

操作系统与虚拟化安全课题研究

|  |  |
| --- | --- |
| 题目： | Android安全体制实现分析 |

|  |  |
| --- | --- |
| 成 员： | 龙东恒 王帅 宋文浩 |
| 院 系： | 软件与微电子学院 |
| 导师姓名： | 沈晴霓 |

二〇一七年 十二月

# **目录**

[**目录** 1](#_Toc502827091)

[**一、选题背景** 2](#_Toc502827092)

[**二、AVD安装** 3](#_Toc502827093)

[2.1 Windows 4](#_Toc502827094)

[2.2 Mac 4](#_Toc502827095)

[**三、Android安全机制** 5](#_Toc502827096)

[3.1安全机制简述 5](#_Toc502827097)

[3.2 安全机制迭代更新过程 5](#_Toc502827098)

[3.3 权限管理——核心机制 5](#_Toc502827099)

[**四、权限管理** 6](#_Toc502827100)

[4.1 权限管理介绍 6](#_Toc502827101)

[4.2 权限管理版本迭代 6](#_Toc502827102)

[4.3 权限管理现有问题 6](#_Toc502827103)

[4.4 权限管理改进建议 6](#_Toc502827104)

[五、实验总结 7](#_Toc502827105)

# **一、选题背景**

简要讲述Android近来的市场规模，以及2016年Android安全报告简述。

# **二、AVD安装**

Android虚拟设备（AVD）是一种仿真器配置，允许开发人员通过模拟真实设备功能来测试应用程序，开发人员无需使用装有Android操作系统的手机、平板电脑及相关电子设备，在AVD中开发人员可以完成绝大部分的开发、测试工作。AVD允许开发人员通过硬件和软件设备对其参数（如API版本、底层Android系统等）进行配置，以满足开发需求。同时，AVD提供用户友好的人机交互图形界面来帮助开发人员轻松地创建和管理AVD。

目前，AVD支持在多种IDE中安装，常见的安装的方式有如下两种：

1. Ecplise + Android Plugin，即选择Ecplise作为开发环境，通过安装Android插件和依赖包完成Android开发环境的配置。
2. Android Studio，直接选择Android专用开发环境。

对于选择何种安装方式，是开发者个人的开发习惯。在Android Studio流行前，Ecplise + Android Plugin基本是安卓开发从业人员的首选，但是在多年的使用过程中，IDE外加插件的方式逐渐暴露出其无法解决的弊病，插件下载安装缓慢、IDE容易崩溃、不齐全的开发工具、无版本更新提示、不友好的用户界面等等，阻碍开发进度。为此，Google出于各种目的，推出自家的Android开发环境——Android Studio，其具有以下强大优势：

* Ecplise中所有功能；
* 基于Gradle的构建支持；
* 直接支持应用签名功能；
* Android特定重构和快速修复；
* 内置Google云、svn、git工具支持；
* 自带布局编辑器，允许用户拖放组件并在多屏幕配置上预览布局；
* 提示工具更好地对程序性能、可用性、版本兼容和其他问题进行控制捕捉；
* 支持NDK等框架。

除此之外，Android Studio的定期更新使得开发者能够第一时间使用到最新特性（目前最新版本为Android Studio 3），大大提高应用开发效率，并一步步取代Ecplise固有的市场地位。

综上，组内选择Android Studio作为实验的开发环境，并介绍在Windows如何安装配置。此外，对于Android开发而言，另一个必备工具是adb，即安卓开发调试器，实验中也会对其进行安装配置介绍。

安装Android模拟器主要分为以下几个步骤：下载最新版本Android Studio 3、SDK，安装IDE，对IDE配置SDK路径，新建项目并下载相关依赖，选择制定版本Android模拟设备，并运行Hello World项目。

# **三、Android安全机制**

在移动设备持有量持续增长的时代，用户对隐私保护的需求愈来愈强烈，Android作为移动设备市场中市场份额最大的移动端操作系统，其不同于IOS封闭的开源性，一方面为它带来了IOS所不能具备的社区活力，但另一方面也为它带来了潜在的安全风险，熟悉Android其底层所具有的安全机制能够使我们更好了解手中的设备，更加清楚如何去保护自己的信息，以防泄露。本章将从Android的安全机制开始，讲解Android的基本安全机制，同时对其版本迭代过程中新加或删减的安全特性做简要的介绍，而实验的核心在于Android中的权限管理机制，一个让用户、开发者既爱又恨的安全机制。

