论文题目：基于多上下文特征的Android恶意软件家族检测方法研究

专 业：软件工程

硕 士 生：\*\* （签名）

指导教师：\*\*\* （签名）

摘 要

随着Android应用程序的普及，Android恶意软件的产生和传播呈爆发式增长，从而引起人们对Android应用程序安全性的高度关注，迫切需要开发有效的解决方案来缓解和防御恶意攻击。目前常用的解决方法有基于数据流的污点分析方法和基于机器学习的恶意性检测方法。由于基于数据流的污点分析方法要对程序恶意行为的机理进行分析，因此通常具有分析复杂度高，分析方法过于保守等缺点。而基于机器学习的检测方法通过对程序特征数据进行统计分析，建立程序特征与行为的映射模型，从而对程序的恶意性进行判断，该方法具有简单，高效等优点。但目前现有的机器学习工作仍存在以下问题：（1）由于分类方法的检测准确性取决于特征的质量，因此特征的选择和生成是影响此方法的主要因素，而现有的大多数工作在没有完全检查程序体系结构的情况下选择特征，因此丢失了与特征相关的重要语义信息，导致检测的准确率低；（2）基于机器学习的工作将恶意软件检测视为二元分类问题，只能判断出程序是否为恶意程序，而无法让用户了解到程序具体存在的恶意行为。为了对抗以上问题，本文提出了一种基于多上下文特征的Android恶意软件家族检测方法，主要研究内容如下：

1. 基于多上下文特征的恶意性判别方法。为了解决传统基于机器学习算法的检测方法丢失了程序语义信息这一问题，本文首先选择敏感权限，广义敏感API和敏感系统广播这三类特征作为原始特征，然后将这些原始特征与其发生的上下文结合形成程序特征，从而将程序的语义信息考虑在内，提高了恶意程序的检测效率。另外，为了有效提取特征，本文构造了基于回调函数的iCFG，并定义了一组应用于该图的约简规则，能够在不丢失图信息的基础上降低图规模；
2. 基于文本分析技术的恶意家族预测方法。随着代码重用技术的不断发展，以及Android恶意软件家族内应用程序行为的相似性，本文将文本分析技术应用于Android恶意程序的家族预测问题，首先分析样本集中每个恶意软件家族中应用程序的行为特性，然后为每个恶意软件家族构造家族特征向量，最后构造1-NN分类器，对（1）中判断为恶意但未知家族的程序进行家族预测；
3. 基于人工分析的安全摘要生成方法。通过分析提取Android恶意软件家族中所有应用程序存在的共同行为，将该行为作为恶意软件家族产生的恶意类别，然后对提取出的共同行为进行人工分析，从而为每一个恶意软件家族生成一个安全摘要，最后在（2）预测出恶意程序所属家族的基础上，通过家族的安全摘要从而给出恶意程序的安全摘要。

关键词：Android系统；多上下文；广义敏感API；机器学习；文本挖掘；安全摘要

研究类型：理论研究

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject** | **:** | **Research on Android Malware Family Detection Based on Multi-context Features** | |
| **Specialty** | **:** | **Software Engineering** | |
| **Name** | **:** | **\*\*** | **(Signature)** |
| **Instructor** | **:** | **\*\*\*** | **(Signature)** |

**ABSTRACT**

With the popularization of Android applications in our lives, the generation and spread of Android malware has been greatly stimulated, which has drawn attention to Android security. Therefore, it is urgent to develop effective solutions to mitigate and prevent malicious attacks. . At present, the commonly used method to solve this problem is to treat malware detection as a binary classification problem, and use traditional machine learning techniques to distinguish unknown applications as benign or malicious. The problems with this method are: (1) Since the detection accuracy of the classification method mainly depends on the quality of the features, the generation and selection of features is the main factor affecting this method, and most of the existing work generates and Features are selected, so important semantic information related to these features is lost, resulting in low detection accuracy; (2) Treating malware detection as a binary classification problem can only determine whether the program is a malicious program and cannot let users know To the specific malicious behavior of the program. In order to combat the above problems, this paper proposes a method for analyzing the behavior of Android malicious programs based on multi-context features. The main research contents and results are as follows:

1. In order to solve the problem that the traditional machine learning algorithm-based detection method loses program semantic information, this paper first selects three types of features: sensitive permissions, generalized sensitive API and sensitive system broadcast as original features, and then uses these original features to occur with them. The combination of the context and the program features, which greatly improves the detection efficiency of malicious programs. In addition, in order to effectively extract features, this paper constructs a callback function-based iCFG and defines a set of reduction rules applied to the graph, which can reduce the graph size while maintaining the graph information;
2. With the continuous development of code reuse technology and the similarity of application behaviors in the Android malware family, this paper applies the text mining technology to the family prediction problem of the Android malware, and first analyzes each malware family in the data set. The behavior characteristics of the application program, then construct a family feature vector for each malware family, and finally construct a 1-NN classifier to perform family prediction on the malicious program judged as malicious but unknown in (1);
3. By analyzing and extracting common behaviors existing in all applications in the Android malware family, using this behavior as a malicious category generated by the malware family, and then manually analyzing the extracted common behaviors to generate for each malware family A security summary, based on (2) predicting the family to which the malicious program belongs, gives the security summary of the malicious program through the family's security summary.

**Key words:** Android system; Multi-context; Generalized sensitive API; Machine learning; Text mining; Security summary

**Thesis :** Theoretical Research

目 录

[第 1 章 绪 论 1](#_Toc8069)

[1.1 选题景及意义 1](#_Toc21553)

[1.2 国内外研究现状及发展方向 3](#_Toc2490)

[1.2.1基于机器学习检测方法 3](#_Toc23206)

[1.2.2 基于图相似的检测方法 4](#_Toc3768)

[1.2.3 基于数据流的污点分析方法 5](#_Toc19372)

[1.2.4 基于上下文的检测方法 6](#_Toc24647)

[1.3 研究内容 7](#_Toc31559)

[1.4 论文结构安排 8](#_Toc11844)

[第 2 章 相关知识介绍 9](#_Toc7685)

[2.1 Android体系结构及安全机制 9](#_Toc13963)

[2.1.1 Android体系结构 9](#_Toc28622)

[2.1.2 Android安全机制 10](#_Toc19699)

[2.2 Soot概述 11](#_Toc9829)

[2.2.1 Soot简介 11](#_Toc25688)

[2.2.2 Jimple中间表示 11](#_Toc13527)

[2.2.3 Soot数据结构 12](#_Toc10439)

[2.3 函数调用图 12](#_Toc20319)

[2.4 过程内控制流图 13](#_Toc31821)

[第 3 章 基于图变换的Android应用程序特征提取 16](#_Toc23444)

[3.1引言 16](#_Toc29431)

[3.2 样例分析 16](#_Toc5851)

[3.3 特征选择 18](#_Toc4091)

[3.3.1 原始特征 18](#_Toc19813)

[3.3.2 原始特征的上下文 20](#_Toc21812)

[3.4 特征获取 21](#_Toc3666)

[3.4.1 特征获取过程 21](#_Toc32583)

[3.4.2 回调函数iCFG的构造 22](#_Toc28073)

[3.4.3 iCFG的约简 24](#_Toc23600)

[3.4.4获取广义敏感API和敏感权限特征 27](#_Toc11073)

[3.4.5 获取敏感系统广播特征 28](#_Toc22287)

[第 4 章 基于多上下文特征的Android应用程序恶意性判定 29](#_Toc3541)

[4.1 引言 29](#_Toc27489)

[4.2 特征变换 29](#_Toc31703)

[4.2.1 特征向量定义 29](#_Toc785)

[4.2.2 特征值的计算 29](#_Toc26563)

[4.3 机器学习算法 32](#_Toc7683)

[4.3.1 随机森林算法 32](#_Toc5105)

[4.3.2 朴素贝叶斯算法 33](#_Toc22280)

[4.3.3 支持向量机 33](#_Toc4684)

[4.4 方法实现与实验评估 34](#_Toc14684)

[4.4.1 方法实现 34](#_Toc17993)

[4.4.1 数据集和度量指标 35](#_Toc13079)

[4.4.2 约简转换的有效性验证 36](#_Toc9240)

[4.4.3 基于多种分类算法的分类器训练 38](#_Toc5187)

[第 5 章 基于文本分析技术的家族预测和安全摘要生成 42](#_Toc32234)

[5.1 引言 42](#_Toc18283)

[5.2 API图环压缩处理 43](#_Toc30901)

[5.2.1 环的识别 43](#_Toc6115)

[5.2.2 环的压缩 43](#_Toc19432)

[5.3 API调用序列提取 45](#_Toc15133)

[5.4 模糊处理 46](#_Toc19431)

[5.4.1 API调用序列的局限性 46](#_Toc15805)

[5.4.2 模糊处理规则 46](#_Toc32077)

[5.3 Android恶意程序家族中行为块分析 49](#_Toc14933)

[5.3.1 分析指标 49](#_Toc7999)

[5.3.2 分析与讨论 50](#_Toc12012)

[5.4 基于文本挖掘技术的家族预测器构造 57](#_Toc19798)

