西安科技大学

学 位 论 文 诚 信 声 明 书

本人郑重声明：所呈交的学位论文（设计）是我个人在导师指导下进行的研究（设计）工作及取得的研究（设计）成果。除了文中加以标注和致谢的地方外，论文（设计）中不包含其他人或集体已经公开发表或撰写过的研究（设计）成果，也不包含本人或其他人在其它单位已申请学位或为其他用途使用过的成果。与我一同工作的同志对本研究（设计）所做的任何贡献均已在论文中做了明确的说明并表示了致谢。

申请学位论文（设计）与资料若有不实之处，本人愿承担一切相关责任。

学位论文（设计）作者签名：　　日期：　　年　　月　　日

学 位 论 文 知 识 产 权 声 明 书

本人完全了解学校有关保护知识产权的规定，即：在校期间所做论文（设计）工作的知识产权属西安科技大学所有。学校有权保留并向国家有关部门或机构送交论文的复印件和电子版。本人允许论文（设计）被查阅和借阅；学校可以公布本学位论文（设计）的全部或部分内容并将有关内容编入有关数据库进行检索，可以采用影印、缩印或其它复制手段保存和汇编本学位论文。

保密论文待解密后适用本声明。

学位论文（设计）作者签名：　　指导教师签名：刘晓建

年　　月　　日

分 类 号　　密 级

学校代码 10704　　学 号 17408070309

西安科技大学

学 士 学 位 论 文

题目：App 个人信息安全合规性自动检测系统

作者：秦子琳

指导教师：刘晓建　　专业技术职称：副教授

学科专业：软件工程　　申请学位日期：2021 年 6 月

摘　要

随着移动设备市场的持续扩大，Android 系统在电话市场占有着非常大的市场份额，手机是一种包含了大量用户隐私信息的移动设备。Android 系统是开源的，所以可能会有信息泄漏问题。虽然 Android 系统设有安全机制，但这些机制导致了系统在开放性、性能和安全性方面的妥协，无法阻挡黑客的攻击。与 Android 系统相关的恶意软件正在逐渐增加，越来越多的广告软件也被列为恶意软件。因此，如何检测恶意 Android 应用程序窃取隐私信息已经成为急迫的研究问题。

FlowDroid 可以用来保护内部开发的 Android 应用程序，以及协助甄别 Android 恶意软件。Android 生命周期的精确模型对上下文、流、域和对象完全敏感，这种设计最大限度地提高了精确度和召回率，即其目的是最大限度地减少泄漏和错误警告。但其操作过程相对复杂，且输出结果不清晰。故对 FlowDroid 进行封装，简化操作过程，直观地获取结果信息。

本文的主要工作有以下几个方面：

1. 介绍了 Android 系统架构、组件、权限机制和反编译 APK 的相关工具，以及 FlowDroid 静态分析工具的原理。

2. 完成了 App 敏感信息泄漏自动检测系统的开发，即实现通过界面方式，在后台通过命令行自动调用 FlowDroid 工具，完成对 APK 文件的分析，获取信息泄漏的具体路径。

3. 设计了一组测试用例，对 FlowDroid 静态分析工具的分析能力进行了测试，得出了相关结论。总结了 Android 系统隐私泄漏的原因，分别从 Android 设备和应用、以及用户使用设备的行为两个层面出发，提出了隐私保护的针对性措施。

关键词：静态污点分析；FlowDroid；隐私泄漏；Android；隐私保护。

ABSTRACT

With the continuous expansion of the mobile device market, Android system occupies a large market share in the mobile phone market. The mobile phone is a kind of mobile device which contains a large amount of users' privacy information. The Android system is open source, so there may be information leakage issues. Although Android system has security mechanisms, but these mechanisms lead to the system in terms of openness, performance and security compromise, can not prevent the attack of hackers. Malware associated with Android is on the rise, with more and more adware being listed as malicious software. Therefore, how to detect malicious Android applications to steal privacy information has become an urgent research problem.

FlowDroid can be used to protect internally developed Android applications and to assist in the screening of Android malware. The precise model of the Android lifecycle is fully sensitive to context, flow, domain, and object, and is designed to maximize accuracy and recall rates, i.e. the goal is to minimize leaks and false warnings. But its operation process is relatively complex, and the output result is not clear. Therefore, FlowDroid is encapsulated to simplify the operation process and intuitively obtain the result information.

The main work of this paper includes the following aspects:

1. Introduced the Android system architecture, components, permission mechanism and relevant tools for decompilating APK, as well as the principle of FlowDroid static analysis tool.

2. Finished the development of APP sensitive information leakage automatic detection system, that is, realized through the interface, through the command line in the background automatically call FlowDid tool, complete the analysis of APK file, obtain the specific path of information leakage.

3. A set of test cases is designed to test the analysis ability of FlowDroid static analysis tool and draw relevant conclusions. This paper summarizes the reasons for privacy leakage in Android system, and puts forward targeted measures for privacy protection from two aspects: Android devices and applications, and users' behavior when using devices.

Key Words：Static taint analysis; FlowDroid; Privacy leak; Android; Privacy protection.

目 录

1 绪　论

1.1 研究背景和意义

根据相关研究[1]，Android 在移动电话市场的份额不断增长，目前已达到 81%。随着 Android 手机的普及，它们成为了侵犯用户隐私敏感数据的重要目标。Felt 等人对不同类型的 Android 恶意软件[2]进行分类，发现恶意 Android 应用构成的主要威胁之一是获取隐私信息，是通过将位置信息、个人信息、图片、短信等敏感信息泄漏给攻击者。但是，即使是那些没有恶意且经过精心编程的应用程序也可能遭受此类泄漏，例如，当其包含广告库[3]时，公共库会提取用于定向广告的个人信息，如唯一标识符(例如 IMEI、mac 地址等)、国家或位置信息等。因此，如何检测 Android 恶意应用是否窃取隐私信息已经成为了紧迫的研究问题。

由于 Android 系统的开源特性、应用分发市场的无序竞争以及恶意 App 制作的利益驱动，大量不同形态和变种的恶意 App 不断被制作、发布和广泛传播，它们利用移动智能终端上普遍存在的网络链接和个人信息（如通讯录、银行证书和网页浏览历史等）作为途径，窃取敏感信息，获取非法利益，给用户带来巨大的经济损失。因此，有必要设计和开发一种 App 个人信息安全合规自动检测工具，发现隐藏在 App 逻辑中违法违规收集使用个人隐私信息的行为，为政府监管部门的认定和执法提供技术支撑，为 App 运营者、开发者自检、自评估提供辅助检测工具，为用户安全使用 App 提供技术保障。

1.2 研究现状

1.2.1 国内研究现状

刘效伯[4]为 Android 应用程序设计并开发了一个隐私泄漏检测系统：动态分析，进行了基于 TaintDroid 污点检测技术的一系列改进。依据不同需要，实现高覆盖率、高效率或自动化方法采集应用程序的敏感传输数据；将静态分析将提取的应用特征信息与敏感传输数据进行关联,并进行风险评估过滤获取隐私泄漏信息。该系统使用图形用户界面进行隐私泄漏检测操作，并以报告的形式反馈检测结果。

张小贝[5]综合了特征检测、静态检测、动态检测的特征,提出一套高效可行的联合检测方案,并在此基础上构建一套 web 系统。这是一项在线服务，用户可以检测 APK 安装包中的恶意代码，并开发 Android 终端检测系统，检查 Android 终端应用程序的安全性。

孙贝[6]在深入分析安卓系统安全现状和现有隐私泄漏检测方法的优缺点的基础上，提出了一种有效的、改进的安卓隐私泄漏检测方法。该方法使用与 Android 应用程序中特定软件行为过程相关的上下文信息。通过提高检测精度，可以有效地减小静态污点分析代码的范围，提高检测效率。

曾述可[7]介绍了一种构建 Android 隐私保护机制攻击图的方法。这种方法首先确认 Android 系统的界面和个人隐私数据,并读取或者发送隐私数据的敏感接口，以便阅读或发送个人隐私数据；然后，在静态分析工具的基础上，分析这些敏感接口的具体执行路径。由此，创建了一个隐私保护机制的攻击图。

王奕钧[8]结合隐私泄漏的检测与应用程序的改进，提出了一种基于权限机制的隐私保护模型 PRIMOD。主要研究方向是分析 Android 平台，包括系统结构和应用组件，以及其本身的安全机制及漏洞分析。

欧阳金彬[9]基于 FlowDroid 静态数据流分析框架实现了 FragHarden，这是一种具有更全面分析结果的隐私数据泄漏检测工具。大量测试集的实验证明，动态和静态注册的 Fragment 对隐私信息泄漏检测结果有不同的影响，FragHarden 的有效性得以验证。并对 FragHarden 的执行效率进行了比较和评价，平均性能开销为 16.7%。

袁洋[10]提出了一种动静结合的隐私信息保护系统。静态部分静态分析应用程序文件并生成保护方案，然后通过动态运行过程中通过解析方案保护隐私数据。通过结合 Java 语言堆栈的特点和 Android 系统 API 的分析，解决了多组件场景中隐私数据传播路径的问题。

张力智[11]从 Android 权限机制出发，改进了 FlowDroid 静态检测方法。解决了 FlowDroid 检测消耗时间长、内存资源过剩等问题。消除了静态检测的冗余分析，提高了检测效率，引入了污染数据流的风险水平和传播，提高了静态检测的精度。基于上述方法，设计并实现了 BPFlowDroid 工具。

胡英杰、张琳琳、赵楷[12]等人提出了一种基于静态污点分析技术的隐私数据泄漏的检测方法。通过提取对 Android 敏感的权限和 API，在它们之间创建关系，并为 Android 应用程序创建函数调用图，可以检测大型应用程序中潜在的隐私数据泄漏。实验结果表明，此方法适用于大规模检测。

1.2.2 国外研究现状

Arzt S，Rasthofer S，Fritz C 等作者[13]提出 FlowDroid，是一种新颖且高度精确的 Android 应用程序静态污点分析工具，可以同时保持高效率和高精度。Android 生命周期

的精确模型允许分析和处理对 Android 框架调用的回调；Android 系统上下文、流、字段和对象敏感性，可以进行分析，以减少错误警报。

Shancang，Li Junhua，Chen Theodoros，Spyridopoulos 等作者[14]在 Android 系统中，开发了一个实时监控系统，用来追踪潜在的隐私数据滥用。此应用程序可以跟踪全部已安装完成的应用程序的权限请求，并分析潜在的隐私滥用行为。

Youngho Kim，Tae Oh，Jeongnyeo Kim 等人[15]为衡量移动应用程序的隐藏意图程度，以保持其泄漏活动对用户不可见。使用 Taint Droid 移植的模拟器，对用户事件和隐私之间的时间距离进行了实验泄漏。实验表明，大多数泄漏案例是由用户显式事件或隐式用户参与驱动的，这些事件使用户意识到泄漏。这些发现可以帮助恶意软件检测系统通过考虑恶意意图来降低误报率。

Lo N W，Yeh K H，Fan C Y 等[16]为 Android 平台提供了 LRPdroid 用户隐私分析框架。LRPdroid 的目的是检测信息泄漏，评估用户隐私泄漏，评估应用程序隐私风险。使用原型系统评估了两种常见的应用程序使用场景，验证了 LRPdroid 框架在用户隐私管理方面的可行性和实用性。

Yifei Zhang，Yue Li，Tian Tan 等人[17]指出了 Android 应用程序的不完全信息环境（IIE）的普遍性，并设计了 Ripple，是 Android 应用程序的第一个 IIE 感知静态反射分析，比字符串推断的结果更可靠。通过 Ripple，FlowDroid 能够检查出数百个隐私信息泄漏，否则这些泄漏将被遗漏。

Gulshan Shrivastava，Prabhat Kumar 等人[18]设计了一个名为 SensDroid 的框架，评估 Android 意图和权限的效率，将其作为一种区分特性，通过敏感分析技术来识别恶意应用程序。通过将这些特征与其他恶意软件检测属性相结合，实现了效率的升级。使用了从官方和第三方 Android 应用市场收集的足够数量的样本，对多个参数进行评估，并与现有技术进行比较，实现了高识别率的恶意软件分类。

Kang Hongzhaoning，Liu Gang，Wu Zhengping 等人[19]基于特征权限和风险评级，提出了一种改进的静态检测方法，以解决 FlowDroid 检测过于复杂和误报的问题。首先利用卡法检验提取与恶意应用相关的权限，利用互信息用于对权限进行分组，生成特征权限簇。其次，为了识别危害数据流，提出了一种基于权限和权限组合的风险计算方法。经过实验验证，该方法的检测效率得到了显著提高。

