并不能有效控制第三方应用程序对隐私数据的访问和使用.

软件泄露隐私的主要途径分为组件内隐私泄露、组件间隐私泄露和跨 APP 组件间隐私泄露.组件内隐私泄露是指在 Android APP 内部,某组件能够独立完成隐私的获取和泄露[5];而组件间隐私泄露是指 Android APP 内部的 2 个组件通过合作完成获取和泄露隐私[6];跨 APP 组件间隐私泄露是指某个恶意软件利用存在组件间通信(inter-component communication, ICC)漏洞[7-8]通过第三方 APP 来泄露隐私;据 ComDroid[8]研究表明,60%的应用程序至少存在 1 个以上的 ICC 漏洞.因此,跨 APP 组件间隐私泄露普遍存在并且不容忽视.

虽然目前存在很多隐私检测的研究[9-16],但是针对跨 APP 组件间隐私泄露问题的研究涉及较少.FlowDroid[5]提出一种新型按需算法的静态污点分析技术检测隐私泄露问题;LeakMiner[17]结合 Android 自身特有的安全机制以及编程模型,基于函数调用关系实现了对程序中隐私信息的识别、跟踪和泄露检测.上述隐私泄露检测方法的精确度很高,但仅限于组件内隐私泄露问题;IccTA[6]和 Amandroid[18]等方法针对组件间隐私泄露检测.其中,IccTA 结合 Epicc[7]与 FlowDroid 方法,通过跟踪组件间的隐私信息传播达到检测的目的.

跨 APP 组件间隐私泄露检测的实现存在很大的挑战.原因在于 APP 中组件数量庞大、依赖关系复杂,存在很多与跨 APP 组件间隐私泄露无关的组件序列路径.因此,直接利用现有的隐私泄露检测

技术,会在构建控制流图(control flow graph, CFG)时引发空间爆炸,严重影响检测效率.而且,由于跨APP组件间隐私泄露往往涉及到多个应用程序的集合,APP间代码不连续将导致在分析隐私泄露路径时无法建立一个连续的CFG,而现有工作不能解决此问题.

因此,本文提出了一种检测跨APP组件间隐私泄露的方法PLDetect.PLDetect提出生成潜在泄露隐私的组件序列,精简与跨APP组件间隐私泄露无关的组件序列的方法,有效地解决了在生成CFG时路径爆炸问题;进一步地,PLDetect通过将生成虚拟主函数和插桩技术有效结合,解决跨APP组件间隐私泄露中的多种代码不连续性问题;最后,生成基于精简组件的控制流图,并执行应用程序间静态污点分析,输出跨APP组件间隐私泄露的路径.

本文的主要贡献有3个方面:

1) 设计并实现1个采用静态污点分析方法检测跨APP组件间隐私泄露的系统PLDetect;

2) 在保证PLDetect覆盖泄露隐私信息路径的情况下,提出生成潜在泄露隐私的组件序列的方法,用以提高检测效率;

3) 针对APP间代码不连续造成无法构建完整的CFG,从而导致不能执行数据流分析的问题,通过移植改进IccTA中的插桩方法对此问题进行了有效解决.

1 跨APP组件间隐私泄露的攻击模型

为了准确描述跨APP组件间隐私泄露攻击模型,首先给出跨APP组件间隐私泄露的相关定义.

假定 App_i 为Android Market中应用程序,$SandBox_i$ 为其对应的运行沙箱环境,则表示为 App_i in $SandBox_i$.假定 fp 为隐私信息获取通道表征函数,$Privacy_i$ 为 App_1 能获取的隐私信息,则表示为 $Privacy_i = fp(App_i)$.假定 lp 为隐私信息泄露通道表征函数,若 App_i 不能泄露隐私信息,则表示为 $lp(App_1)$=false,否则,$lp(App_1)$=true.

在 App_1 in $SandBox_1$ & App_2 in $SandBox_2$ 且 $SandBox_1 \cap SandBox_2 = \varnothing$ 的条件下,如果 App_1 满足 $Privacy_1 = fp(App_1) \neq \varnothing$ 且 $lp(App_1)$=false,App_2 满足 $Privacy_2 = fp(App_2) \neq \varnothing$ 且 $lp(App_1)$=true,并且存在一种关联关系函数 R,使得 $Privacy_2 = fp(R(App_2, App_1)) \neq \varnothing$ 且 $Privacy_2 \in Privacy_1$,那么 App_1 和 App_2 可能存在跨APP组件间隐私泄露的路径.

根据定义,对跨APP组件间隐私泄露的攻击模型进行实例化描述.