## 3.1安全机制简述

Android提供了一个适用于移动设备的开放源代码平台和应用环境，其软件栈由七层组成，如图3-1所示。自顶向下的各层软件栈，亦称组件，都假定其底层实现的各栈均已采取适当的安全措施。值得注意的是除了作为Root代码运行的少量Android操作系统代码外，Linux内核上方的所有代码都受到应用沙盒的限制。

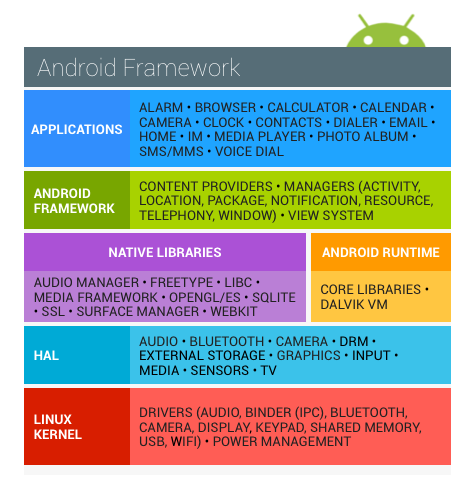


图3-1 Android七层软件栈

在七层软件栈中，其主要构造有以三块：

* HAL：设备硬件，Android能够在多种硬件设备配置中运行，其中包括智能手机、平板电脑、手表、汽车等。Android独立于处理器，但它确实利用了一些针对硬件的安全功能，例如ARM eXecute-Never。
* Linux Kernel：Android核心操作系统是在Linux内核之上构建的。所有设备资源（例如摄像头、蓝牙等）都需要通过操作系统的访问控制才能使用。
* Android Runtime：Android应用通常都是使用Java进行编写，并在ART中运行。不过，仍有许多应用（包裹核心Android服务和应用）实在本机应用或包含本机库。ART和本机应用在相同的安全环境中运行。应用在文件系统中有一个专用部分，他们可以在其中写入私密数据，包括数据库和原始文件。

除此之外，Application层对Android核心操作系统进行了扩展，其应用有两个主要来源，一是预安装应用，由开放源代码Android平台或设备制造商开发的一套预先安装的应用，二是用户自主安装应用，由任何第三方应用开放式开发环境提供。

针对以上的七层软件栈，Android在各层或多层中建立其成熟的多层安全模型，如表3-1所示，其中Linux Kernel层实现的是系统和内核安全性，指依赖于底层Linux所提供的相关特性，自底部保证上层的安全，Android运行时和本地库层实现的是实现安全性，通常指在开发流程、产品实现中常用的为了保障安全的实施方案，Android应用层实现的是应用安全性，指应用使用过程的安全保障。

表3-1 安全机制表

|  |  |
| --- | --- |
| 安全模型组成部分 | 主要安全机制 |
| Linux kernel | 基于POSIX的应用沙盒  基于DAC实现的文件访问控制  基于SElinux实现的强制访问控制  用于文件系统、私有密钥的加密功能  验证启动（类TPM） |
| Android Runtime & Native Libraries | 内存管理单元（MMU）  基于编程语言的强制类型安全  移动设备安全 |
| Android Application | 权限机制  应用签名机制  组件封装  Dalvik虚拟机 |

实验的重点在于上表中的应用层的权限机制的管理，因此，上表中的绝大部分内容将不会做介绍，若是想了解相关的概念，请参考附录中的网址。

## 3.2 安全机制迭代更新过程

了解Android版本更新过程中安全特性的增强，有助于了解设备安全的发展趋势，下图展示了历代版本更新中安全特性的增强项。表3-2的两表展示了Android从5.0开始到最新版本8.0的安全特性的增强，从中可以看到Google在开源Android操作系统的同时，也在不断地将Android OS推向更多的移动设备，同时更加侧重操作系统级别和应用开发级别的安全。4.0和5.0中一个重大的进步在于开始提供加密功能，并将其作为默认安全策略；5.0和6.0则有更多的进步，一方面Google开始重新正视一直受人诟病的应用权限管理，另一方面在新版本中加入指纹识别和外接SDK的额外功能；6.0和7.0版本中，Android把安全特性的重点放在系统与内核的安全性，以及实现层安全，加入类似TPM度量功能的验证启动，强化应用发布过程时的应用签名；7.0和8.0则把重心完全放在内核层的安全上，强化加密功能和密钥管理、对SELinux安全模块做进一步的加强。