1.3 研究内容

此设计是运行在 Windows 上的 Android App 个人信息安全合规性自动检测软件工具的制作。重点任务是：

（1）学习 FlowDroid 静态分析工具的基本原理。搭建并运行 FlowDroid 环境，编写测试代码对 APK 文件进行检测，并读懂输出结果。

（2）对整个软件进行整体构思，并绘制出初步的流程图，对系统要实现的目标先罗列一份说明。

（3）通过使用 Java + Python 语言对自动检测系统进行程序编写。

（4）在已经设计出的自动检测系统中通过软件测试。

（5）针对 APK 文件进行定点分析，测试 FlowDroid 工具的检测性能。

1.4 论文结构

论文主要由以下几个部分组成：

第一章，绪论。本章介绍了研究的背景和意义、国内外研究的现状、研究的具体内容和论文的结构。

第二章，背景知识。本章介绍了系统开发所需的前置知识。

第三章，系统总体设计。本章确定了系统的流程设计、业务流程以及系统架构，并辅以图片进行理解。

第四章，系统实现。本章介绍了系统的功能结构、各模块的具体实现和核心功能编码。

第五章，系统测试。本章介绍了系统的具体测试实施，并对系统进行评价。

第六章，Android 隐私保护研究。本章为 Android 用户提供了对隐私泄漏原因、隐私保护方法和使用建议的分析。

第七章，总结和展望。

1.5 本章小结

本章首先分析了课题的研究背景及意义。设计出一个 App 个人信息安全合规性自动检测系统，可以为用户安全使用 App 提供技术保障。在第 2 小节，对国内外的研究现状进行了分析。在第 3 小节，列举了本系统的研究内容。在第 4 小节，对论文结构做了梳理。

2 背景知识

2.1 Android 基础

Android 应用程序的无限市场和开源使得它成为第三方应用程序的流行平台。Android 通过一个广泛的 API 支持第三方开发，该 API 允许应用程序访问电话设备(例如，摄像头)、WIFI 和蜂窝网络、用户数据和电话参数，在安装过程中，应用程序权限系统控制对 API 中与隐私和安全相关部分访问。

Android 的系统架构和所有操作系统的系统架构是一样的，都使用了分层的架构。Android 系统架构分为四层，从低层到高层依次为：

（1）Linux 内核层：主要涉及底层驱动和一些系统服务，提供了基本的系统功能。

（2）系统运行库层：Android 提供了一组 C/C++库，以确保不同平台组件的使用。该层还包含了一个由一组 Android 核心库集和一个 Dalvik 虚拟机组成的 Android Runtime。

（3）应用程序框架层：该层用于开发人员自己编写的每一个应用程序和系统内置的应用程序，不需要关心底层的具体实现。

（4）应用程序层：一般来说，应用层的开发是在该层上进行的，包括系统内置的应用程序组，使用的是 Java 语言。

Android 应用程序的四个组件是活动、服务、广播接收器和内容提供者。Android 应用程序多由一个或多个 Android 组件构成，组件可以相互通信和协作，实现了应用程序的核心功能。一个 Activity 是应用程序和用户之间交互的接口界面。Service 在后台运行，并执行不需要用户交互的功能，如播放和录制声音。广播接收者接收在广播中注册的广播，如开关机或地理位置的更改，应用程序会对其做出响应。内容提供者的主要任务是向外界传输数据，应用程序可以通过内容提供者将自己的数据传输到其他应用程序。恶意应用程序可以通过内容提供程序访问用户通讯簿、呼叫历史记录和其他数据保护信息。

反编译 APK，就是利用反编译软件对 APK 进行反编译，获取源代码、图像、XML 资源等程序文件。反编译需要用到的工具有：

（1）apktool，用于获取资源文件，提取图片文件、布局文件，和一些 XML 的资源文件；

（2）dex2jar，将 APK 反编译为 Java 源码，即将 classes.dex 转化为 jar 包；

（3）jd-gui，用于查看转换后的 jar 包，即查看 Java 文件。

2.2 Android 权限机制

Android 系统是基于 Linux 操作系统的扩展，提供了权限机制[20]。应用程序可以执行的操作被指定为限制软件操作系统或其他软件的能力。Android 的权限机制要求开发者在 Android 的 Manifest.xml 中申请他们需要的权限，并在安装过程中获得用户的同意，通过调用相关的 API 来访问系统资源和功能组件。

Android 系统手机用户可以通过 Android Market 或 Amazon Appstore 安装第三方应用。这些第三方应用程序的质量和可靠性差异很大，所以 Android 认为所有应用程序都存在潜在的漏洞或恶意。每一个应用程序在具有低权限用户 ID 的进程中运行。默认情况下，应用程序只能访问自己的文件。应用程序是用 Java 编写的（可以附加本地代码），每个应用程序都在自己的虚拟机上运行。

Android 通过限制应用程序访问系统资源的权限来保护敏感系统和用户信息。Android 使用一种粗粒度的权限管理机制，在授予权限后不再审查运行进程；因此，恶意应用利用用户对权限的无知和 Android 权限机制的粗粒度权限管理来访问甚至泄漏敏感信息。Android 8.0 提供 135 个权限和相应的 API 来访问系统资源，分为三种威胁级别：

（1）Normal 权限保护对 API 调用的访问。这些调用可能会干扰用户，但不会对用户造成伤害。例如，SET_WALLPAPER 控制更改用户背景墙纸的能力。

（2）Dangerous 权限控制对潜在有害 API 调用的访问。例如，与货币交易和个人信息收集相关的 API 调用。比如，发送 SMS 或读取联系人列表将需要 Dangerous 权限。

（3）Signature/System 权限控制对最危险特权的访问。例如，可以管理备份进程或删除应用程序包。这些权限很难获得：Signature 权限仅授予由设备制造商证书签名的应用程序，而 Signature/System 权限授予已签名或安装在特殊系统文件中的应用程序。这些限制主要限制预先安装的应用程序的 Signature/System 权限，而其他应用程序的 Signature/System 权限请求将被忽略。

应用程序可以定义自己的权限来保护自己，但是我们感兴趣的是由 Android 定义的保护系统资源的权限。在分析的任何阶段，都不考虑开发人员定义的权限，这些权限不是操作系统的一部分。

与系统 API、数据库和消息传递系统交互可能需要权限。公共 API 描述了 8648 种方法，其中一些方法受到权限保护。用户数据存储在内容提供者中，需要权限才能与某些系统内容提供者一起工作。例如，必须允许应用程序拥有 READ_CONTACTS 权限以执行 READ 请求操作。应用程序还需要权限获取操作系统的意图（即消息）。消息通知应用程序事件，如网络连接的更改，系统发送的某些消息只发送给具有适当权限的应用程序。

此外，发送模仿系统消息内容的消息需要授权。

2.3 FlowDroid 静态分析工具

2.3.1 相关概念

1.Jimple 中间表示

为了降低分析的复杂性，基于一种简化的中间表示形式（Jimple）进行静态分析。Jimple 只有 15 中不同类型的语句，可以包含 29 种不同类型的表达式。不同于 Java 字节码是一种基于堆栈的语言，执行操作前会将操作符放在堆栈上，并依次从堆栈中弹出参数进行计算，随后将计算的结果放回堆栈上。在 Jimple 中，没有任何堆栈。更恰当地说，Jimple 更接近 Java 源代码语言，因为其使用的本地变量值是可以从中读取和分配的本地变量。

2.Source 和 Sink 定义

数据流分析的目标是找到 source 和 sink 之间的连接，即需要定义什么是 source、什么是 sink。一般来说，我们关心的是隐私信息是否泄漏给潜在的不受信任的外部人员。例如，如果应用程序从手机中读取用户的通讯录，并将其传输到网络服务器，就会出现这种情况。本例中，source 是读取通讯录数据的方法，sink 是将此数据（或从中派生的任何数据）传输到远程服务器的方法，用户的设备 ID 被读取并作为 SMS 消息的文本发送出去。即在图 2-1 中，getDeviceId()方法（在第 4 行调用）是 source，sendTextMessage()方法（在第 8 行调用）是 sink。在进行任何数据流分析之前，必须确定 source 方法和 sink 方法，它们构成了隐私分析时应该关心的信息流。相反地，例如，将一个常量值流入显示给用户，通常不是应该被关心的信息流）。

```
1 void onCreate() {
2     // Get the data
3     TelephonyManager mgr = (TelephonyManager) this.getSystemService(TELEPHONY_SERVICE);
4     String deviceId = mgr.getDeviceId();
5
6     // Leak the data
7     SmsManager sms = SmsManager.getDefault();
8     sms.sendTextMessage("+49 1234", null, deviceId, null, null);
9 }
```

图 2-1 简单的数据泄漏示例

3.信息泄漏路径

Java 程序和 Android 应用程序都在字段中存储数据，并读取数据。在图 2-2 的示例中，A 类的 fld 对象被污染，只有第 5 行语句可以访问此堆对象，从而构成真正的泄漏。

sink 方法的其他调用不应被检测为污染的 sink，因为这将会导致泄漏信息的误报。实际中，每当在污染分析过程中创建新的访问路径时，必须将其简化为相应的符号访问路径。

```
1 void onCreate() {
2     A a = new A();
3     A b = new A();
4     a.fld = source();
5     sink(a.fld);  // true leak
6     sink(a.foo);  // no leak
7     sink(b.fld);  // no leak
8 }
```

图 2-2 信息泄漏路径示例

2.3.2 FlowDroid

1.功能简介

FlowDroid 是一个用于 Android 应用程序和 Java 程序的静态分析工具。可以用来保护内部开发的 Android 应用程序，以及协助甄别 Android 恶意软件。无论数据流是由于粗心还是恶意所致，都可以用 FlowDroid 来检测数据流。

对于恶意的情况，假设攻击者模型如下：攻击者可以向应用程序提供任意恶意的 Dalvik 字节码。通常，攻击者的目标是用户授予的特别广泛的权限泄漏私有数据。FlowDroid 对安装环境和应用程序输入做出合理的假设，这意味着攻击者也可以自由地篡改这些。然而，FlowDroid 假定攻击者没有办法绕过 Android 平台的安全措施或利用侧通道。此外，我们假定攻击者不使用隐式流来掩盖数据泄漏。考虑到现有的恶意软件，这是一个非常合理的假设。FlowDroid 充分地模拟了 Android 特定的挑战，如应用程序生命周期或回调方法，这有助于减少漏报或误报。

2.Source 和 Sink 的处理

FlowDroid 为 Java 程序提供了一个默认的实现，即假设 source 始终是方法调用。通常情况下不需要这样，自定义 Source 和 Sink 时可以将任何语句定义为 Source 和 Sink。默认的方法是，检测目标方法是否在名为 SourcesAndSinks.txt 的文件中。如果目标方法在该文件中，其返回值将被无条件地污染。只有当方法返回 void，或者用户代码不处理返回值（即调用语句没有被赋值），并且被调用地方法不是静态地时，调用的对象才会被污染。当无条件地污染 source 值时，source 值中的所有字段也会受到污染，即生成的访问路径总是以一个星号结束。通常，source 返回新的数据对象，并且合理地假设这些对象中的所有值都是敏感的。特别是在 Android 的上下文中，如果需要用到特殊权限，调用 source 方法值会发生权限检查。因此，该方法返回的所有值都被保护，可以假定为已受污染。此外，这些简单的语义与 source 和 sink 定义相匹配，我们从中获取默认的

SourcesAndSinks.txt 文件。如果接口中的方法或抽象类中的抽象方法被标记为 source 或 sink，则此定义适用于该方法的所有实现者。

3.FlowDroid 的分析过程

FlowDroid 基于 Soot[21]，在字节码级别工作，不需要访问应用程序的源代码。FlowDroid 通过解析 Android 应用程序的 APK 文件，将 Java 代码转换为 Jimple 中间代码，模拟一个 Android 应用程序的生命周期来处理回调函数，并生成一个调用图（CG）和程序间控制流图（ICFG）来跟踪污染（如图 2-3 所示）。它使用解释有限分布子集（IFDS）来建模数据流传播，并生成完整的跟踪污染，即通过 Heros 框架生成完整的、受污染的数据流路径。因此，FlowDroid 对计算和内存资源有很高的要求。