[5.4.1 向量空间模型 57](#_Toc12213)

[5.4.2 案例说明 59](#_Toc7663)

[5.4.3 预测器构造 59](#_Toc26797)

[5.5 实验及结果分析 61](#_Toc2016)

[5.5.1 环压缩的有效性 61](#_Toc11509)

[5.5.2 1-NN分类器的有效性 62](#_Toc29405)

[5.5.3 与相关工作对比 63](#_Toc2590)

[5.6 安全摘要生成 64](#_Toc12317)

[5.6.1 家族行为提取 64](#_Toc18122)

[5.6.2 家族安全摘要生成 64](#_Toc17654)

[第 6 章 总结与展望 66](#_Toc32651)

[6.1 工作总结 66](#_Toc26637)

[6.2 工作展望 67](#_Toc19909)

[致 谢 68](#_Toc32086)

[参考文献 69](#_Toc20775)

[附 录 72](#_Toc24601)

第 1 章 绪 论

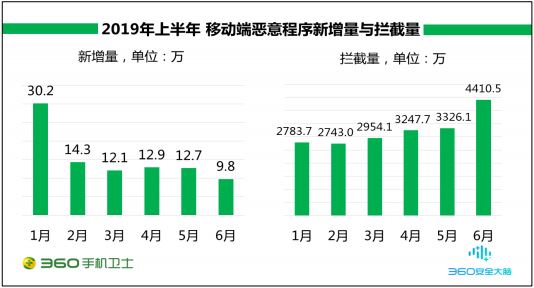
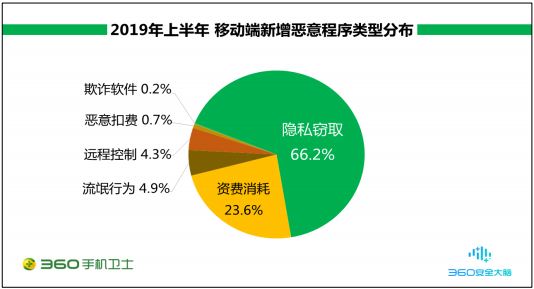
## 1.1 选题景及意义

近年来，移动设备以其特有的便利性，全面性和高性能成为人们生活中的必需品，人们在网络上的一切活动均可在移动设备上完成，它成为个人处理重要事务的核心，成为存储个人信息最多的设备。但与此同时，Android系统存在着非常大的安全风险，主要原因有：（1）Android系统源码的开放性；（2）系统碎片化的现象难以遏制，导致系统漏洞难以完全修复；（3）权限机制的局限性；（4）Android应用程序市场管理混乱，缺乏严格的审核机制。因此，Android平台成为黑客的主要攻击对象。如FireEye Labs发现的Kemoge是一款来自于第三方应用商店的恶意软件，该软件通过利用root工具获取root权限，从而提升自身权限，并将一些恶意程序安装包植入系统分区。而且该软件能够获取设备信息上传至远程服务器，根据服务器的返回指令对设备进行操作，如卸载安全保护程序，安装恶意程序等。层出不穷的恶意软件实现恶意攻击的手段不尽相同，目前常见的恶意软件攻击方式和对应的实现手段如表1.1所示。

表1.1 恶意程序攻击方式及实现手段

|  |  |
| --- | --- |
| 攻击方式 | 实现手段 |
| 权限提升 | 获取root权限，从而拥有对设备进行任意操作的权限，然后对设备进行恶意操作。 |
| 恶意扣费 | 发送短信给运营商定制某些收费服务，后台监听RECEIVE\_SMS广播，拦截运营商返回的短信消息，从而使得用户意识不到该事件的发生。 |
| 隐私窃取 | 后台窃取用户的个人信息，如电话簿，设备id，位置信息等，然后发送给远程服务器。 |
| 远程控制 | 后台接收由远程服务器发送的指令并执行，并返回某些信息给远程服务器。 |
| 跟踪监视 | 通过后台获取系统服务，实现监听手机通话状态，监视短信信息的收发状态，跟踪用户地理位置等功能。 |
| 恶意传播 | 获取手机联系人并向联系人发送带有恶意链接的短消息，当其他用户点击该链接时便会感染恶意程序。 |
| 流量消耗 | 通过后台下载软件，浏览网页，下载网页，上传数据等操作消耗用户流量，造成经济损失。 |

由360发布的《2019上半年手机安全状况报告》[1]显示，在2019年上半年，360平台共截获恶意软件程序共92万，360手机卫士累计拦截恶意程序攻击约1.9亿次，平均每天拦截手机恶意程序107.5万次。增长量同比2018年上半年的增长（283.1万个）虽有所下降，但其增长数目依旧庞大，且新增恶意程序类型主要为隐私窃取，占比高达66.2%，其次为资费消耗（23.6%），造成巨大的用户经济损失。图1.1(a)展示了2019年上半年移动端恶意程序新增量与360恶意程序的拦截量，图1.1(b)展示了同时期Android平台新增恶意程序的类型分布。

1. (b)

图1.1 恶意程序新增量与拦截量和新增程序的类型分布

随着恶意程序数量的不断增长，其带来的安全问题也日益严重。针对用户隐私信息，财产安全甚至是国家经济政治安全的威胁都势必带来巨大的经济损失。另一方面，随着代码混淆，加密等技术的不断发展，使得现有的恶意程序进化速度非常快，但对应的反恶意程序的解决方案严重滞后。因此，分析现有恶意程序的行为特征，总结其实现恶意攻击的行为模式，进而研究相应的检测策略，形成能够对未知危害性样本进行准确判断的检测工具势在必行。Android逆向分析技术的发展，使我们摆脱字节码文件，能够直接从接近源码的反编译文件着手分析，这对于缓解目前移动安全严峻的现状具有不可忽视的意义。

现阶段基于机器学习的恶意程序检测方法已成为目前主流的Android恶意软件研究方法之一，该方法主要通过从程序样本中抽取典型特征，构造特征向量，进而采用机器学习算法进行分类器训练。这种技术的关键在于特征集的选择，不同的特征集会对检测结果产生很大的影响。API和权限是应用程序执行恶意操作的基础，权限代表了应用程序能够访问的敏感资源和数据集合，而API预示了应用程序将要在这些资源和数据上进行的所有操作，因此，现有大多数基于机器学习的工作都是以这两类特征作为依据进行恶意性判别[2-11]，但该类工作将恶意软件检测视为一个二元分类问题（即将应用程序区分为恶意或良性），并且这种方法仅仅考虑了与程序特征相关的语法信息，不能完全研究程序的语义信息。而现有的恶意程序通常通过模仿良性程序的安全敏感行为和后台服务隐藏其恶意负载的发生，这为基于机器学习的恶意程序检测方法带来了一个新的挑战。从文献[12,13]可以了解到，恶意程序和良性程序的主要区别在于恶意应用通常在用户无意识的情况下在后台调用敏感API，从而完成某种恶意操作。因此，敏感API调用的恶意性与其上下文有着密切联系。并且，Android恶意软件可通过系统广播或某些系统回调事件触发其恶意负载，如BOOT\_COMPLETED事件会在系统完成启动时被触发，恶意软件通常会监听此事件在系统启动后启动其后台服务。因此，关注应用所监听的系统回调事件也能够对Android恶意程序的检测提供一定的帮助。

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。但由于其粗粒度的管理使得应用程序可能会请求多余权限，攻击者也可能会请求无用的正常权限来隐藏自己的恶意性。因此，仅通过程序申请的权限判断其可能的行为并不是完全可靠的。并且同一权限在不同组件中使用，产生恶意行为的可能性也是不同的，比如通过分析发现，恶意程序经常会在service组件中执行某些恶意操作，使用到某些敏感权限，而很少在Activity组件中出现，因此若某service组件使用到敏感权限，则该应用产生恶意行为的可能性要比Activity使用该权限产生恶意行为的可能性要大得多。

基于以上背景，本文提出了一种基于多上下文特征的恶意软件家族检测方法，首先结合程序的语法和语义特征，采用机器学习算法对程序的恶意性进行判定，然后在程序被判定为恶意的基础上，采用文本分析技术对恶意程序的家族进行预测，最后根据家族内程序的共同行为生成家族的安全摘要。

## 1.2 国内外研究现状及发展方向

目前针对Android恶意程序的检测技术路线主要分为两条技术路线：静态检测和动态检测。动态检测方法[14-18]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。静态检测方法不执行程序，而是直接对程序文本进行分析，其实质是对程序行为做某种程度的抽象和逼近。相比于动态分析，静态分析具有代码覆盖率高，运行速度快，分析结果不受运行环境制约等优点。本文主要研究静态检测方法，因此主要讨论静态检测方法现有的工作以及进展。

静态检测方法又可以进一步细分，依据是否对程序内部结构进行详细分析，可以将其分为两大类：“白盒”方法和“黑盒”方法。白盒方法通过对程序内部结构和行为的分析，建立程序语义与程序性质之间的关系，从而进行恶意行为检测。白盒分析方法的实质是运用不同的抽象手段，对程序行为进行不同程度的逼近。典型的白盒方法有基于数据流分析的检测方法和基于上下文的检测方法。