**表3-2 Android 5.0与6.0新增特性**

|  |  |
| --- | --- |
| 版本 | 安全增强功能 |
| android 5.0 | 1. **默认加密** 2. **经过改进的全盘加密功能** 3. **通过 SELinux 得到增强的 Android 沙盒** 4. **Smart Lock** 5. **面向手机和平板电脑的多用户功能、受限个人资料和访客模式** 6. **不使用 OTA 的 WebView 更新方式** 7. **经过更新的 HTTPS 和 TLS/SSL 加密功能** 8. **移除了非 PIE 链接器支持** 9. **安全修复程序** |
| android 6.0 | 1. 运行时权限 2. 验证启动 3. 硬件隔离安全措施 4. 指纹 5. 加装 SD 卡 6. 明文流量 7. 系统加固 8. USB 访问控制 |

从Android 4到Android 8的安全特性变化过程中，Android七层软件栈中的Linux Kernel和Application层几乎在每个迭代版本中都有新特性的添加，内核层新增的安全特性主要以修复漏洞、新增加密安全模块、强化SELinux为主，而应用层新增的安全特性则更偏重于产品发布的应用签名和应用权限管理。

在我们的实验中，选择Android安全机制中属于应用安全性的权限管理作为分析的对象。后面章节将会对权限管理做进一步的介绍与说明。

**表3-2 Android 7.0与8.0新增特性**

|  |  |
| --- | --- |
| 版本 | 安全增强功能 |
| android 7.0 | 1. 文件级加密 2. 直接启动 3. 验证启动 4. SELinux 5. 库加载顺序随机化和经过改进的 ASLR 6. 内核加固 7. APK 签名方案 v2 8. 可信 CA 商店 9. 网络安全配置 |
| android 8.0 | 1. 加密 2. 验证启动 3. 锁定屏幕 4. KeyStore 5. 沙盒 6. 内核加固 7. 用户空间加固 8. 流式操作系统更新 9. 安装未知应用 10. 隐私权 |

# **四、权限管理**

权限可以保护设备中的敏感信息，并且尽在应用为正常形式功能而必须访问这些信息时使用。作为用户可以直接控制的安全保障措施。Android在权限管理发生过一次重点的转折，Android 6.0前应用安全成功前必须获得所有的权限，Android 6.0之后应用安装时无需获得所有权限，权限的获取可以在需要访问特定资源前请求即可。当前Android设备市场上，已经罕有4.0版本的移动设备，大多数设备基本在Android 6.0以上。

## 4.1 机制介绍

权限管理是Android系统中最重要的安全措施之一，处于应用性安全，设计之初为了达到方便易用、简单快速的目标，决定以牺牲一定程度的安全性，采用粗细粒度的权限管理机制。此机制遵循“最小特权原则”，即所有的Android应用程序都被赋予了最小权限，若一个应用在安装时不声明任何权限，则该应用没有访问任何敏感资源的权限，即无法获得用户的敏感信息。因此，若应用想访问诸如相册、联系人目录等用户敏感资源，则必须在特定时间向用户请求权限的获取，否则如果缺少必要的权限，由于Android底层基于SELinux的应用沙盒机制，应用将无法正常提供所期望的功能与服务。

权限管理中，Android将所有权限命名为：系统权限。系统权限被划分成四个保护级，分别为：

1. NORMAL：正常权限只需申请即可使用，无需用户授权；
2. DANGEROUS：危险权限涉及用户敏感信息，需要在安装之处或指定操作前向用户申请并获得授权，才可以访问资源
3. SIGNATURE：签名权限仅当申请该权限的应用程序与声明该权限的程序使用相同的签名时，才赋予权限；
4. SIGNATUREORSYSTEM：签名或系统权限仅当申请该权限的应用位于相同的Android系统镜像中，或申请改权限的应用程序与声明该权限的程序使用相同的签名时，才赋予该权限。

对于用户和开发者而言，需要了解的两个最重要保护级别是正常权限和危险权限正常权限涵盖应用需要的访问其沙盒外部数据或资源，但对用户隐私或其他应用操作风险很小的区域。利于，设置时区的权限就是正常权限。如果应用声明其需要正常权限，系统会自动向应用授予该权限，而无需用户参与，在当前的API版本中，正常权限的完整列表可以在官方文档中获得，表4-1截取34个正常权限中的前16个作为展示。

