****

**学 位 论 文 诚 信 声 明 书**

本人郑重声明：所呈交的学位论文（设计）是我个人在导师指导下进行的研究（设计）工作及取得的研究（设计）成果。除了文中加以标注和致谢的地方外，论文（设计）中不包含其他人或集体已经公开发表或撰写过的研究（设计）成果，也不包含本人或其他人在其它单位已申请学位或为其他用途使用过的成果。与我一同工作的同志对本研究（设计）所做的任何贡献均已在论文中做了明确的说明并表示了致谢。

申请学位论文（设计）与资料若有不实之处，本人愿承担一切相关责任。

学位论文（设计）作者签名： 日期：

**学 位 论 文 知 识 产 权 声 明 书**

本人完全了解学校有关保护知识产权的规定，即：在校期间所做论文（设计）工作的知识产权属西安科技大学所有。学校有权保留并向国家有关部门或机构送交论文的复印件和电子版。本人允许论文（设计）被查阅和借阅；学校可以公布本学位论文（设计）的全部或部分内容并将有关内容编入有关数据库进行检索，可以采用影印、缩印或其它复制手段保存和汇编本学位论文。

保密论文待解密后适用本声明。

学位论文（设计）作者签名： 指导教师签名：

年 月 日

分 类 号 密 级

学校代码 10704 学 号 1308010204

**西安科技大学**

**学 士 学 位 论 文**

**题目：基于逆向工程的APP权限提取和恶意行为检测**

**作者：雷倩**

指导教师：刘晓建 专业技术职称：副教授

学科专业：软件工程 申请学位日期：2017年6月

**摘 要**

权限机制是Android系统内建的一种保护敏感资源或用户数据不被非法访问的一个重要访问控制机制。但是Android权限机制存在权限过度申请的问题，即在应用程序安装时，用户通常会将应用程序请求的权限全部授予。而一些恶意应用程序往往就是利用了某些过度申请的权限，从而实施恶意行为。本文主要针对这一问题，采用静态分析的方法开展了如下研究内容：

1. 设计和实现了一种在Android应用程序虚拟机可执行代码（smali代码）层次上，提取应用程序实际使用权限集合的算法，并与AndroidManifest.xml文件中声明的权限集合进行比对，从而判断应用是否存在权限过度申请或权限申请不足的安全漏洞；
2. 在上述权限提取的基础上，通过统计分析大量良性或恶意应用（约1300个）的权限使用情况，实现了一种利用权限特征判断应用恶意性及恶意行为的算法（LWD算法）；
3. 在前两项研究的基础上，设计并开发了一个判断Android应用恶意性及恶意行为的工具，使用户在安装之前，能够对下载的Android应用进行检测，并提供有关启发性安全警告，以期提高移动应用的安全性。

通过对样本进行分析检测，LWD算法的分析正确率达到75%，误报率控制在30%范围内；设计和开发的分析工具界面友好，集成度高，便于用户的使用。该课题的研究以及产生的成果对于提高Android系统的安全性，保护个人隐私，甚至社会经济政治安全将产生有益效果。

**关键词：**Android安全；恶意软件；敏感资源；权限过度；静态分析

**ABSTRACT**

Permission mechanism is a built-in Android system to protect sensitive resources or user data is not an illegal access to an important access control mechanism. When the application is installed, the user usually grants all of the permissions requested by the application. And some malicious applications are often using some of the excessive application of the permission to implement malicious acts. The main work of this paper includes:

(1) by analyzing the Android application's virtual machine executable code (smali code), extract the actual use of the program permissions set, and AndroidManifest file declared in the set of permissions to compare, to determine whether the application has permission to apply too much or Permission to apply for security vulnerabilities;

(2) Based on the above analysis of permission, a method of judging the malware of the application is designed and implemented by statistical analysis of the use of a large number of benign or malicious applications (about 300);

(3) Based on the previous two studies, we designed and developed a tool to determine the maliciousity of Android applications, so that users before the installation, the download of Android applications and testing, and provide inspiration safety warning, thereby enhancing The security of mobile applications.

Through the analysis of samples to detect, the correctness rate of the malware detection algorithm is 75% and the false alarm rate is within 30%. The design and development of the analysis tool Friendly interface, high integration, user-friendly. The study of the subject and the results produced will improve the security of the Android system, protect personal privacy, and even social and political security will have a beneficial effect.

**Key Words:** Android security; malicious software;sensitive resources; overprivilege; static analysis

**目 录**

**[1 绪论](#_Toc23866)** [- 1 -](#_Toc23866)

[1.1 论文研究背景及意义 - 1 -](#_Toc20524)

[1.2 研究现状 - 1 -](#_Toc22185)

[1.2.1 国外研究现状 - 2 -](#_Toc20644)

[1.2.2 国内研究现状 - 2 -](#_Toc30728)

[1.3 本论文研究内容 - 2 -](#_Toc18637)

[1.4 论文组织结构 - 4 -](#_Toc11484)

**[2 Android体系结构及安全机制](#_Toc15370)** [- 5 -](#_Toc15370)

[2.1 Android体系结构 - 5 -](#_Toc24866)

[2.1.1 Linux内核层 - 5 -](#_Toc11230)

[2.1.2 核心类库层 - 5 -](#_Toc18380)

[2.1.3 应用框架层 - 5 -](#_Toc16654)

[2.1.4 应用程序层 - 6 -](#_Toc27093)

[2.2 Android安全机制 - 6 -](#_Toc32453)

[2.2.1 进程沙箱隔离机制 - 6 -](#_Toc27481)

[2.2.2 应用程序签名机制 - 6 -](#_Toc22501)

[2.2.3 权限声明机制 - 6 -](#_Toc7702)

[2.2.4 进程通信机制 - 7 -](#_Toc32756)

[2.3 Android权限管理模型 - 8 -](#_Toc30010)

[2.3.1 权限概述 - 8 -](#_Toc23610)

[2.3.2 权限的使用方法 - 8 -](#_Toc19571)

[2.3.3 权限的授予与管理 - 9 -](#_Toc24335)

[2.4 Android权限机制有效性 - 11 -](#_Toc17554)

[2.4.1 权限保护的职责划分 - 11 -](#_Toc9842)

[2.4.2 使用权限和申请权限的不一致性 - 11 -](#_Toc19017)

[2.4.3 权限提升攻击 - 11 -](#_Toc16465)

**[3 Android应用分析技术](#_Toc16324)** [- 13 -](#_Toc16324)

[3.1 Android应用程序结构 - 13 -](#_Toc18996)

[3.2 Android可执行文件 - 13 -](#_Toc18486)

[3.3 反汇编字节码文件 - 15 -](#_Toc3884)

[3.4逆向分析工具apktool - 16 -](#_Toc27785)

[3.5 PScout提取显式API-权限映射表 - 17 -](#_Toc7721)

[3.6 恶意权限组合 - 17 -](#_Toc19153)

**[4 Android权限检测方法及工具设计](#_Toc25647)** [- 19 -](#_Toc25647)

[4.1 权限提取方法 - 19 -](#_Toc20032)

[4.1.1 权限提取流程 - 19 -](#_Toc28928)

[4.1.2 权限提取算法 - 20 -](#_Toc7065)

[4.2 LWD算法 - 22 -](#_Toc557)

[4.2.1 特征权限向量 - 22 -](#_Toc32389)

[4.2.2 权限恶意程度值 - 22 -](#_Toc4022)

[4.2.3 权限恶意度列向量 - 23 -](#_Toc17964)

[4.2.4 权限特征向量 - 23 -](#_Toc11932)

[4.2.5 应用恶意程度值 - 23 -](#_Toc13316)

[4.2.6 恶意度阈值 - 23 -](#_Toc276)

[4.2.7 线性分类器 - 23 -](#_Toc30138)

[4.2.8 算法伪码 - 24 -](#_Toc26939)

[4.3 检测工具的设计 - 24 -](#_Toc11151)

[4.3.1 功能模块 - 24 -](#_Toc6777)

[4.3.2 类图 - 25 -](#_Toc18104)

**[5 实验与结果分析](#_Toc18168)** [- 27 -](#_Toc18168)

[5.1 实验环境搭建 - 27 -](#_Toc10095)

[5.2 实例研究 - 27 -](#_Toc20953)

[5.2.1 研究步骤 - 27 -](#_Toc24793)

[5.2.2 结果分析 - 29 -](#_Toc29712)

[5.3 实验步骤 - 29 -](#_Toc30968)

[5.4 权限提取与恶意行为检测工具 - 33 -](#_Toc28934)

[5.5 实验结果及分析 - 39 -](#_Toc27962)

**[6 结论与展望](#_Toc8568)** [- 41 -](#_Toc8568)

[6.1 结论 - 41 -](#_Toc18529)

[6.2 展望 - 41 -](#_Toc3630)

**[致谢](#_Toc15580)** [- 43 -](#_Toc15580)

**[参考文献](#_Toc24956)** [- 44 -](#_Toc24956)

**1 绪 论**

**1.1 论文研究背景及意义**

Android已经成为全球市场占有份额最高的智能手机平台。市场研究机构Kantar Worldpanel最新发布的调查报告显示，截止到2016年4月底，Android系统在全球所占市场份额已高达76%。尽管Android设备的用户量非常大，但是由于其开放性和自身安全机制存在的一些漏洞，使其成为不法分子的主要攻击对象，出现的安全及隐私问题也越来越严重。

2014年腾讯安全实验室发布的《2013年手机安全报告》[1]指出，2013年全年该实验室共检测截获Android手机病毒高达763351个。据央视网消息称：在2017年手机安全报告中显示，99%安卓手机存在安全漏洞。一些恶意应用窃取用户账号、访问号码簿向联系人发送诈骗信息等；木马应用控制用户的手机设备，盗取用户的银行账号，窃取通话记录、拦截短信等，这些都严重威胁着手机用户的个人隐私以及经济安全；还有一些新的安全问题也不断涌现，如网络爬虫、软件及系统漏洞等。在以上背景下，对手机端恶意应用的检测成为了当前Android安全问题领域最重要的研究课题之一。

权限机制是Android系统内建的安全机制之一。Android系统采用权限机制保护设备中的敏感资源和隐私数据。在权限机制下，所有的硬件设备、用户数据、敏感资源都被赋予一定的权限，应用程序如果要对这些敏感资源或硬件进行访问，首先需要声明所使用的权限，然后在安装应用时由用户决定是否授权。但是通常情况下，用户总是完全授予所声明的权限，这就使得一些恶意应用通过过度申请权限从而达到企图的恶意目的。因此，有效的提取和比对应用所申请和所实际使用的权限，能够有效的检测应用是否存在权限过度申请或权限申请不足的问题，从而为判断应用的恶意性提供一定的启发。另外，权限作为应用程序的一项显式特征，蕴含了程序行为的重要特性，因此，通过权限特征分析，能够在一定程度上揭示和发现程序的恶意行为。总之，对程序权限特征的研究和分析，是检测程序恶意性的一种重要手段和途径。

