分 类 号：TP309.5　　密　　级：

学　　号：201508379

西安科技大学
XI'AN UNIVERSITY OF SCIENCE AND TECHNOLOGY

硕士学位论文

Thesis for Master's Degree

基于敏感权限和 API 的安卓恶意软件检测方法

申请人姓名：姜婷

指 导 教 师：刘晓建

学 科 门 类：工学

学 科 名 称：软件工程

研 究 方 向：软件安全

2018 年 6 月

西安科技大学

学位论文独创性说明

本人郑重声明：所呈交的学位论文是我个人在导师指导下进行的研究工作及其取得研究成果。尽我所知，除了文中加以标注和致谢的地方外，论文中不包含其他人或集体已经公开发表或撰写过的研究成果，也不包含为获得西安科技大学或其他教育机构的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

学位论文作者签名：姜婷　　日期：2018.6.12

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：研究生在校攻读学位期间论文工作的知识产权单位属于西安科技大学。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。本人允许论文被查阅和借阅。学校可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律注明作者单位为西安科技大学。

保密论文待解密后适用本声明。

学位论文作者签名：姜婷　　指导教师签名：刘晓建

2018 年 6 月 12 日

论文题目：基于敏感权限和 API 的安卓恶意软件检测方法

学科名称：软件工程

硕 士 生：姜 婷　　（签名）姜婷

指导教师：刘晓建　　（签名）刘晓建

摘 要

移动技术推陈出新的今天，Android 软件丰富了我们的日常生活，然而恶意软件数目随之持续上升，用户隐私泄露、手机扣费等事件频发，对用户的财产和隐私产生危害。因而，高效地检测安卓恶意软件成为智能手机安全领域的一个研究重点。考虑到恶意软件分析技术正确率和效率不匹配，本文在反编译技术的基础上，采用特征信息提取和处理以及模糊层次分析法提出了一种基于敏感权限和敏感 API 的 Android 恶意软件静态检测方法。

本文主要研究工作如下：(1) 研究了反编译处理过程。以安卓恶意软件家族名称为切入点，研究了不同恶意家族的恶意应用程序，并且以恶意家族名称进行分类统计。重点对样本软件采取反编译操作。(2) 特征信息提取。本文深入分析全局配置文件和 smali 文件，提取出特征信息，并以统一的格式进行存储。(3) 特征信息处理。对权限信息进行处理，得到权限频率值文本和权限敏感强度文本。对 API 信息进行处理，得到 API 频率值，同时根据敏感 API 确定 API 调用序列，从而确定 API 敏感强度。将这四类数据作为影响因子，使用模糊层次分析法确定阈值，从而提出静态分析安卓恶意软件的分析模型。(4) 方法测试与评估。通过实验对两类软件测试，统计出本方法和其他三种方法在五个衡量指标方面的数据，得出该方法有效性较高并且阈值的设置也相对合理。使用该方法分类精度达到85.5%，可以快速高效的对很大比例的安卓软件进行恶意性判别。

本文的创新点如下：(1) 权限频率值和 API 频率值的设定。以权限频率值设定为例说明，考虑到权限申请的次数间差距过大，为了既突显出差距，又减少差距过大带来的误差，本文采取频率线性操作的方式设置权限的频率值。(2) API 敏感强度值的设定。将敏感权限对应的 API 作为敏感 API，根据敏感 API 的溯源调用序列设置 API 敏感强度值。(3) 使用 FAHP 方法设定阈值。将影响因子的影响程度转化为数值，通过各影响因子的加权和分布情况确定阈值。

关 键 词： 静态检测；敏感权限；敏感 API；模糊层次分析法；反编译；

研究类型： 理论研究

Subject : **Android malware detection method based on sensitive permission and API**

Specialty : **Software Engineering**

Name : **Jiang Ting** **(Signature)** Jiang Ting

Instructor : **Liu Xiaojian** **(Signature)** Liu Xiaojian

ABSTRACT

With the development of mobile technology, software enriches our daily life, meanwhile the number of malware continues to rise, users' privacy leaks, mobile phone deductions and other incidents occur frequently, causing harm to the property and privacy of the users. Therefore, efficient detection of Android malware has become a research focus in the field of smart phone security. Considering the mismatch between the accuracy and efficiency of malware analysis technology, a static detection method based on sensitive permissions and sensitive API is presented based on sample decompile, feature information extraction, feature information processing and fuzzy analytic hierarchy process.

The following researches are done in this paper: (1) the process of decompile processing is studied. Taking Android malware family name as entry point, we studied malicious applications of different malicious families, and classified them by malicious family names. The emphasis is on the decompile operation of the sample software. (2) extraction of feature information. This paper analyzes the global configuration files and smali files, extracts feature information and stores them in a unified format. (3) feature information processing. The privilege information is processed to get the permission frequency value text and permissions sensitive intensity text. The API information is processed to get the API frequency value, and at the same time, the API sensitivity sequence is determined to determine the API sensitive intensity. Taking these four data as the influencing factors, we use the fuzzy analytic hierarchy process to determine the threshold, and propose a static analysis method of Android malware. (4) method implementation and evaluation. By testing two kinds of software tests, the ratio of this method and the other three methods on the five measure indexes is collected, and the validity of the method is higher and the setting of the threshold is relatively reasonable. Using this method, the classification accuracy is achieved, and it can quickly and efficiently discriminate malicious value for a large proportion of Android software.

This paper has the following innovations: (1) the establishment of permission frequency value and API frequency value. Taking the privilege frequency value setting as an example, considering the gap between the number of privileges is too large, in order to show the gap and reduce the error caused by the large gap, this paper adopts frequency linear operation to set the frequency value of the authority. (2) setting of API sensitive intensity value. The API corresponding to sensitive permissions is used as sensitive API, and API sensitive strength value is set according to the traceability calling sequence of sensitive API. (3) setting the threshold using FAHP. The influence degree of influence factors is converted to numerical value, and thresholds are determined through weighting and distributing factors.

Key words: Static detection; sensitive permissions; sensitive API; Fuzzy analytic hierarchy process (FAHP); decompile;

Thesis : Theoretical Research

目 录

1 绪　论

1.1 选题背景及意义

Android系统是由Google公司研发的操作系统，广泛应用在手机设备与PAD中。2016－2021年我国手机操作系统领域的研究报告统计[1]，第三季度Android系统在智能手机设备领域的市面占据比例高达87.5%[2]。凭借其开放的平台和丰富的应用软件，在手机市场分布方面，Android系统的使用率力压苹果的iOS稳居榜首。

随着大量Android软件丰富着公众的学习、生活和工作，各种隐私泄露、手机扣费、系统损坏事件不断发生，给使用者的生活工作造成巨大的威胁。安卓恶意应用程序的数量和种类增长速度较快，目前常见的恶意应用程序分为以下几类[3]，如表1.1所示。

表1.1 恶意软件分类说明

恶意软件名称	说明
恶意扣费	没有认证或授权，应用程序诱使用户执行恶意代码，导致财物损失。
隐私盗窃	没有认证或授权，应用程序访问或窃取用户的信息、个人隐私或敏感数据。
远程控制	没有认证或授权，应用程序静默接收和执行远程命令。
恶意代码传播	应用程序可能通过复制，感染，下载等方式，隐秘的传播自身或其他恶意代码。
费用消耗	没有认证或授权，应用程序可能自动拨打电话，发送短信，反复下载或其他方式对用户造成额外费用消耗。
系统损坏	恶意代码可能通过各种方式损坏系统功能，如移动终端，系统应用程序，用户文件，网络服务等。
欺骗	恶意软件可能欺骗用户伪造或篡改正常的应用程序。
流氓	恶意软件对系统或用户没有直接的损害。但是，它可能驻留在内存中，消耗CPU资源，自动绑定，弹出广告等。

从2012年到2016年，Android系统的年新增恶意应用软件数目从几十万增加到一千多万。新增加的安卓恶意软件平均每天就有38000个，无论是恶意软件的数量还是种类都增加的较快，安卓移动安全面临重大挑战。对Android程序进行恶意代码的分析和检测势在必行，方便、快捷、高效的检测技术成为手机用户的护航使者。图1.1是2012-2016年Android平台新增恶意应用软件样本数目，图1.2的(a)、(b)分别是2016年各季度Android平台恶意应用软件的新增数量和感染数量。

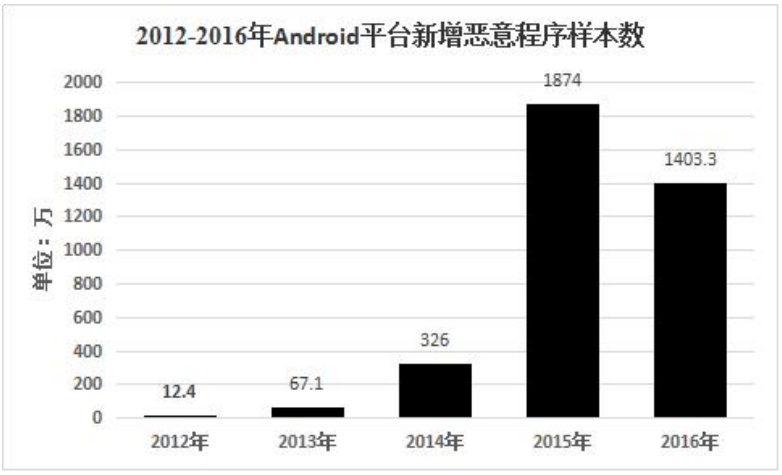


图 1.1　2012-2016 年 Android 新增恶意应用软件数

(a)

(b)

图 1.2　2016 年 Android 恶意应用软件新增数和感染数

随着软件保护技术不断更新，反编译与对抗反编译，静态分析与对抗静态分析，动态调试与对抗动态调试 ，重编译与对抗重编译等技术层出不穷[4]。软件保护技术的发展不仅是恶意代码与良性代码的斗争，更是开发者和目的不纯分子间的博弈。恶意软件分析的主要目的是对恶意应用程序的特征信息进行分析，进而研究相应的检测策略和防御方式，并形成可用的检测工具。便捷地获取实现恶意功能的特征信息，可以有效地增加恶意应用分析效率。同时对未知危害性的安卓软件进行有效的分析，可以降低当前恶意代码分析中面临的困难程度。安卓恶意软件逆向分析与检测方法将加快恶意应用程序检测技术的发展，对于提高对新型恶意安卓软件的判断效率，对于缓解目前移动安全严峻的现状具有不可忽略的重要意义。

1.2 国内外研究现状及发展方向

基于安卓平台应用程序分析技术可分为静态分析和动态调试两类方式。随着机器学

习技术的逐渐完善，很多研究人员将静态分析、动态调试分别与机器学习结合起来，应用分类算法进行软件检测[5]。静态行为检测不用运行程序，而是对其代码进行分析[6]，比较成熟的方法有可达路径分析、符号执行和污点分析，如图 1.3 所示。符号执行方法一般包括符号测试和定向模糊测试两类方式。动态检测需要运行软件，获取动态运行中的数据特征，一般使用模糊测试方法进行检测。

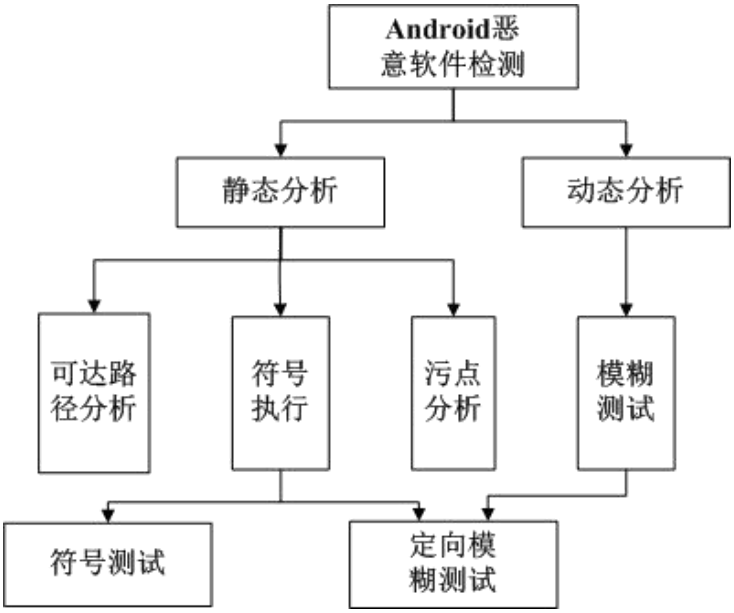


图 1.3　Android 恶意软件检测方法

动态调试方法必须运行 Android 软件，获取运行过程中的特征信息，耗费相对较大，一般使用安卓模拟器或沙箱机制来隔离运行。与此同时，触发安卓软件恶意行为的方式很多，恶意软件开发人员很容易绕过检测工具，存在较高的漏报率。典型的 Android 恶意软件动态检测方法包括应用程序运行中行为分析和运行过程中硬件资源的损耗分析等[7,8]。恶意软件静态检测方法不需要运行程序，一般通过逆向技术对应用程序进行处理，提取字符串信息、MD5 值、数字签名、权限、函数调用序列[9]等特征。静态分析方法不必运行软件，因此更加快速便利，然而误报率较高。代表性的静态检测方法主要包括提取程序依赖图和函数调用序列图等[10,11]。基于两类检测技术所存在的优点和不足，一些研究者提出了将两种方法进行结合的分析方法[12-14]。虽然可以结合两种方法的优点，提高检测的准确性，但是必须考虑动态调试所带来的时间损耗问题[15,16]。

1.2.1 静态分析研究现状及发展方向

静态分析(Static　Analysis)是在不运行安卓软件的情况下使用的。一般需要先对应用程序进行反汇编或反编译处理，然后通过反汇编代码或反编译代码来分析安卓应用程序行为。与动态分析的主要区别就是应用程序是否需要运行，特征信息的获取是否需要

动态得到的。静态分析一般常见的步骤是，获取特征信息，通过对特征信息的判断比对，确定软件是否对用户存在威胁。由于一般可以获取到的是应用软件的可执行文件，而无法获得源码，因此大多数分析人员采用逆向工程的方法，通过诸如apktool和Dex2Jar等进行反编译得到相应的dex、smali格式的文件，从dex、smali文件中提取特征信息。由于dex文件属于类汇编层次的程序，处理难度相对较大，而smali格式文件比较接近于Java语言，相对较容易理解，因此更易于识别和提取所需特征信息。IDA Pro和Androguard[17]在静态分析实验中应用的比较多。

Android软件常见的静态特征信息包括以下几类。(1) apk特征：Android软件包的大小，包中实体数目，每个类型文件数目，常见文件夹数目，MD5校验值等。(2) XML特征：XML元素、属性、命名空间，distinct strings的数目，每个元素名特征，每个属性名特征，权限[18-20]等。(3) dex特征：字符串，类型，类，原型，方法，字段，静态值，继承，操作码等[21]。(4) smali特征：类，父类，源文件，静态字段和实例字段，direct method和virtual method，参数，局部变量，接口，属性等。

通常，静态分析检测方法包括以下五类：(1) 基于MD5校验，MD5值是将安卓软件的特征信息提取并进行哈希处理得到的值，一但软件发生改变其MD5值就会发生变化；(2) 基于数字签名的检测技术，由于应用程序的数字签名信息可以唯一的确定安卓软件，因此，很多研究人员将数字签名信息作为判断软件恶意性的因素；(3) 基于正则表达式比对的检验技术，提取的特征信息是一些字符的排列组合。取这些字符和预先设定的正则表达式进行比对过滤，进而取得符合规则的符号集；(4) 基于权限分析的检验技术，权限和软件行为具有对应关系，因此权限通着作为软件恶意性的判断因子；(5) 基于源码API的分析检验技术。如图1.4所示。

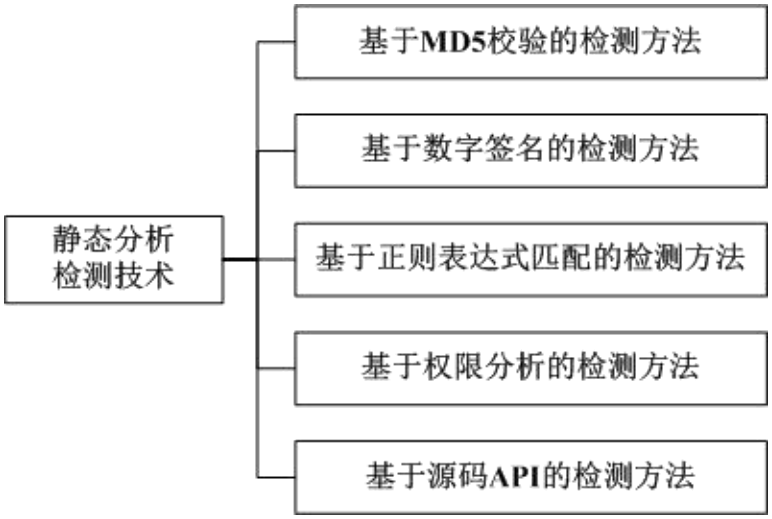


图1.4 常见静态分析检测技术分类

Enck W 等作者提出的静态检测工具 Kirin[22]，提取安卓软件的权限，将这些权限与其定义的单权限安全规则和多权限安全规则进行对比，从而进行判断。Fuchs AP 等作者提出的 SCanDroid 通过加载 java 字节码，进行数据流过滤，使用数据流分析方法进行字符串数据分析[23]。Chan PPF 等作者提出的 DroidChecker[24]，其工作流程是，首先将安卓软件进行反汇编得到 dex 格式程序，通过 dex2jar 将其 dex 格式转化为 jar 格式，一方面反编译得到源码，一方面通过 AndroidManifest.xml 文件提取组件形成组件列表，然后对源码和组件列表进行脆弱性检测，从而通过检测情况判断安卓软件是否是正常软件。Lu L 等作者提出的 Chex[25]，在 Android APK 文件中找到入口点，产生分离的数据流总结 SDS(Split Data-flow Summary)，处理每个单独的数据流总结得到分离的排序和排序数据流总结(Split Permutation & PDS Generation,PDS 即 Permutation Data-flow Summary)，最后启用劫持流记录。 Gibler C 等作者提出的静态检测方法 AndroidLeaks[26]，通过 AndroidManifest.xml 文件提取出 intent 信息，获取权限信息，将权限和 API 进行映射，最后用污点分析技术进行特征信息处理。Mohsin Junaid 等作者提出的 Dexteroid，主要包括两方面，其一是获得 Dalvik 字节码和 AndroidManifest.xml 文件，其二是从 APK 的逆向工程生命周期模型中获取驱动事件序列，结合两方面获得驱动的调用序列，从而产生调用的语句序列，根据调用语句序列匹配恶意行为，进而生成攻击报告[27]。Octeau D 等作者提出的 Epicc[28]，考虑到了组件间通信。这种静态检测方法采用过滤过的组件信息、权限、关键字符串值为特征信息。Cui XM 等作者提出的 CoChecker，它采用将组件列表，enter,intent-filter list,callbacks registered 等信息提取处理得到抽象语法树，根据抽象语法树得到每个方法的控制流图[29]，进行判断。Kim J 等作者提出的静态检测方法 ScanDal[30]，提取 Dalvik 虚拟机字节码的核心将其转化为中间语言，然后进行路径不敏感分析、内容敏感分析、流敏感分析，以分析 Android 软件的隐私数据外泄。文伟平等作者提出的恶意代码检测方法，获取安卓应用的权限信息，根据单个恶意权限规则库和危险权限组合规则库对比，从而进行判断[31]。

现在云计算，大数据等新型技术发展比较迅速，基于新型技术来进行 Android 应用程序的恶意代码分析势在必行。越来越多学者研究结合云计算[10,31]、机器学习的 Android 恶意软件分析方法[10,32-37]，相信必然给 Android 恶意代码分析方法引入新思路[38]。机器学习方法也可以用于静态分析[32,34-36,38-49]。Huang CY 等作者以权限作为特征，应用贝叶斯算法和 SVM 作为分类器，进行判断[50]。Sanz B 等作者提出 Puma，应用机器学习方法进行静态分析，提取权限作为特征信息，并使用不同分类器，包括 simple logistic、SMO 、IBK 、J48 、朴素贝叶斯、贝叶斯网络、随机树和随机森林等，进行分类学习[41]。Wolfe B[42]等作者将 dex 格式转化为 jar 格式，应用主要成分分析，矩阵内存存储稀疏化，去中