在图1中,App_1 是获取隐私信息的良性Android应用程序,App_2 是泄露隐私信息的恶意Android应用程序.App_1 中的组件 A 通过授权的API权限得到设备ID值(*getDeviceId*),并通过

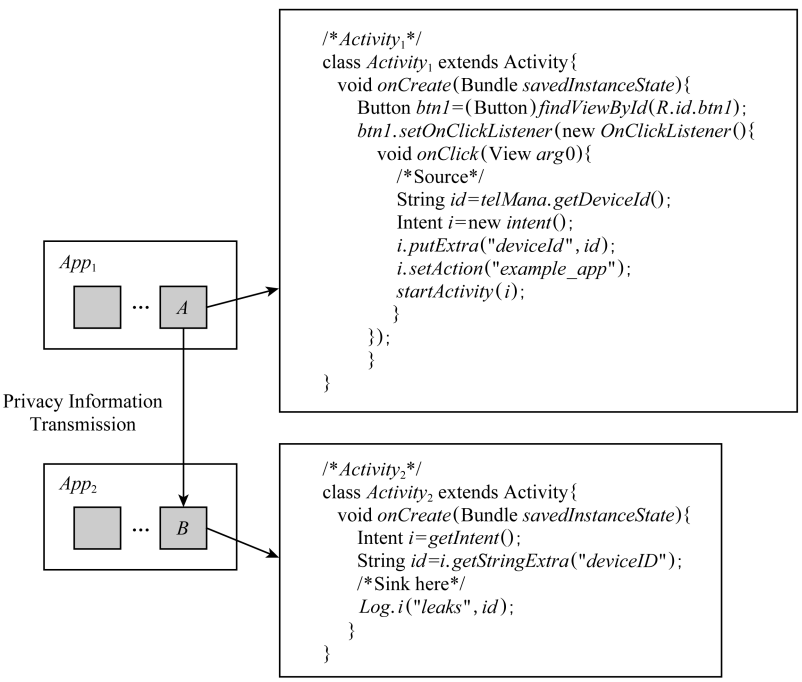

Fig. 1 The example of privacy leakage across application components

图1 跨APP组件间隐私泄露实例

ICC 方法(*startActivity*)将 ID 值传递给 App_2 中的组件 B,并由 B 将 ID 值泄露.图 1 右侧的代码分别是 App_1 中组件 A 与 App_2 中组件 B 的关键代码简略描述.另外,在实际用户使用环境中,不排除 2 个恶意程序通过合作互通的形式更复杂更隐蔽地达到泄露隐私的目的.在上述实例中,App_1 可以利用恶意攻击者所规定的某种数据拆分格式,如图像格式或自定义文件等形式,将设备 ID 发送至 App_2,随后由 App_2 将接收到的信息经过一系列数据重组以及信息转换手段写出,最终以非授权分发方式发送至攻击者.

上述中不论隐私信息以哪种形式组织,App_1 满足 $Privacy_1=fp(App_1)\neq\varnothing$ 且 $lp(App_1)=$false,App_2 满足 $Privacy_2=fp(App_2)\neq\varnothing$ 且 $lp(App_2)=$true.而且 App_1 与 App_2 存在关联关系 R 函数(*startActivity*),使得 $Privacy_2=fp(R(App_2,App_1))\neq\varnothing$,因此,$App_1$ 和 App_2 间可能存在跨 APP 组件间隐私泄露的路径.

2　PLDetect 系统实现

PLDetect 由 5 个执行步骤组成,如图 2 所示,在 Step1 中,利用 Epicc 提取输入中每个 APP 的组件属性;在 Step2 中,利用 Step1 的组件属性对待分析的多个 APP 进行分类;在 Step3 中,利用分类结果精简与跨 APP 组件间隐私泄露无关的组件,生成潜在泄露隐私的组件序列,从而解决由于组件间依赖关系复杂而造成的路径爆炸问题;在 Step4 中,在潜在泄露隐私组件的基础上,利用虚拟主函数和插桩技术解决由于 Android 代码不连续性造成无法进行静态污点分析的问题,并构建基于精简组件的 CFG;在 Step5 中,PLDetect 利用 SuSi[19](一种采用机器学习方法收集 Android 敏感 API 的工具)收集到的敏感源和泄露点,在 CFG 上执行应用程序间的静态污点分析,最终输出检测到的跨 APP 组件间隐私泄露的路径.

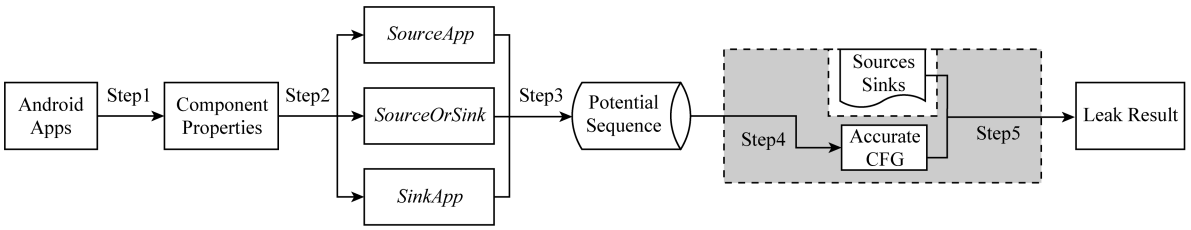

Fig. 2　Overall framework of PLDetect

图 2　PLDetect 系统整体框架

2.1　组件属性提取

在图 2 的 Step1,PLDetect 利用 Epicc 提取 Dex 和 AndroidManifest 文件中的相关信息.Epicc 是一个基于 Soot 编译框架[20]和 Heros 数据流分析工具[21]的 ICC 拓扑工具,能够提取组件属性、ICC 方法和参数.