**1.2 研究现状**

现阶段关于Android应用程序的安全性研究主要分为静态检测方法和动态检测方法两条技术路线。静态分析主要是从应用的源码入手进行分析，如程序所调用的API、程序所申请的权限特征等，来判断程序的安全性质；动态方法主要通过应用程序在运行过程中表现出来的行为特征并结合数据分析方法，对应用的安全性质作出判断。本论文主要采用静态分析方法对Android应用程序的安全性进行研究。下面分别介绍国内外针对静态检测技术已有的研究成果。

**1.2.1 国外研究现状**

加州大学伯克利分校的Felt构建了Stowaway[2]，它是一个用于在已编译程序中检测权限过度申请的工具。Stowaway首先确定应用程序使用的一组API调用，然后将这组API调用映射到权限，构建了完整的API-权限映射表。Enck[3]等人最早开始了基于权限机制的研究，设定了一种安全权限规则Kirin规则，在应用程序访问权限的过程中，如果有悖于该规则，Kirin方案就会拒绝安装此应用。但该规则过于严格，如果仅使用该规则判断一个应用程序的安全性问题，则误报率会非常高，将一些正常的应用也拒之门外。在以权限判断恶意应用的研究方面，文献[4]通过总结恶意权限组合判断应用是否为恶意应用，但该方法并不准确。因为对于某些应用来说，一些恶意的权限组合却是必须的。例如INTERNET+FINE\_LOCATION，该权限组合可用来实时跟踪用户的地理位置，但却是天气预报类应用用来确定用户地理位置以显示本地天气所必须的权限。Suleiman et al[5]在逆向得到反编译文件后，从smali文件中抽取API的调用信息，用贝叶斯方法构造分析模型，以此判断应用的安全性。Borja Sanz et al[6]通过对一些重要字符串信息进行统计，这些字符串信息均在应用源码中出现并使用过，然后建立空间矢量模型，根据特征对字符串进行分类，从而实现对应用的安全性检测。

**1.2.2 国内研究现状**

2015年重庆邮电大学黄梅根[7]等人发表的论文中，提出了一系列恶意权限组合，这些权限组合均可威胁用户的隐私，然后根据Google Play对Android应用的分类，计算出对于每一类应用每个恶意权限组合的恶意程度值。最后通过恶意程度值计算出待检测应用的恶意值，从而判断该应用的恶意性。

天津大学的焦浩峰[8]在其2014年的硕士毕业论文中提出了一种基于权限的混合检测框架。先根据应用申请的权限进行初步检测，检测出正常应用和恶意应用；然后跟踪可疑应用的行为，收集与敏感权限相关的接口调用进行检测，进而确定应用类型。

华中师范大学的郑吉飞[9]在其2013年的硕士毕业论文中提出了一种检测框架，该检测框架基于静态分析对Android恶意代码进行检测。

中国科学技术大学彭凌[10]在其2015年的硕士毕业论文中提出了一种Android应用隐式权限检测技术。

童振飞[11]采用抽取apk类名和调用函数名作为特征属性，通过计算与样本相似度对apk进行分类。实验证明，该方法可以有效的对apk进行分类，但该检测方法局限于Java层，未考虑动态共享库的加载。

秦中元[12]提出了一种基于危险权限和行为分析的静态综合检测方法，对已检测过的apk，提取消息摘要的MD5值作为签名用来进行快速匹配和判定；未检测apk根据权限和行为分析来判定，使用污点传播和语义分析，检测apk中是否存在隐私窃取行为。

**1.3 本论文研究内容**

本论文首先介绍了Android系统的架构及其安全机制，然后着重从权限(静态特征)的角度对Android的权限管理机制的有效性进行分析，并构建了一个恶意应用检测工具。主要研究内容如下:

（1）学习Android系统的权限管理机制，并做了一个实例研究，确认了权限检查是在运行时进行检查的，而不是在编译时进行的。Android应用程序中对系统硬件资源以及受保护资源的访问和操作都是通过权限来进行控制的。默认情况下，应用程序不拥有任何权限，要想访问系统敏感资源，需声明访问资源所需的权限并由用户授予。但存在一种特殊情况，若某应用程序内调用了某API，根据权限审核机制可知，权限的审核函数是在API内部进行调用的，如果程序运行过程中，该API调用是不可达的，那么该API的权限审核函数将永远不会被执行。通过实例研究，确认权限检查发生的时间，为准确地判断应用程序的权限使用情况提供基础。

（2）通过逆向静态分析方法得到Android应用程序的smali代码，并在smali代码层次设计并实现了一种权限提取算法，并使用该算法分析和统计了若干应用软件的权限使用情况，判断该应用是否具有权限过度申请或权限申请不足的安全漏洞。

（3）通过分析统计大量的实验样本(约300个)，提取特征权限，实现了一种利用权限特征判断应用恶意性的算法。

（4）按照Google Play对应用的分类，对大量的正常样本进行分析(约1000个)，统计在每类应用中恶意权限组合的分布情况，从而判断哪些权限组合对哪些类别的应用是必须的，而哪些则是不必要的。在上一步对应用恶意性判断的基础上，利用恶意权限组合，根据恶意应用的类别进一步判断其可能存在的恶意行为。

（5）将前三项研究工作封装成一个自动化的检测工具，使用户在安装应用程序之前可以对apk包进行一个预判，为用户提供该包的恶意性启发式信息，减少恶意应用对用户造成的危害。

**1.4 论文组织结构**

本论文总共分为六章，其组织结构如下：

第一章：主要阐述了本论文的研究背景及现状，并介绍了本论文的研究内容及组织结构。

第二章：详细介绍了Android平台架构以及Android的安全机制。对Android权限的管理模型进行了总结分析，并提出了现有的安全机制的不足之处。

第三章：介绍了Android应用程序、Android可执行文件以及反编译文件的结构以及语法格式，并对反编译工具apktool、API-权限映射表和恶意权限组合进行了简要介绍。

第四章：介绍了从应用程序的中间代码中提取权限信息的基本流程、权限提取算法以及检测应用程序恶意性的算法，并绘制了实现本工具模型的类图以及功能模块图。

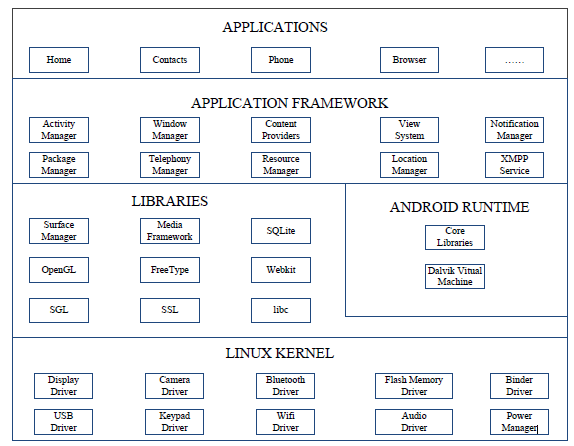
第五章：通过对大量的样本进行实验，确定LWD算法的参数，实现检测工具并展示，然后将大量实验样本应用于该工具，对实验结果进行分析，与市场上主流的杀毒软件进行比较。

第六章：总结本论文的主要工作，通过实验总结本论文方法的不足与欠缺之处，并展望后续可加强的工作及方向。

**2 Android体系结构及安全机制**

**2.1 Android体系结构**

Android系统由Linux内核层、核心类库层、应用框架层、应用程序层构成，各层都有自己的分工，层层隔离，互不影响。具体如图2-1所示。



**图2-1 Android系统架构**

**2.1.1 Linux内核层**

Android系统借助Linux内核服务，实现进程和存储器管理、网络协议、无线通信等核心功能。

**2.1.2 核心类库层**

核心类库的系统库提供了C/C++库，使得开发者可以在自己的应用中调用一些C/C++本地库中的接口实现某些功能。如OpenGL/EL库提供了3D绘图的支持。

**2.1.3 应用框架层**

应用框架层实际上是隐藏在一系列应用之后的服务，该层采用java语言编写，为应用开发者提供了一套实现核心功能的API。开发者在开发应用程序时可以调用这些API。它主要又包括以下几部分：可以用来构建应用程序的视图，如布局、文本框等；用于共享应用程序数据，在一个应用程序中访问另一个应用程序数据的内容提供者；管理应用程序整个生命周期的活动管理器。

**2.1.4 应用程序层**

应用程序层是所有安装在手机上的应用的集合，例如系统自带的拨号程序、图库、或者从Android市场上下载的小应用等都属于这一层。

**2.2 Android安全机制**

Android将安全设计贯穿系统架构的各个层面，力求不仅要做到开放，还需适当保护用户数据、敏感资源、设备以及应用的安全。Android系统主要提供以下安全机制。

**2.2.1 进程沙箱隔离机制**

安装应用时，Android系统会给应用赋予一个UID，运行于Dalvik虚拟机实例的应用程序都在独立的Linux进程空间中运行，UID不同的应用程序之间相互隔离。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UID，运行于同一进程，并共享其资源和权限。  
**2.2.2 应用程序签名机制**

所有Android应用在发布时都要被打包成apk包，apk包在发布时都必须进行签名，该机制是应用程序用来进行自我认证的。Android系统利用签名机制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信任关系，而不是用来判定应用程序是否应该被安装。

在对开发的应用签名时，使用调试私有密钥自动进行，这类应用称为调试模式应用，这些应用不能发布到Google Play市场上。当开发者需在Google Play上发布自己的应用，必须用安卓的签名机制使用其私有密钥对应用进行签署，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。

**2.2.3 权限声明机制**

Android通过权限机制保护设备资源以及用户数据，采用权限来限制应用程序对硬件设备、敏感资源的访问能力。当某应用程序要在某些资源对象上执行操作时，首先必须获得相应的权限。

Android 框架提供了一套定义好的权限，这些权限存储在android.manifest.permission类中，开发者在开发应用程序时可申请。然后在程序安装时由用户授予。同时Android系统允许用户自定义权限。其他应用程序若想访问该应用程序或其中某个组件，需申请自定义的权限。所以，Android系统主要安全准则是应用只有在获得权限后，才有权利执行一些会对系统或其他应用程序造成影响的敏感操作。

Android应用程序使用权限时，在AndroidManifest.xml文件中用<use-permission>元素申请权限。在程序安装时，将请求权限展示给用户，用户可选择同意安装或拒绝安装。如申请拨打电话权限：

<uses-permission android:name=”android.permission.CALL\_PHONE”/>

应用可用<permission>标签来定义自己的权限，从而限制对本应用某些组件或程序的访问。在安装程序时，自定义的权限被加入到系统中。开发者在自定权限时可指定权限的危险级别，其危险级别分为以下三个等级：

1. 正常权限：保护应用程序对可能但不会伤害用户的API调用的访问。例如SET\_WALLPAPER控制改变用户背景壁纸的能力。
2. 危险权限：控制对潜在有害的API调用的访问。例如与支付或收集私人信息相关的API调用。例如，发送短信或读取联系人列表就需要危险权限。

（3）签名/系统权限：管理对最危险权限的访问，例如删除应用程序包的能力。这些权限一般情况下很难获得：签名权限仅授予那些使用设备制造商的证书签名的应用程序，并且SigenatureOrSystem权限仅授予在特殊系统文件夹中签名或安装的应用程序。这些限制实质上将签名/系统权限仅用于预安装的应用程序中，并且其他应用程序对签名/系统权限的请求将被忽略。