心化，获取 java 字节码中的 n-gram frequencies 作为特征信息，对特征信息应用不同分类算法，如SVM 、 RF 、 NB 、 KNN 、 BDT 和BBN，进行分类学习。Shabtai A[43]获取 APK 信息、XML 文件信息、dex 文件信息作为特征，使用Chi square 、information gain、Fisher Score进行特征选择，使用不同分类器进行测试。CEN L 使用包特征、类特征、函数特征作为特征信息，使用 2 类先验贝叶斯、逻辑回归作为分类算法，进行训练[38]。

目前 Android 恶意应用程序的逆向分析与静态分析方面主要存在的问题包括以下四方面：

(1) 由于软件保护技术的发展，很多恶意软件采取了对抗反编译技术、对抗静态分析技术（代码混淆技术、 NDK 保护技术、外壳技术）、对抗动态调试技术和防止重编译技术（检查签名、校验保护）使得对 Android 恶意软件的逆向和分析更加困难[4]。

(2) 恶意应用增长速度快，种类多，数量大，构建一个敏感特征数据库是一个漫长而持续的过程，需要学者持之以恒的努力，一直更新数据库。

(3) 静态检测方法速度快，轻量级。缺点是误报率高。动态分析方法以分类精度相对高为优点，但存在的不足是重量级，消耗时间长。

(4) 利用 zero day 漏洞的 Android 应用，更是 Android 恶意软件分析面对的又一大挑战。

Android 恶意应用程序的逆向分析与检测方法的发展方向主要有以下三个方面：

(1) 特征信息再处理。特征信息作为软件恶意性检测的判断依据，采用对特征信息进行再处理的方式是恶意软件检测的基础和前提，准确高效地描述应用程序特征不仅可以节约特征提取阶段的时间，对恶意应用分析正确率的提高更是大有裨益。

(2) 结合静态分析和动态分析技术的优势。传统的静态分析技术和动态调试技术优势明显，但均存在瑕疵，因此结合两类方法的优点，屏蔽其缺点的方案是 Android 恶意应用分析方案的又一个热点。

(3) 模糊层次处理方法。通常安卓软件恶意性的影响因子有多个，各影响因子的重要程度不尽相同，因此采用模糊层次分析法将各影响因子间定性的关系转化为定量的关系，便于数字化处理。

研究快速、有效的 Android 恶意软件分析方法是 Android 移动安全以后发展的重心和必然趋势。在安卓恶意软件过多的今天对 Android 恶意软件检测技术成为手机用户的护身符，本文主要针对可进行反编译操作的 Android 恶意软件进行特征信息提取处理。在对这些特征进行研究的前提下，提出一种结合静态分析和动态监测优点屏蔽其缺点的静态检测方法。

1.2.2 动态分析研究现状及发展方向

动态分析[51]主要是通过运行安卓软件发现应用漏洞，常用的技术是模糊测试，通过在沙箱中运行应用，了解应用行为[52]，分析是否对系统产生影响，是否泄漏用户敏感信息，是否篡改或者含有诱骗信息，通过改变部分代码观察行为是否改变。早期有代表性的动态分析工具如 Stowaway[53]，重新定制 Android 系统的工具如 Quire[54]。

Dietz M[54]等作者提出的 Quire 属于定制的轻量级智能手机操作系统，它追踪设备上基于进程间通信的调用链。 Enck W 等作者提出的 TaintDroid 采取动态污点分析技术，实时监视智能手机上的隐私敏感数据[55]。Huang J 等作者提出的 Asdroid[56]模型将安卓软件的 dex 文件转化为 jar 文件，选择 WALA 作为分析引擎，主要是基于 WALA 进行分析。Burguera I 等作者提出的 Crowdroid 是基于行为的安卓应用动态监测方法，其客户端 Crowdroid 主要监视 Linux 内核系统调用并将这些预处理的数据发送给中心服务器，远程服务器主要分析数据并为每个使用安卓应用进行交互用户创建系统调用向量[57]。文伟平[31]等作者提出的恶意代码检测方法使用动态监测的方法，利用 trace 工具记录系统调用的行为。赵洋等作者提出的使用沙箱的安卓恶意应用程序动态检测方法[58]。Zhaoguo Wang 等作者提出的 DroidChain[59]是一种基于行为链方法的动态检测方法，将软件行为抽象为有函数名、属性、顺序属性、符号、静态字符串、连接符等组成的模型，并总结了四种行为链模型，分别为隐私泄露、财务管理短信、恶意软件安卓和权限提升。Bugiel S 等作者提出的 XManDroid 改善 Android 系统的监视机制，来监测和检验软件运行过程中的权限提升攻击[60]。机器学习方法也可用于动态分析[50,61,62]。Shabtai A 等作者提出的基于行为的安卓设备恶意软件检测框架 Andromaly[61]应用机器学习方法进行动态分析，它实时获取应用级信息、操作系统信息、时序信息、内存信息、键盘、网络信息、硬件信息、电源信息等作为特征信息。Amos B[50]等作者提出了分布式训练和评估安卓恶意软件的框架，包括测试平台，加载远程和本地信息，设备预备，设备注入，客户输入仿真器，恶意执行监控与特征向量收集。

1.3 研究内容

为了高效简便的分析 Android 恶意应用程序，避免恶意应用程序对用户造成不利影响，本文设计了一种结合行为特征的静态分析方法。通过分析正常安卓软件和恶意安卓软件的特性，本文将权限信息和 API 信息作为特征信息，接着对这些信息进行处理，得到四个影响因子，根据影响因子确定阈值。具体内容主要包括以下部分。

(1) 研究了反编译处理过程。以安卓恶意软件家族名称为切入点，研究了不同恶意家族的恶意应用程序，并且以恶意家族名称进行分类统计。重点对样本软件采取反编译

操作。

(2) 特征信息提取和处理。本文深入分析全局配置文件和 smali 文件，提取出特征信息，并以统一的格式进行存储。对权限信息进行处理，得到权限频率值文本和权限敏感强度文本。对 API 信息进行处理，得到 API 频率值，同时根据敏感 API 确定 API 调用序列，从而确定 API 敏感强度。

(3) 方法设计。由于传统静态分析存在严重误报，动态检测耗时，两类方法均存在弊端这个问题，本文重点研究确定既能快速分析安卓软件恶意性又能提高准确率的方法。以敏感 API 的调用序列模拟应用程序动作，继而提高恶意性判断的准确性。将权限频率值、API 频率值、权限敏感强度值和 API 敏感强度值作为四个影响因子，使用 FAHP 方法将影响因子的作用程度转化为数值，通过各影响因子的加权和分布情况确定阈值。从而提出静态分析 Android 恶意应用程序的方法。

(4) 方法测试与评估。通过实验对两类软件测试，统计整理本方法和其他三种方法在五个衡量指标方面的比例，得出该方法有效性较高和阈值的设置也相对合理。使用该方法分类精度达到85.5%，可以快速高效的对很大比例的安卓软件进行恶意性判别。

1.4 论文结构安排

本文采用六个章节来阐述，详细说明如下：

第一章 绪论。首先简单描述本方法的背景和作用。其次说明了国内外关于静态分析技术和动态分析技术的发展现状。最后描述了本文后续章节设置。

第二章 相关知识介绍。主要讲述了 Android 系统的安全机制，介绍了 Android 应用软件包的文件结构，dex 文件体系结构和 smali 文件体系结构，Android 逆向工程与重打包技术和 Android 软件保护技术。

第三章 Android 软件静态特征的提取。本章重点说明了该静态检测方法的过程，然后反编译 1072 个恶性安卓软件和 409 个良性安卓软件，分别得到相应的 smali 文件。分别获得恶意和正常样本 AndroidManifest.xml 中的权限数据。递归获取每个 smali 包里面任何以.smali 为后缀的文本，提取出 API 数据并进行统计。

第四章 Android 软件静态特征处理。主要详细描述了对特征数据的处理情况。对于权限信息，统计每个权限出现频率，并进行线性处理。同时设定权限的敏感强度。对于 API 信息，统计 API 的频率，并进行线性处理。同时设定 API 的敏感强度。然后根据 FAHP 方法确定权值，从而确定阈值。

第五章 方法测试与评估。首先介绍了实验环境，样本的选取，然后对大量恶意测试软件和良性测试软件进行处理，获取其去冗余后的特征信息。依据静态检测方法对测

试样本进行检测，根据评估标准对比其他方法对本文提出的检测方法进行评估。

第六章 总结与展望。主要是整理本文所做的工作，并简要分析了本文需要改进的部分，同时介绍安卓恶意软件检测领域以后需要研习的方面。

2 相关知识介绍

2.1 Android 安全机制

Android 系统在 Linux 基础上添加了自身的功能模块，因此除了保存有 Linux 系统的安全机制外，Android 系统还有其自身的安全机制如表 2.1。Android 特有的安全机制包括权限机制、签名机制、组件封装机制和沙箱机制[63]。沙箱机制由 Dalvik 虚拟机和可移植操作系统接口用户安全机制组成。如图 2.1 说明了安卓软件签名的生成过程。其中安卓应用程序权限等级分类说明如表 2.2。

表 2.1 安卓安全机制

应用框架层名称	安全机制	存在攻击
应用程序层	杀毒软件、防火墙、入侵检测系统	基于应用程序的攻击
应用程序框架层	权限机制、签名机制、组件封装机制	
安卓本地库及运行环境	内存管理单元、强类型语言安全、移动设备安全	基于系统核心程序的攻击
Linux 内核	POSIX USER、文件访问控制	基于 Linux 内核的攻击
硬件		基于硬件的攻击

表 2.2 安卓权限等级

等级	说明
normal	normal 用于修饰低风险权限，只要在 APK 在 AndroidManifest.xml 中申请了就可以使用。在 APK 安装时，可以看到相应的安全级别，但不需要用户确认授权。
dangerous	dangerous 用于修饰高风险权限，APK 使用这类权限时，系统会明确要求用户进行确认。
signature	signature 安全级别表明，如果其它应用需要使用当前应用自定义的权限，则必须使用和当前相同的 key 做签名认证。对于仅限应用内部使用的权限，选择 signature 安全级别比较合适。权限授予时，系统不会通知用户。
signatureorsystem	主要针对系统应用之间的通信。当在系统应用中，定义该安全级别的权限时，其它系统应用均可以申请该应用的授权。对于普通应用（非系统），该权限的用途退化到与 signature 级别一致。

通常在 Android 软件包中的 AndroidManifest.xml 配置权限，比如设置允许查看网络

状态功能：

< uses - permission android : name ="android.permission.ACCESS_NETWORK_STATE" / >

以安全级别为 signature 为例，设置权限的等级方式例子如下：

< permission android : name ="ru.android.apps.permission.C2D_MESSAGE" android : protec tionLevel ="signature" / >

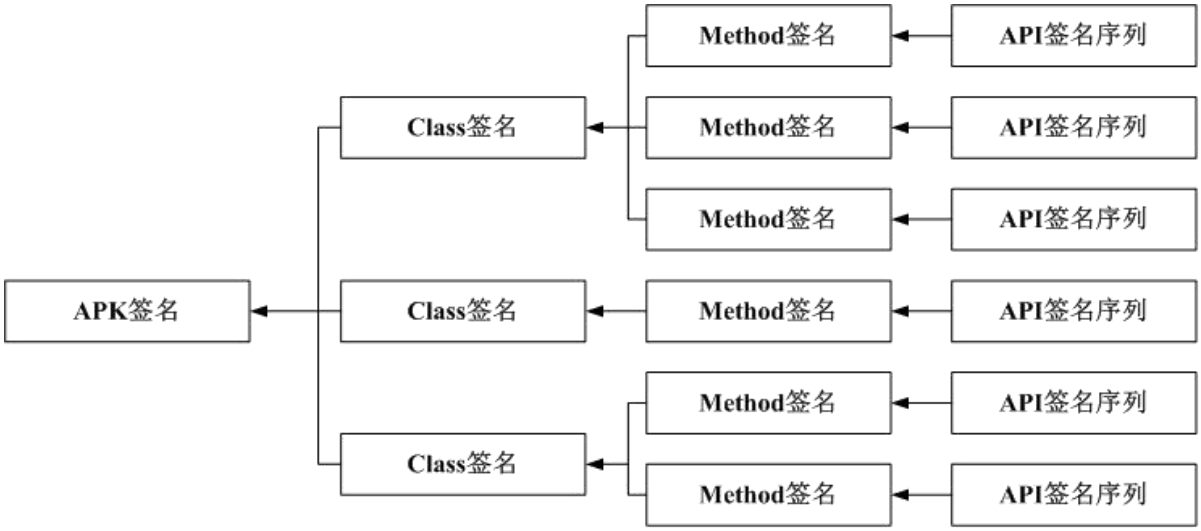


图 2.1 安卓软件签名的生成过程

2.2 Android 应用软件包结构

Android 应用程序包有其特有的软件结构，与之相关还有 dex 语言体系和 smali 语言体系。dex 文件是接近于汇编的一种语言规则。smali 语言是反编译 apk 文件生成的文件格式，有自己的语言规范。

2.2.1 Android Package 文件结构

任何安卓应用后缀都是.apk，即 Android application package file。由于 apk 相当于将 zip 后缀名换位 apk，因此可以用 zip 解压缩。以恶意样本 bad(1).apk 为例，解压后的软件包包括 assets 文件，里面存放的是一些设置信息； META - INF 包中主要保存 Android 软件的签名信息。任何 apk 都有独一无二的签名数据，用来保证 Android 软件的完整性，如果安卓应用中的程序存在改动，对应的签名信息也会发生变化； res 文件主要保存资源信息，比如图片，字符串，布局数据等；AndroidManifest.xml 文件主要用来设置应用名称，版本，编码方式，权限信息，引用的库文件，组件名称，组件属性等全局配置信息；classes.dex 文件是该应用在 Dalvik 虚拟机上的可执行文件；resources.arsc 保存编译过的二进制资源数据。APK 文件的结构如图 2.2。

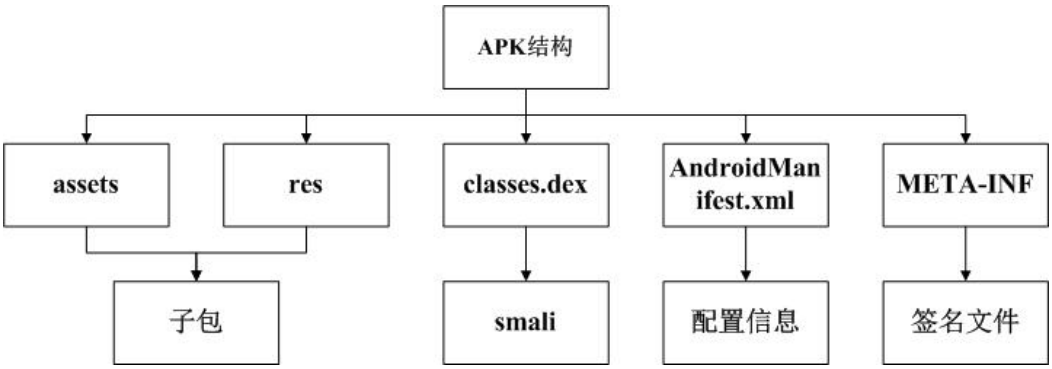


图 2.2 APK 文件结构

2.2.2 dex 文件结构

dex(Dalvik execute)，Android软件对应的dex一般由九个部分构成。dex header，即dex文件头，该文件指定了dex文件的一些属性，同时记录了其他部分内容在dex文件中的偏移量。dex文件的索引结构区一般包括string_ids 到class_def 这六部分。 data 部分保存真正的数据信息。 link_data 是静态连接数据区。dex文件结构如图2.3。

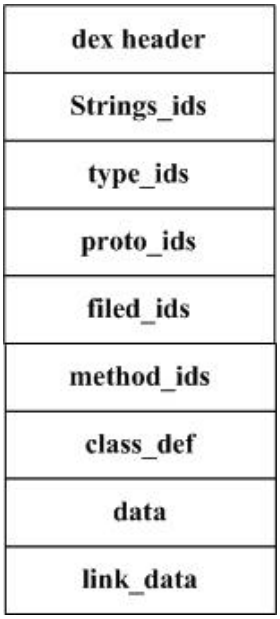


图2.3 dex文件结构

dex文件使用的数据结构包括u1、u2、u4、u8、sleb128、uleb128、uleb128p1。其中u1代表一字节的无符号数，与uint8_t 等价。u2 代表两个字节的无符号数。u4 代表四个字节的无符号数。 u8 代表八个字节的无符号数。 sleb128 代表有符号 LEB128，可变长度1～5 字节。 uleb128 代表无符号 LEB128，可变长度1～5 字节。 uleb128p1 代表无符号 LEB128 的数值加一，可变长度1～5 字节。

Odex即Optimized DEX，指的是经过优化处理的dex文件，主要有odex文件头、dex

文件、依赖库和辅助工具四部分组成。其中辅助数据段主要负责保存dex文件被优化后增加的一些信息。

2.2.3 smali 文件结构

apk 文件经过工具 apktool 反编译生成 smali 格式数据包。smali 语言代码比汇编程序更容易掌握和运用，比较接近 Java 程序，比 dex 程序更加容易读懂。smali 格式的前三行主要说明该类的基本信息，通用的模式如下：

.class ＜访问权限＞ [修饰关键字] ＜类名＞

.super <父类名>

.source <源文件名>

其中.class用于说明目前的类名，.super声明目前类的父类名称，.source用来说明目前类的源文件名称。剩下的是类的主体部分，一个类包括多个字段或方法。smali 文件中字段又分为 static field 与 instance fields。方法包括 direct methods 和 virtual methods。方法在“.method”到“.end method”之间定义，包括参数个数，调用函数等。

2.3 Android 逆向工程与重打包技术

2.3.1 Android 逆向工程

因为 Android 应用软件并不是开源的，我们下载到的是 apk 文件，并不是应用源程序，对于想要了解安卓软件是否是恶意软件的人员来说，逆向工程是个很不错的操作手段。几种格式的转换如图 2.4。

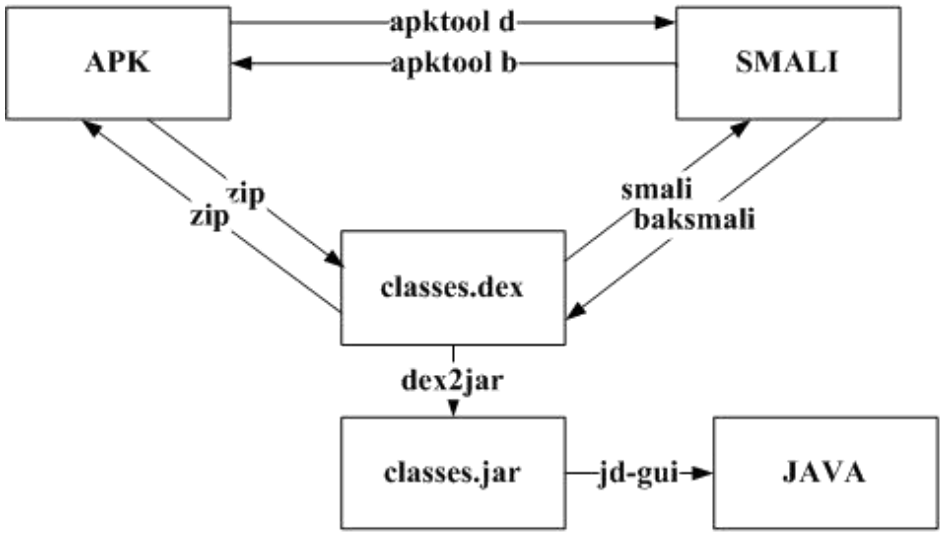


图 2.4 apk 文件转化图

安卓逆向工程就是将 apk 转化为人或工具可识别的文件格式。目前比较完善的安卓逆向工程流程有两类，一类是将 apk 转化为可读性较高的 Java 代码，一类是将 apk 转化为中间语言 smali 文件。