PLDetect 提取组件属性为:

1) 声明的组件列表;

2) 每个组件的 *intent_filter* 标签;

3) 每个组件的 *exported* 属性值;

4) 每个组件的 *intent* 参数值.

算法 1. 计算组件的 *exported* 值.

输入:Android 应用程序的组件;

输出:true 或 false.

```
if explicitExported(component)=true then
/*判断当前 Activity 是否可以被另一个 Application 的组件启动*/
  return getTheValue("exported");
end if
if componentType(component)=ContentProvider
/*componentType 方法返回类的组件类型*/
then
  if SDKversion≤16 then
    return true;
  else
    return false;
  end if
end if
if ComContainIntentFilter(component)=true
/*判断 IntentFilter 中是否包含该组件*/
then
  return true;
```

```
else
  return false;
end if
```

通过属性 1 得到应用程序中声明的组件列表.当组件 *exported* 属性值为 true 时,则表示允许第三方 APP 访问当前组件;反之则拒绝任何来自第三方 APP 访问.因此,PLDetect 采用算法 1 计算组件的 *exported* 属性值,并通过属性 2 和属性 4 得到组件的 *intent* 参数值和 *intent_filter* 属性值,用于组件间的匹配.

2.2 APP 分类

跨 APP 组件间隐私泄露是由 2 个 APP 组合完成的,但是,任意 2 个 APP 组合不一定能够达到泄露隐私的目的.因此,针对所有的 APP 进行分类及相应的筛选处理是非常必要的.

1) 根据跨 APP 组件间隐私泄露的条件,PLDetect 认为 2 种 APP 是良性的:

① 如果当前 APP 拥有获取隐私信息的权限,并能够获取隐私信息,则不能将隐私信息传递给第三方应用程序.

② 如果当前 APP 拥有泄露隐私信息的权限,并能够泄露隐私信息,则不能被第三方应用程序调用.

上述 2 种 APP 不能作为跨 APP 组件间隐私泄露集合中的 APP.所以 PLDetect 对具有以上条件的 APP 进行剪枝,从而避免不必要的分析时间.

2) Android 组件中是否拥有隐式 *intent* 决定该 APP 是否能够调用第三方 APP,因此,假设当前 APP 拥有隐式 *intent* 的组件,则将该 APP 归类为 *SourceApp*.Android 组件的 *exported* 属性值决定组件能否被第三方应用程序调用,假设当前 APP 拥有 *exported* 的值为 true,则将该 APP 分为 *SinkApp* 类.假设应用程序同时满足上述 *SourceApp* 和 *SinkApp* 的条件,则将该 APP 分为 *SourceOrSink* 类.

3) 根据 2)中分类结果,本文得到可能实现跨 APP 组件间泄露隐私信息的 APP 组合有 4 种形式:(*SourceApp*, *SinkApp*),(*SourceApp*, *SourceOrSink*),(*SourceOrSink*, *SinkApp*),(*SourceOrSink*, *SourceOrSink*).

2.3 生成潜在泄露隐私的组件序列

为了能够有效精简组件间的依赖关系,构造真实存在的并能引发隐私泄露的组件序列,PLDetect 根据 2.2 节中得到的潜在泄露隐私的 APP 组合,提出一种生成潜在泄露隐私的组件序列的方法,从而解决路径爆炸的问题.PLDetect 生成潜在泄露隐私的组件序列的方法步骤为

Step1. 根据潜在泄露隐私的 APP 组合中的 2 个应用程序,分别生成潜在泄露隐私的组件子序列 Seq_A, Seq_B.

Step2. 利用潜在泄露隐私的组件子序列 Seq_A, Seq_B,构建完整的潜在泄露隐私的组件序列.

生成潜在泄露隐私的组件子序列的具体方法是:首先,根据 2.1 节中得到的 *intent* 参数与 *intent_filter* 标签,并利用组件间匹配规则生成图 3 和图 4 中的组件调用关系;然后,根据组件调用关系,生成应用程序运行时可能出现的组件执行序列;最后,PLDetect 判断组件执行序列中是否存在能够构建完整的潜在泄露隐私的组件序列的组件子序列.

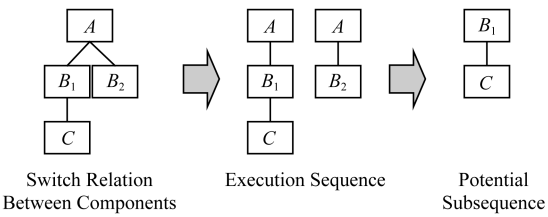

Fig. 3 Generation process of leakage sequence for element 1 in an APP combination

图 3 APP 组合中元素 1 泄露序列生成过程

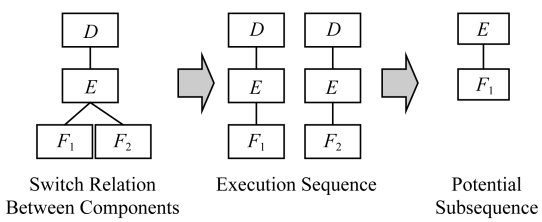

Fig. 4 Generation process of leakage sequence for element 2 in an APP combination

图 4 APP 组合中元素 2 泄露序列生成过程

图 3 表示的是 2.2 节中 APP 组合中第 1 个元素生成潜在泄露隐私的组件子序列的过程描述,PLDetect 根据组件调用关系生成了 2 种组件执行序列:1) $A \rightarrow B_1 \rightarrow C$;2) $A \rightarrow B_2$.然后,PLDetect 采用算法 2 判断 1), 2)中是否存在潜在泄露隐私的组件子序列.类似地,图 4 表示的是 2.2 节中 APP 组合中第 2 个元素生成潜在泄露隐私的组件子序列的全过程.