与之相反，黑盒方法不对程序内部结构和行为进行详尽分析，而是运用统计分析技术，通过建立程序的一系列外在特征与程序恶意性之间的数学模型来检测目标程序的性质。特别是随着大量程序样本的积累和机器学习，自然语言处理和图匹配等技术的快速发展和日臻成熟，基于机器学习和图相似的检测技术已经成为解决恶意行为识别和检测问题的主要技术手段。

1.2.1基于机器学习检测方法

基于机器学习的检测方法通过对大量程序特征数据的提取，然后运用统计方法，建立程序的外在特征与程序恶意性质之间的数学模型，从而达到检测目的。该方法并不考察程序的内部结构或程序的语义特征，而是通过程序一系列外在特征检测程序的行为。由于API和权限是应用程序的两大重要典型特征，因此很多方法方法都以这两类特征作为判据。

代表性的工作如下。Enck[2]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。邵舒迪[3]等人提出了基于权限和API特征结合的检测方法，首先分别从Manifest.xml文件和smali文件中提取权限和API特征，并将这两者结合作为一个整体特征集合，然后构造特征向量，采用机器学习算法进行分类器训练。该方法的权限特征来源于Manifest.xml文件，而Manifest.xml文件只提供了应用程序申请的权限集合，并不代表应用程序实际使用的权限集合，当应用申请的权限与实际使用的权限存在不一致性时，该方法便会造成较高误报率。Drebin[4]从程序的Manifest文件和反编译代码中提取组件、权限、intent过滤器、敏感API、网络地址等程序基本特征，然后将这些特征嵌入到特征向量空间中，并采用支持向量机等学习算法建立学习模型。为了克服黑盒方法不能对判定结果进行解释的缺陷，Drebin还实现了一个解释器，对程序的特征集进行解释，使检测结果容易理解。DroidDeep[5]提出了一种基于深度学习模型的Android平台的恶意软件检测方法，该方法首先考虑能反映Android应用程序行为模式的静态信息，如权限、API调用、组件部署等，并从包含30000个特征的应用程序集中提取了一个多级特征集，然后将这些提取的特征提供给深度学习模型，以学习典型的分类特征。杨欢等人[6]采用动静结合的方法提取应用的组件、函数调用以及系统调用3类特征，针对这3类特征设计了三层混合系综算法THEA，该算法通过构建适合3类特征的最优分类器来综合评判Android应用的恶意行为。

1.2.2 基于图相似的检测方法

基于图相似的检测方法与基于机器学习的检测方法类似，将恶意程序的检测问题定义为模式匹配问题，通过将程序抽象成图结构，用图来反映程序的行为特征，然后通过计算图之间的相似性从而判断程序的恶意性质。

Fan M等人[19]提出了一种基于敏感API权重的图匹配方法检测恶意家族共性恶意行为的方法，并采用频率子图来表示恶意家族共性行为特征，开发了FalDroid工具对恶意程序进行检测和分类。[20]采用恶意程序可视化技术[21]，首先把程序转化为2维灰度图像，然后通过对图像纹理的识别和分类来解决恶意程序的识别和分类问题。Silvio C等人[22]采用控制流图作为程序行为特征，把控制流图变换为字符串作为行为标签，并且用字符串编辑距离作为控制流图相似性判据，进行恶意程序的分类。陈凯等人[23]提出了用3D-CFG作为程序行为特征，并采用3D-CFG的“质心”（Centriod）距离作为行为相似性的判据，应用到程序克隆检测中。[24]利用Androguard工具获得程序中每个类方法的控制流图（Control Flow Graph, CFG），然后基于Cesare and Xiang提出的语法将CFG转化为字符串，利用TF-IDF方法挖掘每个恶意家族的代码结构分布，为每个恶意家族构造特征向量，最后通过计算未知家族样本的特征向量与每个恶意家族特征向量之间的余弦距离从而确定样本的所属家族。

1.2.3 基于数据流的污点分析方法

基于数据流的污点分析方法的基本思路是：首先对程序中的隐私数据进行污点标记，然后定义污点随程序语句不断传播的逻辑，并采用数据流分析法，分析污点最终是否传播到特定的信息输出端口（如文件、短信、网络等）来判断是否产生了信息泄露。由于Android程序普遍采用面向对象程序设计语言来开发，并使用消息通信进行组件交互，因此在进行污点分析时，还要结合组件间消息传递（以下简称ICC）、过程间数据流分析（Interprocedural dataflow analysis）、指向和别名分析（Points-to & Alias analysis）等分析方法。静态污点分析具有不同的分析精度，一般按照流敏感、上下文敏感、对象属性敏感和路径敏感等维度来划分[25]精度，分析精度越高、复杂度和开销也就越高，因此要在二者之间达到平衡。

静态污点分析最具代表性的工作有德国Darmstadt技术大学的Eric Bodden小组提出的FlowDroid[26]、美国MIT Michael I. Gordon等人提出DroidSafe[27]以及美国南佛罗里达大学Fengguo Wei等人提出的AmanDroid[28]。FlowDroid采用APKTool反编译器，把APK反编译成Jimple中间格式，并基于Soot[29]在iCFG（Interprocedural CFG）上应用IFDS算法进行数据流分析。Eric Bodden小组在SuSi[30]工作中，对约11万个Android SDK的类方法，采用监督机器学习算法进行了敏感数据源和汇的划分，认定了13类共18000多个敏感数据源类方法和8000多个数据汇类方法，这为FlowDroid的实际应用提供了基础。由于SuSi只认定了ADK中内建的敏感数据源和汇，实际上很多敏感数据（如用户名、密码、个人健康信息等）是通过用户UI输入的，为了认定这些通过UI界面传递的隐私信息，SUPOR[31]提出了一种对应用程序UI敏感性进行分析的静态方法，可以看作是对SuSi的有益补充。SUPOR主要对UI控件布局和属性进行分析，对于某些缺少属性信息的控件，通过自然语言处理技术分析控件描述的关键字，确定控件敏感性。

有些隐私信息泄露路径跨越多个构件边界，因此必须分析端到端的污点传播，而FlowDroid侧重于分析单个构件内部的污点传播，对于跨构件的污点传播支持较弱，DroidSafe、IccTA[32]和Amandroid对此进行了改进和增强。由于Android普遍使用隐式intent通信作为组件间传递数据流和控制流的手段，而隐式通信的目标组件通常是以字符串变量值的方式指定，因此跨组件的污点分析都必须很好的解决字符串变量值的静态分析这一基础性问题。DroidSafe采用JSA[33]解析出intent和uri中包含的字符串变量值的正则表达式，并通过一个映射表将源组件与目标组件关联起来，然后通过程序变换技术把组件间的间接通信模拟成组件直接调用，这样就可以用传统的过程间数据流分析方法解决跨组件的污点分析问题。IccTA的解决方法与DroidSafe类似，不同之处在于它使用了IC3[34]字符串静态分析工具。

静态污点分析的优点是：路径覆盖度较高、分析精准，而且由于是白盒方法，有利于分析信息泄露的行为模式和内部机理。基于这些优点，很多研究工作都以FlowDroid为基础进一步开展工作。静态污点分析的不足主要有：（1）对于经过加密和代码混淆的应用程序还无法处理，这是所有静态分析方法所存在的普遍性问题；（2）静态污点分析方法仅提供了恶意隐私信息泄露的“必要性”判据，但不是所有信息泄露都具有恶意性，有些应用程序把暴露部分隐私信息作为正常功能的一部分，因此还需要额外的人工分析进一步确认泄露是否具有恶意性；（3）由于对字符串分析支持较弱，对于ICC、反射、动态代码加载以及本地代码的分析不够精准。（4）有些静态污点分析工具的可扩展性不够好。如FlowDroid在使用过程中，对于结构较为复杂、规模较大的应用程序，污点分析的效率下降较快，经常出现溢出。

1.2.4 基于上下文的检测方法

恶意代码通常作为负载与程序的正常功能混杂在一起，如果对此不加辨识，分析就会过于保守，导致误报率较高。将程序的恶意行为与正常功能准确区分正是基于上下文的检测方法着力解决的问题。

基于上下文的检测方法的基本思路是：发掘恶意行为的典型特征和行为模式，通过对程序界面、内部结构的分析，提取应用程序的上下文要素，并利用启发式规则或机器学习方法判断行为的恶意性质。常见的上下文要素包括：恶意负载的触发方式、触发条件、权限、调用的敏感API及调用模式、程序中的字符串常量以及程序行为与它的意图之间的差异等。不同的研究工作选取的特征向量以及处理方式有所不同。