2.3.2 Android 重打包技术

Android 重打包过程又称为植入技术，是将安卓软件改动后重新进行打包签名的过程。鉴于 Android 平台的开源共享，Android 应用软件越来越多，也产生了很多第三方 Android 软件商城。目前在第三方软件商城重打包问题还是很严重的。一些开发人员或出于给应用程序加入广告或者个人签名，从而获取利益；或出于在应用程序中添加恶意代码，在进行重打包，以收集使用者的身份信息，欺骗诱诈用户，实现自己的不法目的。使用重打包一方面侵犯的研发者的知识产权，破坏研发者的利益，一方面植入恶意代码的应用程序损害了用户的利益。

2.4 Android 软件保护技术

随着安卓恶意软件的增长和软件保护技术的快速发展，应用软件保护技术的安卓恶意应用程序数量增多，使得逆向处理和分析 Android 程序更加困难。目前成熟的对抗 Android 应用检测方法的软件保护策略有很多。

2.4.1 对抗反编译技术

对抗反编译技术主要采用处理 apk 软件或者 dex，使得 apk 不能使用一些特定方式对其进行处理，或者处理后不能获取有效的反汇编程序，从而保证安卓恶意软件不被研究人员截获处理。

绕过特定工具的一般过程是检查这些特定工具转换 apk 或 dex 文件过程中的薄弱点，然后在预绕过的程序中进行使用。让转换工具处理这些处理过的 apk 软件时出现错误，使得转换过程无效[4]。使用处理过的应用程序能够在移动设备上无差别下载运行，而不能用特定转换工具进行处理。

2.4.2 对抗静态分析技术

目前可以对抗静态分析技术有如下三种：

第一种是代码混淆技术，大多采用 Native 代码取代 Java 代码，是很好的代码保护措施。由于大多数人还不具备灵活分析 Native 代码的能力，因此使用代码混淆技术可以很大程度的对应用程序进行保护。ProGuard 工具可以进行混淆处理。

第二种是 NDK 保护技术，逆向 NDK 程序的汇编代码难度很高，对逆向人员的要

求特别高，不容易实行。

第三种即加壳保护技术，是采取对代码加密的方式。Java 程序不存在外壳保护这个问题。外壳保护主要针对的是 Android NDK 编写的 Native 代码，逆向 Native 代码的难度很高，如果还要考虑外壳的问题，更是要求高。当前对 ARM Linux 内核程序进行加壳处理的工具是 upx，它是一款开源的加壳工具，支持多平台、多类型的文件加壳，其主页为 http://upx.sourceforge.net/。

2.4.3 对抗动态分析技术

动态分析通常选择调试器对 Android 软件进行挂钩，提取 Android 软件运行过程中的特征数据。在应用程序中用于检测调试器的程序，若一旦调试器连接应用程序，会触发一系列的操作。

通常拿到要检测的 Android 软件，分析人员会在模拟器上运行。然而，一旦 Android 软件无法在模拟器上运行，将给 Android 恶意应用程序逆向与检测带来很大的影响。安卓模拟器与实际的 Android 设备有很多差异，研究人员可以通过在命令提示符下执行“adb shell getprop”查看并对比两者的属性值，通过观察和比较，存在三个属性值，如表 2.3 所示。

表 2.3 安卓软件属性值

属性值	模拟器	正常安卓手机
ro.product.model	sdk	手机型号
ro.build.tags	test-keys	release-keys
ro.kernel.qemu	1	无

2.5 本章小结

本章主要描述了 Android 平台特有的安全机制。其次，介绍了安卓软件包的结构，其 Dalvik 虚拟机上的可执行文件 dex 的结构，并简要介绍了 apk 文件反编译处理得到的 smali 文件体系。接着描述了 Android 软件的逆向分析方法，说明了成熟的两种逆向过程，还描述了 Android 软件的植入方法。最后从三方面说明了安卓软件存在的保护措施。

3 Android 软件静态特征信息提取

Android 应用程序种类比较多，同时安全隐患也比较突出。第三方应用市场恶意软件的数量也在逐渐增加，人们期待有效简洁快速地检测恶意软件。本文提出的 Android 恶意软件静态检测方法，尽可能的从用户的实际需求考虑，重点在有效和快速。为了考虑函数行为对 Android 软件的作用，本文以 API 敏感强度做为影响因子，其设置方式依赖于函数的方法调用序列。权限敏感强度则依赖于权限对系统或用户的危害程度，分为四个等级。API 频率信息和权限频率信息的设定，都是在忽略了无效信息，屏蔽了干扰信息的基础上进行的。在本章中，安卓软件特征信息的提取，一般分为两个步骤：先对 apk 文件进行反编译处理，接着再获得需要的特征信息。本章先描述了该静态检测的方法过程，接着介绍了 Android 软件的反编译过程，重点描述了权限信息和 API 信息的获取。

3.1 方法概述

本文所提出的安卓恶意软件静态分析方法，同时考虑到了应用程序的静态特征和应用程序各模块行为对软件的影响。采用静态分析的方法不必在沙箱中运行安卓应用程序，因此，有效地节约了时间，提高了正确率。采用权限信息和 API 信息作为特征信息，其中 API 敏感强度是依据 API 调用序列确定的，有效地描述了软件行为。该方法的基本思想是：统计分析设置四个影响因子。权限数据刻画 Android 软件产生某一行为的权利，因而将其作为影响因素。考虑到不同样本权限的影响程度不同，设置权限频率值。考虑到不同权限的危险程度不同，设置权限敏感强度进行评判软件。API 信息用来刻画应用程序如何完成某一行为，因而将其作为影响因素。API 频率值用于表示不同样本 API 的影响程度。API 敏感强度用来刻画不同 API 的影响程度并且使用其刻画应用程序行为。最后使用 FAHP 确定四个影响因子的作用大小。该静态检测方法框图如图 3.1 所示。

该静态检测方法的步骤如下所述：

(1) 样本反编译：对分过类的良性样本和恶意样本采取批量反编译处理，生成相应的 smali 格式程序，为特征信息的获得做准备。

(2) 特征信息提取：本文所提取的特征信息包括权限信息和 API 信息。权限信息的提取相对简单，API 信息的提取主要是分析 smali 文件的特点，从而提取涉及的 API 信息。

(3) 特征信息处理：通过统计得到权限的频率信息和 API 的频率信息，进行线性处理得到权限频率值和 API 频率值。依借权限对用户的危害程度对其进行分类，每一类分别设置不同的强度值。根据敏感权限关联的方法确定敏感函数，敏感函数作为源点，遍

历每个样本，提取和统计出方法调用序列，根据方法调用序列设置 API 敏感强度值。

(4) 阈值设定：分别根据模糊层次分析法为四个影响因子设定不同权值，根据影响因子与权值的乘积之和 Fi，统计恶意样本和正常样本的阈值范围，将最能准确区分恶意性和良性的分界点 Fi 设定为阈值 F。

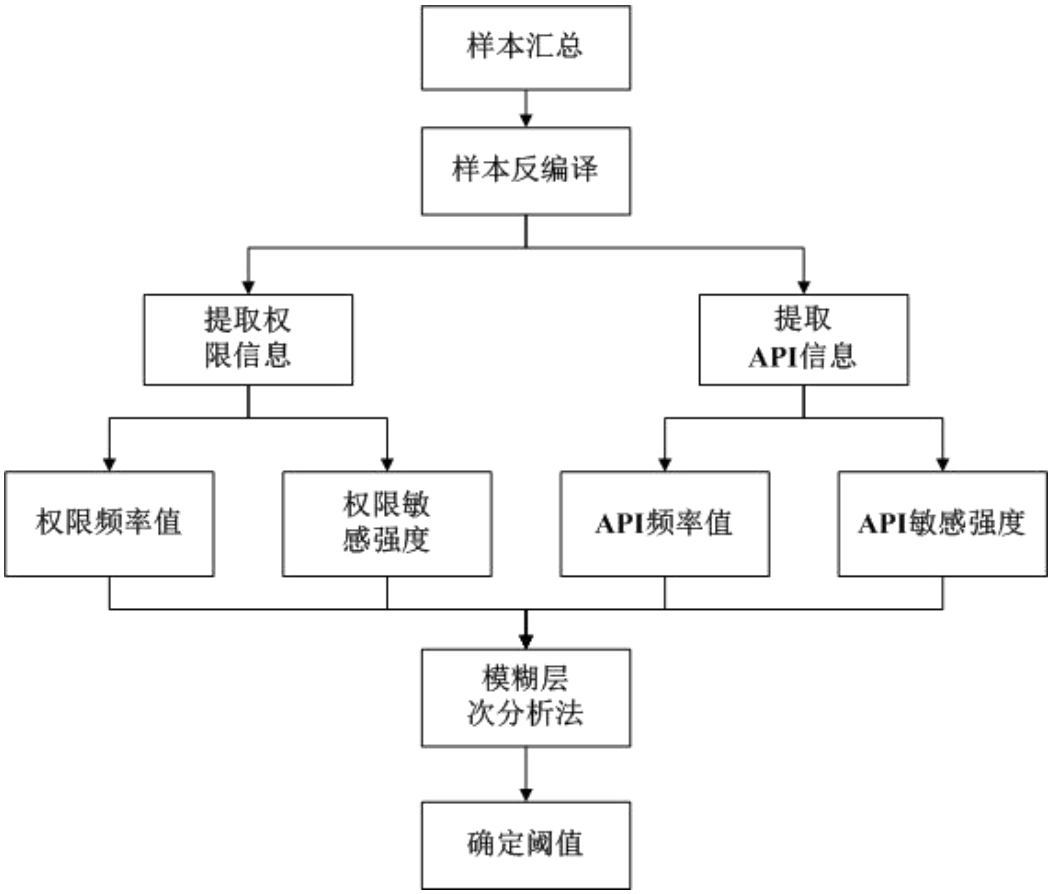


图 3.1 静态检测方法框图

3.2 Android 软件反编译

本实验从virusshare网站上下载了1072个恶意样本，从Google商城下载了409个良性样本作为良性样本库分别进行实验。其中样本的选择不固定，种类尽可能的多。用360安全卫士和百度管家两种工具进行检查，恶意样本和良性样本的确定需两个工具均结论相同。

良性样本包括视频播放器、聊天软件、浏览器软件、办公软件、输入法、音乐软件、游戏娱乐软件、教育软件、理财软件等多种类型的应用程序。恶意样本的选择按安卓恶意软件家族名称进行查找下载统计，共整理出恶意家族80个，以家族名称首字母为顺序，统计整理得恶意软件家族表如表3.1。

表 3.1 恶意软件家族统计表

序号	字母	名称
1	A	ADRD,AnserverBot,Asroot,AdSMS,AnserverBot,Arspam,Adware.Airpush
2	B	BgServ,BaseBridge,BeanBot,BaseBrid
3	C	Crusewin,CoinPirate,CarrielQ!Android
4	D	DroidDream,DroidDreamLight,DroidKungFu,DogWars,DroidCoupon,DroidDeluxe,, DroidKungFuSapp,DroidKungFuUpdate
5	E	Endofday,ExploitLinuxLotoor
6	F	FakePlayer, FakeNotify ,FakeBattScar,FikeNefix,FakePalyer,FakeTimer ,FlexiSpy, Foncy,FakeInstaller,FakeRun,FakeDoc
7	G	GPSSMSSpy,Geinimi,GGTracker,GamblerSMS,GoldDream,GingerMaster,Gone60, GoldDream,Gappusin,GinMaster
8	H	HippoSMS,Hamob
9	I	Iconosys,Imlog
10	J	jSMSHider,Jifake
11	K	KMin
12	L	Lovetrap,Lotoor,LeNa,LinuxLotoor
13	M	MobileTx,Moghava
14	N	Nickyspy,NickyBot
15	O	Opfake
16	P	Pjapps,Plankton
17	R	RogueSPPush,RogueLemon,RootSmart
18	S	SMSReplicator,SndApps,Spitmo,SmsHider,SerBG ,SendPay,SMSreg
19	T	TapSnake
20	W	Walkinwat
21	Y	YZHC,YZHCSMS
22	Z	zHash,Zsone,Zitmo

对于下载的安卓软件，得到的是程序的应用包，并不是源码。研究人员需要将APK转化为可读性较高的格式进行分析检测。目前比较成熟的路线有两种，一种是将APK转化成Java源码，一种是转化为介于汇编和Java之间的smali文件格式。APK仅是将ZIP后缀更改得到的，解压便可得到包含class.dex的文件夹，这个文件是该应用程序在Dalvik虚拟机上的可执行文件，掌握理解不易，要求研究人员的汇编阅读能力强。因此，使用 jd - gui 将jar文件转化为Java源码。本方法选择转化为smali进行处理，详细说明如下。

用户从第三方应用程序市场下载得到的是apk软件包，解压安装即可运行。然而无法得到其源代码进行分析，故而可以对apk包中的dex程序采取反编译操作，转为可读性较高的程序格式，以便于分析人员分析研究。本文实验使用apktool这个工具进行反编译。apktool不仅可以将apk转化为smali文件，还可以对smali文件进行编译打包。使用apktool d -f <源.apk绝对地址> -o <目的内容绝对地址>可以生成smali格式。使用Notepad++可查看编辑smali程序。若开发人员没有对源码进行混淆处理操作，则可看到源码，如果进行了混淆处理，只能看到a,b,c,d等名字的函数名称。然而，并不是所有的dex文件都可以反编译生成smali文件。采取apktool工具进行转化，可能存在错误。不过命令行一般会回显错误类型，有的问题修改后仍可以使用apktool处理。比如命令行提示错误，显示“Exception in thread "main" brut.androlib.AndrolibException.....”，则说明apktool的版本太低了，需要更新最新版本。目前apktool已经更新到2.0版本了，需要JDK1.8兼容。在Windows平台，运行cmd.exe程序，直接输入apktool会列出该工具的用法，如表3.2。以恶意样本virus(1).apk为例，反编译后生成virus(1)文件，其中virus(1)的文件结构目录如图3.2所示。

表3.2 apktool命令统计表

apktool d[ecode] [options] <file_apk>		
-f	--force	Force delete destination directory.
-o	--output <dir>	The name of folder that gets written.
-p	--frame-path <dir>	Uses framework files located in <dir>.
-r	--no-res	Do not decode resources.
-s	--no-src	Do not decode source.
-t	--frame-tag <tag>	Uses framework files tagged by <tag>.
apktool b[uild] [options] <app_path>		
-r	--force-all	Skip changes detection and build all files.
-o	--output <dir>	The name or apk that gets written.
-p	--frame-path <dir>	Uses framework files located in <dir>.

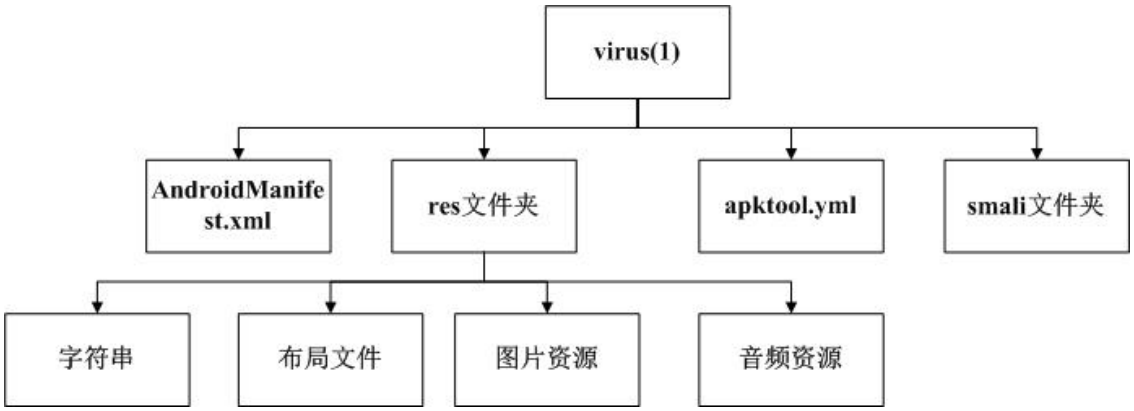


图3.2 virus(1)文件结构目录

其中res文件夹中存放的是安卓软件的资源信息，包括图片信息、音频信息、布局信息、字符串信息等。这些文件的下一层和包括的内容和软件研发时源码软件包的文件组织结构相同。smali文件中保存有为每个类产生的.smali程序。AndroidManifest.xml文件内容包括应用名、版本信息、权限信息、引用的库文件、组件名称、组件属性等全局配置信息。apktool.yml是反编译成功的标志。

本次实验利用脚本进行统一反编译dex格式，转化为相应的smali程序。分别批量反编译1072个恶意软件及409个良性软件，相应的生成smali文件。恶意应用设置为virus(i).apk ($1 \le i \le 1072$)，恶意样本反编译后生成的包设置为virus(i) ($1 \le i \le 1072$)。良性软件编号为benign(j).apk ($1 \le j \le 409$)，反编译后生成文件编号benign(j) ($1 \le j \le 409$)。

3.3 权限特征

权限作为通行证，是Android软件拥有某种行为的权利。权限在一定程度上限制了用户的误操作和手机系统存在的危险操作的进行，是手机安全的一层保护。由于行为与权限挂钩，存在一些对应关系，用户要想实现某一功能需获得相关权限。比如想要查看短信，必须有READ_SMS权限。因此，将权限作为Android软件恶意性的判据之一，是主要的影响因素。

其中权限可以分为两类：直接读写设备的底层(low - level)权限与间接读取设备的高级(high - level)权限。高层权限和底层权限存在差异，高层权限通过Framework层的权限检查代码来进行权限控制的[4]。

若用户未请求权限却进行特定的行为，Android软件在运行过程中会回显“Security Exception”错误。Android系统的源码ContextImp1.java文件提供了权限检查方法。依据检查内容与处理方式的不一样，可以将权限的检查方法进行分类。每个权限的控制粒度与处理方法都不同，根据处理对象的不同，又可以分为应用程序一般的权限检查与Uri的权限检查。权限检查的对象可以是Permission字符串匹配、pid、tid、Uri。

将权限信息作为特征信息，当应用程序申请READ_SMS权限，如果该应用程序是短信，那么无可厚非。但如果这个应用程序是个拍照软件，其调用READ_SMS权限，并且调用访问网络，连接某网站，上传信息的权限，那么READ_SMS在这个应用中便属于敏感权限。有的研究人员将敏感的权限组合作为特征信息数据库，与待测试样本中的数据库进行匹配。权限一方面限制了应用程序的行为，但其粗粒度的机制使得权限很容易被目的不纯的开发人员绕过。若将权限信息作为唯一的特征信息，存在较高的误判率。恶意软件开发人员可能会过多的申请无效的权限作为干扰，因此，本文将权限因素作为影响因子之一。

通过分析AndroidManifest.xml文件的特点，本文分别获取恶意应用和正常应用中的权限信息，为每个样本创建一个文本用来存储提取的权限信息，提取权限的流程图如图3.3，其算法描述如下。

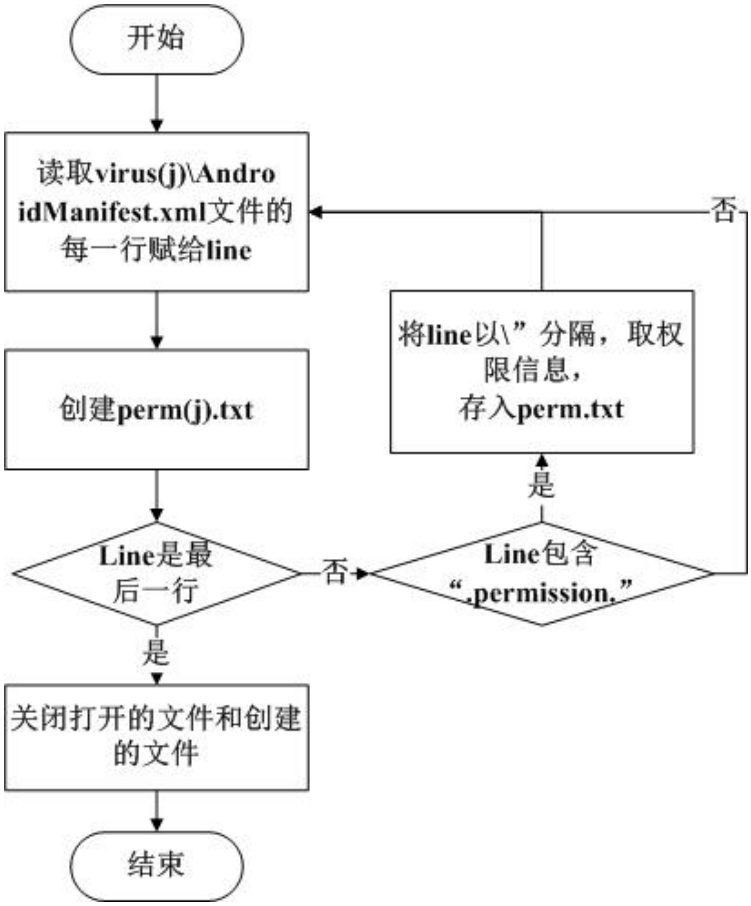


图3.3 权限提取流程图

对已提取的权限信息采取去重复操作，将所有权限进行汇总后，去除重复的内容，以每个权限占据一行的形式保存到权限汇总文本中。对正常软件和恶意软件采取同样的操作，将获得如表3.3的权限信息。

