分 类 号　　密 级

学校代码 10704　　学 号 1408010318

西安科技大学

学 士 学 位 论 文

题目：基于 Soot 框架的 Android App 权限提取和溢权漏洞检测

作者：伊扬贵

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：2018 年 6 月

摘　要

为了保证用户数据的安全，Android 系统设计了一种用于管理对敏感数据访问的权限机制。然而，由于对权限的声明和使用门槛较低，没有指定一些申请权限所必要的协议或手续。开发者可以很容易就能声明某些权限，但是对于用户而言，通常是不知道自己所授予的那些权限会给恶意应用以可乘之机、可能会有哪些信息泄露危险的。所以对应用权限使用漏洞的检测就显得尤为重要。

本次课题根据目前安卓平台的使用情况和 Android 系统的权限机制发展现状，设计出一种基于权限的溢权漏洞检测方法。在实现上扩展已有的静态分析工具 Flowdroid，实现了一个用于检测 Android 应用溢权漏洞的工具。并通过一些实际的 apk 文件进行试验验证，总结出工具的有效性和使用价值。本次论文主要研究内容为以下 3 个部分：

（1）分析的 Android App 静态分析方法，介绍 Soot 和 Flowdroid 项目的核心数据结构和使用方法，并通过拓展自 Flowdroid 工具实现了一个自动化的 API 和权限提取工具；

（2）在实现 API 和权限提取的基础上，通过对应用中清单文件中声明的权限和代码中实际使用的权限的分析，得到应用是否存在溢权漏洞的结论，帮助用户识别可能引起信息泄露的应用。

（3）在实现溢权漏洞检测工具的基础上，通过对一些典型良性和恶意应用的检测判断发，统计出检测工具的有效性和实用性。

关键字：Android 系统；权限管理；静态分析；权限溢出

ABSTRACT

In order to guarantee the security of user data, Android system designs a permission mechanism for managing access to sensitive data. Furthermore, due to the low threshold for the declaration and use of rights, there are no agreements or procedures necessary to apply for permission. Developers can easily apply permissions for function and service requires. But for users, it is usually not known which permissions they grant will give malicious applications an opportunity to exploit, and what information may be dangerous. Therefore, the detection of application permission exploits is particularly important. Based on the current circumstance of the An-droid platform and the development status of the Android system, this research put forward a permission overflow detection method. It extends the existing static analysis tool Flowdroid and implements an Android application. The tool that overflows the vulnerability is verified through some actual apk files, and the usefulness and value of the tool are summarized. This article primarily includes the following modules:

(1) Research the existing Android App static analysis method, study and study the core data structure and usage of Soot and Flowdroid project, and implement an automated API and permission extraction tool by extending Flowdroid tool;

(2) On the basis of the above API and permission extraction, through the analysis of the permissions declared in the Manifets.xml in the application and the actual rights used in the code, the conclusion is drawn as to whether there is an overflow vulnerability in the application, helping the user to identify the information that may be caused. Leaked application.

(3) Based on the realization of the tool for detecting breach of vulnerability, based on the detection and judgment of some typical benign and malicious applications, the effectiveness and practicality of the detection tools are counted.

Key Words: Android System; Permission Manager; Static Analyze; Permission Overflow

目录

1 绪　论

1.1 论文研究背景及意义

近年来移动互联网的迅速发展，智能手机已经成为用户访问网络的重要设备，在人们的日常生活中扮演着重要的角色。相比于传统的个人计算机，智能手机虽然只是一种通信工具，但是随着近年来硬件水平的提高，智能手机已经能够完成个人计算机的大部分功能。与此同时，和个人计算机一样，智能手机也暴露出一系列的安全性问题。安卓中可以添加各种具有 Android 智能手机扩展功能的应用程序，这可能会传播恶意应用程序。另一方面，在智能手机上存放着用户的各种用户账号用户及密码等个人隐私数据，这些数据如果泄露，就有可能给智能手机使用者带来了精神上或者经济上的损失。

Felt 等人[1]调查了用户在安装 Android 程序时，对权限的理解程度。他们通过设计若干测试用例，来判断安装 Android 应用程序时呈现给用户的权限申请列表对用户是否有效。研究结果说明安装时的权限授予提醒并没有很好地帮助用户做出可靠的决定。这说明了虽然有权限的授予机制，但是用户对用户来说并不完全是有利的，因为用户也不知道哪些权限的授予会造成不安全事件的发生。因此，一些研究人员研究能否对现有的权限机制进行改进。使得在安装应用时，应用的权限信息能够得到一个详细的说明，让用户明白权限的作用，并帮助用户分析应用程序是否可能存在恶意行为。

安卓恶意行为检测方法主要分为两大类：静态检测和动态检测[2]。静态检测方法是指在不需要运行应用的情况下，对 apk 文件进行分析的一种方法[3]。通常的静态分析步骤是：通过反编译工具对 apk 文件进行反编译，将 apk 文件反编译为更接近源码的表示形式，然后实施代码层次的分析与检测。动态检测方法是指在程序执行的情况下，进行动态地提取应用中的行为或者操作，分析出应用程序的恶意性行为[4]。

1.2 研究现状

面对 Android 市场比重的不断扩大和持续恶化的 Android 应用安全现状，Android 应用恶意行为的检测技术也在不断的加强，不同的学者、科学院都提出了自己的应对措施或检测工具。从大的方向上看，Android 应用安全领域最主流的研究方法可划分为静态分析和动态分析两大方向[3]。

在基于权限的 Android 安全研究方面, 朱佳伟、喻梁文[3]对当前的权限管理机制做了深入的研究工作, 阐述了权限管理机制的实现流程。并对其进行了详细地探索,还对权限

管理机制提出了一些完善改进的措施。杨晶、金伟信[4]提出了一套从安卓系统扩展而来的权限配置方法，并通过权限管理工具来实现了这个配置方案。当用户进行安装的时候,对权限配置进行限制，能够禁止高度敏感的权限被使用,从而防止隐私数据的泄漏。只有当用户给程序授予比较危险的权限时才可继续,这种措施在一定程度上对安卓的权限授予机制进行了优化[4]。

1.2.1 国外研究现状

国外在安卓平台的恶意应用检测方面,Di Cerbo Francesco 和 Girardello Andrea 提取安卓应用程序的权限信息，将这些信息作为检测的数据基础来检测恶意软件。并对谷歌应用商店上的应用程序进行了实际地测试，最后取得了客观的效果[4]。Blasing Tho-mas 和 Batyuk_ Leoni 通过对 Android 系统的沙箱机制的研究，提出了一种针对安卓应用程序的沙箱（AASandbox）[5]。这种区别于系统沙箱的分析工具，能够有效的识别恶意应用，很大程度上检测到危险应用。在安卓系统的应用权限申请管理方面, OngtangM 和 Mc-Laughlin S 研究出了一个名为 SAINT 的权限控制模型[6],这种模型允许用户在使用应用的过程中动态修改应用的权限，这样就能够实现不同场景下的权限管理。Zemin Liu 和 Choon-Sung 对 Android 安全机制进行了扩展了工作，发表了一个基于密级的权限控制模型[7] 。扩展后的权限机制可以实现多用户场景下的权限管理,只有授权等级比该应用设定的等级更高的使用者才能够打开该并读写应用中的数据。

WeiWang 和 XingWang 等人提出了三个不同层次的权限授予风险检测研究[8]。先使用 3 种不同的方式根据潜在风险对个人特权进行分类;其次，使用顺序抢先和 PCA 策略分析总特权的子集。在他们的研究工作中，构建了一个决策树分析树来分析不同恶意应用的功能。该项研究可以实现在扫描 Android 应用程序安装包，同时就能准确地判断是否为恶意应用。

1.2.2 国内研究现状

Wen Weiping，Mei Rui 等[9]在检测恶意 Android 应用的检测技术时提出了一种检测移动代码的方法，该方法与移动终端和服务器协同工作，移动终端使用授权分析策略，可以实现小规模的检测功能。服务器可以检测从手机发送的样本，可以实现对软件进行行为分析[10]。 Zhang Yiting，Zhang Yang，Zhang Tao[11]等人提出了一种利用朴素贝叶斯算法实现的安卓应用恶意行为识别方法。该方法全局考虑了安卓系统的运行环境和特征，并收集用户的行为特征进行分析判断。如果软件应用了太多的权限，那么有可能存在敏感权限的组合，如果权限的使用作为分类存在并且 Android 安全框架被扩展，则实现行为是有害的。

1.3 本论文研究内容

本论文主要研究基于 Soot 框架下的 Android App 权限提取，熟悉 Soot 及 Flowdroid 框架的源码结构和使用方法。然后扩展 Flowdroid 工具实现一个可视化工具，生成安卓应用程序的函数调用图（CallGraph）,并遍历函数调用图获取整个应用的 API 调用集合。再借助 PScout 项目的 API-PERMISSION 映射表，查询到程序中实际使用到了哪些权限。最后将提取到的 API 和 PERMISSION 集合作为溢权分析的输入，综合已有的工具提出一个基于权限的溢权漏洞检测方法，并借助 Java 语言实现了一个可视化的工具辅助用户识别不安全应用。

1.4 论文组织结构

本论文总共分为六个大模块，全文的结构划分如下：

第一章：介绍本次论文题目的研究内容，介绍论文题目的研究背景和国内外研究现状。

第二章：从 Android 官网整理最新版本的平台架构，从整体上熟悉安卓平台。整理 android 平台各架构层采取的安全措施和实现原理。

第三章： 从源码角度介绍了 Soot 和 Flowdroid 工具的关键数据结构，通过对 Soot 和 Flowdroid 工具相关源码和文档的阅读。编译并运行 Soot 和 Flowdroid 项目源码，并运行项目中的案例，分析 Soot 和 Flowdroid 工具的运行结果和实际用法。

第四章：介绍基于权限的 Android 应用静态分析方法，详细介绍安卓应用的安装包文件结构，因为安装包是所有静态分析工作的依据。最后，结合数据图的邻接表结构，将函数调用信息存入邻接表中，借助于栈实现图的路径搜索算法，找到敏感 API 的调用路径。

第五章：通过前面章节的理论技术和研究成果，实现拓展自 Flowdroid 工具的权限提取工具。从网上下载实际的 apk 文件进行提取和结果分析，并对比已有的同类工具提取结果，分析当前工具的性能指标和有效性。

第六章：综合课题的研究现状和本论文实验的结果，总结本论文方法的不足与欠缺之处，并展望后续可加强的工作及方向。

2 安卓系统架构与安全机制

2.1 安卓平台架构

Android 系统由系统应用层、JavaAPI 框架层、原生 C/C++库与运行环境层、硬件抽象层和 Linux 内核层，各层都有自己的分工，层层隔离，互不影响。具体如图 2-1 所示。

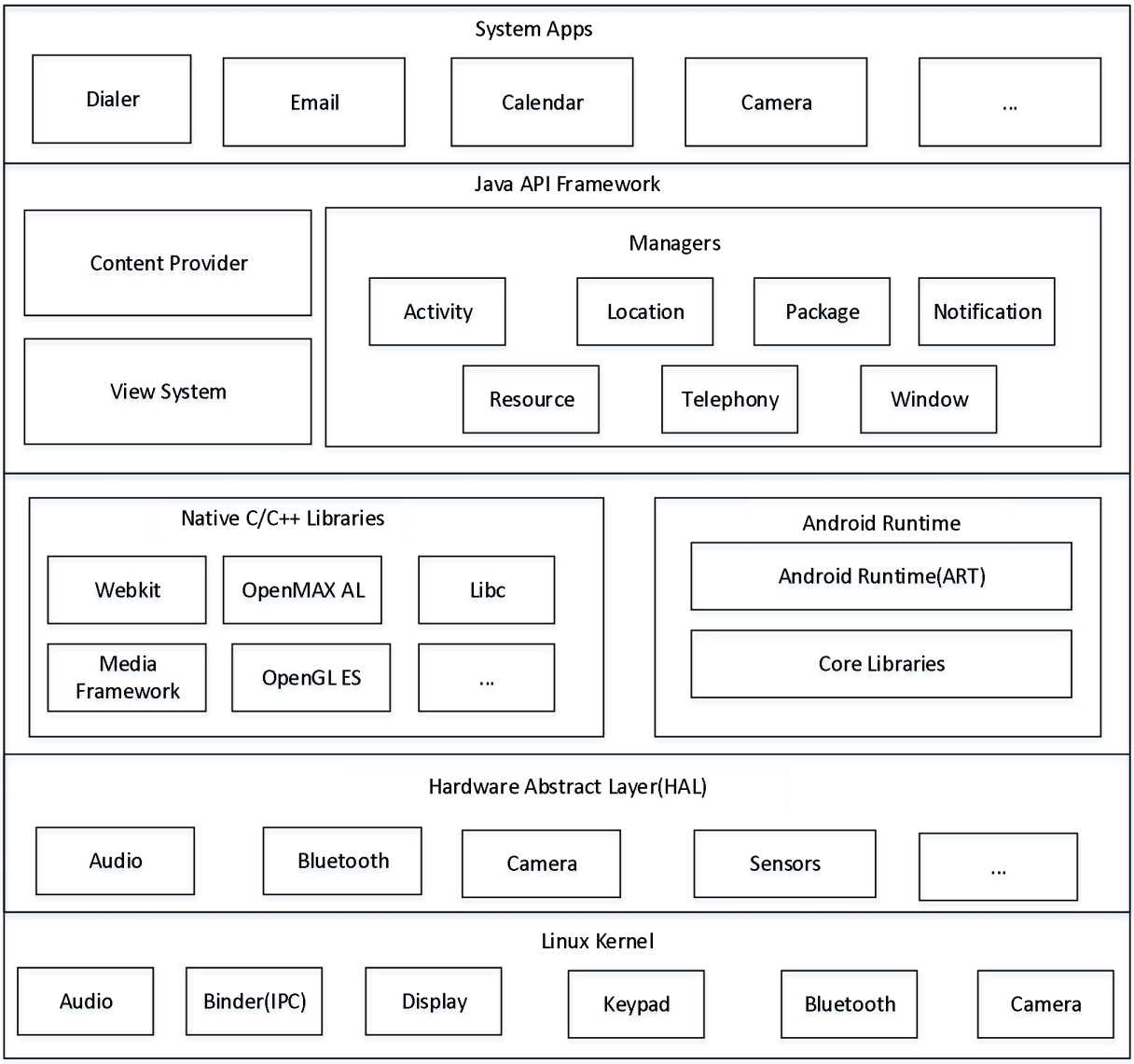


图 2-1 Android 平台架构

2.1.1 系统应用层

系统应用层提供了一些常用的组件程序，比如：基本的应用程序组件，如邮件，联

系人信息，发短信和网络浏览器等等。系统应用程序可以充当用户应用程序，并提供开发人员从其应用程序访问的主要功能。

2.1.2 Java API框架层

Android 应用开发者可以通过编写 Java 代码实现具有特定功能的 App,关键就是借助 Java API 框架层的功能集合实现的。经由 Java API 框架层供给的 API 就可以够调用安卓系统提供的全部功能。这个 API 形成创建 Android 应用所需的组件集合，使用它们可以很大程度地减少核心模块系统服务或组件的重复调用。API 框架层通常包括如表 2-1 中的组件和服务。

