北京郵電大學

本 科 毕 业 设 计（论文）

题目：Android 应用细粒度访问控制系统的设计与实现

姓　　名　卢文雄

学　　院　计算机学院

专　　业　计算机科学与技术

班　　级　2013211306

学　　号　2013211312

班内序号　1

指导教师　王浩宇

2017 年　5 月

Android 应用细粒度访问控制系统的设计与实现

摘　要

当前 Android 系统采用基于应用的权限模型进行访问控制。Android 应用的一个特点是广泛使用第三方库，且第三方库与宿主应用本身共享相同权限，因此很多第三方库存在侵犯隐私或者越权行为。对于不同的开发者，用户有着不同的信任级别，而且同一个开发者的多个应用中存在着隐私信息共享的情况，所以只需要对不同的开发者设置权限，而不是根据应用。所以，本文提出基于开发者的访问控制模型，将访问控制的粒度细化到开发者。对于一个应用内部，不同开发者适用不同的控制策略。对于同一开发者，在不同应用当中的同类行为，应用同样的访问控制规则。

网页浏览器对于脚本的执行采取的是同源策略，与移动应用领域基于应用的访问控制策略并不匹配。针对这些问题，本次设计提出一个 Android 应用细粒度访问控制原型系统，作为对现有访问控制机制的补充。借鉴浏览器的同源策略，用户可在开发者维度上对敏感行为进行控制，即允许/不允许来自某个开发者的代码使用某些敏感 API（或执行特定行为），将应用中特定敏感信息的可见范围控制在指定开发者。为验证这种细粒度访问控制方式的可行性，本系统针对两类敏感信息（精确位置信息，联系人信息）共计 3 种常用应用行为（位置监听，获取上次定位信息，联系人/通话记录列表查询）进行访问控制。

为实现细粒度访问控制，在 Android 系统 Framework 层，设计并实现了用于设置管理及设置检查的工具类。这些类提供了相关 API 供应用调用，并且基于敏感信息相关的 Android API，实现了敏感信息的模糊处理。在 Android 应用层，面向用户实现了用于管理相关设置的 APP 和服务。在 PC 端，基于 Soot 分析框架，实现了用于行为识别、应用行为修改（应用插桩）的工具类，用于对 Android 应用进行修改。并将各个模块部署到了 PC 端和移动端。

通过对 Android 应用进行插桩及测试，结果表明本系统能够起到允许或限制特定开发者执行特定敏感行为的作用，初步验证了对 Android 应用基于代码来源的细粒度访问控制方式的可行性。

关键词 Android 系统 敏感 API 细粒度访问控制

The design and implementation of fine-grained access control system for Android

ABSTRACT

Android system currently uses app-based permission model to control access to resources. One character of Android apps is the widely use of third-party libraries, and that third-party libraries and their host application itself share the same permissions. As a result, privacy violation and over-privileged manners exist in many third-party libraries. For different developers, users have different trust level, and private information sharing can be found among apps of the same developer, so it is only necessary to set permission for developers instead of apps. As a result, this paper suggests a developer-based access control model, which fine-grains the granularity of access control to developers. In one app, internally, different developers are applied with different control schemes. For the same developer, the same type of manners in different apps are applied with identical access control rules.

Web browsers use same-origin policy in script execution, while mobile apps use app-based access control, the policies do not match. To address these problems, this work suggests a prototype system of fine-grained access control for Android, in addition to current access control scheme. Inspired by same-origin policy of web browsers, this system allows users to gain control over sensitive manners in the dimension of developers, that is, to allow or disallow code from certain developers to use some sensitive APIs or extcute certain manners, limit the accessibility of certain sensitive information to certain developers. To study the feasibility of that fine-grained access control scheme, this system applies access control over two types of sensitive information (fine location and contacts) and three kinds of common app manners (location listening, getting last location, querying contacts and call log list).

To implement fine-grained access control, tool classes for configuration check and management are designed and implemented on Android Framework level. These classes supply related API for apps to call, and implement the fuzzing process of sensitive information based on related Android APIs. App and service used to manage related settings are designed for user on Android App level. On PC platform, tool class for manner identification and manner modification (app instrumentation) is implemented to modify Android apps. All modules are deployed to PC and mobile device respectively.

Instrumentation and tests on Android apps show that this system can allow (or limit) certain developers to execute certain sensitive manners, preliminarily examining the feasibility of origin-based fine-grained access control scheme.

KEY WORDS Android system Sensitive APIs fine-grained access control

目　　录

第一章 研究背景及动机

1.1 Android 访问控制机制简介

Android 系统采用权限模型对敏感信息的访问进行控制。在早期 Android 版本(<6.0)中，用户根据应用声明的权限决定是否安装，安装即允许应用使用相应权限。自 Android 6.0 版本起，增加了运行时权限检查机制，在 Target API≥23 的应用中，对于部分敏感权限，应用需要在首次使用之前向用户发出请求，获得允许后方可使用

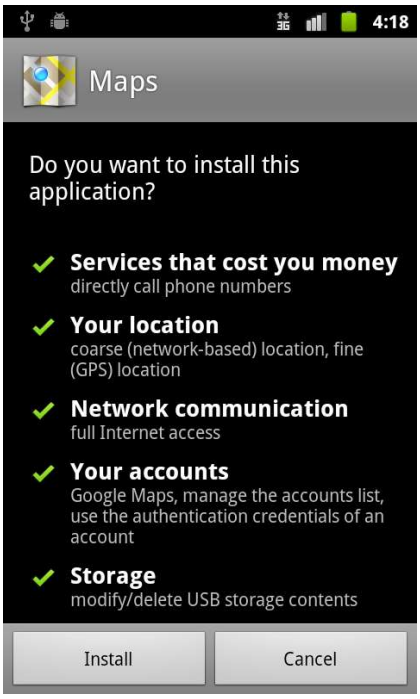

图 1-1 Android 早期版本权限确认示意图[10]

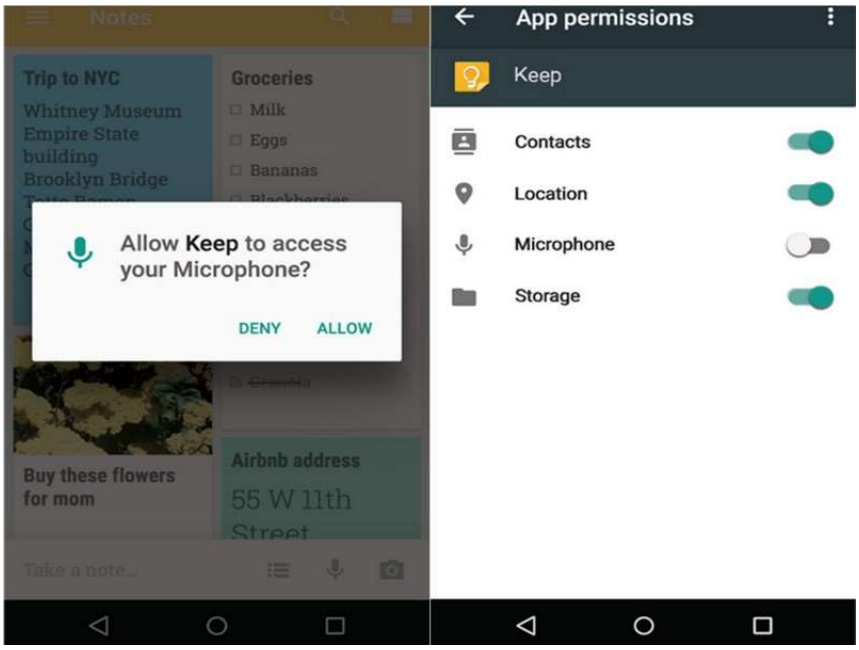

图 1-2 Android 6.0 版本运行时权限请求（左）和权限设置（右）[11]

Android 系统中，普通应用所用到的权限级别一般有[1]：

1）正常权限

如：蓝牙，访问网络状态

2）危险权限

如：获取联系人，获取位置信息

此外，还有一些特殊权限事关系统安全，保护级别要求更高，如“修改系统设置”。

危险及以上级别的权限，由于存在隐私泄露或系统安全风险，需要经用户允许方可使用。

Android 6.0 及以上系统将危险权限分为 9 个权限组[1]，用户可以根据应用功能，以组为单位选择是否允许使用权限。

表 1-1 危险权限和权限组[1]

权限组	权限
CALENDAR	READ_CALENDAR
	WRITE_CALENDAR
CAMERA	CAMERA
CONTACTS	READ_CONTACTS
	WRITE_CONTACTS
	GET_ACCOUNTS
LOCATION	ACCESS_FINE_LOCATION
	ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	READ_PHONE_STATE
	CALL_PHONE
	READ_CALL_LOG
	WRITE_CALL_LOG
	ADD_VOICEMAIL
	USE_SIP
	PROCESS_OUTGOING_CALLS
SENSORS	BODY_SENSORS
SMS	SEND_SMS
	RECEIVE_SMS
	READ_SMS
	RECEIVE_WAP_PUSH
	RECEIVE_MMS

STORAGE	READ_EXTERNAL_STORAGE
	WRITE_EXTERNAL_STORAGE

1.2 Android 权限模型存在的风险

1.2.1 应用开发中第三方库的使用

在 Android 应用开发中，第三方库是一种可复用的软件资源[2]，在当今的应用开发中使用越来越广泛。

常见的第三方库类型有：[2]

1）广告库，例如 MoPub, AdMob。

2）社交平台库，例如 Facebook。

3）实用工具库（包括定位库，分析库），例如 Google GMS

在 Android 应用开发中，第三方库多以 jar/class 文件的形式提供给开发者[2]，开发者除编写应用核心代码外，还可通过导入的方式在应用中引入第三方库。在应用编译时，核心代码与导入的第三方库都会被转换成 JVM 字节码，存储到应用的 APK 包中。

1.2.2 基于应用的权限模型的风险

在当今 Android 应用开发中，第三方库的使用越来越普遍，用途主要有定位，消息推送，广告服务等。然而当今 Android 权限机制，均是以应用为基准的，用户允许应用使用某种权限，即允许该应用的所有代码（包括核心代码和第三方库）使用相应权限，基于以上第三方库的使用现状，现有权限机制存在越权风险，第三方库可以利用宿主应用被允许的所有权限，进行敏感操作。

上文已经提到，应用所用的第三方库代码和应用本身的核心代码均在同一个应用的字节码中。在权限模型中，一个应用实际执行时，其权限声明及设置的作用范围是整个应用，包括核心代码及第三方库，因此第三方库具有和宿主应用核心代码相同权限。如果一个应用被用户授予某种隐私相关权限，则第三方库也能访问相关隐私信息。

在实际的应用中，这种授权方式存在越权访问风险[3]，例如，导航应用的开发者为了实现功能，声明了位置相关危险权限；用户为了确定自己所在位置，用于地图导航，授予该应用获取位置的相关权限。现有权限机制下，同一个应用中的广告库、分析库等也可以访问用户的位置。隐私信息可以通过上述越权访问方式，在用户及开发者对第三方库均缺乏了解的情况下泄露到非预期的域。

此外，在网页浏览器中，为了防止跨域隐私信息泄露，采用的是同源策略，而 Android 应用的访问控制是基于整个应用的，包括核心代码和第三方库。由于移动应用的网络交互日渐丰富，网页和移动应用的安全策略在这一方面并不匹配，产生安全风险。

1.3 Android 应用权限风险及访问控制的相关研究

关于第三方库的越权访问风险，Ryan Stevens 等[3]曾进行相关研究，进行了广告库所用到的权限与文档说明的权限的对比，发现很多广告库会使用一些在其使用文档中没有