因此,假定经过算法 2 处理之后生成的潜在泄露隐私的组件子序列分别为 $B_1 \rightarrow C$, $E \rightarrow F_1$.那么,最终由 PLDetect 生成的潜在泄露隐私的组件序列

为 $B_1 \rightarrow C \rightarrow E \rightarrow F_1$.其中,本序列中各个元素是与跨 APP 组件间隐私泄露问题的相关组件,同时也是需要构建 CFG 的组件.

算法 2. 判断 APP 组合序列中第 i 个元素的执行序列是否存在潜在泄露隐私的组件子序列.

输入:应用程序的执行序列;

输出:潜在泄露隐私的组件子序列.

```
if findLefttoRight(component_sequence) =
  false then /* findLefttoRight 方法在执行
  序列中寻找获取隐私信息的组件 */
  return false;
else
  component_first←findLefttoRight
    (component_sequence);
end if
if findRighttoLeft(component_sequence) =
  false then /* findRighttoLeft 方法在执行
  序列中寻找能够调用第三方 APP 的组件 */
  return false;
else
  component_second←findRighttoLeft
    (component_sequence);
end if
m ←component_sequence.indexOf(component_
  first);
/* 判断组件 component_first 在执行序列中的
   位置 */
n ←component_sequence.indexOf(component_
  second);
if m>n then
  return false;
else
   return component_sequence.substring(m,
     n+1);
end if
```

2.4　生成精确的控制流图

Android 的代码不连续性将导致不能把多个独立的 APP 构建成一个完整的 CFG,因此,本节将详细介绍如何解决由 Android 中生命周期、回调函数以及 ICC 方法而引入的代码不连续性问题.

2.4.1　生命周期

Android 应用程序没有主函数,而是由组件生命周期函数组成的多个入口,即 Android 应用程序可以根据用户事件或系统事件,调用组件的每个生命周期函数.如当 Activity 组件处于 *onPause* 状态时,如果该 Activity 被重新启动,则该 Activity 直接进入 *onResume* 状态,而非 *onCreate* 状态;如果该 Activity 由于内存不足被杀死而后重新启动时,则该 Activity 进入 *onCreate* 状态.因此,由于 Android 框架为每个组件定义了完整的生命周期,从而造成 Android 应用程序多入口的问题,只有通过精确的模拟生命周期的变化,才能保证下一步静态分析时得到的隐私泄露路径的精确性.

为了解决由于生命周期引入而带来的的不连续性问题,根据 2.3 节中生成的潜在泄露隐私的组件序列,PLDetect 为序列中的每个组件生成 *dummyMain*(虚拟主函数).按照 Android 开发文档中生命周期的调用顺序,PLDetect 在 *dummyMain* 中精确地模拟生命周期的执行顺序,即在 *dummyMain* 中插入调用生命周期函数的语句.

2.4.2　回调函数

Android 操作系统允许注册回调函数.如 Button 的监听器,用户触发该按钮的点击事件后,由 Android 框架来自动调用 *onClick* 后的具体内容.PLDetect 使用 FlowDroid 处理回调函数的方法,在 *dummyMain* 中模拟回调函数的调用.然而,回调函数被调用的顺序无法预测,只有当回调函数所在的组件运行时,回调函数才可能被调用执行.因此,为了考虑模拟的精确性,PLDetect 建立了回调函数与组件间的对应关系,并利用 FlowDroid 收集的回调函数集合,判断潜在泄露隐私组件序列中的每个组件是否存在回调函数,如若存在回调函数,则在 *dummyMain* 中模拟回调函数.即:在组件的 *dummyMain* 函数中的 *onResume* 与 *onPause* 调用语句之间,生成调用回调函数的语句,使得回调函数在 *dummyMain* 中可达,从而解决代码不连续性问题.

2.4.3　ICC 方法

Android 应用程序中组件交互数据是通过 *Bundle* 对象和 *intent* 对象传递的,即 ICC 模型.但是,通过 *intent* 对象实现的代码并没有直接调用对应组件,在代码层,2 个组件是独立的、不连续的.它是由 Android 框架通过匹配 *intent* 与 *intent_filter* 的参数,从而实现组件间的匹配以及数据传递.

因此,本文利用 Soot[13] 生成 Jimple 中间语言,并结合插桩技术修改潜在泄露隐私的组件序列中组件间的 ICC 方法,从而解决 ICC 带来的组件间代码的不连续性问题.