表 2-1 API 框架层包括的组件和服务

可拓展的视图结构	可用于构建安卓程序的 UI,比如：列表、网格、文本框、按钮或者嵌入的网络浏览器
资源管理中心	用于访问非代码资源，例如本地化的字符串、图形和布局文件
通知管理器	用于管理应用的生命周期，提供 Activity 管理栈
Activity Manager	用于管理应用的生命周期，提供 Activity 管理栈
Content Provider	用于实现不同应用间的数据共享

2.1.3 原生C/C++库与运行环境层

（1）原生 C/C++库：安卓系统部分组件的功能并不是完全在 Java 层面实现的，比如一些音视频处理组件或者图像处理组件，它们需要用系统提供的 C/C++库来实现。

（2）运行环境：主要是包含一个执行 DEX 可执行文件的运行时环境，。

2.1.4 硬件抽象层（HAL）

支持一些标准化的接口，提供应用层的硬件支撑。

2.1.5 Linux内核层

安卓平台的内核是基于 Linux 操作系统内核的。比如，Android 运行时凭借 Linux 内核实现功能，线程和底层由 Linux 内核控制管理，借助 Linux 内核允许可以将 Linux 系统

的安全措施引入到安卓系统中。比如：基于用户的权限模式、进程隔离和进程间安全通信机制等等。

2.2 安卓平台安全机制

2.2.1 沙箱隔离机制

沙箱（Sandbox）模型，是一种能够保障安卓系统安全的关键性技术，目前已经在浏览器的设计上取得成功应用。

如图 2-2 为 Android 沙箱模型示意图。

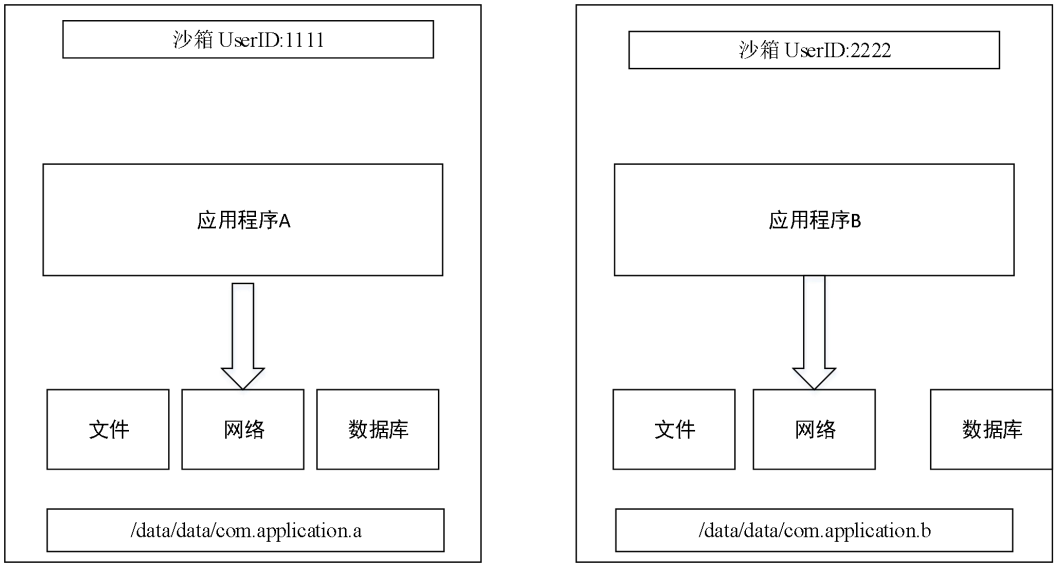


图 2-2 Android 沙箱模型示意图

2.2.2 权限管理机制

Android系统通过一种能保护设备资源以及用户数据的权限机制，来管理应用程序对硬件设备、敏感数据或资源是否能正常访问。当某应用程序要在某些资源或相关数据上执行操作时，首先必须获得对应的权限，而这些权限的申请和授予，就需要开发者和用户共同决定。

倘若需要申请某一个权限，可以在 Manifest.xml 清单文件中进行申请。Android 系统也将不同版本的系统进行了权限分层级处理。开发者在开发应用程序时可申请由安卓系统预先定义好的权限集合，然后在程序安装时询问用户是否授予。同时安卓系统允许用户自定义权限，控制当前组件被其他组件访问时其他组件需要申请的权限，其他应用程序若想访问该应用程序或其中某个组件，需要申请组件预先自定义好的权限。所以，

Android 系统主要安全准则是应用只有在获得权限后，才有权利执行一些会对系统或其他应用程序造成影响的敏感操作。

Android 应用程序使用权限时，在 AndroidManifest.xml 文件中用<use-permission>标签申请权限。在程序安装时，将需要用户授予的权限信息展示给用户，用户可选择同意安装或拒绝安装。在声明权限是，在应用代码中可用<permission>标签来定义自己的权限，从而限制对本应用某些组件或程序的访问。在安装应用程序时，自定义的权限被加入到系统中作为记录。另外，开发者在自定权限时可标示权限的危险级别，其危险级别分为以下三个等级：

（1）正常权限(Normal)：保护应用程序对可能但不会伤害用户的 API 调用的访问。

（2）危险权限(Dangerous)：控制对潜在有害的 API 调用的访问。例如与支付或收集私人信息相关的 API 调用，是危险权限需要声明和谨慎判断。

（3）签名/系统权限(SignatureOrSystem)**:**这些权限一般情况下很难获得：签名权限仅授予那些使用设备制造商的证书签名的应用程序，并且 SigenatureOrSystem 权限仅授予在特殊系统文件夹中签名或安装的应用程序。

2.2.3 数字签名机制

Android 系统凭借数字证书来搭建应用程序开发者和应用程序之间的可靠关系，该数字证书通常不需要权威的数字认证机构进行申请认证，仅用于允许认证的应用程序包。在开发过程中签名应用程序时，使用调试私钥自动进行，这类应用称为调试模式的应用，这些应用不能发布到谷歌应用市场上。当开发者需要在谷歌应用商店上发布自己的应用时，需要用安卓的签名机制使用其私钥对应用进行签署。这样可以实现在用户安装应用时验证应用的开源。

2.3 Android 平台应用程序文件结构

2.3.1 apk文件格式分析

用 Java 编写的 Android 应用程序，利用 Android SDK 编译代码并将应用中需要的数据和资源文件压缩为一个 APK 文件。APK 文件是一个特殊格式的 zip 压缩包，解压 apk 文件后可以看到以下文件结构。如表 2-2 所示：

表 2-2 apk 文件结构

res	存放资源文件的目录，包括应用的图片、布局、字符串等资源文件
META-INF	存放应用签名信息，用来保证 apk 的完整性和系统的安全
lib	存放 ndk 编译出来的 so 库，即动态链接库
resources.arsc	存放资源文件编译后的二进制信息
classes.dex	存放经过 java->.class->.dex 编译过程的字节码文件，是安装系统的可执行文件
AndroidManifest.xml	应用的配置文件，包含组件、应用权限、sdk 版本等信息

2.3.2 dex文件格式研究

Android 应用程序使用 Java 语言编写功能代码，编译过程相比于普通的 Java 程序，多了从.class 文件经过编译和优化操作到.dex 文件的过程。对 java 编写的源代码用 javac 编译后，然后由 SDK 工具下的 dx 工具，将 Java 字节码文件打包转换为.dex 文件[12]。如表 4-2 所示，dex 文件的结构由 9 个部分组成。Dex 可执行文件的编译生成过程如图 2-3 所示。

图 2-3 dex 文件的生成过程

关于 dex 文件每个组成部分的详细说明如表 2-3 所示。

表 2-3 dex 文件组成部分说明

名称	说明
dex_header	dex 文件头结构
string_ids	字符串表，该表存放的是 dex 文件中所引用的所有的字符串
type_ids	类型标识表，存放的是 dex 文件引用的所有类型的标识符，在该表中按照 string_id 索引进行排序
proto_ids	函数原型标识符表，该表中按照返回值以及参数的类型进行排序
field_ids	字段标识符表，存放的是应用程序所引用的字段的标识符，在该表中按照字段类型、字段名称以及字段所属类型进行排序
method_ids	函数名表，该表中按照声明的函数类型、函数名以及函数原型进行排序
class_defs	类定义表，在该表中，父类以及实现的接口必须出现在被引用类之前
data	数据区
link_data	静态链接文件使用的数据

3 Soot 和 Flowdroid 框架

3.1 Soot 概述

Soot[12]是麦吉尔大学的 Sable 研究小组所研究开发的一款 Java 字节码优化框架，现在已经被全球各地的研究学者用来分析 Java 和安卓应用。Soot 可以处理：Java 字节码和源码、Android 字节码、jimple 中间表示和 jasmin 格式。处理后输出格式可以为 Java 字节码、Android 字节码、jimple 中间表示和 Jasmin。比较强大的是，Soot 的输入和输出格式是没有明确限制的，可以根据实际的需要自己设置。比如可以从.DEX 字节码到.JAVA 代码、.JAVA 源码到.jimple 中间代码输出。Soot 提供的主要类型分析主要包括: 指向分析、内部数据流分析、与 Flowdroid 工具协作完成的污点分析和生成调用图等。Soot 的工作原理是：将应用程序转换为一种特定的中间，然后分析这些中间表示形式，在这些中间表示形式上优化、完善等。

3.1.1 Soot工具组织架构

Soot 文件组织结构复杂，有近两千个 java 源文件，stable 项目组按其是否为自动生成，将他们分别安置在 generated 和 src 目录中。soot.Main 是主类。soot 包中定义有为各种 IR 都可以使用的基本类，类 Soot.G 收集 Soot 的全局变量，大部分为类 Singletons 收集的 singleton 对象，可通过相应类中的 Scence.v()方法取得。

3.1.2 关键数据结构

Scene，SootClass，SootMethod，SootField 和 Body 是 Soot 框架的关键数据结构。Scence 类代表整个分析变换工作的场景，其主要的类成员如表 3-1 所示。

表 3-1 Scene 类的主要成员

成员名	含义
Classes	所有类的链式集合类
mainClass	待分析程序的主类
applicationClasses	用于分析的应用类形成的链
libraryClasses	库中类形成的链
nameToclasses	类名到类的映射
sootClassPath	命令行参数或环境变量中指定的 classpath
activeHierarchy	类层次图，Hierarchy 类型
activeCallGraph	方法调用图
activePointsToAnalysis	PAG(Pointer Assignment Graph)

续表 3-1

activeSideEffectAnalysis	sideveSideE 信息
entryPoints	程序入口点

SootClass 表示一个加载到 Soot 中或由 Soot 创建的类。SootMethod 类中方法的表示类。SootField 表示类中的一个成员，Body 表示一个函数的函数体，Body 的类结构如图 3-1 所示。

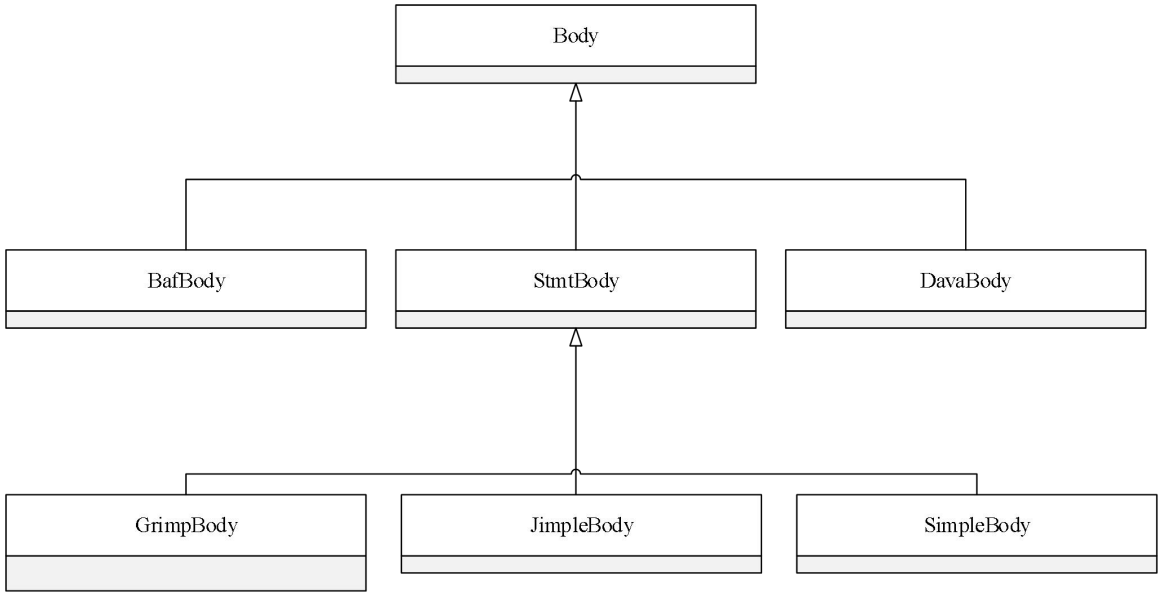


图 3-1 Body 的类继承结构

3.1.3 Jimple中间表示

Jimple[13]是 Soot 中最重要中间形式代码，目前很多的分析工作都是在 Jimple 基础上进行的。它是一种类型化的、三地址、基于语句的中间表示。在 Java 字节码到 Jimple 的转换过程中，为了简化程序操作，移除 Java 规范 jsr 指令，Jimple 引入很多的局部变量。同时，Jimple 还会在编译过程中移除多余的无效的代码。比如：去掉了从未使用过的变量和未使用的内存空间。

