

编号

西安科技大学

# 毕业设计（论文）

（ 2017 届）

题 目	<u>基于自动测试的安卓应用程序行为抽取</u>
学生姓名	<u>李妍娜</u>
学 号	<u>1308010302</u>
专业班级	<u>软件工程 1303 班</u>
指导教师	<u>刘晓建</u>
所在学院	<u>计算机科学与技术</u>
日 期	<u>2017 /6/8</u>

## 摘 要

从 Android 系统产生至今，已经占领了移动设备操作系统约 75% 的市场，但同时由于 Android 系统巨大的用户量和开源的特性，Android 系统的安全性受到了极大威胁。数字营销网站 MarketingLand 报道，在过去的 12 个月里 Google Play 应用程序下载量已经上升到 500 亿次，大量的应用在丰富用户体验的同时也带来了日益凸显的安全问题。因此研究基于 Android 操作系统的安全机制对于防御恶意软件攻击有着重要的作用。

大部分恶意软件实现某些功能时需要调用特定 API 来完成，通常通过静态分析和动态分析这两种方式来对恶意 apk 进行检测，动态分析则是指通过运行该软件并且跟踪和监控其运行时的行为来分析程序特征，以便更准确地捕捉到软件的恶意行为。在 Android 平台中，为了抽取应用程序运行时的行为特征，获取软件运行时调用敏感 API 的信息，本文设计并完成了对 apk 反编译后的 smali 文件中的敏感 API 注入日志的系统，可以自动地在 apk 中注入日志代码，在软件运行的时候记录该软件调用的敏感 API 信息。本文的工作内容如下：

- ①设计和实现了对 smali 文件中植入监控代码的算法，并通过若干实例验证了算法的正确性和有效性；
- ②用 Monkey 自动化测试方法对软件进行了自动运行和测试，避免了手工测试的费时费力；
- ③开发了基于 Java swing 的自动化工具，实现了对 apk 的反编译、注入代码、重新打包和签名、并用 Monkey 进行自动运行和测试这一完整过程的封装。通过若干实例测试验证了本工具的正确性。

**关键词：**敏感 API 调用；动态分析；Android；smali

## ABSTRACT

From the Android system has been created, has occupied the mobile device operating system about 75% of the market, but at the same time due to the huge amount of Android system and open source features, Android system security has been a great threat. Digital marketing website MarketingLand reports that Google Play app downloads have risen to 50 billion times over the past 12 months, and a large number of applications have experienced a growing user experience and a growing security issue. So the study based on the Android operating system security mechanism for the defense of malicious software attacks have an important role.

Most malware to achieve certain functions need to call a specific API to complete, usually through static analysis and dynamic analysis of these two ways to detect malicious apk, dynamic analysis is through the operation of the software and track and monitor its run time Behavior to analyze program features in order to more accurately capture the malicious behavior of the software. In the Android platform, in order to extract the behavior of the application runtime behavior, access to the software runtime call sensitive API information, this paper designed and completed the anti-compiled ap51 ap51 file in the sensitive API injection log system, In the apk into the log code, the software is running when the software records the sensitive API information. The work of this paper is as follows:

①The algorithm of implanting the monitoring code in smali file is designed and implemented, and the correctness and validity of the algorithm are verified by several examples.

②with the Monkey automated test method of the software for automatic operation and testing, to avoid the manual test time and effort;

③Developed a Java swing-based automation tool that implements the decompilation, injection code, repackaging, and signature of apk, and automates and tests this complete process with Monkey. The correctness of this tool is verified by several examples.

Keywords: sensitive API call; dynamic analysis; Android; smali

# 目 录

<b>1 绪论</b> .....	- 1 -
1.1 研究背景及意义 .....	- 1 -
1.2 研究现状 .....	- 3 -
1.3 论文结构 .....	- 4 -
<b>2 Android 操作系统介绍</b> .....	- 5 -
2.1 应用程序包结构 .....	- 5 -
2.2 Dalvik 虚拟机与 Java 虚拟机的比较 .....	- 6 -
2.3 Android 应用程序四大组件 .....	- 7 -
2.4 Android 系统安全机制 .....	- 9 -
2.5 Android 系统安全机制的缺陷 .....	- 11 -
<b>3 技术背景</b> .....	- 12 -
3.1 反编译工具 Apktool .....	- 12 -
3.2 smali 语法介绍 .....	- 13 -
3.3 apk 签名机制 .....	- 14 -
3.4 Monkey 随机测试 .....	- 15 -
3.5 批处理 .....	- 16 -
3.6 ADB 调试桥 .....	- 16 -
<b>4 基于自动测试的应用程序行为抽取系统的设计实现</b> .....	- 18 -
4.1 系统总体框架 .....	- 18 -
4.2 反编译模块 .....	- 20 -
4.3 日志代码植入模块 .....	- 22 -
4.4 回编译模块 .....	- 24 -
4.5 签名模块 .....	- 26 -
4.6 安装、运行和显示敏感 API 调用模块 .....	- 27 -
4.7 测试与数据统计 .....	- 32 -

5 结束语 .....	- 37 -
致谢 .....	- 38 -
参考文献 .....	- 39 -

# 1 绪 论

## 1.1 研究背景及意义

近几年，Android 系统的发展非常迅速，在全球智能手机市场的占有份额已经超过 50%。由于 Android 系统具有开源性、易操作性、界面美观、允许多个应用程序同时运行等特点，所以很受用户喜欢。并且 Android 系统为程序开发人员提供了大量的功能性接口，方便了开发人员的工作，同时也使 Android 系统更易扩展和操作。但 Android 系统的开放性及其便利性也给用户带来了安全隐患，滋生了大量恶意软件，使得 Android 系统受到极大威胁。

截止 2016 年，中国智能手机的用户规模已达到 6.31 亿，相比 2015 年增长了 4.9%<sup>[3]</sup>。图 1-1 为 15-16 年中国智能手机用户规模的柱状图。

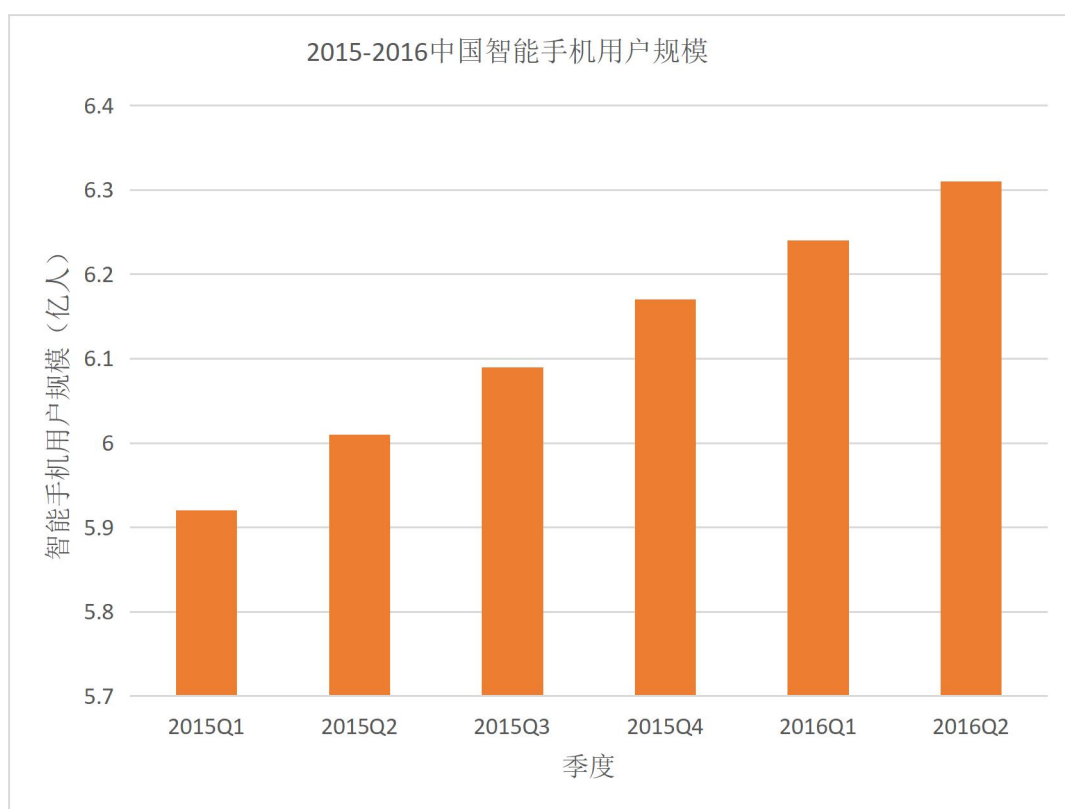


图 1-1 15-16 年中国智能手机用户规模

图 1-2 为 2016 年手机恶意软件感染用户系统分布图，从图可以看出，Android 系统占了大部分，占比 91.45%；其次是苹果 IOS 系统，占比 4.33%。

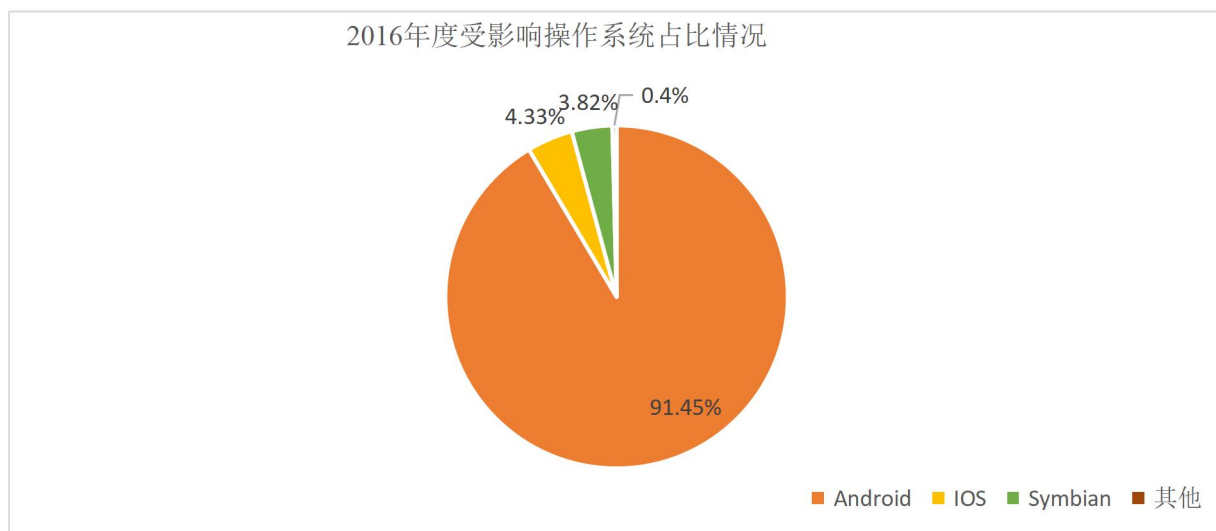


图 1-2 手机恶意软件感染用户系统分布图

图1-3为按伪装应用维度统计的恶意程序分布图：色情播放类占比70%，伪装成系统程序占比15%，伪装成实用工具的占比3%，伪装成支付类的占比1%。

图1-4为按危害类型统计的恶意程序分布图：资费消耗最高占比44%，恶意扣费占比37%，隐私窃取占比13%，恶意广告占比5%。

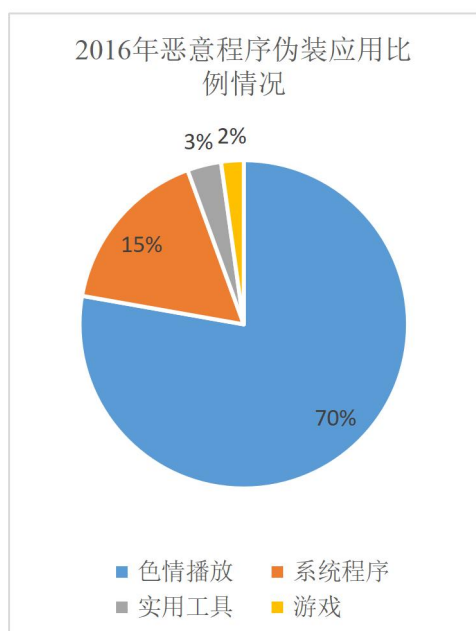


图1-3 伪装应用饼状图

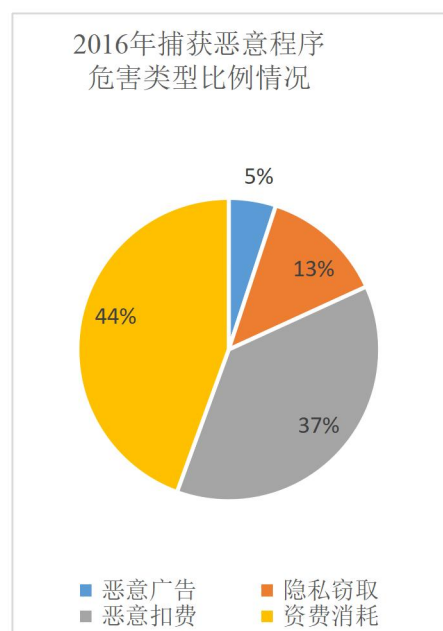


图1-4 危害类型饼状图

但是当前应用市场上的手机安全软件大多都只针对手机病毒、系统优化加速、短信拦截等进行控制，没有直观地反映手机中有哪些应用程序涉及到了敏感权限访问，也没有告诉用户哪些应用程序可能会泄漏个人隐私或具有泄漏个人隐私的意图。而这一问题可通过动态分析方法得以解决，因此对 Android 应用程序调用敏感 API 的动态分析是一个非常重要的研究课题，具有实用价值和指导意义。