复旦大学和美国北卡大学谢涛团队提出的AppContext[35]和美国雪城大学Mu Zhang提出的DroidSIFT[36]、DESCRIBEME[37]是基于上下文的检测方法的典型研究成果。他们以敏感API作为主要特征，API的上下文包括四个要素：API之间的数据依赖关系、触发条件、调用入口点和特定API调用的常量参数等。根据程序控制流图，得到API上下文依赖图ADG，并采用图编辑距离作为ADG相似度的度量。对于一个待测应用程序，DroidSIFT将它的ADG与大量良性样本的ADG图数据库进行匹配，以区分良性或恶性，对于恶意程序，进一步采用多标签分类法认定出它的具体恶意家族。DESCRIBEME采用了另一个思路，它并不判断待测应用程序的恶意性，而是根据该程序的ADG，形成一个安全摘要，由用户自己根据摘要判断程序的安全性。人民大学梁彬等人提出了AsDroid[38]，将界面操作行为与界面意图之间的一致性作为判断恶意行为的依据。采用后向数据流分析认定界面入口函数的恶意性质，采用自然语言处理的方法认定界面的意图，然后判断二者的一致性。中科大曾凡平提出了RepassDroid[13]方法，以实际使用的权限和敏感API为特征，将这些API调用的入口点是否与用户界面事件相关作为判定恶意性的条件。显然这一方法是粗粒度的，但是具有简单、高效的优点；哈工大王兆国等人提出了DroidChain[39]，对恶意行为进行了分类（隐私信息泄露、短信诈骗、恶意安装和权限提升），针对每个类别给出了API调用模式，即API调用链，对于给定的应用程序，通过分析调用图来判断是否存在相应的行为模式。AppoScopy[40]把应用程序中的组件通信关系、方法调用关系以及污点数据传播关系表示为谓词，把恶意家族的行为特征表示为Datalog规则。在此基础上，把恶意行为检测问题转化为规则匹配问题。该工作的优点是能够判别程序具体属于哪一个恶意家族。

## 1.3 研究内容

本课题主要通过研究现有Android恶意程序的行为特征，Android恶意家族内部的程序结构特点以及行为模式，提出了一种基于多上下文特征Android恶意程序家族检测方法。通过分析，本文选取API，权限和系统广播作为原始特征信息，并与其发生的上下文结合形成程序特征，训练效果良好的分类器；然后根据恶意家族内程序的行为相似性构造家族特征向量，进而构造家族预测器；最后提取家族共性行为，生成程序存在的恶意行为的安全摘要。具体内容包括以下部分：

1. 基于图变换的Android应用程序特征提取：基于Soot生成的函数调用图（Call Graph, CG）和控制流图（Control Flow graph, CFG），构造基于回调函数的iCFG（Inter Cntrol Flow Graph），定义一系列图处理规则并应用于iCFG，将其简化成API图，在API图上进行程序特征提取。
2. 基于多上下文特征的Android应用程序恶意性判定：对从API图中提取的携带上下文信息的程序特征进行特征变换，并将变换后的特征嵌入到特征向量，采用随机森林，朴素贝叶斯等机器学习算法进行分类器训练；
3. 基于文本分析技术的恶意程序家族分类：从API图中提取以回调函数为单位的API调用序列，然后对API调用序列进行模糊处理，将处理后的序列作为行为块，研究每个行为块在不同恶意家族中的分布情况，将行为块作为家族特征，采用TF-IDF方法为每个恶意家族计算家族特征向量，最后构造1-NN家族预测器。
4. 安全摘要生成：提取同一家族所有应用程序共有的行为块集合，将该行为块集合作为该家族存在的行为特征。最后将行为块还原到API调用序列，手动分析序列中API的执行流程及其产生的恶意行为，并将分析结果作为安全摘要给出。

## 1.4 论文结构安排

本课题的论文章节安排如下：

第一章为绪论，通过介绍目前Android系统的普遍性和恶意程序数目的迅猛增长，从而引入本课题所具有的现实意义。并介绍了目前国内外的相关研究现状，对传统工作进行总结分析，从而引入本课题的主要研究内容。

第二章进行了相关知识介绍。首先对Android系统的体系结构，安全机制进行了详细阐述，这是我们进行研究的理论基础，并对本课题在研究过程中用到的相关工具和知识进行了展示。

第三章进行了基于图变换的Android应用程序特征提取。首先对本文所采用的上下文特征信息进行了详细介绍。然后构造了能够进行特征提取的API图，最后介绍了在图上进行特征提取的详细算法和过程。

第四章介绍了基于多上下文特征的Android应用程序恶意性判定。通过对第三章提取出来的程序特征进行特征变换，将其嵌入到特征向量，形成特征空间，然后采用多种机器学习算法进行分类器训练，并通过多组对比试验验证了该方法的有效性。

第五章介绍了基于文本分析技术的家族预测和安全摘要生成。首先从第三章构造的API图上进行API调用序列提取，对该序列进行模糊处理，将处理后的文本序列作为行为块，然后对恶意软件家族中行为块的分布进行挖掘分析，采用空间向量模型构造1-NN家族预测器。通过实验验证了该预测器的有效性。并且提取每个家族中所有应用程序共有的行为块，将其作为家族的恶意类别，然后对行为块进行人工分析，最终给出家族的安全摘要，将家族的安全摘要作为属于该家族的所有应用程序的安全摘要。

第六章对本课题的工作进行了总结，并提出了本课题仍存在的不足之处，介绍了未来的工作计划方向。

第 2 章 相关知识介绍

## 2.1 Android体系结构及安全机制

2.1.1 Android体系结构

Android系统是一款专门针对移动设备开发的操作系统，该系统自下而上共有的四层体系结构。首先，以Linux内核作为底层，借助Linux内核服务实现进程管理，网络协议等核心功能。其次，位于内核层之上的是核心类库和Android运行时库，其中核心类库提供了一些能够支持开发者使用各个组件的C/C++库，如SQLite数据库，Web浏览器引擎等。Android运行时库又包括核心库与Dalvik虚拟机，核心库主要提供了Java语言核心库的大多数功能，使得开发者可以使用Java语言来开发Android应用程序，Dalvik虚拟机提供了Android程序的运行环境。然后是应用框架层，这一层封装了一套实现核心功能的API，开发者在开发应用程序时可以通过调用特定API实现自己的特定功能，从而简化开发难度。最后，最顶层为应用程序层，该层即为手机上安装的所有应用程序集合，如手机自带的应用程序，用户在应用市场上下载的应用程序等都属于这一层，用户就是在应用程序层上完成对设备的操作。Android系统的四层结构分工明确，各层之间互相隔离，互不影响。具体的层次关系如图2.1所示。

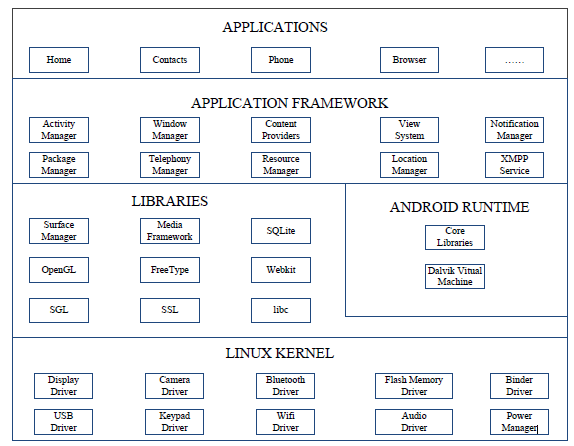


图2.1 Android系统架构

2.1.2 Android安全机制

由于Android系统构建于Linux内核之上，因此继承了Linux系统的一部分安全机制，与此同时，Android系统也提供了其独有的安全机制。本文主要阐述三种Android系统独有的安全机制。

（1）沙箱隔离机制

为了防止不可信代码对程序的恶意侵犯，Android提供了沙箱隔离机制，保证每个应用程序能够独立运行在一个相对安全的环境中。该机制的实现过程为：应用程序运行时，Android系统会为每个应用程序分配一个UID，用来唯一标识该应用程序，每个应用程序运行在自己的Dalvik虚拟机实例中，UID不同的应用程序之间相互隔离。如果需要两个应用程序之间共享数据，则需借助sharedUserID字段声明两个应用程序共用一个UID，则这两个应用程序即可运行于同一进程中。

（2）签名机制

Android应用程序在发布之前需进行签名，从而确保程序来源的可靠性。这就是Android系统的签名机制。签名机制的实现过程为：在开发者开发完应用程序后，首先对apk文件进行哈希计算并将结果存储在MANIFEST.MF文件中，然后用自己的私钥对该文件进行加密，将加密的结果保存在CERT.SF中，最终将证书，公钥存入META-INF目录下。在用户安装应用程序时，系统会对apk文件进行检查，使用公钥对CERT.SF进行解密，根据证书确认应用程序开发者的身份，然后对apk中的文件内容同样进行哈希计算，最后将计算结果与MANIFEST.MF文件中的内容进行对比，如相同则证明应用程序来源可靠。

（3）权限机制

Android系统通过内建的权限机制保护设备中的敏感资源和隐私数据不被非法访问，当应用程序需要访问用户的某些隐私数据时，系统会检查该应用程序是否拥有足够的权限，如果没有，则拒绝访问。

权限的本质为一个可以表示执行特定操作的能力的字符串。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，开发者可根据其需要在Manifest.xml文件中使用<uses-permission>标签进行权限的申请，但并不是开发者申请的权限系统都一定会授予，这与权限的保护级别有关。Android系统中权限的危险级别可分为以下三个等级：