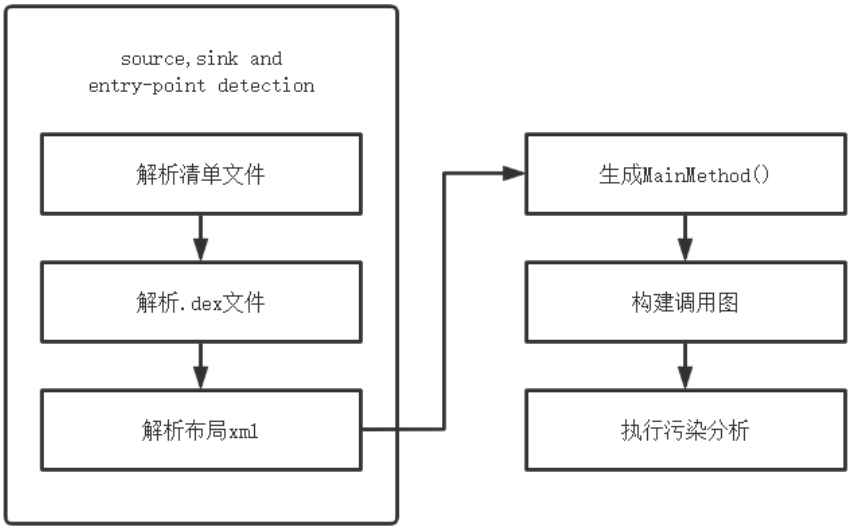

图 2-3 FlowDroid 工作流

使用 FlowDroid 对 Android 应用程序进行静态分析时，FlowDroid 的输入应为 Android 的应用包 APK 文件，将 APK 文件进行反编译后，分别分析 AndroidManifest.xml 文件、classes.dex 文件和 layout 中的布局文件。通过以上文件的解析获取应用程序所对应的 source、sink 和 entry-point detection，获取其生命周期和回调函数列表。随后，根据获取的生命周期和回调函数列表生成 MainMethod()方法，解析 APK 中的.dex 文件，转换为 Jimple 中间代码，并生成 CG 和 ICFG。FlowDroid 输出是程序的数据流和泄漏路径。

FlowDroid 还提供了一个最常见的 Android 框架方法的精确模型。对于其余的方法，它采用保守策略，用至少一个被污染的参数，污染所有参数和方法的返回值。为了提高性能，FlowDroid 允许用户手动指定他们想要包含或排除[22]分析的包和方法。FlowDroid 只能重新解决参数为常量字符串的反射调用；其异常处理机制是基于 Soot 的异常处理机制。在最初的设计中，该工具不支持隐式流跟踪，但新版本的 FlowDroid 可以进行处理隐

式流[22]。此外，FlowDroid 进行程序间的数据流分析，因此不能处理 ICC 或 IAC。

2.4 Android 隐私保护研究

2.4.1 隐私泄漏原因分析

Android 手机中存储了大量的用户隐私信息，根据信息来源的不同，可以分为以下四类[14]：

（1）设备标识信息。设备标识信息是指手机交付或在手机上安装 SIM 卡后的硬件信息，包括手机号码、国际移动设备标识（IMEI）、国际移动用户识别码（IMSI）、IC 卡标识符（ICCID），用户通常不会修改此信息。设备标识信息通过应用程序编程接口 API 可以获取到。有些应用程序在注册和登录时，会对设备标识信息进行绑定，恶意应用程序就会由此识别不同的用户，进行相应的攻击。

（2）存储在设备中的信息。手机上可以存储的信息量已经从几百 M 增加到几百 G。用户使用智能手机作为移动存储设备，把他们的工作文件、日记、照片等保存到 SD 卡。只要应用程序具有 SD 卡的读/写权限，就可以访问这些信息。

（3）从传感器获取的信息。智能手机传感器变得更加敏感，收集的信息更加准确，恶意应用程序可以利用这些传感器获取实时信息。智能手机的健康监测功能日益普及，智能手机的信息采集功能也在不断扩展，比如带有传感器的智能手环，它可以通过智能手机和蓝牙连接。恶意应用程序可以获取用户一天内的步行数、心跳和睡眠等。

（4）App 生成的信息。用户在使用应用程序时也会生成隐私数据和信息。许多应用程序程序员，在日志中记录了一些应用程序信息，便于记录程序异常并调试程序。用户使用的聊天软件，会创建聊天记录；用户使用的购物软件，会生成交易记录。这些记录通常是为了让业务功能更具可持续性，改善用户体验而设计的。其他应用程序很难在未经许可的情况下获取这些信息，但是，如果应用程序本身被打包过，则使用该应用程序的用户生成的数据可能会被泄漏。此外，恶意应用程序为了窃取这些信息，可能会滥用系统弱点。

现如今，Android 应用程序的隐私泄漏方式主要包括以下几种[18]：

（1）通过网络连接泄漏。网络连接是最常见的隐私数据泄漏的方法，也是一种非常隐秘的方法。只要应用程序可以访问 Internet，就可以通过网络发送隐私数据。而网络访问是应用程序中非常常见的权限，大多数应用程序都必须申请连接网络的访问授权。

（2）通过短信泄漏。如果应用程序有发送短信地权限，则可以在不通知用户的情况下，将包含用户隐私信息的消息发送到后台的特定号码。

（3）通过其他组件泄漏。通过跟踪程序之间的信息流地方式，有许多工具可以发现隐私信息的泄漏。为了避免信息泄漏，恶意应用程序在组件收到数据保护信息后不会直接发送隐私信息，而是通过一些方式将隐私数据传输给其他组件，来逃避某些工具的检测。

（4）通过第三方包泄漏。为了提高程序编写的效率，一般都是基于第三方包提供的基础功能来开发新功能的。除此之外，许多应用程序也会加入广告包。这些第三方包和应用程序在一个进程中运行，用于应用程序的所有权限，可以随意访问其中的数据。当应用程序中加入了恶意第三方包，就会导致隐私泄漏的发生。

根据隐私信息的来源和泄漏方式，可以将隐私泄漏的原因概括为以下几种：

（1）应用程序的安全性没有保障。在大数据的时代背景下，有的正规应用程序也开始窃取用户隐私信息，用于进行大数据分析。Android 用户多使用第三方市场、发布在网站中的链接进行下载应用程序，这种方式没有采取监管，应用程序的安全性自然也就没有任何的保障。

（2）Android 系统的安全性没有保障。通常情况下，Android 操作系统自身不会进行检查更新，直接导致了许多 Android 手机仍运行的是低版本的操作系统，这些系统中存在的安全漏洞极有可能导致隐私信息的泄漏。

（3）数据的安全性没有保障。通常情况下，应用程序使用中的痕迹信息以文件的形式存储在该应用程序的数据文件夹下。如果手机丢失，或者数据文件夹被窃取，隐私信息也会泄漏。

（4）网络的安全性没有保障。绝大多数应用程序在使用中必须访问网络，即需要获取网络方面的权限。而绝大部分的隐私泄漏都是通过网络发送出去的。

2.4.2 隐私保护方法

针对上述提出的隐私泄漏原因并不能从根本上解决，但也可以采取一些针对性的措施。一方面，可以从 Android 设备和应用的层面出发，用技术的方法对个人隐私信息进行保护；另一方面，也可以从用户使用设备的行为出发，减少可能会导致个人隐私信息泄漏的行为。

为了达到保护用户隐私信息的目的，技术方面应考虑从权限入手，阻止应用程序获取相关的权限；也应考虑更好的保护 APK 包中的隐私数据，从源头上切断发送数据。

（1）设置权限时间。可以考虑在系统端设置敏感权限的授权时间，在授权时间后，自动解除授权，如果用户仍需使用该权限，则重新进行授权。授权的时间期限也可以设为自定义的，让用户根据自己的需要进行选择。

（2）伪装隐私信息。采用伪造的相关数据，在不影响用户正常使用应用程序的基础上，避免了真实隐私信息的泄漏。伪造隐私信息只能针对设备信息和位置信息有用，其它的信息通常存储在应用程序的数据库中，可以使用加密的手段进行解决。目前最新的安全应用程序（例如，安卓隐私大师）已经设置了类似功能。

（3）修改 APK 文件[4]。修改 APK 文件，可以从源头上拒绝 App 访问隐私信息。通过反编译提取 APK 中的.xml 文件，使用 XML 文件编辑器将声明权限的条目删掉，然后重新打包为 APK 文件。应用程序就不会向系统申请本来需要的权限，当 App 运行到相应流程的时候，会被系统阻止，即拒绝了隐私信息的泄漏行为。

2.4.3 Android 用户使用建议

关于 Android 应用程序的隐私保护方法，基本上都需要对系统或者应用程序进行修改，需要一定的专业技能。如果操作不当，可能会损坏设备，甚至导致更严重的隐私泄漏问题。如果用户在使用 Android 应用程序时，养成良好的使用习惯，也可以减少隐私信息的泄漏。

（1）使用正规的应用程序。因为 Android App 开发和发布都很自由，可以通过各种渠道获取大量的应用程序。这其中就充斥了许多恶意应用程序，给用户的选择造成了巨大的困难。因此，当用户需要获取 App 时，应尽量从官方网站或者知名应用平台进行下载，不要使用来历不明的软件。

（2）设置复杂的密码。大多数情况下，访问应用程序需要账号和密码进行身份认证。而账号通常对应用户的手机号码或者邮箱账号等容易获取的信息，因此密码的复杂程度直接决定了其所保护的用户信息的安全性。用户设置密码，应对不同的应用程序设置不同的密码，应使用包含大小写字母、特殊字符和数字等组成的复杂密码。对于设计隐私信息较多的应用程序，需要每次手动输入密码进行登录，并且定时更换密码。

（3）使用安全的网络。现如今，公共 WIFI 几乎随处可见，然而公共 WIFI 的安全性却十分堪忧。用户要谨慎使用 WIFI，连接前确认是否属于正规来源的网络。不要开启自动连接 WIFI，在需要使用时进行手动连接，不需要上网时及时关闭。使用设计重要隐私的应用程序时，建议使用运营商连接网络。

（4）留意短信费用。部分恶意应用程序会在后台会通过短信的方式，给指定号码发送含有用户隐私数据的短信。因此，当用户发现自己手机号码的短信费用突然增多，或者经常给未知号码发送短信时，就应该怀疑自己是否安装了恶意应用程序。

2.5 小结

本章在第 1 小节，介绍了 Android 的基础知识，便于理解 Android 权限机制。在第 2 小节，对 Android 权限机制进行了简要的分析。在第 3 小节，介绍了 FlowDroid 静态分析工具，是整个系统设计和实现的依据。在第 4 小节，分析了 Android 隐私保护的相关内容，总结了产生隐私泄漏的原因，提出了隐私保护的方法。

3 FlowDroid 性能测试实验

3.1 FlowDroid 性能测试实验

在完成上述系统设计和实现后，在本系统的基础上，针对 FlowDroid 静态分析工具的检测能力做了如下测试。

3.1.1 分析能力测试实验

1. 编写 Demo 进行分析检测

通过实现简单的、具有内存泄漏的 App，将其打包为 APK 文件。使用 FlowDroid 静态分析工具进行分析，将在本地生成的日志文件、漏洞文件、程序调用图文件、控制流图文件，与实现的 App 进行对比。

测试 Demo_01 将污染值存储在数组中，使用 System.arraycopy 将数据复制到新数组中，然后将其泄漏到日志中。source 方法为 getDeviceId()，sink 方法为 Log.i()，如图 3-1 所示。经 FlowDroid 分析，可以测试有一处信息泄漏，测试输出结果如图 3-2 所示。程序的调用图如图 3-3 所示。

```
protected void onCreate(Bundle paramBundle) {
  super.onCreate(paramBundle);
  setContentView(2130903040);
  String str = ((TelephonyManager)getSystemService("phone")).getDeviceId();
  String[] arrayOfString = new String[1];
  System.arraycopy(new String[] { str }, 0, arrayOfString, 0, 1);
  Log.i("DroidBench", arrayOfString[0]);
}
```

图 3-1 Demo_01 的关键代码

```
Result as follows
[$r6 = virtualinvoke r5.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() ->
staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("DroidBench", $r6)]
```