**2.2.4 进程通信机制**

Android的Binder机制由Client、Server、Service Manager和Binder驱动程序组成，这四组件的工作关系如图2-2所示。



**图2-2 Android系统Binder机制中的四个组件的关系**

**2.3 Android权限管理模型**

**2.3.1 权限概述**

Android为第三方应用程序提供了丰富的API，其中包括对手机硬件、配置和用户数据等的访问。在第三方应用程序对与隐私和安全有关的API进行访问时，Android系统使用权限对该访问进行控制，同时应用程序之间的组件访问也由权限控制。开发者在访问某些敏感资源时，需在项目的AndroidManifest.xml文件中声明相应的权限。为了执行权限，系统的各个部分通过调用权限验证机制来检查给定应用是否具有指定的权限。权限验证机制被作为可信的系统进程的一部分，并且其调用分布在整个API中。  
**2.3.2 权限的使用方法**

1.使用系统自带的权限

Android系统为应用程序开发者已提供了170多种权限，以供应用在执行某些操作时使用，这些权限被定义在Android系统源码的android.manifest.permission类中。如果应用程序想要系统的某个权限，首先需要使用<uses-permission>声明。该声明位于AndroidManifest.xml文件的<manifest>标签中。

2.用户自定义权限

Android支持程序开发人员自定义权限，以此来限制某些应用对自身组件的访问。自定义权限使用<permission>标签来完成。代码2-1、2-2为一个例子。当要限制某些应用对D中的DActivity访问时，需在AndroidManifest.xml文件中对DActivity进行声明。

**代码2-1 自定义权限**

|  |
| --- |
| <activity  android:name=”com.example.testandroid.DActivity”  android:exported=”true”  android:label=”D”  android:permission=”corn.permission.CORN\_OWN”>  </activity> |

此时如果应用C想打开D中的DActivity，则需要加上自定义权限。

**代码2-2 使用自定义权限**

|  |
| --- |
| <uses-permission android:name=”corn.permission.CORN\_OWN”>  </uses-permission> |

**2.3.3 权限的授予与管理**

1.权限的授予

Android的每一个应用程序都运行在自己的进程中，系统可以通过进程ID区分各个应用程序，并为每个应用程序赋予不同的权限。

权限机制的实施贯穿于整个Android体系结构。其授予的具体过程是，应用程序层通过应用的AndroidManifest.xml文件申请权限，框架层在platform.xml的<permission>下指定该权限与内核的权限关系，内核层在android\_filesystem\_config.h中定义了权限。图2-3展示了Android系统对应用权限的授予过程。



**图2-3 Android系统权限授予过程**

2.权限的动态检查

程序在运行过程中，Android系统会随时监控应用对敏感资源的访问。当需要访问某些系统资源时，Android的管理组件Activity Manager会检查该应用是否拥有该资源的访问权限，只有申请了并被用户授权的应用程序才能访问受保护的资源。图2-4展示了程序安装时的审核流程（由虚线表示）与应用在运行过程中的权限检查过程（由实线表示）。



**图2-4 安装时与运行时权限审核过程**

Android框架会在外部程序要对某些敏感资源访问时进行权限检查，该检查通过一组接口完成，主要通过Context Wrapper提供，具体实现在ContextImpl中。如果package接收到外来者的访问请求，便会调用这些接口，接口的参数pid和uid可以指定。表2-1列举了常用的几个权限检查函数。

**表2-1 常用的权限检查函数**

|  |
| --- |
| public int checkPermission(String permission, int pid, int uid) |
| public int checkCallingPermission(String permission) |
| public int checkCallingOrSelfPermission(String permission) |
| public void enforcePermission(String permission, int pid, int uid, String message) |
| public void enforceCallingPermission(String permission, String message) |
| public void enforceCallingOrSelfPermission(String permission, String message) |
| public int checkUriPermission(Uri uri, int pid, int uid, int modeFlags) |

**2.4 Android权限机制有效性**

**2.4.1 权限保护的职责划分**

Android系统权限管理职责的划分可分为三类：

1. 应用开发者：程序开发人员在开发应用、申请权限时应遵守最小权限原则；
2. 手机用户：在安装一款应用时必须首先了解应用所申请的权限，充分考虑到授予权限后可能会带来的后果后再决定是否安装程序；

（3）Android系统：Android系统保证应用只有在拥有权限的情况下才能执行某些受限操作。

然而，据调查，上述三类参与者的职责并没有被很好地履行，如应用开发者由于对Android权限系统的不了解导致其在权限申请的过程中，为了完成应用本身具有的功能往往会过多的申请权限。大多数的Android用户在安装一款应用时并没有耐心去仔细地了解每一类权限，并且Android的权限系统过于复杂，普通用户根本无法了解这些权限。Android系统会在应用程序安装时授予应用所有的权限，并且无法改变，若想解除应用与权限的联系，则必须卸载该应用。  
**2.4.2 使用权限和申请权限的不一致性**

开发者在开发应用程序时，由于对权限系统的不了解，往往会选择将所有可能用到的权限都申请，以此来完成自己本应用的功能，而这些申请的权限中，可能会存在一部分权限在该应用中根本不需要，这便会产生应用申请的权限和应用实际使用到的权限存在不一致性。这类问题可能会产生如下两种后果：

（1）过度申请权限：这一类权限是开发者在申请权限时由于对权限系统的不了解而过度申请的，这类权限在应用程序的运行过程中并无实际作用，本身不会产生什么恶意行为，但有可能会造成权限提升攻击等危害。

（2）过度使用权限：这一类权限是应用在实现其功能时实际使用到的权限但并未在AndroidManifest.xml文件中申请，这一类权限造成的危害是在应用程序执行过程中执行某些敏感操作时，如果应用没有这一类操作的相关权限，程序执行将会抛出异常。

**2.4.3 权限提升攻击**

在Android平台上，每个应用程序都拥有自己独立的虚拟机实例，运行在自己的进程中。一般情况下不允许访问其他应用程序的组件。但通过Android组件的通信机制，恶意应用仍然可以通过与其他应用程序通信获得权限，从而提升自己的权限。

2011年Davi[13]等人提出的权限提升攻击模型如图2-5所示。



**图2-5权限提升攻击模型**

从图中可以看出，三个应用程序分别运行在自己的虚拟机实例当中，每个应用都有属于自己的组件，APP1没有赋予任何权限并且没有被任何权限保护，APP2被赋予了Permission1，APP3的两个组件分别被Permission 1和Permission2保护起来。因为APP2中的组件没有被权限保护，所以APP1的任何组件都可以访问APP2的任何一个组件，APP3的Comp1由Permission1保护，那么拥有Permission1的APP2便可以访问APP3的Comp1，虽然APP1没有被赋予权限访问APP3的Comp1，但它可以通过访问APP2，再由APP2访问APP3的Comp1，从而达到APP1访问APP3的Comp1的效果。

**3 Android应用分析技术**

**3.1 Android应用程序结构**

在Android系统下，一个Android项目是由多个文件以及文件夹组成的，这些文件分别用于不同的功能，常用的文件和文件夹如下：

src：该目录是放置所有Java代码的地方。

gen：该目录下有一个R.java文件，该文件会为项目中所有的非代码资源生成一个资源编号id，该文件会随着项目中资源的变化重新编译。

Android 4.1.2：该目录存放的是工程使用的Android SDK。

assets：该目录通常会包含一些多媒体资源，这里的资源不会在R.java文件中生成索引，在应用程序打包时会一起被打包。

bin：该目录存放可执行的二进制文件，其中还会有一个apk文件，该文件就是应用程序的安装包，可直接安装，但该安装包不能作为发布版本使用。

libs：该文件夹下存放的是项目中用到的Jar包，这些包会被添加到build path中去，并在项目中用到的地方被引入。

res：该目录存放的是项目中用到的各种资源，这些资源会在gen目录下的R.java文件中生成资源id。图片资源就会放置在drawable文件夹下，字符串资源会放置在values目录下。Drawable文件夹以其不同大小的手机荧幕又将其分为drawable-hdpi、drawable-ldpi、drawable-mdpi、drawable-xhdpi、drawable-xxhdpi，以便做荧幕适配。

AndroidManifest.xml：该文件是项目的配置文件，组件的注册就在该文件中进行，并且该文件也是开发者申请权限的地方。应用启动时，系统会根据该文件设置的信息打开组件。

project.properties：该文件记录了Android项目运行时的环境，这个版本可以手动更改，但必须是已下载的版本。

**3.2 Android可执行文件**

Android的可执行文件即为运行在Dalvik虚拟机上的文件。apk实际上就是一个zip文件，通过对该文件查看可知该文件夹包含一个classes.dex文件，这个文件就是运行在Dalvik VM上的可执行文件。dex文件的结构如图3-1所示。



**图3-1 dex文件结构**

关于dex文件每个组成部分的详细说明如表3-1所示。

**表3-1 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 |
| class\_defs | 类定义表，在该表中，父类以及实现的接口必须出现在被引用类之前 |
| data | 数据区 |
| link\_data | 静态链接文件使用的数据 |

**3.3 反汇编字节码文件**

反汇编字节码文件即为通过借助某种反编译工具对编译打包好的apk文件进行逆向得到接近于源码的中间代码，例如用apktool工具对一个应用程序包反编译后得到smali文件，smali文件相当于Android项目中的Java代码。smali的数据类型如表3-2所示。

**表3-2 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 |

smali文件相当于Android项目中的src目录下的Java代码，但与之不同的是，无论是什么类都会相应地生成单独的smali文件。每个smali文件的前三行都会给出一些关于本类的相关信息，格式如下所示：

.class<类名>

.super <父类名>

.source<源文件名>

一个类由多个属性和方法组成。smali文件中属性的声明使用“.field”指令。方法的声明使用“.method”指令。

对一个字段的读写操作在smali文件中使用字段操作指令来完成。字段的类型可以是java中有效的数据类型。对普通字段与静态字段操作有两种指令集，分别是“iinstanceop vA,vB,field@CCCC”与“sstaticop vAA,field@BBBB”。

如果一个方法中调用了类实例的方法，在smali文件中使用方法调用指令来完成，它的基础指令为invoke，方法调用指令有“invoke-kind{vC，vD，vF，vG}，meth@BBBB”与“invoke-kind/range{vCCCC...vNNNN}，meth@BBBB”两类，两类指令在作用上并无不同，只是后者在设置参数寄存器时用了range来指定寄存器的范围。如smali文件中的一个方法调用语句：invoke-virtual{p1,p3}, Landroid/app/NotificationManager;->notify(I)V，其中Landroid/app/NotificationManager代表该方法所属的类，V表示该方法无返回值。

**3.4逆向分析工具apktool**

apktool是用于反编译apk文件的反编译工具，借助该工具可实现对apk文件的静态分析。可在多种操作系统平台上使用，本论文选择在Windows平台下使用。apktool工具由以下三部分组成：