Jimple 表示形式中只有 15 中语句，因此只需要处理这 15 种语句就可以了。但是在 Java 字节码中需要处理 200 多种指令，因此，Jimple 很大程度的简化了 Java 字节码。借助这个原理，可以先对 Jimple 中间代码进行优化操作，然后再将优化后的 jimple 中间代码重新转换回 Java 字节码。这样可以减小系统加载和运行字节码的开销，使得代码运行效率更高。Soot 框架为安卓静态分析工作提供了必要的基础架构支撑，而 Flowdroid 框架在 Soot 原有的基础上进行了拓展，使得支持更多的静态分析。比如：Flowdroid 中添加了可以进行精确的数据流分析的框架 Spark[4]。

3.1.4 CallGraph函数调用图

Soot 框架中提供了获取函数调用图的 API,将函数间的调用关系封装到了 CallGraph 数据机构中。调用图中维护了一个存储函数调用的边的集合。这个表示函数调用的集合包含了显式方法调用和隐式方法调用，另外还包括多线程 Thread.run()方法的调用和 finalizers()方法的调用。

这里使用 Flowdrod 框架中一个测试的 apk 文件进行生成 CallGraph 的测试，生成 CallGraph 的关键代码如代码 3-1 所示：

代码 3-1 soot 工具生成 CallGraph

```
SetupApplication application = new SetupApplication(sJarPath, apk);
app.calculateSourcesSinksEntrypoints("sourcesAndSinks.txt");
Scene.v().loadNecessaryClasses();
SootMethod mainMethod = application.getEntryPointCreator().createDummyMain();
Options.v().set_main_class(mainMethod.getSignature());
Scene.v().setEntryPoints(Collections.singletonList(mainMethod));
PackManager.v().runPacks();
CallGraph callGraph = Scene.v().getCallGraph();//获取函数调用图
visit(callGraph, mainMethod);//递归遍历 CallGraph 建立函数调用关系
exportMIG("flowdroidCFG","D:\\GraduationProject\\FlowAndroid\\flowdroidcg");
```

生成的 CallGraph 通过构造函数调用图，再进行可视化，最后写入到 gexf 文件中，如图 3-2 为 Soot 测试 apk 文件 enrick1.apk 的函数调用图。

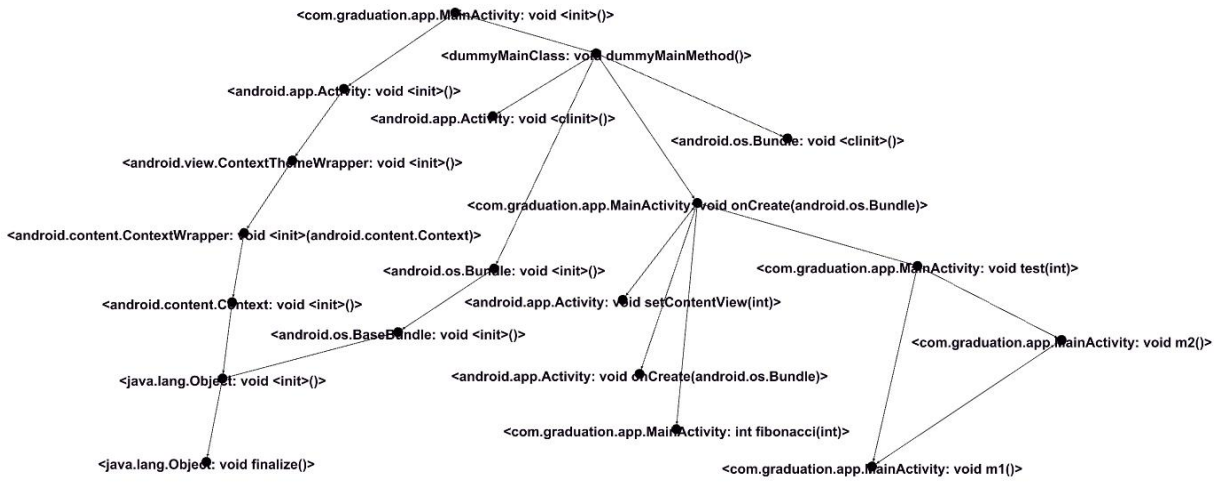


图 3-2 enrich1.apk 的函数调用图局部图

3.2 Flowdroid 工具

3.2.1 Flowdroid概述

Flowdroid 工具基于 Soot 框架开发[14]，由 Paderborn 大学的安全软件工程实验室开发维护。与其他许多针对 Android 的静态分析方法不同，Flowdroid 具有非常高的查全率和精确度的分析。 为了实现这个目标，必须完成两个主要挑战：为了提高精度，我们需要构建一个对上下文，流，场景和对象敏感的分析; 为了增加回忆，我们必须创建一个完整的 Android 应用生命周期模型。

3.2.2 Flowdroid工具组织架构

Flowdroid 工具并不是由一个单独的项目构成，而是由多个组件共同称之为 Flowdroid, 完整的 Flowdroid 项目由 Jasmin、Heros、Soot、Soot-inflow-android 和 Soot-inflow 组件构成，5 个组件之间的依赖结构如图 3-3 所示。

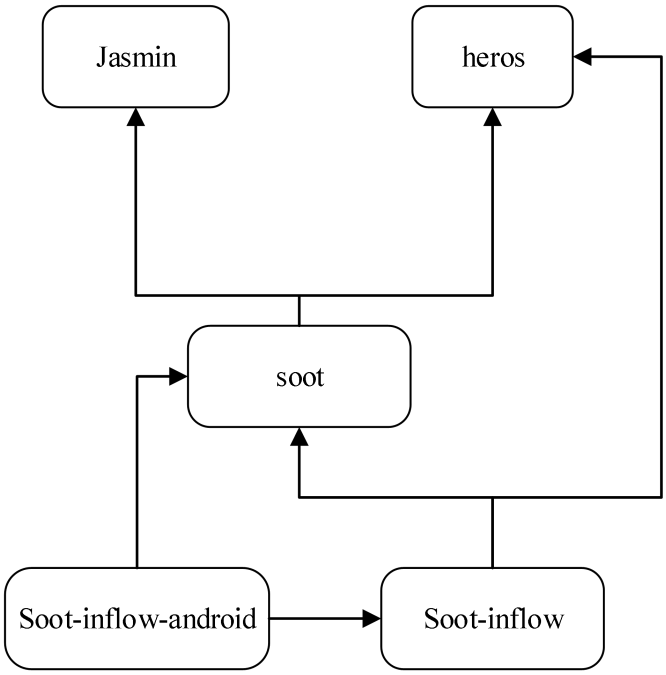


图 3-3 Flowdroid 相关项目依赖关系

3.2.3 分析流程

FlowDroid[15]是对 Android 应用进行静态分析的一个工具，它基于 Soot 和 Heros，它的输入为 Android 应用安装包（apk 文件），输出为 jimple 中间代码。安卓应用程序和一般意义上的代码程序有所不同，它没有一个标准的 main 方法。Android 程序包含了许多

入口方法，也称之为回调方法，它们在系统底层源码中被调用。Android 操作系统将每一个组件都用了一系列称为生命周期的方法定义其不同的状态，由这些生命周期方法序列可以反映出组件运行的状态变化。安卓系统中提供了四大组件，这些组件包括 Activity、Service、Content -Provider 和 BroadcastReceiver，这些组件在使用时都需要在清单文件 AndroidManifest.x-ml 中进行配置。因此，当构建一个调用图时，不能简单地从一般意义上的 main 方法开始。而是用 Android 系统中提供的标准生命周期方法作为入口点，因此，FlowDroid 构建一个通用的 dummyMainMethod()方法来模拟程序的入口点[16]，然后将整个程序构建为一个包含 main 方法的完整程序，由 dummyMainMethod 方法调用各种生命周期的回调方法。FlowDroid 工具的检测流程如图 3-4 所示。

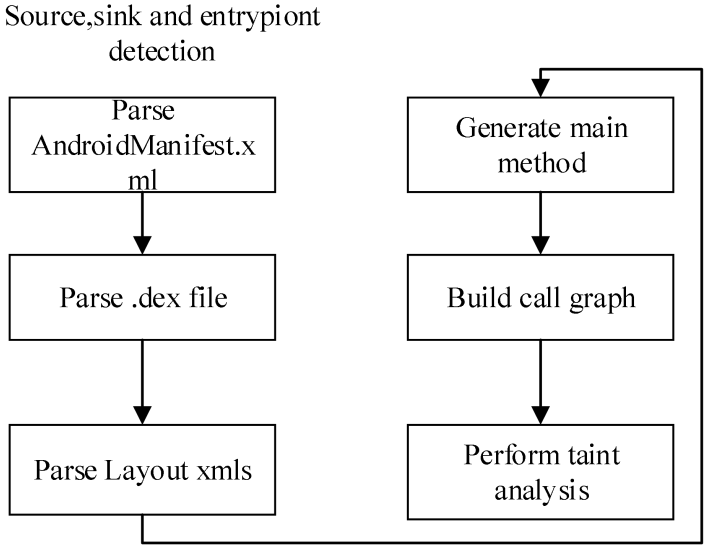


图 3-4　Flowdroid 分析流程

首先，Flowdroid 以 apk 文件作为输入，将其反编译之后，分别分析清单文件、字节码文件和布局文件。通过对这些文件的解析来获得应用的 source、sink 以及入口节点，得到生命周期及回调函数列表。

然后，Flowdroid 根据安卓系统源码，识别各组件的回调方法的回调方式和回调的位置，然后构造一个方法名为 dummyMainMethod()的方法[18]。这个方法可以将程序中的回调关系组织在一起，这样做的效果就是看让本来是回调关系的程序看起来如同直接调用一样，便于阅读和查看。Flowdroid 基于 IFDS[15]框架实现了一套安卓静态的污点分析的体系，它在安卓框架和应用程序功能方面做了很多的建模工作。Flowdroid 工具分析的准确性高[8]，但是项目的代码量比较庞大。在实际进行分析的时候会因为应用的安装包的太大而非常耗时和消耗计算机的处理器和内存资源。另外，由于 APK 的一些正常应用需求，该方法存在一定的误报率。

Flowdroid 使用能准确描述 CallGraph（函数调用图），这有助于我们确保流程和上下文敏感度[17]。其基于 IFDS 的流程功能可确保对场景和对象敏感。由于 Android 的源和汇由 SuSi 提供，因此我们只需查找入口点。 除了必要的源信息之外，它们还从 Android 的清单文件和界面布局文件中提取程序的交互的信息。后者允许我们考虑用 XML 定义的用户交互回调（例如按钮点击），并根据密码字段发现其他来源。 由于用户交互不能静态预测，因此 FlowDroid 会生成一个特殊的主要方法，它考虑所有可能的组合，以确保不会丢失污点。

4 Android 平台静态分析与溢权漏洞检测工具的实现

4.1 静态分析技术

应用程序静态分析[13]（Program Static Analysis）是指在不运行程序代码的情况下，通过词法分析、语法分析、控制流分析或者数据流分析等技术手段对程序代码进行扫描，然后验证代码是否满足规范性、安全性、可靠性、可维护性等指标的一种静态代码分析技术。在 Android 平台上，静态分析主要是指分析其安装包（apk）文件。生成安装包的过程中称之为编译，所以，将程安装包逆向分析的过程称之为反编译[14]。在 Android 台上进行 APK 静态分析的主要目的是为了了解代码的结构、逻辑流程，也可在需要的时候借助修改、插入、删除逻辑，替换和修改资源等策略辅助分析应用程序。

4.2 应用权限提取方法

Android 应用程序的安装时申请的权限是在开发者在清单文件中声明的。对于权限提取工作，如果只从清单文件中提取应用程序的权限，则和实际代码中使用的权限集合会有很大差异，以为目前 android 系统或者开发工具并没有是否有申请多余的权限的检测工具。安卓 Google 官方对 Android6.0 及以上版本中对权限做了分类管理，对某些涉及到用户隐私信息的权限在运行时根据用户的需要进行动态授予。这样做就不会让用户在安装应用之初就被强制同意授予大量的权限。因此，从软件的实际代码中提取是一种可行性比较高的方式。提取应用的权限时，首先需要建立 Android 编程系统中所有 API 与权限的映射关系，也就是找到每个权限对应的 API 集合。这样就可以遍历源码或中间代码扫描出程序中使用的所有 API 集合。最后用提取到的函数调用信息在 API-权限映射表中查询到目标应用使用了哪些权限，在这个过程中，权限提取算法和 API-权限映射表是决定提取结果的两个关键因素。详细的应用程序的权限提取过程如图 4-1 所示。

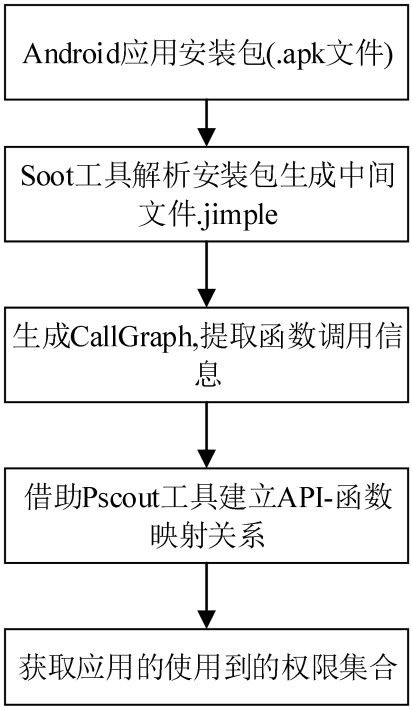


图 4-1 权限提取流程

4.3 权限-API 映射与 PScout 项目

4.3.1 API与权限的映射关系

在 Android 系统中，每一项数据资源或者服务的访问都依赖于应用被授予了相应的权限。与敏感数据的访问有关的 API 称之为敏感 API[15]。因此，在代码中每个敏感 API 的使用需要拥有对应的权限，这样就给利用权限映射关系提取应用程序中实际使用到的权限。但是，以为 Android 系统的版本数量很多，每个版本都有 API 上的差异。所以想要建立整个系统内 API 的权限映射关系是一个工作量很大的研究。