图 3-2 Demo_01 的泄漏信息

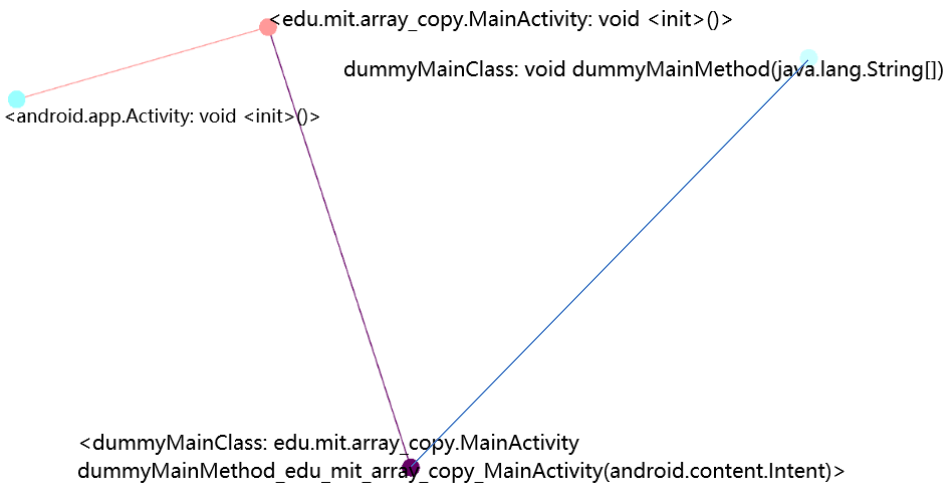

图 3-3 Demo_01 的调用图

2. 编写 Demo 进行跨构件分析检测

将 Source 和 Sink 设置到不同的构件中，使用 FlowDroid 静态分析工具进行分析，测试是否可以检测出信息泄漏。

测试 Demo_02 的 IMEI 在一个 Activity 中获取并存储在 SharedPreferences 中，该 SharedPreferences 被另一个 Activity 泄漏。其 source 方法为 getDeviceId()，sink 方法为 putString()，如图 3-4 所示。经 FlowDroid 分析，可以测试有一处信息泄漏，测试输出结果如图 3-5 所示。程序的调用图如图 3-6 所示。

MainActivity.java

```
protected void onCreate(Bundle paramBundle) {
  super.onCreate(paramBundle);
  setContentView(2130903040);
  String str = ((TelephonyManager)getSystemService("phone")).getDeviceId();
  SharedPreferences.Editor editor = getSharedPreferences("MyPrefsFile", 0).edit();
  editor.putString("imei", str);
  editor.commit();
}
```

AnotherActivity.java

```
protected void onCreate(Bundle paramBundle) {
  super.onCreate(paramBundle);
  setContentView(2130903040);
  Log.i("DroidBench", getSharedPreferences("MyPrefsFile", 0).getString("imei", ""));
}
```

图 3-4 Demo_02 的关键代码

```
Result as follows
[$r4 = virtualinvoke r3.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() ->
interfaceinvoke $r6.<android.content.SharedPreferences$Editor:
android.content.SharedPreferences$Editor putString(java.lang.String,java.lang.String)>("imei", $r4)]
```

图 3-5 Demo_02 的泄漏信息

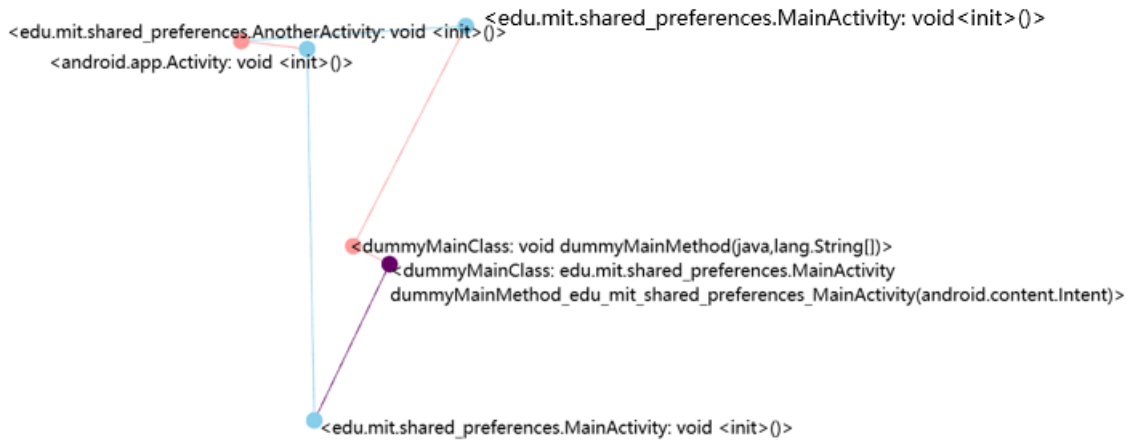

图 3-6 Demo_02 的调用图

3. 编写 Demo 进行跨 callbacks 分析检测

将 Source 和 Sink 设置到不同的 callback 中，使用 FlowDroid 静态分析工具进行分析，测试是否可以检测出信息泄漏。

测试 Demo_03 在另一个回调的处理程序中注册了一个新的回调，第二个处理程序泄漏了第一个处理程序获得的数据。其 source 方法为 getDeviceId()，sink 方法为 sendTextMessage()，如图 3-7 所示。经 FlowDroid 分析，可以测试有一处信息泄漏，测试输出结果如图 3-8 所示。程序的调用图如图 3-9 所示。

Button1Listener.java

```
public void onClick(View paramView) {
  TelephonyManager telephonyManager = (TelephonyManager)this.act.getSystemService("phone");
  this.act.imei = telephonyManager.getDeviceId();
  ((Button)this.act.findViewById(2131230722)).setOnClickListener(new Button2Listener(this.act));
}
```

Button2Listener.java

```
public void onClick(View paramView) {
  SmsManager.getDefault().sendTextMessage("+49", null, this.act.imei, null, null);
}
```

图 3-7 Demo_03 的关键代码

```
Result as follows
[$r5 = virtualinvoke r4.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() ->
virtualinvoke $r2.<android.telephony.SmsManager:
void sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.PendingIntent)>
("+49", null, $r4, null, null)]
```

图 3-8 Demo_03 的泄漏信息

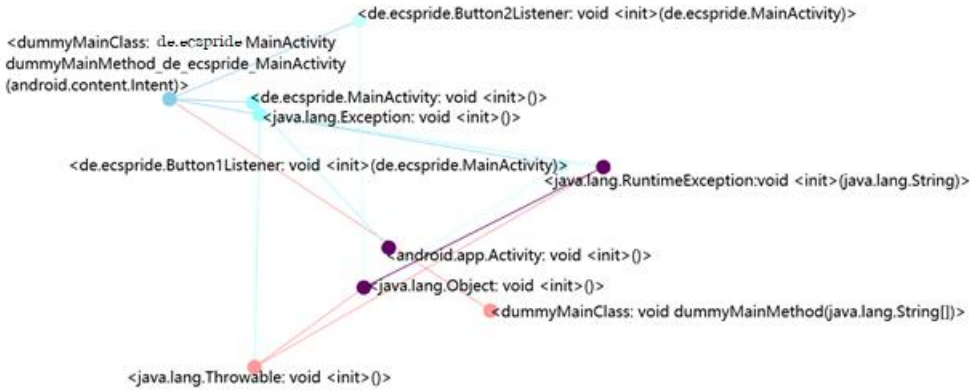

图 3-9 Demo_03 的调用流图

4. 编写 Demo 进行跨函数分析检测

将 Source 和 Sink 设置到不同的方法中，使用 FlowDroid 静态分析工具进行分析，测试是否可以检测出信息泄漏。

测试 Demo_04 启动另一个 Activity 的 Intent 通过 LinkedList 的 add()传递，并使用 LinkedList.get()检索，检索到的 Intent 用于启动泄漏的 Activity。其 source 方法为 getDeviceId()，sink 方法为 Log.i()，如图 3-10 所示。经 FlowDroid 分析，无法分析出信息泄漏，测试输出结果如图 3-11 所示。程序的调用图如图 3-12 所示。

```
protected void onCreate(Bundle paramBundle) {
  super.onCreate(paramBundle);
  setContentView(2130903040);
  String str = ((TelephonyManager)getSystemService("phone")).getDeviceId();
  Intent intent = new Intent((Context)this, InFlowActivity.class);
  intent.putExtra("DroidBench", str);
  LinkedList<Intent> linkedList = new LinkedList();
  linkedList.add(intent);
  startActivity(linkedList.get(0));
}
```

```
protected void onCreate(Bundle paramBundle) {
  super.onCreate(paramBundle);
  setContentView(2130903040);
  Log.i("DroidBench", getIntent().getStringExtra("DroidBench"));
}
```

图 3-10 Demo_04 的关键代码

```
Found 0 leaks
Exception as follows
null
Result as follows
null
```

图 3-11 Demo_04 的泄漏信息

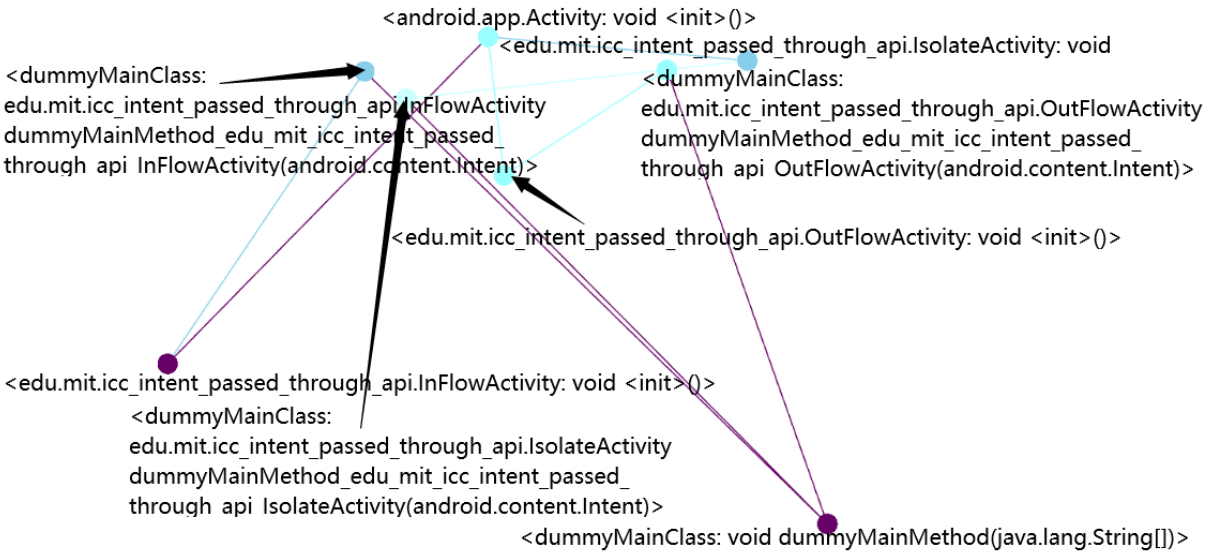

图 3-12 Demo_04 的调用图

3.1.2 性能测试实验

重新设置 SourceAndSink.txt 文件，给出确定的单个 Source 和单个 Sink，传入所实现的 App 文件，实现定点分析。

1. 编写 Demo 进行分析检测

在重新设置的 SourceAndSink.txt 文件中，使用 FlowDroid 静态分析工具进行分析，对比修改 SourceAndSink.txt 前后的运行时间和内存。修改后的 SourceAndSink.txt 文件如图 3-13 所示，测试结果如表 3-1 所示。

SourcesAndSinks.txt

```
1  <android.telephony.TelephonyManager: java.lang.String getDeviceId()> android.permission.READ_PHONE_STATE -> _SOURCE_
2
3  <android.util.Log: int i(java.lang.String,java.lang.String)> -> _SINK_
```

图 3-13 Demo_01 的 SourceAndSink.txt 文件

表 3-1 Demo_01 的测试结果

	时间消耗	内存消耗
修改前	0.025s	44MB
修改后	0.014s	18MB

2. 编写 Demo 进行跨构件分析检测

在重新设置的 SourceAndSink.txt 文件中，使用 FlowDroid 静态分析工具进行分析，对比修改 SourceAndSink.txt 前后的运行时间和内存。修改后的 SourceAndSink.txt 文件如图 3-14 所示，测试结果如表 3-2 所示。