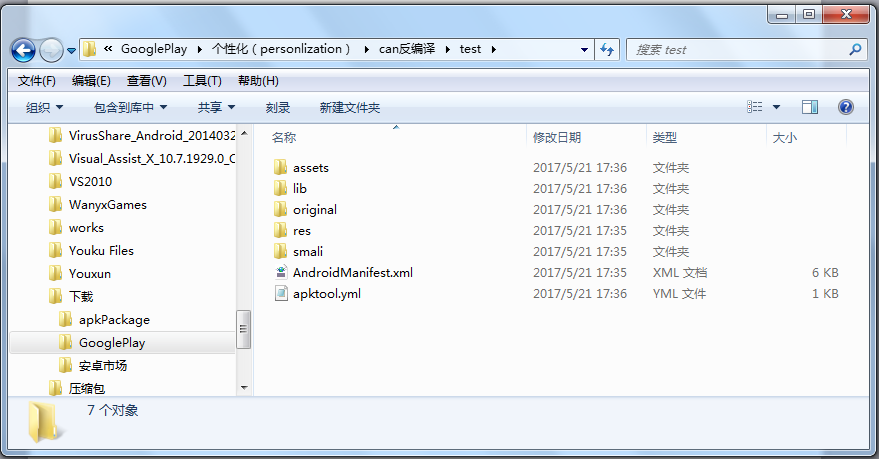
1. aapt.exe：Android程序的反编译工具；
2. apktool.bat：这是一个批处理文件，文件包含执行反编译工作的一系列命令；
3. apktool.jar: apktool在对Android应用反编译时需要Java运行环境，apktool.jar为反编译工作提供了环境。

apktool反编译Android应用程序的流程图如图3-2所示。



**图3-2 apktool反编译文件流程图**

反编译文件的目录结构如图3-3所示。



**图3-3 apktool反编译生成的文件目录**

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。

**3.5 PScout提取显式API-权限映射表**

PScout是美国伯克利大学目前在做的一项著名的研究，该研究通过对Android系统的源码进行分析，得到Android API对应的权限需求，并与Android文档结合得到目前最为完整的API-权限映射表。PScout的部分权限映射条目如图3-4所示。



**图3-4 PScout部分API-权限映射条目**

**3.6 恶意权限组合**

权限是应用程序访问系统资源的先决条件，拥有了权限，应用便可以对系统资源进行访问。那么在拥有多个权限的情况下，应用便可以以权限组合的方式产生某些恶意行为。如一个音乐播放器同时具有ACCESS\_FINE\_LOCATION+INTERNET权限，便可将用户的位置信息通过网络发送出去。本论文引用文献[7]提出来的一系列恶意权限组合，根据该组合可以向用户提示恶意应用可能存在的恶意行为。部分恶意权限组合如表3-3所示。

**表3-3 部分恶意权限组合**

|  |  |  |
| --- | --- | --- |
|  | **序号** | **恶意权限组合** |
| 窃听用户通话，自动拨打扣费电话 | 1 | READ\_PHONE\_STATE+RECORD\_AUDIO+INTERNET |
| 2 | PROCESS\_OUGOING\_CALLS+RECORD\_AUDIO+INTERNET |
| 3 | CALL\_PHONE |
| 4 | PROCESS\_OUTGOING\_CALLS+SEND\_SMS |
| 5 | PROCESS\_OUTGOING\_CALLS+INTERNET |
| 6 | READ\_CONTACTS+INTERNET |
|  | 7 | READ\_CONTACTS+SEND\_SMS |
| 8 | READ\_CALL\_LOG+INTERNET |
| 9 | READ\_CALL\_LOG+SEND\_SMS |
| 窃取用户短信 | 10 | RECEIVE\_SMS+INTERNET |
| 11 | RECEIVE\_SMS+SEND\_SMS |
| 12 | RECEIVE\_MMS+SEND\_SMS |
| 13 | RECEIVE\_MMS+INTERNET |
| 14 | READ\_SMS+INTERNET |
| 15 | READ\_SMS+SEND\_SMS |
| 跟踪用户位置 | 16 | ACCESS\_FINE\_LOCATION+INTERNET |
| 17 | ACCESS\_FINE\_LOCATION+SEND\_SMS |
| 18 | ACCESS\_COARSE\_LOCATION+INTERNET |
| 19 | ACCESS\_COARSE\_LOCATION+SEND\_SMS |
| 窃取其他个人隐私信息 | 20 | READ\_CALENDAR+INTERNET |
| 21 | READ\_CALENDAR+SEND\_SMS |
| 22 | READ\_PHONE\_STATE+INTERNET |
| 23 | READ\_PHONE\_STATE+SEND\_SMS |
| 24 | READ\_LOGS+INTERNET |
| 25 | READ\_LOGS+SEND\_SMS |
| 26 | READ\_PROFILE+INTERNET |
| 27 | READ\_PROFILE+SEND\_SMS |
| 28 | READ\_HISTORY\_BOOKMARKS+INTERNET |
| 29 | READ\_HISTORY\_BOOKMARKS+SEND\_SMS |
| 30 | READ\_EXTERNAL\_STORAGE+INTERNET |

**4 Android权限检测方法及工具设计**

**4.1 权限提取方法**

Android通过权限授予机制控制应用对API的调用。合理地请求并使用权限是保障Android系统安全性的关键。但权限请求的自主权在于软件开发人员，很难保证他们在请求权限的过程中不会过度申请，而且程序在安装过程中展示给用户的权限信息仅仅是开发人员在AndroidManifest.xml文件中申请的权限，并非代码中真正使用到的权限。所以，在源码中提取权限是一种有效的解决方案。

**4.1.1 权限提取流程**

从Android程序反编译代码中提取权限的流程如图4-1所示。



**图4-1 权限提取流程图**

API-权限映射关系：Android利用权限机制管理第三方应用对与安全和隐私相关的资源访问，在Android系统中，每项权限对应一项资源。本论文中用到的API-权限映射表来自于PScout发布的最新版本的all API call mapping的android5.1.1版本的文档。该文档中总共记录了75个权限对应的API调用。

反汇编代码：对一个应用进行检测分析时，我们通常只能得到打包好的apk文件，而非程序的源代码，而apk文件中的classes.dex文件是可直接在Dalvik虚拟机上运行的可执行文件，对该文件直接进行分析是非常复杂的。所以，我们通常借助反编译工具对apk文件进行处理，得到接近于源码的反汇编代码，这将大大降低分析难度。目前市场上流行的反编译工具非常多，有的甚至可以直接反出来源码，如dex2jar工具，但此处我们只关注源程序中对API的调用情况，而apktool反编译出来的中间代码可以满足我们的基本要求，并且apktool工具安装起来比较方便，所以我们选择使用apktool工具来完成反编译工作。

API调用信息：利用迭代的方式对其smali文件夹底下的所有smali文件进行逐行扫描，将所有包含invoke以及敏感API的行输出到一个文件中，最终得到的文件就是该应用调用的所有敏感API。

信息匹配：通过对smali文件的扫描得到的文件信息只是该应用的API调用信息，并不能直观的看出该应用使用到了哪些权限。要想获得应用真正使用到的权限信息，需将上一步得到的API调用信息与API-权限映射表进行信息匹配，即将API-权限映射表以二维表的形式存储在Mysql数据库中，然后以API调用信息为查询条件查找相应的权限。

权限报告：通过上一步的信息匹配，将查询出来的所有权限输出到一个文件中，得到的即为关于该应用的权限报告。

**4.1.2 权限提取算法**

1.应用使用权限提取

将smali文件的路径作为输入，遍历该路径下的所有smali文件，抽取ＡＰＩ调用信息，并将该信息与API-权限映射表进行信息匹配，从而得到应用的权限报告。算法的伪码如代码4-1所示。

**代码4-1 提取使用权限的算法**

|  |
| --- |
| 输入： filePath //smali文件夹路径  *L*API//敏感API集合  *L*API-permission //API-权限映射表  输出：*L*all-permission //权限报告  function get\_smali\_permission(filePath)  1. fileList=filePath.list(); // 获取该文件夹下的所有文件  2. for i=0 to fileList.Length //对所有的文件进行遍历  3. {  4. *L*smali-API=chackStr(fileList[i],*L*API); //提取fileList[i]文件的敏感API调用信息  5.  *L*permission=getPermission(*L*smali-API, *L*API-permission ); //匹配API调用对应的权限信息  6. *L*all-permission=*L*all-permission∪*L*permission*;* //将得到的权限写入总的API权限报告  7.  *L*smali-API=null;  8. *L*permission=null;  9. }  10. return *L*all-permission; |

2.应用申请权限提取

应用程序所申请的权限可从AndroidManifest.xml文件中获取，提取算法如代码4-2所示。

**代码4-2 申请权限提取算法**

|  |
| --- |
| 输入：Mfile //AndroidManifest.xml文件路径  输出：*L*re-permission //申请的权限集合  function get\_require\_permission(Mfile)   1. *L*temp=readFile(Mfile); //将文件的所有行存入一个字符串数组中 2. for i=0 to *L*temp.Length //对字符串数组进行扫描 3. {   4. if(*L*temp[i].contains(“uses-permission”))  5. *L*re-permission=*L*re-permission∪*L*temp[i];  6. else  7. continue;  8. }  9. return *L*re-permission; |

3.启发式信息提取

启发式信息包括过度申请的权限和过度使用的权限两部分，该信息的获取主要以应用申请的权限集合和应用使用的权限集合作为输入，对这两个集合求差集便可得到相应的启发式信息。具体算法如代码4-3所示。

**代码4-3 提取启发式信息**

|  |
| --- |
| 输入：*L*all-permission  // 应用使用的权限集合  *L*re-permission  // 应用申请的权限集合  输出：*L*over-use  // 过度使用的权限集合  *L*over-re  // 过度申请的权限集合  function getOvered(*L*all-permission,*L*re-permission)  1. *L*use-re=*L*all-permission∩*L*re-permission; // 求使用的权限集合和申请的权限集合的交集  2. *L*over-use=*L*all-permission-*L*use-re; // 求过度使用的权限集合  3. *L*over-re =*L*re-permission-*L*use-re; // 求过度申请的权限集合  4. return *L*over-use,*L*over-re; |

**4.2 LWD算法**

LWD（Light Weight Detection）算法全称为轻量级检测算法。该算法首先统计出在恶意样本中出现次数最多的前20个权限作为特征权限，然后再统计出这20个权限在正常样本中的出现次数，根据特征权限在恶意样本和正常样本中出现的次数的差异性确定每个权限的恶意程度值，然后根据这些值计算待测样本的恶意程度值，并与规定的阈值进行比较，判断程序的恶意性。LWD算法的流程图如图4-2所示。



**图4-2 LWD算法流程图**

**4.2.1 特征权限向量**

特征权限是从恶意应用中选取出来的具有代表性的部分权限构成的向量，该向量是LWD算法的基础。特征权限向量被定义为：

*P*=(*p*1，*p*2，...，*p*20) (4-1)

其中*p*i为恶意应用样本集中统计权限的次数从高到低排序为第i位的权限。

**4.2.2 权限恶意程度值**

权限恶意程度值用来定量衡量一个特征权限产生恶意行为的能力大小，该值由特征权限在正常应用和恶意应用中出现概率的差异性决定。记权限恶意程度值函数为*φ*(*p*i)，定义为：

*φ*(*p*i)=*λ*m2(*pi*)*/λ*n(*p*i)*=λ*m(*p*i)*\*λ*m(*pi*)/*λ*n(*p*i) (4-2)