4.3.2 PScout项目

PScou 项目是美国伯克利大学研究的一个著名项目，基于安卓系统的开源性，该项目通过对安卓系统内的源码进行分析，得到 Android API 对应权限的映射关系。并借助 Android 开发文档对映射关系做了补充和完善，让权限映射关具备一定的权威性。如图 4-2 所示为 pscout 项目 API-PERMISSION 关系的建立流程图。图 4-3 是 PScout 项目中的部分权限映射条目举例。

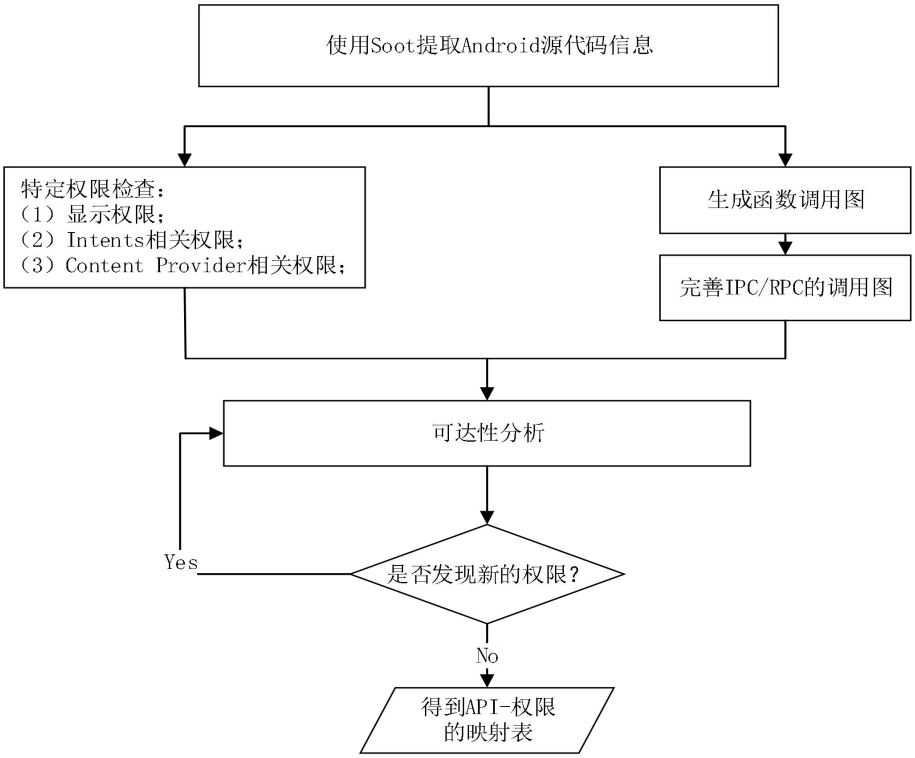


图 4-2 PScout 框架的 API-权限工作流程

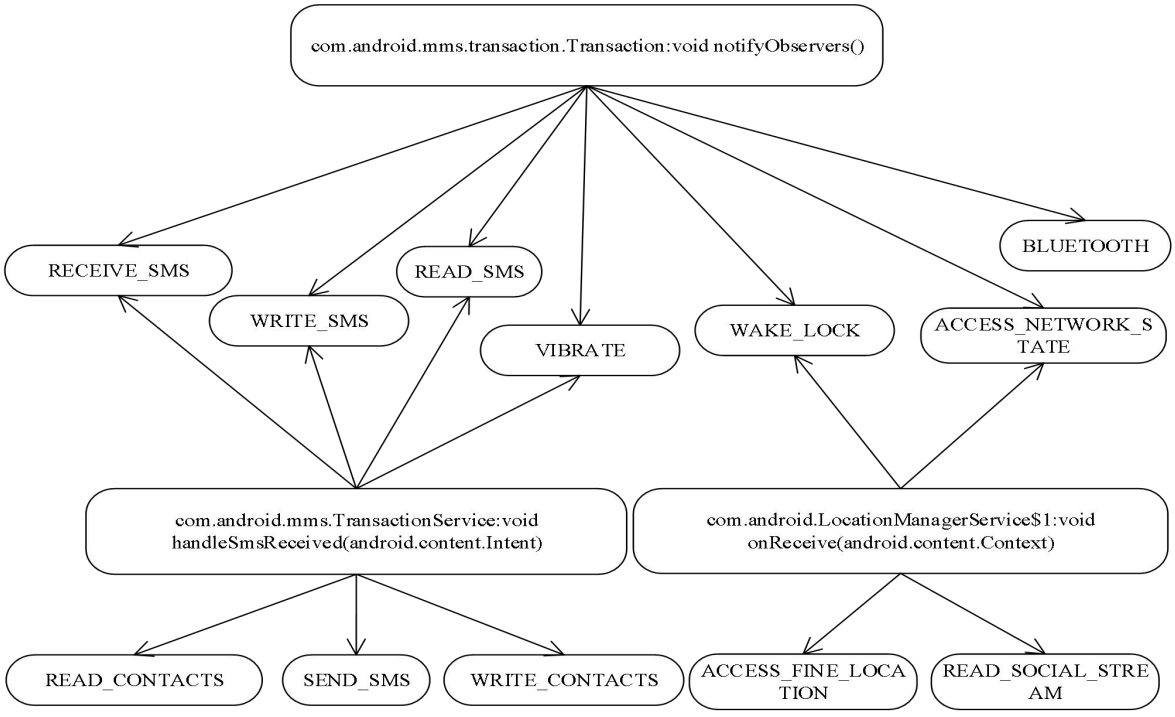


图 4-3 PScout 部分 API-权限映射条目

4.4 敏感 API 调用路径获取与分析

在 Android 系统中用权限机制控制一些敏感数据的访问，比如：短信，联系人信息，手机地理位置等。在访问这些感敏数据时，需要申请对应的权限，而是否需要特定的权限实际上是在一些 API 时进行申请并授予的。我们称这些和访问敏感数据相关的 API 为敏感 API。敏感 API 都有对应的权限作为调用的前提条件，如果在调用这些敏感 API 之前没有授予特定的权限，那么程序就无法正常运行。

为了更进一步的分析恶意应用，可以根据查询到的敏感 API 去遍历图中的路径，从而找到敏感 API 的调用路径，帮助分析发现权限漏洞。

4.4.1 图的邻接表结构和路径搜索算法

（1）图的节点和边之间的连通关系如图 4-4 所示。

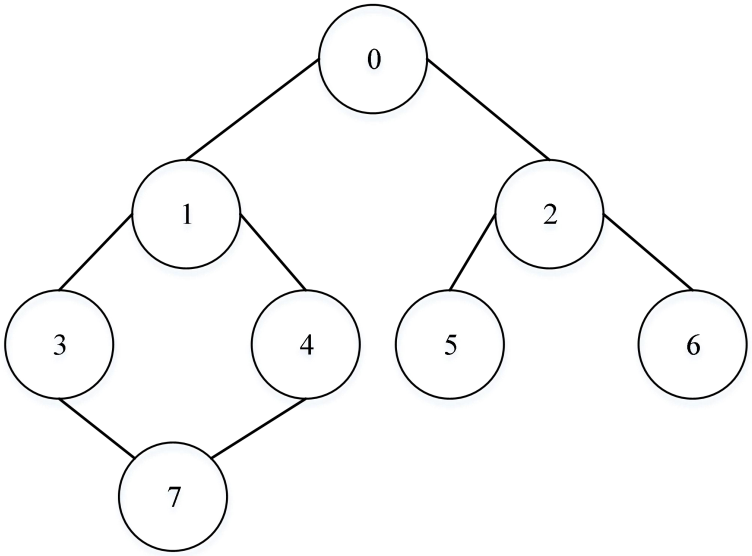


图 4-4 图的连通关系

（2）对应连通图 4-4 的邻接表如图 4-5 所示。

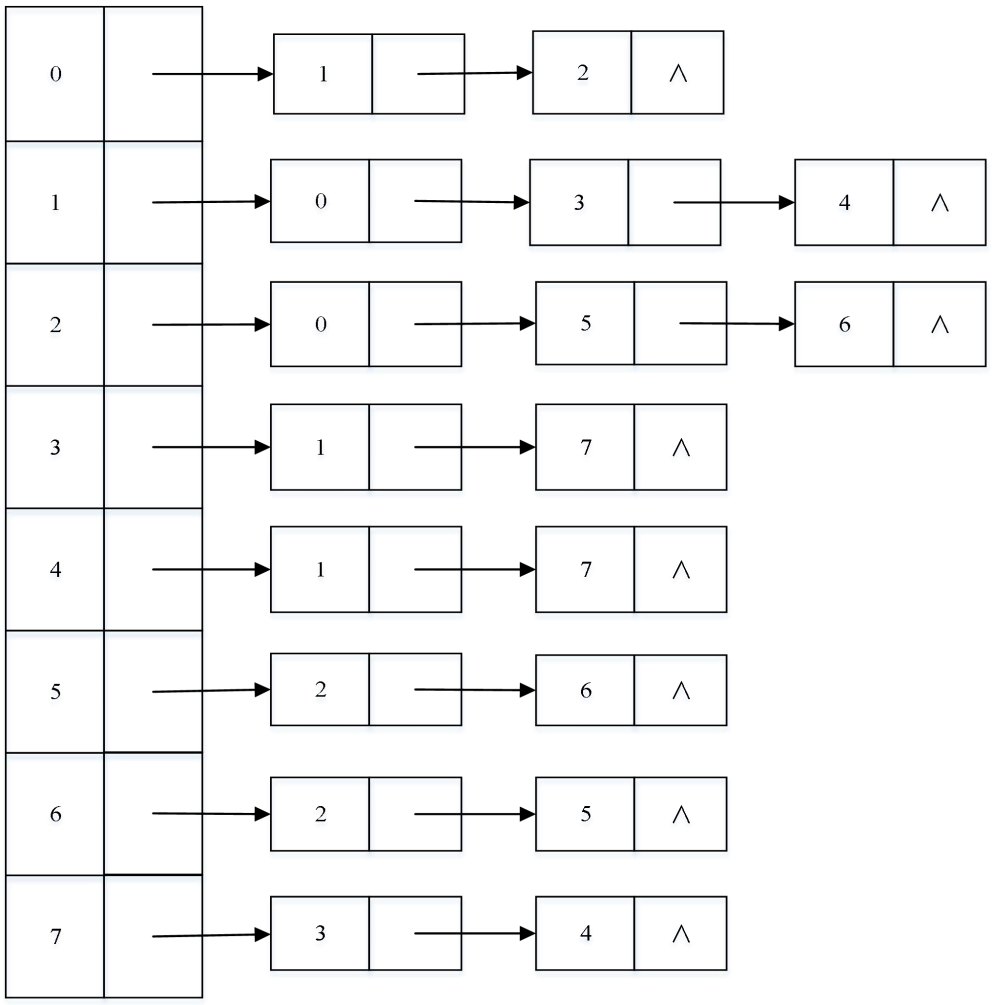


图 4-5 邻接表的数据结构

（3）给定起点和终点的路径搜索算法的实现如代码 4-1 所示。

代码 4-1 路径搜索算法

```
input://输入节点和节点间的连接关系
      G(V,E) //adjList
      (start,end)
output://所有从 start 开始，end 结束的路径
       start...end0
       start...end1
       start...end2

//查找 node 未访问的邻接表
```

```
function getNeighbour(node):
        list = adjList.get(start)
        for i to list.length -1
                node = list(i)
                if(!node.isVisited())//判断是否被访问
                return node
         return null
//查找图中的路径
function find_all_path(start, end):
         stack.push(start)
         for stack is not null
                topElement = stack.peek()
                if(topElement == end)//判断栈顶元素是否为终结点
                        printPath(stack)//打印出路径
                         stack.pop(topElement)
                else
                         visit(start)//访问 start 节点
                        visited.add(start,true)//将 start 节点标记为已经访问
                        //查找 start 节点的第一个未被访问的邻接点
                        neighbour = getNeighbour(start)
                        if neighbour is not null
                              stack.push(neighbour)
                              visited.add(neighbour, true)
                              start = neighbour
                        else //邻接点为空则出栈
                                visited.add(stack.peek(), false)
                                stack.pop()
                                start = stack.pop()
```

4.5 权限漏洞检测工具的实现

4.5.1 权限漏洞检测的工作流程

权限漏洞检测的实现，主要分成反编译、提取应用声明的权限、构建函数调用图、提取应用程序的 API 集合和提取应用程序的实际使用权限集合着几大步骤。具体的工作流程如图 4-6 所示。