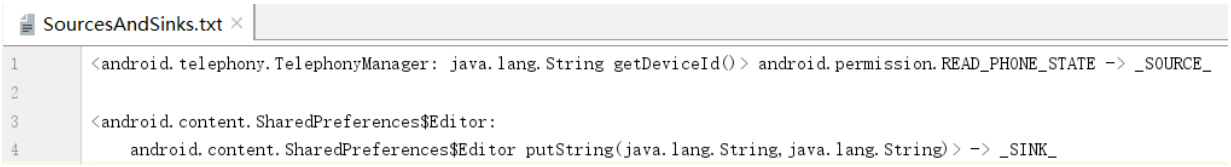

图 3-14 Demo_02 的 SourceAndSink.txt 文件

表 3-2 Demo_02 的测试结果

	时间消耗	内存消耗
修改前	0.027s	44MB
修改后	0.015s	17MB

3. 编写 Demo 进行跨 callbacks 分析检测

在重新设置的 SourceAndSink.txt 文件中，使用 FlowDroid 静态分析工具进行分析，对比修改 SourceAndSink.txt 前后的运行时间和内存。修改后的 SourceAndSink.txt 文件如图 3-15 所示，测试结果如表 3-3 所示。

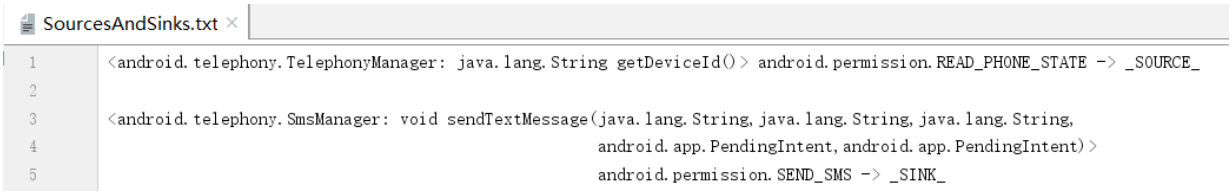

图 3-15 Demo_03 的 SourceAndSink.txt 文件

表 3-3 Demo_03 的测试结果

	时间消耗	内存消耗
修改前	0.037s	52MB
修改后	0.019s	23MB

3.2 实验结果分析

针对以上测试案例，可以得出以下结论：

（1）FlowDroid 静态分析工具可以对 Android 应用程序和 Java 程序进行静态分析，检测出是否存在信息泄漏，保护用户的安全。经上述实验证明，FlowDroid 对设置在跨构件、跨 callbacks 的 source 和 sink，均可完成检测；而在两个 Activity 中，跨函数设置包含泄漏信息的 source 和 sink，则无法被 FlowDroid 成功检测出，即存在漏报现象。

（2）FlowDroid 在进行静态分析时，设有默认的 SourcesAndSinks.txt 文件。实验中，对默认的文件进行修改，设置为仅包含当前测试 APK 文件所用到的单个 source 和单个 sink，并对其进行测试。可以从实验结果得出，修改后的检测能力较修改前相比，无论是时间还是内存消耗上，都有明显的提升。

对以上结论进行简要分析，可以得出以下结论：

（1）FlowDroid 存在漏报现象的主要原因为：在静态污点分析的整体架构中，FlowDroid 不考虑 Service 组件中的回调函数。因此，FlowDroid 的检测结果中，存在与 Service 组件相关的漏报。

（2）FlowDroid 默认的是高精度的分析，但其运行速度慢，且对检测环境的要求高。主要原因有：使用 Soot 将 APK 代码解析为 Jimple 中间字节码，进行分析检测的代码夸大，被用于分析的代码量大；除此之外，Heros 为了加快 IDFS 问题解析的速度，存储了大量辅助数据表，这些表缓存在内存中，使得应用程序的检测变得复杂。因此，导致了 FlowDroid 需要大内存的环境进行分析检测，否则运行会抛出错误。

由此，可以得出 FlowDroid 污点静态分析工具的分析能力和性能仍需提高。

3.3 小结

第 3 章对 FlowDroid 静态分析工具的检测能力进行了测试实验，分别针对定点、跨构件、跨画面的情况，从生成各文件和测试能力方面进行了测试，并做出总结。

4 系统总体设计

4.1 系统流程设计

自动检测系统的开发基于 FlowDroid 静态分析工具，目的是让用户可以更加简便地使用 FlowDroid 工具，并且可以更直观地获取 FlowDroid 的结果。在开发工作开始前，首先，有必要分析和定义系统的基本模型，并根据所需的功能划分模块。然后，根据日常编码习惯对功能模块进行分析和分解。在功能模块开发完成后，进行编写代码，对模块进行集成。最后在系统调试和测试完成后对系统进行使用。App 个人信息安全合规性自动检测系统设计流程图如图 4-1 所示。

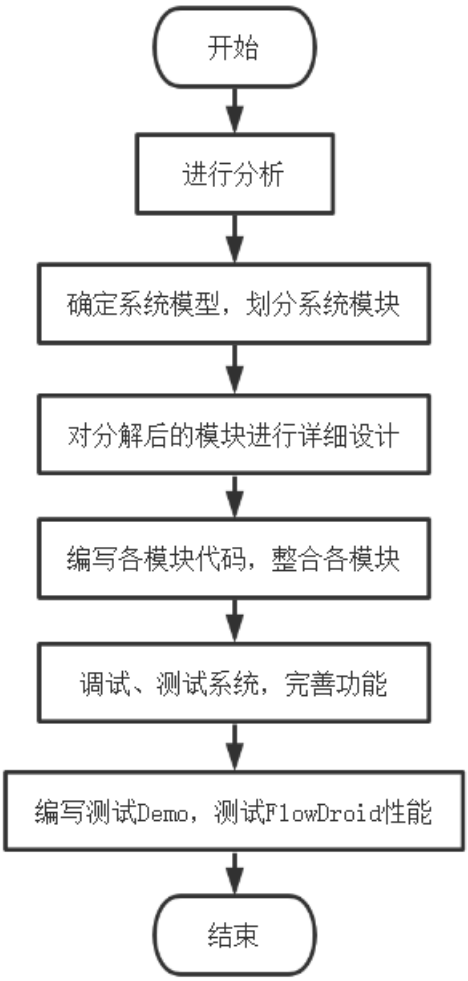

图 4-1 设计流程图

4.2 系统业务流程

打开界面，首先需要在首页进行本地 APK 文件的选择，可以同时进行一个或多个 APK 文件的分析。选择完成后，对所选 APK 文件进行分析，并输出日志文件、漏洞文件、程序调用图文件、控制流图文件，将文件保存至本地。搭建 flask server，查看生成的 gexf 文件，获取相关泄漏信息。系统的业务模块图如图 4-2 所示。

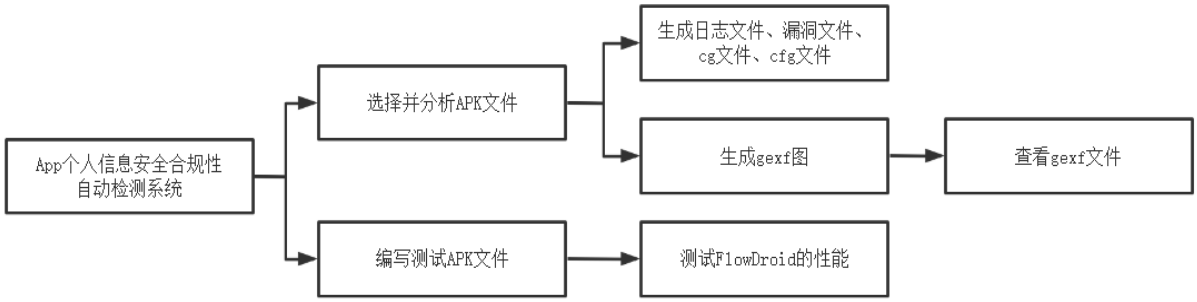

图 4-2 系统业务流程图

4.3 系统架构设计

4.3.1 分析 APK 文件

本系统考虑到直接使用 FlowDroid 需要进行部分代码编写，其过程比较复杂。为了简化用户的操作，将 FlowDroid 相关的 jar 包作为依赖导入，调用相关方法，实现可以通过修改存放 APK 路径的字符串，对该 APK 文件进行分析。该功能的具体信息如表 4-1 所示。

表 4-1 分析 APK 文件

目的：	方便用户使用 FlowDroid
触发者：	用户
前提条件：	完成程序相关的环境配置
结束条件：	分析完成
非功能性需求：	提供本地的 APK 文件，可以直接进行分析测试
假设、问题：	系统（FlowDroid）正常运行

4.3.2 选择界面设计

用户不再需要关心代码是如何实现的，可以直接通过界面进行选择 APK 文件、分析

APK 文件。在界面上直接部分输出漏洞信息，并将日志文件、漏洞文件、程序调用图文件、控制流图文件保存至本地，便于查看相关信息。具体信息如表 4-2 所示。

表 4-2 选择界面的设计

目的：	通过界面选择的方式，对 APK 文件进行分析，并获取相关的信息
触发者：	用户
前提条件：	完成程序的安装，修改 config.json 文件中于程序运行相关的路径
结束条件：	关闭界面
步骤：	选择本地的 APK 文件，通过鼠标点击的方式开始分析，完成后打印相应的输出结果
假设、问题：	系统（FlowDroid）正常运行

4.3.3 查看 gexf 图

设计一个 flask server，用于查看 gexf 图。该功能可以让用户直接在界面上看到信息泄漏的完整路径，便于后续对 APK 文件进行修改，或者避免个人信息泄漏。具体信息如表 4-3 所示。

表 4-3 查看 gexf 图

目的：	查看具体的泄漏路径
触发者：	用户
前提条件：	完成 APK 文件的选择和分析，生成 gexf 图
结束条件：	关闭界面
非功能性需求：	可以直接看到 source -> sink 路径中调用的方法属于的类
假设、问题：	gexf 文件已经生成

4.4 小结

在第 4 章进行了系统的设计，首先在第 1 小节，进行系统的流程设计。在第 2 小节，进行业务流程的分析，对生成的 gexf 图可以选择进行查看。在第 3 小节，进行系统架构设计，其中包括分析 APK 文件、选择界面设计、查看 gexf 图。

5 系统实现

5.1 系统功能结构图

本系统的结构设计思想是将程序一共分为两大部分：

第一部分是分析 APK 文件。用户可以在后端通过修改存放 APK 文件路径的字符串，运行程序，对 APK 文件进行分析；也可以通过在界面上点击的方式，完成 APK 文件的选择，随后后台通过调用命令行的方式，启动 FlowDroid 工具，对用户所选择的 APK 文件进行分析，检查是否存在信息泄漏。

第二部分是存储并查看泄漏信息。如果在分析 APK 文件完成后，检测出该 APK 文件存在泄漏，则在界面上打印泄漏信息。并生成泄漏信息相关的文件，包括：日志文件、漏洞文件、程序调用图文件、控制流图文件等，将这些文件存储在本地，方便进行查看。对于 gexf 图，采用搭建一个 flask 的 server 进行查看，可以直接从途中得到泄漏的路径。