## 1.2 研究现状

国外学者对于 Android 恶意应用检测做过很多研究，例如 Androidguard 工具，它是基于签名原理的，用来检测恶意代码，但是这种方法存在不足，它不能对未知的恶意样本进行分析检测。David Kleidermacher<sup>[4]</sup>提出了 ARMTtustZone 技术来提升 Android 移动设备的安全，从系统底层检测系统运行行为，它像软件提供了两个运行区域，用户经常使用的普通软件放在普通区中运行，安全关键软件运行在安全区中，安全区里的软件有权访问普通区，而普通区的软件则不允许访问安全区。XManDroid<sup>[5]</sup>是防止权限提升攻击的安全框架，可以使由 Android System service 或 Content Provider 建立的（隐蔽）信道得到有效的检测。

国内很多学者和教授也对恶意应用的预防做了很多研究，路程等<sup>[6]</sup>人提出通过系统网络日志和系统广播监控恶意应用行为的方法，彭智俊等人提出利用 soot 工具来检测系统的恶意行为，腾讯实现了动静结合的 MalDroidAnylyzer<sup>[7]</sup>分析沙盘，可以对行为数据进行提取生成分析报告。

对于恶意软件的研究大体可以分为静态分析和动态分析，静态分析<sup>[8]</sup>是指通过 Android 逆向编译工程得到程序的 smali 语言代码，通过对反编译出来的 smali 代码进行分析，从而在代码层次上实现对软件恶意行为的准确定位，但是由于很多恶意软件采用了加密、代码混淆等方式，很难对其进行逆向编译，因此静态分析无法对其进行检测；动态分析技术有两种方法：一种是动态调试技术，另一种是动态行为分析技术<sup>[9]</sup>。动态调试是指使用 IDA 等反编译工具得到软件的 C 语言伪代码，然后对程序的 C 语言伪代码加断点进行调试，深入理解代码呈现出来的算法核心和逻辑结构，动态行为分析是在运行应用程序时，借助 Android 的 Logcat 系统对 Android 程序运行时的行为进行记录，然后分析它的行为日志，从而进行研究。

动态行为分析时常采用三种方式获取行为信息：第一种是对敏感 API 进行 HOOK，它的原理是在程序运行的过程中动态地修改目标函数地址的数据，然后使用 jmp 跳转至该函数的地址，等执行完成之后再恢复目标函数的内存数据，在代理函数中获取到 API 的调用信息；第二种是在 smali 代码中对敏感 API 注入日志的方法，它需要先获取 apk



的 smali 代码，然后遍历所有的 smali 代码，找到有敏感 API 调用的代码段进行日志代码的植入<sup>[10]</sup>，这种方法能够更加准确的定位到 apk 在运行时的各种行为信息；第三种是捕获进程提取关键的行为特征，对此，Xu 等人<sup>[11]</sup>提出 Aurasium，在应用程序中嵌入跟踪程序，截获应用程序的系统调用，Burguera 等<sup>[12]</sup>人提出 Crowdroid，在 Android 端使用追踪器 strace 采集行为数据，然后传到服务器进行分析。

### 1.3 论文结构

本文的主要内容是设计一个基于 Monkey 自动化测试的应用程序行为抽取系统，可以对 APK 运行中调用的敏感 API 进行输出，本文共有五章，结构如下：

第一章，绪论。本章介绍了研究背景、意义以及研究现状，主要描述了当前阶段 Android 市场的发展状况和面临的安全问题，要处理恶意 APK 对 Android 用户造成的威胁必不可少的步骤就是分析恶意 APK 调用的敏感 API，在此基础上抑制恶意 APK 的增长。

第二章，Android 操作系统的介绍。在本章中着重介绍了 Android 系统的架构，包括应用层、框架层、系统运行层、Linux 内核层这四层各自的功能，然后介绍了 Android 的四大组件以及它们之间的通信和协作，还介绍了 Android 中的 Dalvik 虚拟机与 Java 虚拟机的不同，紧接着分析了 Android 安全机制及安全机制的不足之处。

第三章，技术背景。本章对这次课题涉及到的相关技术做了简明扼要的介绍，包括 Google 的反编译工具 apktool、反编译之后的 smali 语法、apk 签名机制、monkey 随机测试、批处理、adb 调试桥等等。

第四章，系统的详细设计。本章对此系统的各模块的原理、设计思路、流程等进行了分析和介绍，并用实例对本系统的有效性进行了测试，同时统计了结果。

第五章，总结和展望。本章对毕设和论文的工作内容进行了总结同时对以后的工作进行了规划。

## 2 Android 操作系统介绍

### 2.1 应用程序包结构

一个 Android 程序由多个文件以及文件夹组成，这些文件分别用于不同的功能，常用文件和文件夹如图 2-1 所示。

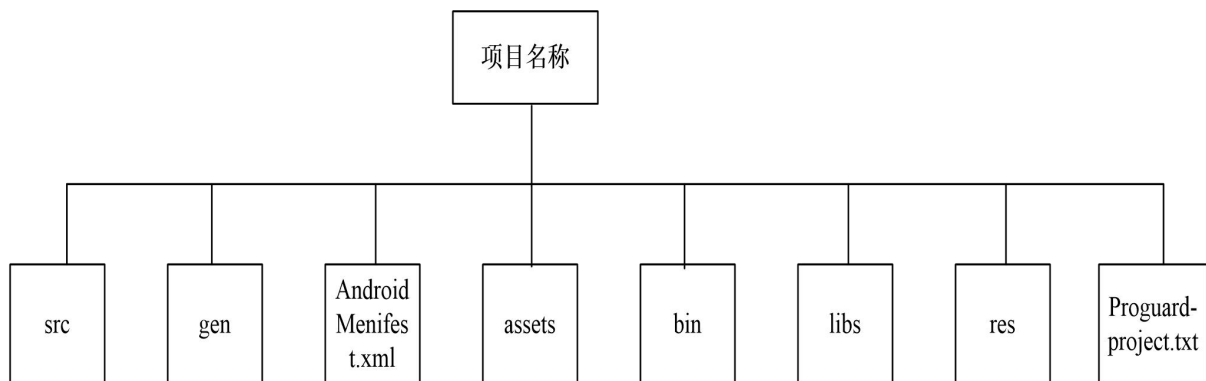


图 2-1 Android 程序的包结构

- **src**: 该目录下放置的是 Android 程序的所有包及 Java 代码，开发者可以在 **src** 目录下创建包，在包中创建 Java 源文件。
- **gen**: 该目录是 Android 程序中的特殊目录，它是自动生成的，该目录有一个 **R.java** 文件，这个文件是用来定义 Android 程序中所有资源的索引，开发者在编写代码时可以通过这个索引对各种资源进行访问。
- **assets**: 该目录中存放的是一些随程序一起打包的文件，通常是一些多媒体文件资源。当 Android 程序打包时会将其一起打包，安装时会解压到对应的 **assets** 目录下。
- **bin**: 该目录主要包含了一些编译时自动产生的文件，其中包括一个当前项目的 **apk** 安装包。
- **libs**: 如果项目中用到了第三方的 Jar 包，就要把这些 jar 包都放在 **libs** 目录下。
- **res**: 该目录下放的是 Android 要用到的各种程序资源，如图片、布局、字符串等。图片放在 **drawable** 目录下，布局放在 **layout** 目录下，字符串放在 **values** 目录下。
- **AndroidManifest.xml**: 该文件相当于清单文件，在程序中定义的 **Activity**、**Service**、**Receiver**、**Provider** 都要在这个文件里注册，另外应用程序的权限声明也在该文件中。
- **proguard-project.txt**: 该文件是 Android 提供的混淆代码工具 **Proguard** 的配置文件，通过该文件可以混淆应用程序中的代码，防止应用程序被反编译出源码。

## 2.2 Dalvik 虚拟机与 Java 虚拟机的比较

Dalvik 虚拟机是由 Google 公司开发的用于执行 Android 程序的虚拟机，它可以简单高效地实现线程管理和进程隔离，还大大地提高了内存的使用效率。每一个 Android 的应用程序都有一个 Dalvik 虚拟机实例与其对应，程序的代码在虚拟机解析后方能执行。

但是 Dalvik 虚拟机不是根据 Java 虚拟机的规范实现的，两者有很多不同之处，而且也不兼容。如图 2-3 所示<sup>[14]</sup>。

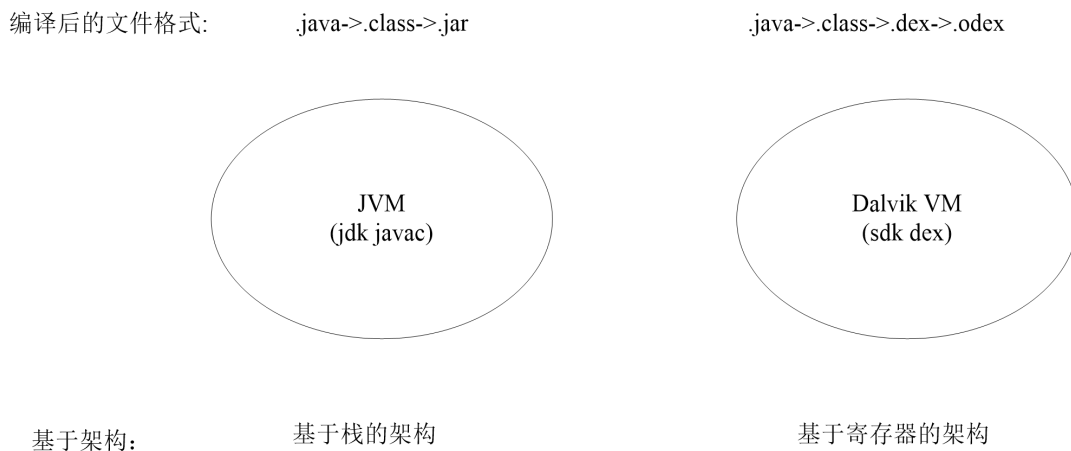


图 2-2 Java 虚拟机和 Dalvik 虚拟机

从图中可以看出，Java 虚拟机和 Dalvik 虚拟机主要有两大区别：一是它们编译后的文件不同；二是它们基于的架构不同。具体如下：

### ①编译后的文件不同

Java 虚拟机上运行的文件格式是.class 文件，而在 Dalvik 虚拟机上运行的是文件格式是.dex 文件。Java 程序中的 Java 类会被翻译成为一个或多个字节码文件（.class）然后把它们打包到.jar 文件，之后 Java 虚拟机会从相应的.jar 文件和.class 文件中获取相应的字节码。而 Android 程序虽然是用 Java 语言编写的，但是在翻译成.class 文件后还会将所有的.class 文件转换成一个.dex 文件，然后 Dalvik 虚拟机会从其中读出数据和指令，最后的.odex 文件是为了在运行过程中提高性能而对.dex 文件进行的优化，能够加快软件的加载速度和开启速度。

### ②基于的架构不同

Java 虚拟机是基于栈结构的，栈是连续的内存空间，存入和取出的速度都比较慢；而 Dalvik 虚拟机则是基于寄存器架构的，寄存器是 CPU 上的一块缓存，它的存取速度比

从内存中存取要快得多，这样就能根据硬件最大限度地优化设备，更适合移动设备的使用。

## 2.3 Android 应用程序四大组件

Android 应用由 Activity、Broadcast Receiver、Content Provider、Service 四大组件构<sup>[6]</sup>。它们的交互模型如图 2-3 所示<sup>[15]</sup>。

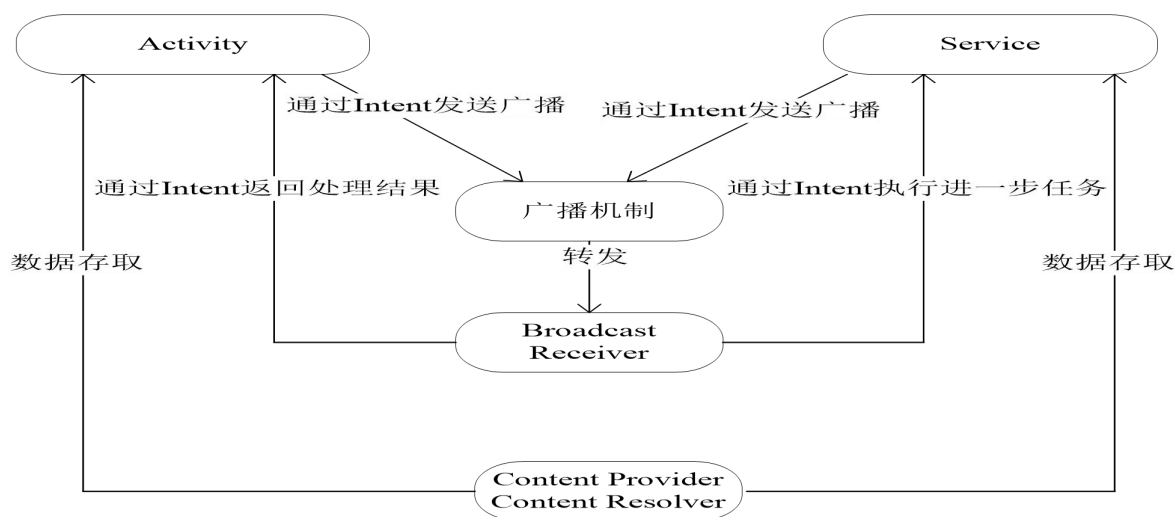


图 2-3 四大组件交互模型

### ①Activity

Activity 是 Android 应用程序中最基本的组件，每个应用程序中，可以有一个或多个 Activity，但是必须有一个入口的 Activity 当程序打开时首先显示给用户。每一个 activity 被给予一个默认窗口，每一个 activity 作为一个独立的类被实现，并且继承了 Activity 这个基类。Activity 类是由一些 View 视图组成的用户接口，并对事件作出响应。Activity 的生命周期如图 2-4 所示。