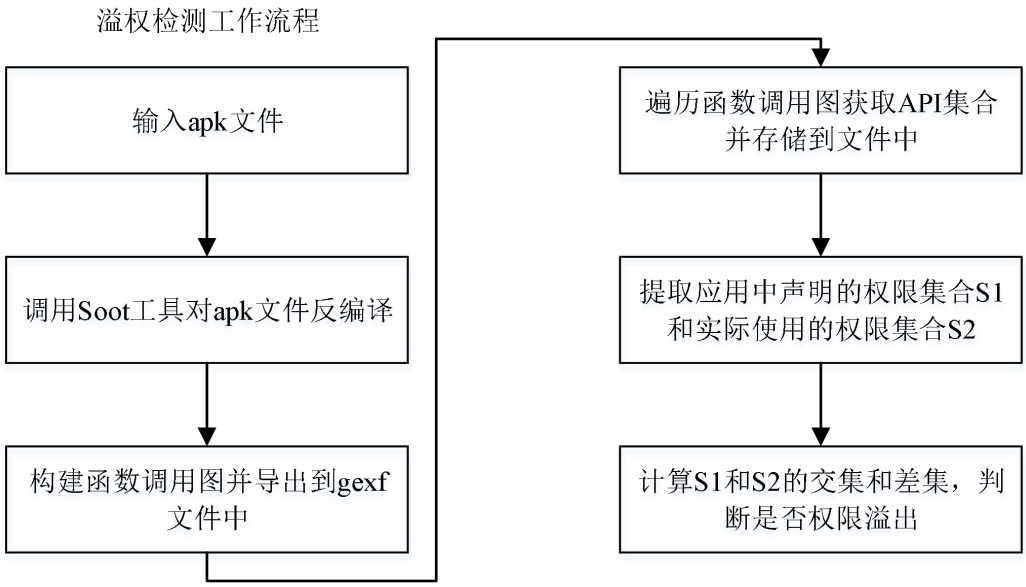


图 4-6 溢权漏洞检测的详细步骤

4.5.2 功能模块实现

如图 4-7 是溢权漏洞检测工具功能模块结构图

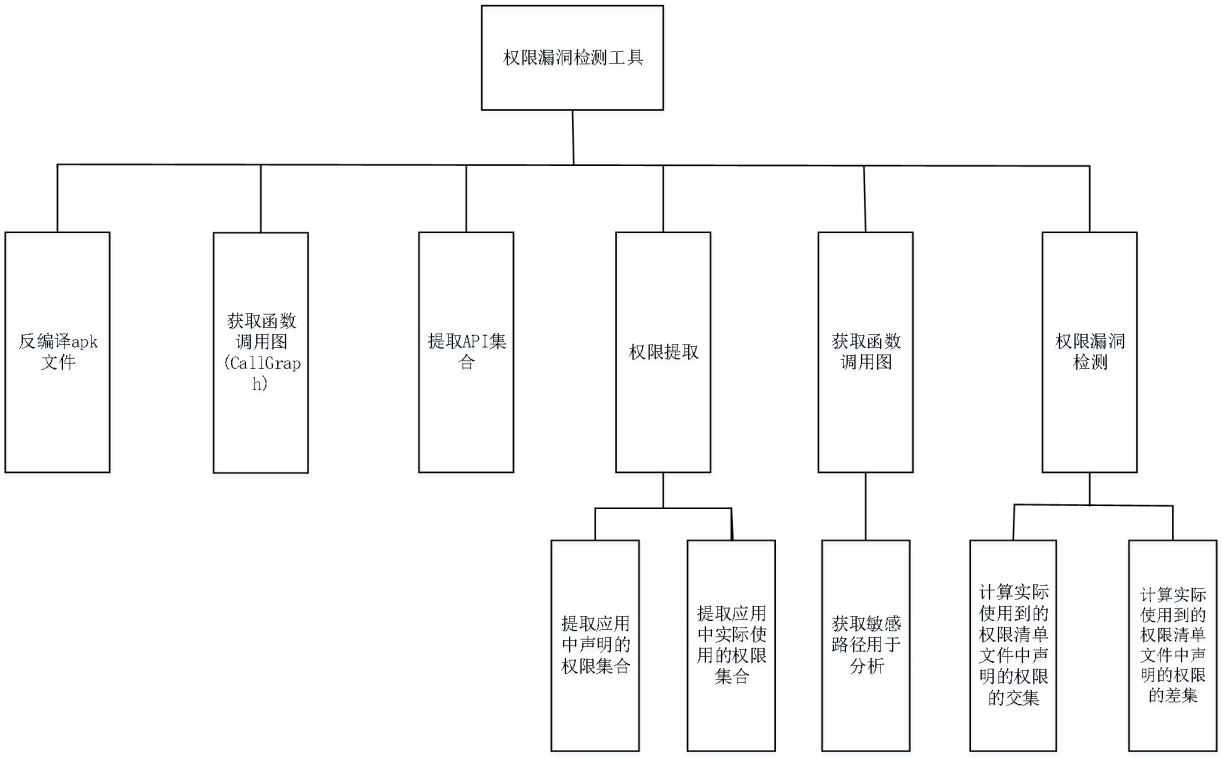


图 4-7 溢权漏洞检测工具功能模块图

（1）获取应用程序调用图并提取函数调用 api 集合，输入到文件中，伪代码如代码 4-1 所示。

代码 4-1 获取应用程序调用图并提取函数调用 api 集合实现伪码

输入：

F_{apk_file} //apk 文件

$P_{platform\_path}$ //android.jar 包文件路径

输出：

$F_{callgrph\_gexf}$

F_{use_api} //将 api 集合存储到文件中

function passCallGraph(CallGraph cg, SootMethod m)//遍历函数调用图的算法

visited.add(m, true) //标记被访问过的节点并存入到集合中

createCgNode(m，$F_{callgrph\_gexf,}$) //创建节点,并写入到 gexf 文件中

S_{use_api}.add(m)

S_{call} = cg.egdesInto(m)

if $S_{call} \neq S_{空}$

```
            while Scall has next element
                temp_method = Scall .nextValue()
                if  !visited.constains(temp_method) //判断节点是否遍历过
                    visit(cg, emp_method)
        Scalled = cg.egdesInto(m)
            if Scalled ≠ S 空
            while Scalled has next element
                temp_method = Scall .nextValue()
                if  !visited.constains(temp_method) //判断节点是否遍历过
                    visit(cg, emp_method)
    function extra_api(Fapk_file, Pplatform_path)
        Soot.initConfig()  //初始化 soot 配置项
        Flowdroid.initConfig()  //初始化 Flowdroid 配置项
        Soot.runPack()  //启动 soot 工具的 pack 任务
        decompile();  //反编译 apk
        CallGraph cg = Scence.v().getCallGraph() //通过 Soot 和 Flowdroid 协作获取
        ////应用的函数调用图 CallGraph
        DummyMainMethod mainMethod = Soot.createDummyMainMethod()// 创建
mainMethid
        passCallGraph(CallGraph cg, SootMethod m)
        saveToFile(Suse_api, Fuse_api )
```

（2）权限提取算法的实现伪码，如代码 4-2 所示。

代码 4-2 权限提取算法的实现伪码

```
输入：
    Fapk_file                  //apk 文件
    Fuse_api                //应用中提取的 api 集合
    Fapi_permission_mapping //api-permission 映射文件
输出：
    Sdeclaration_permission     //应用中声明的权限集合
    Suse_permission             //应用中实际使用使用的权限集合
    Ssame_permisison            //声明和使用权限集合的交集
```

$S_{diff\_permisison}$ //声明和使用权限集合的差集

function extra_permission:

$S_{declaration\_permission}$ = ApkUtil.getApkInfo(F_{apk_file}).getPermisisons()

list1 = readListFromFile(Fapi_permission_mapping)

list2 = readListFromFile(Fuse_api)

for i = 0 to i = list1.length

for j = 0 to j = list2.length

list1Str = list1.get(i)

list2Str = list2.get(j)

String singlePermission = list2Str.split("\\s")[0];//只包含权限名称字段

if (!permisisonSet.contains(singlePermission) and vagueMatch(list1Str, list2Str))

permisisonApiSet.add(list2Str);

$S_{use\_permission}$.add(singlePermission);

$S_{same\_permisison} = S_{declaration\_permission} \cap S_{use\_permission}$

$S_{diff\_permisison} = S_{declaration\_permission} - S_{use\_permission}$

4.5.3 溢权漏洞检测工具的实现

考虑 Flowdroid 和 Soot 项目都是用 Java 语言编写的，所以本次权限漏洞检测工具的实现代码也是用 Java 语言使用，使用 JavaSwing 提供的额 GUI 组件构建一个可视化的工具，实现封装了 apk 文件的反编译、权限提取、构建函数调用图和溢权漏洞分析等功能。

Flowdroid 项目中提供了函数调用图的数据结构 CallGraph,其中封装了程序中的函数调用关系。为了便于函数调用关系的分析，使用递归的方式遍历整个 CallGraph 结构，然后导出到.gexf 文件中，可以使用 Gephi 可视化工具查看应用的函数调用关系。

本论文构建的工具基于 Flowdroid 框架开发，将 apk 文件的反编译、权限提取以及恶意性检测等几大功能做了封装，实现了一个面向开发者的 GUI 界面工具。如图 5-4 所示是检测工具的主功能界面。