根据上述的分析，最终可以得出 App 个人信息安全合规性自动检测系统的系统功能结构图。如图 5-1 所示。

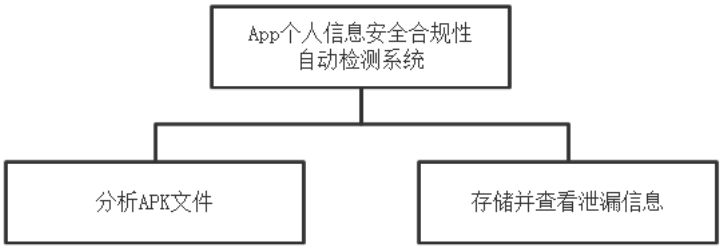

图 5-1 功能模块图

分析 APK 文件主要包括：选择 APK 文件、调用 FlowDroid、分析 APK 文件。如图 5-2 所示。

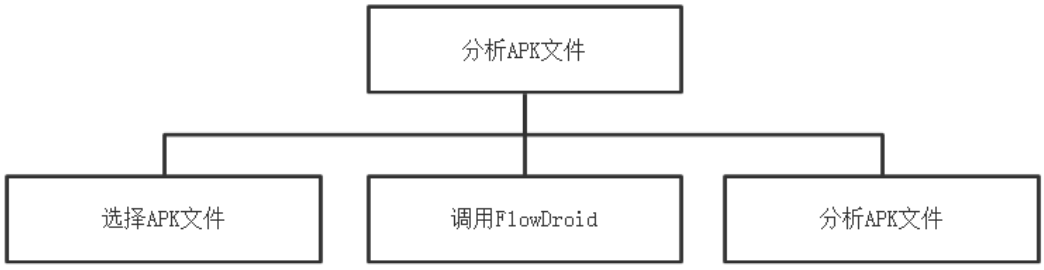

图 5-2 分析 APK 文件

存储并查看泄漏信息主要包括：存储泄漏信息文件、查看泄漏信息文件、查看 gexf 图。如图 5-3 所示。

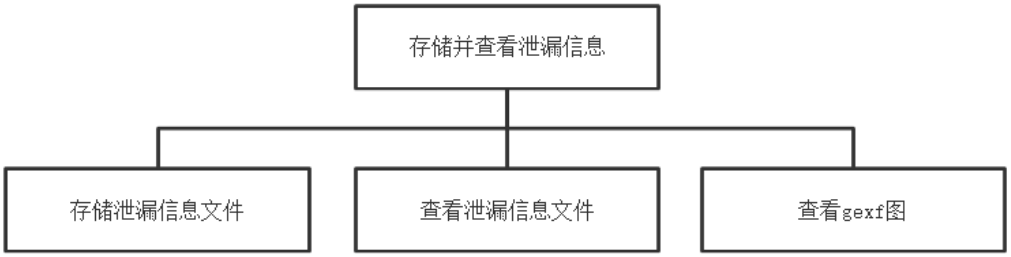

图 5-3 存储并查看泄漏信息

分析 APK 文件使用 Java 语言的 Idea 开发环境进行代码编写。界面实现部分使用 Python 语言的 PyCharm 开发环境进行编写，通过将 Java 语言开发的程序打包，将 jar 包作为依赖导入的方式，实现通过 Python 程序调用 Java 代码。系统调用框架如图 5-4 所示。

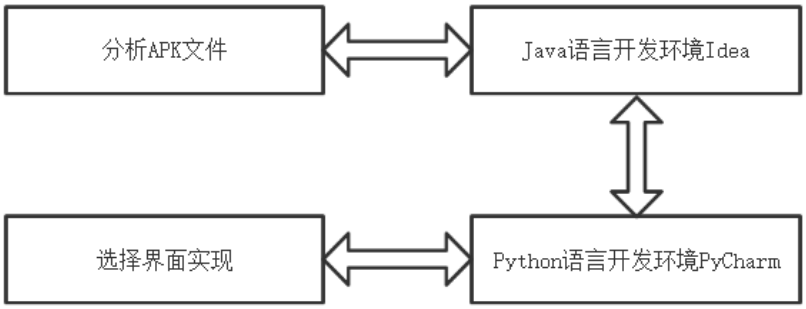

图 5-4 系统调用框架

5.2 APK 文件分析实现

该部分将 FlowDroid 相关的 jar 包作为依赖导入，调用相关方法，实现可以通过修改存放 APK 路径的字符串，即可对该 APK 文件进行分析。关键代码实现内容如下：

（1）导入 FlowDroid 的 jar 包依赖，如图 5-5 所示。

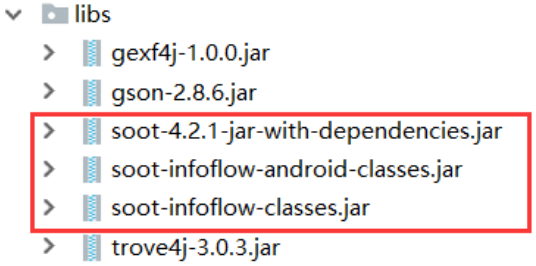

图 5-5 导入 FlowDroid 的 jar 包

（2）将 APK 的路径存储在字符串中，并通过系统定义的 String 类的方法获取 APK 文件和 APK 名。如图 5-6 所示。

```
String apkPath = System.getProperty("apk");
String apkFile = apkPath.split( regex: "/")[apkPath.split( regex: "/").length - 1];
String apkName = apkFile.substring(0, apkFile.length() - 4);
```

图 5-6 存放 APK 路径

（3）通过调用 SetupApplication 类的 runInflow()方法，对所选择的 APK 文件进行分析，判断是否存在泄漏。如图 5-7 所示。

```
public static InfoflowResults analyzeAPKFile(String sourcesAndSinksPath, String easyTaintWrapperSourcePath,
                                             String AndroidSdkPath, String apkPath)
        throws IOException, XmlPullParserException {
    SetupApplication setupApplication = new SetupApplication(AndroidSdkPath, apkPath);

    // 找到污染包装文件
    File taintWrapperFile = new File(easyTaintWrapperSourcePath);

    setupApplication.setTaintWrapper(new EasyTaintWrapper(taintWrapperFile));
    setupApplication.getConfig().setEnableArraySizeTainting(true);

    return setupApplication.runInfoflow(sourcesAndSinksPath);    // 分析
}
```

图 5-7 分析 APK 文件

（4）如果该 APK 文件存在信息泄漏，则需要将泄漏信息打印在界面上。如图 5-8 和图 5-9 所示。

5.3 界面实现

该部分将 5.2 中使用 Java 代码编程实现的功能打包，以 jar 包的形式作为依赖导入。使用 Python 语言进行界面元素的绘制，并调用 Java 程序，实现在界面上通过鼠标点击事件，完成对 APK 文件的分析和泄漏信息的打印。关键代码实现内容如下：

（1）导入已经准备好的 Java 程序的 jar 包依赖，如图 5-10 所示。并且在 Python 程序中调用 jar 包（如图 5-11 所示），通过命令行的方式运行 FlowDroid，如图 5-12 所示。

```java
public static InfoflowResults runOtherAPKs(String sourcesAndSinksPath, String easyTaintWrapperSourcePath, String AndroidSdkPath, String apkPath) {
    try {
        InfoflowResults res = analyzeAPKFile(sourcesAndSinksPath, easyTaintWrapperSourcePath, AndroidSdkPath, apkPath);

        if (res != null) {
            System.out.printf("Found %d leaks\n", res.size());
        }

        System.out.println("Exception as follows");
        System.out.println(res.getExceptions());
        System.out.println("Result as follows");
        System.out.println(res.getResultSet());
        System.out.println("Other Information as follows");
        System.out.printf("Termination State: %d\n", res.getTerminationState());
        System.out.printf("Callgraph Construction: %d seconds\n", res.getPerformanceData().getCallgraphConstructionSeconds());
        System.out.printf("MaxMemory Consumption: %d MB\n", res.getPerformanceData().getMaxMemoryConsumption());
        System.out.printf("Path Reconstruction: %d seconds\n", res.getPerformanceData().getPathReconstructionSeconds());
        System.out.printf("Sink Count: %d\n", res.getPerformanceData().getSinkCount());
        System.out.printf("Source Count: %d\n", res.getPerformanceData().getSourceCount());
        System.out.printf("Taint Propagation: %d seconds\n", res.getPerformanceData().getTaintPropagationSeconds());
        System.out.printf("Total Runtime: %d seconds\n", res.getPerformanceData().getTotalRuntimeSeconds());

        return res;
    }catch (Exception e) {
        System.out.println(e);
        return new InfoflowResults();
    }
}
```

图 5-8 打印泄漏信息

图 5-9 存储泄漏信息

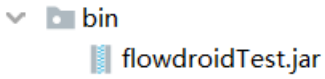

图 5-10 导入 Java 的 jar 包

```
jar_dir_path = os.path.join(WORK_DIR, "bin")
jar_files_path = os.path.join(jar_dir_path, 'flowdroidTest.jar')
```

图 5-11 调用 jar 包

```
def run(command):
    cmd = subprocess.Popen(command,
                           stdin=subprocess.PIPE,
                           stderr=subprocess.PIPE,
                           stdout=subprocess.PIPE,
                           shell=True)

    stdout, stderr = cmd.communicate()
    cmd.communicate()
    return cmd.returncode, stdout, stderr
```

图 5-12 运行 FlowDroid

（2）选择 APK 文件，并确认分析。如图 5-13 所示。

```
def check(self, event):
    path = QFileDialog.getOpenFileName(self, "Open File Dialog", "/", "Android files(*.apk);")
    if path[0]:
        button = QMessageBox.question(self, "提示",
                                      self.tr("是否开始对选中的apk进行信息泄露分析？"),
                                      QMessageBox.Ok | QMessageBox.Cancel,
                                      QMessageBox.Ok)
        if button == QMessageBox.Ok:
            self.newTab(path[0])
        elif button == QMessageBox.Cancel:
            pass
        else:
            return
    else:
        return
```

图 5-13 选择并确认 APK 文件

（3）确认分析 APK 文件，打印泄漏信息到界面，并写入将泄漏信息文件存储在本地。如图 5-14、图 5-15 所示。

```
def update_out_put(self, string):
    """Write console output to text widget."""
    cursor = self.output.textCursor()
    cursor.movePosition(QTextCursor.End)
    while not self.buffers.isNone():
        cursor.insertText(self.buffers.pop())
    cursor.insertText("\n")
    cursor.insertText(".target/log/{}.log 日志文件已生成\n".format(self.apkName))
    cursor.insertText(".target/out/{}.out 漏洞文件已生成\n".format(self.apkName))
    cursor.insertText(".target/cg/{}.cg 程序调用图cg文件已生成\n".format(self.apkName))
    cursor.insertText(".target/cfg/{}.cfg 控制流图cfg文件已生成\n".format(self.apkName))

    cursor.insertText(string)

    self.output.setTextCursor(cursor)
    self.output.ensureCursorVisible()
    self.scrollToEnd()
```

图 5-14 打印泄漏信息

```
OutPutPath = CONFIG["OutputPath"]
cg_output_path = os.path.join(OutPutPath, "cg")
cfg_output_path = os.path.join(OutPutPath, "cfg")
log_output_path = os.path.join(OutPutPath, "log")
out_output_path = os.path.join(OutPutPath, "out")
gexf_output_path = os.path.join(OutPutPath, "gexf")

if exsits_or_return_dir(OutPutPath):
    exsits_or_create_dir(cg_output_path)
    exsits_or_create_dir(cfg_output_path)
    exsits_or_create_dir(log_output_path)
    exsits_or_create_dir(out_output_path)
    exsits_or_create_dir(gexf_output_path)
else:
    print("输出目录不存在！请重新更改设置")
    return
```

图 5-15 存储文件

（4）分析完成，存储泄漏信息文件的目录结构如图 5-16 所示。

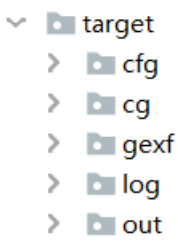

图 5-16 文件存储目录结构

5.4 gexf 图查看实现

该部分搭建了一个 flask 的服务器，用于实现 gexf 图的查看。关键代码实现内容如下：

（1）搭建 flask 的服务器。如图 5-17 所示.

```
from utils import handleGEXF

app = Flask(__name__)
```

图 5-17 搭建 flask 服务器

（2）通过调用 handleGEXF()函数，实现 gexf 图的查看。如图 5-18 所示。

```
@app.route('/')
def index():
    work_path = os.getcwd()
    static_path = os.path.join(work_path, "static")
    gexf_path = os.path.join(static_path, "Merge1.gexf")
    gen_file_name = handleGEXF(gexf_path)
    return render_template("view.html", file="static/{}".format(gen_file_name))
```

图 5-18 查看 gexf 图

5.5 系统运行结果

在执行测试用例前，必须保证程序可以正常运行，即修改环境配置文件 config.json 文件。如图 5-19 所示。

```
{
    "AndroidSdkPath": "C:/Users/Lenovo/AppData/Local/Android/Sdk/platforms/android-18/android.jar",
    "SourcesAndSinksPath": "E:/Java_workspace/end/flowdroidTest/SourcesAndSinks.txt",
    "EasyTaintWrapperSourcePath": "E:/Java_workspace/end/flowdroidTest/EasyTaintWrapperSource.txt",
    "OutputPath": "E:/Java_workspace/end/flowdroidGUI/target/"
}
```

图 5-19 环境配置文件

（1）将程序打包为可执行文件（.exe 文件）格式，双击.exe 文件即可运行程序。如图 5-20 所示。

图 5-20 打开界面

（2）点击“选择 APK”按钮，进行本地 APK 文件的选择。在文件框中，当只选定以“.apk”结尾的文件，点击“打开”按钮，或者双击选定的文件，结果如图 5-21 所示。