表4-1 当前版本部分正常权限展示

|  |  |
| --- | --- |
| 权限符号 | 权限含义 |
| * ACCESS\_LOCATION\_EXTRA\_COMMANDS | 允许应用程序访问额外的位置提供命令。 |
| * ACCESS\_NETWORK\_STATE | 允许应用程序访问有关网络信息 |
| * ACCESS\_NOTIFICATION\_POLICY | 对希望获得通知权限的应用程序的授权 |
| * ACCESS\_WIFI\_STATE | 允许应用程序存取有关Wi-Fi网络信息。 |
| * BLUETOOTH | 允许应用程序连接到配对的蓝牙设备 |
| * BLUETOOTH\_ADMIN | 允许应用程序发现并配对蓝牙设备 |
| * BROADCAST\_STICKY | 允许应用程序粘性广播请求 |
| * CHANGE\_NETWORK\_STATE | 允许应用程序更改网络连接状态 |
| * CHANGE\_WIFI\_MULTICAST\_STATE | 允许应用程序进入Wi-Fi多播模式 |
| * CHANGE\_WIFI\_STATE | 允许应用程序更改Wi-Fi连接状态 |
| * DISABLE\_KEYGUARD | 允许应用程序在不安全的情况下禁用键盘锁 |
| * EXPAND\_STATUS\_BAR | 允许应用程序展开或折叠状态栏 |
| * GET\_PACKAGE\_SIZE | 允许应用程序查询任何包使用的空间 |
| * INSTALL\_SHORTCUT | 允许应用程序在Launcher中安装快捷方式 |
| * INTERNET | 允许应用程序打开网络套接字 |
| * KILL\_BACKGROUND\_PROCESSES | 允许应用程序调用killBackgroundProcesses |

正常权限需要访问的绝大多数是非隐私数据，不在沙盒保护之内，Android将设备的敏感数据全部保存在设备的指定区域，并通过强制访问控制MAC进行保护，而对于各应用自有的隐私数据，则保存在沙盒之中。危险权限的定义是为了控制隐私数据的访问存取，而所有危险的Android系统权限都有各自所属的权限组。在危险权限中，共定义了9大分组，涉及24个危险权限（组和各危险权限的分类如表4-3所示）。

表4-3 危险权限组及具体权限

由于正常权限的不涉及用户具体隐私信息，也无需用户授权，实验的重点放在危险权限。

## 4.2 权限组

|  |  |
| --- | --- |
| 权限组 | 权限 |
| [CALENDAR](https://developer.android.com/reference/android/Manifest.permission_group.html#CALENDAR) | * [READ\_CALENDAR](https://developer.android.com/reference/android/Manifest.permission.html#READ_CALENDAR) * [WRITE\_CALENDAR](https://developer.android.com/reference/android/Manifest.permission.html#WRITE_CALENDAR) |
| [CAMERA](https://developer.android.com/reference/android/Manifest.permission_group.html#CAMERA) | * [CAMERA](https://developer.android.com/reference/android/Manifest.permission.html#CAMERA) |
| [CONTACTS](https://developer.android.com/reference/android/Manifest.permission_group.html#CONTACTS) | * [READ\_CONTACTS](https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) * [WRITE\_CONTACTS](https://developer.android.com/reference/android/Manifest.permission.html#WRITE_CONTACTS) * [GET\_ACCOUNTS](https://developer.android.com/reference/android/Manifest.permission.html#GET_ACCOUNTS) |
| [LOCATION](https://developer.android.com/reference/android/Manifest.permission_group.html#LOCATION) | * [ACCESS\_FINE\_LOCATION](https://developer.android.com/reference/android/Manifest.permission.html#ACCESS_FINE_LOCATION) * [ACCESS\_COARSE\_LOCATION](https://developer.android.com/reference/android/Manifest.permission.html#ACCESS_COARSE_LOCATION) |
| [MICROPHONE](https://developer.android.com/reference/android/Manifest.permission_group.html#MICROPHONE) | * [RECORD\_AUDIO](https://developer.android.com/reference/android/Manifest.permission.html#RECORD_AUDIO) |
| [PHONE](https://developer.android.com/reference/android/Manifest.permission_group.html#PHONE) | * [READ\_PHONE\_STATE](https://developer.android.com/reference/android/Manifest.permission.html#READ_PHONE_STATE) * [CALL\_PHONE](https://developer.android.com/reference/android/Manifest.permission.html#CALL_PHONE) * [READ\_CALL\_LOG](https://developer.android.com/reference/android/Manifest.permission.html#READ_CALL_LOG) * [WRITE\_CALL\_LOG](https://developer.android.com/reference/android/Manifest.permission.html#WRITE_CALL_LOG) * [ADD\_VOICEMAIL](https://developer.android.com/reference/android/Manifest.permission.html#ADD_VOICEMAIL) * [USE\_SIP](https://developer.android.com/reference/android/Manifest.permission.html#USE_SIP) * [PROCESS\_OUTGOING\_CALLS](https://developer.android.com/reference/android/Manifest.permission.html#PROCESS_OUTGOING_CALLS) |
| [SENSORS](https://developer.android.com/reference/android/Manifest.permission_group.html#SENSORS) | * [BODY\_SENSORS](https://developer.android.com/reference/android/Manifest.permission.html#BODY_SENSORS) |
| [SMS](https://developer.android.com/reference/android/Manifest.permission_group.html#SMS) | * [SEND\_SMS](https://developer.android.com/reference/android/Manifest.permission.html#SEND_SMS) * [RECEIVE\_SMS](https://developer.android.com/reference/android/Manifest.permission.html#RECEIVE_SMS) * [READ\_SMS](https://developer.android.com/reference/android/Manifest.permission.html#READ_SMS) * [RECEIVE\_WAP\_PUSH](https://developer.android.com/reference/android/Manifest.permission.html#RECEIVE_WAP_PUSH) * [RECEIVE\_MMS](https://developer.android.com/reference/android/Manifest.permission.html#RECEIVE_MMS) |
| [STORAGE](https://developer.android.com/reference/android/Manifest.permission_group.html#STORAGE) | * [READ\_EXTERNAL\_STORAGE](https://developer.android.com/reference/android/Manifest.permission.html#READ_EXTERNAL_STORAGE) * [WRITE\_EXTERNAL\_STORAGE](https://developer.android.com/reference/android/Manifest.permission.html#WRITE_EXTERNAL_STORAGE) |