说明需要使用的权限，并指出，在应用中使用广告库，实际上允许了广告库利用和该库的宿主应用相同权限，当这些广告库动态检测到所在应用具有相应权限时，它们就会使用这些权限，其中很多都是敏感权限，广告提供者进而从用户设备提取相关信息。

该研究还指出，这种形式的泄漏，在浏览器上是比较困难的。因为浏览器采用了同源策略。

在各类网页浏览器中，出于系统安全性考虑，采用同源策略，即某网页加载一个脚本，执行时只能与网页自身所在域进行信息交流或页面请求。而对于 Android 应用而言，一旦被授予某种权限，则第三方库代码自动获得该权限，导致敏感信息泄露到应用核心以外的域。

关于 Android 访问控制机制的改进，Bin Liu 等[4]曾提出一个 Android 应用细粒度访问控制系统，主要针对第三方库中广告库的安全风险进行访问控制。该研究中访问控制的机制是：利用机器学习区分应用核心以及第三方库，以应用为基准，允许或者禁止应用核心或第三方库访问特定资源。且系统对该应用的所有第三方库的同一类访问采用相同设置，即都禁止或都允许。例如，针对某款社区生活服务应用，可以设置允许应用核心访问精确位置，但禁止所有第三方库访问精确位置。以此将应用核心及第三方库能访问的信息进行区分。

第二章 系统分析及技术介绍

2.1 细粒度访问控制系统分析

基于以上对 Android 第三方库及权限模型风险的分析，本次设计需要实现一个 Android 应用细粒度访问控制原型系统，对应用中第三方库的越权行为进行限制。用户可以根据需要，允许或禁止应用核心或者第三方库进行访问敏感信息的行为。该原型系统作为对现有访问控制机制的扩展，用来验证基于开发者（域），借鉴同源策略的细粒度访问控制的可行性。

现有 Android 访问控制机制，以及 Bin Liu 等提出的细粒度访问控制系统，都是基于每个 App 的。鉴于网页和移动系统（应用）的安全策略并不匹配，我们提出了如下设想：在本文设计的细粒度访问控制系统中，引入类似浏览器同源策略的控制机制，允许用户进行设置，以允许或者限制来自特定开发者（相当于网页同源策略中“域”的概念）的指定隐私信息访问行为。

本系统与 Bin Liu 等设计的控制系统的主要区别，在于引入了类似同源策略的访问控制机制，打破了应用之间的界限，并将粒度细化到开发者，从单个应用角度看，进行了比原有 Android 权限模型更加细粒度的访问控制，但如果考虑到多个应用使用同一开发者的第三方库的情况，同一开发者的应用行为控制又服从了同样的规则。

值得注意的是，同一个开发者可能会开发很多种第三方库，很多流行的第三方库往往会被多个应用使用。如果用户可以根据开发者选择是否允许访问指定隐私信息，将有助于控制特定开发者在多个应用（或多个第三方库）中给用户隐私带来的潜在风险。

在访问控制阶段，为验证系统可行性，本系统选取与用户隐私信息（精确位置信息、联系人）有关的 2 类，3 种具体的应用行为进行识别，并加上相应的设置检查与信息保护动作，具体如下：

1） 利用 Google Play Service 进行精确位置获取：

a) 实现位置监听接口 LocationListener，具体实现 onLocationChanged()方法，用于监听到位置变化时执行响应动作。

b) 利用 getLastKnownLocation()方法，获取设备上一次定位成功时的位置。

以上两种行为是应用开发时，与获取/使用精确位置有关的常用行为。

2） 利用 ContentProvider 类的 query()方法[9]，查询用户设备的联系人/通话记录列表。

对于上述位置相关行为，可以在获取到的位置上加上随机偏差，或直接设定一个虚假位置，实现位置信息模糊化。对于联系人/通话列表查询行为，可以通过截断的方式阻止查询。

设计思路如下：

在用户设置阶段，需要用户自行制定访问控制策略，允许或者禁止特定开发者的代码访问特定资源。

在应用插桩阶段，该系统需要对应用进行如下处理：对应用中使用特定 API 获取敏感信息的字节码进行静态分析，插桩用于修改或清除相关敏感信息的代码段，并在应用中增加设置检查步骤。

在访问控制阶段，在敏感信息访问处，需要通过被插桩的代码段动态检查访问行为是否被用户的策略设置允许；若允许，则直接执行访问行为；若不允许，则执行被插入的特定代码段，改变应用相关行为。

为此，该系统需分块实现以下功能：

1）针对 Android 应用敏感信息使用行为，进行识别及代码插桩，生成修改后的应用。

2）允许用户通过界面，向系统中添加基于开发者的访问控制设置

3）对系统设置项进行管理，并实现设置项与系统设置存储的互通。

4）针对识别出的隐私信息使用行为，在运行时检查用户设置，以进行访问控制。

5）在访问控制过程中，针对具体行为，对隐私信息进行修改或访问截断。

2.2 技术介绍

本篇论文中使用的主要技术有：

1） 基于 Soot 分析框架的应用分析及插桩技术

2） 基于 Android 系统设置内容提供者的系统设置存取技术

3） Android 应用（含界面）及服务设计技术

由于前两项技术在一般的 Android 应用开发中并不常用，下面会进行重点介绍。

2.2.1 Soot 分析框架简介

Soot 是一个用于 Java 程序分析、优化及修改的开源框架，在其诞生的初期，在 Java 程序的逆向分析及优化中有广泛的应用。

Soot 可以使用 Jimple 中间代码作为代码分析过程中的统一表示[5]。Jimple 中间代码是一种三地址代码，可以表示 15 种语句和 29 种表达式[6]，这些语句和表达式类型包括了运算，比较，赋值，调用，条件分支等常用操作。

在 Soot 框架内，经过变换的输入程序，其所有类均存放在一个容器里，称之为 Scene，其中每一个类构成一个 SootClass，类中每一个方法构成一个 SootMethod，每个 SootMethod 包含一个方法体 Body，每个 Body（即方法的具体实现）由一个语句链构成，语句链中每个 Jimple 语句均可包含 Jimple 表达式。

Soot 允许的输入形式有 class 文件，Java 源码，jar 包，Android APK 安装文件等，输出形式有 class/Jar/APK 等，在进行分析或变换时，无论何种输入形式，均被转换成 Jimple 中间代码进行处理。

Android 应用 APK 文件中代码的存在形式是 Dalvik 字节码。利用 Dexpler[7]转换工具，Soot 可以将输入的应用 APK 包反编译成 Jimple 中间代码，并将分析/修改后的中间代码回编译至 Dalvik 字节码，输出到一个新的 APK 文件。

Soot 框架的工作流程如下图所示：

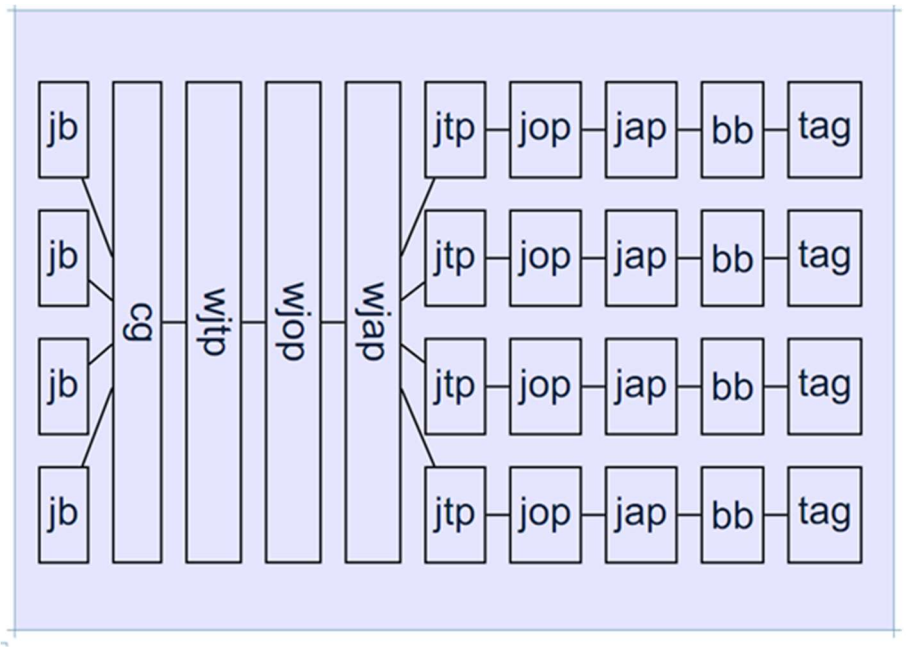

图 2-1 Soot 工作的各阶段示意图[8]

其中 jb 阶段是将输入程序中实现的每一个方法均转换为 Jimple 方法体（Soot 框架中称 Body），用于对方法的分析与变换。

wjtp 阶段是针对整个程序的分析/变换，将每个方法均视为整体，可用于在程序中添加/删除类的实现，以及在已有类中添加/删除整个方法。

jtp 阶段是对每个方法对应的 Body 进行分析/变换。常用于对方法中的语句进行识别，以及对方法的具体实现进行修改，如增加局部变量，添加删除语句等。

默认情况下，Soot 框架只会对输入后得到的中间代码进行分析，不做修改。如要修改，需要在相应工作阶段添加自定义 Transform 对象，在对象构造过程中，通过继承 Transformer 的相应抽象子类(BodyTransformer/SceneTransformer)，并具体实现相应类的 InternalTransform 方法，实现自定义代码修改动作。

下图中，MyBodyTransformer 的 InternalTransform 方法，实现的是对方法的分析和方法实现的修改，运行于 jtp 阶段，作用于输入程序的每一个方法，每次执行作用于一个方法。而 MySceneTransformer 的 InternalTransform 方法，实现的则是对类的分析和修改，运行于 wjtp 阶段，视方法为一个整体。

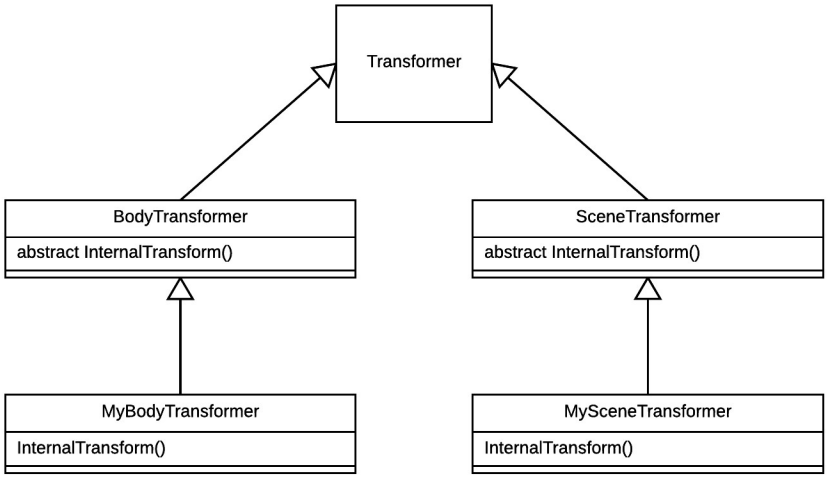

图 2-2 Transformer 类继承关系图

2.2.2 系统设置存取机制及 SettingsProvider 简介

在本次设计的控制系统中，用户做出的关于访问控制的设定存储在“系统设置”区域。

Android 系统使用 SettingsProvider 为应用提供设置数据共享，允许应用读写系统设置。在 Android 6.0 以下的版本，SettingsProvider 的具体实现是通过数据库操作；6.0 及以上版本 Android 系统中改为 XML 文件。

系统设置以“键-值”对的形式进行存取，“键”即设置项名，“值”即设置项数值。“键”的类型只能是字符串，“值”的类型可以是字符串或整数、浮点数。

为了系统安全，Android 系统将系统设置分为 3 类，Settings.System，Settings.Global 和 Settings.Secure，这 3 类设置均允许任何应用读取，但只有 Settings.System 中系统预定义条目允许普通应用写入，其他设置的写入均需要系统或 root 权限。