*λ*m(*p*i)*=C*m(*p*i)/*C*m，其中*Cm(p*i*)*为权限*p*i在恶意样本集中出现的次数，*C*m为恶意样本集总数。*λ*n(*p*i)*=C*n(*p*i)/*C*n，其中*C*n(*p*i)为权限*p*i在正常样本集中出现的次数，Cn为正常样本集总数。

**4.2.3 权限恶意度列向量**

权限恶意度列向量是所有特征权限恶意程度值构成的列向量，记权限恶意度列向量为*F*：

*F*= *φ*( *P*T) = (*φ*( *p*1)，*φ*(*p*2)，*…*，*φ*( *p*20))T (4-3)

**4.2.4 权限特征向量**

特征权限向量是由一个应用映射而成的，该向量可以反映特征权限在该应用中的分布情况。记权限特征向量为*Q*，则应用A的权限特征向量定义如下：

*Q*A*=* (*q*1，*q*2，*…*，*q*i，*…*，*q*20)，*q*i*∈*{*0*，*1*} (4-4)

其中*Q*A中每一项与*P*中的每一项一一对应。对*Q*A中每一项*q*i定义如下：设*P*A为应用程序A申请的所有权限的集合，如果*q*i对应项 *p*i*∈P*A，则令*q*i= 1，否则令*q*i= 0。

**4.2.5 应用恶意程度值**

应用的恶意程度值用来定量衡量一个应用产生恶意行为能力的大小，通过将该值与阈值进行比较即可判断应用的恶意性。记应用恶意程度值为*V*，则应用A的应用恶意程度值定义如下：

*V*A*=Q*A*F=*( *q*1，*q*2，*…*，*q*20)(*φ*(*p*1)，*φ*(*p*2)，*…φ(p*20))T*=* (4-5)

**4.2.6 恶意度阈值**

恶意度阈值是一个临界物理量，是判断应用是否恶意的一个标准。当应用恶意程度值*V*大于阈值时该应用为恶意应用，小于阈值时为正常应用。记恶意度阈值为*V*阈，其计算公式如下：

*V*阈= *k*1*V*m+ *k*2*V*n  (4-6)

其中0*≤k*1*≤*1，0*≤k*2*≤*1，*k*1+ *k*2= 1。*V*m是恶意应用样本集的平均恶意程度值，*V*n为正常应用样本集的平均恶意程度值。通过不断修正*k*1、*k*2值，确定最佳的阈值*V*阈，从而达到最好的检测效果。

**4.2.7 线性分类器**

对于给定权限恶意度列向量*F*和阈值*V*阈，我们可以构建一个线性分类器用于检测，记线性分类器函数为*G*(*X*)，其定义如下：

*G*(*X*)*= XF－V*阈=*φ*(*p*i)－*V*阈  (4-7)

对任意权限特征向量*X*，若*G(X)＞0*，则表明该应用是恶意应用，若*G*(*X*)*＜0*，则该应用是非恶意应用。

**4.2.8 算法伪码**

本论文使用LWD算法完成对应用的恶意性检测，首先需要获得应用的反编译文件，提取出其实际使用的权限报告，将权限报告、特征权限集合和权限恶意程度值列向量作为输入，算法如代码4-4所示。

**代码4-4 LWD算法**

|  |
| --- |
| 输入：*L*all-permission  //权限报告  *L*special-permission  //特征权限集合  *F*  //权限恶意度列向量  输出：malicious app OR normal app//应用的恶意性  funcation appVector(Lall-permission,Lspecial-permission)  1. for i=0 to *L*special-permission.Length  2. if *L*special-permission[i]∈Lall-permission  3. *Q*[i]=1;  4. else  5*. Q*[i]=0;  6. return *Q*;  function judge()  7. *Q*=appVector(*L*all-permission,*L*special-permission );//应用映射成权限行向量  8. values=*Q*\**F*; //计算应用的恶意程度值  9. if values > *V*阈 then //与恶意度阈值进行比较  10. malicious app;  11. else  12. normal app; |

**4.3 检测工具的设计**

**4.3.1 功能模块**

本论文将权限提取和恶意性及恶意行为检测等工作封装成一个自动化工具，以便用户使用。该工具的功能模块图如图4-3所示。



**图4-3 功能模块图**

**4.3.2 类图**

本工具采用Java语言实现，界面的开发主要使用Java的Swing工具包，数据库连接采用JDBC技术。类图可以反映该工具的静态模型结构，以及各个功能模块之间的组织关系。实现工具的部分类图如图4-4所示。



**图4-4 部分类图**

**5 实验与结果分析**

**5.1 实验环境搭建**

本实验是在内存为4GB，处理器为Inter(R) Core(TM) i5-4200U CPU @1.60GHz的计算机上完成的，操作系统为Windows7 32位。

本论文构建的工具基于Jdk1.8+eclipse-jee-neon-3-win32+apktool v2.0.0-RC4环境运行。

**5.2 实例研究**

本论文中的实例研究主要是为了验证权限检查是运行时进行检查，而不是在编译时进行。这一研究结果能够证明本论文研究的可行性。如果权限检查是在运行时进行，那么对于使用到某些系统资源但未申请权限的应用，本论文中的所构建工具能够在应用程序运行之前给出用户一个提示，避免程序在运行过程中出现异常。

**5.2.1 研究步骤**

第一步：开发一个Android应用程序，命名为TextSix，编写布局界面activity\_main.xml，该界面为应用程序启动时的主界面，此处代码略。

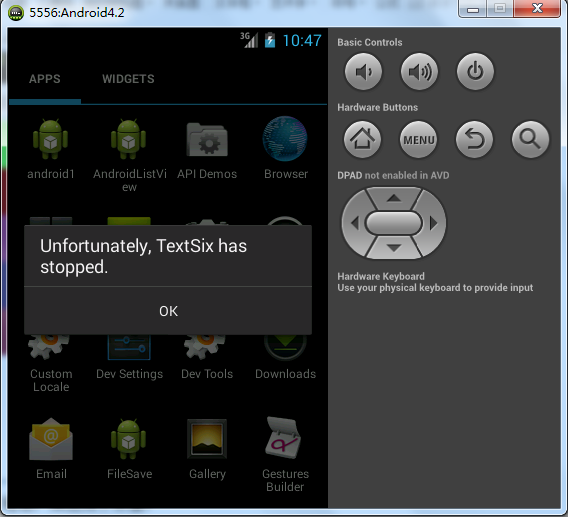
第二步：编写MainActivity.java类，添加拨打电话代码。

第三步：将该应用程序配置在一个已经启动的虚拟机上并安装运行，界面如图5-1所示。



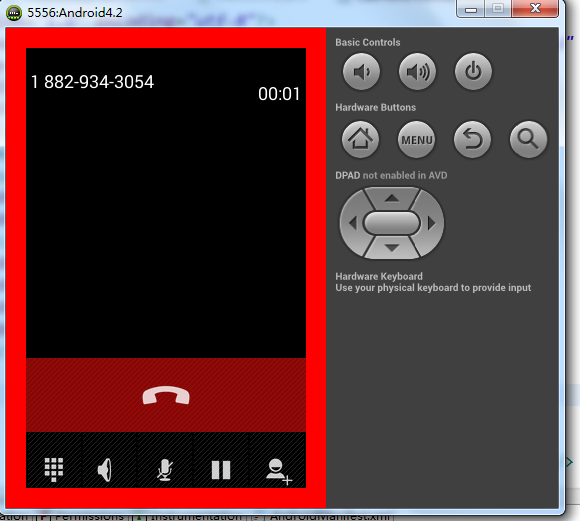
**图5-1 程序主界面**

第四步：拨打电话，程序出现异常，如图5-2所示。



**图5-2 程序异常界面**

第五步：添加拨打电话的权限，重新安装运行程序，弹出如图5-3界面。



**图5-3 拨打电话**

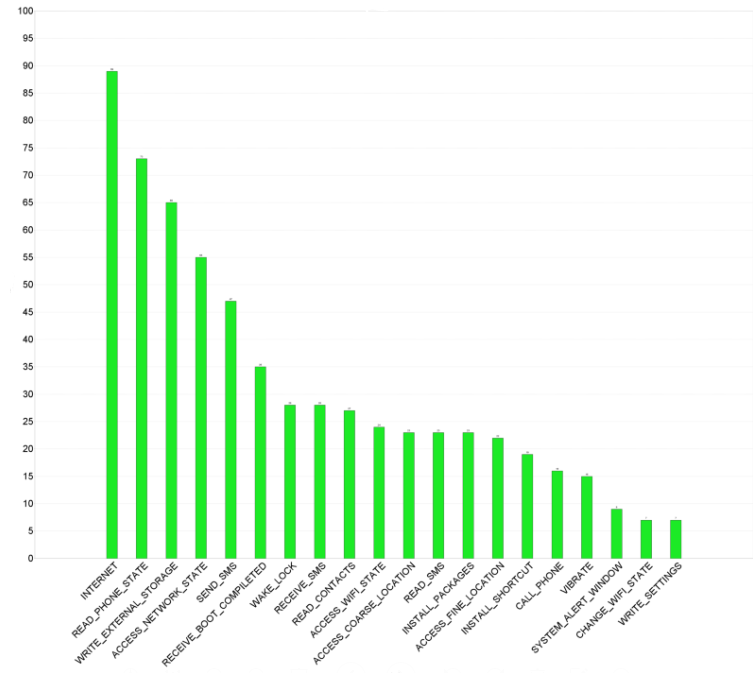
**5.2.2 结果分析**

通过对以上实例的分析可知，用户安装应用程序后，在使用拨打电话这一功能时，系统会进行权限检查，判断该应用是否拥有启动拨打电话这一Intent的权限。由此验证我们的观点：权限检查是运行时进行检查，而不是在编译时进行。

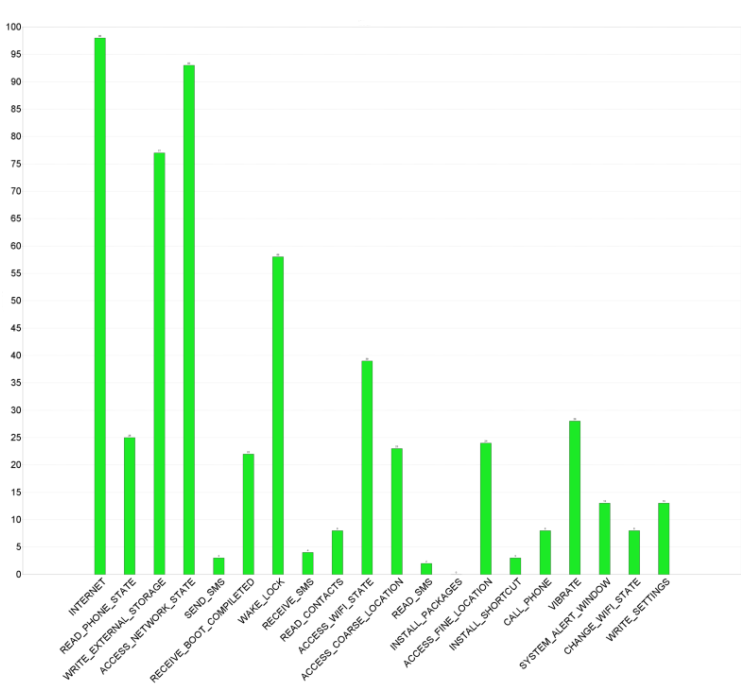
**5.3 实验步骤**

第一步：首先从各大网站论坛如卡饭论坛、吾爱破解等下载1147个apk实验样本，通过使用apktool工具对这1147个实验样本进行反编译，然后对能反编译出来的和不能反编译出来的apk进行统计，结果显示存在35%的apk是经过特殊处理的，不能通过apktool反编译出来反汇编代码，而65%的apk是可以反编译的，这说明本论文构建的工具可以对65%的apk进行检测分析，证明该工具有一定的有效性。