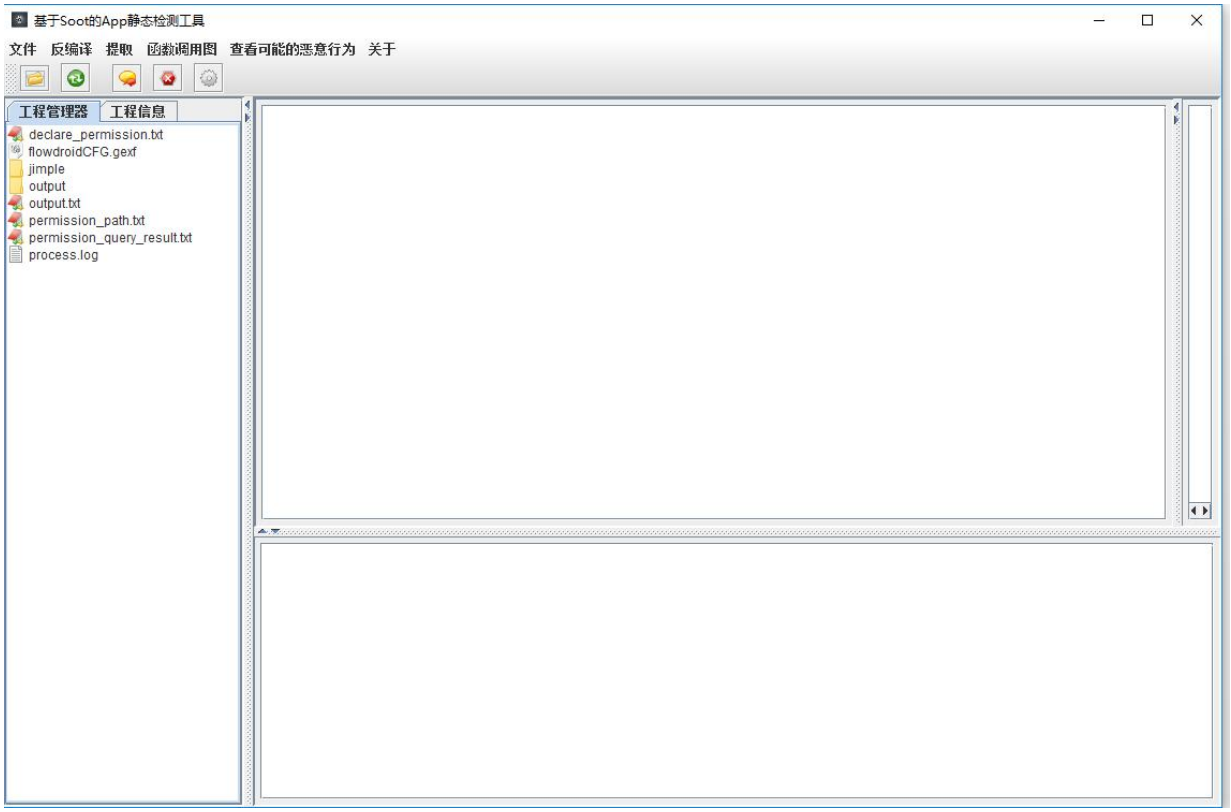


图 5-4 工具主界面

该工具的工作流程如下所示。

（1）配置 android.jar 文件的路径并选择要反编译的 Android 安装包文件路径，如图 5-5 所示。

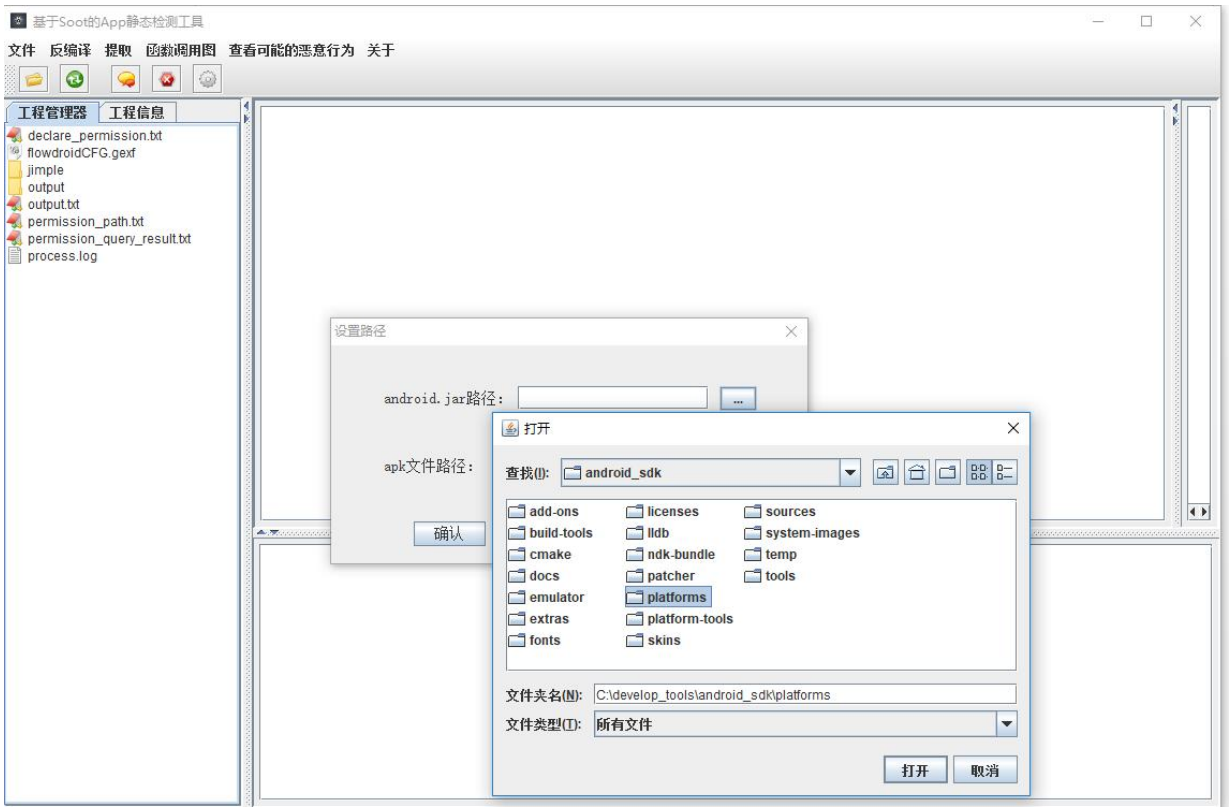
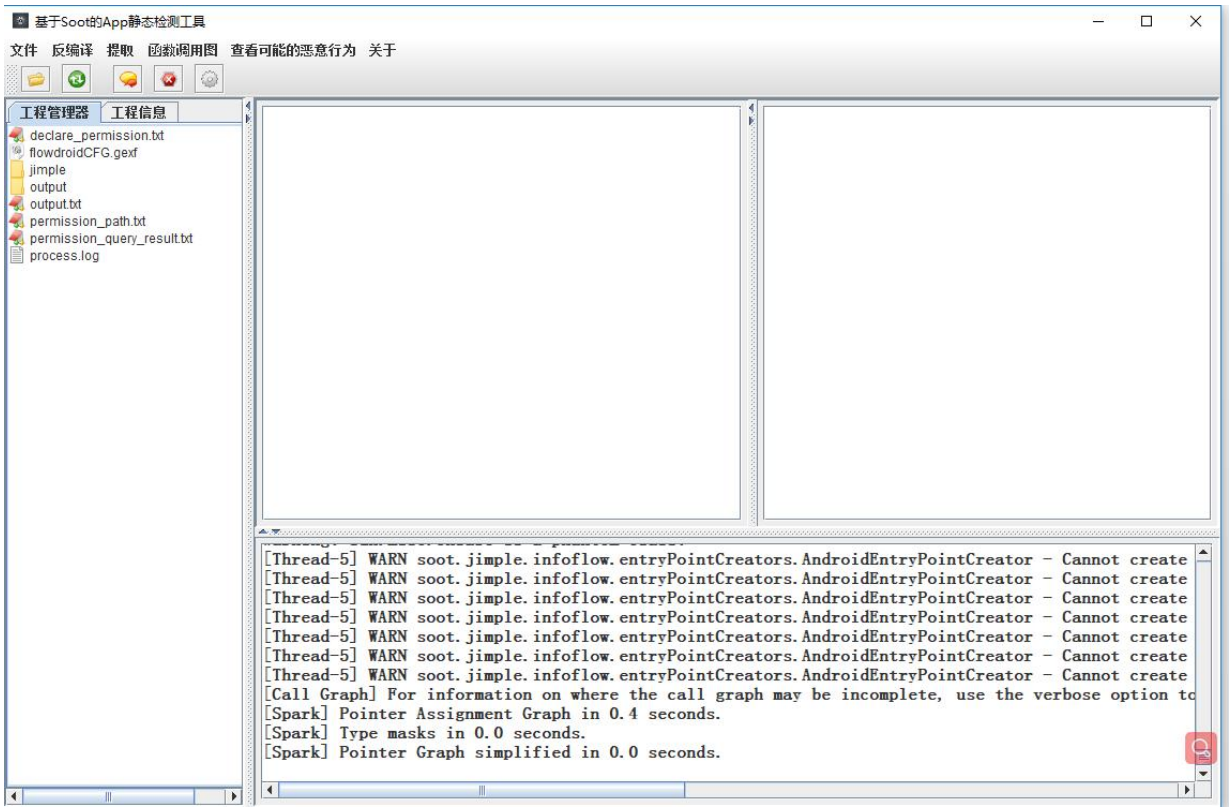


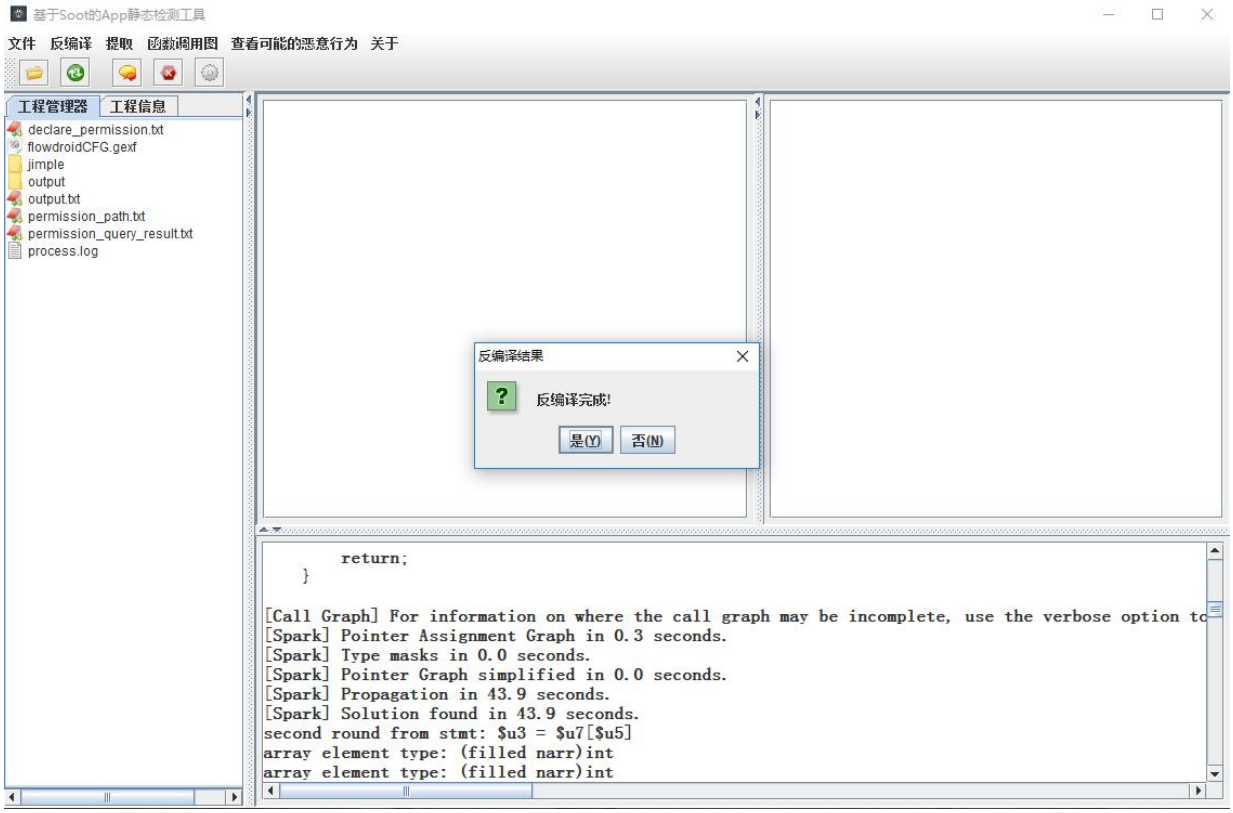
图 5-5 设置 android.jar 和 apk 文件路径

（2）加载完需要检测的安装包文件，点击工具栏的反编译按钮开始反编译，如图 5-6 所示。反编译完成会弹出提示对话框，如图 5-7 所示。



```
[Thread-5] WARN soot.jimple.infoflow.entryPointCreators.AndroidEntryPointCreator - Cannot create
[Thread-5] WARN soot.jimple.infoflow.entryPointCreators.AndroidEntryPointCreator - Cannot create
[Thread-5] WARN soot.jimple.infoflow.entryPointCreators.AndroidEntryPointCreator - Cannot create
[Thread-5] WARN soot.jimple.infoflow.entryPointCreators.AndroidEntryPointCreator - Cannot create
[Thread-5] WARN soot.jimple.infoflow.entryPointCreators.AndroidEntryPointCreator - Cannot create
[Thread-5] WARN soot.jimple.infoflow.entryPointCreators.AndroidEntryPointCreator - Cannot create
[Thread-5] WARN soot.jimple.infoflow.entryPointCreators.AndroidEntryPointCreator - Cannot create
[Call Graph] For information on where the call graph may be incomplete, use the verbose option to
[Spark] Pointer Assignment Graph in 0.4 seconds.
[Spark] Type masks in 0.0 seconds.
[Spark] Pointer Graph simplified in 0.0 seconds.
```

图 5-6 反编译 apk 文件



```
        return;
    }

[Call Graph] For information on where the call graph may be incomplete, use the verbose option to
[Spark] Pointer Assignment Graph in 0.3 seconds.
[Spark] Type masks in 0.0 seconds.
[Spark] Pointer Graph simplified in 0.0 seconds.
[Spark] Propagation in 43.9 seconds.
[Spark] Solution found in 43.9 seconds.
second round from stmt: $u3 = $u7[$u5]
array element type: (filled narr)int
array element type: (filled narr)int
```

图 5-7 反编译完成

（3）反编译完成得到中间代码，提取应用程序调用的所有 API 以及对应的权限集合，将提取到的结果存入到 output.txt 文件中，查看提取到的权限集合分别为图 5-8 所示。

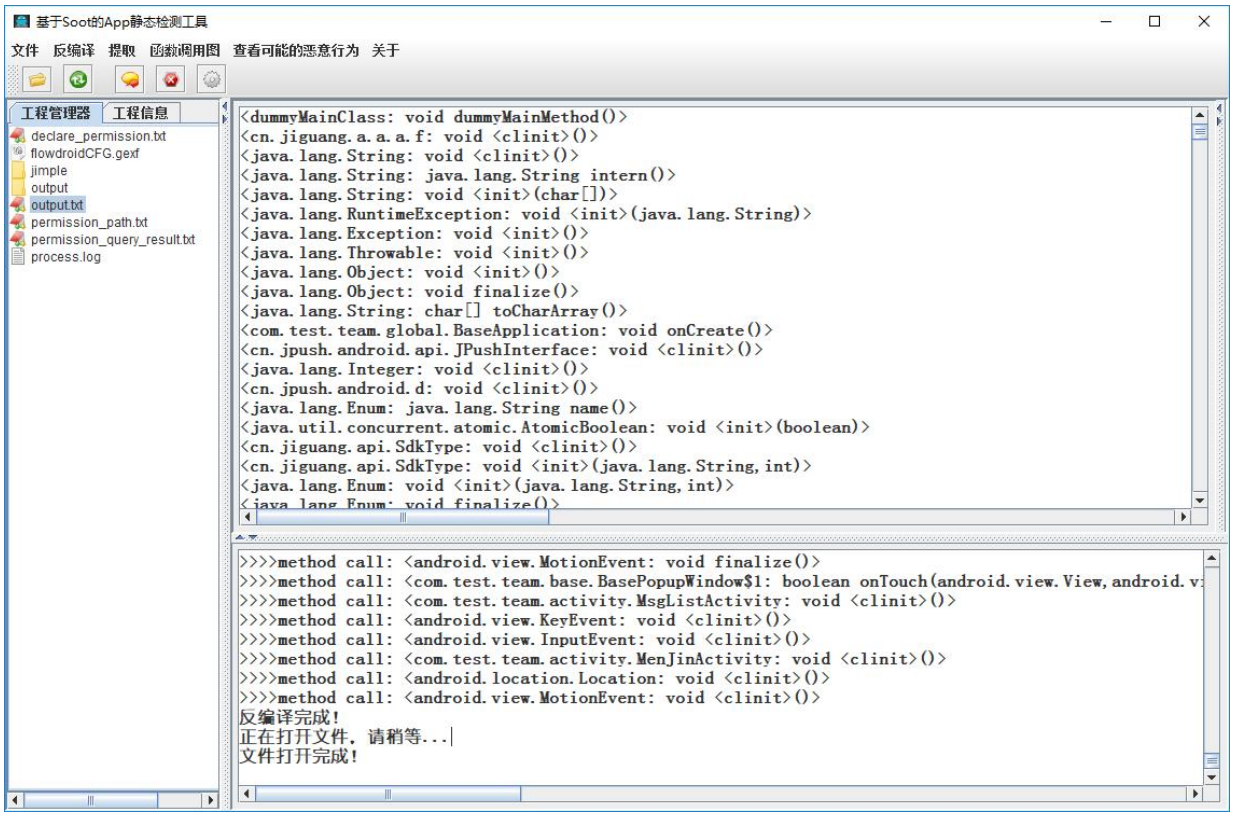


图 5-8 查看 API 提取结果

（1）提取应用程序中声明的权限，如图 5-9 所示。

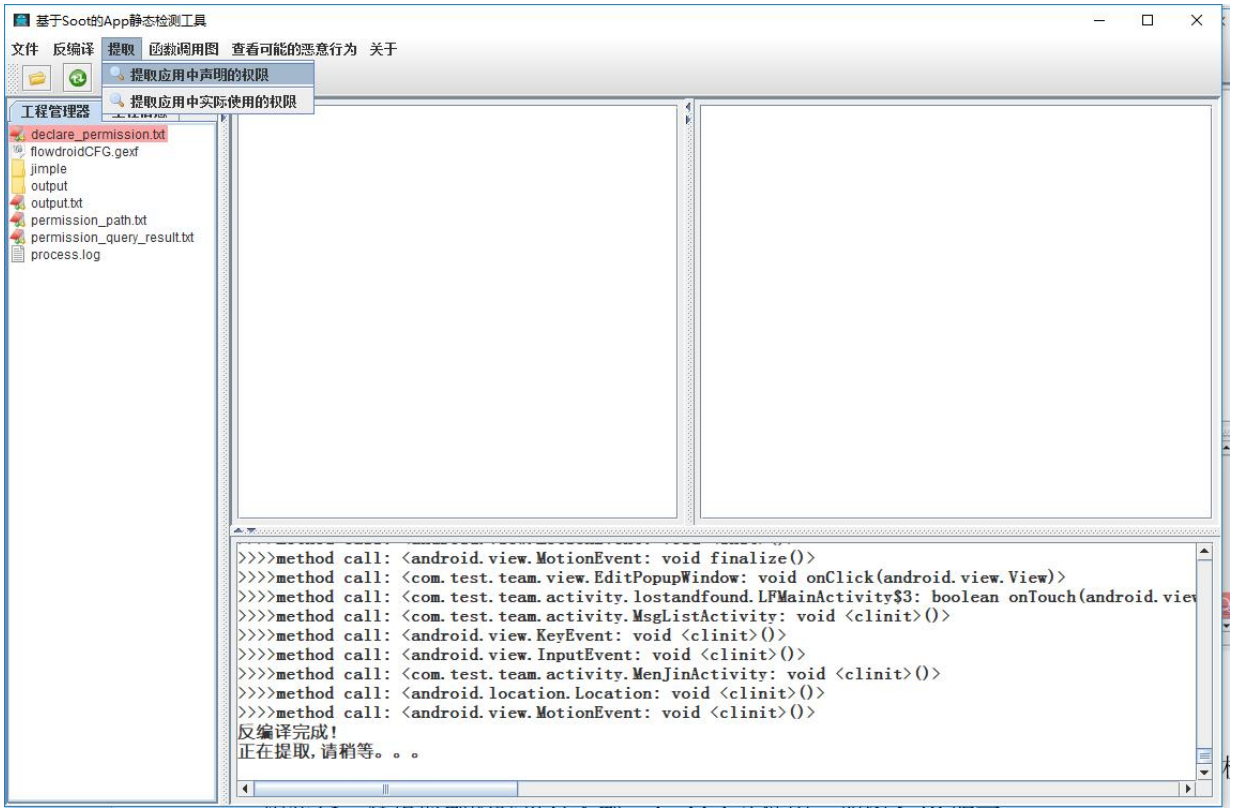


图 5-9 提取应用中声明的权限

（2）提取到代码中实际使用的 API 集合后，使用 pscout 项目的映射关系，可以查询到代码中实际使用到的权限集合，将提取到的结果存入到 permission_query_result.txt 文件中，如图 5-10 所示。