* Normal级别：默认的权限保护级别，无需用户确认，只要开发者在Manifest.xml文件中声明了即可授权，该类权限一般不会对用户造成伤害。
* Dangerous级别：该级别的权限在授权前会弹出对话框，向用户显式请求该权限，由于该类权限可能会对用户造成伤害，所以用户需谨慎。
* Signature级别：该级别的权限危害性最高，也是管理最为严格的级别。该权限只会赋予给那些与声明权限使用了相同证书的应用程序。

Android 6.0之后的Android系统在权限申请上发生了变化。Android 6.0版本之前，应用程序所需的所有权限都需在Manifest.xml文件中进行配置，然后安装应用程序时，将应用申请的所有权限展示给用户，让用户授予该权限，否则安装失败。而Android 6.0之后，用户可以在程序运行时，根据应用程序的需要适当地授予权限，这种机制能够使用户更好地管理应用的权限。

## 2.2 Soot概述

2.2.1 Soot简介

Soot是麦吉尔大学的Sable研究小组所研究开发的一款Java字节码优化框架，它可以处理：Java字节码和源码、Android字节码、jimple中间格式代码和jasmin格式代码。处理后输出格式可以为Java字节码、Android字节码、jimple中间格式代码和jasmin格式代码。Soot的输入和输出格式是没有明确限制的，可以根据实际的需要自己设置，比如，可以从.dex字节码到.java代码、.java代码到.jimple中间代码输出等。Soot提供的主要分析类型有：生成调用图和控制流图、指向分析（Point-to）、过程内部数据流分析（Intro-procedural dataflow）、与FlowDroid工具协作完成的污点分析（Taint analysis）和过程间数据流分析（Inter-procedural dataflow）等。Soot的工作原理是：将应用程序转换为一种特定的中间表示，然后在这些中间表示形式上进行各种优化和分析。

2.2.2 Jimple中间表示

Jimple[41]是Soot中最重要的中间形式代码，目前很多的分析工作都是在Jimple基础上进行的。它是一种类型化的、三地址、基于语句的中间表示。在Java字节码到Jimple的转换过程中，为了简化程序操作，Jimple移除了Java规范jsr指令，并引入很多的局部变量。同时，Jimple还会在编译过程中移除多余的无效的代码。比如：去掉了从未使用过的变量和未使用的内存空间。

Jimple表示形式只有15种语句，因此只需要处理这15种语句就可以了。但是在Java字节码中需要处理200多种指令，因此，Jimple很大程度的降低了Java字节码处理的复杂度。Soot框架为Android静态分析工作提供了必要的基础架构支撑，而Flowdroid框架在Soot原有的基础上进行了拓展，使得支持更多的静态分析。比如：Flowdroid中添加了可以进行精确的数据流分析的框架Spark[42]。

2.2.3 Soot数据结构

为了支持代码分析工作，Soot提供了几种对象，分别是：Scene、SootClass、SootMethod、SootField、Body，其功能如下：

* Scene：Scence类代表整个分析变换工作的场景，可以获取一些过程调用（Inter-procedural）分析的信息，如call graphs和points-to information（指向性分析）。
* SootClass：表示Soot加载的待分析的类或者Soot创建的类。
* SootMethod：表示SootClass中的单个方法。  
  SootField：表示SootClass中的某个字段。
* Body：表示方法体，由Locals链（body.getLocals()）、Units链（body.getUnits()）、Traps链（body.getTraps()）组成。Locals链存储方法中的变量定义，Units链存储方法中的语句，Traps链存储方法中的异常处理语句。
* Stmt与Unit：它们都表示方法中的语句。Unit是一个接口（Interface），继承于Stmt接口。AssignStmt、IdentityStmt、IfStmt、ReturnVoidStmt、NopStmt等也继承于Stmt接口。它们的不同点在于：Unit表示由基本语句构成的复合语句，而AssignStmt等则表示赋值、定义、条件语句等基本语句类型。

## 2.3 函数调用图

Soot框架中提供了获取函数调用图CG（Call Graph）的API，将函数间的调用关系封装到了CallGraph数据结构中。CallGraph表示为一组方法调用边的集合，而边的数据结构表示为Edge=<Src，srcUnit，tgt，kind>，其中Src表示发生调用的方法，srcUnit表示发生调用的语句，tgt表示被调方法，即调用边的目标方法，kind表示调用类型，如虚拟调用，静态调用，接口调用等。

如使用Flowdrod框架中的测试apk文件进行生成CG的测试，生成CG的关键代码如代码2.1所示。

代码2.1 Soot工具生成CG

SetupApplication application = new SetupApplication(sJarPath, apk);

Scene.v().loadNecessaryClasses();

SootMethod mainMethod = application.getEntryPointCreator().createDummyMain();

Options.v().set\_main\_class(mainMethod.getSignature());

Scene.v().setEntryPoints(Collections.singletonList(mainMethod));

PackManager.v().runPacks();

CallGraph callGraph = Scene.v().getCallGraph();//获取函数调用图

visit(callGraph, mainMethod);//递归遍历CallGraph建立函数调用关系

exportMIG("flowdroidCFG",".\\flowdroidcg");//将生成的图片文件保存在固定目录下

对生成的CallGraph对象进行可视化，最后写入到gexf文件中。如图2.2为Soot为enrick1.apk生成的函数调用图。

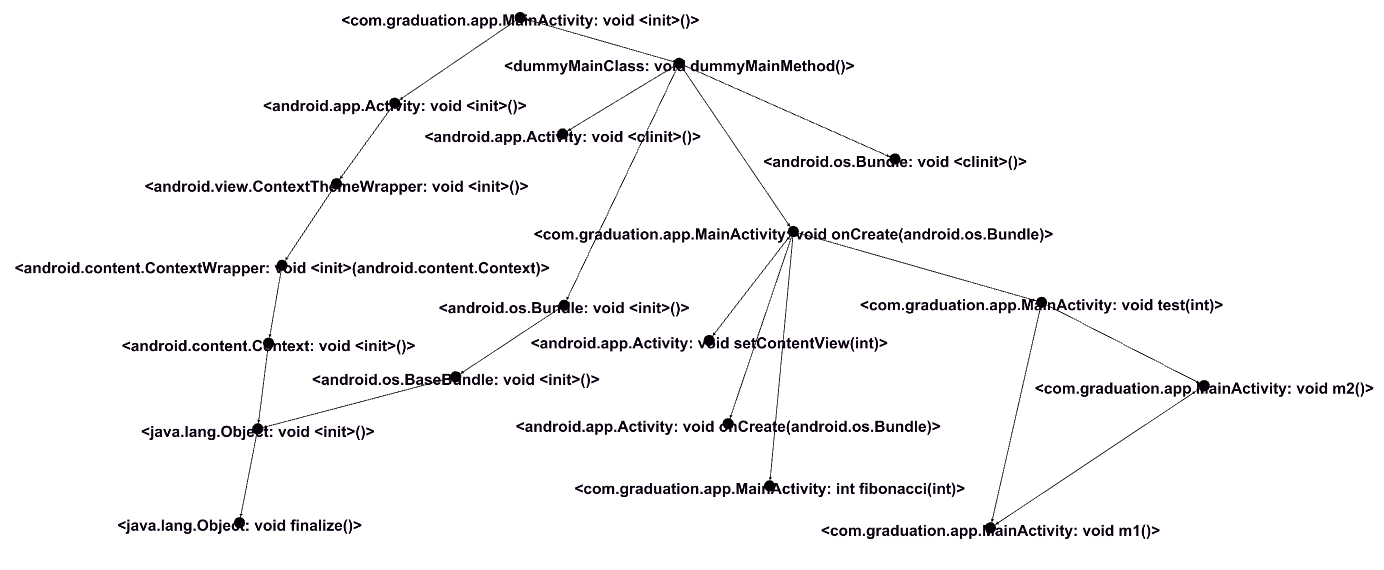


图2.2 enrich1.apk的函数调用图

## 2.4 过程内控制流图

过程内控制流图CFG（Control Flow Graph）是一个过程的抽象表示，决定了程序语句或指令的执行顺序，Soot为每个方法生成了一个CFG，一个CFG就是一个有向图。在Soot中，CFG的数据结构定义为DirectedGraph={Units, Map<Unit,List<Unit>> Preds, Map<Unit,List<Unit>> Succs, List<Unit> Heads,List<Unit> Tails}，其中，Units为节点集，方法中的每条语句对应图中的一个节点，节点的类型为Unit；Preds为节点集中每个节点对应的前驱节点的集合；Succs为节点集中每个节点对应的后继节点的集合；Heads和Tails分别为方法的入口点集合和出口点集合。Soot 2.4版本之后，拥有了专门将控制流图可视化的工具类CFGToDotGraph。如代码2.2为某程序入口Activity组件上某Button控件的响应事件onClick()方法的代码，其对应的控制流图如图2.3所示。

代码2.2 onClick()方法体

|  |
| --- |
| public void onClick(View v)  {  SmsManager sms=SmsManager.getDefault();  Test1.method1();  for(int i=0;i<3;i++)  {  Log.v("onClick", "onClick");  sms.sendTextMessage("+44 020 7321 0905",null,pwd,null,null);  }  } |