在本次设计的访问控制系统中，相关设置存储在 Settings.System 中的自定义条目，条目的“键”在编程时给定。

在应用开发中，为方便开发者使用 SettingsProvider，Android 提供了 Settings 相关的 API，用于键值对的读写操作。为了以字符串的形式读写系统中访问控制相关设置，需要实现将这些设置格式化生成单个字符串，以及系统设置中读取的字符串的解析。

第三章 系统设计

本次设计提出了一个基于同源策略的 Android 应用访问控制原型系统，在设计上由以下几个模块构成：

1）一个基于 Soot 框架的应用插桩工具类 AndroidInstrument，用于应用分析及修改。

2）隐私信息修改、截断工具类：LocationFuzzer 用于根据已知位置生成随机偏差，得到模糊化处理后的位置。

3）一个用于管理用户相关设置，及进行运行时检查的工具类 OriginPermission，对应用提供一个可供调用，以检查设置的 API。

4）一个允许用户手动添加，查看，修改，删除相关设置的应用 OriginPermAssist。

5）在 4）的应用中实现了一个后台服务 OriginPermPromptOnReq，用于监听来自应用的添加相关设置的请求，弹出窗口，允许用户做出选择。

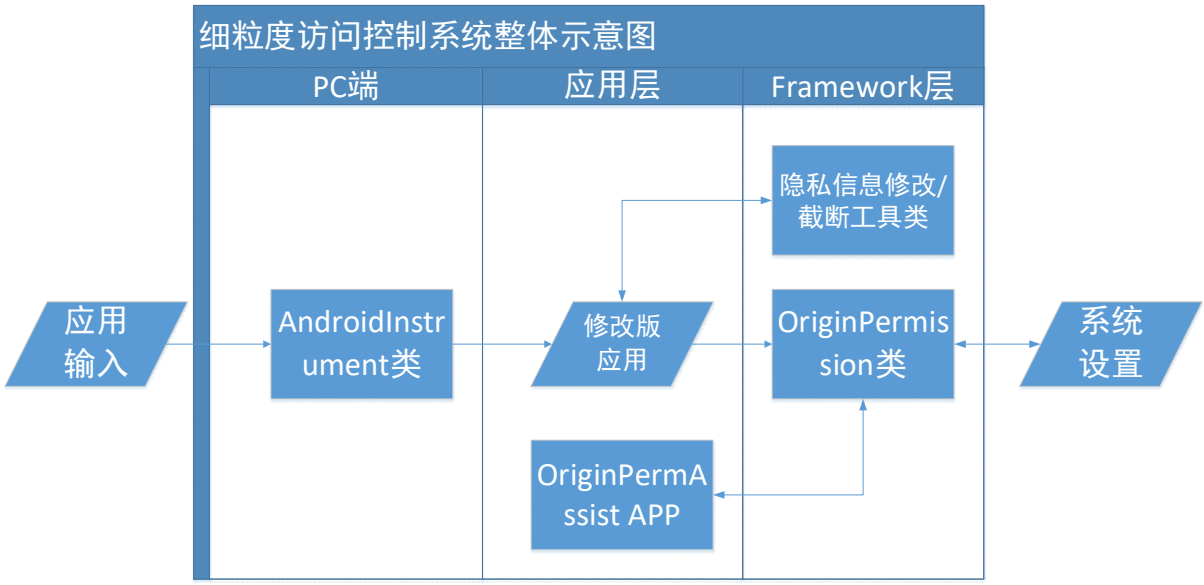

图 3-1 细粒度访问控制系统整体示意图

系统工作流程如下：

1）应用修改：

调用 AndroidInstrument 类的主函数，将应用 APK 文件及隐私信息修改/截断工具类统一导入 Soot 框架，并对给定应用进行特定敏感行为分析及代码插桩，以改变应用行为。

2）将修改后的 APP 进行重新签名，并安装到 Android 手机上。

3）用户设置：

用户对访问控制进行设置，有两条途径：

a) 直接打开 OriginPermAssist 应用，在界面上手动添加或修改相关选项，允许（或者禁止）特定开发者执行指定的行为。

b) 1）中被修改的应用运行时，会检查相关选项，如果未设置，会通知

OriginPermAssist 的后台服务，该服务随后会弹出一个对话框，内容为相关代码的开发者及敏感行为，提示用户设置相关选项，选择允许或禁止。

4）访问控制：

在 1）中被修改的应用运行时，敏感行为附近被插入的代码段会进行以下动作：

a) 调用 OriginPermission 类的静态 API，检查用户的相关设置，返回结果（允许/不允许），若未设置，则先请求设置。

b) 若结果为“不允许”，会调用相关工具类，对隐私信息做出模糊或清除处理，保护用户隐私。

系统工作流程示意图如下：

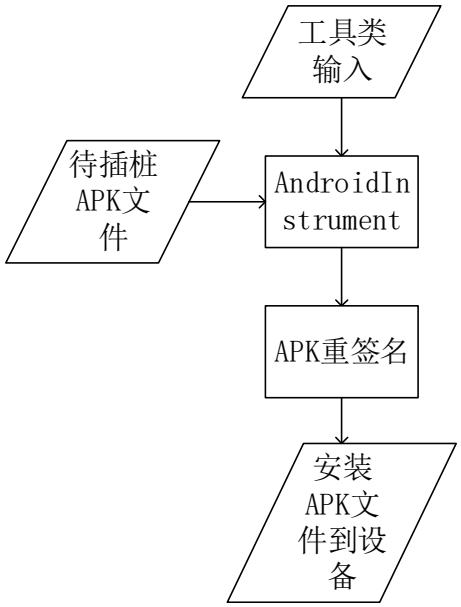

图 3-2 应用插桩及部署流程

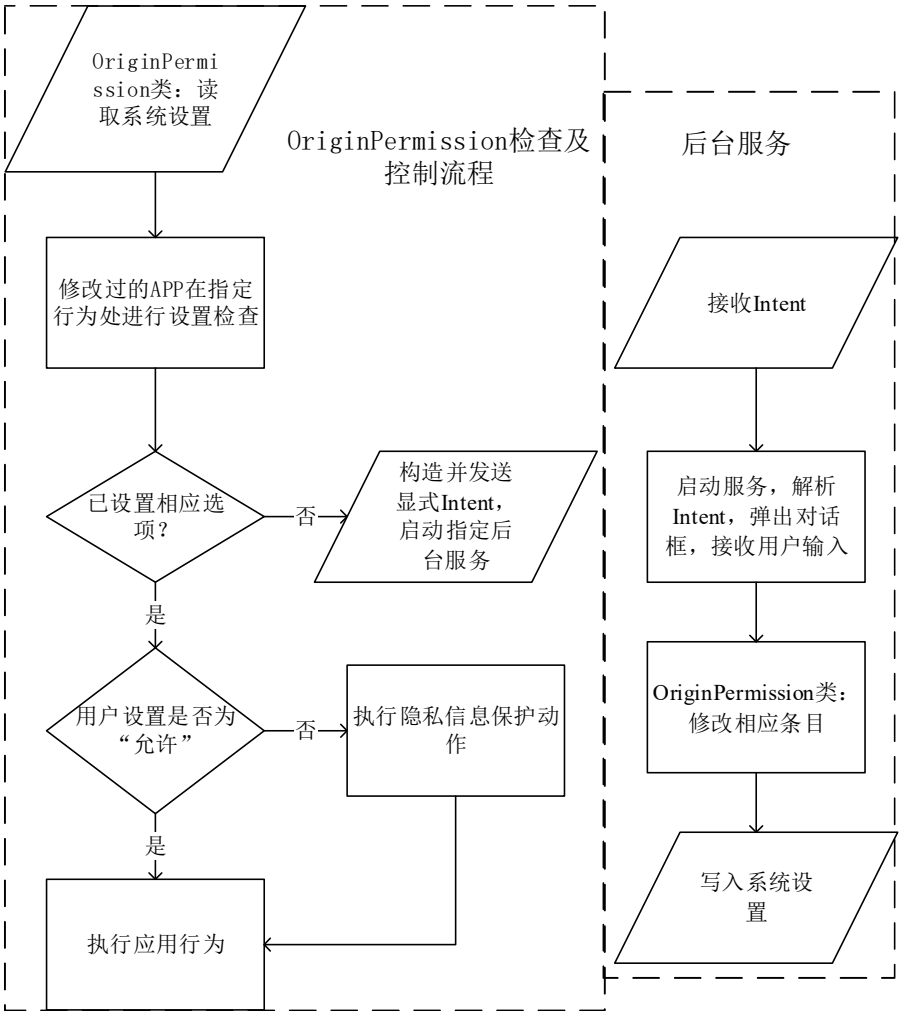

图 3-3 应用的设置检查及访问控制流程

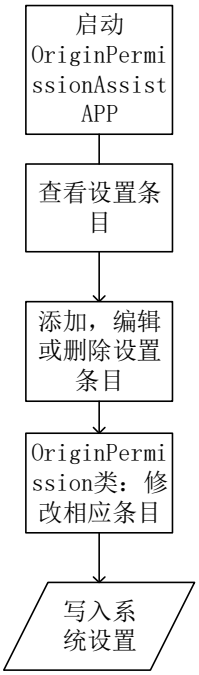

图 3-4 手动编辑相关系统设置的流程

以下对本系统各个模块的设计分别进行详细介绍。

3.1 AndroidInstrument 应用插桩工具类设计

该工具类利用 Soot 分析框架，静态分析应用中利用特定 API 访问敏感信息的代码段，并插入用户策略设置检查及隐私数据保护的相关代码段，生成修改后的应用。

该类在系统中的功能是，驱动 Soot 框架，对需要做访问控制的应用进行修改，并打包成一个新的 APK 文件。本章开头已经提到，修改应用的目的是改变应用的隐私相关行为，所以在设计 AndroidInstrument 类时，需重点考虑如何识别并改变相关行为。本节内容将对这部分做重点论述。

根据 Soot 框架的应用分析原理，本原型系统对应用行为的修改遵循以下流程：

图 3-5 AndroidInstrument 类对应用行为的修改流程

该系统针对的应用行为，“获取上次位置”和“联系人列表查询”以 API 调用的形式存在于方法的具体实现中，“位置监听”行为本身就是方法的具体实现。故对这些行为，前文所述识别及插桩流程的作用范围均为方法内部，均可通过具体实现 BodyTransformer.InternalTransform 方法执行。

在识别应用相关行为时，对于 API 调用行为，一般可根据调用者对象的类名，方法名，参数类型进行静态识别，特别地，对于需要依靠具体参数值才能确定的代码行为，如 2）中调用 query()，需要额外插入对参数值进行运行时判定的语句，以实现动态的二次确认。

在插桩设置检测代码时，插入两步即可：

i. 调用相应检测 API，存检测结果

ii. 根据结果进行跳转操作。

在插桩隐私信息保护动作代码时，应根据具体操作，生成相应的 Jimple 中间代码，插入到合适位置。

行为识别及语句插桩过程的实现，将在第四章具体论述。

此外，为使这个类能起到从系统直接驱动 Soot 的作用，需实现一个 main()方法，以便在命令行环境中调用，也将在第四章具体论述。

3.2 隐私信息修改、截断工具类的设计

在 2.2.1 小节对 Soot 框架的介绍中提到，Soot 可以以 class 文件作为输入，用于程序的分析以及代码插桩。因此，对于隐私信息的修改动作（尤其是当动作中有很多语句时），可以先行在开发环境中实现好相应的类与方法，在使用 Soot 插桩时，只需要插入方法的调用语句即可。省去了手工生成具体实现相关功能的 Jimple 代码的繁琐过程。