第二步：众所周知Google Play是Android应用的官方市场，其对发布的Android应用程序有着严格的审核机制，所以从该网站上下载的apk一定都为正常应用。本次实验首先从GooglePlay官网上下载150个正常apk，再通过各种不同渠道下载200个恶意样本，利用apktool对这350个样本集进行处理，分别得到100个正常应用的AndroidManifest.xml文件和100个恶意样本的AndroidManifest.xml文件，然后通过权限统计算法提取出在恶意样本中出现次数最多的前20个特征权限，并统计这20个权限在正常应用中出现的次数。统计结果分别如图5-4、5-5所示。



**图5-4 恶意应用特征权限分布图**



**图5-5 正常应用特征权限分布图**

从以上两幅图中可以看出，某些权限在恶意应用与正常应用中的分布情况存在着明显的差异，如SEND\_SMS权限在正常应用中很少被申请，而在恶意应用中却频繁被请求，这意味着SEND\_SMS属于敏感权限，可能会产生某些恶意行为。

对以上的柱状图进行统计，根据LWD算法中的公式（4-2）分别计算这20个权限的恶意程度值。结果如表5-1所示。

**表5-1 不同权限恶意程度值度量表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **权限** | ***λ*m(*p*)** | ***λ*n(*p*)** | ***φ*(*p*)** |
| 1 | INTERNET | 0.89 | 0.98 | 0.8080 |
| 2 | READ\_PHONE\_STATE | 0.73 | 0.25 | 2.1316 |
| 3 | WRITE\_EXTERNAL\_STORAGE | 0.65 | 0.77 | 0.5487 |
| 4 | ACCESS\_NETWORK\_STATE | 0.55 | 0.93 | 0.3253 |

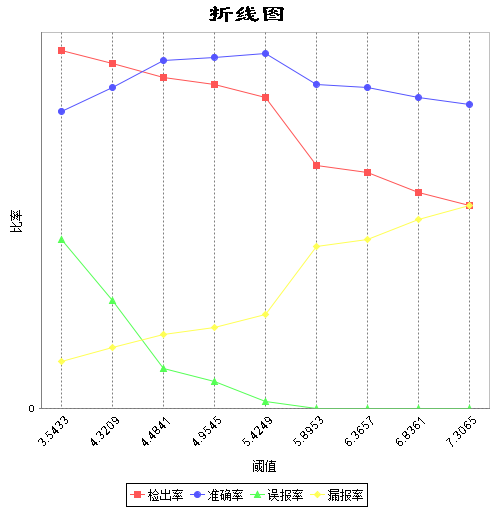
**续表5-1 不同权限恶意程度值度量表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | SEND\_SMS | 0.47 | 0.03 | 7.3633 |
| 6 | RECEIVE\_BOOT\_COMPLETED | 0.35 | 0.22 | 0.5568 |
| 7 | WAKE\_LOCK | 0.28 | 0.04 | 1.96 |
| 8 | RECEIVE\_SMS | 0.28 | 0.58 | 0.1352 |
| 9 | READ\_CONTACTS | 0.27 | 0.08 | 0.9113 |
| 10 | ACCESS\_WIFI\_STATE | 0.24 | 0.39 | 0.1477 |
| 11 | ACCESS\_COARSE\_LOCATION | 0.23 | 0.23 | 0.23 |
| 12 | READ\_SMS | 0.23 | 0.02 | 2.645 |
| 13 | INSTALL\_PACKAGES | 0.23 | 0.01 | 5.29 |
| 14 | ACCESS\_FINE\_LOCATION | 0.22 | 0.24 | 0.2017 |
| 15 | INSTALL\_SHORTCUT | 0.19 | 0.03 | 1.2033 |
| 16 | CALL\_PHONE | 0.16 | 0.08 | 0.32 |
| 17 | VIBRATE | 0.15 | 0.28 | 0.0804 |
| 18 | SYSTEM\_ALERT\_WINDOW | 0.09 | 0.13 | 0.0623 |
| 19 | CHANGE\_WIFI\_STATE | 0.07 | 0.08 | 0.0613 |
| 20 | WRITE\_SETTINGS | 0.07 | 0.13 | 0.0377 |

第三步：将上一步得到的20个*φ*(*p*i)值组成一个权限恶意度列向量*Q*，然后利用本论文构建的工具分别对60个恶意样本和60个正常样本进行反编译，将从应用反汇编代码提取出来的权限报告作为应用真实的权限信息，统计这些权限报告中特征权限的分布状况，将其构造成权限分布行向量*F*，利用公式（4-5）计算每个应用的恶意程度值。然后将60个恶意应用的恶意程度值进行平均，得到恶意应用的平均恶意程度值为7.777，同样计算出60个正常样本的恶意程度值的平均值为3.0729。利用公式*V*阈= *k*1*V*m+ *k*2*V*n，通过对*k*1，*k*2取从0.1到0.9不同的值得到不同的阈值，然后计算在阈值不同的情况下检测率和误报率的变化情况。阈值结果如表5-2所示，检测率、误报率如图5-6所示。

**表5-2 不同比例下的阈值表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| （*k*1，*k*2） | （0.1,0.9） | （0.2,0.8） | （0.3,0.7） | （0.4,0.6） | （0.5,0.5） |
| *V*阈 | 3.5433 | 4.3209 | 4.4841 | 4.9545 | 5.4249 |
| （*k*1，*k*2） | （0.6,0.4） | （0.7,0.3） | （0.8,0.2） | （0.9,0.1） |
| *V*阈 | 5.8953 | 6.3657 | 6.8361 | 7.3065 |

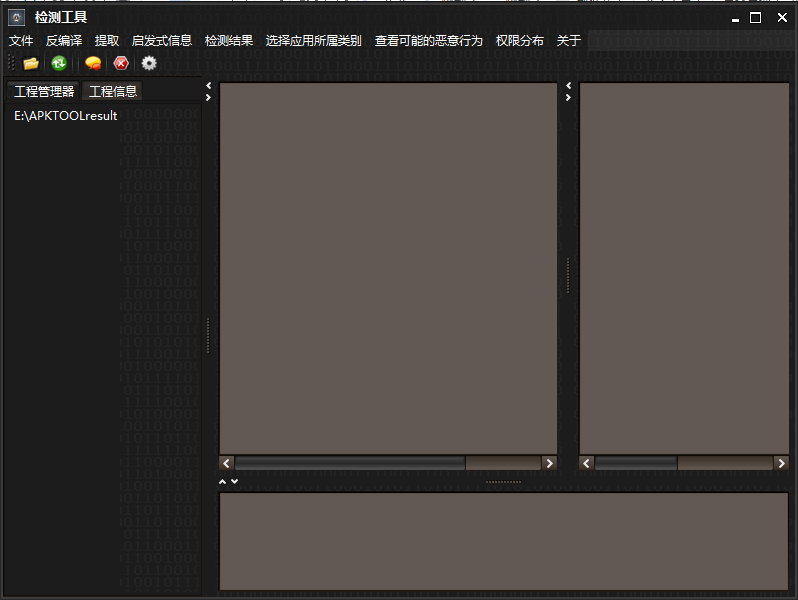


**图5-6 不同阈值下该工具的检测率、误报率和漏报率**

对于一个恶意应用检测工具来说，检测率和漏报率是检测该工具实用性最重要的两个因素。检测率越高且漏报率越低证明检测效果越好，通过对折线图展示的数据权衡对比选择4.9545作为最终的阈值，此时*k*1=0.4，*k*2=0.6，检出率为81.6%，漏报率为20%。

**5.4 权限提取与恶意行为检测工具**

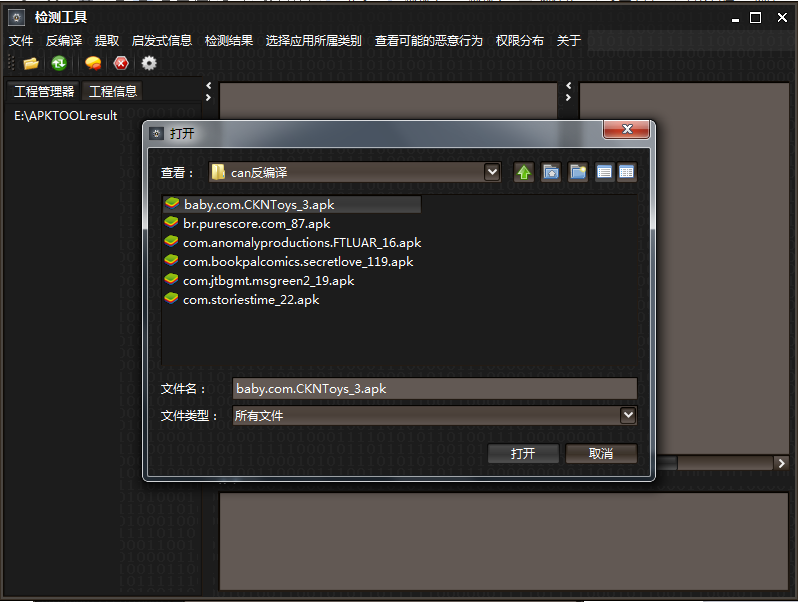
本论文构建的工具将对apk文件的反编译、权限提取以及恶意性检测等几大功能封装了起来，能够实现自动化检测。工具的界面如图5-7所示。