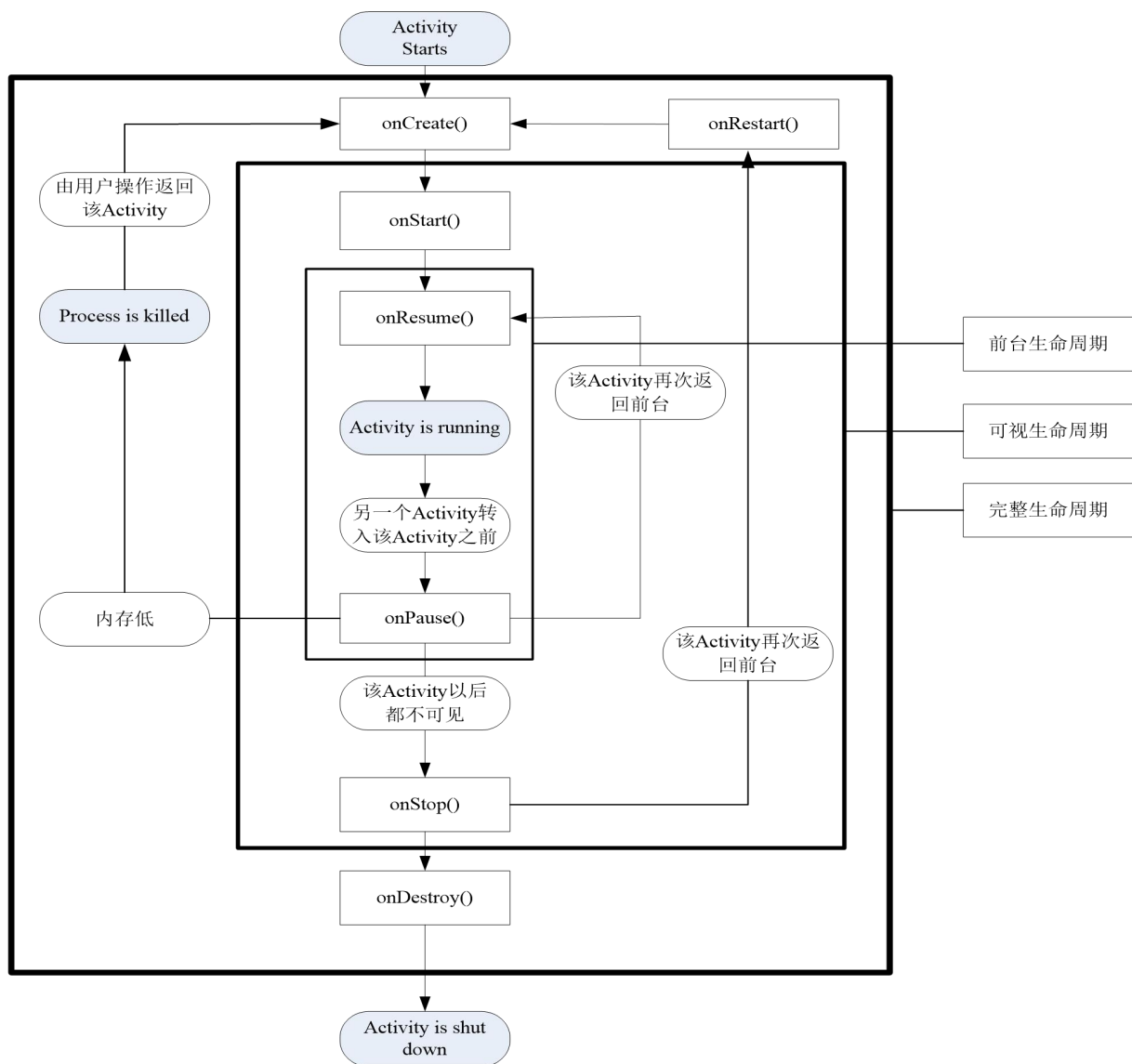


图 2-4 Activity 的生命周期

## ②Broadcast Receiver

Broadcast Receiver 翻译过来就是广播消息接收器，它不负责执行某个任务。它其实相当于一个全局的监听器。其他组件是 Broadcast Receiver 的事件源，这一点和普通事件中的监听器很类似，但是普通监听器是监听组件中的对象，而 Broadcast Receiver 监听其他组件。

开发者创建自己的类继承 **Broadcast Receiver** 这个类，然后重写 **onReceive()**方法，当其他组件发送广播消息时，如果 **IntentFilter** 中配置的消息与其相同，则 **onReceive()**方法被触发。**Broadcast Receiver** 通过 **Context.registerReceiver()**方法注册，然后再在 **AndroidManifest.xml** 中用<receiver>元素注册。

### ③Service

**service** 没有用户界面，它同样是一个 **Android** 组件，它通常不用和用户进行交互，也没有图形用户界面，它会一直运行在后台，它具有自己的生命周期。每个 **service** 都扩展自类 **Service**。**Service** 组件监控组件的运行并且为其他应用提供后台的服务。

### ④Content Provider

**Android** 应用运行在各自的进程中，它们相互之间是独立的，当应用程序之间需要进行数据交互时就需要用到 **ContentProvider**，用户要实现自定义的 **Content Provider**时需要实现 **insert(Uri,ContentValues)**、**delete(Uri,ContentValues)**、**update(Uri,ContentValues,String,String[])**、**query(Uri,String[],String,String[],String)**这些抽象方法。

## 2.4 Android 系统安全机制

在 **Android** 中，安全设计覆盖贯穿在系统架构的每个层面，包括 **Linux** 内核层、应用程序框架层、运行库层、应用程序层等。**Android** 力求在开源开放的同时保障系统的安全，保护用户设备和程序以及数据信息等安全，**Android** 的安全模型主要包括以下机制<sup>[6]</sup>：

- 进程沙箱隔离机制
- 应用程序签名机制
- 权限声明机制
- 访问控制机制
- 进程通信机制
- 内存管理机制

①进程沙箱隔离机制，**Android** 程序在其安装的时候被赋予了独特且固定的用户标识（**UID**），系统为每个程序分配一个沙箱，每个应用程序在其各自的 **Dalvik** 虚拟机中运行，拥有独立的运行空间和资源，互不干扰，默认情况下不能使用系统资源或其他应用的资源，而签名相同的两个应用程序之间可以通过在 **manifest** 文件中添加 **sharedUserId** 属性来共享数据，使得 **Android** 应用程序在安装时被赋予独特的用户标识（**UID**），并永久保持。应用程序及其运行的 **Dalvik** 虚拟机运行在独立的 **Linux** 进程空间，与其它应用程序完全隔离。如图 2-5 所示。

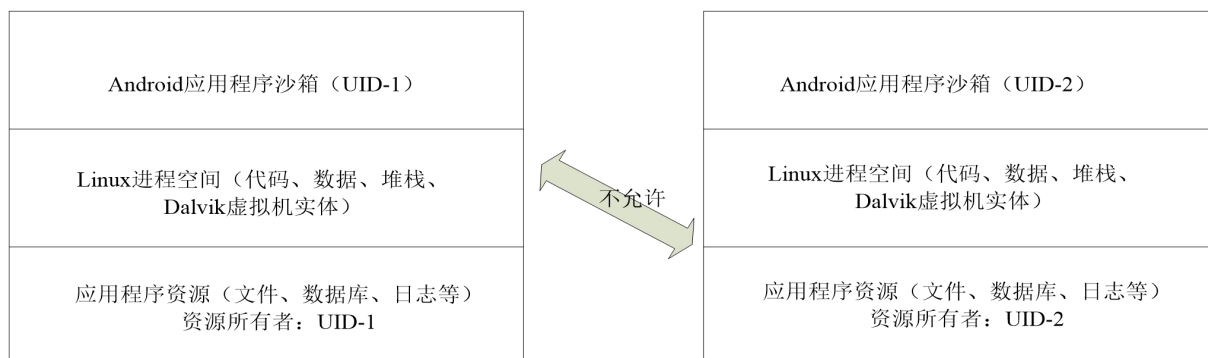


图 2-5 Android 沙箱隔离机制

②应用程序签名机制，Android 规定了开发者必须对开发出的 APK 文件进行数字签名，通过签名来表示一个应用程序和它的开发者之间具有某种信任关系。系统在安装应用程序的时候，首先会检查该 apk 是否具有签名，如果没有将不允许安装，有签名才会安装成功。并且当该应用程序有更新的版本时，系统会检查其新版本的签名是否与之前的相同，如果相同则替换之前的版本，如果不同，则当成一个新的应用程序安装。如果 Android 开发者开发了几个具有相同名字的安装包，这时则可以通过签名来区分，如果签名不相同，则不会被替换掉，也防止了恶意的软件替换该应用程序。

③权限声明机制，权限分为系统权限、Android ROOT 权限、Android 应用程序权限。对于应用程序权限，这个权限是由开发者在 Manifest 文件中声明的，然后安装的时候由用户授权。判断 UID 的值来确认属于哪种进程从而判断所具有的权限，如果 UID 为 0，则为 root 权限，对权限不做控制，如果 UID 是 system server 的 uid，说明是 system server 也不做控制，否则的话通过 PackageManagerService.checkUidPermission() 在 mGrantedUri-Permissions 这个表中查找并判断是否具有相应的权限。

④访问控制机制，这一机制是为了防止系统文件和用户的数据安全被非法访问。Linux 内核包含强访问控制机制和自主访问控制机制，但是由于某些原因谷歌并没有将强访问控制机制加入 Android 系统中去，文件访问权限控制实现了自主访问控制机制，文件的权限通过 user、group、other 与读、写、执行组合起来实现的，一般只有 root 和 system server 才有系统文件访问的权，其他应用程序若要访问文件必须要向系统申请权限。

⑤内存管理机制，Linux 的进程管理机制是在进程活动停止后结束进程，但是在 Android 平台上，则使尽可能把进程留在内存中，一直到内存不够才会根据 “oom\_adj” 的值决定结束哪个进程。Android 将进程分为前台进程、可见进程、次要服务、后台进程、

内容供应节点、空进程 6 种进程，其中前台进程、可见进程、次要进程不会被终止。内容供应节点在由于内存原因需要终止时具有较高的优先权。空进程是程序结束后在进程中存留的没有数据的空进程，系统的内存低于某个界限时会自动关闭空进程。

## 2.5 Android 系统安全机制的缺陷

Android 操作系统存在以下漏洞和问题：

①签名机制管理松散。为了提高 Android 系统的开放性，Google 采用了数字自签名的方法，这种宽松的签名证书意味着不用数字证书认证机构授权，可以随意发布应用程序<sup>[17]</sup>，意味着恶意程序也能随意发布至网上，因此导致恶意程序泛滥。

②应用程序权限控制只针对单个应用程序。应用程序权限控制针对单个应用程序能有效地防止权限越级，但是多个应用在组合以后，能轻易提升权限，达到隐式权限提升<sup>[18]</sup>，使得恶意代码获得高级权限，导致信息泄露、系统受到破坏，这也是目前很多恶意代码获得运行的主要途径。

③缺乏对权限的动态运行时监测<sup>[19]</sup>。权限一经被授权给应用程序后，在应用程序的生命期间，它将不会被移除，即使声明此权限源程序被删除。



## 3 技术背景

### 3.1 反编译工具 Apktool

APKTool 是 GOOGLE 提供的 APK 反编译工具，能够反编译及回编译 apk，同时安装反编译系统 apk 所需要的 framework-res 框架，清理上次反编译文件夹等功能。

#### ①环境配置

- 安装 JAVA。
- 配置 JDK 和 JRE。
- 测试：运行 CMD（开始，运行，输入 cmd，回车），输入 java -version，回车，如出现图 3-1 所示的 JDK 版本，那就是已经安装成功了。

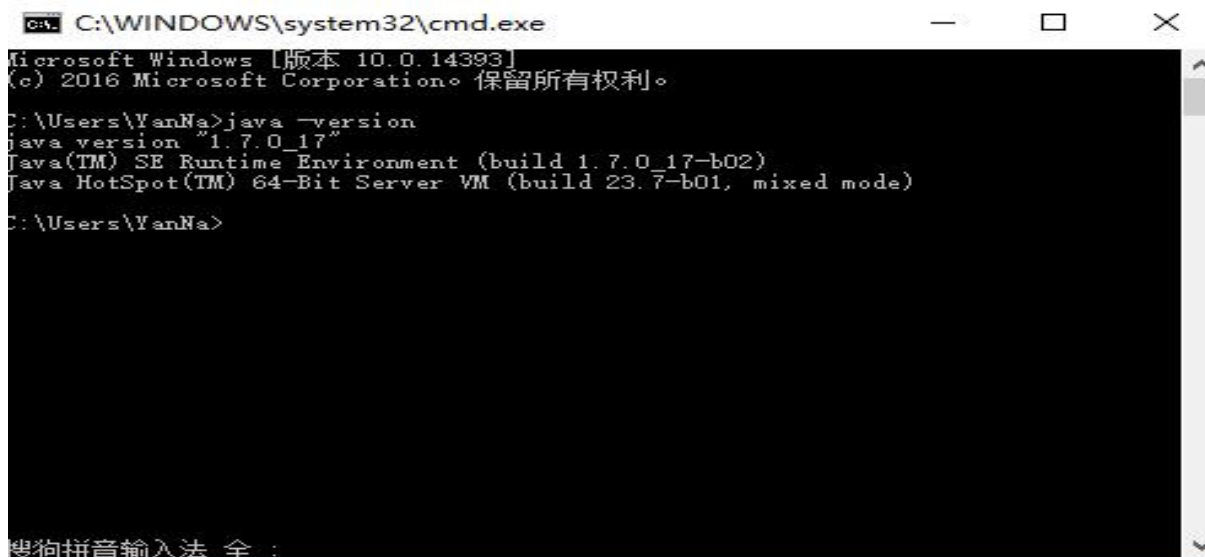


图 3-1 查看 JDK 版本

#### ②命令

- decode

该命令用于进行反编译 apk 文件，一般用法为 `apktool d D:\apkset\my.apk D:\apktoolresult`，D:\apkset\my.apk 代表了要反编译的 apk 文件的路径。D:\apktoolresult 代表了反编译后的文件的存储位置。

如果指定的路径中文件夹已经存在，那么输入完该命令后会有提示信息，并且无法执行，可以修改命令加入 -f 指令 `apktool d -f` 这样就会强行覆盖已经存在的文件。