Android中的权限以Permission Group的形式来组织与划分与设备功能或特性相关的权限。该体系分成三层，自顶向下依次为权限组、具体权限、具体权限对应的API调用。在这一体系中，权限请求在组级别进行处理，相当于应用清单中的多个权限声明。例如对于表4-3中的STORAGE权限组的声明，则相当于同时向用户请求授权READ\_EXTERNAL\_STORAGE和WRITE\_EXTERNAL\_STORAGE权限（读取和写入外存）。

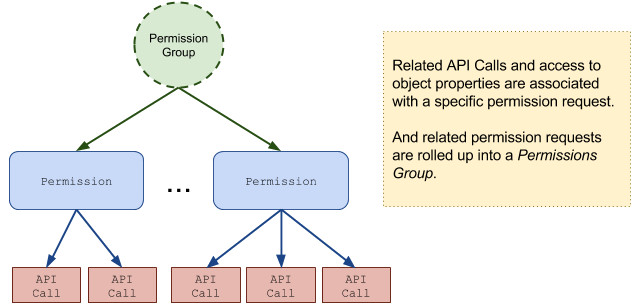


图4-1 Android权限组体系

采用这种结构的优势在于提供用户友好度，应用一旦获得组的访问权限，就能够使用该权限组具有的所有API调用，而无需用户进行无意义的多次点击授权。虽然以权限组的方式能够提高效率，为用户带来便捷，但同样也带来相关的安全风险。

## 4.3 权限机制风险

对于以牺牲安全性，采用粗粒度的权限管理机制来实现让用户自主控制应用安全的方式而言，这种机制在实际生活中逐渐暴露出其安全缺陷，通过相关的文献查询和实践操作，综合来说有下面几种缺陷。

**过粗的授权机制**。应用若请求对某种资源的指定操作时，往往并不以权限组中指定危险权限去请求，而是以权限组级别去请求。这有可能是由于产品开发团队在开发过程中图便利或为扩展权限所做的准备，也有可能是恶意开发者故意为之。这种过粗的授权机制，极易造成用户的隐私信息泄露，若其性质属于恶意，则可以把这类授权行为归之为下文中的溢权问题。图4-2选择用短信权限组作为例子进行展示，图中应用Application的业务逻辑中只涉及到发送短信，所以该应用理应指向用户请求发送短信单一权限即可，但是在图中，Appcation以SMS权限组的形式向用户提出授权并获得许可，此时对于应用Application而言，由于这种过粗的授权机制、用户安全意识薄弱或开发人员故意所为导致它拥有读取短信内容等允许访问短信相关的操作权限，致使用户的短信信息有可能遭到窃取。