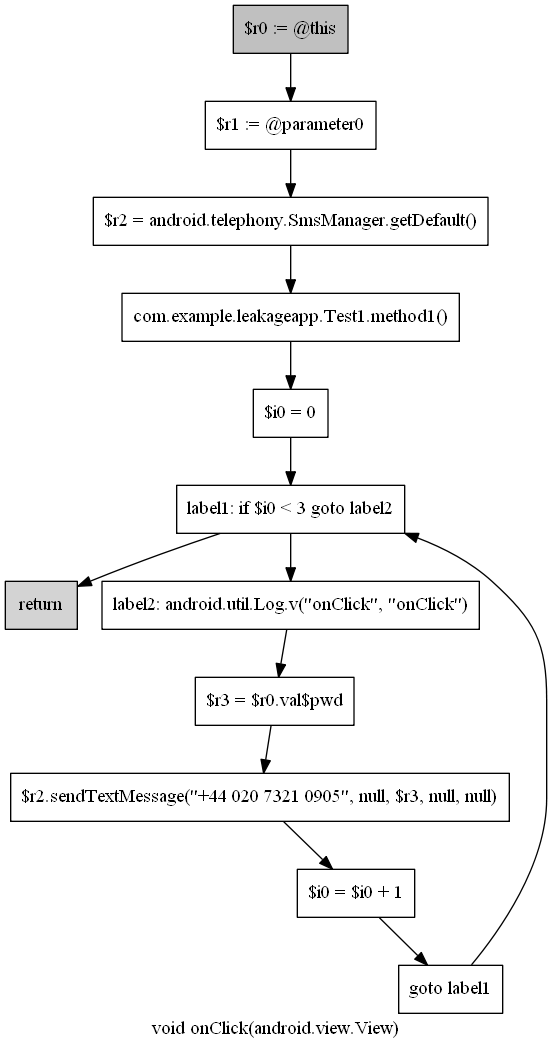


图2.3 onClick()方法对应的控制流图

在控制流图中，展示了一个方法内语句的所有可能的执行序列。节点的类型为Unit，通过查看Unit，我们可以获取每条语句使用到的变量和值，在该语句中定义的变量以及该语句是否为分支语句等。要为一个给定的方法创建CFG，只需将其对应的body对象传递给一个CFG的构造函数即可，如：UnitGraph g = new ExceptionalUnitGraph(body);其中ExceptionalUnitGraph为UnitGraph的子类，表示一种CFG的类型。

第 3 章 基于图变换的Android应用程序特征提取

## 3.1引言

传统的Android应用程序特征提取方法通过对apk文件进行反编译，获取程序的反编译文件smali文件并从中提取应用程序调用的敏感API集合，然后从AndroidManifest.xml文件中提取应用申请的权限集合，最后以此作为特征，对程序的恶意性进行判断。然而这是一种粗粒度的提取和分析方法，丢失了程序重要的语义信息。本章借助Soot框架提供的技术支持，构造既能保留特征信息，又能反映出特征所处上下文的API图，然后在该图上进行特征的提取。并且，本文在保留了API和权限特征的基础上新增了系统广播特征，该特征能够在一定程度上反映程序的恶意行为。

## 3.2 样例分析

本文以Drebin[4]恶意软件样本库中的Zitmo恶意家族作为贯穿本章的例子。Zitmo主要通过截获手机短信，获取设备信息，并通过网络将其发送到远程服务器，从而实现窃取用户隐私信息的目的。本文以Zitmo家族中的某恶意软件作为恶意软件样例，分析其实现恶意行为的工作方式。

代码3.1为Zitmo恶意软件样例对应的Manifest.xml文件的信息。从第4行可以看出，SmsReceiver组件监听了SMS\_RECEIVER广播。因此，当用户接收到短信时，系统会触发SMS\_RECEIVER广播并执行SmsReceiver. onReceive()方法，如图3.1右侧所示。在onReceive()方法中，首先获得存储短信消息的数组“puds”并存储在intent对象中传递给MainService组件，然后隐式启动后台服务MainService组件，执行onStartCommand()方法。在onStartCommand()方法中，先后获取了设备ID、SMS短信的来电地址、以及短信内容等隐私数据，最后调用execute(HttpUri Request, Response Handler)方法将该数据通过网络发送给远程服务器。由于获取短信消息、设备ID等这些信息的操作是在服务中完成的，因此用户意识不到这一系列操作的执行。

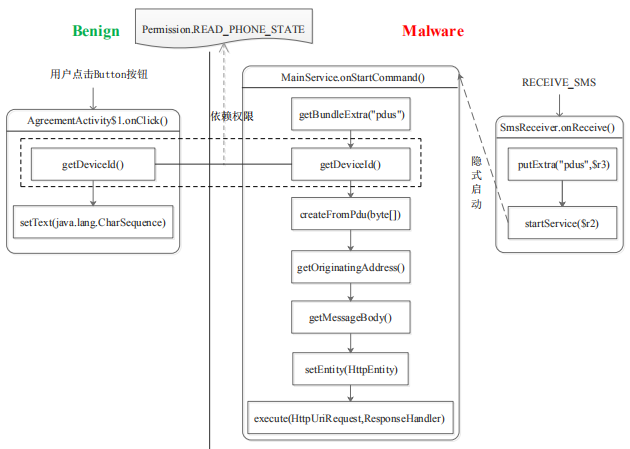
与此相反，良性样例在执行敏感操作时通常是在用户有意识的情况下进行。如图3.1左侧所示为一个良性样例对应的代码执行片段。在界面控件Button按钮的响应事件onClick()方法中，同样调用了getDeviceId()获取了设备的ID信息，然后通过setText()方法将该信息显示在了界面上，这说明该操作很有可能是由用户主动触发的，不含有恶意性。因此，检测应用程序恶意性时，不仅要关注敏感API特征，还要分析API发生的上下文。本文将API调用发生的上下文分为用户界面和后台回调，对于一个敏感API，如果其发生的上下文为后台调用，那么该调用很可能存在恶意性，反之则偏向于良性调用。

再来分析权限特征。图3.1所示getDeviceId()方法依赖的权限是READ\_PHONE\_STATE，在良性样例中，该权限的上下文为onClick()回调函数，onClick()是Activity界面上的控件，因此将READ\_PHONE\_STATE权限加入Activity组件的权限集合中。同样地在恶意样例中，将READ\_PHONE\_STATE权限加入Service组件的权限集合中，很明显恶意程序在Service组件中使用该权限访问敏感资源从而逃避用户的注意。而良性程序合理访问敏感资源，会在用户可控的情况下进行。因此，根据权限所在的组件能够为检测某些蓄意隐藏其恶意性的程序提供帮助。

并且，本恶意样例通过监听RECEIVE\_SMS系统广播，从而实现短信拦截功能，达到其恶意目的。因此，提取应用所监听的系统广播也能够对Android恶意应用的检测提供一定的帮助。

代码3.1 恶意样例Manifest.xml文件代码片段

|  |
| --- |
| 1 <receiver android:name=".SmsReceiver">  2 <intent-filter android:priority="10000">  3 <action android:name=  4 "android.provider.Telephony.SMS\_RECEIVED"/>  5 </intent-filter>  6 </receiver>  7 <service android:name=".MainService" /> |

**** 图3.1 恶意样例与良性样例部分CFG的对比

## 3.3 特征选择

以广义敏感API、敏感权限和敏感系统广播三类信息作为原始特征，并将这些原始特征与它们的上下文相结合，形成判断程序恶意性质的一个特征*F*。该特征可以表示为一个二元组：



其中，*feature*为任何一个原始特征，*context\_set*为它的所有上下文集合。下面分别介绍原始特征和其上下文。

3.3.1 原始特征

（1）广义敏感API

应用程序使用的API预示了应用程序将对特定资源进行的特定操作，因此选择API调用作为原始特征能够在一定程度上表征程序的行为。本文将所有对恶意行为检测有所贡献的API称为“广义敏感API”，包含以下四种类型API：

受权限保护的API：是指API的执行必须以特定的权限作为前置条件。这类API被认定为PScout[43]项目中给定的API列表。

* 与动态加载相关的API：很多恶意程序通过动态加载或反射调用恶意代码，从而规避某些静态检测工具的检测。本文将这些实现动态加载的函数也看作敏感API。包含这类API的类有DexFile，ClassLoader，Method和Field[13]。
* Source & Sink API：所谓Source API是指获取敏感数据信息的API，如android.app.Activity→findViewById(int)；而Sink API是指将数据写入文件或发送出移动终端的API，如android.os.Bundle→putChar(java.lang.String,char)。这类API预示着信息的获取和泄露，因此也将其作为敏感API。这类API被认定为SUSI[30]中的API列表。
* 其它恶意应用常用API：还有一些经常被恶意程序调用的API，如Crypto.Cipher中的方法用于代码混淆，System.loadLibrary()用于动态加载库等。

（2） 敏感权限

权限反映了应用程序能够访问的敏感资源和数据集合。通过对大量恶意样本进行分析，本文提取出了恶意程序中使用最频繁的20个权限作为敏感权限，并与其在良性程序中发生的次数进行对比，如图3.2所示。结果显示，敏感权限在良性程序与恶意程序中的分布存在明显差异，这意味着敏感权限能够表征恶意程序的某些行为。