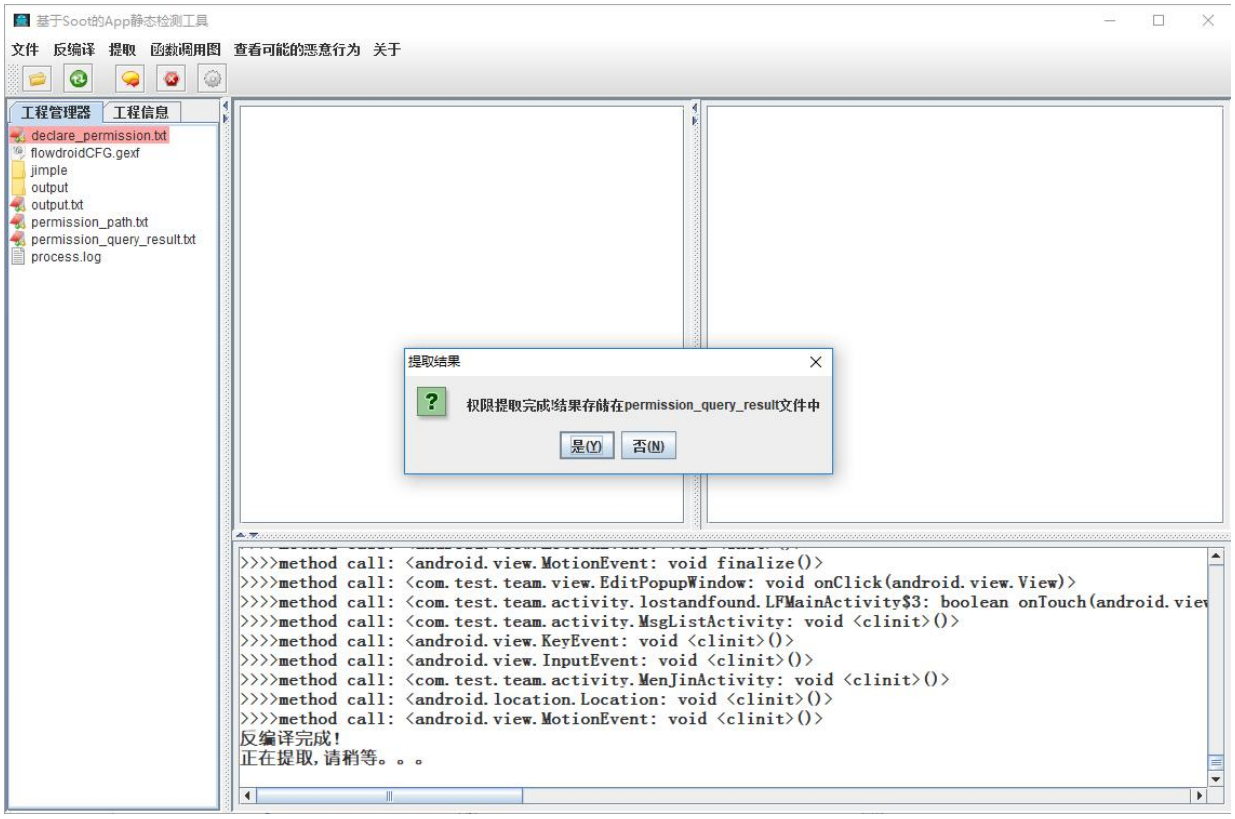


图 5-10 提取应用中实际使用到的权限集

（3）生成应用的函数调用图并存入到 gexf 文件中，然后用 Gephi 可视化工具打开，示例如图 5-11 中所示。

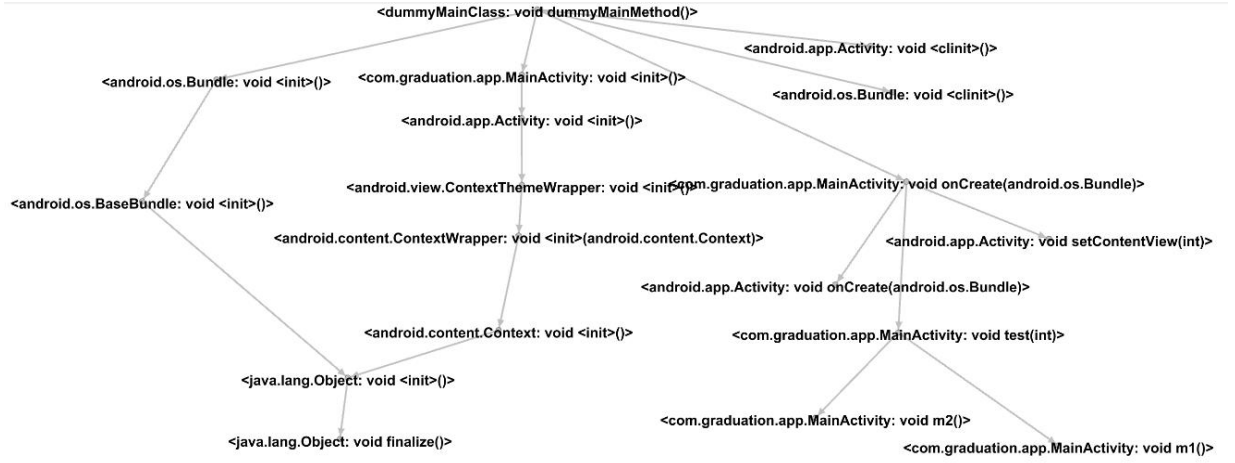


图 5-11 Gephi 可视化工具打开函数调用图

（4）查看恶意行为，如图 5-12 所示。

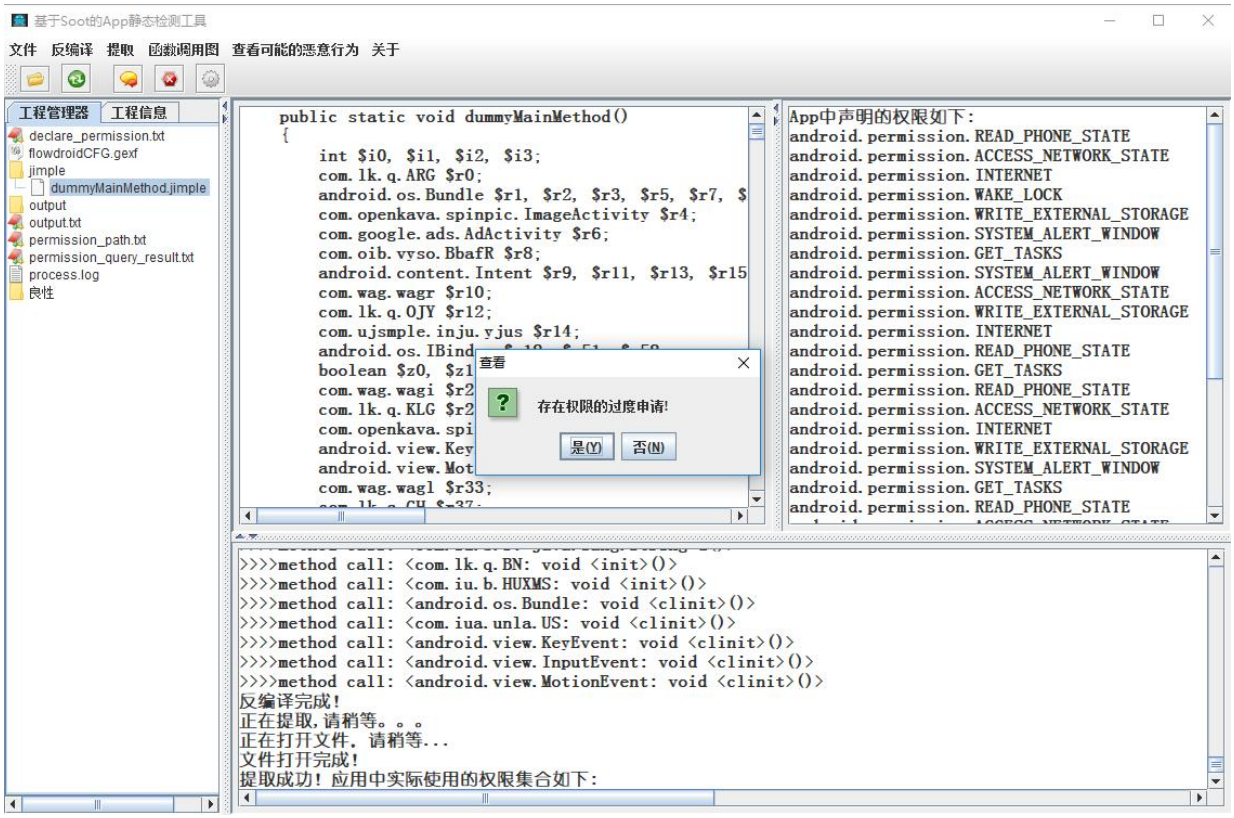


图 5-12 查看恶意行为

5 实验与结果分析

5.1 实验环境

本次实验环境配置如表 5-1 所示:

表 5-1 实验环境配置清单

操作系统	Windows10
处理器	I7-6700HQ
内存	8G(DDR4)
硬盘	500G 机械+250G 固态
实验工具	ApkToolBox(apktool、dex2jar、jadx)
IDE 工具	Eclipse、AndroidStudio3.1d+Android SDK

5.2 实验研究

本论文中的实例研究主要是为了验证基于 Flowdroid 中提取的 Android 权限集合是否为应用中实际使用的权限，不包括应用中从未被调用过的代码中使用到的权限。这一研究结果能够证明 Flowdroid 作为静态分析工具对权限提取的准确性和有效性。如果 Flowdroid 中提取到的权限只是 Manifest.xml 文件中出现的权限，那么对于申请了某些权限却未使用的权限，就有可能作为恶意行为的发起途径。

5.2.1 研究步骤

（1）先使用 Android Studio 编写一个简单的应用，在应用中申请一些代码中使用到的和未使用到的权限，这里使用 Android Studio 编写一个简单的 app 应用进行验证,清单文件中声明的权限展示在图 5-1 中。

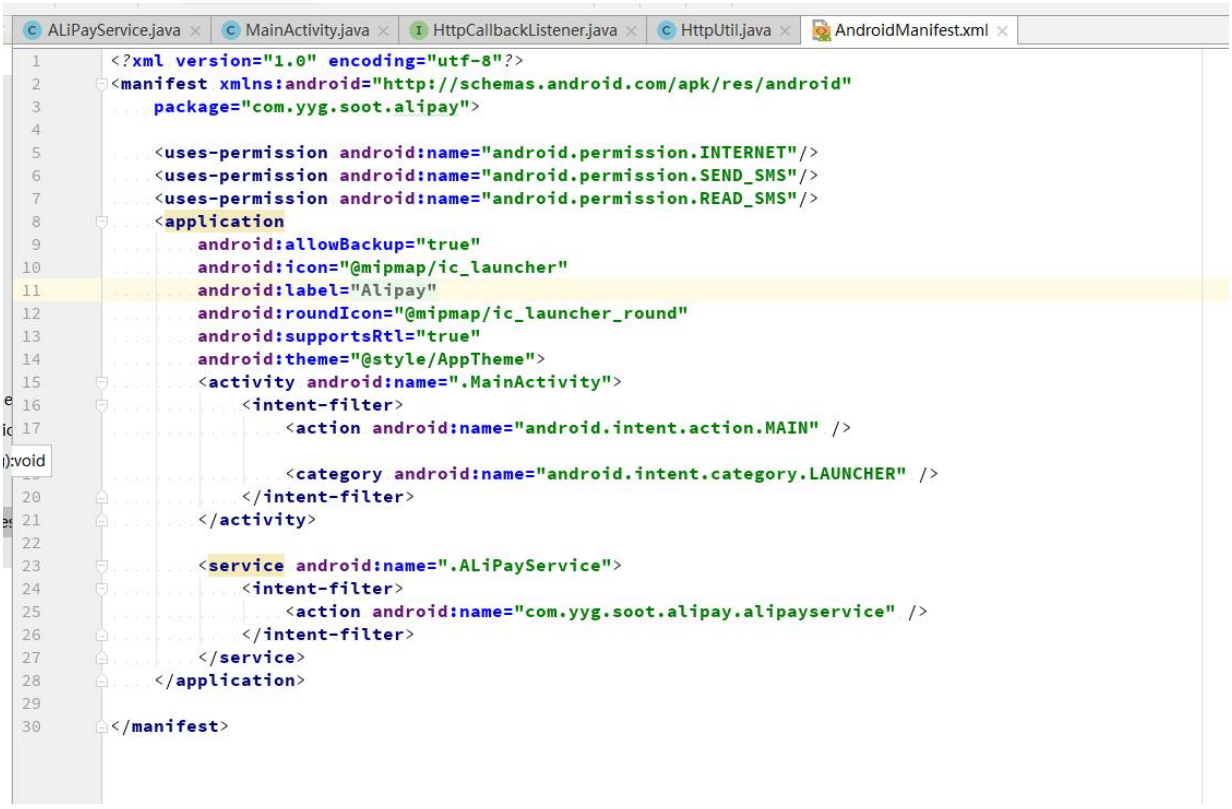


图 5-1 清单文件中声明的权限信息

（2）在申请的权限数量比较下的情况下，可以直接借助于 AndroidStudio 的搜索功能，查询提取出来的实际使用的权限的 api 位置，通过查看 api 的调用关系，判断是否是有效代码（被真正调用的代码为有效代码）。如下所示，图 5-2 是提取应用中声明的权限集合，图 5-3 是提取应用中的功能代码。