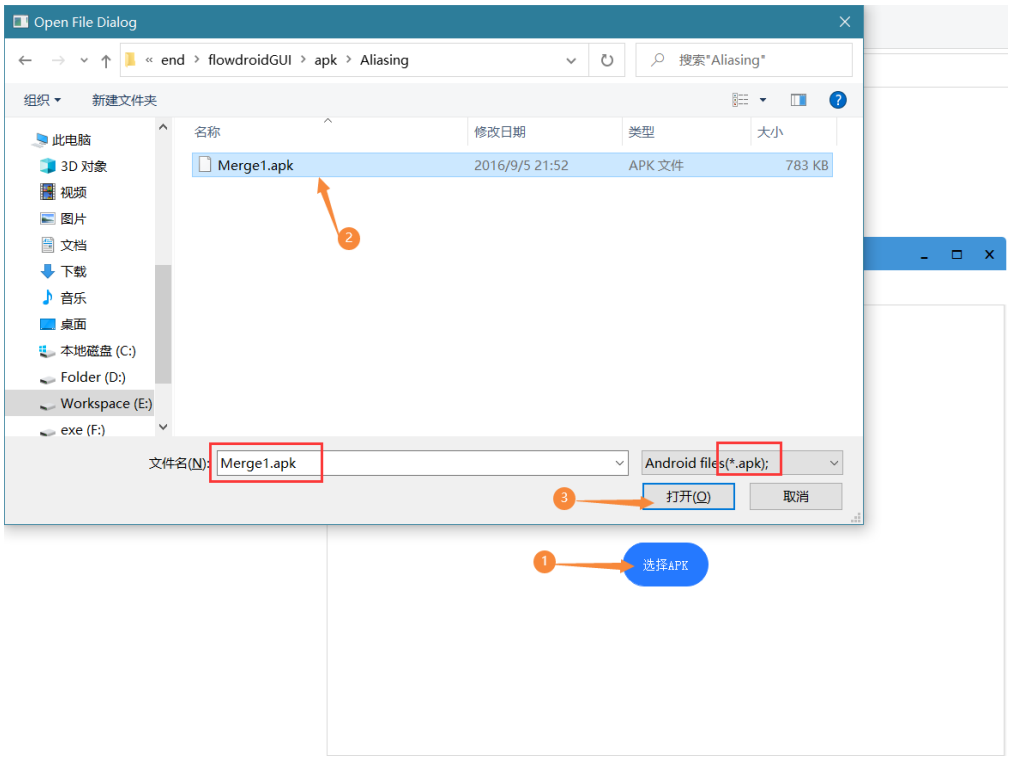

图 5-21 选择 APK 文件

（3）点击提示框中的“OK”按钮，结果如图 5-22 所示。

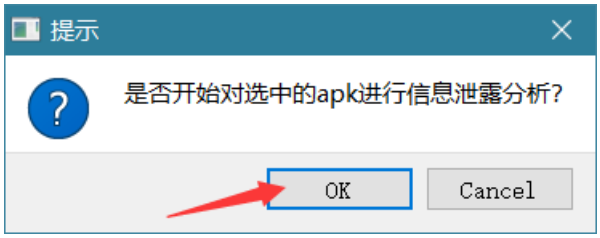

图 5-22 提示框

（4）在分析页面中，如果不点击“开始分析”按钮，返回首页，仍可继续进行 APK 文件的选择。如图 5-23 所示。

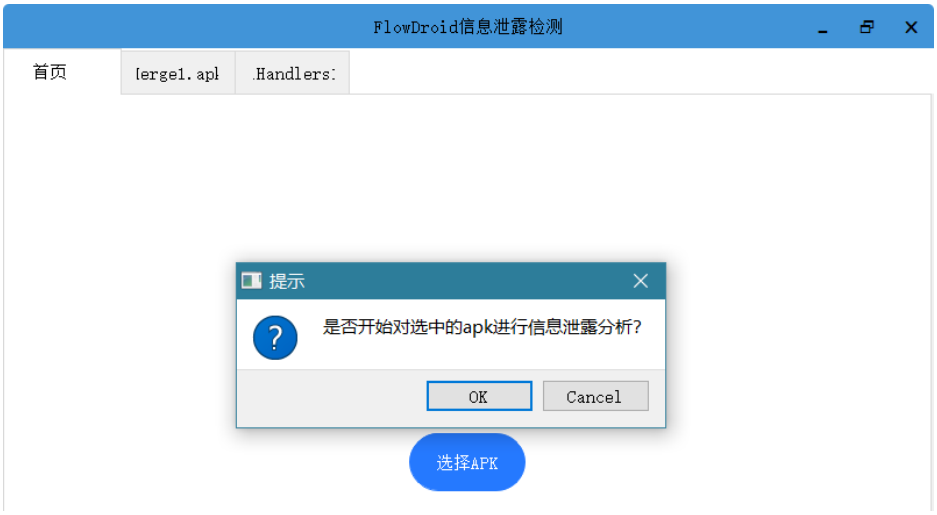

图 5-23 重新选择 APK 文件

（5）点击“开始分析”按钮，程序开始对所选择的 APK 文件进行分析。分析结束后，打印泄漏信息并存储。如图 5-24、图 5-25 所示。

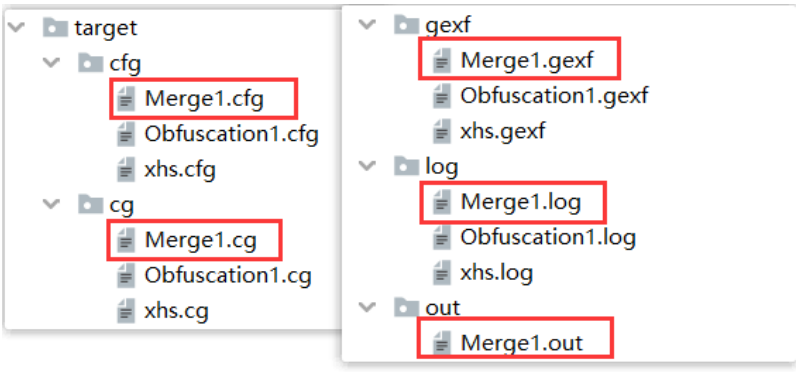

图 5-24 生成的本地文件

图 5-25 运行结果的界面信息输出

（6）选择没有泄漏信息、格式正确的 gexf 文件，界面会呈现空白，即不存在泄漏路径的控制流图。选择有泄漏信息、格式正确的 gexf 文件，界面会显示泄漏路径的控制流图，并显示每一个节点所代表的函数名、函数路径，以及调用过程。如图 5-26 所示。

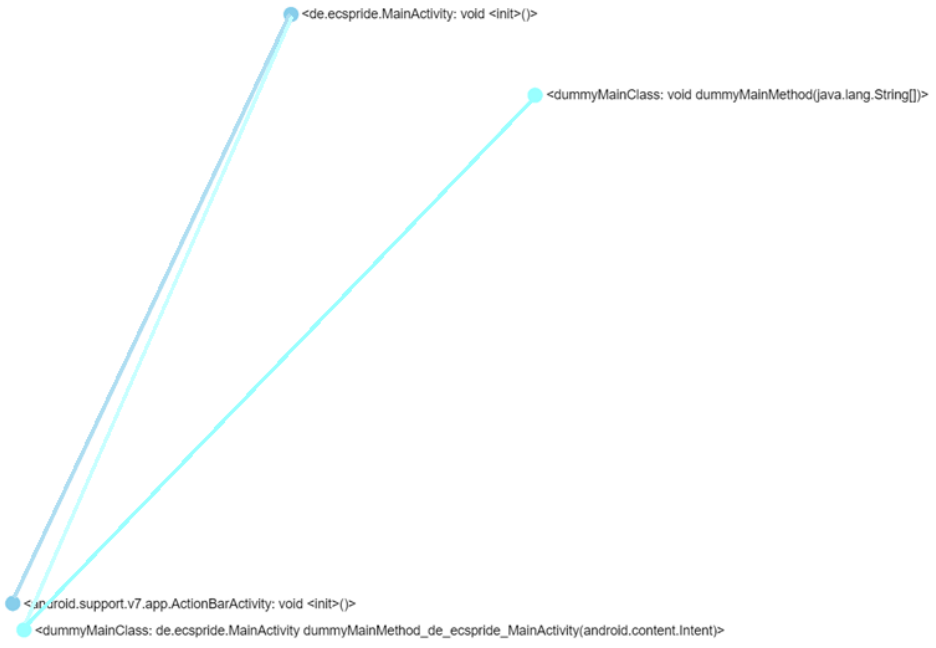

图 5-26 查看对应的 gexf 文件

5.6 小结

第 5 章对系统的实现进行了具体的描述。在第 1 小节对主要的功能逐一进行介绍，并画出了系统的功能结构图和系统调用框架。根据第 1 小节系统功能结构图进行代码的编程实现，在这个过程中，包括 APK 文件分析实现、界面实现、gexf 图查看实现等，陈列关键代码并进行实现的说明。

6 系统测试

6.1 测试意义和目标

6.1.1 测试的意义

在设计系统和编写代码的过程中，由于经验不丰富，很容易遇到一些没有考虑到的功能或实现，而这种现象在反复推敲的过程中，总能或多或少的得已处理解决。同时在整个软件开发的生命周期中，由于能力有限，错误的出现也是无法避免的，从多个角度进行思考、尝试，是减少错误的一种有效的方法。这两种解决方法都属于对软件进行测试的方法。

6.1.2 测试的目标

测试是通过不断地运行程序，从而发现程序中不完备的地方的一个过程。测试的根本目的不在于检验程序的正确性，而在于发现程序中的不足和错误。软件测试不是一遍又一遍的展示程序运行的结果。因此，我们需要正确地对待软件测试，需要全面考虑问题，抱着发现问题的态度进行此项工作，发现程序中存在的更多问题。

6.2 测试阶段

软件测试包括：单元测试、集成测试、系统测试和验收测试几个阶段。针对本项目，每个测试阶段的具体工作如下：

1.单元测试

单元测试旨在检查代码每一个最小单位是否符合预期结果。对于本系统而言，单元测试应分别在分析 APK 文件、界面显示、查看 gexf 图中进行，分析每部分的功能，并设计测试用例。

（1）分析 APK 文件。该部分要求通过修改存储 APK 路径的字符串 apkPath，实现对任意的 APK 文件进行分析，对于不同的 apkPath 应该进行不同的响应。编写测试用例时，需要考虑输入不同 apkPath 的情况。通过以上分析，可以设计出的测试用例如表 6-1 所示。

表 6-1 分析 APK 文件功能测试用例表

测试用例	输入数据	预期结果
O1	apkPath = null;	打印“Not Found APK.”
O2	选择没有泄漏的 APK 文件	打印“Found 0 leaks.”
O3	选择有一条泄漏路径的 APK 文件	打印“Found 1 leaks.”