- build

该命令用于编译修改好的文件，一般用法为 `apktool b D:\apktoolresult, D:\apktoolresult` 代表修改后的文件路径，可以是之前反编译的文件存放路径，输入这行命令后，如果一切正常，D:\apktoolresult 内会出现 dist 文件夹，里边存放已打包未签名的 apk。

## 3.2 smali 语法介绍

Smali 是 Dalvik 虚拟机指令语言，在使用 Apktool 反编译 apk 文件成功后，会在输出目录下自动生成一个 smali 文件夹，其中存放着所有反编译出的 smali 文件，这些文件会根据程序包的层次结构生成相应的目录，程序中所有的类都会在相应的目录下生成独立的 smali 文件。Smali 语言相当于一种中间语言，介于 java 源代码与 dex 可执行文件中间，由于 dex 可执行文件很难阅读，所以反编译生成 smali 语言，可以有助于更好的理解源代码。Smali 的数据类型如表 3-1 所示。

表 3-1 smali 数据类型

基本数据类型	语法	含义
	V	Void,无返回值类型
	Z	boolean
	B	byte
	S	short
	C	char
	I	int
	J	long
	F	float
	D	double
	L	Java 类类型
	[	数组类型

续表 3-1

对象	Lpackage/name/ObjectName;	相当于 java 中的 package.name.ObjectName。L 表示这是对象类型, package/name 该对象所在的包, ;表示对象名称的结束
数组	单维整形数组用[I 来表示, 每增加一个维度就增加一个[	[I 表示 int[], [[C 表示 char[][]
方法	采用类似于 Lpackage\name\ObjectName;->MethodName(III)Z 的形式	对象调用了 boolean 类型的 MethodName(int,int,int)
变量	采用类似于 Lpackage\name\ObjectName;->FileName:Ljava\lang\String;	表示对象类型为 string 类型, 名称为 FileName
寄存器	V 命名方式和 P 命名方式, P 表示参数寄存器。V 表示本地寄存器	指令 .registers 指定方法中寄存器的总数, 指令 .locals 表明方法中非参数寄存器的数量, 当一个方法被调用时, 方法的 K 个参数被置于最后 K 个寄存器中, 而非静态方法中的第一个参数总是调用该方法的对象

### 3.3 apk 签名机制

所有 Android 应用在发布时都要被打包成.apk 文件, APK 文件在发布时都必须进行签名, 这与在信息安全领域中所使用的数字证书的用途有所不同, 它是应用包用来进行自我认证的一种机制。它不需要权威的数字证书签名机构的认证, 而是由开发者自己来进行数字证书的使用和控制。Android 系统利用数字签名来对应用程序的作者进行标识, 并在应用程序间建立一种相互的信任关系, 而不是用来判定应用程序是否应该被安装。

对 Android 应用签名可以采用调试模式和发布模式: 使用 Android 开发工具(命令行为和 Eclipse 等)开发的应用是用一个调试私有密钥自动签名的, 这些应用被称为调试模式应用, 这些应用只能用于开发者自行测试, 不能够发布到 Google Play 官方应用市场上。当开发者需要在 Google Play 市场上将自己的应用程序发布时, 就必须用安卓的签名机制使用其私有密钥签署应用, 在应用安装时对其进行验证, 用以实现对应用程序安装时的来源鉴定。

为了便于安装, 安装时需要将应用程序打包成.apk文件。 .apk 文件中包含了应用程序的.dex文件以及其它的非代码资源。Android要求所有的应用程序(代码和非代码资源)

都进行数字签名，从而使应用程序的作者对该应用负责。只有在证书有效的情况下，且签名通过了公钥的验证，那么签名后的.apk 文件才是有效的。

### 3.4 Monkey 随机测试

Monkey 顾名思义就是像猴子一样任意对设备输入事件，它是由 Java 语言编写的并由“monkey”来启动执行，通过 Monkey 测试可以模拟用户滑动轨迹球、敲击键盘、触摸屏幕等动作对设备上的程序进行随机和压力测试。在使用 Monkey 测试之前必须搭建好 SDK 的环境和 JDK1.5 以上版本，然后在环境变量中把 adbtools 的目录添加到 path 里，就可以在命令行里对 adb 当前接入的 Android 设备进行测试了。

在 PC 机上打开 CMD 命令提示符，输入 adb shell monkey+{命令参数}即可开始测试。常用命令参数如表 3-2 所示。

表 3-2 monkey 命令常用参数

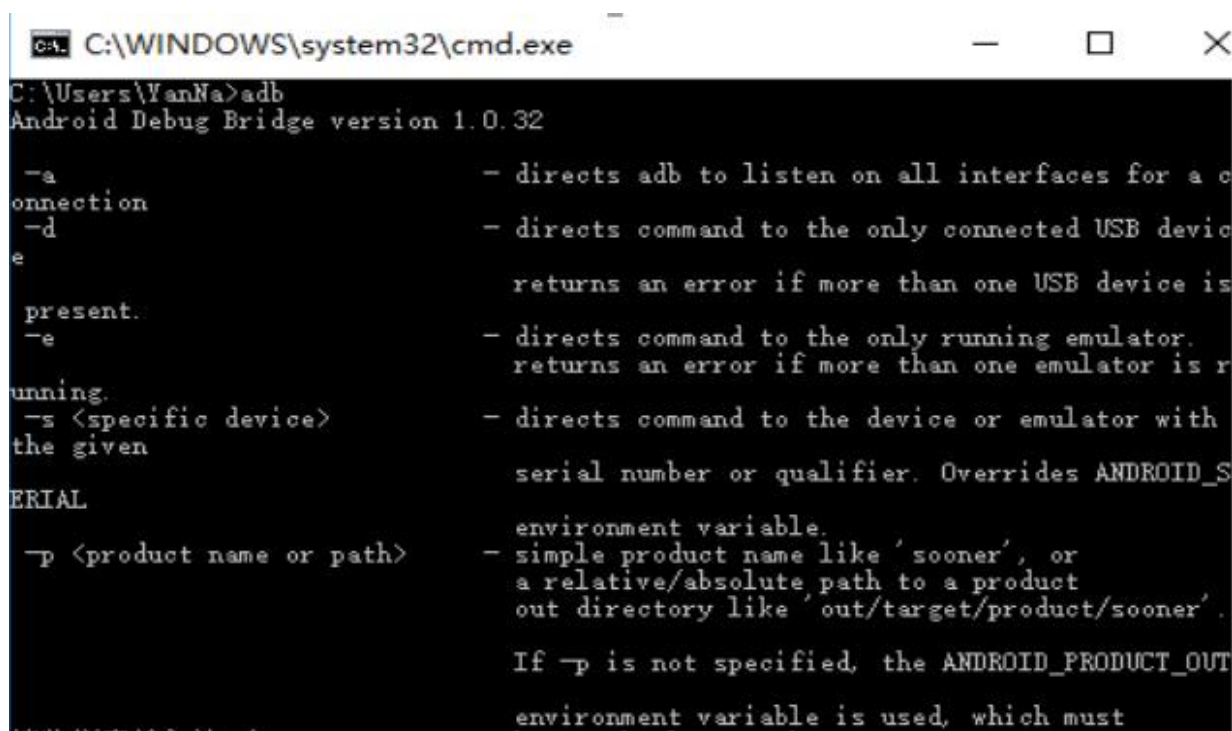
基础参数	-v	用于指定反馈信息级别, level 0: -v, level 1: -v-v, level 2: -v-v-v
	-s	随机数种子, 指定伪随机生成器的 seed 值, 若 seed 值相同则两次测试的事件序列也相同
	-throttle	用于指定用户操作的时延, 单位是毫秒
	-p	指定要测试的一个或多个包名
发送的事件类型	--pct-touch	调整触摸事件的百分比
	--pct-motion	调整动作事件的百分比
	--pct-trackball	调整轨迹事件的百分比
	--pct-nav	调整“基本”导航事件的百分比
	--pct-syskeys	调整“系统”按键事件的百分比
	--pct-anyevent	调整其它类型事件的百分比
调试选项	--ignore-crashes	用于指定当应用程序崩溃时, Monkey 是否停止运行
	--ignore-timeouts	用于指定当应用程序发生 ANR 错误时, Monkey 是否停止运行
	--ignore-security-exceptions	用于指定当应用程序发生许可(如证书许可, 网络许可等)错误时, Monkey 是否停止运行
	--kill-process-after-error	用于指定发生错误时, 停止运行程序
	--monitor-native-crashes	用于指定是否监视并报告应用程序发生崩溃的本地代码

### 3.5 批处理

批处理是应用于 DOS 和 Windows 系统中的一种类似于 Unix 的 shell 脚本的简化脚本语言，它通常是由 COMMAND.COM 或者 CMD.EXE 解释运行的。批处理文件的扩展名为 .bat 或者 .cmd，批处理文件也叫批处理程序，是由一条条的 DOS 命令组成的普通文本文件，可以用 DOS 命令创建，可以用记事本直接编辑，也可以用 EDIT.EXE 来编辑。在“命令提示符”下输入 bat 文件的名称，也可以双击某个批处理文件，系统就会调用 CMD.EXE 运行这个批处理程序。系统要解析并运行某个批处理文件时，首先会扫描整个批处理文件，如果没有遇见 exit 命令或者发生错误而意外退出，就会从第一行命令开始往下依次执行，一直到程序的结尾。

### 3.6 ADB 调试桥

Android 调试桥是指 adb.exe 工具（Android Debug Bridge，ADB），存在于 SDK 的 platform-tools 目录中，允许开发人员与模拟器或者连接的 Android 设备进行通信。为了使用该指令快速完成某项操作需要将 adb.exe 所在的目录配置到 path 环境变量中，之后就可以在命令行窗口中使用了，如图 3-2 所示。



```
CA C:\WINDOWS\system32\cmd.exe
C:\Users\VanNa>adb
Android Debug Bridge version 1.0.32

-a          - directs adb to listen on all interfaces for a connection
-d          - directs command to the only connected USB device
            returns an error if more than one USB device is present.
-e          - directs command to the only running emulator.
            returns an error if more than one emulator is running.
-s <specific device> - directs command to the device or emulator with the given
            serial number or qualifier. Overrides ANDROID_SERIAL
-p <product name or path> - simple product name like 'sooner', or
            a relative/absolute path to a product out directory like 'out/target/product/sooner'.
            If -p is not specified, the ANDROID_PRODUCT_OUT
            environment variable is used, which must
```

图 3-2 adb 命令

ADB 的常见指令介绍如下：

- adb start-server：开启 adb 服务。
- adb devices：列出所有设备。
- adb logcat：查看日志。
- adb kill-server：关闭 adb 服务。
- adb shell：挂载到 Linux 的空间。
- adb -s<模拟器名称>install<应用程序（加扩展名）>：安装应用程序到指定模拟器。

## 4 基于自动测试的应用程序行为抽取系统的设计实现

### 4.1 系统总体框架

本系统的实现思路是：首先利用 APKTOOL 工具对需要分析的 apk 包进行反编译，得到其 smali 代码，然后对 smali 代码进行日志插桩，即遍历程序中所有的 smali 文件，定位到敏感 API，在其有敏感 API 调用的代码的下一行注入 smali 语法格式的日志代码段，然后利用 APKTOOL 工具回编译，得到新的未签名的 apk，未签名的 apk 不能运行在安卓设备上，所以还需要利用 signAPK 这个工具进行签名，签名完成之后在命令提示符中用命令的方式将签名之后的 apk 安装在安卓模拟器上，利用 Android ADB 自带的随机测试工具 MONKEY 进行测试，这时就可以得到需要分析的 apk 在运行过程中调用了哪些敏感 API。具体过程如图 4-1 所示。