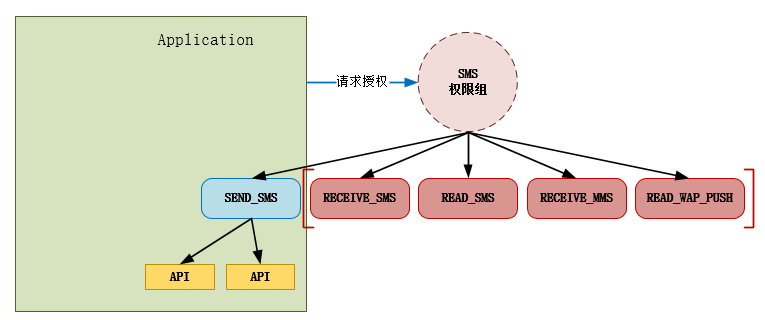


图4-2 过粗的授权机制

**溢权问题**，是指应用程序向用户申请的权限中，存在超出应用业务逻辑的额外权限，违反了Android在权限管理中所遵循的最小特权原则。和上文的过粗的授权方式有所不同，溢权问题不仅仅指申请权限组内其他子权限，还包括申请其他权限组的权限，也就是超出业务逻辑以外的。造成溢权问题的原因可能有以下方面：

1. 用户敏感资源与具体权限的绑定关系不清楚。由于敏感资源的访问方式与特定权限下的API调用之间没有清楚的关系定义，所以开发者/攻击者可以利用模糊的绑定关系，间接获得敏感资源信息。
2. 应用业务逻辑与敏感资源间没有绑定关系。应用中的所有业务流程，如果要访问敏感资源信息，则必须要显式的表现并且由用户主动触发，即用户可以通过人机交互的方式触发敏感资源的访问，了解敏感资源的数据流动。然而在实际中，业务逻辑与资源之间的绑定是困难的，用户在应用的生命周期中只是使用的角色，无法完全了解应用的具体业务逻辑，其次源码层无法做到检测资源隐式调用。

溢权现象由于其检测的困难性，一直无法得到很好的解决方式，对于此类现象需要用户、软件开发商、硬件制造商以及国家法律共同的合作才能够有效的抑制。图4-3和4-4分别表示同权限组溢权现象和跨权限组溢权现象。第一张图中，Application只有Send Message业务需要使用SMS权限组中的单一SEND\_SMS发送短信权限，但是却发出权限组级别的授权请求，导致Application拥有了其不该拥有的其他权限。第二张图中Application没有需要使用Camera的业务，但是却也向用户请求了使用CAMERA的权限。

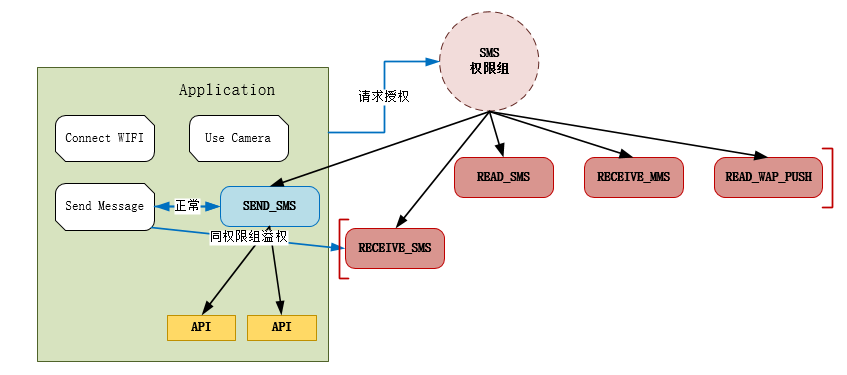


图4-3 同权限组溢权

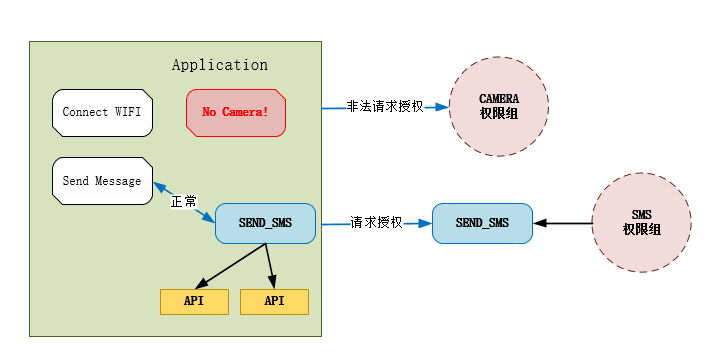


图4-4 跨权限组溢权

**间接侵权问题**，是指应用B不具备访问特定敏感资源的权限，应用A由于恰当或不恰当的用户授权后拥有访问该资源的权限，且应用A开放或不保护自身应用与其他应用的通信权限，导致应用B通过应用A间接获得该资源信息。图4-5展示简单的间接侵权流程。在实际中，由于应用往往是自闭的，不对外或仅对少数应用开放接口，所以间接侵权中所经过的应用数量不单一，而且应用间合谋的可能性也相当大。此外，间接侵权所依托的宿主应用也有可能存在溢权问题，所以，Android权限管理的安全隐患可能由多重隐患叠加而成。