（2）界面显示。该部分要求用户先点击“选择 APK”按钮，选择本地的待检测 APK 文件；随后点击“OK”，确认对选中的 APK 进行分析；最后点击“开始分析”，等待分析结束后，在界面上输出结果。可以进行多次 APK 文件的选择和分析，直至点击“关闭”按钮。编写测试用例时，需要考虑在单击界面上得不同按钮时，必须触发不同的事件。通过分析，可以设计出的测试用例如表 6-2 所示。

表 6-2 界面显示功能测试用例表

测试用例	输入数据	预期结果
P1	点击“选择 APK”按钮	弹出文件框
P2	文件框中点击“取消”按钮	界面没有变化
P3	文件框中不选择文件点击“打开”按钮	文件框没有变化
P4	文件框中选择不是.apk 的文件	无法进行选择
P5	文件框中双击选择.apk 文件	弹出提示框
P6	文件框中选择.apk 文件，并点击“打开”	弹出提示框
P7	提示框中点击“Cancel”按钮	界面没有变化
P8	提示框中点击“OK”按钮	文件选择成功， 跳转至分析界面
P9	不点击“开始分析”，重新返回“首页”	仍可继续选择
P10	点击“开始分析”按钮	分析完成后输出结果， 并生成本地文件
P11	继续返回“首页”	仍可继续选择
P12	点击“最大化”	全屏显示窗口
P13	点击“最小化”	隐藏窗口
P14	点击“关闭”	退出系统

（3）查看 gexf 图。该部分要求通过写入正确的 gexf 文件，运行程序，即可在网页上显示泄漏路径的控制流图。编写测试用例时，需要考虑写入 gexf 文件的正确性，以及是否是对应 APK 包的信息泄漏路径的控制流图。通过分析，可以设计出的测试用例如表 6-3 所示。

表 6-3 分析 APK 文件功能测试用例表

测试用例	输入数据	预期结果
Q1	gexf_path = null;	打印“Not Found GEXF.”
Q2	选择不是以.gexf 结尾的文件	打印“文件格式错误，不为 gexf”
Q3	选择没有泄漏的 gexf 文件	gexf 图为空
Q4	选择 A 的 gexf 文件	A 的 gexf 图
Q5	选择有一条泄漏路径的 gexf 文件	gexf 图有一条路径
Q6	选择有五条泄漏路径的 gexf 文件	gexf 图有五条路径

经过测试，以上所有的测试用例都可以通过测试。

2.集成测试

集成测试是对系统各部分进行组合的接口进行测试，检测当每个模块组合后，系统是否还能符合预期的结果。由于本系统进行单元测试的前置条件是，已经实现的模块必须通过单元测试，也就是说在单元测试阶段已经完成了集成测试的工作。

3.系统测试

系统测试的目的是验证当用户根据未来的需要进行调度时，系统始终可以工作。针对本系统的功能，设计出如下测试用例，如表 6-4 所示。

表 6-4 系统测试的测试用例表

测试用例	输入数据	预期结果
T1	在进行分析过程中，选择 APK 文件	可以选择
T2	在进行分析过程中，分析其他 APK 文件	可以分析
T3	在进行分析过程中，最小化窗口	可以执行此操作
T4	在进行分析过程中，最大化窗口	可以执行此操作
T5	在进行分析过程中，关闭窗口	可以执行此操作

4.验收测试

在开发完成后，系统将支持其他系统的持续测试，包括性能测试、恢复测试、强度测试和安全性测试等。

至此，整个程序所有的功能运行完毕。

6.3 测试评价

经过上述测试的设计和执行，对系统的主要功能进行了测试，对遇到的问题即使做了修改。系统逐步完善，整个整体基本上已经达到稳定运行状态。但当用户选择超过三个 APK 文件同时进行分析时，系统运行较缓慢，此时，最大化窗口、返回首页界面的点击事件会出现延迟显示的情况。

因此，可以得出结论：该系统满足了用户的基本需求，但在改善与用户的交互方面仍有许多工作要做。到此，系统的测试已经完成。根据以上测试工作，对该系统进行评价：

（1）功能评价：

本系统所有功能皆已完成，界面简单，便于用户操作。

（2）数据评价：

本系统将检测结果以较为易读的方式打印在界面上，并以文件的形式存储到本地，方便用户阅读和检查。

（3）性能评价：

本系统具有良好的内容性能和交互性。

（4）稳定性评价：

本系统运行时偶有发生操作延迟的现象，这与 FlowDroid 工具分析的复杂性有关，且发生频率较低，对正常的使用影响不大。

6.4 小结

第 6 章对系统测试进行了具体的描述。在第 1 小节，对测试的意义和目标做了明确的解释。在第 2 小节，对系统测试所用到的测试方法、测试用例做了详细的介绍，并以截图的形式展示出了测试结果。在第 3 小节，根据前两小节的理解和结果，对整个系统进行了评价。

7 总结与展望

7.1 总结

本系统基于 FlowDroid 静态分析工具，完成了在界面上通过鼠标点击事件，通过命令行的方式，在后台调用 FlowDroid 工具，对 APK 文件进行分析。通过封装 FlowDroid，在一定程度上简化了用户使用 FlowDroid 工具的操作，但还有存在一定的局限性。首先，FlowDroid 本身的效率问题没有得到更完美的解决。其次，本系统在检测过程中，如果同时选择多个 APK 文件，就会出现点击事件延迟显示的现象。

在系统设计和实现完成后，针对 FlowDroid 的检测能力做了相关实验，可以得出以下结论：FlowDroid 可以进行 Android 应用程序隐私信息泄漏的检测，对 source 和 sink 设置在不同构件、不同 callbacks 的情况均可进行检测，但对于将 source 和 sink 设置在不同方法中的情况不能完全适用；在简化 FlowDroid 的 SourceAndSink.txt 文件后，FlowDroid 检测的时间和内存消耗都有了明显的提升。由此可得，FlowDroid 静态分析工具在分析能力和性能方法仍需提高。

Android 手机变得越来越流行，许多手机制造商已经修改了 Android 本地系统，并将其应用到自己公司的手机品牌上，于是带来了许多问题。因此对手机隐私泄漏的研究具有很强的实用性。在未来的研究中，将在这个工具的基础上进一步完善，并结合动态检测技术，开发一个操作简单、高效率的自动检测系统，用于检测 Android 的恶意应用程序。

7.2 展望

随着毕业设计的即将结束，在不断努力下，当前的系统也初具规模。虽然各项功能并不是很完善，还有一定的提升空间。比如简化 FlowDroid 输出的路径，让用户可以在不用对原 APK 文件进行反编译的前提下，读懂信息泄漏 source 到 sink 的路径。对于 FlowDroid 静态分析工具，可以对其性能和分析能力的提高做出进一步的探讨，已优化该工具。相信可以做得更好，给自己一个更好的交代。

在毕业设计的整个过程中，我学习到了丰富的新知识，比如 FlowDroid 静态分析工具的使用、gexf 图及其查看的方法。同时也提高了自己的专业综合素质，更加熟悉了软件开发的全过程，提高了代码编写的能力。我意识到自己的知识储备还远远不够，需要继续学习和提高自己的专业技能和文档编写能力。过程虽然曲折，但是现在再回头看看

这段时间，收获满满，不管是人生阅历还是开发经验都得到了提升，一切都是值得的。

人生是不断学习的过程。通过这次毕业设计的完成，增加了我对编程的热爱，知识永远是学不完的，但只有不断地学习才能得到提升。

致　谢

时间像一条粼粼的河流，缄默地改变着岁月。四年一闪而过，对于我而言，大学的四年是人生中承前启后的时间，承载着过往十多载的寒窗苦读，开启着以后辽远亦或是平淡的漫长岁月。

在此，我想要郑重地感谢。

感谢软件工程专业，激发了对神秘互联网世界的热爱，让我对软件开发有了极大的兴致，也让我有勇气追逐心之所向。

感谢我的毕业设计指导老师刘晓建老师，从毕设选题到设计实现，从模糊的了解到清晰的明确，老师的指导给了我无尽的信心和动力。感谢老师为我提供的设计思路和想法，让我从中学习到了很多知识。

感谢四年中所有给予我教诲的老师们，坐在教室听你们讲课的时光，或许是四年中最斑斓的日子，也是现在最想回到的日子。这四年中所学到的每一门课程，既帮助我向内审视自身，找到了对软件工程的喜爱与兴趣所在；又帮助我向外打量世界，关心新技术，关注互联网的发展，将所学映照于每一次的程序开发中，实现自我与社会的连接。

感谢挚友罗雪儿，于我困顿时安抚我，于我孤独时陪伴我，亦会将快乐分享于我，或是将烦恼倾诉于我。双向奔赴的友谊，早已变成如亲情一样的存在。

感谢我的父母，你们永远是我最真切的依靠，也是我永恒的动力。

岁月忽如寄，四年一瞬，在西安科技大学度过的时时日日、月月年年都值得珍重以待。四年里所收获的笑与泪，都将融汇为浪漫又温柔的光河。不再触及我，却能在以后人生的每一段路途中遥远的照耀我。

在此，谨以寥寥几句，将大学四年的时光珍藏于此。在此感谢所有对我有过帮助的人。

参考文献

[1] I. D. Corporation. Worldwide quarterly mobile phone tracker 3q12, Nov. 2012. https://www.idc.com/tracker/showproductinfo.jsp?containerId=IDC_P8397

[2] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. 2011. A survey of mobile malware in the wild. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM '11). Association for Computing Machinery, New York, NY, USA, 3–14.

[3] Michael C. Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. 2012. Unsafe exposure analysis of mobile in-App advertisements. In Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks (WISEC '12). Association for Computing Machinery, New York, NY, USA, 101–112.

[4] 刘效伯.Android 系统隐私泄漏检测与保护研究[D].东南大学,2017.

[5] 张小贝.基于 Android 平台的恶意代码检测技术研究[D].北京邮电大学,2017.

[6] 孙贝. 基于上下文的 Android 隐私泄漏检测技术研究[D].南京理工大学,2018.

[7] 曾述可.基于静态分析的 Android 操作系统隐私保护机制评估方法研究[D].中国科学技术大学,2014 年

[8] 王奕钧.基于 Android 权限机制的隐私保护方法研究[D].哈尔滨工程大学,2016.69.

[9] 欧阳金彬. 基于数据流分析的 Android 应用隐私泄漏检测研究[D].南京大学,2017.

[10] 袁洋. Android 隐私数据保护的安全容器研究与实现[D].北京邮电大学,2019.

[11] 张力智. 基于权限的 FlowDroid 静态检测方法研究[D].西安电子科技大学,2019.

[12] 胡英杰,张琳琳,赵楷,方文波,于媛尔.基于静态污点分析的 Android 隐私泄漏检测方法研究[J].信息安全学报,2020,5(05):144-151.

[13] 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.

[14] Shancang Li, Junhua Chen, Theodoros Spyridopoulos, Panagiotis Andriotis, Robert Ludwiniak, Gordon Russell. Real-Time Monitoring if Privacy Abuses and Intrusion Detection in Android System. 2015 International Conference on Human Aspects of Information Security, Privacy, and Trust.

[15] Youngho Kim,Tae Oh,Jeongnyeo Kim,Kamal Deep Singh. Analyzing User Awareness of Privacy Data Leak in Mobile Applications[J]. Mobile Information Systems,2015.

[16] Lo N W, Yeh K H, Fan C Y. Leakage Detection and Risk Assessment on Privacy for Android

Applications: LRPdroid[J]. IEEE Systems Journal, 2016, 10(4):1-9.
[17] Yifei Zhang,Yue Li,Tian Tan,Jingling Xue. Ripple : Reflection analysis for Android apps in incomplete information environments[J]. Software: Practice and Experience,2018,48(8).
[18] Gulshan Shrivastava,Prabhat Kumar. SensDroid: Analysis for Malicious Activity Risk of Android Application[J]. Multimedia Tools and Applications,2019,78(24).
[19] Kang Hongzhaoning, Liu Gang, Wu Zhengping, Tian Yumin, Zhang Lizhi. A Modified FlowDroid Based on Chi-Square Test of Permissions[J]. Entropy,2021,23(2).
[20] Reshetova E, Bonazzi F, Asokan N. SELint: an SEAndroid policy analysis tool[J]. arXiv preprint arXiv:1608.02339, 2016.
[21] Vallée-Rai R, Gagnon E, Hendren L, et al. Optimizing Java bytecode using the Soot framework: Is it feasible?[C]//International conference on compiler construction. Springer, Berlin, Heidelberg, 2000: 18-34.
[22] Arzt S. Static data flow analysis for android Applications[J]. 2017.
[23] 颜慧颖,潘璠,安庆杰,叶益林.第 4 讲 Android APP 隐私泄漏检测技术研究进展[J].军事通信技术,2017,38(01):97-104.
[24] 张力智. 基于权限的 FlowDroid 静态检测方法研究[D]. 西安电子科技大学, 2019.
[25] 蒋理. 基于 Android 平台的隐私泄漏检测与保护技术的研究[D]. 南京航空航天大学, 2018.
[26] Qiu L, Wang Y, Rubin J. Analyzing the analyzers: Flowdroid/iccta, amandroid, and droidsafe[C]//Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2018: 176-186.
[27] 马绍菊, 万良, 杨婷, 等. 一种基于 FlowDroid 的 Android 隐私保护方法[J]. 计算机应用与软件, 2017, 34(5): 317-321.
[28] Developers A. What is android[J]. Dosegljivo: http://www. academia. edu/download/30551848/andoid--tech. pdf, 2011.
[29] 马绍菊,万良,杨婷,马林进.一种基于 FlowDroid 的 Android 隐私保护方法[J].计算机应用与软件,2017,34(05):317-321.
[30] 张悦. 移动 APP 个人信息安全合规测试[J]. 网络安全技术与应用, 2020.
[31] 章辉. Android 应用隐私泄漏检测技术研究[D].西安电子科技大学,2018.