针对第二章 2.1 小节所述位置访问行为，可以设计一个工具类，类中包含静态方法，以当前位置，偏移量范围为参数，返回加偏之后的位置。

针对第二章 2.1 小节所述联系人相关查询行为，由于截断操作只有一条语句，为此专门设计一个工具类及方法意义不大，所以并未设计、实现相应工具类。

本节工具类的具体实现，将在第四章进行说明。

3.3 相关系统设置项的设计

3.3.1 与本设计相关的用户设置项设计

在本次设计的控制系统中，用户设置项由一个或者多个，以开发者名称为关键字的条目构成，每个开发者对应唯一条目。每个条目中包含两个选项：是否允许使用精确位置，是否允许查询联系人/通话记录列表。针对开发者的这两类行为的控制设置，均有以下三种状态：允许、不允许、未设置。

之所以引入一个“未设置”状态，是因为考虑到该系统实际应用中，可能会有用户通过后台服务弹出的对话框，对某个开发者的某个行为进行设置，该开发者条目之前在设置中不存在。在写入设置时，其他类型行为是否允许并未设置，不能默认为“允许”或“不允许”，以免给工具类的设置检查带来问题

每个条目构成一个三元组：

T=(DEV，OP1，OP2)……（式 3-1）

3.3.2 与本设计相关的设置项存储结构设计

2.2.2 节开头已经提到，本系统将在“系统设置”中存储用户的访问控制设置。由于系统中没有相关预定义条目，存取时需要用到自定义键。为了编程方便，该键统一使用同一个常量字符串。

由于本设计中用户的设置需要包含开发者名称，且要以单值存储，故采用单个格式化字符串存储设置，形成系统设置中的一个键-值对。字符串格式设计如下：

DEV1_OP|DEV2_OP|……

每个条目之间用“|”分隔。

每个条目 DEVx_OP 分别表示对一个开发者的隐私使用行为进行设置，格式如下：

DEV;OP1;OP2

“OPx”用整数表示，0,-1,-2 分别代表允许，不允许，该选项未设置。

元素之间用英文“;”号分隔。

举例：一个包含两个条目的用户设置，在 Settings 里面存储的字符串值是："com.google;0;-1|com.tencent;0;-2"，其含义如下表所示：

表 3-1 示例中设置项字符串的含义

条目	子串	含义
1 com.google;0;-1	com.google	开发者
	0	允许用指定 API 访问精确位置
	-1	不允许查询联系人列表
2 com.tencent;0;-2	com.tencent	开发者
	0	允许用指定 API 访问精确位置
	-2	是否允许查询联系人列表未设置

3.4 OriginPermission 工具类的设计

由于上文中提到的设定构成部分较多，直接操作存取有一定难度，因此在该系统的设计中，引入一个工具类，方便对相关设置进行管理。

OriginPermission 工具类需要实现的功能有：系统中相关设置的读写操作，应用行为与用户设置的对应检查，相关设置条目的管理操作。由于该类的属性值来源于系统中相关设置值，因此构造函数可设计成通过读系统设置的方式完成对象构造。

3.4.1 OriginPermission 类中设置项的存储结构

由于设置项由一个或多个条目构成，且每个开发者只有一个条目，对应唯一一组（2个）选项，因此在类内部存储时应考虑映射的问题。在类的实际应用中，根据本章开头的设计思路，应用在检查设置时会以代码开发者为关键词进行检索。

针对一个开发者名对应"一组"选项的问题，在 OriginPermission 类中定义一个静态内部类 OriginPermissionConf，用来存储及操作一组选项。按照这样设计，OriginPermission 中设置项的存储结构就是开发者（字符串）到 OriginPermissionConf 对象的映射，结合 Hashmap/ArrayMap，能方便地进行策略检查以及选项的编辑、管理操作。

3.4.2 OriginPermission 类的主要公有方法

类的构造方法：实现读取系统相关设置，存入私有属性。

设置写入方法：将相关系统调用封装到一个 write()方法以便调用。

在读/写系统设置操作中，为了和系统的 SettingsProvider 进行交互，需要重写 toString()方法，并且实现以 String 为参数的构造方法。

根据访问控制系统的设计整体要求，该类需要提供一个静态的 API，供实际应用调用，以检查代码开发者的应用行为是否被允许，应声明为 public static，并返回布尔值。系统设置的读操作在构造方法中进行。设置项的增删改查操作，也要设计对应方法。本类方法的具体设计与实现，将在第四章详细论述。

3.5 OriginPermAssist APP 设计

为了方便用户进行基于开发者的访问控制策略设置，本设计在 OriginPermission 类的基础上，进一步设计了一个 APP，提供了触摸屏上的图形界面，允许用户对相关策略进行查看，添加，修改，删除等操作。

由于 OriginPermission 类提供了相关设置的操作方法，在响应界面事件时，只需调用相应方法即可。

3.6 后台弹窗服务设计

在 Android6.0 及以上的权限模型中，当应用首次使用某个危险权限组里的权限时，系统会弹出一个对话框，询问用户是否允许应用使用该权限组。

在本次设计的访问控制系统当中，可以借鉴这一机制，在应用执行动态设置检查的代码时，如果针对某个开发者的某类行为控制尚未设置，则请求弹出一个对话框；用户点击对话框上的是/否按钮之后，OriginPermAssist APP 将相应设置写入系统 Setting。

一个可行的方案是，在 OriginPermAssist APP 中设计一个服务，用于监听来自应用的相关设置请求；根据请求内容弹出系统级对话框，浮于正在运行的应用上方。用户点击按钮后，该服务执行响应代码，更新系统相关设置，对话框关闭，用户可直接继续使用刚才的应用。

该服务需要实现的功能点有：请求内容处理，对话框显示，界面响应。该部分的具体实现，将在第四章论述。

第四章 系统实现

访问控制系统的整体实现与部署，有两种实施方案：

1） 将 OriginPermission 工具类，用于隐私信息修改/截断的工具类放入系统的 Framework 层；AndroidInstrument 类实现时，对需要修改的应用只插桩相应代码调用，而不会有 OriginPermission 类及隐私信息修改/截断工具类的具体实现。

该方案的优势：确保工具类的一致性，工具类更新只需修改 Framework 层源码，并重新编译、刷入 framework.jar 文件即可。

缺点：对于一些不易修改 Framework 层的系统（如厂家自带系统，开源组织发布的某些“正式版”（Framework 已 Odex 化）），在实验及测试环境中，这种部署方案效率并不高，需要将整个 System 分区编译再刷机。

解决方法：使用开源系统，用源码编译，选择 userdebug 版本分支。

2） 在 Soot 运行的 wjtp 阶段，继承 Soot 框架的 SceneTransformer 类，将 OriginPermission 工具类以及隐私信息修改/截断工具类的具体实现，全部插入到要修改的应用当中。

相应方法的调用插桩同方案 1。

该方案的优势：不需对系统层进行修改，在应用层即可实现访问控制功能。

缺点：不同应用中插入的工具类难以保持一致性，在开发调试阶段会引起混乱。

本系统的测试验证阶段，访问控制系统的具体实现及部署选用方案 1。采用开源 Android 项目，在测试手机上安装用修改后源码编译出的 userdebug 版本开源系统，修改 Framework 只需刷入一个 jar 包。

下面分模块介绍系统每一部分的具体实现：

4.1 OriginPermission 类实现

4.1.1 属性

在实现该类时，作为工具类并未继承 Activity，为了实现本系统相关设置的读写，需要获取调用该类方法的应用的上下文。故需要定义属性 private Context context。

在 OriginPermission 类中还需要定义一个属性，以临时储存用户在控制系统中的相关设置，用于列表展示及设置条目管理。根据第三章的设计要求，定义为一个 Hashmap<String,OriginPermossionConf>类型的私有属性。

4.1.2 方法

该类实现了如下方法：

表 4-1 OriginPermission 类实现方法列表

方法	说明
系统设置读写方法	

public OriginPermission()	类的构造方法，实现系统当前上下文读取及设置读操作，将操作系统中与本系统相关设置读入对象
public static void write(OriginPermission O)	静态方法，实现系统设置的写操作
设置检查方法	
public static boolean checkOriginPermission(String package_name, String Permission)	本类对应用提供的系统 API，用于相关设置检查。根据传入的包名提取开发者信息[表 4-1 注]，根据开发者信息及行为类型，调用 selfcheckOriginPermission 检查系统设置。
public boolean selfcheckOriginPermission(String developer, String Permission)	检查对象中用户的相应设置，返回是否允许相应行为，若用户未设置，则先启动 OriginPermAssist 的后台服务，等待用户设置后再检查。
private void promptForGrant(String developer, String Permission)	根据开发者信息及行为类型，构造 Intent 以启动后台服务
public boolean checkExists(String developer, String Permission)	检查(Developer，Permission)的组合，在系统设置中是否已设置
private boolean checkExists(String developer)	检查设置项中以 developer 为关键字的条目是否存在
条目管理方法	
public void addConf(String developer, boolean isAllowLocation, boolean isAllowContacts)	添加一个以 developer 为关键字的条目，两个设置选项都进行设置
private void addConf(String line)	将字符串转换成条目并加入对象
public void updateConf(String developer, String permission, boolean allow)	用于弹窗响应，将用户对 developer 使用某类行为的设置合并进对象。若相应 developer 条目之前不存在，则将新建条目的其他类行为是否允许均设为“未设置”
public void removeConf(String key)	删除指定 developer 的所有设置
其他对外交互方法	
public String toString()	重写 toString 方法，用于将类的属性写入系统设置
public List<HashMap<String, String>>	把对象属性转换成 OriginPermAssist 应用

getDisplayList()	主界面的显示列表
private static Application getApplicationUsingReflection()	反射调用 currentApplication 方法获取调用该类中读/写系统设置方法的应用 context，用于读写数据。

[表 4-1 注]根据包名提取开发者信息的具体实现如下：一般地，从包名提取形如“a.b”的前缀，直至第 2 个点号。例外情况：若包名为“android.*”，则直接提取“android”，若包名形如“国家代码.co.abc.xxx”，则提取到第 3 个点号才截止。

4.1.3 静态内部类 OriginPermissionConf 的实现

OriginPermissionConf 类对单个条目内的选项进行管理，每一个对象管理一个完整条目中与开发者对应的部分，即用户对该开发者不同类别行为做出的控制设置。

为方便在条目内部按行为类别查询，用 Hashmap 建立了从行为类别（字符串）到设置值（布尔值）的映射，作为类的私有成员。在目前已实现的系统中，该成员行为类别的集合为{“LOCATION”,”CONTACTS”}，分别代表本系统所识别并控制的位置相关行为，以及联系人相关行为。设置值的集合为{true,false}，分别代表允许，不允许。未设置的行为类型不存入 HashMap。

该类实现的方法如下表所示：

表 4-2 OriginPermissionConf 类实现方法列表

方法	说明
OriginPermissionConf(boolean allowLocation, boolean allowContacts)	重载构造方法，用于从相应布尔值构造该类的对象
OriginPermissionConf(String allowLocation, String allowContacts)	重载构造方法，用于将系统设置中单个条目中的设置值字符串转换成该类对象
private boolean convert(String setting)	将已设置的设置值从字符串形式转换为布尔值（允许/不允许），用于构造对象
boolean isSet(String Permission)	检查该条目某类型行为控制选项是否已设置
boolean check(String Permission)	检查某类型行为是否被允许
public String toString()	将该条目的设置值转换为字符串，用于系统设置写入
private String getDescription()	获取该条目设置对应的描述字符串