表3.3 权限信息

部分恶意样本中汇总权限	部分良性样本中汇总权限
android.permission-group.NETWORK	android.permission.C2D_MESSAGE
android.permission.ACCESS_ASSISTED_GPS	android.permission.ACCESS_ALL_DOWNLOADS
android.permission.ACCESS_CACHE_FILESYSTEM	android.permission.ACCESS_BACKGROUND_SERVIC
android.permission.ACCESS_CELL_ID	E
android.permission.ACCESS_COARSE_LOCATION	android.permission.ACCESS_COARSE_LOCATION
android.permission.ACCESS_COARSE_UPDATES	android.permission.ACCESS_COURSE_LOCATION
android.permission.ACCESS_DOWNLOAD_MANAGER	android.permission.ACCESS_DOWNLOAD_MANAGER
android.permission.ACCESS_FINE_LOCATION	android.permission.ACCESS_DOWNLOAD_MANAGER
android.permission.ACCESS_GPS	_ADVANCED
android.permission.ACCESS_LOCATION	android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_LOCATION_EXTRA_COM	android.permission.ACCESS_GPS
MANDS	android.permission.ACCESS_LOCATION
android.permission.ACCESS_MOCK_LOCATION	android.permission.ACCESS_LOCATION_EXTRA_CO
android.permission.ACCESS_NETWORK_STATE	MMANDS
android.permission.ACCESS_SURFACE_FLINGER	android.permission.ACCESS_MOCK_LOCATION
android.permission.ACCESS_WIFI_STATE	android.permission.ACCESS_MTK_MMHW
android.permission.ACCOUNT_MANAGER	android.permission.ACCESS_NETWORK_STATE
android.permission.ADD_SYSTEM_SERVICE	android.permission.ACCESS_SUPERUSER
android.permission.AUTHENTICATE_ACCOUNTS	android.permission.ACCESS_WEATHERCLOCK_PROV
android.permission.BATTERY_STATS	IDER
android.permission.BIND_APPWIDGET	android.permission.ACCESS_WIFI_STATE
android.permission.BIND_WALLPAPER	android.permission.ACCOUNT_MANAGER
android.permission.BLUETOOTH	android.permission.AUTHENTICATE_ACCOUNTS
android.permission.BLUETOOTH_ADMIN	android.permission.BATTERY_STATS
android.permission.BROADCAST_PACKAGE_ADDED	android.permission.BIND_ACCESSIBILITY_SERVICE
android.permission.BROADCAST_PACKAGE_REMOVE	android.permission.BIND_NOTIFICATION_LISTENER_
D	SERVICE
android.permission.BROADCAST_SMS	android.permission.BLUETOOTH
android.permission.BROADCAST_STICKY	android.permission.BLUETOOTH_ADMIN
android.permission.BROADCAST_WAP_PUSH	android.permission.BROADCAST_STICKY

3.4 API 特征

想要实现某一功能，就需要使用API实现，因此将API作为判断Android恶意软件的一个因素。每个恶意软件都有其特色的属性，属于其个性，考虑到恶意软件的共性，因此提取API的频率作为影响因素。考虑到各个API对使用者的影响不一样，还需对API数据进行处理。可以这么说，若一个应用程序想要实施某个恶意的行为，必然要使用敏感的API实现。即便不使用Google提供的API，而使用开发人员自己编写的API，其最终还是要调用作用对象，调用敏感API。

由于反编译的过程中，将Android软件的每个类都转化为相应的.smali程序，因此获取API的过程相对较复杂。需要遍历每个smali文件，提取出所有API数据，包括API名称、参数和API返回值。不同于权限，每个样本提取的API都存在大量的冗余，去重必不可少。需要分别将恶意样本和良性样本中提取的API信息进行去重，然后将所有的API信息进行汇总，统计出恶意样本和良性样本中出现的API。

通过分析恶意样本和良性样本中出现的函数方法信息，可以发现两类样本都调用了大量的初始化函数<init>。这个数据对实验结果的作用很小，因此，采取忽略这类数据。在实验中发现，由于样本过多，某些样本所占存储空间较大，即使只是提取函数信息，对所有API信息采取汇总操作后，会产生较大的API数据文件。就良性样本的函数信息汇总中，所存储的函数信息占了1.23G，处理起来比较费时，与本方法的初衷不符。因此在良性样本提取汇总并存储函数信息的过程中，除了进行去冗余操作外，还采取了无影响或影响较小的函数信息忽略，不操作，这样有效的节约了处理时间。

通过对smali文件的研究分析，可以发现在smali文件中有关API调用的函数有两大类。一类是在“.method ”和“.end method ”之间API的自定义函数，另一类是以“.invoke - ”开头的被调用函数。因此提取API信息只需遍历查找以两个关键字“.method ”和“.invoke-”开头的行，然后对每行进行处理，提取出单独的API信息。同时，还可以发现API在smali文件中的声明格式入下：

.method <访问权限> [修饰关键字] < API原型 >

其中API原型包括API的名称、参数类型与返回类型三部分。本实验只需提取函数原型信息，即找到包含两种特征信息的行，提取行信息，以空格键隔离行信息，将隔开的倒数第一个符号集数据写入特定的api(i).txt文档中。“.invoke-”开头的被调用函数提取API信息方式与“.method”所在行操作类似，再此不细说。提取API信息的程序流程图如图3.4。由于提取的API信息种类较多，本文只选取了两个样例进行介绍。以下分别是提取的良性软件与恶意软件中的一个API信息。

Landroid/accessibilityservice/AccessibilityServiceInfo;->getResolveInfo()Landroid/content/pm/ResolveInfo;（良性）

Landroid/bluetooth/BluetoothDevice;->createRfcommSocketToServiceRecord(Ljava/util/UUID;)Landroid/bluetooth/BluetoothSocket;（恶意）

可以看出这两个API信息都包含方法名，参数，返回值，smali文件对于方法原型的写法有其特有的规范，不过较之汇编语言还是可读性比较高，其更接近于Java语言，对于Android系统研究的学者来说，相对方便熟悉使用。

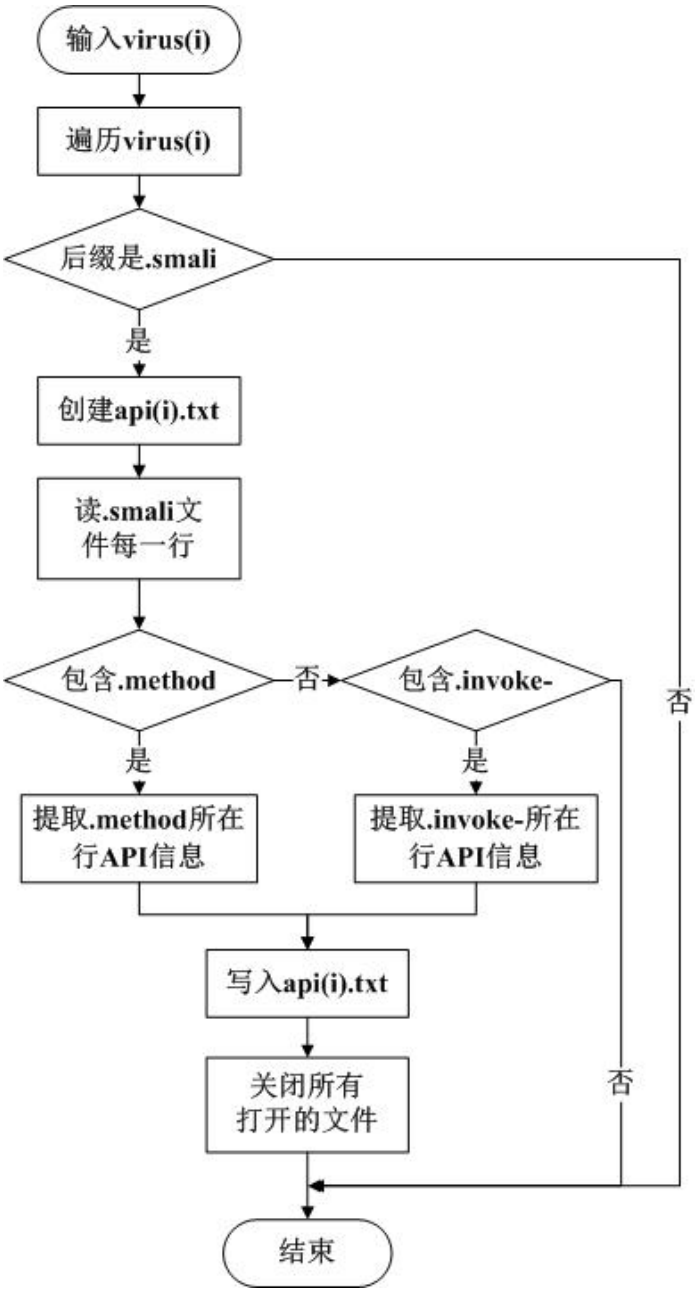


图3.4 API提取流程图

3.5 本章小结

本章首先介绍了该静态检测方法的流程，接着从 Android 应用的反编译开始介绍，讲述了 dex 文件反编译生成 smali 文件的过程以及可能出现的问题。然后详细介绍了权限特征的获取，API 特征的获取过程，为下一章特征信息的处理，静态检测方法的实现做出了准备。

4 Android 恶意软件静态特征处理

上一章主要获取特征数据，本部分重点是对所获得的特征数据进一步操作。根据提取的权限信息，统计权限频率。根据API信息统计API频率。由于频率数值间的差距明显，影响实验的准确性，本章对所统计出的频率进行线性处理。根据危害程度划分权限敏感强度，根据方法调用序列确定API敏感强度值。最后，本文采用FAHP方法，通过对四个影响因子进行设定不同的权值，统计对应模块与权限乘积的和的范围，从而确定阈值F。

4.1 权限信息处理

权限信息处理主要包括两方面，提取出权限频率值和设定敏感权限强度。在本节前作如下说明：

敏感权限表示会对用户或手机产生安全隐患的一些权限。安卓官网上公布了存在安全隐患的权限和权限组合，本实验添加了在恶意软件中申请的次数值大并且在良性软件中申请次数值小的权限，将这些权限作为敏感权限。如表4.1是敏感的权限和权限组合。

权限频率值不同于权限频率，是将权限频率进行处理得到的数据，作为影响因子之一。本节权限频率处理将详细的介绍权限频率值的设定过程，此处不赘述。

权限敏感强度表示权限对用户或手机的危害效果。其值的分布初始值为5，步长为5，分为4类，数值越大说明危害程度越高。

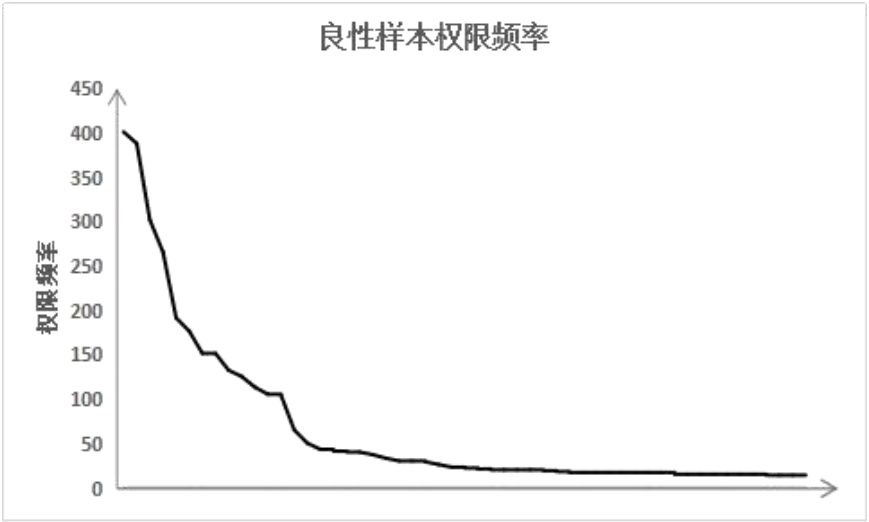
表4.1 危险权限组和权限

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

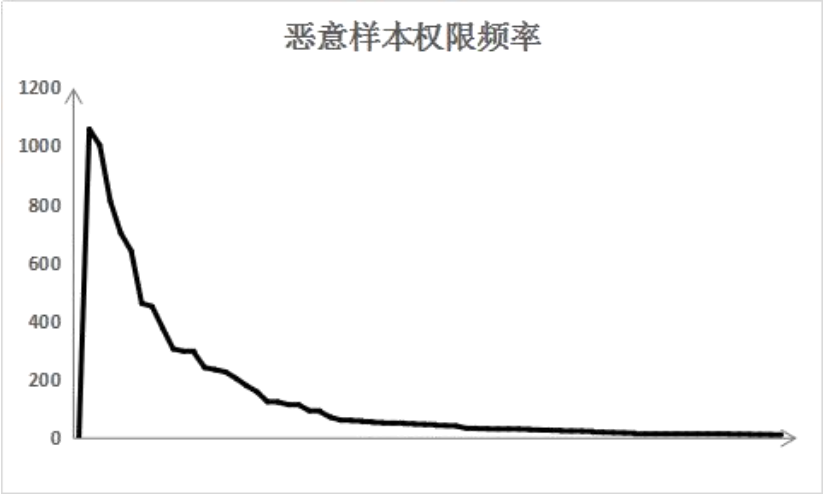
4.1.1 权限频率处理

为了分析两类样本中权限的分布情况，本小节主要是统计不同权限在两类样本中出现的频率情况。通过对比两类样本中权限的分布情况，研究不同样本权限的出现规律。分离出在恶意软件中申请次数值大并且在正常软件中申请次数值小的权限，作为敏感权限的一部分，为后续小节的特征信息处理做准备。同时，分离出两种样本中申请频率都较高的权限，若这些权限同时属于危险的权限组或者危险的权限，也将这些权限作为敏感权限。

本小节权限频率的处理首先是一个统计的过程，以恶意样本为例，需要先将去重后的每个权限信息文本进行整理，对整理后获得的文本进行去重复操作，遍历整理的权限信息依次与每个样本权限信息文本进行匹配，若匹配上计数器加一，最后按频率的大小进行排序，将递减排序的频率信息以“频率 权限信息”的格式存入权限频率文本。由于不同功能可能需要申请相同的权限，因此存在冗余的权限，需要对提取的权限信息进行去重。为了清晰的观察权限在不同样本的分布情况，本实验采取对恶意样本和正常样本同样的操作。分别将良性软件和恶意软件中的权限数据进行汇总，统计出良性软件和恶意软件中每个权限申请的次数。如图4.1的(a)，(b)分别是正常样本和恶意样本中统计的权限频率，纵轴均表示权限的频率，横轴均表示权限信息。由于不仅权限信息种类多，而且具体的每个权限的名称较长，因此横轴表示并为显示具体的权限名称，并且只显示部分权限的频率。



(a)



(b)

图4.1 部分良性样本和意性样本权限频率

通过对比良性软件中权限申请的频率和恶意软件中权限申请的频率，可以发现有的权限在恶意软件中申请次数多，在良性样本中出现的同样多，比如访问网络连接的权限 INTERNET 、获取网络状态信息的权限 ACCESS_NETWORK_STATE 、允许程序在手机屏幕关闭后后台进程仍然运行的权限 WAKE_LOCK 、获取无线网络信息的权限 ACCESS_WIFI_STATE 、允许震动的权限等 VIBRATE 。说明这些权限对恶意样本和良性样本的作用程度相似，不具有唯一区分软件恶意的作用。同时考虑到安卓官网公布的危险权限组和危险权限，本小节采用过滤这些危险权限，剩余的权限若在两类样本中出现频率都较高的，本实验默认这些权限对恶意安卓软件的影响较低。重点考虑在恶意软件中申请次数较多并在正常软件中申请次数较少的权限和危险的权限组和危险的权限。在本小节依据每个权限的频率对其进行处理和分析，这样全面的考虑每个权限对应用程序的影响可以有效的提高判断的准确性。

由于权限出现的频率最高值与最低值之间相差过大，为了既能表现不同权限频率间的差距，又能减少差距过大带来的不利影响，本文截取统计的权限频率汇总文本中权限频率大于10的所有权限，将这些权限的频率进行求和，求出每个权限对应的频率所占总和的比例的值，对该值进行扩大1000倍，得到的数据作为权限的频率值，将所有的权限及频率值以“频率值 权限信息”的形式存储到权限频率值文本中。权限对应的频率值越大，表示该数据影响程度越强。通过不同的频率值可以有效地表示权限对安卓软件的影响程度。以纵轴表示频率值，横轴表示权限名称，鉴于权限名称较长，因此横轴未显示具体的权限名称。同时由于权限种类比较多，因此统计的部分权限频率值的分布图，如图4.2所示。

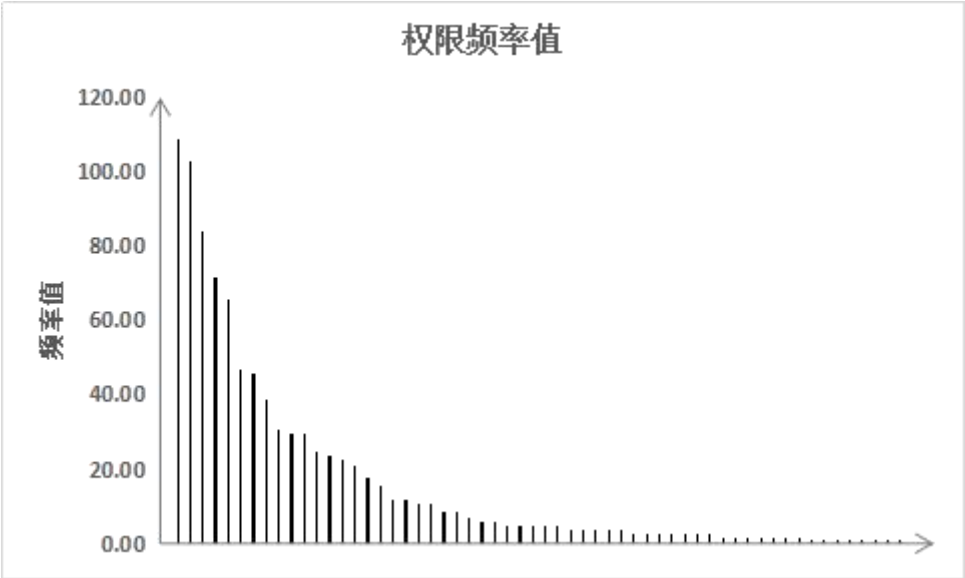


图4.2 权限频率值分布图

4.1.2 权限敏感强度

权限敏感强度是描述权限对手机或使用者的影响强烈的变量。不同的权限对Android应用的影响不同，有的权限仅仅打开手电筒、震动等对用户的影响可忽略；有的权限与费用有关，可能导致用户财产信息泄露；有的权限关系到手机所属人的隐私，滥用可能会导致用户相关数据暴露；有的权限与控制系统资源有关，为用户的危害性更多；相对而言，对用户影响较大的属于交互类权限，很可能给使用者带来一定的危害[64]。交互类权限主要负责本移动设备与其他移动设备的交互，是进行恶意操作的触发点，如果本移动设备不与其他设备进行交互，则无法收到威胁和损害，因此，交互类权限危险性最高。按权限对用户的危害程度可以将其列为四种，财产类、个人信息类、硬件操作类和用户交互类，其危害强度逐渐降低。本文将权限敏感强度分为四级，以5起步，步长为5，数值越大，说明权限敏感强度越强。