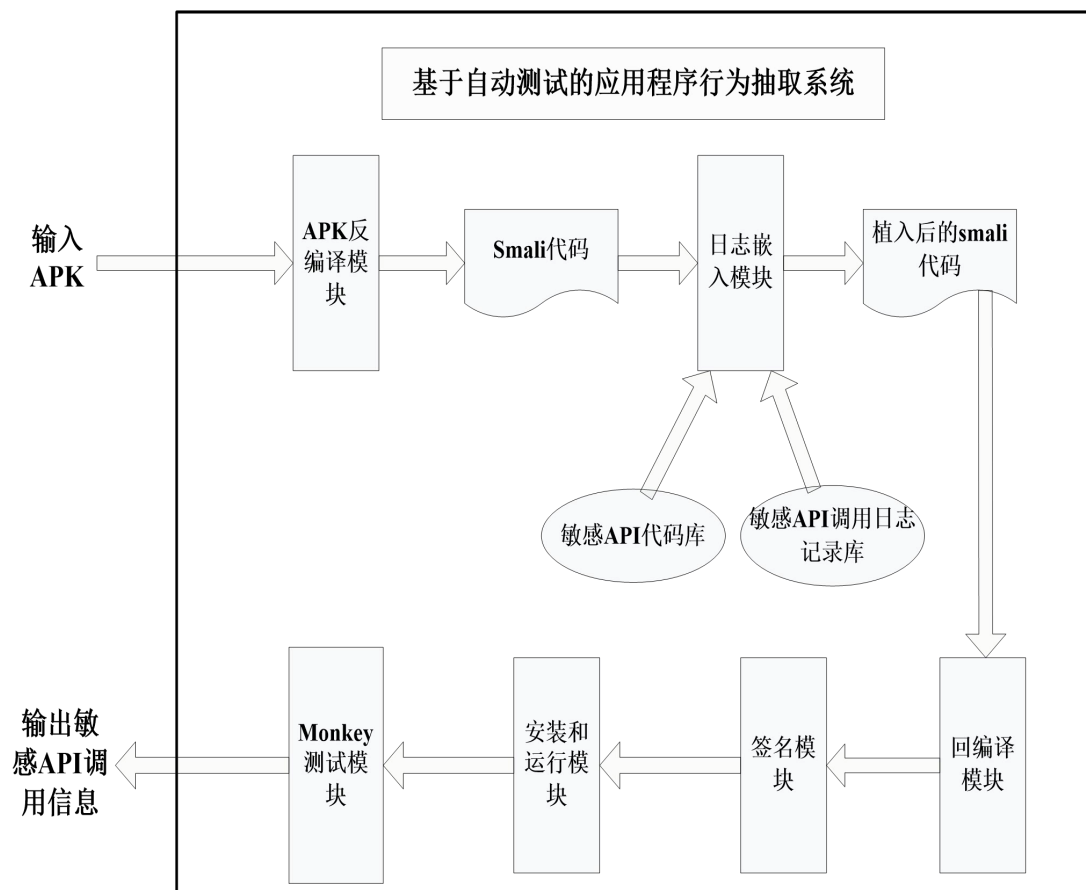


图 4-1 系统总体架构图

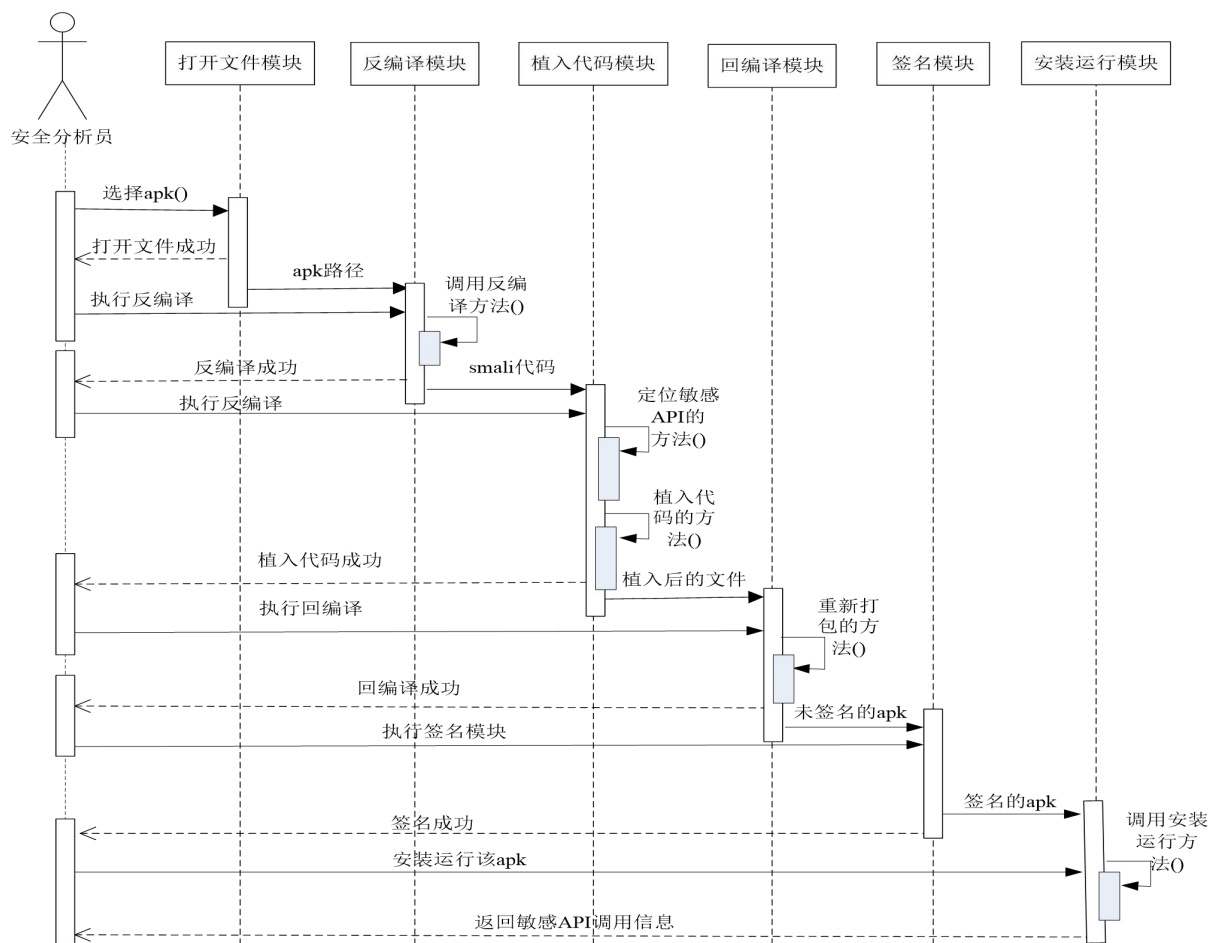


图 4-2 系统序列图

本系统用 Java 语言开发，本系统的界面展示如图 4-3 所示，本系统将 apk 的日志插桩过程封装成了一个简易的小工具，用户只需要按步骤执行即可实现查看 apk 中调用的敏感 API，工具栏按钮依次是打开文件、反编译、植入日志代码、回编译、签名、运行并显示数据结果。



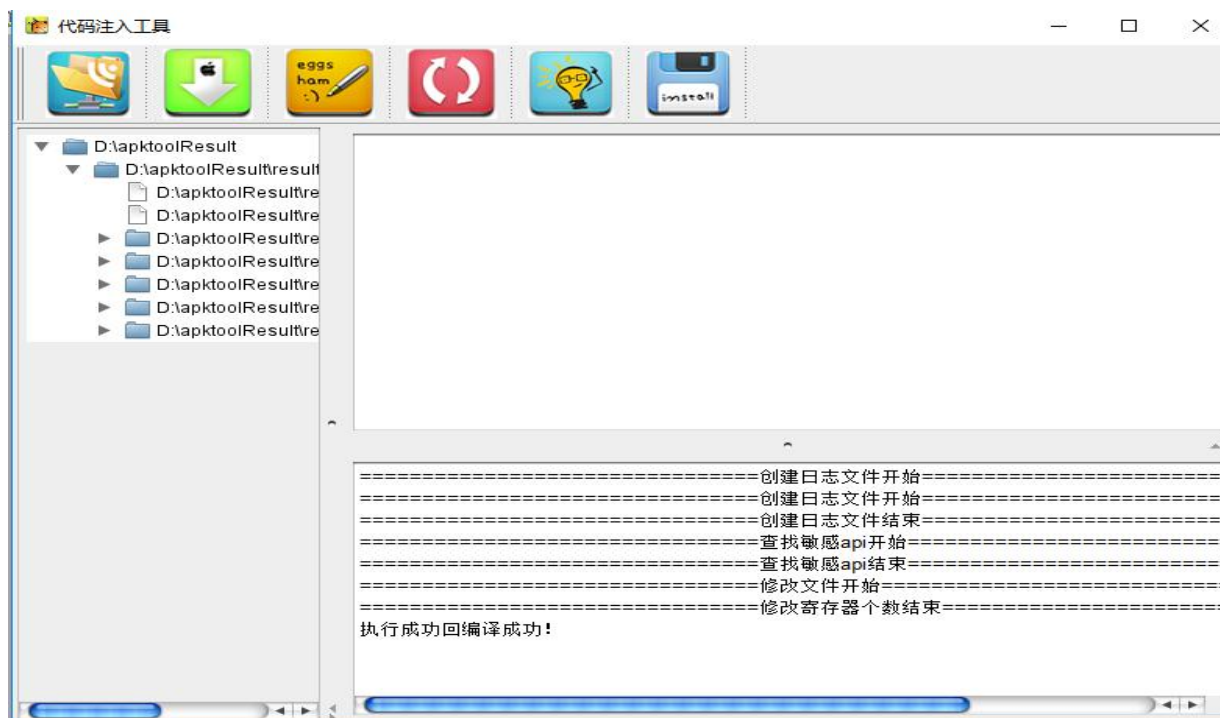


图 4-3 系统界面图

本工具使用说明：本工具是基于 Java swing 设计开发的，用户在使用过程中必须严格按照顺序执行，即在本代码注入工具中从工具栏中最左边的打开文件按钮开始到最右边的安装运行按钮结束，执行过程中必须等上一步执行结束才能开始下一步，具体过程为：点击打开按钮选择感兴趣的 apk->点击反编译按钮将该 apk 中的 dex 文件反编译为 smali 文件->点击植入代码按钮在 smali 文件中插入监听的日志代码->点击回编译按钮将植入后的文件打包成 apk->点击签名按钮对通过回编译得到的 apk 进行签名->点击安装运行按钮将该 apk 安装到海马玩模拟器中然后运行该 apk，将调用的敏感 API 的调用信息打印在命令行中。

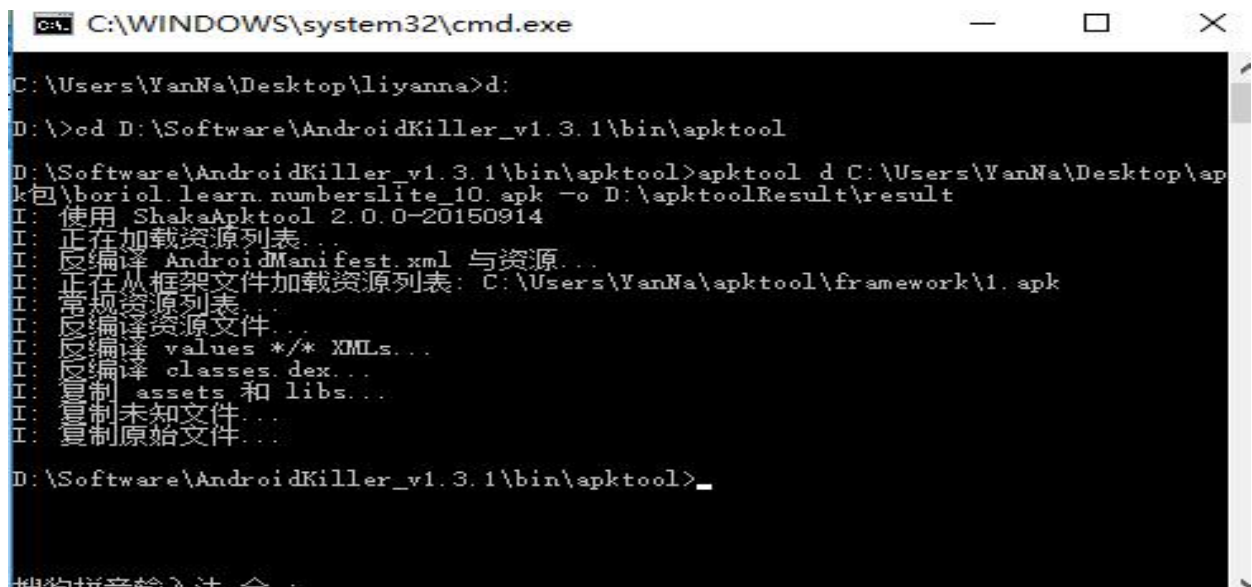
## 4.2 反编译模块

该模块将用户输入的 apk 包中的 dex 文件反编译成 smali 文件，此模块使用到了 Google 公司的 apktool 工具，在这里将命令封装成一个批处理文件，用 Java 中的方法获取用户输入的 apk 包的名称，以参数形式传递给批处理文件，此模算法块伪代码如表 4-1 所示。

表 4-1 反编译模块算法伪代码

```
Procedure backcompile( )  
apkpath=要反编译的 apk 存放路径  
file=反编译结果的存放路径  
my.bat=apktool d apkpath -o file  
exec(my.bat)  
Output(反编译 over!)
```

如图 4-4 为反编译模块的批处理命令执行过程。



```
C:\WINDOWS\system32\cmd.exe  
C:\Users\YanNa\Desktop\liyanna>d:  
D:\>cd D:\Software\AndroidKiller_v1.3.1\bin\apktool  
D:\Software\AndroidKiller_v1.3.1\bin\apktool>apktool d C:\Users\YanNa\Desktop\apk包\boriol.learn.numberslite_10.apk -o D:\apktoolResult\result  
I: 使用 ShakaApktool 2.0.0-20150914  
I: 正在加载资源列表...  
I: 反编译 AndroidManifest.xml 与资源...  
I: 正在从框架文件加载资源列表: C:\Users\YanNa\apktool\framework\1.apk  
I: 常规资源列表...  
I: 反编译资源文件...  
I: 反编译 values ** XMLs...  
I: 反编译 classes.dex...  
I: 复制 assets 和 libs...  
I: 复制未知文件...  
I: 复制原始文件...  
D:\Software\AndroidKiller_v1.3.1\bin\apktool>_
```

图 4-4 反编译命令

执行完批处理程序后会在指定的目录下生成一系列文件和文件夹，如图 4-5 所示。

此电脑 > LENOVO (D:) > apktoolResult > result				
名称	修改日期	类型	大小	
assets	2017/5/27 15:13	文件夹		
build	2017/5/27 15:13	文件夹		
dist	2017/5/27 15:14	文件夹		
lib	2017/5/27 15:13	文件夹		
original	2017/5/27 15:13	文件夹		
res	2017/5/27 15:13	文件夹		
smali	2017/5/27 15:13	文件夹		
unknown	2017/5/27 15:13	文件夹		
AndroidManifest.xml	2017/5/27 15:13	XML 文档	4 KB	
apktool.yml	2017/5/27 15:13	YML 文件	1 KB	

图 4-5 反编译生成文件示意图

### 4.3 日志代码植入模块

该模块在反编译产生的 smali 文件夹中通过如图 4-6 的方法创建一个 logutil 文件夹，在此文件夹中生成 Logutil 文件，文件中定义了一个 log()方法，将 smali 文件逐一遍历，定位敏感 API，如图 4-7 为判断是否含有敏感 API 的方法，然后判断其所在的方法体的第一行.locals 后面的 int 型数是否为 0，若为 0 则给它加 1，否则直接将敏感 API 的名字存入 V0 寄存器中，再插入 Logutil 类中的 log()方法，并将 V0 中的数据作为实参传递给 V()方法，这样就可以通过 Android 中的 logcat 进行查看，本模块算法伪码如表 4-2 所示，原文件及插入代码后的文件如图 4-8 所示。