PLDetect 的具体思路是:基于 Jimple 中间语言利用插桩技术修改 ICC 代码,从潜在泄露隐私的组件序列的左侧第 1 个组件开始,分别在前一个组件中实例化下一个组件,并调用 *helperIpc* 和 *dummyMain* 方法.如图 5 所示:

```
/*Activity1*/
-this.startActivity(intent);
+Activity2 activity=new Activity2();
+activity.helperIpc(intent);
+activity.dummyMain();
```

(a) $Activity_1$ in App_1

```
/*modify Activity1*/
+public void helperIpc(Intent intent){
+this._intent_ipc=intent;
+}
+public Intent getIntent(){
+    return this._intent_ipc;
+}
+public void dummyMain(){
  /*Invoking the lifecycle function or
callback function*/
+}
```

(b) $Activity_2$ in App_2

Fig. 5 The modification method of *startActivity*

图 5 *startActivity* 的修改方法

在图 5 中,以 *startActivity* 的修改方法为例对利用插桩技术解决 ICC 带来的代码不连续性问题进行说明.其中,图 5(a)表示 App_1 中的 $Activity_1$ 组件,图 5(b)表示 App_2 中的 $Activity_2$ 组件.而 $Activity_1$ 和 $Activity_2$ 组件符合组件匹配规则并能传递数据.PLDetect 通过修改图 5 中的代码,使得 $Activity_1$ 和 $Activity_2$ 在代码层上可达.

在图 5(b)中,PLDetect 生成 *helperIpc*(*intent*)和 *getIntent*()方法.*helperIpc* 方法将携带的 *intent* 对象赋值给 *_intent_ipc*;*getIntent* 的返回值为 *_intent_ipc*.*helperIpc* 和 *getIntent* 实现 *intent* 对象的传递过程,从而替换由 Android 框架完成的工作.其中,图 5(b)中 *dummyMain* 方法实现了 2.4.1 节和 2.4.2 节的思想.

在图 5(a)中,删除了 *startActivity* 方法,并实例化 $Activity_2$ 对象,然后调用 *helperIpc* 方法传递 *intent* 对象,将 *intent* 对象的信息传递到 $Activity_2$ 对象,并且通过调用 $Activity_2$ 中 *dummyMain* 方法,使得 $Activity_1$ 和 $Activity_2$ 在代码层彻底链接起来.

因此,在潜在泄露隐私的组件序列中,任意 2 个相邻的组件都会采用上述相同的方法.

在 Android 系统中,Implicit Intent 没有以一个指定的组件命名,而是声明需要执行的操作,该操作允许第三方应用程序的组件去处理它.例如:如果在地图上展示用户的位置,可以使用 1 个 Implicit Intent 去请求第三方应用程序在地图上展示指定的位置.因此,基于跨 APP 组件间隐私泄露的问题,应用程序间的数据传递本质上是使用 *intent* 对象携带、ICC 方法(如 *startActivity* 等)传递实现的,所以,针对 APP 间代码的不连续性,本文采用上述方法处理.基于潜在泄露隐私组件序列的插桩技术,一方面有效解决了 IccTA 等方法存在的路径爆炸、工作量庞大、分析效率低的问题,另一方面克服了现有方法作用域为单独应用程序内部组件的局限性.

最后,在解决代码不连续性问题后,PLDetect 将这些 APP 之间与隐私泄露相关的组件构建成 1 个完整的 CFG.并且利用 SuSi 收集 Android API 中的敏感源和泄露点方法集合,在构建的 CFG 上执行应用程序间的静态污点分析,跟踪隐私信息的数据流流向,得到并输出检测到的跨 APP 组件间隐私泄露的路径.

PLDetect 以非侵扰的方式工作在后台,并将上述步骤中得到的隐私信息泄露路径作为输入,使用插桩技术来修改路径中组件所在的 Android 应用程序.以此来保证隐私信息在被恶意应用泄露前被加密或替换为无效的数据,从而在不影响 APP 正常执行的情况下缓解隐私信息泄露造成的风险和危害.具体来讲,因为 Soot 支持阅读和重写 Davilk 字节码,因此 PLDetect 在上述 Soot 生成 Jimple 的基础上,通过插桩修改相关 Android 程序来达到数据加密或替换的目的.假设 APP_1 的 $Activity_1$ 与 APP_2 的 $Activity_2$ 经检测是 2 个符合跨 APP 间隐私泄露的组件并通过 *intent* 传递隐私信息.那么,PLDetect 将在 $Activity_1$ 中插入 1 个字段,且该字段的作用是唯一标识 $Activity_2$ 所在的 APK 的包名.同时,在 $Activity_2$ 组件中,PLDetect 会以 Jimple 形式插入一段功能代码.这段代码的主要作用是判断从 *intent* 中获取到的字段值是否与自身包名一致,若一致则加密或替换隐私信息的值,若不一致则不修改.因此,PLDetect 采取了简单有效的方法,在保证用户良好体验的前提下解决了隐私被泄露的隐患.