**图5-7 工具主页面**

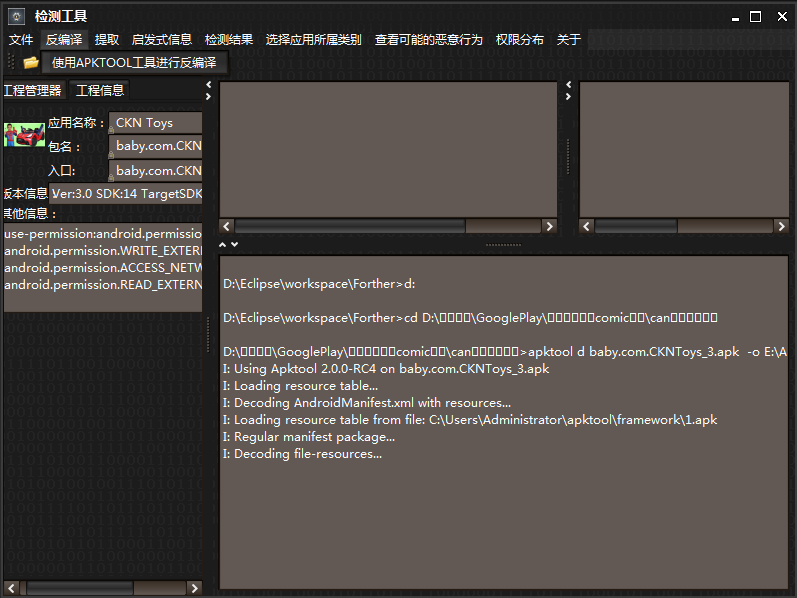
该工具的工作流程如下所示。

（1）选择需要检测的反编译文件，如图5-8所示。



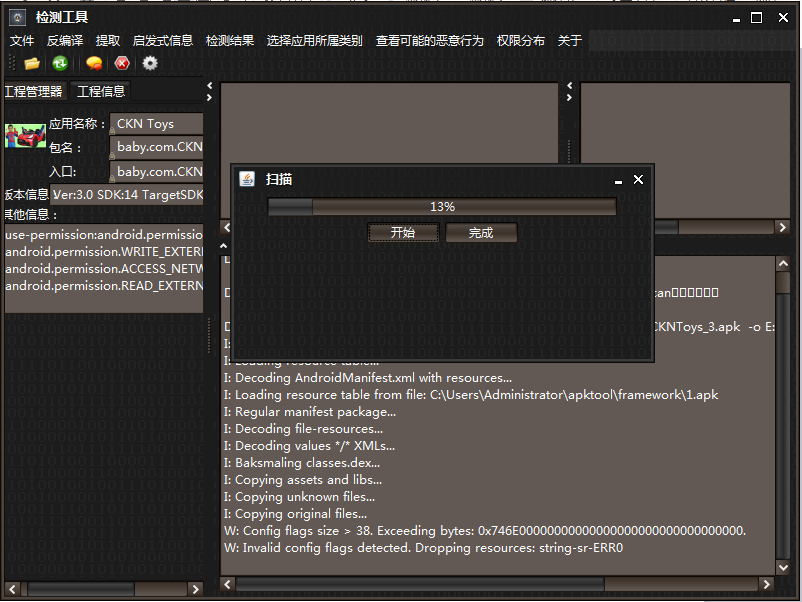
**图5-8 选择文件**

（2）加载完需要检测的apk文件，进行反编译，如图5-9所示。



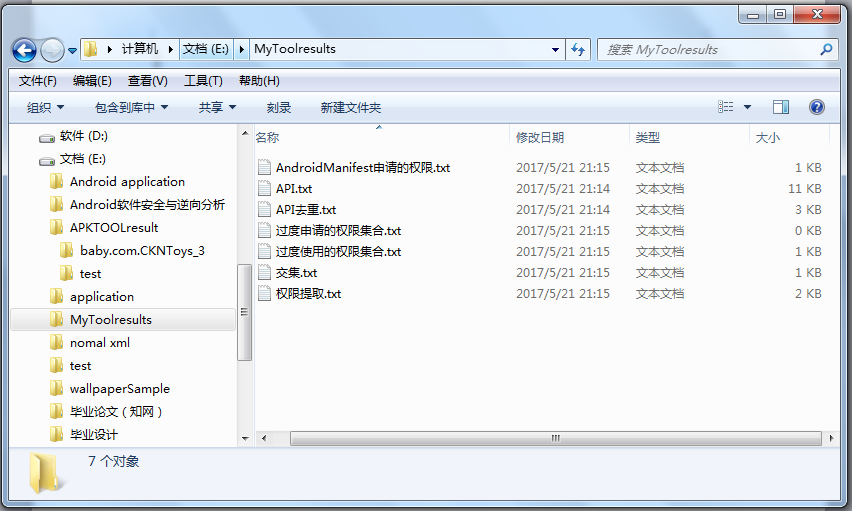
**图5-9 反编译文件**

（3）反编译完成得到反编译文件，提取应用程序调用的所有敏感API以及对应的权限报告，将其写在一个固定的目录下，如图5-10所示。



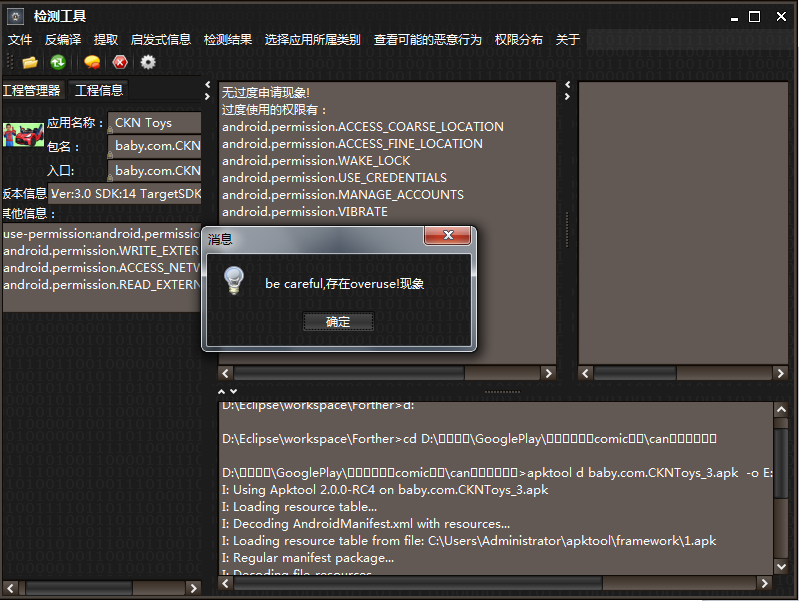
**图5-10 提取权限**

固定目录下生成的API调用信息、权限报告等如图5-11所示。



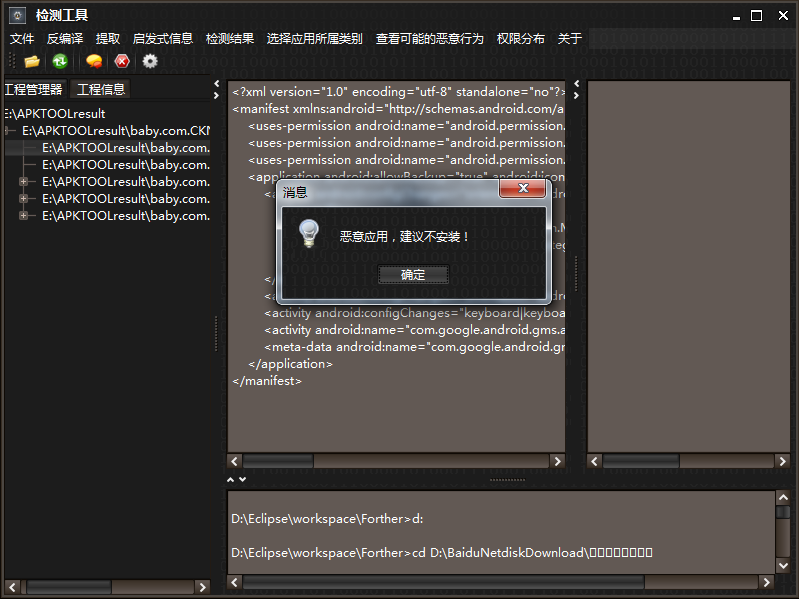
**图5-11 权限报告目录**

（4）提取完敏感API的调用信息，得到权限报告后，将权限报告与应用在AndroidManifest.xml文件中申请的权限做一个交集和差集，并给出一个启发式信息，如图5-12所示。