对于提取的权限信息，通过统计每个权限在总权限中出现的频率，并进行递减排序。考虑到不同权限的影响程度不同，将权限分为四类，其中用户交互类危害最大，硬件操作类次之，个人信息类，危害相对更低，财产类相对最低，则敏感权限强度值逐渐减少。

诸如权限CHANGE_NETWORK_STATE、CHANGE_WIFI_MULTICAST_STATE、CHANGE_WIFI_STATENSCT、SEND_RESPOND_VIA_MESSAGE等交互类对用户的影响很大，将其敏感强度设置为20。诸如REQUEST_COMPANION_RUN_IN_BACKGROUND、REQUEST_COMPANION_USE_DATA_IN_BACKGROUND、REQUEST_DELETE_PACKAGES等硬件操作类权限，其对用户的影响降低，将其敏感强度设置为

15。诸如权限 READ_FRAME_BUFFER 、 READ_INPUT_STATE 、 READ_LOGS 、 READ_PHONE_NUMBERS 等与用户隐私相关的权限，其敏感强度设置为10。诸如权限 WRITE_EXTERNAL_STORAGE 、SYSTEM_ALERT_WINDOW 等对用户影响较小，其敏感强度设置为5。纵轴表示权限的敏感强度值，横轴表示权限信息，截取部分权限敏感强度表，整理的权限敏感强度柱状图，同样横轴并为显示具体统计的部分权限的名称，如图4.3。

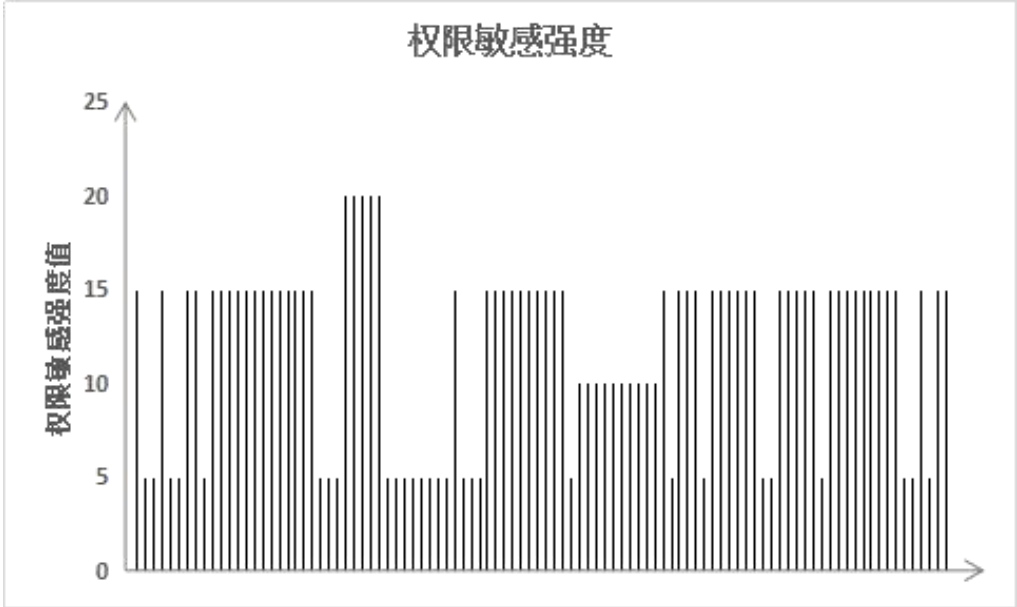


图4.3 部分权限敏感强度值图

4.2 API 信息处理

API信息处理主要包括API频率的统计、API频率值的设定和API敏感强度值的设定。其中，API敏感强度依据API调用序列设定，有效地考虑了软件行为对应用程序的危害程度。在API信息处理前做以下说明：

敏感API表示会对使用者或Android设备造成安全隐患的一些函数，以“方法名称 参数 返回值”的形式存储和处理。在本实验中首先找到上一节确定的敏感权限，根据权限和方法的关联情况，确定敏感权限对应的函数将其作为敏感 API 。

API频率值不同于API频率，是在统计得到各API频率的基础上，为了忽略API频率间的差距，减少差距过大带来的实验判断结果不准确性而采用的对频率进行线性处理得到的数据。本实验将API频率值作为影响因子考虑。

API敏感强度表示API对用户或手机产生的危害程度，在本实验中以敏感API作为原点，溯源性的遍历查找该API的调用序列，通过调用情况设定API敏感强度值。同样，其值的分布初始值为5，步长为5，分为4类，数值越大说明危害程度越高。

4.2.1 API 频率

在上一章已经提取出样本的函数信息，为了分析不同样本中API的分布情况，本节主要是对这些函数信息进行处理，统计出其出现的频率。对于API信息，遍历每个样本中的方法，将其汇总到一个文件中，并对总样本中的API信息进行去冗余操作。整理出汇总文本中不同API在总样本中使用的频率，并按递降方式进行排序。统计出恶意软件中API被使用的次数。如图4.4所示。

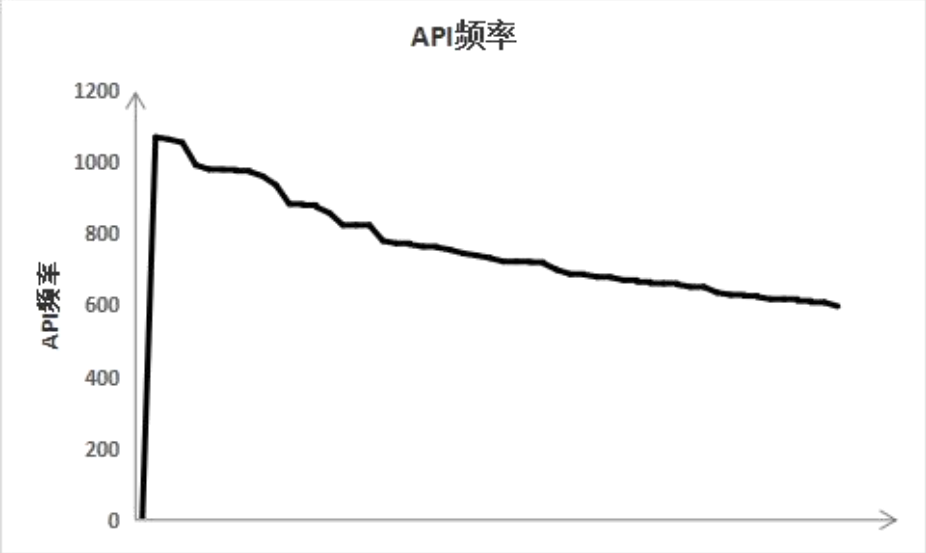


图4.4 部分恶意样本API频率

观察恶意样本中方法出现的频率，可以发现初始化方法<init>出现的频率特别高，换言之，就是方法的初始化无法真正对系统或用户产生不利的影响，因此，即便其频率很高，也不对其做处理。大多方法基本都调用了<init>函数。为了高效地确定Android应用程序的危害性，测试方法对该函数进行了过滤处理。同时，通过对比恶意软件中函数使用次数和正常软件中使用次数，可以发现一些方法，在两类样本中的使用次数较多，将其做为无效数据，过滤处理。

有的函数在总样本中出现的频率有几百次，而有的只有几次，为了弱化频率间的差距又保持其相对差距，本节对这些频率进行处理。首先选取API频率文本中出现频率大于99的方法，然后求出这些方法对应的频率之和，每个频率占频率和的比值扩大10000倍就是这个方法对应的频率值。将这些函数信息以其频率值进行增减排序，以

<频率值><API 名称><API 参数><返回类型>

的格式存储API频率值信息，存入API频率值文本，为后续操作做准备。可以发现频率值均保持在两位数，之间的差距减少，并且存在区分度，排序后的函数频率值如图4.5所示，横轴表示API信息，纵轴表示API的频率值，鉴于API种类过多并且名称过长，横坐标未

显示API信息，只显示了部分API的频率值。

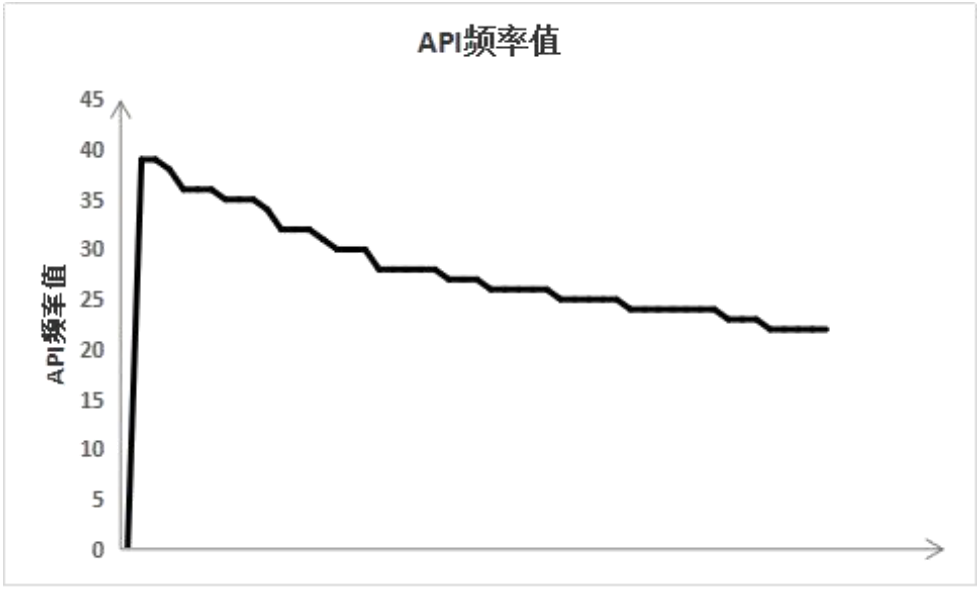


图4.5 部分恶意样本API频率值

4.2.2 API 敏感强度

API敏感强度用来表示API方法对手机或者使用者的影响强烈。众所周知，不同API对应用程序的影响不同。方法对应用程序的影响除了与方法自身实现的功能有关，与方法的调用序列也有关系。API敏感强度作为影响因子之一，其完全取决于软件的行为，是将动态特征以静态的方式进行提取处理所得到的结果。本方法之所以正确率高很大一部分原因便在于结合了动态分析的优点。去除正常软件和恶意软件中申请次数同样较多的权限。确定出现次数较多的权限及危险的权限组和危险的权限，将其作为敏感权限。而敏感权限和API间含有对应关系，即要申请某一权限也需要调用某些函数。根据敏感权限可以确定一些敏感API。附录二显示了在本实验样本中API与权限的映射关系。在确定敏感权限和API的对应关系时，因为不同版本的权限和API映射存在差异，而所收集的样本种类过大，存在使用不同版本的可能性，因此，根据敏感权限确定的API和恶意样本提取的函数名称对应不上。在本文，通过匹配敏感权限所对应的类名称关键字，查询恶意样本方法汇总文件，从而确定映射的API。

敏感API已经确定，下一步便是确定这些敏感API的调用序列。API调用序列也对判断恶意软件有影响，因此，获取 API 调用序列至关重要。本文初步的设想是将敏感权限映射的方法作为源点，通过遍历获取样本中每个.smali文件。然后判断smali文件中是否包含敏感API，找到其，查询其调用的函数一层层查找。最后确定方法调用序列，提取出API调用序列的过程，具体如图4.6所示。

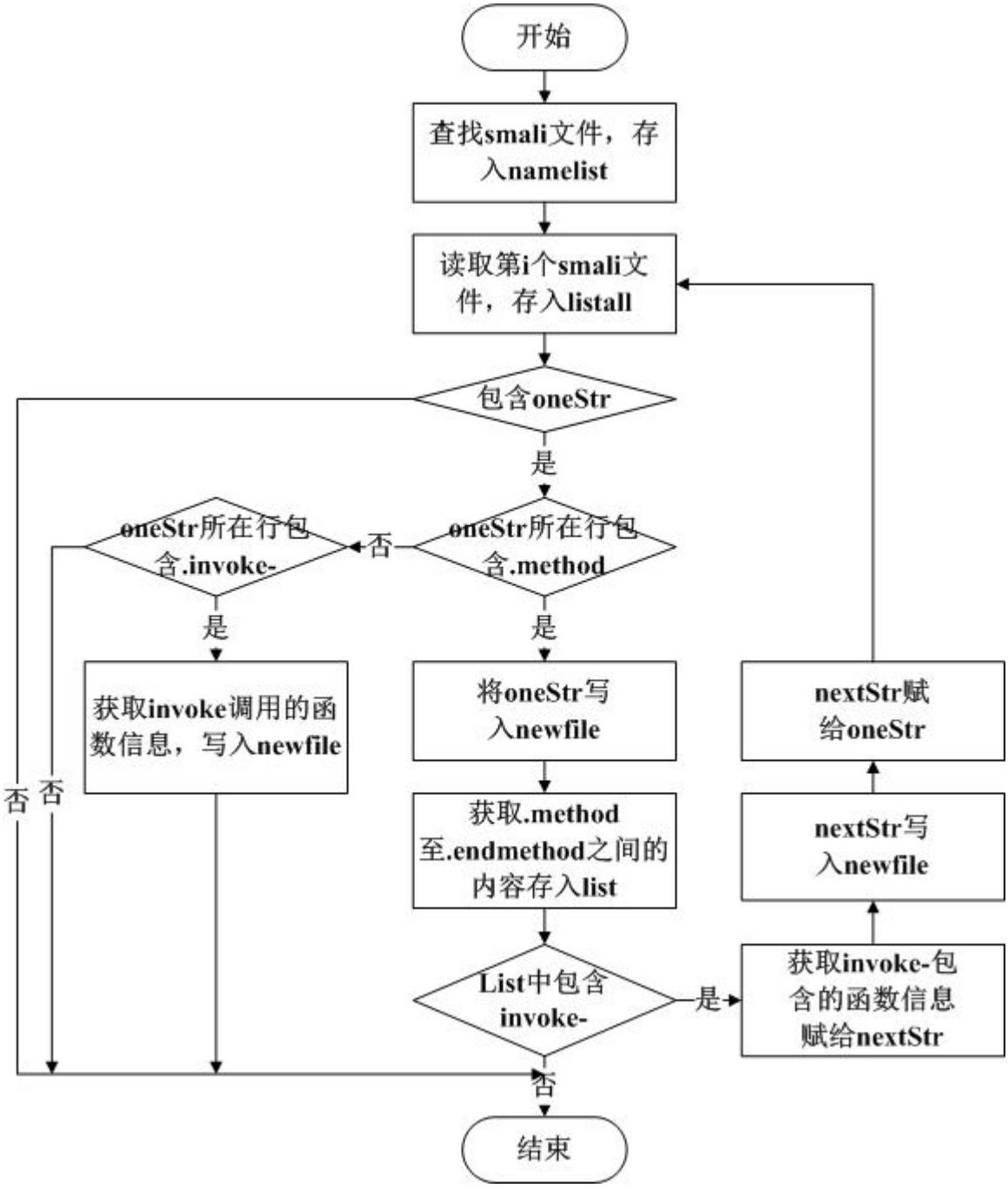


图4.6 API调用序列方法

通过实验发现所有的敏感权限映射的函数都是被调用函数，并且不再调用其他函数，这说明这些敏感API是最靠近敏感行为的接触点函数，是最危险的API。因此本实验采取溯源的方式查找以映射函数为终点的函数方法调用序列，根据调用序列中的函数到终点函数的步长，设置其函数敏感强度。通过遍历获取样本中每个.smali文件，一层一层确定初始的敏感函数都被哪个函数调用。将调用API作为上级API，继续查找上级API

的调用API，这样回溯地获取方法调用序列，提取API调用序列的具体过程如图4.7所示。

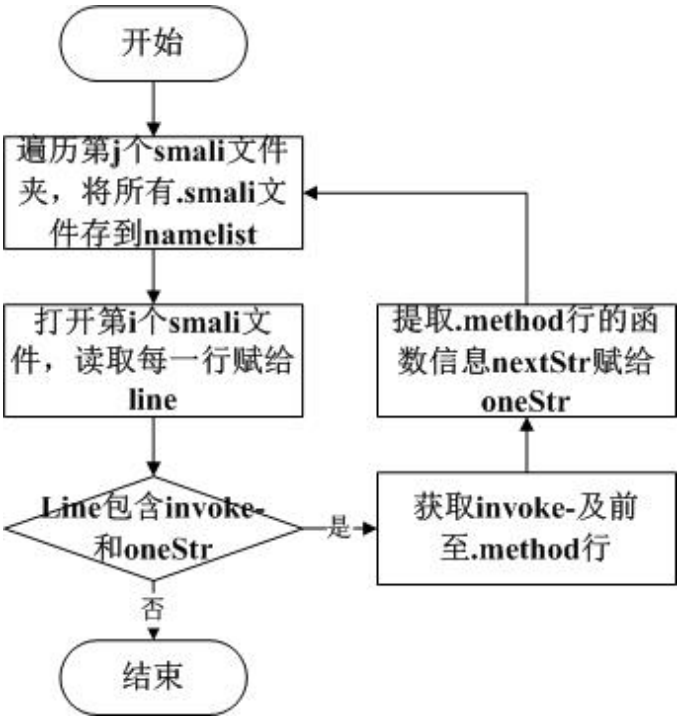


图4.7 溯源API调用序列方法

在上面，获取到了方法调用序列，接下来是对这些信息进行处理。众所周知，多诺米效应是因为触动了某块骨牌导致该骨牌及其后面的骨牌全部倒下。应用到时间轴上，因为昨天过去了，所以今天来临，因为今天流逝，明天必然降临。这告诉我们，当下的作为影响到即将到来的明天，条件导致结果。如果拿掉多诺米骨牌的部分骨牌，那么，确实部分前面不管如何倒下，也无法影响到缺失部分后面的骨牌。这告诉我们，直接条件才是导致直接结果的决定因素，间接条件如果无法形成完整的条件链对直接结果也是影响甚微。应用到恶意软件领域，要想形成一定的破坏效果，必须具备有效的行为链或者必须具备形成这一恶性结果的直接条件。如果调用一个方法直接导致系统或用户收到损害，那么这个方法的API敏感强度最强，所属等级最高，而如果一个方法想要实现一个恶意的行为需要调用别的方法，那么可以认为其API敏感强度较弱。类似于，将API作为节点，将API调用关系作为连线的调用图中，将敏感API作原点，画出一组组同心圆，离圆心距离越小，其危害程度越强。

使用方法调用序列模拟安卓软件的行为，根据行为的危害程度设置API敏感强度值。在上面得到的方法调用序列中，依据调用源点API的函数序列，将函数划分为不同的等级。离敏感API距离越小说明危险程度越强，因此确定的敏感强度值越大。以初始值为5，步长为5，将敏感API及调用序列中的API划分为四类，危险程度和敏感强度值成正比例。统计整理得到API敏感强度如图4.8所示。

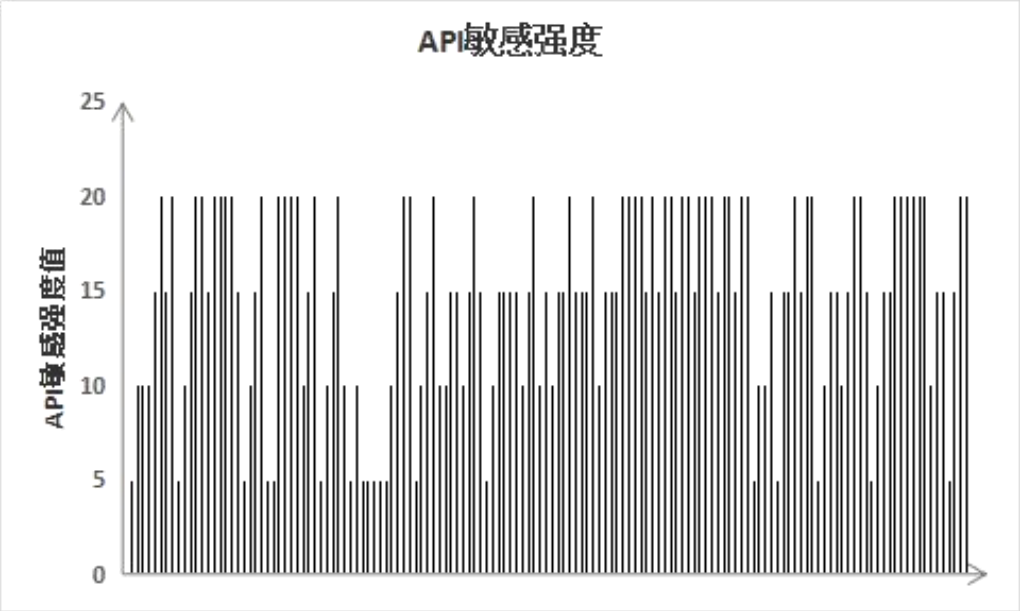


图4.8 部分API敏感强度

4.3 阈值设定

对于实验的每一个样本，提取出其权限，将权限信息和权限频率文本进行比对，获取匹配上的权限对应的频率值，求其均值记为A_i。将提取的权限与权限敏感强度文件进行匹配，匹配上便提取对应的等级值，求所有匹配上权限的均值记为B_i。提取出样本中出现的函数信息，出现次数大于等于一，均记为一次，将函数信息与函数频率文件进行匹配，求取匹配上的函数对应频率值的均值记为C_i。将提取的函数信息与敏感API强度文件进行匹配，同样，对匹配上的函数对应的敏感强度值求平均数，记为D_i。