图3.2 敏感权限在恶意样本和良性样本中的对比

（3）敏感系统广播

Android恶意程序通常依赖Android系统广播来触发或启动其恶意负载[44]，因此系统广播也可作为检测应用程序是否具有恶意性的重要依据。例如，BOOT\_COMPLETED是系统在开机加载完毕后由系统发送的广播事件，恶意程序通过监听该事件，并在其发生后触发恶意负载，启动后台服务。

文献[44]收集了Android恶意应用最感兴趣的25个系统广播，本文以此作为敏感系统广播，表3.1展示了部分敏感系统广播及其对应的系统状态信息。

表3.1 敏感系统广播及其系统状态信息

|  |  |
| --- | --- |
| 广播名称 | 系统状态信息 |
| BOOT\_COMPLETED | 开机完成 |
| NEW\_OUTGOING\_CALL | 拨打电话 |
| SMS\_RECEIVED | 收到短信 |
| CONNECTIVITY\_CHANGE | 网络连接状态变化 |

3.3.2 原始特征的上下文

恶意行为的发生不仅与上述原始特征有关，还与它们所处的上下文有着密切关系。文献[12]指出，很多情况下恶意程序和良性程序都要使用同样的权限和API，主要区别在于正良性程序通常在用户界面操作中使用这些特征，而恶意应用程序通常在用户无意识的情况下在后台使用这两类特征（如在Service生命周期回调函数中使用），完成某种恶意操作。因此，将原始特征与其上下文结合起来能够更好的表征程序的语义，从而提高检测的准确率，降低误报率。

本文将上下文分为三个层次：应用程序层、组件层和回调函数层（如图3.3所示），它们分别与上述原始特征，即敏感系统广播、敏感权限和广义敏感API相结合，共同组成程序的特征。



图3.3 上下文层次信息

敏感系统广播发生在应用程序层，因此它只与应用程序上下文相关联；敏感权限发生在上述三个层次中的每一层，如果将其与应用程序上下文相结合，那么粒度过大，不能反映应用程序中每个组件所使用的权限情况，而如果将其与回调函数相结合，粒度又过小，增加了处理的复杂度，因此我们将敏感权限与使用该权限的组件相结合，即把组件作为敏感权限的上下文；由于回调函数是Android程序执行的基本单位，因此将广义敏感API与调用它的回调函数相结合，能够反映各种事件（用户界面事件或系统事件）所触发的广义敏感API调用。

值得注意的是，一个原始特征对应的上下文可能不止一个，一个敏感权限可能在多个组件中被使用，一个广义敏感API也可能在多个回调函数中被调用。因此，一个原始特征对应了一个上下文集合。比如，回调函数OnReceive()和OnClick()都调用了敏感API getDeviceId()，那么getDeviceId()的上下文为集合{OnReceive(), OnClick()}，它们构成的特征为：

<getDeviceId(), {OnReceive(), OnClick()}>

## 3.4 特征获取

由于本文采用静态分析方法，所以通过对程序文本的分析来提取上述特征。

敏感系统广播及其上下文（即应用程序）可以通过Manifest.xml文件直接获取；敏感权限通常在Manifest.xml中定义，但是这里定义的是整个应用程序所使用的权限，并非每个组件所使用权限，因此必须通过程序分析得到每个敏感权限的组件上下文；广义敏感API及其上下文（即回调函数）也必须通过对程序文本的分析得到。下面重点介绍广义敏感API和敏感权限特征的提取。

3.4.1 特征获取过程

提取广义敏感API和敏感权限可以同时进行。首先，解析apk文件，获取能够反映程序中方法之间调用关系的调用图（Call Graph），并获取程序中的所有方法（包括回调函数）的控制流图（CFG）；然后把调用图和控制流图相结合，构造每个回调函数的iCFG；之后，再对iCFG进行约简，使它仅保留广义敏感API节点和图的出入口节点，这样就可以得到每个回调函数所调用的广义敏感API以及广义敏感API所在的上下文；最后，利用PScout给出的API与权限的映射表，得到每个回调函数所使用的权限集，并根据回调函数所属的组件得到一个组件所使用的敏感权限集，也就得到了敏感权限及其上下文。这一过程如图3.4所示。



图3.4 特征获取过程

3.4.2 回调函数iCFG的构造

在上述特征获取过程中，关键是得到每个回调函数的iCFG，并对它进行约简得到API图。iCFG是通过对程序调用图和每个类方法的控制流图进行综合形成的，下面给出调用图、控制流图的定义，以及iCFG的构造过程。

**定义3.1**（Call Graph）一个程序的调用图***CG***定义为一个有向图：



其中，

* *N*是程序中所有类方法的有限集{*m*1, *m*2, …, *mn*}；
* *S*是方法调用点的有限集；
* ****是带标记的有向边的有限集。每条边****表示方法*m*i在调用点*sk*处调用了方法*mj*；
* ****是***CG***的唯一入口点。

调用图仅描述了方法之间的调用关系，但不能描述一个方法内部的API调用过程。为此需要引入一个类方法的控制流图。

**定义3.2**（Control-Flow Graph）一个类方法*m*的控制流图***CFG***定义为一个有向图：



其中，

* *N*为一个类方法内程序语句的有限集{*s*1, *s*2,…, *sn*}。程序语句包括赋值、条件判断、函数调用、调用返回等；
* ****是边的有限集。每条边表示控制流从节点*ni*转移到节点*nj*；
* ****是唯一的入口节点；
* 是唯一的出口节点。

由于程序中的类方法可以在其方法体的任意位置调用其他类方法（也可以调用自身以形成递归调用），因此本文需构造一个图来描述方法调用和调用返回点的控制流转移，这种图称为过程间CFG，或简称为iCFG。iCFG是将多个单独的CFG集成在一起的超图。

给定一个回调函数*m*0，是回调函数*m*0调用的所有类方法的集合，是方法*mi* ()对应的控制流图，方法*mi*中的每个方法调用都会贡献两个节点：一个调用节点和一个返回节点。设和分别为***CFG****i*的调用节点集合和返回节点集合。

将***CFG****i*的边集合分为两个不相交的子集：，其中是普通控制流边的集合，表示控制流直接从一个节点转移到另一个节点，是调用返回边的集合，当且仅当节点*n*1是调用节点且*n*2是对应的返回节点。

**定义3.3**（Interprocedural CFG）一个回调函数*m*0的iCFG表示为一个超图：



其中，

* 为节点的有限集；
* ****是边的有限集。其中是所有普通控制流边的集合，是所有调用返回边的集合，是调用边及返回边的集合，当且仅当满足以下两种情况之一：（1）*n*1为方法调用点，*n*2为被调方法的入口点。（2）*n*1为方法*mi*的退出节点，*n*2为紧随*mi*调用之后的返回节点。对于一个调用边和一个返回边，如果*i*=*j*，则这两条调用边和返回边相互对应，且边；
* ****是回调函数*m*0的入口节点；
* 是回调函数*m*0的出口节点。

为了简单起见，本文使用*Call*，*Return*，*Head*，*Tail*分别代表iCFG中所有的调用节点集合，返回节点集合，入口节点集合，出口节点集合，并将其统称为*StructuralNodes*集合。











3.4.3 iCFG的约简

由于本文所构造的基于回调函数的iCFG用于特征提取，而我们只关注iCFG中与安全安全敏感行为相关的API调用，也就是说，我们可通过删除与特征无关的无用节点和边从而将原始的iCFG约简为简化版本，在介绍约简规则之前，我们介绍以下相关术语。

给定一个通用的有向图，其中*N*为图中所有节点的集合，*E*为边的集合，设图*G*中从节点*u*到节点*v*的路径表示为，该路径表示为一个节点序列：



其中，因此，我们称节点*u*经过路径可到达节点*v*，即节点*u*到节点*v*的路径经过了节点。

假设是*N*的子集，对于图*G*中的一条路径，如果且对于，则称从节点*u*到节点*v*的该条路径遍历了集合。图3给出了一个样例，设集合，在该样例中，总共存在3条，其中，，，并且只有路径*p*1遍历了集合，而路径*p*2，*p*3没有遍历该集合。

**定义3.4**（iCFG的约简）设为回调函数*m*0对应的iCFG，则对于图*g*的约简就是将一条约简规则**T**应用于图*g*，从而生成新的iCFG，即，该过程表示为，使得：

* ****，是包含了中所有的*structural* 节点以及敏感API调用的节点集合；
* ****
* ****；
* ****；

很显然，该约简规则**T**满足以下两个条件：

1）关键节点得以保留：*g*中的所有敏感API调用节点和S*tructuralNodes*都保留在了中；

2）可达性得以保留：对于任意节点，如果从*g*中的*n*1可到达*n*2，则从中的*n*1也可到达*n*2。

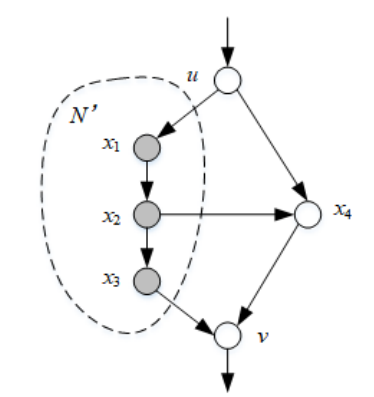


图3.5 遍历的样例

简单起见，本文将约简后的iCFG称作API图，算法3.1给出了将iCFG约简成API图的具体实现。首先给出以下概念。

给定一个有向图，对于任意节点，本文使用和分别表示节点*n*的前驱节点集合和后继节点集合，然后使用表示以*n*为起点或终点的边的集合：







|  |
| --- |
| 算法3.1 iCFG的约简算法 |
| **输入**：一个iCFG  **输出**：一个API图  1  2 for each node  3 for each  4 for each  5 将边添加到中；  6 从图*g*移除节点*n*和中的所有边；  7 end  8 end  9 end |