3 系统评测实验与分析

为了验证跨 APP 组件间隐私泄露的检测模块的检测效果,本文主要从触发程序和商业应用 2 方面进行验证.其中,触发程序是指实现跨 APP 组件间隐私泄露的 6 组 APP 集合,此类应用程序没有多余干扰路径,主要是为了验证 PLDetect 功能而开发的应用程序.这些应用程序包含 Activity, Service, Broadcast Receiver 等不同的 Android 组件.如表 1 所示.商业应用是指在 360 Market 中随机选择的 81 个真实应用,部分应用程序信息如表 2 所示.

Table 1　The Sets of Triggered Program

表 1　触发程序测试集

Serial Number	Test Case	Abbreviation	Activity	Service	Broadcast Receiver	ICC Method	Category
1	GetDeviceID1.apk	G_1	2	0	0	*startActivity*	*SourceApp*
2	LeakDeviceID1.apk	L_1	1	0	0		*SinkApp*
3	LeakDeviceID2.apk	L_2	1	1	0	*startService*	*SinkApp*
4	LeakDeviceID3.apk	L_3	1	0	1	*sendBroadcast*	*SinkApp*
5	GetDeviceID2.apk	G_2	2	0	0	*startActivity*	*SourceApp*
6	LeakDeviceID4.apk	L_4	1	0	0		*SinkApp*
7	LeakDeviceID5.apk	L_5	1	1	0	*startService*	*SinkApp*
8	LeakDeviceID6.apk	L_6	1	0	1	*sendBroadcast*	*SinkApp*

Table 2　The Sets of Business Application

表 2　商业应用测试集

Serial Number	Application Name	Version
1	com.sinelife.theone	1.5.3
2	com.qiyi.video	7.5.1
3	com.baidu.BaiduMap	6.0
4	com.qidian.QDReader	5.1.1
5	com.cubic.autohome	5.0.1
6	com.tencent.qqmusic	6.0
7	com.sdu.didi.psnger	5.1.1
8	com.tmall.wireless	5.19.1
⋮	⋮	⋮

3.1　触发应用

如表 1 所示,为了验证 PLDetect 系统功能的有效性,本文共构造了 6 组实现跨 APP 组件间隐私泄露的触发程序,这些触发程序中没有设置多余干扰检测的组件与路径.我们通过人工对比 PLDetect 的检测结果与预设的泄露隐私路径,从而验证 PLDetect 对跨 APP 组件间隐私泄露的检测效果.

经过 PLDetect 分析,表 1 中 6 组(共 8 个)触发程序的分类结果为:*SourceApp* 类别的 Android APP 有 G_1 和 G_2;*SinkApp* 类别的 Android APP 有 L_1, L_2, …, L_6; *SourceOrSink* 类别中没有 Android APP.在分类结果的基础上,我们可以得到的潜在泄露隐私的 APP 组合序列共计 12 组,如图 6 所示.

PLDetect 分析得到,12 组潜在泄露隐私的 APP 组合序列中存在 6 组 Android APP 能够泄露隐私信息,并输出泄露隐私的路径.经过人工比较 PLDetect 的检测结果与预设的泄露隐私的路径,两者完全一致,正确率达 100%.

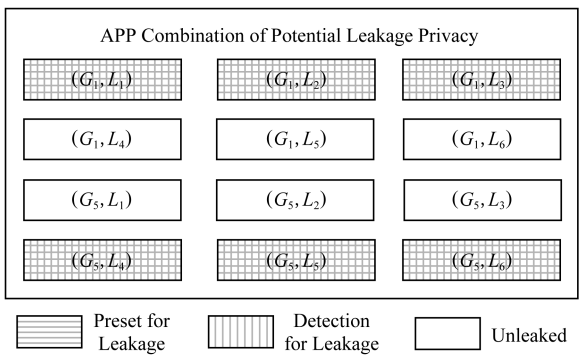

Fig. 6　The comparison of test results

图 6　检测结果对比

3.2　商业应用

在我们选取的实际商用 Android APP 中,平均每个应用程序包含组件高达 134 个,面对平均 134 个组件间错综复杂的关系,很容易影响 PLDetect 的检测结果.因此,为了进一步验证 PLDetect 的检测效果,选取了 81 个真实应用作为实验对象.

根据 2.2 节的分类规则,PLDetect 得到的分类数据如图 7 所示,组件 *exported* 属性值为 true 的数目为 2 301 个,占组件总数 10 815 的 21.28%.其中,29 个应用程序包含 *exported* 属性值为 true 的组件,即此 29 个 Android APP 能够被第三方应用

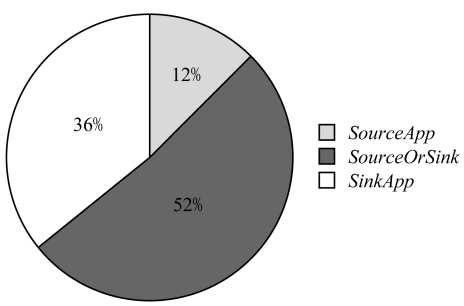

Fig. 7　The results of classification

图 7　分类结果

程序调用;10 个应用程序包含有隐式 *intent* 或显式调用第三方应用程序的组件,即此 10 个 Android APP 能够调用第三方应用程序;42 个应用程序既包含 *exported* 属性为 true 的组件,同时能够调用第三方应用程序.实验结果说明 52%的应用程序同时具备 *SourceApp* 与 *SinkApp* 的特点,将成为泄露隐私信息的重要组成部分.