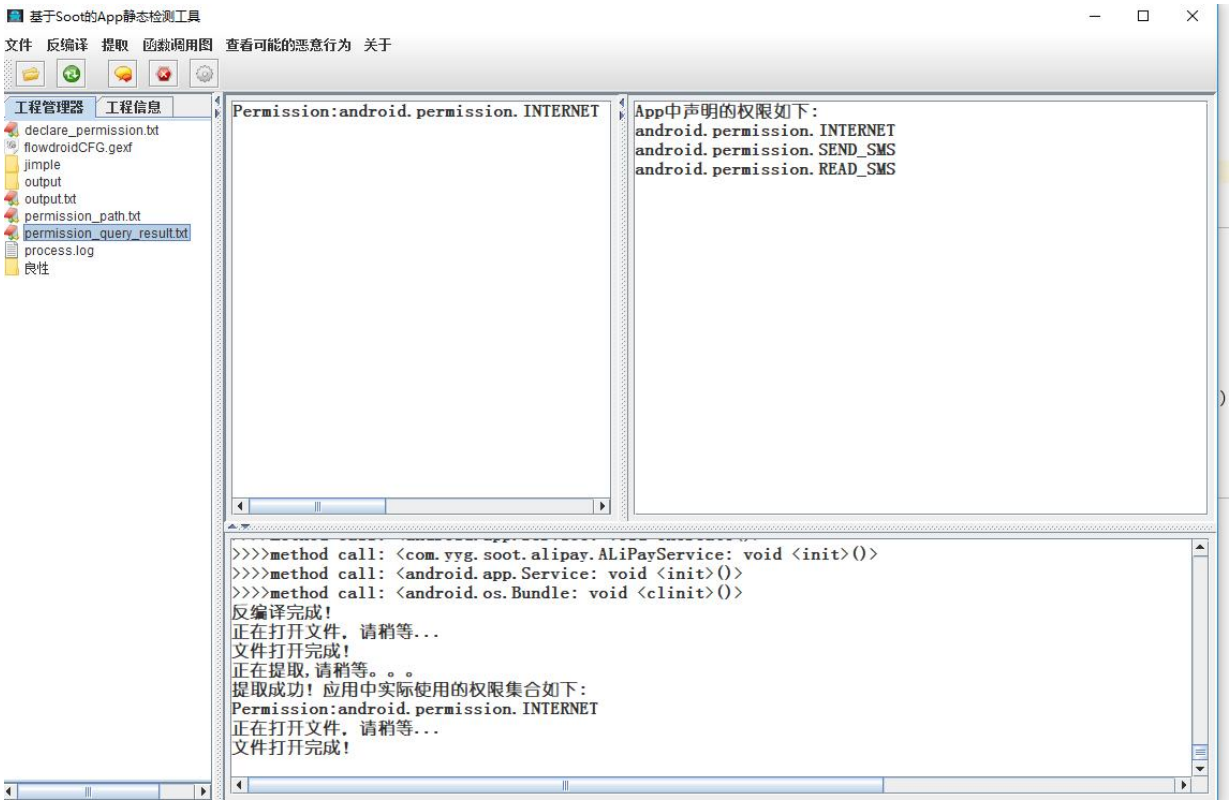


图 5-2 提取应用中声明的权限集合

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mServiceIntent = new Intent();
    //Android5.0以后不允许隐式启动Service
    mServiceIntent.setAction(INTENT_ACTION_SERVICE);
    mServiceIntent.setPackage("com.yyg.soot.alipay");
    //这里为了表明使用了网络访问，真正开发不建议在onCreate中编写耗时操作，以免造成初始化卡顿
    HttpURLConnection connection = null;
    try {
        URL url = new URL( spec: "http://www.baidu.com");
        connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setConnectTimeout(8000);
        connection.setReadTimeout(8000);
        connection.setDoInput(true);
        connection.setDoOutput(true);
        //获取输入流
        InputStream in = connection.getInputStream();
        //存储获取到的结果 设置编码方式
        BufferedReader reader = new BufferedReader(new
                InputStreamReader(in, charsetName: "GBK"));
        StringBuilder response = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            response.append(line);
        }
        //log
        Log.i( tag: "response------", response.toString());
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (connection != null) {
            connection.disconnect();//关闭连接
        }
    }
    //sendSMS("10086", "CXYE");//这里为了验证注释掉发送短信的方法
}
```

图 5-3 应用程序中的功能代码

5.2.2 结果分析

通过以上具体的实验案例可以看出，清单文件中虽申请了三个权限，但是真正在代码中使用的有两个，分别是发短信和网络访问权限。但是通过将发送短信功能的调用代码注释掉之后，实际使用的权限只剩下网络访问，这说明如果一个代码段没有出现在应用程序的调用关系中，在 Flowdroid 工具中是不会被检测到的，这样的处理更符合实际情况，可以降低漏报率。所以结论为：Flowdroid 工具提取出来的权限集合为项目实际代码中被调用到的，如果一个代码段未被真正的调用，它就不会被检测到。

5.3 实验步骤

第一步：从吾爱破解论坛或者应用商店下载一些 android app 安装包，作为实验的输入。

第二步：借助于实现的 GUI 工具，对上述下载的 apk 进行反编译、获取函数调用图、权限提取和恶意行分析，通过统计实验结果，得出实验结论。

5.4 实验结果与分析

5.5.1 实验结果

实验结果统计项说明：

（1） 检测成功数：因为开发者对应用的安全措施的不同，对 apk 文件的可能反编译工作并不能全部成功。比如：对应用做了安全加固处理、一些关键的功能放在了 native 代码中等等。所以这里需要统计真正能够反编译并提取成功的应用数量，作为统计比例的基数。

（2） 存在溢权漏洞的应用数：即提取应用中声明的权限集合大于真正使用的集合数。

（3） 溢权应用比例：存在溢权漏洞的应用数在检测成功应用总量的比重。

使用上述溢权检测工具对 100 个恶意应用和良性应用进行检测，应用从安全性检测网络下载，统计检测的结果如表 5-2 所示，

表 5-2 检测结果统计

	检测成功数	检测成功率	存在溢权漏洞的应用数	溢权应用比例
恶意样本	75	75%	58	77.33%
良性样本	89	89%	30	33.71%

（4）使用已有的权限检测工具也对上述相同的样本应用进行检测，并统计出数据如下表 5-3 所示。

表 5-3 已有的检测工具检测统计结果

	检测成功数	检测成功率	权限过度申请的应用数	溢权应用比例
恶意样本	90	90%	75	83.33%
良性样本	75	75%	42	56.00%

5.5.2 实验结果分析

通过分别对 100 个典型的良性应用的检测和恶意应用的检测，apk 文件的检测结果统计，其中恶意应用中存在溢权漏洞的达 77.33%，在良性应用中存在溢权漏洞的应用有

33.71%。相比于现有的溢权漏洞检测工具，对良性应用的检测更加准确，因为本次使用的工具是对真实使用的权限进行检测，而已有的工具是对代码进行扫描，一旦出现使用了某些权限的代码，都认为使用了这些权限，所以这种方式存在很高的误报率。但是对于本次的工具来说，因为 PScout 并不能包含所有版本的 API 和权限映射关系，所以检测出来的溢权应用比例偏低。但是从数据中可以看出：扩展自 Flowdroid 的工具对权限的提取更准确，误报率也低于现有的检测工具。

针对以上实验结果，进行正常应用和恶意应用的去权限分布统计，正常应用的权限分布统计如图 5-4，恶意应用的权限分布统计如图 5-5 所示。

6 总论与展望

6.1 总结

本论文以 Android 应用的权限提取和溢权检测为目标，在结合目前静态分析技术的基础上，利用了安卓应用静态代码分析的重要典型框架 Soot，并且重点研究学了基于 Soot 框架的 Flowdroid 框架。在 Flowdroid 框架的基础上实现了对 Android 应用的反编译、获取 CallGraph（函数调用图）、权限提取和溢权漏洞检测等功能，并将这些功能封装到一个可视化的工具中，方便权限的提取和分析工作。课题研究内容如下：

（1）查阅并学习 Android 权限机制与 Soot 框架相关的文献资料，对课题的背景和实际意义、国内外研究现状进行了详细的分析；

（2）在熟悉 Android 权限申请授予的过程，分析权限机制的工作原理，将权限-API 映射关系引入本课题，作为溢权漏洞分析的重要突破口；

（3）在熟悉 Soot 和 Flowdroid 的数据结构和用法的基础上，编写一系列的功能代码，设计并实现了溢权漏洞检测工具，达到协助 Android 应用静态权限提取与分析的作用。

6.2 展望

Android 平台的开源性是其迅速发展的一大优势，但同时也是其安全性保证的一大阻碍，因为 Android 系统源码是完全对外开发的。如果恶意者熟悉了系统的源码和工作机制，就会找到应用层的漏洞，就比如权限的漏洞，从而给恶意应用威胁用户数据安全的机会。其实，任何事物的高速发展，都必须要有对应的措施去保证其良性运作。比如，世界人口控制在一个合理的水平才能保证人类和生态环境之间保持平衡一样。安卓操作系统的市场比重居市场榜首，那么就应该有对应的安全保障措施，虽然目前 android 系统提供了各种安全机制，但是每年的 android 移动手机的恶意应用数量仍然持高不降。所以，对于 Android 系统安全性的研究和漏洞分析还有很长的路要走，这方面的研究价值也很重大。

致谢

四年时光如白驹过隙，转眼间便要毕业了，毕业设计虽然只有短短三个月的时间，但是在这三个月里我受益匪浅。非常感谢我的指导老师在这三个月里的帮助和指导。从刚开始的选题到学习交流成功中，特别佩服刘老师严谨负责的研究态度。在学习过程中，经常给我提出一些帮助我拓宽思路的问题，最让我手启发的就是任何不确定的结论，最好都要手动的去小心求证，这个事情说起来容易，可是要真正践行是很困难的。这种面对问题积极探索的态度值得去养成，这也将是我今后踏入工作岗位的一个研究学习的重要方法。

另外，感谢四年来我的老师和同学们，老师们传道解惑，教授我们以生存的技能，这将是今后立足社会的重要基础。感谢同学们四年来的关心和帮助，虽然离家偏远，但是并没有因为这个原因影响我正常的学习和生活。

最后，特别感谢学院组织此次毕业设计，从开学初的毕业实习就开始帮我们梳理毕业设计的相关工作内容，还特意邀请了陕西信息网络中心的老师过来指导我们进行毕业设计的前期准备工作。

参考文献

[1] Felt AP，Ha E，Egelman S，et al．Android permissions: user attention，comprehension，and behavior［C］,Proc of the 8th Symposium on Usable Privacy and Security． New York:ACM Press，2012: 3．

[2] 张玉清，王凯，杨欢，等．Android 安全综述［J］．计算机研究与发展，2014，51(7):1385-1396．

[3] 朱佳伟，喻梁文等.Android 权限机制安全研究综述[J],计算机应用研究,2015.10

[4] 杨晶,金伟信,吴作顺.基于 Android 系统的权限管理优化方案研究[J].电子质量,2015,(3):4-10.

[5] BlasingThomas,BatyukLeonid,SchmidtAubrey-Derrick,CamtepeSeyitAhmet,AlbayrakSahin. An Android application sandbox system for suspicious software detection[C]//IEEE Conference Publications.2010:55-62.

[6] OngtangM,McLaughlinS,EnckW,McDanielP.Application-Centric Security in Android. Proceedings of the Annual Computer Security.Applications Conference[C].

[7] ZeminLiu,Choon-Sung,NamDong-Ryeol,A User Authority Manager Model for the Android Platform[C] //ICACT2011. Feb.13~16,2011(1):146-1150.

[8] BartelA KleinJ，Dexpler:converting Android Dalvik bytecode to Jimple for static analysis with Soot［C］ACM Sigplan International Workshop on State of the Artin Java Program Analysis．ACM，2013:27-38．

[9] Einarsson A，Nielsen JD．A survivor's guide to Java program analysis with soot［D］. BRICS，Department of Computer Science，University of Aarhus，Denmark，2008:16-63．

[10] 肖伟，赵嵩正，魏庆琦．基于 AHP 模糊优先规划的虚拟团队成员选择方法［J］．统计与决策，2009(4):151-153．

[11] 许正权，王华清，王红军，等．基于模糊相似优先比法的高校教学团队成员选择［J］.价值工程，2014(26):1-2．

[12] 李浩君，项静，华燕燕.基于 KNN 算法的 mCSCL 学习伙伴分组策略研究［J］．现代教育技术，2014，24(3):86-93．

[13] Ritchie OM，ThompsonK．The UNIX time-sharing system［J］. Bell System Technical Journal，1978，57(6):1905-1929.

[14] 张骁敏，刘静等.基于权限与行为的 Android 恶意软件检测研究[J]，2017-03,第 3 卷第 3 期