4.2 隐私信息修改方法实现

getFuzzedLocation()方法实现：直接用 Java 编写一个静态方法，传入参数分别为当前位置，偏移距离下界 lower，偏移距离上界 upper，利用几何方法计算经度、纬度的偏移值，返回与当前位置距离在 lower 到 upper 之间的一个随机位置(Location 类型)对象。

以上所述位置信息加偏功能，也可以通过编写 Java 代码，依次手动生成 Jimple 语句并插桩的方法实现，由于过于繁琐，且系统中实际未采用，此处略去相关内容，具体代码请参考附录。

截断联系人的方法未实现，以调用语句生成的形式写进了 AndroidInstrument 类，具体的调用语句见本文 4.3.2 小节末尾。

4.3 AndroidInstrument 类实现

该类需要实现的功能有：从系统命令行启动，驱动 Soot 框架运行；导入 Android 应用，生成并插入用于改变应用行为的中间变量和相应方法调用的 Jimple 代码；将转换过的程序回编译，打包到新的 APK 文件。

4.3.1 main()方法实现

main()方法的工作流程如下图所示：

图 4-1 AndroidInstrument 类 main()方法流程图

1）程序初始化：在本系统中，由于需要处理 Android 应用，程序初始化阶段需要设置相应的输入、输出格式，同时指定一个存放各个版本 Android.jar 文件的目录，以便在应用分析及插桩时使用相应的系统 API。

此外，在检查用户设置及修改/截断隐私信息时，需调用自定义类 OriginPermission，LocationFuzzer 等的方法，为此还需要把这些类的 class 文件放到某个子目录下（在本系统是 AndroidInstrument 工程的 bin 目录），在初始化过程中指定该目录，并将这些类作为库类/应用类加到应用中（相当于 Java 开发中的 import）。

2）加插自定义 Transform：实现的重点是 Transformer 类的继承及 InternalTransform()方法的具体实现。为继承 Transformer 的子类 BodyTransformer，在 main()函数中 Transform 的构造时使用了匿名类，并具体实现了 InternalTransform()方法。

3）启动 Soot 框架，启动时将命令行的输入参数传入 Soot 框架。

4.3.2 行为识别及应用插桩的具体实现

1） 识别应用行为的具体实现：

通过识别关键语句及方法实现，以下对本系统选取识别的 3 种行为分别具体说明：

a) LocationListener 的位置监听行为，在应用中通过具体实现 onLocationChanged 方法得以实现，按以下方法进行识别：

利用 InternalTransform()遍历所有方法，对于传入的参数（输入应用的方法体），获取方法名，方法参数及其类型，若方法名为”onLocationChanged”且有唯一参数，则认为该方法实现了相应行为。

b) 获取上一次已知位置的行为，在应用中通过调用 getLastKnownLocation()方法，获取返回值实现。

在 Jimple 中间代码中，一个函数调用的返回值，如果被调用语句（或表达式）后面的语句使用，则在使用前必须赋给一个局部变量，否则只是调用了相应函数。对应的 Jimple 赋值语句如下：

AS=Assign(Local,invokeExpr)……（式 4-1）

因此可以根据 Jimple 赋值语句的操作数 2，即调用表达式进行判断。

具体方法：在 InternalTransform 中，对方法体的每个语句，实现方法 caseAssignStmt，遍历方法中所有的赋值语句，检查是否存在调用表达式，若存在，则检查调用语句方法名。如果方法名为 getLastKnownLocation，则认为该语句执行了相应行为。

c) 利用 query()方法[9]，查询用户设备的联系人/通话记录列表行为

通过向方法传入具备以下 Authority 的 Uri 类型参数实现：

表 4-3 Authority 与查询行为对应表（部分）

Authority	行为
contacts	查询联系人列表
com.android.contacts	查询联系人列表
call_log	查询通话记录列表

初步识别方法与 b）的行为类似，静态检查赋值语句中的调用表达式，若方法名为”query”且方法所在类名为” android.content.ContentResolver”，则为查询行为。但该行为是否用来查询联系人信息还无法确定，需要对调用表达式的第一个参数进行运行时检查：参数是 android.net.Uri 类型，运行时调用自身的 getAuthority()方法，若返回字符串满足预设条件，则认为该语句执行了相应行为。

2） 代码插桩的具体实现

整体思路：在 InternalTransform()方法中编写 Java 代码，生成相应的 Jimple 语句，根据行为识别结果找到插入位点，然后按顺序插入相应语句。

预备知识：

用户相关策略设置检查的伪代码如下：

```
if(!checkOriginPermission(package,behavior)){
```

```
    //do something
}
//do other things
```

按照条件跳转逻辑，重写如下：

```
if(checkOriginPermission(package,behavior)){
    goto unit;
}
//do something
unit:
//do other things
```

其中加粗的代码段构成一个条件跳转语句，在 Jimple 中间代码生成时，用“跳转条件”和“跳转目的语句”即可构造。

其中，跳转条件用一个等号表达式表示，“var==true”对应“eq(var,1)”

Jimple 方法调用语句与方法调用表达式的生成，均遵从如下步骤：

a） 定位需要调用的方法：在 Scene 里面获取方法所在的类，再通过方法名及参数类型，找到具体方法。

b） 确定调用语句/表达式类型：JVM 的调用命令，和 Jimple 中的调用方式是一一对应关系。根据 Jimple 调用方式可直接确定语句/表达式类型。

JVM 常用的调用命令有：

invokevirtual，大多数公有非静态方法调用

invokestatic，静态方法调用

invokespecial，私有/构造/父类方法调用

invokeinterface，接口方法调用

根据调用方法的可见性，是否静态，是否为父类方法等，选取合适的调用方式。

c）逐个确定参数（如果有参数）：根据需要传入的参数，定义需要的常量，局部变量，可根据需要，插桩一些辅助的赋值语句。

d）语句/表达式生成：以方法的 SootMethodRef，及方法调用传入的参数（如果有）为参数生成相应 Jimple 语句或表达式。

在插桩语句时，应避免如下错误：在条件跳转的目的语句之前插入，可能会遇到如下的语句和标号的对应问题。

调用语句插桩前，程序的伪代码表示：

```
 if(checkOriginPermission(package,behavior)){
    goto unit;
 }
 unit:
```

```
//do other things
```

错误插桩的伪代码表示：

```
if(checkOriginPermission(package,behavior)){
  goto unit;
}
unit:
location=getFuzzedLocation(location);//这里 unit 对应关系有误
//do other things
```

正确插桩的伪代码表示：

```
if(checkOriginPermission(package,behavior)){
  goto unit;
}
location=getFuzzedLocation(location); //方法调用并赋值
unit:
//do other things
```

为避免此类错误，在跳转语句的目的语句之前插入语句时，应调用 InsertBeforeNoRedirect 方法，以保持标号与语句的对应关系。

插桩过程实现：

“用户设置检查”语句构造：构造一个 Jimple 赋值语句，在该语句中调用 checkOriginPermission 方法。

由于本系统对应用行为的检查采用基于开发者信息的检查方式，checkOriginPermission 需要以行为代码所在包名和行为类型作为参数。在 Soot 框架中可利用类属性获取包名；对于识别出的行为，其类型是确定的。所以插桩时可将获取到的值作为常量插入到调用表达式的参数列表。

构造一个 Jimple 条件跳转语句：如果 checkOriginPermission 返回 true，跳到 unit。

具体到前文所述 3 种应用行为，插桩过程如下：

a）位置监听响应行为：由于行为通过重写整个方法实现，方法首个参数即为获取到的精确位置，故插桩思路如下：在第一个执行语句之前插入设置检查及位置模糊化处理相关语句。

基准位点选取：找到该方法第一个执行语句，作为语句插桩的位置参考。

先进行“用户设置检查”语句插桩：生成上文伪代码所示条件跳转语句，将跳转目标设为基准位置（当用户允许时跳过位置信息修改步骤），插入基准位置之前。

后进行“位置信息修改”语句插桩：在基准位点之前，插桩一个调用位置信

息加偏方法，指定加偏距离下界，上界作为调用参数，并将返回值赋值给参数的语句。

被插桩的部分，伪代码如下面粗体部分所示：

if(checkOriginPermission(package,behavior)){ //条件跳转语句

goto unit;

}

location=getFuzzedLocation(location,lower,upper); //方法调用并赋值的语句

unit:

//method body originally starts here

b）上次精确位置获取行为：需要插桩的语句及插桩顺序类似，与 a）主要不同之处在于插入位置，需要在行为执行语句之后插入对行为结果进行修改的代码。

按顺序构造待插桩语句，生成一个有序列表，再将该列表插到基准之后。条件跳转语句的跳转目的地也要相应修改。

被插桩的部分，伪代码如下面粗体部分所示：

unit:

location=getLastKnownLocation(location);//初始调用语句，插桩基准

if(checkOriginPermission(package,behavior)){//条件跳转语句

goto unit2;

}

location=getFuzzedLocation(location,lower,upper); //方法调用并赋值的语句

unit2:

//do other things

c）联系人/通话记录列表查询行为：在识别出的 query()调用语句之前依次插桩动态识别行为的语句，然后再插桩查询截断语句。被插入的语句实现逻辑：识别 query()方法传入的参数值，对行为进行动态确认，若不是相关行为，跳转到该调用语句，否则先执行截断语句。

被插桩的部分，伪代码如下面粗体部分所示：

Uri uri;//传入参数

String authority=uri.getAuthority();//动态确认

if(!(authority.equals(“contacts”)||authority.equals(“com.android.contacts”)||

authority.equals(“call_log”)))//不是相关查询行为

{

goto unit;

```
}
uri=Uri.parse("");//查询截断
unit:
query(uri,…);//其他参数略
```

4.4 OriginPermAssist APP 实现

为了方便普通用户查看，管理与本访问控制系统相关的设置，本系统依托 OriginPermission 类，实现了一个以修改相关系统设置为主要功能的系统 APP，为用户提供了可操作的界面，并且在其上实现了一个后台服务，用于接收来自 OriginPermission 类的设置请求，弹出对话框供用户选择。

该 APP 包括以下组件：

MainActivity：主界面，实现了设置查看，修改，添加和删除功能，用户打开 OriginPermAssist APP，进入主界面即可看到条目列表，直观地进行操作。

PromptDialog 界面 Layout：主界面弹出的自定义对话框，用于设置条目的编写，以及设置项的勾选（允许/禁止）。

OriginPermPromptOnReqService：后台服务，用于接收来自 OriginPermission 类的设置请求，根据请求内容，在当前应用上方弹出系统级对话框，询问用户是否允许某开发者执行特定行为，并根据用户选择，写入相关设置。由于后台服务的启动不依赖于应用的启动，用户无需专门打开此应用，就能在运行的被插桩应用界面上方看到弹出的请求设置对话框。

下面分组件介绍该 APP 的实现，以及 OriginPermission 类具体如何与此 APP 的后台服务互动。

4.4.1 MainActivity 实现

MainActivity 的界面如下图所示：

界面元素为一个显示列表和一个“添加”按钮。用户在该界面上可完成如下功能：

1） 查看设置条目：列表将设置项逐条显示，每条第一行是开发者，第二行是用户对不同类行为的设置情况，如图 4-2 左半部所示。

2） 修改设置条目：用户单击条目，会弹出一个对话框，具体显示该条目的内容，用户可以选择是否允许该开发者访问指定敏感信息。

3） 新增设置条目：用户单击界面右下角的“+”号按钮（如图 4-2 左），弹出对话框，允许用户输入 developer 信息，及是否允许该开发者访问指定敏感信息。

在用户输入 developer 时，对用户输入格式进行检查。若不符合格式要求，则拒绝设置并弹出相应提示。

格式要求：不能为空字符串，禁止包含’’，’;’，’.’，’|’这 4 种分隔符，格式必须为形如” a.b”或”国家代码.co.***”的形式（字符串”android”除外）。