表 4-2 注入代码模块算法伪码

```

//创建日志类
Logpath=反编译的 smali 文件路径\\logutil\\Logutil.smali
p=敏感 API
begin
Procedure createlogfile( logpath)
If logpath not exist
Call mkdir() //创建父目录
Call createNewFile() //创建 logpath 文件
Else call write(log()) //调用 File 类中的 write()方法
Procedure smali 格式语言 log(p)

```

续表 4-2

Call android.util.log.d(“liyanna”,p)
End if
//定位敏感 API
for temp=smali 文件第一行 to 最后一行
line++
If (isHasSAPI(temp)=true)
Then if (.locals 0)
Then .locals 0←.locals 1
End if
将 line,敏感 API 存入集合 list
End if
//重写 smali 文件
Procedure writeContent()
Sort(line)//将集合中的 line 按从大到小排序
ReadLine()
If (line=集合中的 line)
Then write(log(API)) //在对应的行写入 log 方法，将 API 字符串作为参数传递给 log
End if

对 atticlab.bodyscanner.apk 进行注入，其中 WakefulBroadcastReceiver.smali 文件中的 startWakefulService()方法注入日志代码前后对比表 4-3 和 4-4 所示。

表 4-3 注入代码前的 startWakefulService() 方法

.method public static
(Landroid/content/Context;Landroid/content/Intent;)Landroid/content/ComponentName;
.locals 8
.param p0, "context"      # Landroid/content/Context;
.param p1, "intent"      # Landroid/content/Intent;
.prologue
.line 81
sget-object v5,
Landroid/support/v4/content/WakefulBroadcastReceiver;->mActiveWakeLocks:Landroid/util/SparseArray;
monitor-enter v5

续表 4-3

```
.line 82
:try_start_0
sget v1, Landroid/support/v4/content/WakefulBroadcastReceiver;->mNextId:I
.end method
```

表 4-4 注入代码后的 startWakefulService() 方法

```
.method public
static(Landroid/content/Context;Landroid/content/Intent;)Landroid/content/ComponentName;
.locals 8
const-string v0,"Landroid/content/Context;
                ->getService(Ljava/lang/String;)Ljava/lang/Object;"
invoke-static {v0},Llogutil/LogUtil;->log(Ljava/lang/String;)V
.param p0, "context"    # Landroid/content/Context;
.param p1, "intent"     # Landroid/content/Intent;
.prologue
.line 81
sget-object v5,
Landroid/support/v4/content/WakefulBroadcastReceiver;->mActiveWakeLocks:Landroid/util/SparseArray;
monitor-enter v5
.line 82
:try_start_0
sget v1, Landroid/support/v4/content/WakefulBroadcastReceiver;->mNextId:I
.end method
```

## 4.4 回编译模块

该模块将已经注入了日志代码的 smali 文件回编译成 dex 文件并重新打包成 APK，但值得注意的是此时的 APK 未经签名，不被允许安装在 Android 平台。利用 GOOGLE 公司的 apktool 工具进行回编译的执行过程如图 4-6 所示，本模块算法伪代码如表 4-5 所示。

表 4-5 回编译模块算法伪码

Procedure recompile( )
file=上一模块反编译结果的存放路径
outfile=回编译结果要保存的路径
huibian.bat=apktool b file outfile
exec(huibian.bat)
Output(回编译 over!)

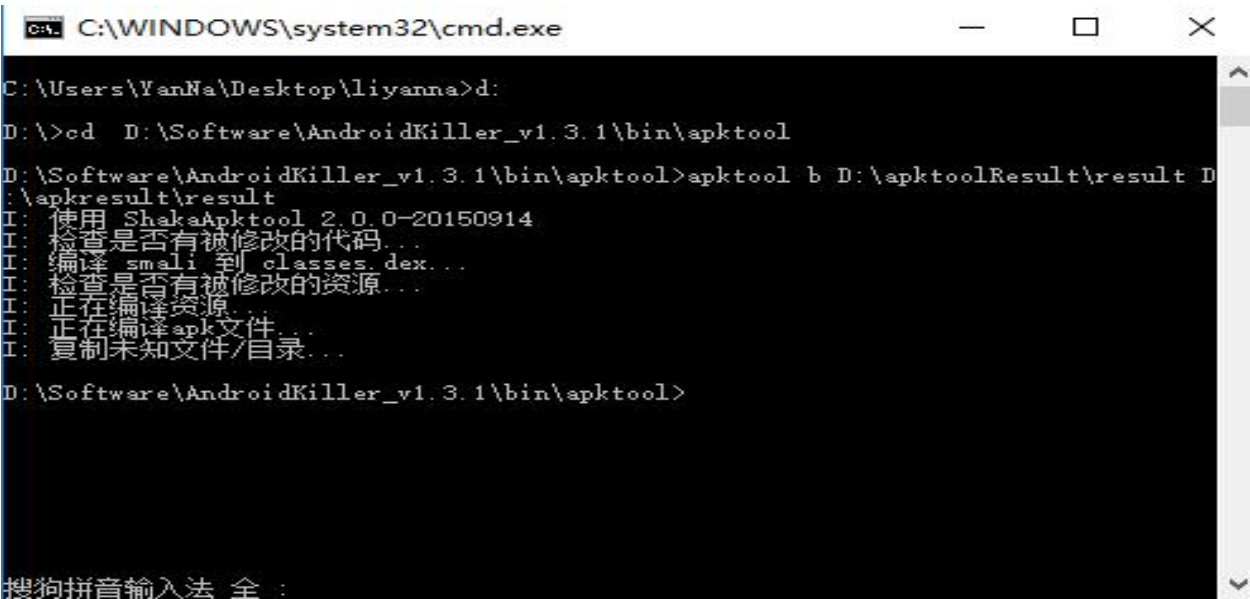


图 4-6 回编译执行过程

回编译之后会在 dist 目录下生成一个未签名的 apk 包，如图 4-7 所示。



图 4-7 未签名 apk 生成目录

## 4.5 签名模块

该模块对回编译模块得到的未签名的 apk 进行签名，使用 Android 平台自带的 signAPK.jar 工具，并使用 Openssl 来创建私钥/公钥对，如图 4-8 所示，然后用命令 `java -jar signapk.jar certificate.pem key.pk8 your-app.apk your-signed-app.apk` 即可进行签名，本模块算法伪码如表 4-6 所示。

 signapk.jar	2009/11/1 12:36	Executable Jar File	8 KB
 testkey.pk8	2008/11/5 15:17	PK8 文件	2 KB
 testkey.x509.pem	2008/11/5 15:17	PEM 文件	2 KB

图 4-8 签名工具和密钥

表 4-6 签名模块算法伪码

```
Procedure signAPK()  
dist=回编译的 apk 存放路径  
signapkname=未签名的 apkname+"_sign.apk"  
qmapkpath=签名后的 apk 存放路径+"\"+signapkname  
qianming.bat=java -jar signapk.jar testkey.x509.pem testkey.pk8 dist qmapkpath  
exec(qianming.bat)  
Output(签名 over!)
```

给 APK 签名的执行过程如图 4-9 所示。

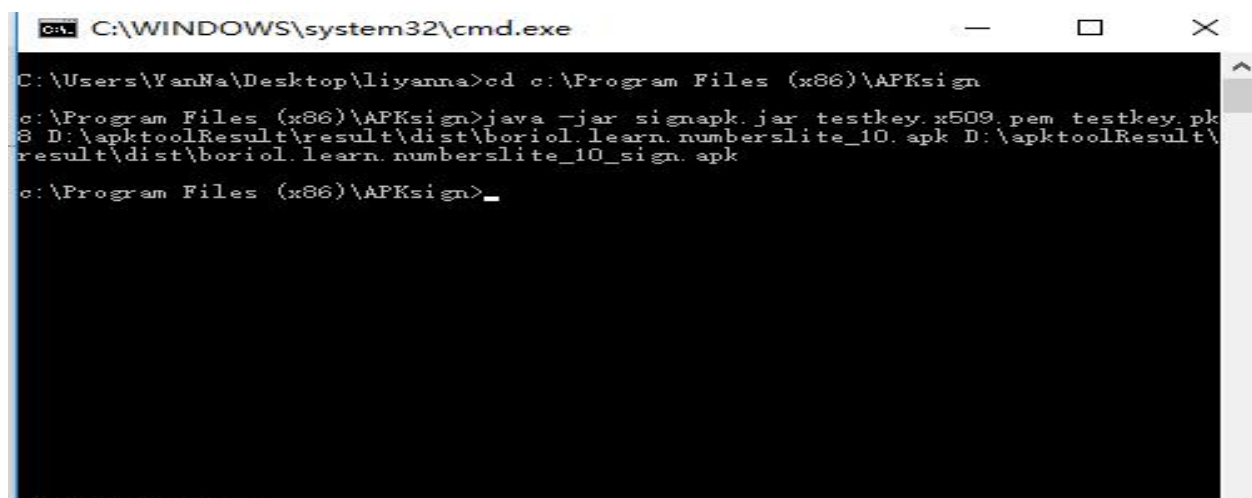


图 4-9 签名的执行过程

在 dist 目录下生成签名后的 APK，如图 4-10 所示。

占板	组织	新建	打开	选择
> 此电脑 > LENOVO (D:) > apktoolResult > result > dist				
名称	修改日期	类型	大小	
 boriol.learn.numberslite_10.apk	2017/6/3 17:21	Droid4X apk file	11,899 KB	
 boriol.learn.numberslite_10_sign.apk	2017/6/3 19:22	Droid4X apk file	11,911 KB	

图 4-10 生成签名 APK 文件

## 4.6 安装、运行和显示敏感 API 调用模块

将签名后的 APK 安装在海马玩模拟器上，然后将该 APK 的包名传递给 monkey，用 adb shell monkey+包名即可启动该 APK 进行随机测试。当软件运行后会在 Android 的日志系统里打印出当前软件调用的敏感 API 信息。