本文处理后的特征信息由四部分组成，分别是权限频率值、权限敏感强度、API 频率值和 API 敏感强度。据此数学模型如下：

$$F = A_{\text{权限频率}} + B_{\text{权限敏感强度}} + C_{\text{API 频率}} + D_{\text{API 敏感强度}} \tag{4.1}$$

$$F_i = \omega_1 A_i + \omega_2 B_i + \omega_3 C_i + \omega_4 D_i \tag{4.2}$$

F_i是某个Android软件的敏感值，F是实验中一切恶意软件和良性软件的分界点F_i值。ω_1，ω_2，ω_3，ω_4分别是四个影响因子的权值。本文采用FAHP确定四个影响因子的作用大小。将四个影响因子的影响程度用数据的形式表示。根据统计恶意样本和良性样本四个影响因子的加权平均数Fi的范围确定阈值。

根据FAHP来求出ω_1，ω_2，ω_3，ω_4。其求解过程如下所示。

第一步　创建模糊矩阵。又称为优先判断矩阵，该矩阵的确定是通过两元素相互比较，根据表4.2的九标刻度法表示的数据确定的。得到$A=(a_{ij})_{4\times4}$，其中，$a_{ij}+a_{ij}=1$，$0\le a_{ij}\le 1(i;j=1,2,3,4)$。若对于该矩阵A满足：$\forall$i、j、k ，有$a_{ij}=a_{ik}-a_{jk}+0.5$，就说明

矩阵A为模糊一致判断矩阵。可采用四个影响因子权限频率值、权限敏感强度、API频率值、API敏感强度中，API敏感强度影响最大，权限敏感强度次之，函数频率值再次，权限频率值影响相对最低。本文采用调查问卷的形式，统计了导师，同学，师弟师妹们，最终取所有调查结果的均值作为模糊矩阵的值，对角线a_{ii}表示与自身进行比较，任何元素与自身进行比较，重要程度都相同，即都为0.5，因此确定的模糊矩阵A如下：

$$A=\begin{bmatrix}0.5 & 0.3 & 0.4 & 0.2\\ 0.7 & 0.5 & 0.6 & 0.3\\ 0.6 & 0.4 & 0.5 & 0.4\\ 0.8 & 0.7 & 0.6 & 0.5\end{bmatrix}$$

第二步　设置模糊一致判决矩阵。由于两者比较，前者比后者强多少，后者便比前者弱多少，即比较双方存在相互间的依赖关系。记$a_i=\sum_{}^{n}a_{ik}$，$i=1,2,3,4$,利用一致性，把$a_{ij}=a_{ik}-a_{jk}+0.5$公式转换为$a_{ij}=(a_i-a_j)/2n+0.5$，那么就可以把第一步设置的矩阵转化成模糊一致性判断矩阵。

第三步　根据矩阵A求各指标的影响值。设元素$A_{权限频率值}$、$B_{权限敏感强度}$、$C_{API频率值}$、$D_{API敏感强度}$的影响值分别为ω_1、ω_2、ω_3、ω_4，i=1,2,3,4，那么可得：

$$\omega_i=\frac{\sum_{j=1}^{n}a_{ij}+\frac{n}{2}-1}{n(n-1)} \tag{4.3}$$

第四步　根据公式4.3计算四个权重值。计算得到ω_1=0.2、ω_2=0.2583、ω_3=0.2417、ω_4=0.3。

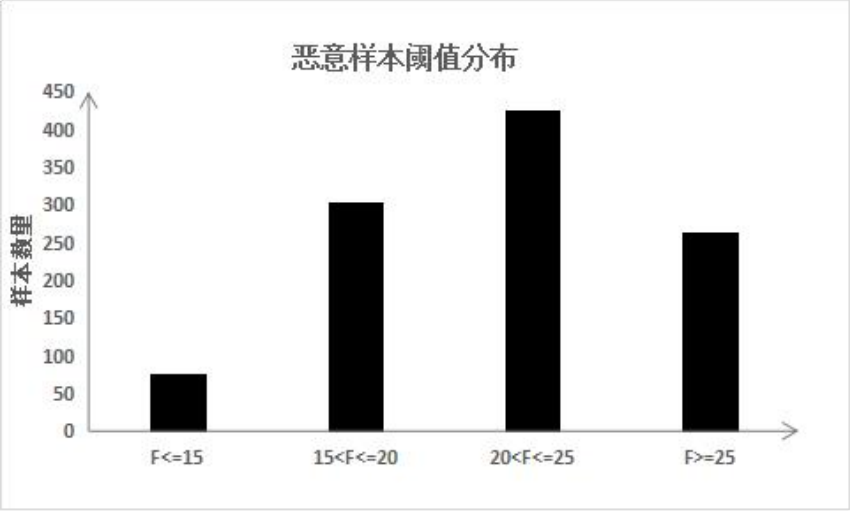
表4.2 数据标度介绍

标度	定义	说明
0.5	同等重要	两元素相比较，同等重要
0.6	稍微重要	两元素相比较，一个元素比另一个元素稍微重要
0.7	明显重要	两元素相比较，一个元素比另一个元素明显重要
0.8	重要得多	两元素相比较，一个元素比另一个元素重要得多
0.9	极端重要	两元素相比较，一个元素比另一个元素极端重要
0.1,0.2,0.3,0.4	反比较	若元素 a_i 与元素 a_j 相比较得到判断 a_{ij},则元素 a_j 与元素 a_i 相比较得到的判断为 $a_{ji}=1-a_{ij}$

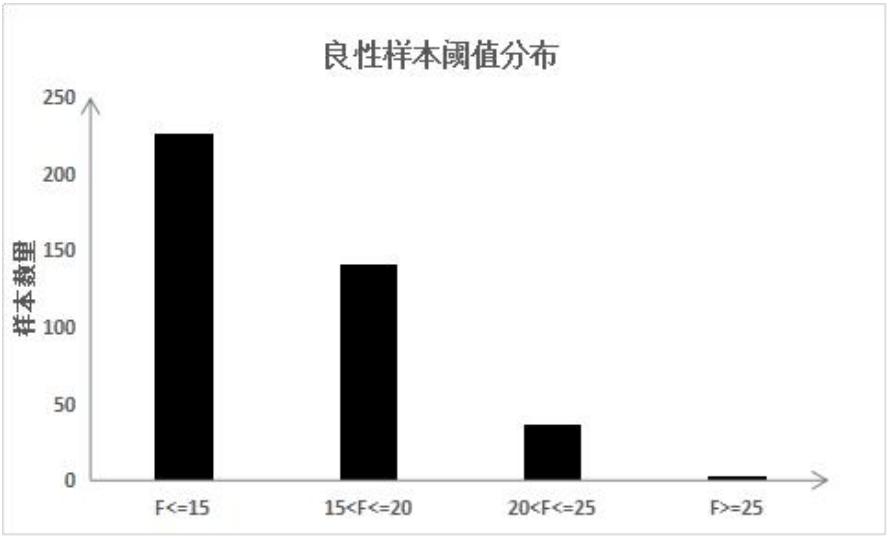
阈值F的确定主要通过大量的统计获得。比如，对于恶意样本一，由于已经统计过

其权限和API信息，因此，首先遍历每个权限与权限频率文件进行比对，如果匹配上，则获取对应的数值，将这些数值求和并求平均，得到的数值记为A_1。然后，将恶意样本一的每个权限和权限敏感强度文件比对，匹配上，则取对应的数值，将这些数值进行求和并求平均，平均数记为B_1。再然后，遍历恶意样本一的每个API，与API频率文件进行比对，匹配上就取得相应的数值，将这些数值求平均数，记为C_1。接着，将API信息与API敏感强度文件进行匹配，若匹配上，同样取平均值，记为D_1。最后，根据模糊层次方法确定的权值，进行加权，得到的结果记为F_1，统计所有恶意样本中的F_i，观察其值的分布范围。对于良性样本，进行同样的操作，根据良性样本和恶性样本F_i的分布范围，找到能明显区分恶意和良性的分界点，定位阈值F。

通过对良性样本和恶性样本求其各影响因子的加权平均数F_i，分别统计恶意样本和良性样本的加权和数值。对比两类样本的F_i值大小，可以发现恶意样本中的F_i值基本都大于17，1072个恶意样本只有几个是小于17的，而对于良性样本，基本都小于17，大于17的也只是极少数。具体的分布情况本文采用分别统计F_i值小于15，在15至20之间，在20至25之间和大于25这几种情况的样本数目。通过统计恶意样本中阈值的分布情况和良性样本中阈值分布情况，可以发现对于恶意样本阈值小于15占得比例很小，而正常软件阈值范围小于15的样本在总样本中数量较多。这说明所选择的特征信息可以匹配上大量恶意样本，而与良性样本的匹配度相对较低。由于阈值范围在15至20之间也存在很多样本，因此可推测阈值应设置在15到20之间的某个数。图4.9的(a)是恶意样本中阈值分布范围整理情况，(b)是良性样本中阈值范围分布整理情况，竖直坐标显示样本的数量，水平坐标显示阈值的范围。



(a)



(b)

图4.9 恶意软件和正常软件阈值分布

为了清晰的观察阈值的分布比两类样本分类情况的影响，分别统计恶意样本阈值大于等于15、大于等于16、大于等于17、大于等于18、大于等于19的恶意样本数目，并计算占恶意样本总数的比例。对于良性样本，分别统计阈值不大于15、不大于16、不大于17、不大于18和不大于19的良性样本数目，并计算分别占所有正常软件的比例，如表4.3所示。

表4.3 阈值比较情况表

	恶意样本数量	占总样本比例	良性样本数量	占总样本比例
15	995	92.8%	182	44.5%
16	920	85.8%	234	57.2%
17	885	82.6%	357	87.3%
18	766	71.5%	362	88.5%
19	730	68.1%	366	89.5%

通过上表，可以看出随着阈值的变大，大于阈值的恶意样本占总样本的比率逐渐减少，这说明恶意样本的待测阈值相对较大，恶意软件和提取的四个影响因子的匹配程度更贴切。而小于阈值的良性样本占总样本的比率逐渐增大，这说明良性样本的待测阈值相对较小，良性样本和提取出的四个影响因子匹配程度低。若选取较低的待测阈值作为阈值，存在不能准确的匹配良性样本的问题；若选取较大的待测阈值作为阈值，存在不能准确的区分恶意样本的问题，因此阈值的选取分布居中。上表中可以看到随着阈值的变化，恶意软件占所有软件比例和良性软件占所有软件的比率此消彼长，仅当阈值分布在17左右时，两个比率均较高。根据上表统计出不同阈值下，两类样本的统计情况，如

图4.10。

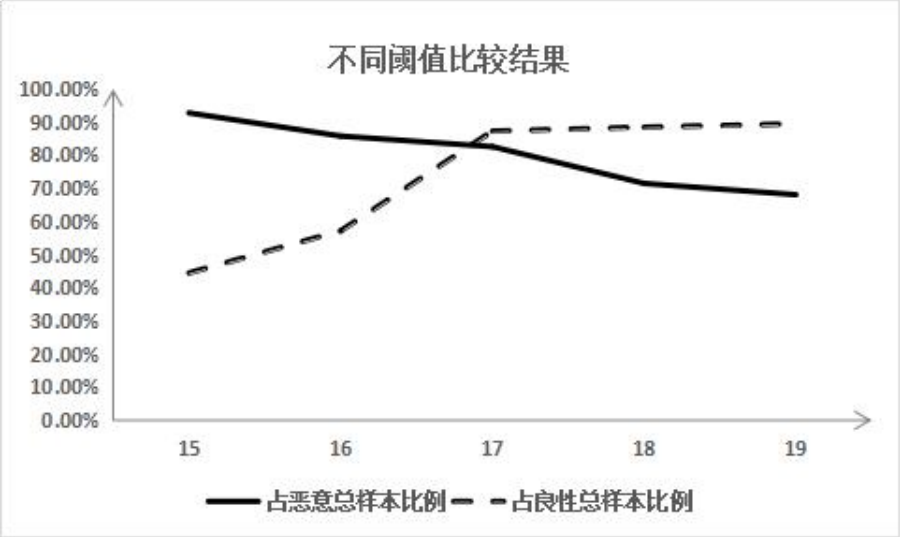
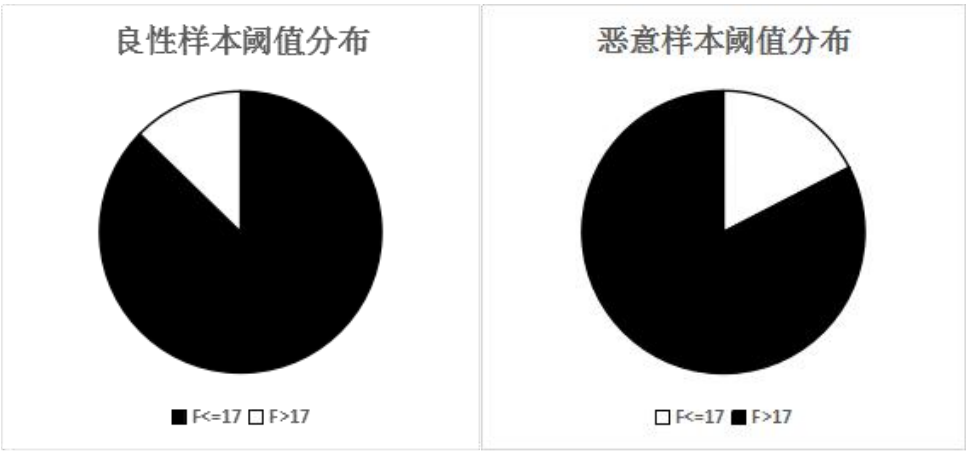


图4.10 不同阈值统计结果折线图

通过观察可以发现，当阈值为17时，不管是正常样本还是恶意样本，统计结果都高达百分之八十以上，因此，17作为阈值最合适。如图4.11的(a)是良性样本中阈值以17为分界点的饼图，(b)是恶意样本中阈值以17为分界点的饼图。



(a) (b)

图4.11 良性样本与恶意样本中阈值以17为分界点的饼图

4.4 实验结果分析

恶意软件的阈值小于等于17的数目仅为187个，大于17的数目为885个，占所有恶意软件的82.6%。良性软件的阈值大于17的数目为52个，小于等于17的数目为357个，占良性样本总数的87.3%，因此，将17作为阈值F。本实验从统计得出阈值的分布范围，从而确定阈值设置在15--20之间比较合适，接着又统计阈值分别为15、16、17、18和19时样

本的数量，得到阈值为17。接下来主要是通过测试样本验证阈值是否合理，测试该静态检测方法是否有效。

4.5 本章小结

本章首先介绍了对权限信息和API信息的处理。通过统计获得权限频率和API频率，对频率进行先行处理得到权限频率值和 API 频率值。接着依据权限对 Android 软件的影响程度，将权限划分为四类，并设置了不同的权限敏感强度值。通过权限与 API 的映射关系，确定了 API 调用序列的源点。通过源点遍历统计 API 的调用序列，根据 API 调用情况设置了 API 敏感强度值。最后通过模糊层次分析法给不同影响因素设置权值，根据恶意软件和正常软件的分布范围，从而确定阈值 F。

5 方法测试与评估

目前动态调试通过对安卓软件的行为进行测试研究，虽然可以检测出 0day 漏洞，但比花费时间较多，同时比较复杂，不适合实际使用。而静态分析大都选择静态信息作为特征，用以区分软件的恶意和良性。虽然操作简单，但不利于检测尚未记录的恶意软件，也就是在已出现的恶意应用程序中正确率高，而无法识别新型的恶意应用程序。本文虽然采用静态检测方法，但提取出函数的调用序列作为软件行为模型，用以模拟预测安卓应用程序的行为，从而更准确并且更简洁地进行恶意应用程序分析。由第四章的研究实验为该静态分析方法确定了阈值。为了测试方法是否可行，阈值的设定是否合理，本章将分别取 500 个恶意样本和 300 个良性样本进行测试评估。通过对测试样本中恶意样本和良性样本进行特征提取、特征处理、阈值对比，从而进行判断。根据测试结果评估该静态检测方法的准确性、误判率和分类精度等。

5.1 测试方法概述

本章主要负责检测上部分得到的静态检测方法是否有效。测试方式的大概思路是：首先分别获取大量的良性软件和恶意软件作为实验测试样本，分别对两大类软件采取反编译操作，然后提取权限信息，API 信息，并对这些信息进行去冗余处理。接着分别匹配权限频率值文本、权限敏感强度文本、API 频率值文本和 API 敏感强度文本，若匹配上取影响因子对应的数值，求出各影响因子的平均值。最后，使用 FAHP 方法确定四个影响因子的权值。计算加权和作为待测阈值与上一章确定的阈值相互对比。检测方法如图 5.1 所示。

该测试方法的步骤如下所述：

(1) 样本反编译：对选取的良性测试样本和恶意测试样本进行批量反编译处理，生成相应的 smali 文件，为特征信息的提取做准备。

(2) 特征信息提取：主要提取测试样本中的权限信息和 API 信息，权限信息和 API 信息的提取和第三章的方式相同，在此不赘述。

(3) 求取待测阈值：采取将获得的权限和权限频率值文本进行匹配，若匹配上，取权限信息对应的频率值，然后求所有匹配上的权限信息频率值的平均数，记为 A_i。同样，将权限信息与敏感强度值文本进行匹配，求匹配上的权限信息的敏感强度值的平均数，记为 B_i。API 信息需先经过去重操作再进行匹配，匹配上的 API 频率值的平均数记为 C_i，匹配上的 API 敏感强度值平均数记为 D_i。使用上一章计算出的ω_1、ω_2、ω_3、ω_4 与 A_i、B_i、C_i、D_i 进行加权求和运算，得到待测阈值 F_i。