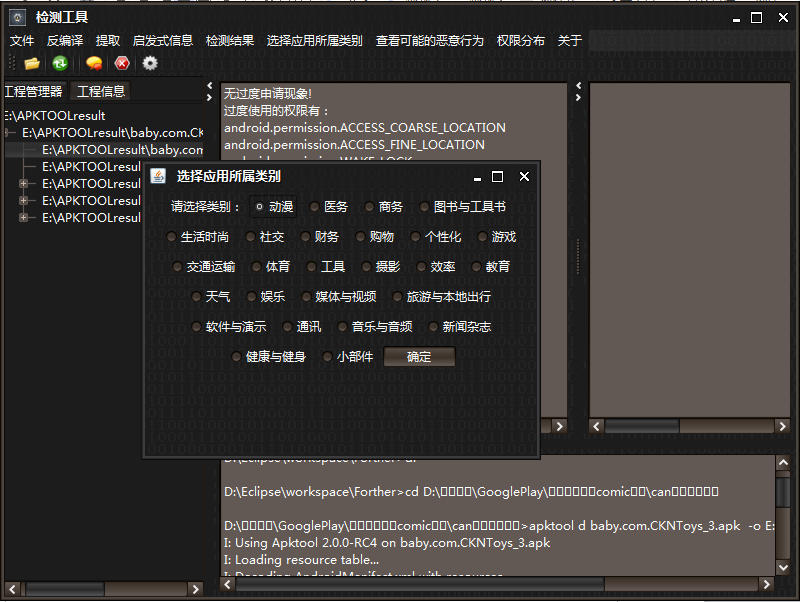
**图5-12 启发式信息**

（5）以权限报告作为输入，通过LWD算法计算该应用的恶意值，并给出检测结果，如图5-13所示。



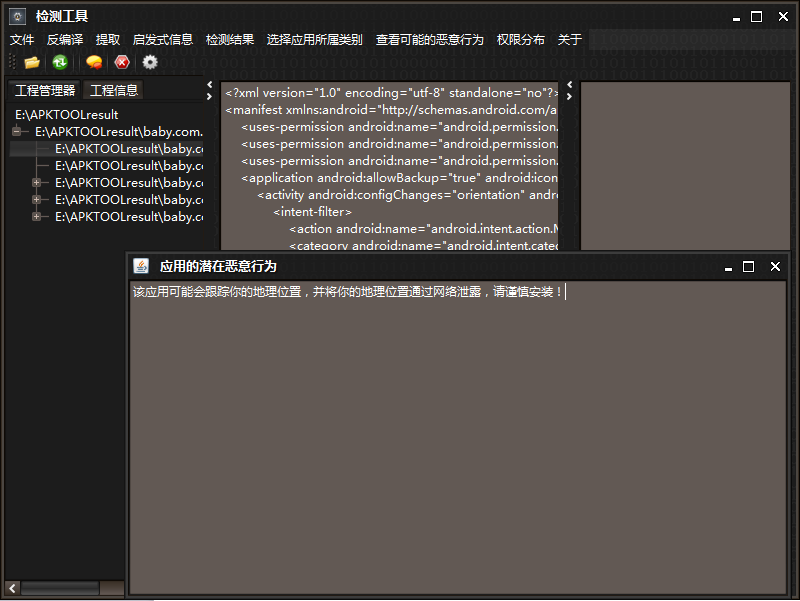
**图5-13 应用检测结果**

1. 判断应用为恶意应用后，选择应用所属的类别，如图5-14所示。



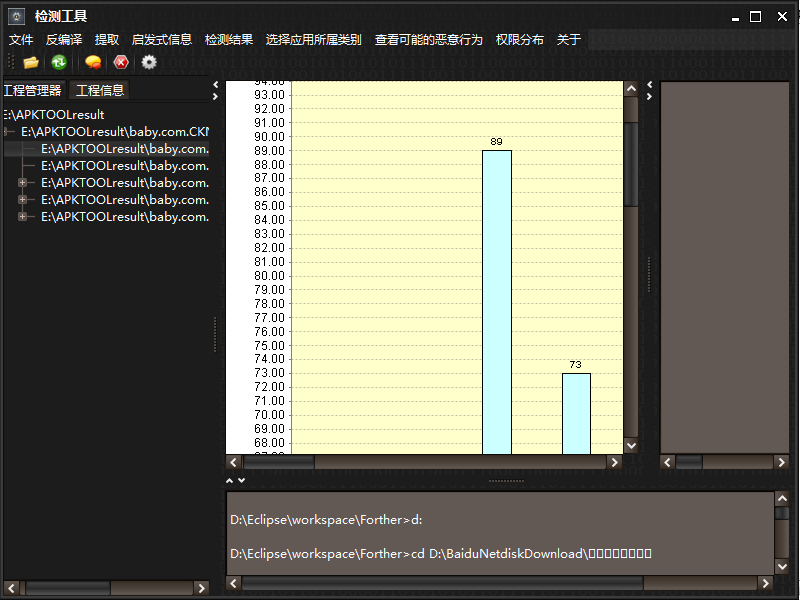
**图5-14 应用检测结果**

（7）向用户提示该应用可能会存在的恶意行为，如图5-15所示。



**图5-15 应用的潜在恶意行为**

（8）权限分布是属于附加信息，供用户查看特征权限在恶意应用和正常应用中的分布状况，如图5-16所示。



**图5-16 权限分布图**

（9）由于本论文对应用的恶意性判断取决于对权限的分析，准确、完整的API-权限映射表是影响权限提取结果的关键因素。尽管本论文所采用的参考文档是PScout最新发布的文档，但随着Android版本的不断更新，该网站对映射表的更新也从未停止，所以本论文中的API-权限映射表仍存在不完整性，加上本算法所采用的阈值是通过大量实验拟合出来的，实验数据量的大小也是影响判断准确性的关键，数据量越大，拟合出来的阈值越准确，判断结果的可信度越高。由于本次课题的时间及资源有限，所以采用的试验数据量也非常有限。基于以上种种因素，在本论文中构建的工具的准确率几乎不可能达到100%，通过对大量实验数据进行统计，引入衡量工具可信度的几个名词，并将统计结果展示给用户，使用户对该工具的可信度有一个具体的了解，如图5-17所示。



**图5-17 关于本工具**

* 1. **实验结果及分析**

在本论文中，首先需要给出以下几个名词的定义：

* 假阴性（Fail Negative）：恶意应用被检测出来为正常应用的总数；
* 真阴性（True Negative）：恶意应用被检测出来为恶意应用的总数；
* 真阳性（True Positive）：正常应用被检测出来为正常应用的总数；
* 假阳性（Fail Positive）：正常应用被检测出来为恶意应用的总数；

在以上参数的基础上，计算描述检测工具有效性的几个量：

* 漏报率：假阴性/（假阴性+真阴性）；
* 误报率：假阳性/（假阳性+真阳性）；
* 检出率：真阴性/（假阴性+真阴性）；
* 准确率：（真阴性+真阳性）/（真阴性+真阳性+假阴性+假阳性）。

本论文首先通过对大量的实验样本进行统计分析，计算出最佳阈值为4.9545，然后再利用该工具对120个新的apk样本进行检测，得到结果如表5-3所示。

**表5-3 实验结果**

|  |  |  |  |
| --- | --- | --- | --- |
| 检出率 | 漏报率 | 误报率 | 准确率 |
| 71.6% | 28.4% | 21.7% | 75% |

将该结果与目前市场上主流的手机杀毒软件进行对比，将工具所检测的120个实验样本分别用于金山毒霸、腾讯手机管家，得到的对比结果如表5-4所示。

**表5-4 对比实验结果**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 检出率 | 漏报率 | 误报率 | 准确率 |
| 本论文的工具 | 71.6% | 28.4% | 21.7% | 75% |
| 腾讯管家 | 76.6% | 23.4% | 38.3% | 69.2% |
| 金山毒霸 | 73.3% | 26.7% | 31.7% | 70.8% |

从表中可以看出，本论文中所构建的工具对恶意应用检测的准确率均高于这两大杀毒软件，但在漏报率方面该工具还稍显不足。但基于该工具属于一个轻量级的检测工具，运行速度快、占用存储器小，能够快速对手机的恶意性做出判断，这对应用的安全性检测研究具有重大意义。

本论文通过从大量apk样本的源码中提取权限，然后将权限报告与其在AndroidManifest.xml文件中申请的权限做交集和差集，发现大约80%的应用存在过度申请权限的现象，65%的应用存在过度使用权限的现象。在对apk文件检测过程中，工具会将这些过度申请的以及过度使用的权限展示给用户，以提供给用户一个简单的启发式信息。

**6 结论与展望**

**6.1 结论**

Android系统以其开源特性成为了当前最流行的智能手机操作系统，并在手机市场上占有着绝对的市场份额。凡事有利则必有弊，正是因为Android系统的开源性和庞大的用户量，使得Android系统成为一些恶意应用以及病毒软件的温床。一些第三方应用通过对Android系统提供的公共权限滥用系统资源、窃取用户的隐私数据。所以在Android安全方面的研究领域中，对权限进行提取并分析是一种很有效的方法。但目前大部分对Android权限的提取着重与在AndroidManifest.xml文件中提取，这种提取方法虽然简单，但提取的结果并不准确。

针对上述问题，本论文在对Android的体系结构，安全机制进行详细介绍，引入权限机制控制着第三方应用对系统中关键资源的访问，通过该机制能有效保护一些与隐私和安全相关的资源被任意访问，并分析了权限对于Android系统安全的重要性之后，提出了基于逆向工程的APP权限提取和恶意行为检测技术，本论文实现的工具能够对未安装的apk进行检测。它主要是通过扫描反编译文件，从中提取权限信息，然后对权限信息进行分析，判断应用的恶意性。在获取反编译文件的过程中，借助了apktool反编译工具对apk文件进行了处理，本工具的构建是基于Eclipse环境的，开发语言为Java语言，数据库为mysql数据库。

为了验证该工具的实用性，本论文对大量的样本进行了实验，首先验证该工具可以应用于65%的应用程序包中，然后通过实验验证该工具给出的结果具有一定的可参考性，并且与同类的其他杀毒软件相比具有较高的检测率，能够很好的对未安装的apk文件进行安装前检测，及时给出用户一些可参考的提示信息，具有一定的时效性。并且该工具具有运行速度快。

**6.2 展望**

本论文从API调用的信息获取其对权限的使用信息，但在应用程序开发过程中，不仅调用某些系统API需要相关的权限，组件之间的调用同样需要权限。并且对于从源码中提取API信息并获得权限报告的检测技术来说，完整、良好的API-权限映射表是决定提取效果的关键因素。本文选用的是PScout发布的旧版本的映射表，所以提取的信息可能并不完善，在以后的研究工作中，随着API-权限映射表的不断更新，可选用较为完整的API文档，进一步完善提取工作。并且对于Android程序的恶意性检测仅采用静态分析是不够的，还要结合动态分析方法进行检测，这一部分工作还待进一步研究和探讨。

在本论文的实验中所采用的实验样本数量并不是非常庞大，所以统计出来的结果可能存在一定的误差。从100个恶意样本中提取的20个特征权限并不足以表示恶意样本本身的特征，在这一方面需要做大量的实验，才能得出更为准确的结果。

对应用恶意行为的检测绝不只停留在应用所拥有的恶意权限组合，这只能反映应用的静态特征，若想检测应用到底访问了系统哪些敏感资源、产生了哪些恶意行为，还需根据API调用序列，使用污点分析等方法对应用进行进一步的检测，这一部分工作还有待于进一步探究。

**致 谢**

四年的大学生活一眨眼已接近尾声，在这四年充实的大学生活中，我不仅学到了专业知识，通过跟同学和老师的接触，也学到了很多做人做事的方法和技巧。尤其通过这次的毕业设计更是让我受益匪浅。

特别感谢刘晓建老师在我的毕业设计期间给我的所有帮助。在最初选定题目的时候，刘老师就非常认真帮我分析题目并给出我专业的建议。在后来的研究思路的整理以及研究内容的确定过程中老师更是循循善诱，帮我拓展思路。每次遇到一些自己想不通或解决不了的问题时，老师总是很有耐心地帮我寻找解决方案，给出我解决问题的具体思路。在论文编写的过程中，老师对我也非常的严格，自己更是不辞辛苦地帮我找出不足之处并给出准确的修改建议。在整个毕业设计过程中，我不仅学到了非常多的专业知识和一些解决问题的思路，更是被老师的严谨、认真地教学态度所折服。

感谢家人这么久以来的陪伴，在我遇到困难时给我鼓励和支持，给我提供良好的物质基础，这也是我安心学习的基础。

感谢室友和同学们，在我学习上遇到问题时给予我帮助，在生活上对我的百般照顾。

最后，感谢各位院领导精心安排的这次毕业设计，使我们在跨入社会之前做好充足的准备，更好地迎接以后在工作上遇到的种种挑战。

**参考文献**

[1] 腾讯移动安全实验室.腾讯移动安全实验室 2013 年手机安全报告[EB/OL].2014-01-10,http://m.qq.com/security\_lab/news\_detail\_223.html

[2] Felt A P,Chin E,Hanna S,et al. Android permission demystified[C].Proceedings of the

18th ACM conference on computer and communications Security.ACM,2011:627-638

[3] Enck W,M Ongtang,P McDaniel.On lightweight mobile phone application certification[C]ACM Conference on Computer and Communications Security,2009:235-245

[4] 王国栋,刘粉林,汪萍等.一种篡改检测与篡改定位分离的图像认证方案[J].计算机学报,2007:1880-1888

[5] Yerima S Y, Sezer S, Mc Williams G, et al. A new android malware detection approach using bayesian classification[C]. Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on. IEEE, 2013: 121~128

[6] Sanz B, Santos I, Ugarte-Pedrero X, et al. Anomaly Detection Using String Analysis for Android Malware Detection[C]. International Joint Conference SOCO’13-CISIS’13-ICEUTE’13. Springer International Publishing, 2014: 469~478

[7] 黄梅根,曾云科.基于权限组合的Android窃取隐私恶意应用检测方法[J].计算机应用与软件,2016,33(9):321-327

[8] 焦浩峰.基于权限的Android恶意软件混合检测方法研究[D].天津:天津大学,2014

[9] 郑吉飞.Android恶意代码的静态检测研究[D].湖北:华中师范大学,2013

[10] 彭凌.Android应用权限检测技术[D].安徽:中国科学技术大学,2015

[11] 童振飞. Android恶意软件静态检测方案的研究[D].南京:南京邮电大学,2013

[12] 秦中元,徐毓青,梁彪等.一种Android平台恶意软件静态检测方法[N].东南大学学报,2013,43(6):1161-1167

[13] L.Davi,A.Dmitrienko,A.Sadeghi,et.Privilege Escalation Attacks on Android[C].In Proceedings of 13th International Conference on Information Security,2011

[14] 丰生强. Android软件安全与逆向分析[M].北京:人民邮电出版社,2013