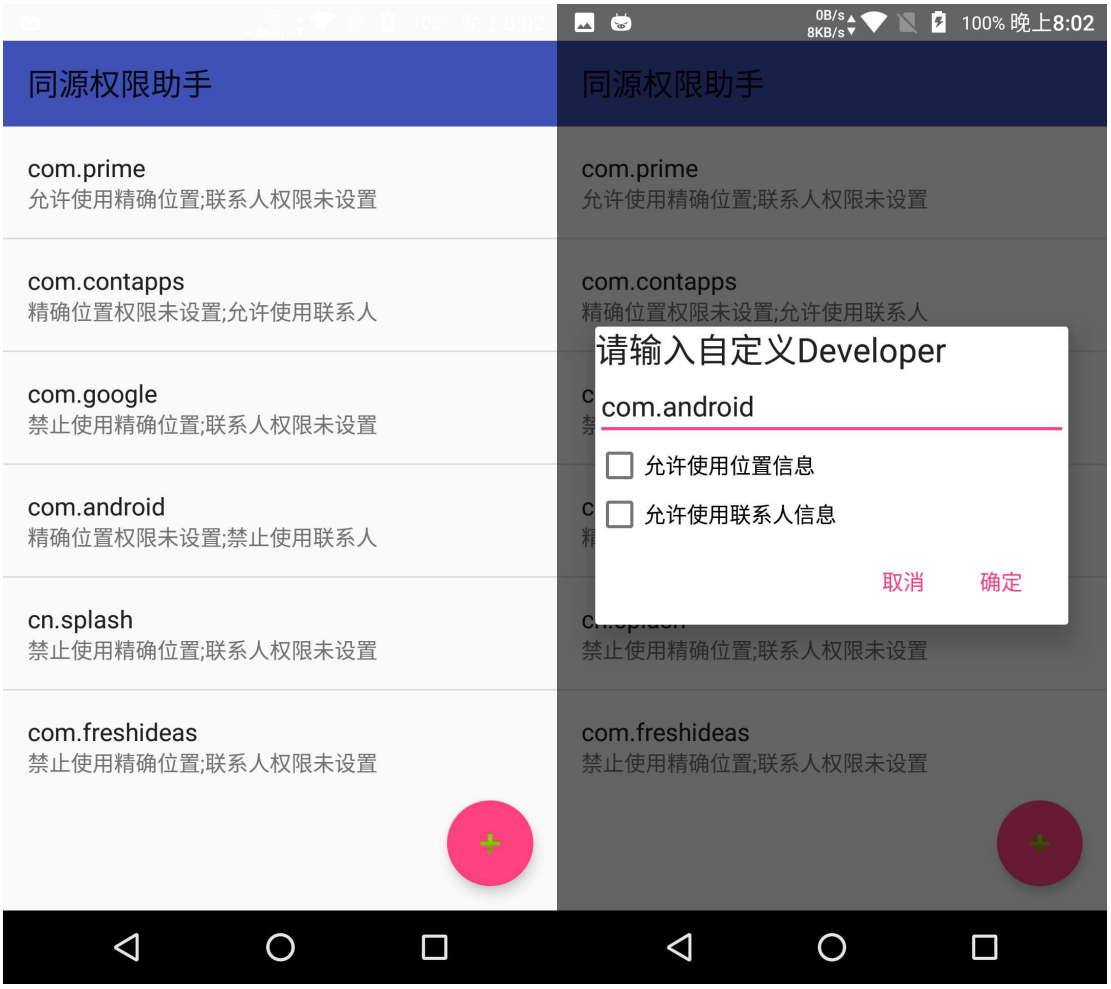

图 4-2 应用主界面及条目添加（编辑）界面

4） 删除条目：用户长按某条目，弹出对话框，用户确认是否删除，如下图所示。

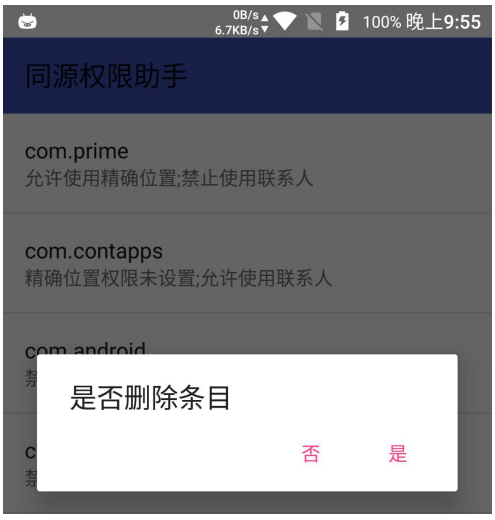

图 4-3 条目删除界面

响应界面事件以修改相关设置的实现，是靠调用 OriginPermission 的构造函数，读取系统设置，建立 OriginPermission 对象，根据界面事件，在对象上进行相应操作，最后调用相应方法，将设置写入系统

4.4.2 PromptDialog 界面实现

界面元素有一个输入框，两个勾选框，以及确定/取消按钮，如图 4-2 右所示。

该界面通过 MainActivity 的 newConfDialog 方法进行构造，传入参数为设置条目的内容（修改条目）或空值（新建条目）。界面事件响应通过重写 onClick 实现，在点击“确定”时执行如下动作：

检查 Developer 输入格式，若不符合要求，拒绝设置；若符合要求，构造一个新条目，将其与系统已有相关设置合并，并将新的设置内容写入系统设置。

4.4.3 OriginPermPromptOnReqService 后台服务实现

该服务主要实现了一个请求响应方法，该方法弹出一个系统级对话框，覆盖于正在运行的应用之上，内容形如“某开发者想要执行某种行为，是否允许？”，向用户请求设置，用户可以做出选择，之后该服务将用户的选择加入现有设置项，并写入系统设置。

弹窗的界面设计及显示效果如下图所示（背景为经过插桩，正在运行的应用）：

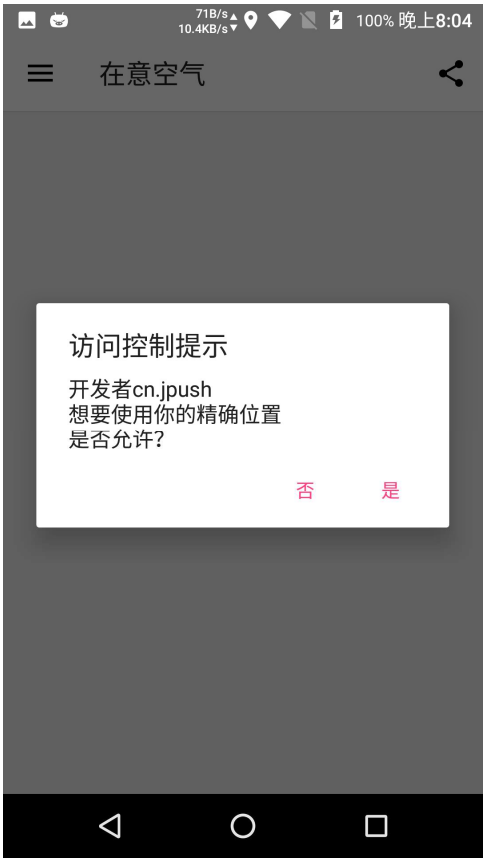

图 4-4 后台服务弹窗界面

后台弹窗请求用户设置的过程，其实质是 OriginPermission 类，调用 OriginPermission 类的应用和 OriginPermPromptOnReqService 服务的交互过程，实现如下：

在系统设计中，OriginPermission 类被应用用来进行设置检查时，通过构造显式 Intent，指定服务的包名、类名，由调用 OriginPermission 类的应用上下文（即 OriginPermission 类的 context 属性）请求启动该服务。服务启动时需要应用行为的开发者及行为信息，用于构造对话框及系统设置条目，故 OriginPermission 类在构造 Intent 时应将相应字符

串加进去。Intent 的数据格式定义为”originperm:开发者;行为类别”。服务接收到 Intent 即启动，在请求响应方法中解析 Intent 数据，构造并弹出对话框。

以上交互过程的流程图，参见第三章系统流程图第 2 张（图 3-2）。

第五章 系统部署及测试

5.1 系统部署

5.1.1 运行环境

本系统的运行环境分为 PC 端环境（用于运行 AndroidInstrument 类）及移动端环境（用于运行被插桩应用、辅助应用及服务、其他工具类）

移动端运行环境：

硬件配置：HTC A9w 手机，骁龙 617 8*1.5GHz+2GB+16GB。

软件配置：Mokee 开源项目针对 HTC A9 系列适配的 Android 6.0.1 版本操作系统，userdebug 分支，使用最新版本源码编译并刷机。

PC 端运行环境：PC 机，Intel i7-4710MQ 4*2.5GHz 超线程+16GB DDR3，软件环境 Windows+JDK+Cygwin+Android Studio。

5.1.2 系统分模块部署

重点是移动端的部署。第四章已经提到，为了保持移动端工具类一致性，选择通过修改源码的方式，在 Android 系统 Framework 层部署工具类。在源码开发环境下，将 OriginPermission，LocationFuzzer 源码放到 Framework 源码目录，修改相应的 Makefile 文件，重新编译 Framework.jar 并在 Root 模式下将该 jar 包刷入系统，完成这两个模块的部署。

OriginPermAssist 及服务：由于 OriginPermAssist 需要写入 Settings.System 的自定义条目，需要系统权限，因此在 Android Studio 生成相应 APK 后，还需要在源码环境下，利用开源系统的 platform.pk8，platform.x509.pem 文件对 APK 重签名，使其具有与其他系统应用相同的签名，然后用 ADB(Root)安装到系统目录。

插桩之后的应用：直接安装。

PC 端部署 AndroidInstrument 类相对简单，只需要下载 Soot 运行所依赖的 jar 文件，编译 AndroidInstrument 类。编写命令行脚本就可以启动 Soot 框架。

5.2 系统测试

测试阶段分两部分进行：第一部分验证具体的隐私信息处理措施，及应用内的细粒度访问控制是否起到了作用，第二部分验证以开发者为基准的系统设计会对访问控制过程及结果起到什么作用。

在测试全过程中，为避免 Android 既有访问控制机制对测试结果的影响，对于插桩处理后的应用，启动前在“权限设置”里将其“位置”、“联系人”访问全部设置为“允许”。

在部署相应模块的开源 Android 系统上，对应用进行插桩，测试，初步验证了：

1）通过对敏感 API 的判定，代码插桩，可以改变应用行为，实现访问控制。

2）通过用户策略的设置与运行时检查，可以根据敏感 API 调用代码的开发者，选择性进行访问控制，改变应用的相关行为。

5.2.1 细粒度访问控制功能测试（隐私信息修改/截断）

对位置信息修改的测试，以应用“com.prime.studio.apps.route.finder.map”（位置信息相关应用）为例进行说明。

利用 AndroidInstrument 类插桩该应用，在插桩时设置位置信息偏移距离范围 2km<dr<5km，将插桩后的应用安装到设备。

清空本系统相关设置条目，启动应用，选择”mylocation”选项，屏幕上会出现 Google 地图，图上会显示通过设备硬件（在测试环境下是通过 Wi-Fi 网络）的定位结果，以及该 APP 根据该位置进行的标注。如果应用是未修改版本，二者应该是同一位置。

由于系统中未设置任何条目，在已修改应用运行中，先后弹出了两个窗口，要求用户对”com.google”（第三方开发者），”com.prime”（应用核心开发者）的位置相关行为进行设置。