进一步地,PLDetect 对获取隐私和泄露隐私的 API 进行了统计.结果显示,在获取隐私的 API 中,*getLongitude* 和 *getLatitude* 被使用的次数最多,达到了 1 834 次,如表 3 所示;在泄露隐私的 API 中,Log 被使用的次数最多,高达 66 711 次,如表 4 所示.根据上述数据推测,随着位置探测设备和地理位置信息系统的开发、应用及推广,使得基于位置信息服务(location-based service, LBS)的 Android 应用程序越来越多,百度地图、微信、滴滴打车等常用 Android 应用程序都需要位置的支持.因此,Android 应用程序中使用 *getLongitude* 和 *getLatitude* 次数非常多,同时导致位置信息泄露的威胁增大.Log 能够输出程序的中间结果,是调试 Android APP 的一种常用手段,但是,上述数据显示 Log 的使用次数过多,可能是由于程序员在调试后没有删除 Log 代码,导致 Log 遗留在 Andorid APP 中,从而造成隐私信息泄露的威胁增大.

Table 3 The Sets of API to Get Privacy Information

表 3 获取隐私的 API

API	Frequency	Description
getLongitude()	1 834	Get longitude
getLatitude()	1 834	Get latitudes
getCountry()	547	Get country code
getLastKnownLocation(*String*)	209	Get the nearest cache location
getSSID()	89	Get SSID

Table 4 The Sets of API to Leak Privacy Information

表 4 泄露隐私的 API

API	Frequency	Description
Log	66 712	LogCat
putString/*putBoolean*/*putInt*/*putLong*	48 959	Storing data
Write	890	Write to file
sendTextMessage	252	Send message
MediaRecorder.start/*MediaRecorder.setVideoSource*	208	Audio

其次,以 PLDetect 发现的 1 个具体实例来描述跨 APP 组件间隐私泄露的细节.PLDetect 检测发现 com.pdswp.su.smartcalendar 和 com.xkfop.xhuioa 能够合作实现跨 APP 组件间隐私泄露.其中 com.pdswp.su.smartcalendar 中用户输入的备忘录内容被 com.xkfop.xhuioa 劫持,导致备忘录内容被 com.xkfop.xhuioa 泄露.具体分析为:1)通过 com.pdswp.su.smartcalendar.bean.NoteItemBean 类中的方法 *getNote* 得到备忘录内容,并将备忘录内容赋给 *putExtra* 方法的参数;2)通过 *startActivity* 传递出去,被 com.xkfop.xhuioa 中的类 com.xkfop.sendService 的 *getIntent* 方法得到参数 *android.intent.extra.TEXT* 的值,并通过 *sendTextMessage* 将备忘录内容泄露.

PLDetect 在这 81 个真实应用中共识别和检测到 5 组跨 APP 组件间隐私泄露程序.通过统计 PLDetect 发现的这 5 组存在跨 APP 组件间隐私泄露情况的程序包发现其中通过 Activity 泄露隐私信息的为 2 个,通过 Service 泄露隐私信息的为 3 个,通过 Broadcast Receiver 泄露隐私信息的为 0 个.理论上 Service 组件隐藏在后台运行,不会与用户进行交互,相对来说难以引起用户的注意.实验数据显示,Service 组件更容易被恶意软件利用而引发隐私泄露,与理论分析一致.

PLDetect 检测到仅有少量的 APP 组合存在跨 APP 组件间隐私泄露问题,主要原因在于:360 Market 中的 APP 基本都属于良性应用程序,即使在一般条件下存在 ICC 漏洞的良性应用程序较多,但是通过 2 个良性 APP 配合达到泄露隐私信息的可能性非常低.更多的情况是,APP 组合中一个为恶意软件,另外一个则是存在 ICC 漏洞的良性 APP,而据 ComDroid 研究证明,60%的应用程序至少存在 1 个以上的 ICC 漏洞.因此,在真实的用户使用环境中将存在更多的跨 APP 组件间隐私泄露攻击情况.

4 结论与工作展望

本文设计并实现了一个检测跨 APP 组件间隐私泄露的自动化工具 PLDetect.PLDetect 有效地去除了与跨组件间隐私泄露无关的组件,并将虚拟主函数和插桩技术进行了有效结合,解决了在构建 CFG 时出现的路径爆炸以及代码不连续性问题.最终 PLDetect 实现了通过静态污点分析技术检测和识别与跨 APP 组件间隐私泄露相关的路径.实验表

明该系统能够有效发现和阻止跨 APP 组件间隐私泄露问题.

本文重点研究了跨 APP 组件间隐私泄露的检测和防护方法,实验评估体现了该系统的有效性.其他相关问题,例如对复杂且较少用到的 ICC 方法的分析、用户体验与隐私泄露之间的权衡等,将在今后的工作中进一步探索.