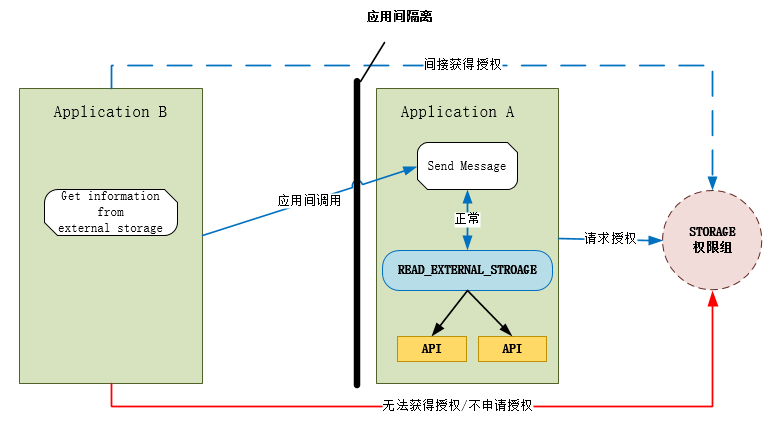


图4-5 间接侵权问题

## 4.4 Android 6.0权限实践

Android 6.0中新加入权限动态管理，由用户在应用中遇到某些需要访问特定敏感资源时再进行授权，一旦该操作获得授权后不再需要再次授权。Runtime-Permission安全特性的加入，优化了固有权限管理的使用流程，具体有以下进步：

* 应用安装时的权限无需全部获得也可成功安装。Android 6.0前的系统中要求应用开发商必须显式罗列应用所需要的全部权限，并且强制要求用户同意上述权限后才能完成安装。若用户拒绝授权某一权限，应用的安装将会失败。
* 应用安装时被拒绝的授权可在应用的使用中动态获得。Android 6.0后对敏感资源的访问若起初是拒绝的，但在业务中需要，应用会提示用户对该资源的访问必须给予授权方可进行，若用户接受则不需再次授权，反之，应用会重复提示。
* 应用的授权可在设置中进行更改。在Android 6.0以前，应用获得所有授权后才能够成功安装，但是要想变更应用的某些授权，则只能够卸载应用，再次安装，而6.0版本以后，应用的某些授权则可以在设置中进行配置，不需要删除也不需要再次安装。

动态权限的加入，使得权限管理在某种程度上，将敏感资源与与业务逻辑间建立联系，能够在一定程度上减少溢权问题的出现。但是Android基于权限组的安全机制，粗粒度权限管理则仍旧存在问题。下面将展示实验中应用安装、使用的动态权限。

以咕咚为例，从应用商店中下载咕咚apk包，并进行安装。下载完成后，进入权限许可页，不授予咕咚任何权限，仍能成功安装。

图4-6 不授予任何权限顺利安装咕咚

打开咕咚应用，并在不授予相机权限的情况下使用相机，应用会弹出权限授权提示，要求用户对应用使用相机进行授权。

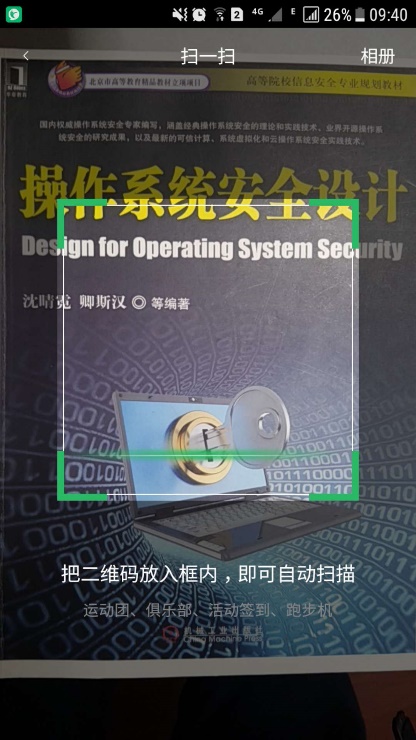
 

图4-7 使用相机时进行请求授予权限

值得注意的是，如果应用在安装时罗列的权限，在其业务逻辑中并不具备，用户忽略且同意授权的话，Running-Permission动态权限管理则形同虚设。我们发现豆瓣App存在这种的嫌疑，豆瓣App在应用安装页面共请求获得电话、相机、位置信息、存储、通讯录等权限，但是在其业务中并没有使用相机的权限。所以，对相机的使用权限属于额外的权限，豆瓣存在溢权安装的嫌疑。