本模块处理流程图如图 4-11 所示。

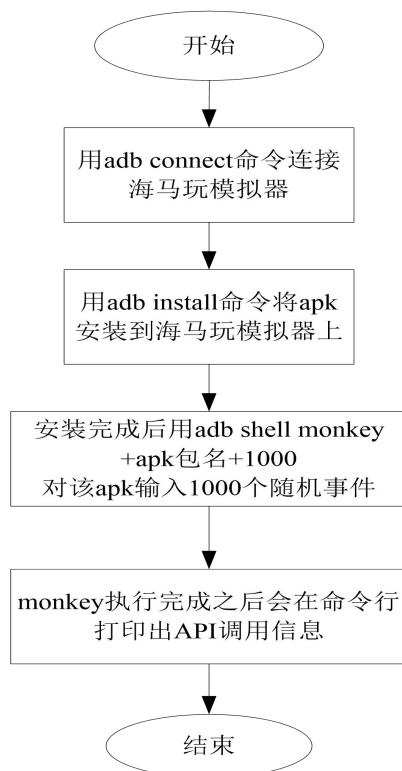


图 4-11 安装运行模块流程图



用 adb install 命令将 APK 安装到海马玩安卓模拟器中，如图 4-12 为植入代码之后的 numberslite.apk 的安装过程。

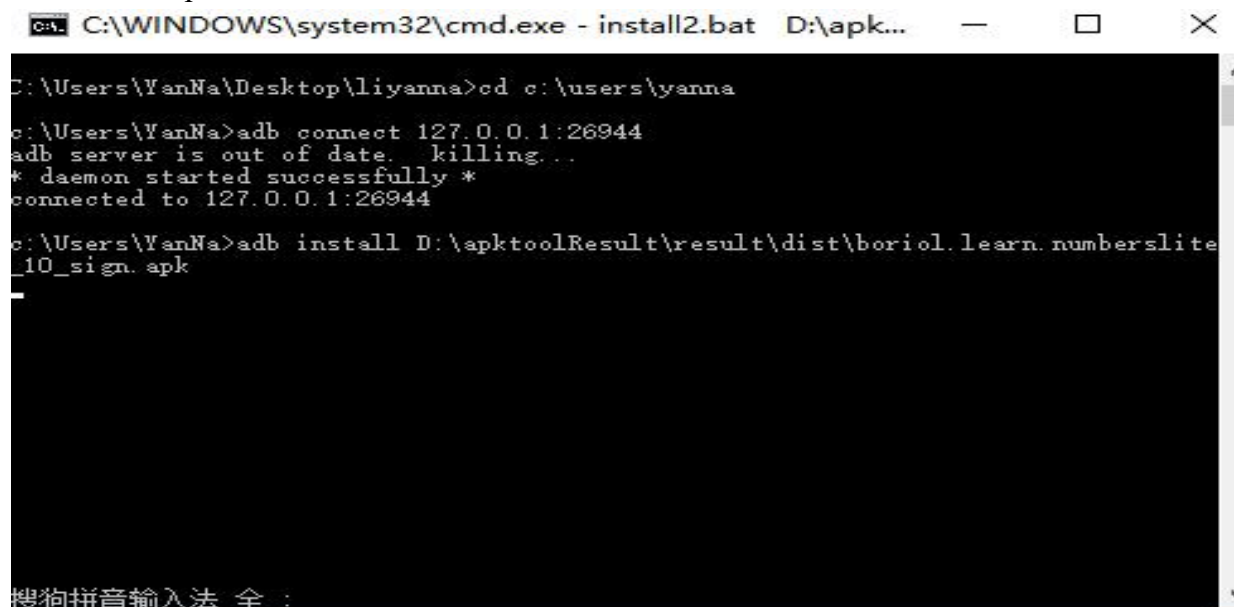


图 4-12 连接 Android 模拟器并安装 apk

用 monkey 命令在海马玩模拟器中启动 numberslite.apk,并输入了 1000 个随机事件，如图 4-13 所示。

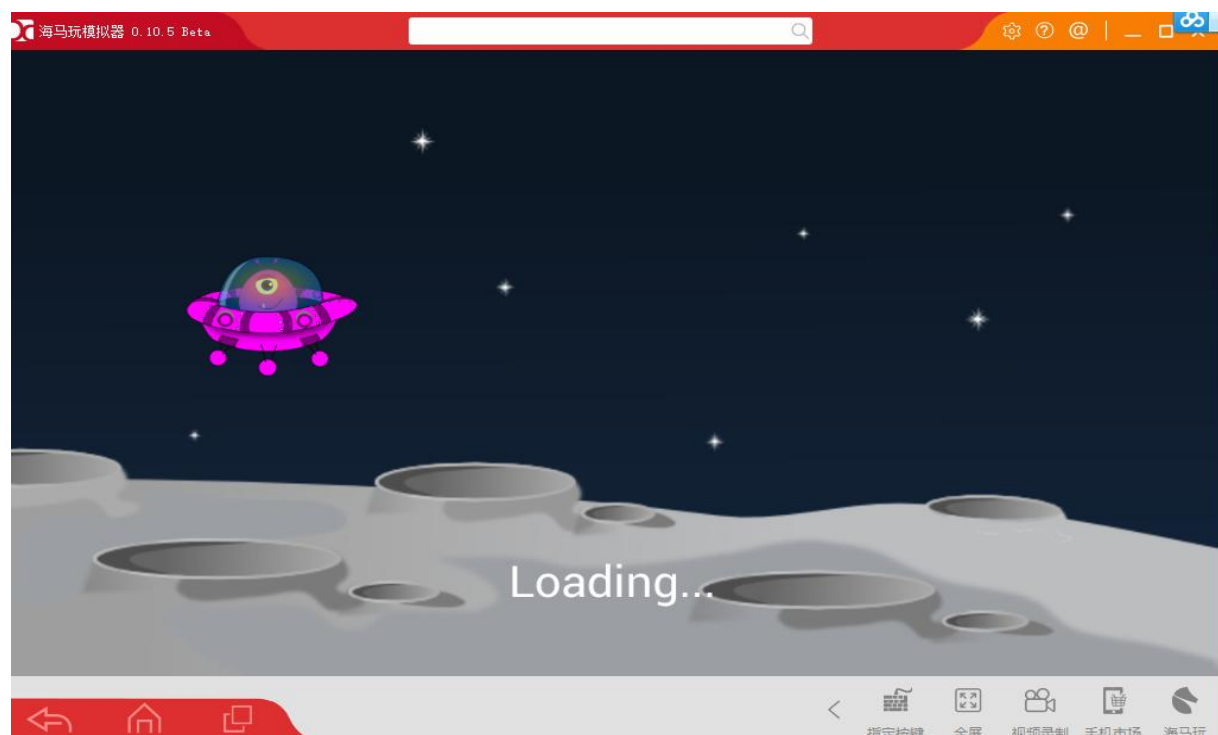


图 4-13 海马玩模拟器中启动 apk

该 APK 调用的敏感 API 打印在命令行中，如图 4-14、4-15、4-16、4-17 所示。

```
C:\WINDOWS\system32\cmd.exe - install2.bat D:\apktoolResult\result\dist\boriol.learn.numberslite_10_sign.apk boriol.learn.numberslite
c:\Users\YanNa>adb logcat -s liyanna
----- beginning of /dev/log/system
----- beginning of /dev/log/main
D/liyanna ( 2194): p1}
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->acquire()V
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->acquire()V
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): p1}
D/liyanna ( 2194): v6
D/liyanna ( 1816): v2
D/liyanna ( 1816): v2}
D/liyanna ( 1816): Landroid/media/Ringtone;->play()V
D/liyanna ( 1816): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): p1}
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->release()V
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->release()V
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->acquire()V
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->acquire()V
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): p1}
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->acquire()V
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->acquire()V
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): v6
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): p1}
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): v6
D/liyanna ( 1816): v2
D/liyanna ( 1816): v2}
```

图 4-14 敏感 API 调用结果

```
C:\WINDOWS\system32\cmd.exe - install2.bat D:\apktoolResult\result\dist\boriol.learn.numberslite_10_sign.apk boriol.learn.numberslite
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): p1}
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): v6
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): p1}
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): p1}
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): v6
D/liyanna ( 2194): v6
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): v6
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): p1}
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): p1}
```

图 4-15 敏感 API 调用结果

```

C:\WINDOWS\system32\cmd.exe - install2.bat D:\apktoolResult\result\dist\boriol.learn.numberslite_10_sign.apk boriol.learn.numberslite
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->acquire()V
D/liyanna ( 2194): Landroid/os/PowerManager$WakeLock;->acquire()V
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): v6
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
D/liyanna ( 2194): Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
D/liyanna ( 2194): Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z

```

图 4-16 敏感 API 调用结果



[illegible]

图 4-17 敏感 API 调用结果

## 4.7 测试与数据统计

通过对 8 个 APK 进行了测试，在海马玩上运行这些 APK 如图 4-18 所示，将每个 APK 中调用的敏感 API 进行统计，在每个饼状图中绘制出每个 API 所占比例。