参 考 文 献

[1] Tencent. Tencent protection report: Network privacy protection report [OL]. [2018-05-04]. https://m.qq.com/security_lab/news_detail_443.html

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

[3] Bartel A, Klein J, Traon Y L, et al. Automatically securing permission-based software by reducing the attack surface: An application to Android [C] //Proc of the 27th ACM Int Conf on Automated Software Engineering. Piscataway, NJ: IEEE, 2012: 274-277

[4] DCCI. Android mobile phone privacy security report [OL]. [2018-05-04]. http://www.dcci.com.cn

[5] Arzt S, Rasthofer S, Fritz C, et al. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android APPs [C] //Proc of the 35th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2014: 259-269

[6] Li Li, Bartel A, Klein J, et al. IccTA: Detecting inter-component privacy leaks in Android APPs [C] //Proc of the 37th Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2015: 280-291

[7] 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 Conf on Security. Berkeley, CA: USENIX Association, 2013: 543-558

[8] Chin E, Felt A P, Greenwood K, et al. Analyzing inter-application communication in Android [C] //Proc of the 9th Int Conf on Mobile Systems, Applications, and Services. New York: ACM, 2011: 239-252

[9] Mann C, Starostin A. A framework for static detection of privacy leaks in Android applications [C]//Proc of the 27th Annual ACM Symp on Applied Computing. New York: ACM, 2012: 1457-1462

[10] 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

[11] Yu Le, Zhang Tao, Luo Xiapu, et al. AutoPPG: Towards automatic generation of privacy policy for Android applications [C] //Proc of ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices. New York: ACM, 2015: 39-50

[12] Rahmati A, Madhyastha H V. Context-specific access control: Conforming permissions with user expectations [C] //Proc of ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices. New York: ACM, 2015: 75-80

[13] Zhang Nan, Yuan Kan, Naveed M, et al. Leave me alone: APP-level protection against runtime information gathering on Android [C] //Proc of the 2015 IEEE Symp on Security and Privacy. Los Alamitos, CA: IEEE Computer Society, 2015: 915-930

[14] Seo J, Kim D, Cho D, et al. FlexDroid: Enforcing in-APP privilege separation in Android [C/OL] //Proc of the 2016 Annual Network and Distributed System Security Symp (NDSS). San Diego, CA: ISOC, 2016 [2018-05-20]. https://dblp.uni-trier.de/db/conf/ndss/ndss2016.html

[15] Sun Mingshen, Wei Tao, Lui John C S. TaintART: A practical multi-level information-flow tracking system for Android runtime [C] //Proc of the 2016 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2016: 331-342

[16] Song Yihang, Hengartner U. PrivacyGuard: A VPN-based platform to detect information leakage on Android devices [C] //Proc of ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices. New York: ACM, 2015: 15-26

[17] Yang Zhemin, Yang Min. LeakMiner: Detect information leakage on Android with static taint analysis [C] //Proc of the 3rd World Congress on Software Engineering (WCSE 2012). Piscataway, NJ: IEEE, 2012: 101-104

[18] Wei Fengguo, Roy S, Ou Xinming, et al. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of Android apps [C] //Proc of the 2014 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2014: 1329-1341

[19] Rasthofer S, Arzt S, Bodden E. A machine-learning approach for classifying and categorizing Android sources and sinks [C/OL] //Proc of the 2014 Annual Network and Distributed System Security Symp. San Diego, CA: ISOC, 2014 [2018-05-20]. https://dblp.uni-trier.de/db/conf/ndss/ndss2014.html

[20] Lam P, Bodden E, Lhoták O, et al. The Soot framework for Java program analysis: A retrospective [C/OL] //Proc of Cetus Users and Compiler Infrastructure Workshop. 2011 [2018-05-20]. https://sable.github.io/soot/resources/lblh11soot.pdf

[21] Bodden E. Inter-procedural data-flow analysis with IFDS/IDE and Soot [C] //Proc of the 1st ACM SIGPLAN Int Workshop on the State of the Art in Java Program Analysis. New York: ACM, 2012: 3-8

Li Zhen, born in 1993. Master candidate. His main research interests include Android security, and reverse engineering.

Tang Zhanyong, born in 1979. PhD, associate professor, master supervisor. His main research interests include network and information security, software security and protection, localization, and wireless sensor network.

Li Zhengqiao, born in 1990. Master candidate. His main research interests include information security, vulnerability mining, and reverse engineering.

Wang Hai, born in 1977. PhD, associate professor, master supervisor. His main research interests include information security, mobile security, and services computing.

Gong Xiaoqing, born in 1974. PhD, associate professor, master supervisor. Her main research interests include network and information security, and software engineering.

Chen Feng, born in 1978. PhD, assistant professor. His main research interests include wireless network, network security.

Chen Xiaojiang, born in 1977. PhD, professor, doctoral supervisor. Member of CCF. His main research interests include localization and performance issues in wireless ad hoc, mesh, and sensor networks and named data networks.

Fang Dingyi, born in 1959. PhD, professor, doctoral supervisor. Member of CCF. His main research interests include mobile and distributed computing systems, network and information security, and wireless sensor networks.