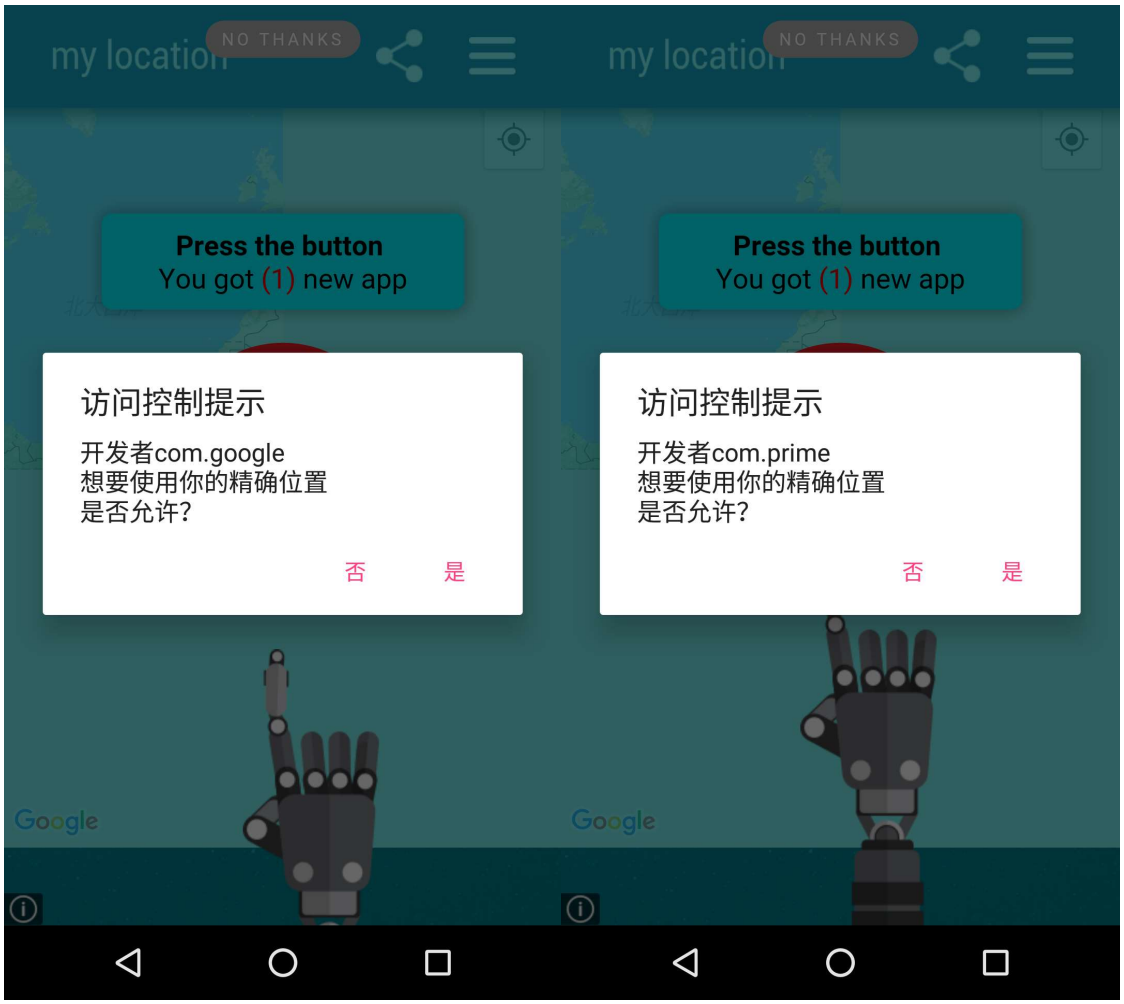

图 5-1 弹窗请求设置界面

通过对该应用的原始版本进行反编译分析得知，地图标注过程中，设备位置信息的流动如下图所示：

图 5-2 地图标注 Activity 的位置数据流图

由以上分析可知，本应用中 com.prime 的地图标注点位置，是通过监听 com.google 定位库的定位结果间接获得的。故在测试时，为方便对比，在实验组中以上两个开发者的 LOCATION 设置项有且只有一个设为“不允许”，以体现位置信息修改对应用行为的影响。

修改针对两个开发者的设置项，观察标注点情况，结果记录如下表：

表中的“偏移量符合预期”，是指 2km<标注点相对于设备位置的偏移距离<5km。

表 5-1 位置信息修改及细粒度控制测试结果

测试组别	行为类别	com.google	com.prime	现象
实验组 1	LOCATION	允许	不允许	标注点与设备定位点不重合，偏移量符合预期
实验组 2		不允许	允许	标注点与设备定位点不重合，偏移量符合预期
对照组 1		允许	允许	标注点与设备定位点重合
对照组 2		不允许	不允许	标注点与设备定位点不重合，有些标注点偏移量超出预期。

为与细粒度控制的实验组 1、2 相比，对照组 2 将地图标记过程所有涉及位置监听的开发者都设置为“不允许”监听精确位置，即进行较粗粒度的访问控制。标记结果偏移量可能超出预期（≥5km 或≤2km），结合应用反编译分析结果，推测可能原因是在地图标记过程的 2 个监听行为，获取到的位置信息均被加偏，且由于图 5-2 所示位置信息传递过程，两次偏移量叠加于最后的结果，导致总的偏移量不在预期范围。

由以上测试可知，通过比较实验组 1、2 和对照组 1（无控制），系统中实现的“位置修改”动作，对该应用中监听器监听到的位置信息进行了修改。并且借由对不同开发者的代码行为采用不同的访问控制设置。通过比较实验组 1、2 与对照组 2，结合反编译分析结果，初步验证了该系统可以对应用内部不同开发者的代码行为分别控制，即实现

应用内部的细粒度控制。

对 com.freshideas.airindex，com.tweakersoft.aroundme 等应用进行测试，也验证了对位置信息的修改。

对联系人信息截断的测试，以应用“com.contapps.android”（联系人相关应用）为例进行说明。

利用 AndroidInstrument 类插桩该应用，并安装到设备。由于该应用在启动时会读取联系人列表，并显示在主界面，故测试的预设条件为：系统中已有至少一个联系人条目。进行 2 次测试，测试条件及结果如下表：

表 5-2 联系人信息截断测试结果

测试次数	行为类别	com.contapps	现象
1	CONTACTS	不允许	启动时不显示联系人
2		允许	启动时显示联系人

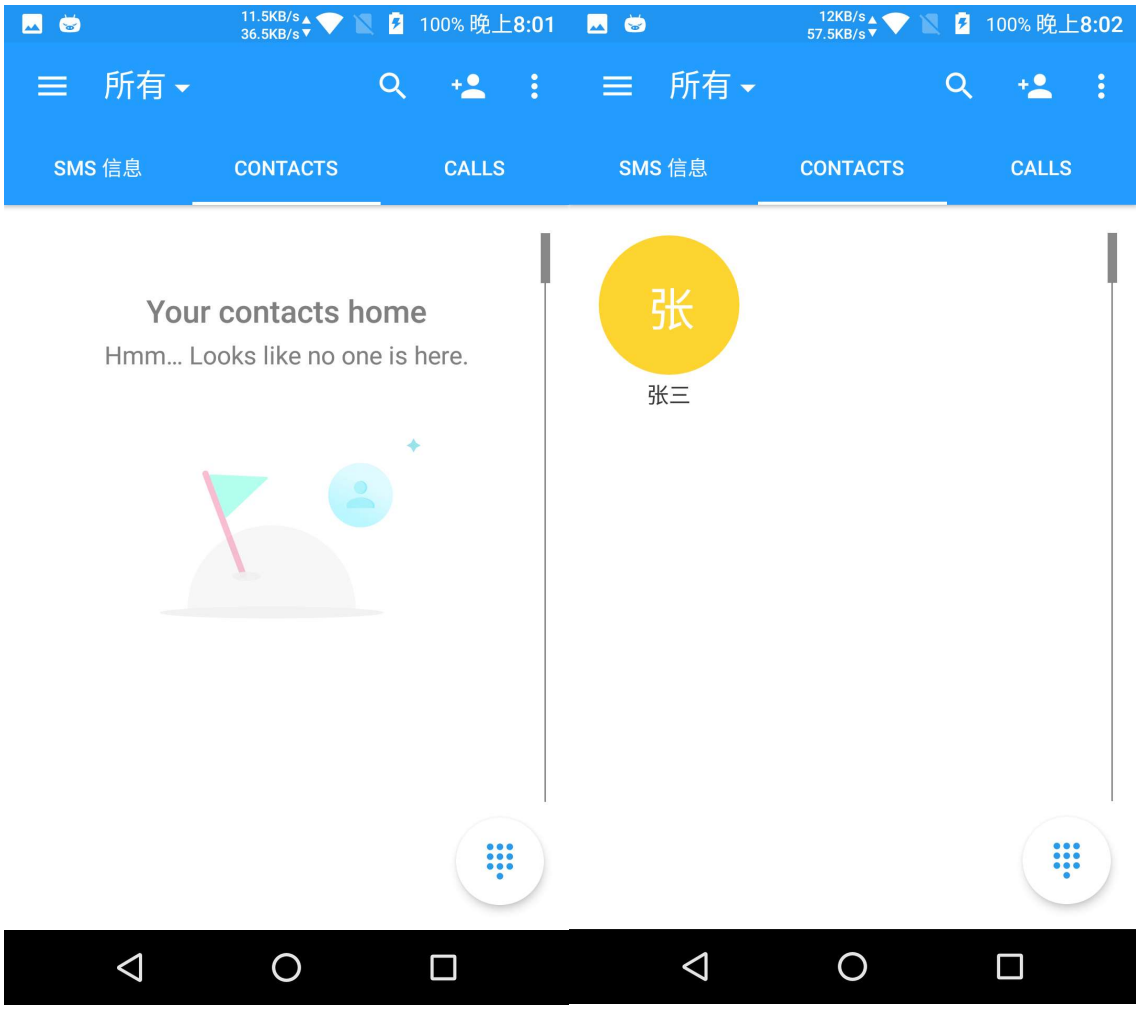

图 5-3 联系人信息截断测试截图（左为测试 1，右为测试 2）

由此可知，插桩的截断代码阻止了 APP 通过调用查询函数获取联系人。

在对部分应用进行同样测试时，出现了应用崩溃的情况，经过反编译原始应用，及

查阅相关文献[4]，分析可能是由于原始应用未处理空值异常造成的。

5.2.2 访问控制验证——跨应用测试

由于不同应用可能会使用相同开发者的第三方库执行相似行为，且本系统依据代码行为的开发者（即域），因此对于同一开发者的同一类行为来说，在设计上，只要首次执行时用户选择允许/禁止，其后该开发者再执行同样行为时，即便在其他应用中也会被相应允许/禁止，这是该系统与 Bin Liu 等设计的控制系统的最主要区别。下面针对这一点进行测试，以验证该系统对同一开发者同类敏感行为的访问控制。

现有两个应用，“com.developersnote.whatsaroundme”（下文简称“whatsaroundme”）以及“com.prime.studio.apps.route.finder.map”（下文简称“GPS Route Finder”），都是位置信息相关应用，在定位时都需要用到“com.google”开发的定位库。测试时，先清空系统中相关设置，将两个应用分别进行插桩，插桩时位置偏差距离范围均设置为 2km<d<5km，并安装到设备，将它们的位置访问权限开启，在 OriginPermAssist 设置中分别添加“允许”两个应用核心的开发者（“com.prime”和“com.developersnote”）使用位置相关敏感行为的设置。

此时启动“whatsaroundme”应用，系统中后台服务弹出窗口，询问是否允许 com.google 访问精确位置。用户选择信任 com.google 的位置使用行为，点击了“是”。从运行结果看，该应用正确地返回了设备位置周边的公交车站等设施信息。