(4) 阈值比较：根据待测阈值与上一章确定的阈值相互对比，若比阈值小则判断为良性样本，输出 TRUE ；若比阈值大则判断为恶意样本，输出 FALSE 。

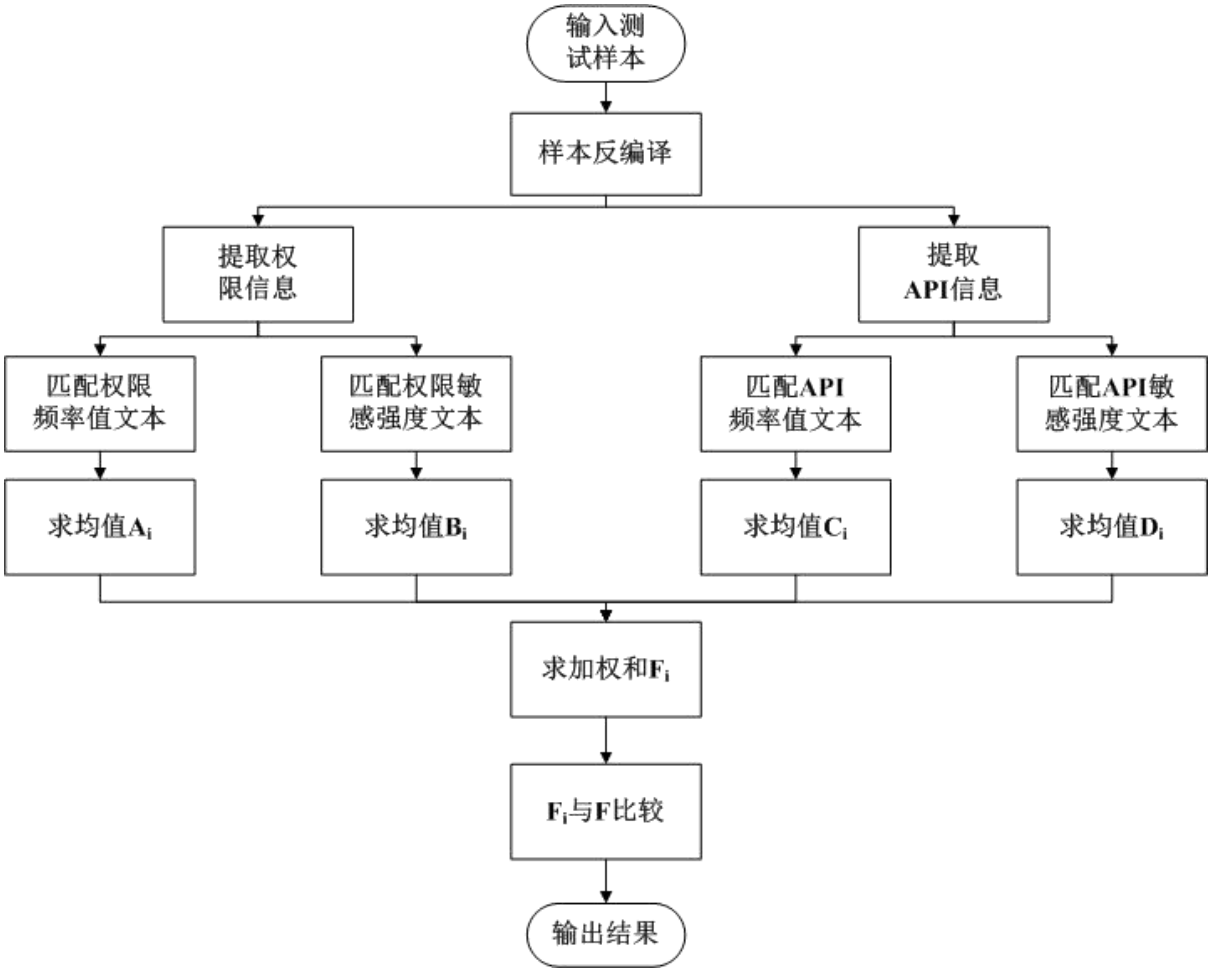


图 5.1 测试方法框图

5.2 样本测试与评估

5.2.1 实验环境

本章所用的的试验开发环境如下：

(1) 硬件配置： PC 机（(Intel® Core ™ i5 处理机，内存大小为 4G ，主频为 2.5GHz ，硬盘容量为 600G ）。

(2) 软件配置：Windows7 系统。

(3) 软件开发环境：Java 环境、myeclipse10 软件、notepad ++ 软件、电脑管家 12.11、360 安全卫士 11。

(4) 辅助工具： apktool 2.0 。

5.2.2 测试样本选取

本文实验主要从 https://virusshare.com 网站上下载了某年的 500 个安卓恶意应用程序组成恶意软件测试样本库。从 Google 应用商城下载了 300 个良性应用组成良性应用程序测试样本库。恶意样本是随机选取了某年的恶意安卓应用程序。随机选取的良性样本包括办公类、娱乐类、学习类、讲课类、运动类、理财等多种应用程序。并且上一章使用的样本与本章使用的样本不同，上一章的样本主要用来进行实验，得出阈值。本章使用的样本主要用来测试该静态检测方法是否合理，是否能有效检测安卓应用程序的恶意性和良性。本文主要目的是提取分析一部分恶意样本的特征，找到其规律，然后根据这些规律对另外一部分恶意样本进行判断测试。

5.2.3 样本测试与评估

本文对安卓软件的恶意性判断主要是进行特征信息匹配，根据匹配程度作出判断，因此对于测试样本，第一步通过批处理脚本对其进行反编译，得到相应的 smali 文件。然后分别获得 AndroidManifest.xml 中的权限数据，并进行去重复处理。查询每个样本的所有 smali 文件，提取出所有的函数信息。接着对这些函数信息进行去冗余处理。良性测试样本编号为 beni(j).apk，其中$1 \le j \le 300$，反编译后生成的文件编号为 beni(j)，其中$1 \le j \le 300$。恶意测试样本的编号为 bad(k).apk，其中$1 \le k \le 500$，反编译后生成的文件编号为 bad(k)，其中$1 \le k \le 500$。

已经提取出样本的权限数据和权限频率数据，接着对特征数据进行匹配。分别将每个样本的权限信息与权限频率值文件和权限等级值进行匹配，若匹配上取其对应的数值，对所有匹配上的的信息对应的数值，求其平均数记为 A_i、B_i。需要注意的是若一个样本中提取的权限信息不止一个，则只匹配一次。分别将每个样本的函数信息与函数频率值文件和函数敏感强度文件进行匹配，同样，对匹配上的信息对应的数值求出其平均数记为 C_i、D_i。同样 api 信息出现的频率大于一，只匹配一次。根据上一章确定的权值 ω_1、ω_2、ω_3、ω_4，求加权平均数 F_i。根据 F_i 与阈值 F 相互对比情况，如果不大于 F，便输出 TRUE，从而判断软件是良性应用程序的；如果大于等于 F，便输出 FALSE，从而判断软件是恶意程序。

通过选择 5 个基本指标度量 Android 恶意应用分析方案的有效性。表示对正常样本进行分类的真阳性(True Positive，简称 TP)和真阴性(True Negative，简称 TN)。表示对恶意样本进行分类的假阳性(False Positive，简称 FP)和假阴性(False Negative，简称 FN)。分类精度，表示无误地将软件进行归类的软件数目占所有软件数目的比率。具体的

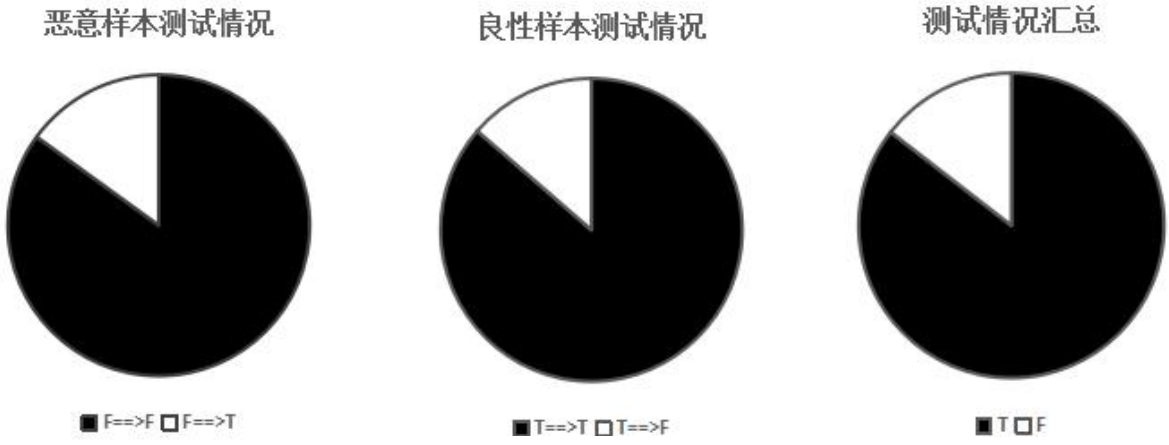
Android 恶意 软件 检测 方法度量指标的介绍如表 5.1。

表 5.1 评判指标说明表

指标名称	说明
检测率(detection rate)	真阳性率 TPR=TP/(TP+FN)，召回率(recall rate)，表示所有良性样本中正确分类为良性软件的比例
误报率(false alarm rate)	假阳性率 FPR=FP/(FP+TN)，表示所有恶意样本中错误分类为正常软件的比例
真阴性率(true negative rate)	TNR=TN/(TN+FP)，表示所有正常样本中错误分类为恶意软件的比例
分类精度(accuracy,简称 ACC)	是所有正确分类的样本与所有样本之比，即：ACC=(TP+FN)/(TP+TN+FP+FN)
正确率(precision)	是指真正的恶意软件在分类为恶意软件中的比例，即：正确率=TP/(TP+FP)。

符号“==>”表示判断为，T 表示良性样本，F 表示恶意样本。表达式 T|F ==> T|F 表示在样本为良性样本或恶意样本时，将其判断为良性样本或恶意样本。如图 5.2 的(a)表示测试样例为恶意样本的前提下，将其判断为恶意样本，即 F ==> F 的扇形图和将其判断为良性样本，即 F ==> T 的扇形图。

图 5.2 的(b)表示测试样例为良性样本的前提下，将其判断为恶意样本，即 T ==> F 的扇形图和将其判断为良性样本，即 T ==> T 的扇形图。在所有正常样本和恶意样本的测试样例中，正确的进行判断，即 T ==> T||F ==> F，将其记为 T；错误的进行判断，即 T ==> F||F ==> T，将它标为 F，T 和 F 占总测试软件的比例如图 5.2 的(c)所示。



(a) (b) (c)

图 5.2 样本判断结果

分别统计本文方法和与本文方法类似的参考文献[9]、[14]和[63]的各种判断比率，其中参考文献[63]选取的是10个特征信息的统计结果，可得到如表 5.2 的数据：

表 5.2 测试结果统计表

	检测率	真阴性率	正确率	误报度	分类精度
本文方法	86.3%	13.7%	85%	15%	85.5%
参考文献[9]	94.1%	5.9%	94.4%	5.6%	94.2%
参考文献[14]	81.05%	18.95%	88.05%	11.95%	84.52%
参考文献[63]	83.8%	16.2%	84.4%	15.6%	84.1%

将判断情况进行整理可得到四种方式判断比率的分布柱状图，如图 5.3 所示：

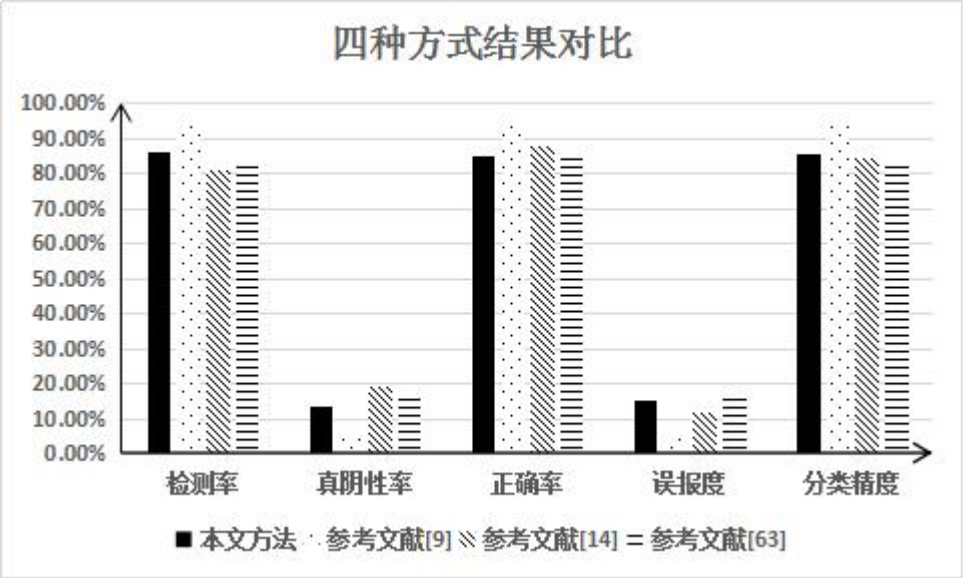


图5.3 四种方法结果柱状图

5.3 测试结果分析

本章主要是对上一章提出的静态检测方法的测试，验证阈值是否合理。起初将阈值设定为 18，发现误报率和漏报率都较高，于是将阈值设定为 17，进行方法验证。在阈值为 17 的前提下，通过统计 500 个恶意样本和 300 个良性样本，可以发现，误报率和漏报率都降低了很多。其中 500 个恶意样本中可以成功的检测出是恶意样本的有 425 个，300 个良性样本中成功检测出属于正常样本的有 259 个。测试结果表明，本文所提出的静态检测方法相对较准确的检测样本是否是恶意软件，不过准确率还有待提高。

参考文献[9]提出的方法判断结果优于本文提出的方法，但其不仅需要动态分析浪费时间，而且使用了机器学习，使得复杂度上升，不便于用户使用。对比参考文献[14]提出的方法，本文提出的方法在检测正常样本和分类精度两方面均优于参考文献[14]提出的方法，由于参考文献[14]提出的方法属于动态方法，虽然检测结果与本文方法不相上下，但在特征信息提取处理方面比本文的方法稍微复杂。参考文献[63]提出的方法判断

结果次于本文提出的方法，同时其选取了 10 个特征信息，比本文的方法复杂，不利于实际应用。而本文提出的静态分析方法对良性软件的分析正确率达到 86.3%，对恶意样本分析的正确率达到 85%，检测率相对较高，又单纯的属于静态方法，比其他动态方法更快速便捷。

5.4 本章小结

本章重点是测试该静态分析方法是否能高效的分出恶意应用。先获取大量恶意样本和正常样本，对两类样本采取反编译操作，获取其相应的 smali 文件。然后提取出权限信息和 API 信息，并对这些信息进行去重复操作。接着根据上一张得到的权限频率值文件、权限等级文件、API 频率值文件、API 敏感强度文件，对样本特征信息进行处理，根据上一章的特征处理方法进行处理，从而判断安卓软件的恶意性。最后根据判断的有效性，从五个方面进行评估。

6 总结与展望

6.1 本文工作总结

由于计算机技术和通讯工具的不断进步，手机在人们的每一天都充分展示其不可或缺的身份。手机缩短了人与人之间的距离，电话、短信让我们的交流更加快速简洁。而现如今聊天软件比如 QQ、微信、微博，几乎成了用户每天必使用的软件。很多人安装了娱乐软件诸如五花八门的音乐应用程序和视频应用程序。很多人也用手机软件来学习，诸如微信读书、知识星球等。手机支付软件也让我们的生活更加便利，吃饭，坐地铁，买东西扫码支付越来越便捷。手机软件的发展和人们的需求契合度越来越高，但同时一个严峻的问题也出现了，恶意软件就像个毒瘤危害着用户的财产安全和个人隐私。为了有效的阻拦恶意软件，保护用户的财产安全和个人隐私，安卓恶意软件检测方法逐渐显现出其独特的作用。近几年，恶意应用程序的处理手段和工具逐渐增多。文章采用静态研究方式，从影响 Android 软件安全的四个方面获取和处理特征信息。由于动态分析是基于软件行为，但需要更多的时间。本文在特征信息的处理中将方法调用序列作为影响 API 敏感强度的重要因素。然后进行实验确定分界点的值。以下描述本文重要的研究内容。

(1) 首先研究了安卓软件静态分析和动态检测方法的发展现状和基本流程。深入研究了静态分析方法的实现过程。针对现实中安卓应用程序静态分析方法误报率高，本文重点研究确定既能快速分析安卓软件恶意性又能提高准确率的方法。以敏感 API 调用序列刻画应用程序的动作，进而提高其恶意性判断的准确性。同时，本文列出了现阶段静态检测领域存在的一些问题。另外，还介绍了与本文相关的基础知识，比如安卓安全体制、安卓相关包结构及相关文件结构等。

(2) 研究了 Android 恶意软件的相关特性。以安卓恶意软件家族名称为切入点，研究了不同恶意家族的恶意应用程序，并且以恶意家族名称进行分类统计。重点对样本软件采取反编译操作，为后面的 Android 软件恶意代码检测打铺垫。本文深入分析全局配置文件和 smali 文件，提取出特征信息，并以统一的格式进行存储。

(3) 研究对 Android 恶意软件进行静态分析的基本理论和方式，创建模型。本文重点通过提取大量恶意样本和良性样本的特征信息进行对比分析。对权限信息进行处理，得到权限频率值文本和权限敏感强度文本。对 API 信息进行处理，得到 API 频率值，同时根据敏感 API 确定 API 调用序列，从而确定 API 敏感强度。使用 FAHP 方法设置权重，从而研究出静态检测安卓恶意应用程序的方法。对 Android 恶意应用程序分析方法

测试，从而评估该方案的正确性。测试数据说明该检测方法可以高效的检测安卓软件。

6.2 未来工作展望

虽然本文随机选取了大量的恶意样本和良性样本，用以进行试验和测试，并对这些样本进行反编译提取出 smali 文件，并对这些 smali 文件进行处理，获取特征信息，对特征信息进行处理，再次基本上提出了一种静态检验方法，但随着安卓软件反保护技术的发展人们对精确检测恶意软件的期望不断提高，在以后的 Android 恶意应用程序分析方面，还存在一些问题。

(1) 本实验的恶意样本是针对某一年的部分恶意 Android 软件。方法是依据恶意软件的特征信息进行一系列处理，只有不断更新恶意样本库才能更高效的进行检查。然而，研究者很难获取大量最新的恶意样本。目前国内开源的恶意软件样本库较少，像一些安全厂商可能会收集一些现在新出的恶意样本，但学者很难得到这些有效的样本。因此，获取和研究大量新型恶意应用程序对恶意软件检测具有重要作用。

(2) 目前安卓恶意应用程序分析方法发展比较完善，但不管是动态还是静态分析或者结合机器学习，这些技术或多或少都存在一些瑕疵，因此，提出结合这些方法的优点或者屏蔽其缺点的方法更有意义。

(3) 恶意软件检测方法最终目的是为用户找到安装或即将安装的软件是否是恶意性的。作为用户，当然希望这个检测方法尽可能操作简便，占内存小，运行时间短，正确率高。因此，研究更加高效简洁的安卓恶意软件检测方法仍需要学者不懈的创新和努力。

致　谢

林花谢了春红，太匆匆。从之前那个知之甚微追着时间跑的孩子，到如今，不经意间时间已过了近三年。而我仿佛还停留在研一日复一日的上课学习中，停留在研二每天在实验室看论文，做实验中，停留在小论文和大论文的撰写时。时间打个响指，突然发现研三已过去一大半。这期间学到了很多东西，也曾欢喜也曾忧伤，左不过去日匆匆，日复一日坚持努力。在即将离开校园生活之前，借此深切感谢一直指导我的老师，陪伴我的 401 的伙伴和鼓励我的朋友们，谢谢你们一直在我身边。

首先，我衷心的感谢敬爱的老师刘晓建老师，感谢老师对我的指导，栽培，给我认识 Android 软件安全世界的机会。我是刘老师带的第一个学生，老师对我一直耐心指导。刘老师为人正直踏实，细心仔细，要求严格，文质彬彬，满腹经纶。从看论文、书籍学习，到自己做实验验证，期间出现了很多想不到的问题，是刘老师悉心地指导，帮我指明方向。感谢老师在我停步不前时一直开导我。感谢刘老师在日常琐碎的生活中教会我踏实认真的道理。感谢刘老师严格的要求我，使我可以全面地了解到 Android 静态分析和动态监测中常见的技术和工具，拓宽了眼界，增长了见识。感谢刘老师让我明白，学业是一个人的硬实力，个人素养是一个人的软实力。老师，谢谢您。

其次，非常感谢 401 实验室的杨彩、李姣、康凯、胡惠凯等同学和董晓峰师弟、李妍娜师妹、雷倩师妹。感谢大家在 Android 逆向学习和安卓恶意应用程序分析方面的探讨学习。感谢师弟师妹们对我的支持和鼓励，因为有他们，我想成为更好地自己，想要不断的学习进步。感谢实验室的同学，让我有一个良好的学习氛围，感谢他们的持之以恒，让我逐渐进步。感谢大家的互相帮助和互相鼓励，让过去的每天有所收获，未来的每一天有所期待，今天有所坚持。感谢西安科技大学计算机学院 2015 级全班同学陪我度过这段有苦有甜有意义的时光。

感谢家人、亲人，感谢他们总是站在我身边，总是默默爱护我，给予我最大的关怀及感动。感谢他们无条件的理解和不问理由的支持。感谢他们一直将最好的一切给我，用汗水和皱纹换我无忧无虑的生活，因为他们，我才是我。

最后，感谢各位审阅本论文的老师。感谢出席本次论文答辩的老师。感谢你们对我的指导。

参考文献

[1] 2016-2021 年中国智能手机操作系统行业市场需求与投资咨询报告[R].中国报告大厅.2016.