图 4-18 海马玩模拟器中安装的已经插桩的 apk

下面为敏感 API 调用的统计结果，绘制成饼状图如图 4-19 至 4-26 下。

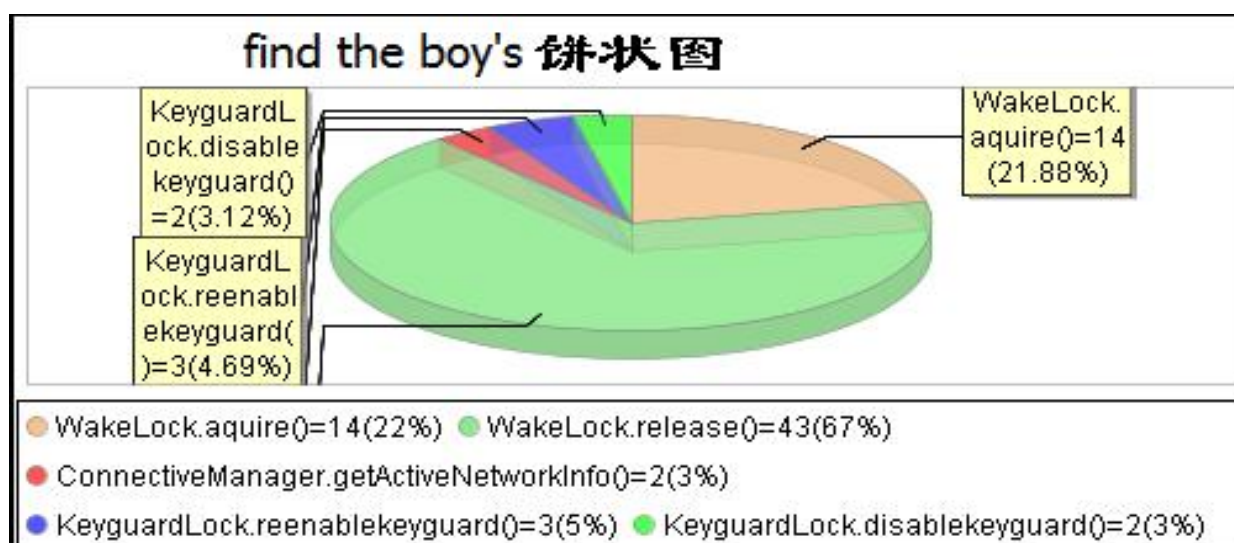


图 4-19 findboy' apk 中敏感 API 统计图

## Kids123free的饼状图

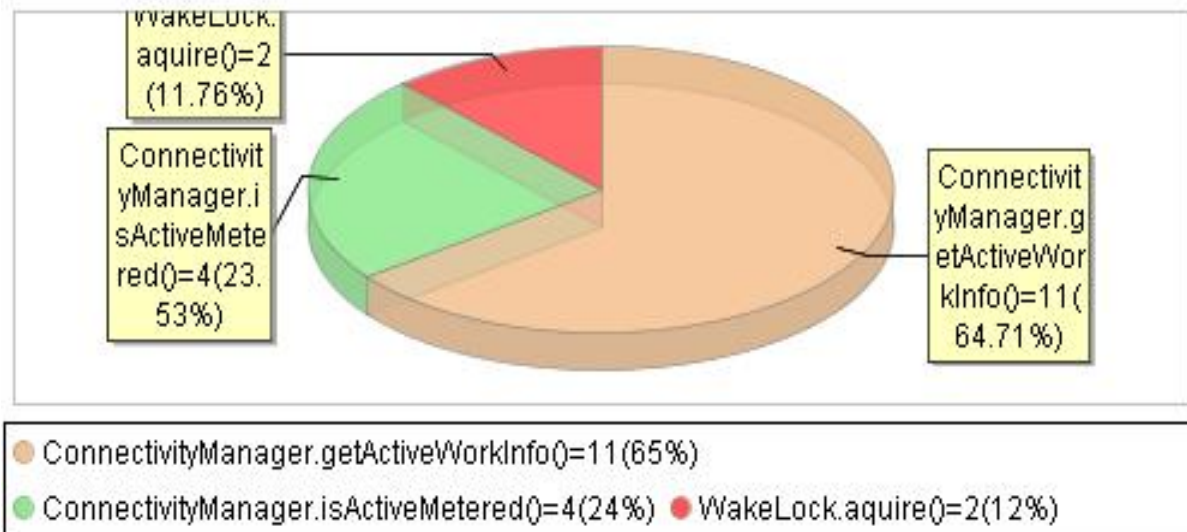


图 4-20 kids123free apk 中敏感 API 统计图

## Irunonline的饼状图

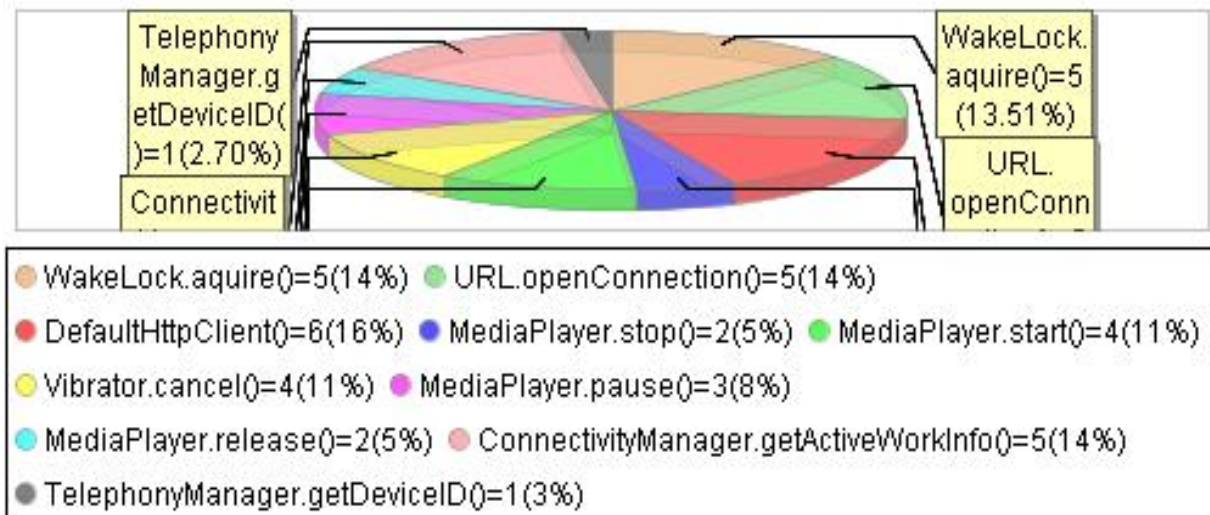


图 4-21 Irunonline apk 中敏感 API 统计图



## southernpoker的饼状图

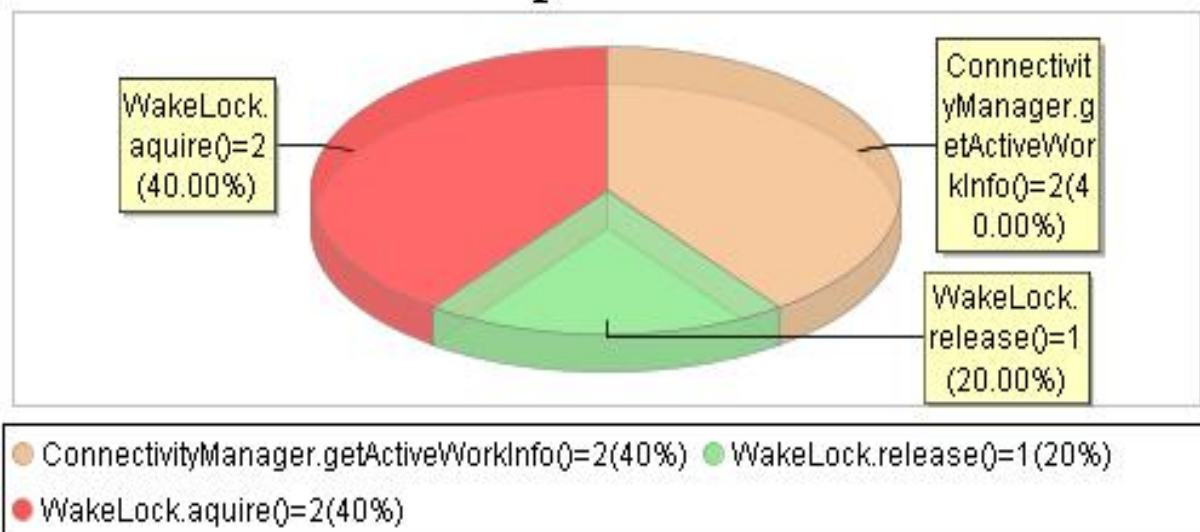


图 4-22 southernpoker apk 中敏感 API 统计图

## trickyshot的饼状图

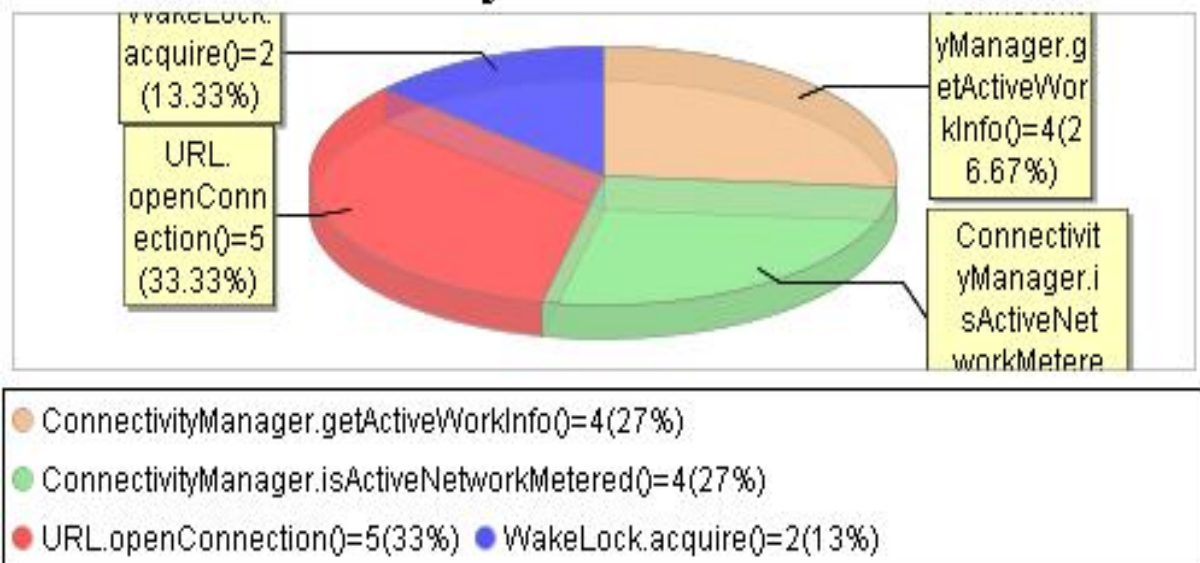


图 4-23 trickyshot apk 中敏感 API 统计图

## 俄罗斯方块的饼状图

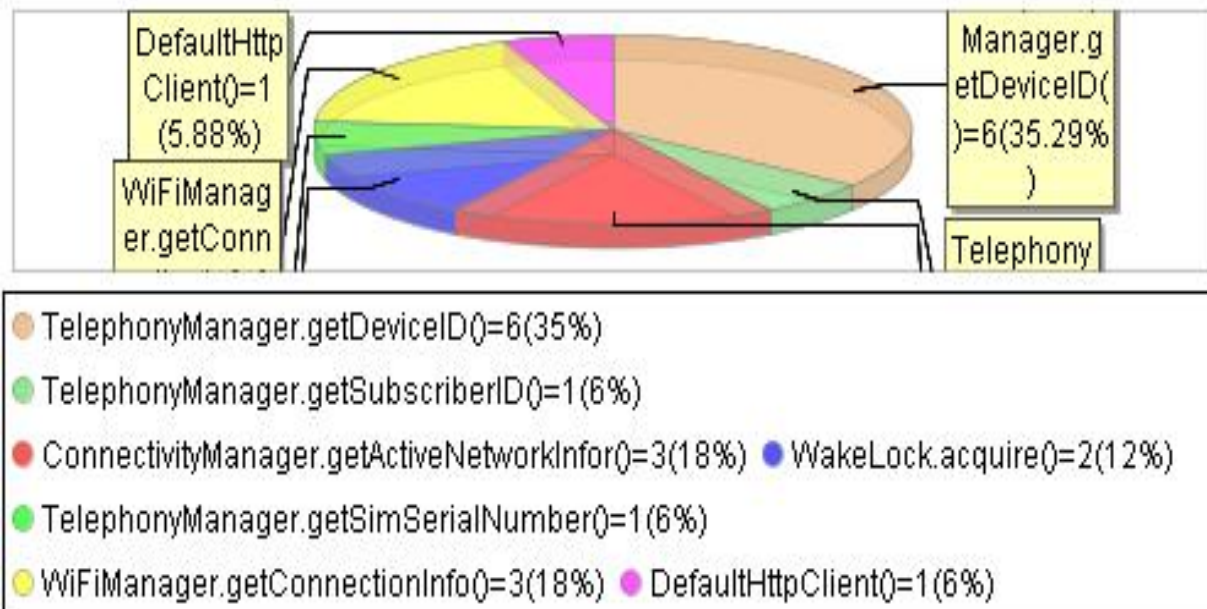


图 4-24 俄罗斯方块 apk 中敏感 API 统计图

## 消灭星星的饼状图

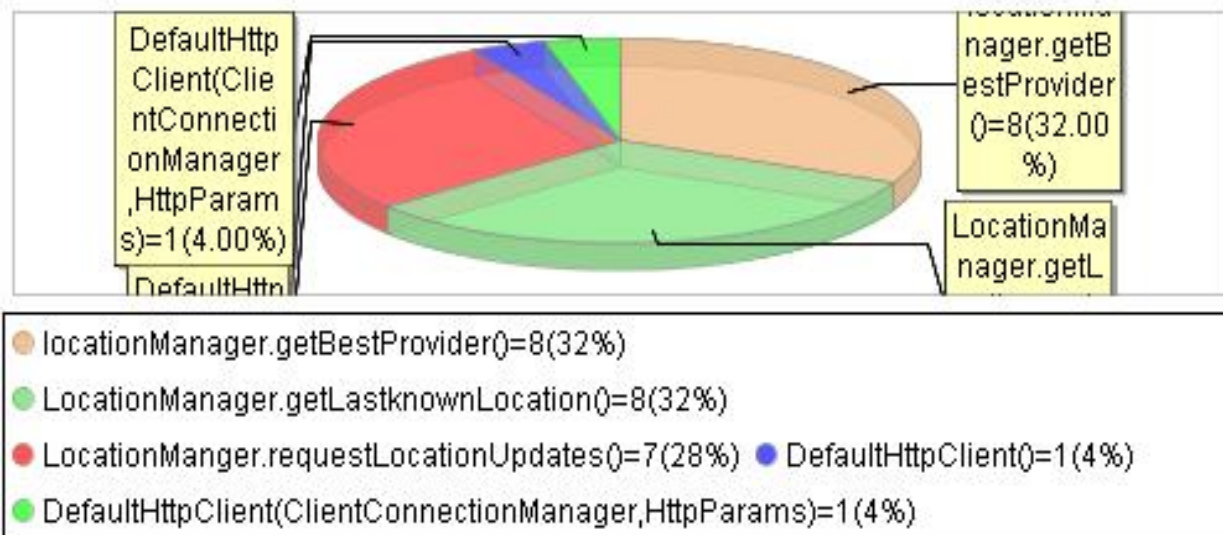


图 4-25 消灭星星 apk 中敏感 API 统计图



## yc tahobka的饼状图

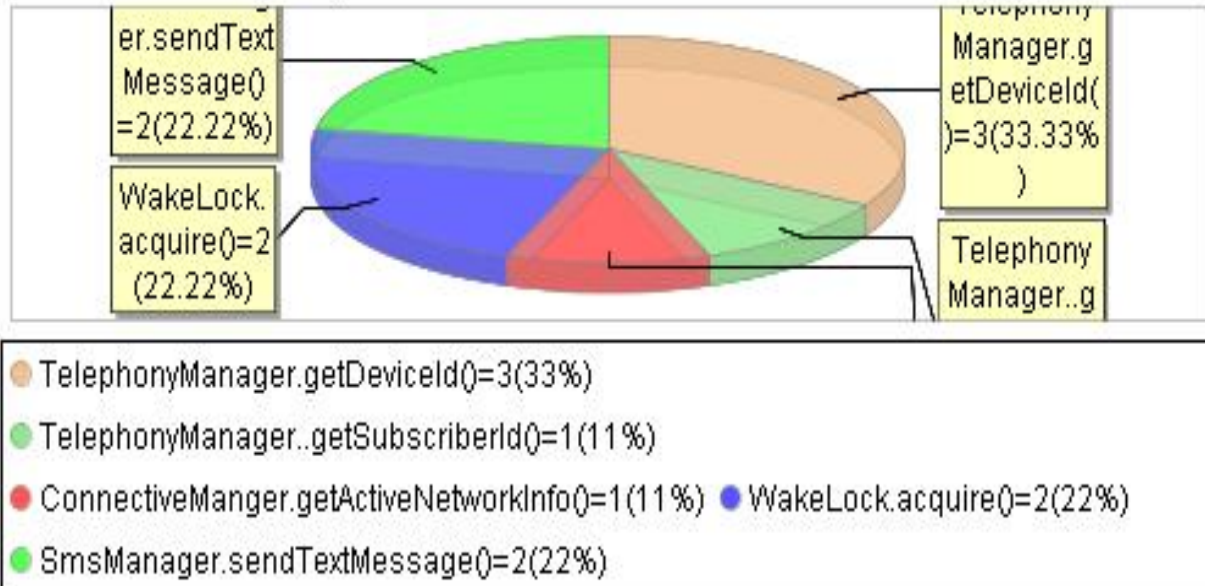


图 4-26 yctahobka apk 中敏感 API 统计图

## 5 结束语

本文首先介绍了 Android 平台的发展状况以及在发展过程中面临的安全威胁，引用网络资源的数据和图标对恶意 APK 的种类进行了分析，并对当前国际的研究现状进行了调研，由于恶意 APK 实现某些功能时往往调用了敏感 API，由此引出本课题的研究意义。在进行本课题开始时首先对 Android 系统的架构和 apk 包的结构进行了详细的介绍，分析了 Android 中的 Dalvik 虚拟机的原理，然后分析了 Android 系统的安全机制和安全机制存在的不足之处。之后介绍了本次毕业设计中所用到的技术，包括 apktool、smali 语法、apk 签名机制、monkey 随机测试、adb 调试桥等等。然后根据需求分析设计完成了该基于 monkey 自动测试的应用程序行为抽取系统，并对各个模块分别进行了详细的介绍，随后用八个实例进行了测试，统计了执行结果。本次毕设开发了基于 Java swing 的自动化工具，实现了对 apk 的反编译、注入代码、重新打包和签名、并用 Monkey 进行自动运行和测试这一完整过程的封装。经过通过实例测试验证了本系统的可行性和正确性。

## 致 谢

美好的大学生涯即将以这最后几个月的毕业设计而宣告结束，这是大学最后的作业。这四年的大学，因为学校，因为老师，我可以容易的学到教程要求的知识，因为学校给予我一个广阔自由的平台，老师的授课欢快又不失严肃，这种环境使我可以在学校安心的充实自己，丰富自己。同学校友间的融洽相处，互帮互助让我心里倍感到温暖。一个离家求学的人仿佛又找到了自己的新家如此的温暖。学会了如何去关心他人，帮助他人，正确的为人处事的方式积极乐观的生活态度。我还学会了学习一切重在自己的学习方法，这会让我在以后的工作中受益匪浅。

在这里感谢我的指导老师刘晓建老师一步一步耐心的指导和讲解，感谢我的同学雷倩在完成本次课题过程中给予我的帮助。同时，我也非常感谢我的室友和同学们，他们给了我很大的关心和支持。陪伴我走过了四年的大学生活。

最后，希望自己能够以这份毕设完美的结束自己的学生生涯，再用激情去迎接更加充满希望而美好的明天。

## 参考文献

- [1] 杨云君. Android 的设计与实现[M]. 北京: 机械工业出版社, 2013:45-49.
- [2] 李刚. 疯狂 Android 讲义[M]. 北京: 电子工业出版社, 2013:25-42
- [3] 张孝祥. Java 就业培训教程[M]. 北京: 清华大学出版社, 2007
- [4] Davi L, et al. Privilege escalation attacks on Android[C]//Information Security Conference-ISC, 2010:346-360
- [5] Bugiel S, et al. Towards Taming Privilege-Escalation Attacks on Android[C]//Information Security Conference-ISC, 2010:346-360
- [6] 路程. Android 平台恶意软件检测系统的设计与实现[D]. 北京: 北京邮电大学. 2012, 35-36
- [7] risksk. Android 沙盘原理与实现[OL].  
<http://security.tencent.com/index.php/blog/msg/7>, 2012.10.2
- [8] 童振飞, 杨庚. Android 平台恶意软件的静态行为检测[J]. 江苏通信科学与实践, 2011(12), 39-47
- [9] 丰生强. Android 软件安全与逆向工程[M]. 北京: 人民邮电出版社, 2013, 2, 156-157
- [10] 吕晓庆, 邹仕洪. 基于 smali 的 Android 软件敏感 API 调用日志模块嵌入系统[D]. 北京: 北京邮电大学. 2012, 1-6
- [11] XU R, SAIDI H, ANDERSON R. Aurasium: practical policy enforcement For Android application[C]//The 21st USENIX Conference on Security Symposium. USENIX Association Berkeley, CA, USA, c2012:27
- [12] BURGUERA I, ZURATUZA U, et al. Crowdroid: behavior-based malware detection system for Android[C]//The 1st ACM Workshop on security and privacy in Smartphones and Mobile Devices. New York, USA, c2011:15-26
- [13] 柯元旦. Android 内核剖析[M]. 北京: 电子工业出版社, 2012, 20-23
- [14] 传智播客高教产品研发部. Android 移动应用基础教程[M]. 北京: 中国铁道出版社, 2014, 4-5
- [15] 杨丰盛. Android 技术内幕系统卷[M]. 北京: 机械工业出版社, 2011, 56-62
- [16] 狄婧. Android 安全机制分析与解决方案解析[J]. 硅谷, 2011, 24:60-62
- [17] 张中文, 雷灵光, 王跃武. Android Permission 机制的实现与安全分析[C]. 第 27 次全国计算机安全学术交流会. 2012. 8:3-5

- [18] 左玲,基于 Android 恶意软件检测系统的设计与实现.[D].成都:电子科技大学,2012,13-18
- [19] A.Shabtai.Malware Detection on Mobile Devices[C].Proceeding of the 11th International Conference Monbile Data Management.Kansas City,Mission,USA, 2010:289-290