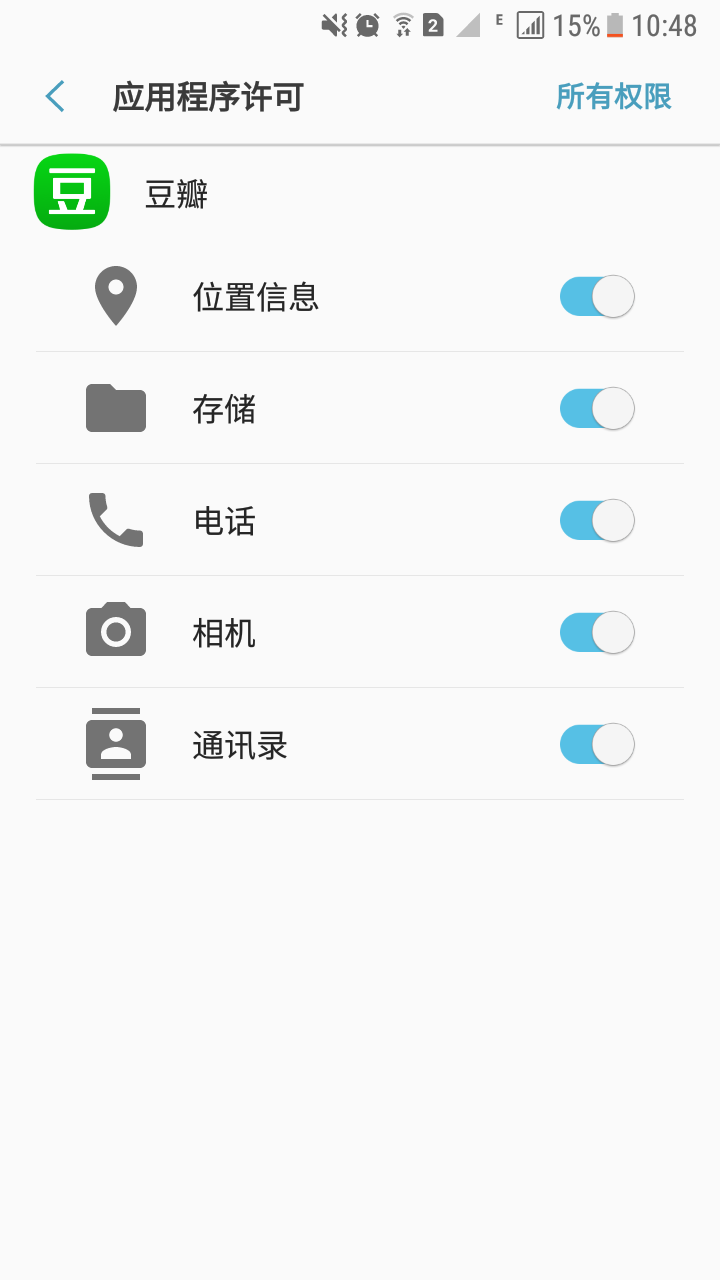
 

图4-8 豆瓣溢权安装嫌疑

# **五、实验总结**

实验中侧重于Android安全机制中关于权限部分的管理，对权限机制在做了简要的介绍，并对其中的安全隐患进行分析，同时在使用现有App应用程序对权限管理机制做展示说明。对Android现有的权限管理机制，我们提出以下建议：

1. 完善“资源-权限-API”的说明文档，中间两种映射关系的不明确，容易导致攻击者间接获得指定资源的访问权限。
2. 建立其业务逻辑与资源的绑定关系，并要求所有与敏感资源的调用都必须要在应用中显式的表现，不允许以隐式的方式调用。
3. 应用安装时所请求的所有权限，都能够在其业务逻辑中轻易找到，以填补动态权限管理的缺失。
4. 强制要求软件开发商在应用说明时对其所要求的权限进行解释说明。

Android平台的安全性在日益提高的同时，希望使用该平台的用户提高保护个人隐私的安全意识，生产者和消费者双方共同的进步才能够更快更好地建立更安全可靠的移动平台。