[2] 2016 全球手机操作系统发展现状：iOS 市场份额继续下跌[R].中国报告大厅.2016.12.28.

[3] Jun Song, Rajiv Ranjan. An integrated static detection and analysis framework for android. Pervasive & Mobile Computing. 2016, 32:15-25.

[4] 丰生强. Android 软件安全与逆向分析[M]. 人民邮电出版社,2010.9.

[5] 卿斯汉. Android 安全研究进展[J]. Journal of Software, 2016,27(1):45-71.

[6] ZHOU Y, JIANG X. Dissecting Android Malware: Characterization and Evolution [C]. IEEE. of the 2012 IEEE Symposium on Security and Privacy, May 20-23, 2012. San Francisco Bay Area, California, IEEE Computer Society Washington, DC,USA, 2012 : 95-109.

[7] ZOU S, ZHANG J, LIN X. An Effective Behavior-based Android Malware Detection System [J]. Security and Communication Networks,2015, 8 (12) : 2079-2089.

[8] KUBOTA A. Kernel-based Behavior Analysis for Android Malware Detection [C]. IEEE. 2011 Seventh International Conference on Computational Intelligence and Security,CIS 2011,December 3-4,2011,Sanya ,Hainan. NJ: IEEE, 2011 : 1011-1015.

[9] 杨欢,张玉清等.基于多类特征的 Android 应用恶意行为检测系统[J]. 计算机学报.2014.37(1).15-27.

[10] WU D, MAO C, Wei T. DroidMat: Android Malware Detection through Manifest and API Calls Tracing[C]. IEEE. 2012 Seventh Asia Joint Conference on Information Security,August 9-10, 2012, Tokyo,Japan. NJ: IEEE, 2012: 62-69.

[11] GASCON H, YAMAGUCHI F, ARP D, et al. Structural Detection of Android Malware Using Embedded Call Graphs[C]. ACM. of the 2013 ACM Workshop on Artificial Intelligence and Security, November. 4-8 Berlin, Germany. NY: ACM, 2013: 45-54.

[12] AVG. Malware Detection Methods[EB/OL]. http://www.avg.com/us-en/avg-software-technology, 2016-1-15.

[13] Android and Security [EB/OL]. http://googlemobile.blogspot.com/2012/02/android-and- security.html, 2016-1-15.

[14] 张骁敏, 刘静等.基于权限与行为的 Android 恶意软件检测研究[J]. 网络与信息安全学报.2017.(3).51-57.

[15] SPREITZENBARTH M, SCHRECK T, ECHTLER F. Mobile-Sandbox: Combining Static and Dynamic Analysis with Machine-learning Techniques[J]. International Journal of Information Security, 2014:1-13.

[16] 蔡林,陈铁明. Android 移动恶意代码检测的研究概述与展望[J]. Netinfo Security,2016. 09.043.

[17] http://code.google.com/p/androguard/wiki/Installation.

[18] Felt AP, Chin E, Hanna S, Song D, Wagner D. Android permissions demystified. In: Proc. of the 18th ACM Conf. on Computer and Communications Security (CCS 2011). 2011. 627-638.

[19] Sarma BP, Li N, Gates C, Potharaju R, Nita-Rotaru C. Android permissions: A perspective combining risks and benefits. In: Proc. of the ACM Symp. on Access Control Models and Technologies (SACMAT 2012). 2012. 13-22.

[20] Barrera D, Kayacik HG, van Oorschot PC, Somayaji A. A methodology for empirical analysis of permission-based security models and its application to Android. In: Proc. of the 17th ACM Conf. on Computer and Communications Security (CCS 2010). 2010. 73-84.

[21] Erik RamsgaardWognsen, HenrikSøndbergKarlsen, MadsChr.Olesen. Formalisation and analysis of Dalvik bytecode Google Project Hosting. Science of Computer Programming.2014:25-55.

[22] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification. In: Proc. of the 16th ACM Conf. On Computer and Communications Security (CCS 2009). 2009. 235-245.

[23] Fuchs AP, Chaudhuri A, http://www.cs.umd.edu/~avik/papers/scandroidascaa.pdf.

[24] Chan PPF, Hui LCK, Yiu SM. Droidchecker: Analyzing Android applications for capability leak. In: Proc. of the 15th ACM Conf.on Security and Privacy in Wireless and Mobile Networks (WiSec 2012). 2012. 125-136.

[25] Lu L, Li Z, Wu Z, Lee W, Jiang G. Chex: Statically vetting Android apps for component hijacking vulnerabilities. In: Proc. of the 19th ACM Conf. on Computer and Communications Security (CCS 2012). 2012. 229-240.

[26] Gibler C, Crussell J, Erickson J, Chen H. AndroidLeaks: Automatically detecting

potential privacy leaks in Android applications on a large scale. In: Proc. of the 5th Int'l Conf. (TRUST 2012). LNCS 7344, Springer-Verlag, 2012. 291-307.

[27] Mohsin Junaid, Donggang Liu. Dexteroid: Detecting malicious behaviors in Android apps using reverse-engineered lifecycle models[J]. computers &security.2016(59): 92-117.

[28] Octeau D, McDaniel P, Jha S, Bartel A, Bodden E, Klein J, Traon YL. Effective inter-component communication mapping in Android with EPICC: An essential step towards holistic security analysis. In: Proc. of the 22nd USENIX Security Symp. (USENIX 2013). 2013.

[29] Cui XM, Yu D, Chan P, Hui Lucas CK, Yiu SM, Qing SH. CoChecker: Detecting capability and sensitive data leaks from component chains in Android. In: Proc. of the 19th Australasian Conf. (ACISP 2014). LNCS 8544, Springer-Verlag, 2014. 446-453.

[30] Kim J, Yoon Y, Yi K, Shin J. ScanDal: Static analyzer for detecting privacy leaks in Android applications. In: Proc. of the IEEE Workshop on Mobile Security Technologies (MoST 2012). 2012. http://www.mostconf.org/2012/papers/26.pdf.

[31] 文伟平, 梅瑞, 宁戈,等. Android 恶意软件检测技术分析和应用研究[J]. 通信学报,2014,35(8):79-85.

[32] Peiravian N, Zhu X. Machine learning for Android: Malware detection using permission and API calls. In: Proc. of the 25th IEEE Int'l Conf. on Tools with Artificial Intelligence (ICTAI 2013). 2013. 300-305.

[33] BURGUERA I,ZURATUZA U. 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.

[34] 卜义云, 基于机器学习的 Android 恶意软件静态检测系统的设计实现[D]. 成都:电子科技大学.2016.

[35] YERIMA S Y, SEZER S, MCWILLIAN G.Analysis of Bayesian classification-based approaches for Android malware detection[J]. Information Security, IET, 2013,8(1): 121-129.

[36] 王聪,张仁斌,李刚. 基于关联特征的贝叶斯 Android 恶意程序检测技术[J]. 计算机应用与软件.2017.01.34(1).286-292.

[37] SCHMIDT A D, BYE R, SCHMIDT H G. Static analysis of executables for collaborative malware detection on Android[C]. IEEE International Conference on

Communications. Dresden, c2009:1-5.

[38] CEN L, GATES C. A probabilistic discriminative model for Android malware detection with decompiled source code[C]. IEEE Transactions on Dependable and Secure Computing. C2013:1-14.

[39] Shina Sheen n, R.Anitha,V.Natarajan. Android based malware detection using a multifeature collaborative decision fusion approach[J]. Neurocomputing 151.2015: 905–912

[40] Huang CY, Tsai YT, Hsu CH. Performance evaluation on permission-based detection for Android malware. In: Proc. of the Int'l Computer Symp. (ICS 2012). Springer -Verlag, 2012. 111-120.

[41] Sanz B, Santos I, Laorden C, Ugarte-Pedrero X, Bringas PG, Álvarez G. Puma: Permission usage to detect malware in Android. In: Proc. of the Int'l Joint Conf. CISIS 2012-ICEUTE 2012-SOCO 2012 Special Sessions. Springer-Verlag, 2013. 289-298.

[42] Wolfe B, Elish K, Yao DF. High precision screening for Android malware with dimensionality reduction. In: Proc. of the IEEE 2014 13th Int'l Conf. on Machine Learning and Applications (ICMLA 2014). 2014. 21-28.

[43] Shabtai A, Fledel Y, Elovici Y. Automated static code analysis for classifying Android applications using machine learning. In:Proc. of the IEEE 2010 Int'l Conf. on Computational Intelligence and Security (CIS 2010). 2010. 329-333.

[44] Aafer Y, Du W, Yin H. DroidAPIMiner: Mining API-level features for robust malware detection in Android. In: Proc. of the 9th Int'l ICST Conf. (SecureComm 2013). Springer-Verlag, 2013. 86-103.

[45] Wolfe B, Elish KO, Yao D. Comprehensive behavior profiling for proactive Android malware detection. In: Proc. of the 17th Int'l Conf.(ISC 2014). LNCS 8783, Springer-Verlag, 2014. 328-344.

[46] Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K. DREBIN: Effective and explainable detection of Android malware in your pocket. In: Proc. of the 21st Network and Distributed System Security Symp. (NDSS 2014). 2014.

[47] Sahs J, Khan L. A machine learning approach to Android malware detection. In: Proc. of the IEEE 2012 European Intelligence and Security Informatics Conf. 2012. 141-147.

[48] Peng H, Gates C, Sarma B, Li N, Qi Y, Potharaju R, Nita-Rotaru C, Molloy I. Using probabilistic generative models for ranking risks of Android apps. In: Proc. of the 19th

ACM Conf. on Computer and Communications Security (CCS 2012). 2012. 241-252.

[49] Songyang Wu, Pan Wang, Xun Li. Effective detection of android malware based on the usage of data flow APIs and machine learning[J]Information and Software Technology. 2016(75) :17-25.

[50] Amos B, Turner H, White J. Applying machine learning classifiers to dynamic Android malware detection at scale. In: Proc. of the 9th IEEE Int'l Wireless Communications and Mobile Computing Conf. (IWCMC 2013). 2013. 1666-1671.

[51] LIU Wu, REN Ping, LIU Ke, Behavior-based Malware Analysis and Detection [C]. IEEE. 2011 First International Workshop on Complexity and DataMining (IWCDM), September 24-28,2011,Nanjing, Jiangsu, China, NJ: IEEE, 2011 : 39-42.

[52] 李政廉.基于 API 关联性的恶意代码分析技术研究[D]. 郑州:解放军信息工程大学.2014.

[53] Stowaway. http://www.android-permissions.org/.

[54] Dietz M, Shekhar S, Pisetsky Y, Shu A, Wallach DS. Quire: Lightweight provenance for smart phone operating systems. In: Proc.of the 20th USENIX Security Symp. (USENIX 2011). 2011. 24.

[55] Enck W, Gilbert P, Chun BG, Cox LP, Jung J, McDaniel P, Sheth AN. TaintDroid: An information flow tracking system for realtime privacy monitoring on smartphones. Communications of the ACM, 2014,57(3):99-106.

[56] Huang J, Zhang X, Tan L, Wang P, Liang B. Asdroid: Detecting stealthy behaviors in Android applications by user interface and program behavior contradiction. In: Proc. of the 36th Int'l Conf. on Software Engineering (ICSE 2014). 2014. 1036-1046.

[57] Burguera I, Zurutuza U, Nadjm-Tehrani S. Crowdroid: Behavior-Based malware detection system for Android. In: Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM 2011). 2011. 15-26.

[58] 赵洋,胡龙等.基于沙盒的 Android 恶意软件动态分析方案[J]. 信息网络安全,2014.12:21-26.

[59] Zhaoguo Wang, Chenglong Li, Zhenlong Yuan.DroidChain: A novel Android malware detection method based on behavior chains[J]. Pervasive and Mobile Computing.2016, 32: 3-14.

[60] Bugiel S, Davi L, Dmitrienko A, Fischer T, Sadeghi AR. Xmandroid: A new Android evolution to mitigate privilege escalation attacks. Technical Report, TR-2011-04, 2011.

[61] Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y. Andromaly: A behavioral malware detection framework for Android devices. Journal of Intelligent Information Systems [J] Intelligent Information Systems.2012,38(1):161-190.

[62] 李晓光. Android 软件恶意行为静态检测技术研究[D]. 哈尔滨:哈尔滨工程大学, 2015.

[63] 徐艳萍,马兆丰. Android 智能终端安全综述[J]. 通信学报.2016.06. 37(6).

[64] 吴俊昌,骆培杰.基于权限分类的 Android 应用程序的静态分析[C]. VARA2011.

附　录（一）

硕士期间发表的论文:

[1] 刘晓建,姜婷.Formal Definition of Program Faults and Hierarchy of Program[J].ICISCE (EI 检索)

硕士期间参与的项目：

[1] 《Android 应用程序恶意隐私信息泄露行为的检测方法研究》

附　录（二）

本实验权限方法映射关系表

权限	API
android.permission.ACCESS_LOCATION_EXTRA_COMMANDS	Landroid/location/LocationManager;->sendExtraCommand(Ljava/lang/String;Ljava/lang/String;Landroid/os/Bundle;)Z
android.permission.CHANGE_NETWORK_STATE	Landroid/net/ConnectivityManager;->requestRouteToHost(II)Z
	Lcom/android/internal/telephony/ITelephony$Stub;->setRadio(Z)Z
	Landroid/net/ConnectivityManager;->getBackgroundDataSetting()Z
	Landroid/net/ConnectivityManager;->setMobileDataEnabled(Z)V
	Landroid/net/ConnectivityManager;->getNetworkInfo(I)Landroid/net/NetworkInfo;
	Landroid/net/ConnectivityManager;->setMobileDataEnabled(Z)V
	Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
	Landroid/net/ConnectivityManager;->startUsingNetworkFeature(ILjava/lang/String;)I
	Landroid/net/ConnectivityManager;->isActiveNetworkMetered()Z
	Landroid/net/ConnectivityManager;->getAllNetworkInfo()[Landroid/net/NetworkInfo;
	Landroid/net/ConnectivityManager;->stopUsingNetworkFeature(ILjava/lang/String;)I
android.permission.CHANGE_WIFI_STATE	Landroid/net/wifi/WifiManager;->startScan()Z
	Landroid/net/wifi/WifiManager;->addNetwork(Landroid/net/wifi/WifiConfiguration;)I
	Landroid/net/wifi/WifiManager;->getConnectionInfo()Landroid/net/wifi/WifiInfo;
	Landroid/net/wifi/WifiManager;->isWifiEnabled()Z
	Landroid/net/wifi/WifiManager;->getScanResults()Ljava/util/List;
	Landroid/net/wifi/WifiManager;->getWifiState()I

	Landroid/net/wifi/WifiManager;->setWifiEnabled(Z)Z
	Landroid/net/wifi/WifiManager$WifiLock;->release()V
	Landroid/net/wifi/WifiManager$WifiLock;->acquire()V
	Landroid/net/wifi/WifiManager;->createWifiLock(Ljava/lang/String;)Landroid/net/wifi/WifiManager$WifiLock;
	Landroid/net/wifi/WifiManager;->saveConfiguration()Z
	Landroid/net/wifi/WifiManager;->disableNetwork(I)Z
	Landroid/net/wifi/WifiManager;->enableNetwork(IZ)Z
	Landroid/net/wifi/WifiManager;->removeNetwork(I)Z
	Landroid/net/wifi/WifiManager;->reconnect()Z
	Landroid/net/wifi/WifiManager;->getConfiguredNetworks()Ljava/util/List;
	Landroid/net/wifi/WifiManager$WifiLock;->isHeld()Z
android.permission.DELETE_PACKAGES	Landroid/content/pm/PackageManager;->deletePackage(Ljava/lang/String;Landroid/content/pm/IPackageDeleteObserver;I)V
android.permission.DISABLE_KEYGUARD	Landroid/app/KeyguardManager;->inKeyguardRestrictedInputMode()Z
	Landroid/app/ActivityManager;->restartPackage(Ljava/lang/String;)V
	Landroid/app/ActivityManager;->getRunningAppProcesses()Ljava/util/List;
	Landroid/app/KeyguardManager;->newKeyguardLock(Ljava/lang/String;)Landroid/app/KeyguardManager$KeyguardLock;
	Landroid/app/ActivityManager;->getMemoryInfo(Landroid/app/ActivityManager$MemoryInfo;)V
	Landroid/app/ActivityManager;->getRecentTasks(II)Ljava/util/List;
	Landroid/app/KeyguardManager;->exitKeyguardSecurely(Landroid/app/KeyguardManager$OnKeyguardExitResult;)V
	Landroid/app/ActivityManager;->getProcessMemoryInfo([I)[Landroid/os/Debug$MemoryInfo;
	Landroid/app/ActivityManager;->killBackgroundProcesses(Ljava/lang/String;)V
	Landroid/app/ActivityManager;->getRunningServices(I)Ljava/util/List;
	Landroid/app/ActivityManager;->getRunningTasks(I)Ljava/util/List;

	Landroid/app/ActivityManager;->getDeviceConfigurationInfo()Landroid/content/pm/ConfigurationInfo;
android.permission.INSTALL_PACKAGES	Lcom/netqin/antivirus/softupdate/SoftwareUpdate;->installPackage(Landroid/content/ContentValues;)V
	Landroid/content/pm/PackageManager;->checkPermission(Ljava/lang/String;Ljava/lang/String;)I
	Landroid/content/pm/PackageManager;->getInstalledPackages(I)Ljava/util/List;
	Landroid/content/pm/PackageManager$NameNotFoundException;->printStackTrace()V
	Landroid/content/pm/PackageManager;->getPackageArchiveInfo(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;
	Landroid/content/pm/PackageManager;->getPackageInfo(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;
android.permission.RECEIVE_SMS	Landroid/telephony/SmsManager;->getDefault()Landroid/telephony/SmsManager;
android.permission.SEND_SMS	Landroid/telephony/gsm/SmsManager;->sendDataMessage(Ljava/lang/String;Ljava/lang/String;S[BLandroid/app/PendingIntent;Landroid/app/PendingIntent;)V
	Landroid/telephony/SmsManager;->sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V
	Landroid/telephony/gsm/SmsManager;->sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V
	Landroid/telephony/SmsManager;->divideMessage(Ljava/lang/String;)Ljava/util/ArrayList;
	Landroid/telephony/SmsManager;->sendMultipartTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/util/ArrayList;Ljava/util/ArrayList;Ljava/util/ArrayList;)V
android.permission.WRITE_SECURE_SETTINGS	Landroid/webkit/WebSettings;->setGeolocationEnabled(Z)V
	Landroid/webkit/WebSettings;->setUseWideViewPort(Z)V

	Landroid/webkit/WebSettings;->setPluginState(Landroid/webkit/WebSettings$PluginState;)V
	Landroid/webkit/WebSettings;->setCacheMode(I)V
	Landroid/webkit/WebSettings;->setUserAgentString(Ljava/lang/String;)V
	Landroid/webkit/WebSettings;->setLoadsImagesAutomatically(Z)V
	Landroid/webkit/WebSettings;->setSaveFormData(Z)V
	Landroid/webkit/WebSettings;->setSavePassword(Z)V
	Landroid/content/ContentResolver;->bulkInsert(Landroid/net/Uri;[Landroid/content/ContentValues;)I
	Landroid/webkit/WebSettings;->setRenderPriority(Landroid/webkit/WebSettings$RenderPriority;)V
	Landroid/webkit/WebSettings;->setAllowFileAccess(Z)V
	Landroid/webkit/WebSettings;->setJavaScriptCanOpenWindowsAutomatically(Z)V
	Landroid/webkit/WebSettings;->setSupportZoom(Z)V
	Landroid/webkit/WebSettings;->setPluginsEnabled(Z)V
	Landroid/webkit/WebSettings;->setBuiltInZoomControls(Z)V
	Landroid/webkit/WebSettings;->setJavaScriptEnabled(Z)V
	Landroid/content/ContentProvider;->attachInfo(Landroid/content/Context;Landroid/content/pm/ProviderInfo;)V
	Landroid/app/ProgressDialog;->setMessage(Ljava/lang/CharSequence;)V