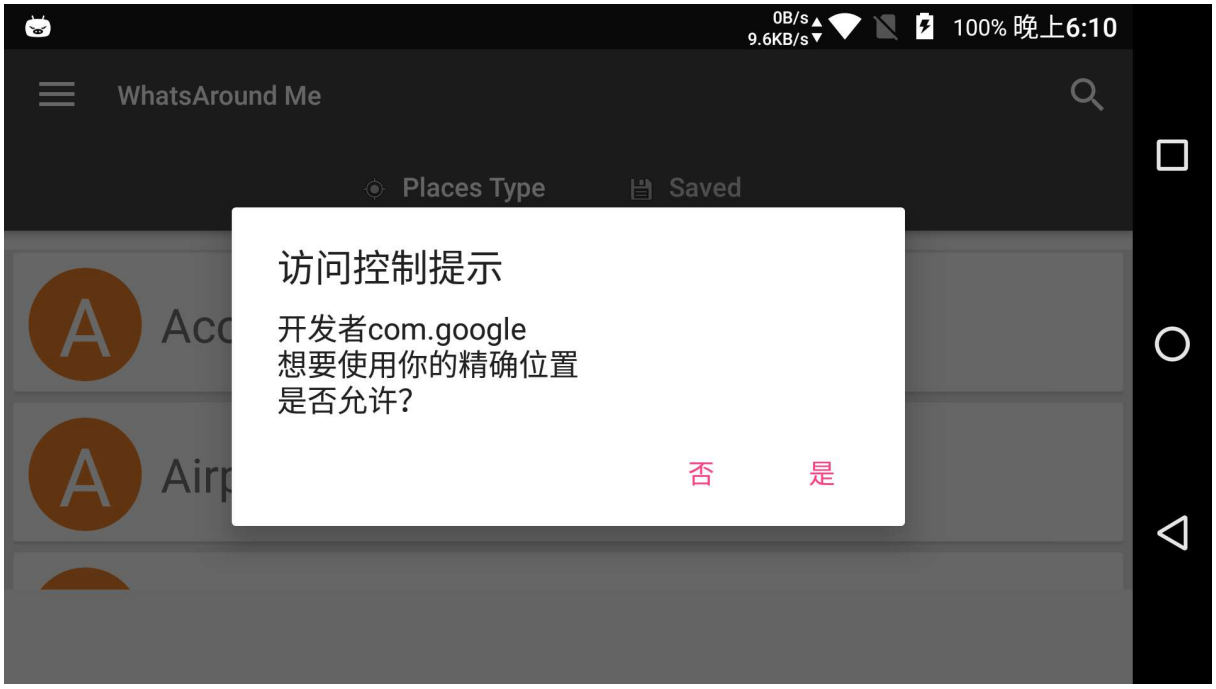

图 5-4 后台服务向用户请求设置

图 5-4 周边公交车站检索结果

关闭该应用，再启动“GPS Route Finder”，选择“mylocation”选项，可以看到，该应用的地图标注功能在地图上正确标注了设备位置，说明通过检测系统中已存在的用户相关设置，该应用中 com.google 的定位库使用了准确的位置信息。

图 5-5 正确的地图标注结果

回到“OriginPermAssist”，与用户的初始设置相反，此时将“com.google”设置为“不允许”位置相关行为，在两款应用中均观察到了标注或搜索偏差。

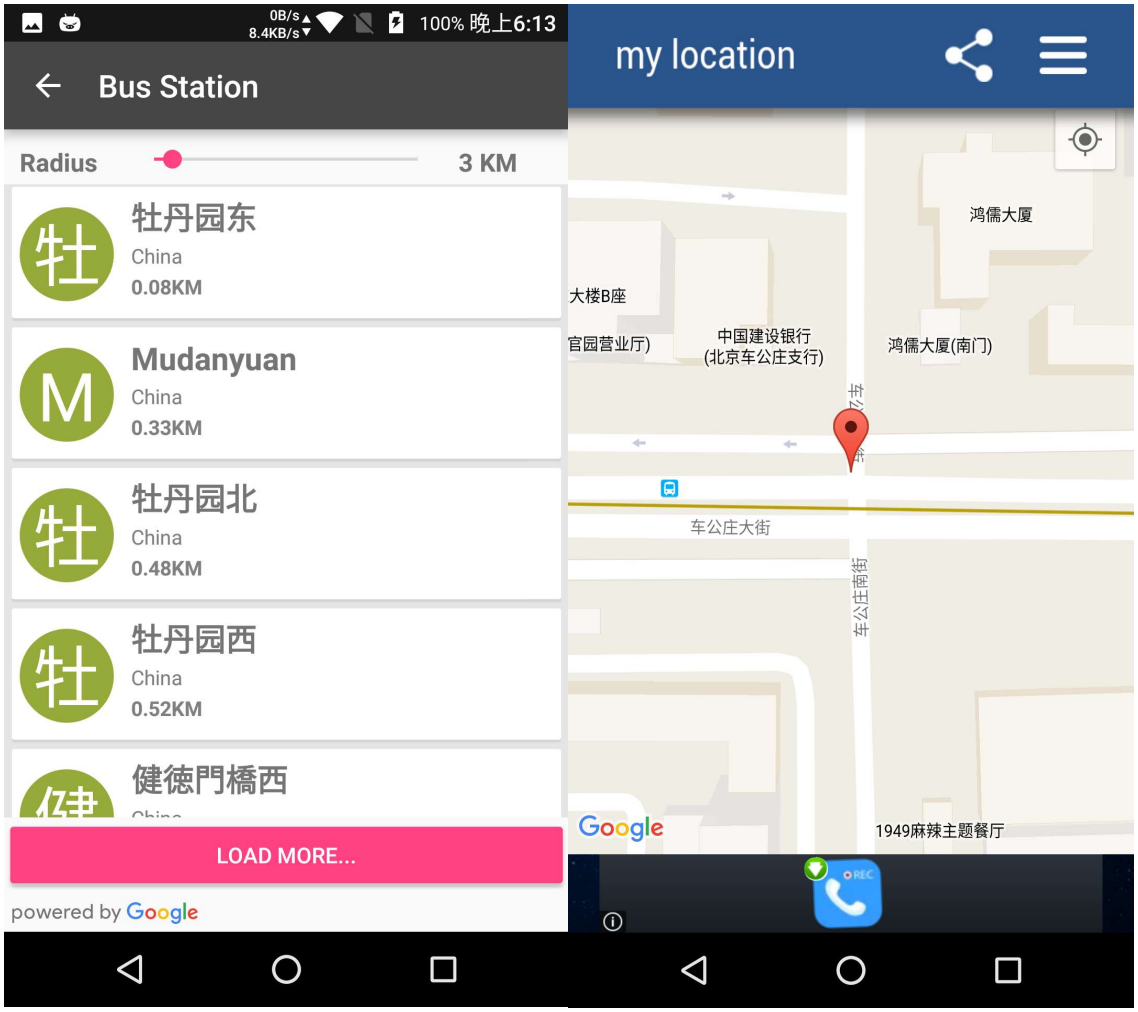

图 5-6 偏差的搜索结果（左）和地图标注结果（右）

由此可见，在本访问控制系统中，基于开发者的访问控制机制，可以使用户以同一设置项控制不同应用中同一开发者的一类访问行为，信任粒度为开发者。在 2.1 节已经提到，这种控制机制的应用意义在于，当多个应用中包含同一开发者的第三方库（如广告库）时，用户可以用单次设置来控制这些第三方库的敏感行为，免于对每个应用单独设置。事实上，很多第三方库都会被多种应用使用，如前文提到的 Google 定位库，以及 Mopub，jPush 等广告库。基于开发者的访问控制设置，在第三方库访问控制当中，能起到打破应用边界，扩展控制作用范围的作用。

第六章 总结

本次设计达成的研究成果：借鉴同源策略，提出并设计了一个以开发者为基准的 Android 应用细粒度访问控制原型系统，并在其上实现了用户设置检查与编辑，以及敏感信息的修改与截断。将系统部署到设备，进行初步测试，验证了 1）系统可在应用内部，针对不同开发者的特定代码行为进行细粒度访问控制。2）系统可针对同一开发者在不同应用中的行为，以同一规则进行控制，用户无需分应用设置。初步验证了该系统功能的可行性。

限于时间、个人水平等因素，本次研究仍存在许多不足，笔者认为最大的两点不足是：应用敏感行为分析有待完善，目前只分析了 2 类 3 种行为，尚未能完全覆盖涉及这两类敏感信息的所有应用行为；在行为识别过程中，仅仅是靠手工编写条件，对中间代码进行的判断。此外，对插桩过程及被插入代码的性能开销未进行分析，针对部分应用自身的重签名措施未进行处理，对 Target API 版本≥24 的应用，由于 dex 文件版本兼容问题，限于测试的软硬件平台因素，未进行测试。该系统在设计及实现上还有许多不成熟之处，会在将来的研究工作中予以完善。

参考文献

[1] 系统权限|Android Developers, https://developer.android.com/guide/topics/security/permissions.html

[2] 王浩宇, 郭耀, 马子昂等.一种大规模的移动应用第三方库自动检测和分类方法.软件学报.28(6). 2017. http://www.jos.org.cn/1000-9825/5221.htm

[3] Ryan Stevens, Clint Gibler, Jon Crussell 等. Investigating user privacy in Android ad libraries[C]. Workshop on Mobile Security Technologies (MoST). 2012

[4] Bin Liu, Hongxia Jin, Ramesh Govindan. Efficient Privilege De-Escalation for Ad Libraries in Mobile Apps. MobiSys'15, May 19–22, 2015, Florence, Italy.

[5]Raja Vallee-Rai, Laurie J. Hendren. Jimple: Simplifying Java Bytecode for Analyses and Transformations. 1998.

[6] Steven Arzt, Siegfried Rasthofer, Eric Bodden. The Soot-based Toolchain For Analyzing Android Apps. MOBILESoft'17, Buenos Aires, Argentinia

[7] Alexandre Bartel, Jacques Klein, Yves Le Traon, and Martin Monperrus. Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot. In Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program Analysis (SOAP '12). ACM, New York, NY, USA. 2012: 27–38.

[8] Prof. Eric Bodden, Ph.D. Packs and phases in Soot. http://www.bodden.de/2008/11/26/soot-packs/

[9] ContentProvider |Android Developers, https://developer.android.com/reference/android/content/ContentProvider.html

[10] Application security | Android Open Source Project, https://source.android.com/security/overview/app-security

[11] Alexandra Burlacu. Android 6.0 Marshmallow: A Closer Look At App Permissions. Tech Times. 2015.11. http://www.techtimes.com/articles/102049/20151103/android-6-0-marshmallow-a-closer-look-at-app-permissions.htm

致　谢

向此次毕设期间帮助过我的所有老师、同学表示衷心感谢！

特别需要感谢的是本次毕设的指导老师，王浩宇老师，以及计算机学院体系结构中心的杨旭东老师（我读研期间的导师）。在毕设的选题阶段，杨老师在选题的方向上给了我一些建议，让我去找王老师面谈，了解移动应用安全及隐私相关研究领域。经过一段时间的了解，我选择了王老师的这个毕设题目。在毕设的准备及实施阶段，王老师向我介绍了 Android 应用反编译工具的使用，以及系统整体的实现思路，每周督促我按进度完成任务。此外还要感谢中期答辩的负责老师，他对我的毕设在系统设计和实现上提了一些建议。在论文写作及毕设答辩的准备阶段，杨老师、王老师分别对我的论文进行了检查，并提出了修改意见。总之感谢几位老师，特别是王老师、杨老师的指导，使我能顺利完成本次毕设任务。

此外还要感谢王老师、杨老师每周组织小组会议，讨论移动应用隐私相关研究问题，感谢实验室及小组里从事相关领域研究的学长学姐与我交流，以及李元春学长在应用修改方面对我的指导。也要感谢毕设期间与我一同讨论技术问题的同学，通过互相交流，共同进步。