在算法3.1中，第1行首先定义了为*StructuralNodes*与广义敏感API的并集，然后求出与的差集，即为待删除的无用节点集合，并在第2行对无用节点进行遍历，第3-8行求出每个无用节点的前驱和后继节点集合，在每个前驱与后继之间建立新边，并添加到中，删除无用节点和与无用节点相关的边，最终将iCFG约简成API图。

为了说明iCFG的约简变换，本文给出一个实际的例子。代码3.2展示了回调函数onClik()的代码片段，该代码片段同时调用method1()和一些安全敏感行为相关的API。图3.5展示了如何将onClick()的iCFG约简为相应的API图。

代码3.2 onClick()及相关方法对应的方法体

|  |
| --- |
| **onClick()：**  0 $r0:=@this  1 com.example.leakageapp.Test1.method1()  2 $r2=android.telephony.SmsManager.getDefault()  3 $i0=0  4 label1:if $i0<3 goto label2  5 $r2.sendTextMessage("+4402073210905",null,” important message”,null,null);  6 $i0=$i0+1  7 goto label1  8 return  **method1：**  9 $r0:=@this  10 $r2=new android.telephony.TelephonyManager;  11 $r3=$r2.getDeviceId();  12 $r4=$r2.getSubScriberId();  13 $r5=$r3+$r4;  14 return $r5; |

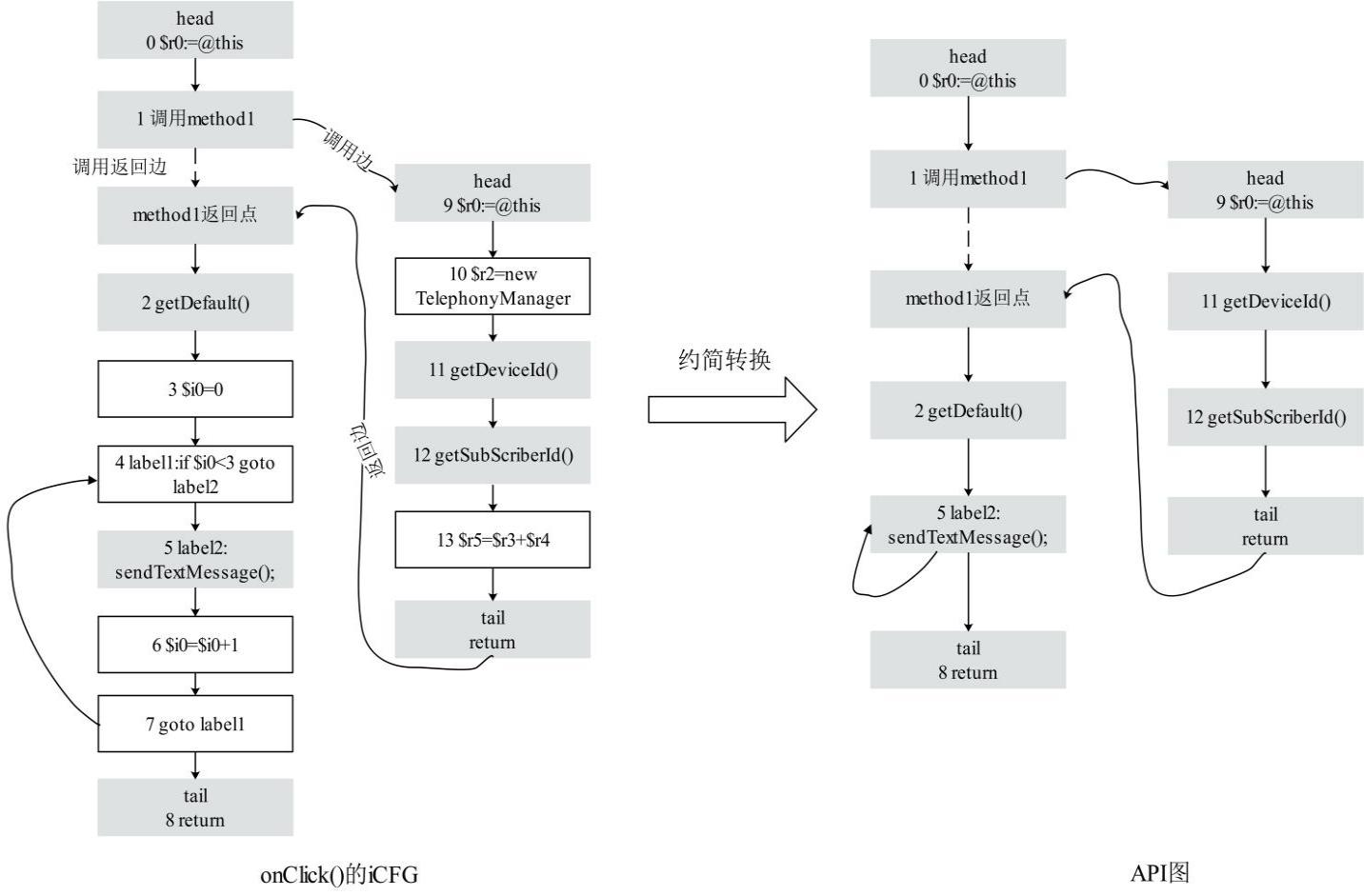


图3.5 onClick()的iCFG的变换过程

3.4.4获取广义敏感API和敏感权限特征

算法3.2给出了基于API图的特征提取过程，最终得到包含上下文信息的广义敏感API和敏感权限特征集合。该过程分为两个关键部分：1）首先在第2行遍历API图，得到API调用集合。然后在第3-5行提取包含上下文信息的广义敏感API特征，其中第3行遍历API图对应的API集合，第4行将API的上下文存储在其对应的<context>集合中；2）第6-12行是包含组件信息的敏感权限提取，首先在第8行利用PScout提供的API-权限映射表获取API对应的权限，然后在第9行获得回调函数所属的组件，并在第10­­行将组件信息加入相应权限特征的<context>集合中。

|  |
| --- |
| 算法3.2 特征提取算法 |
| **输入**：API-graphs=<callback,API图> //每个回调函数对应的API图  API-Permission //PSCout提供的API-敏感权限映射表  **输出**：<API，context\_set>集合  <permission，context\_set>集合  1 for each callback in API-graphs.keySet()  //遍历API图，获取敏感API集合  2 nodes=*dfn*(API-graphs.get(callback));  3 for each node in nodes except head and tail  4 将callback加入到node对应的API的context\_set中;  5 end  6 for each node in nodes  7 if(node是受敏感权限保护的API)  8 permission=API-Permission.get(node);  9 get component=callback所属组件;  10 将component加入到permission的context\_set中;  11 end  12 end  13 end |

3.4.5 获取敏感系统广播特征

由于系统广播的注册在Manifest.xml文件中进行，因此可直接从Manifest.xml文件中提取。通过遍历Manifest.xml文件中每一个广播组件的intent-filter标签，提取<action android:name>子标签的内容获取系统所监听的广播，从而提取出敏感系统广播特征。

第 4 章 基于多上下文特征的Android应用程序恶意性判定

## 4.1 引言

本章基于第三章提取出来的程序特征，对该类特征进行特征变换，将其嵌入到特征向量，形成特征空间。最后以特征向量作为训练数据，采用随机森林，朴素贝叶斯等机器学习算法进行训练，形成能够对未知恶意性的应用程序进行预判的分类器。并通过实验验证了该分类器的有效性。

## 4.2 特征变换

为了便于采用机器学习方法进行程序行为识别，需要将提取出的特征编码为特征向量，并使用一个特征向量来表示一个程序样本。

4.2.1 特征向量定义

将特征向量定义为一个包含1197个元素的数组（如图4.1所示）。特征向量中每个元素的序号*i*（1≤*i*≤1197）对应一个固定的原始特征，该元素的值Vector[*i*]表示由这个原始特征构成的程序特征的量化值。比如，特征向量中的1号元素对应原始特征com.android.phone.HapticFeedback→vibrate函数，而Vector[1]，即*a*1表示由该特征经变换后形成的值。

特征向量中共包含1152个广义敏感API特征值（权限相关API共814个，动态加载相关API共14个，Source&Sink API共297个，恶意应用常用API共27个）、20个权限特征值（如图3.2所示）和25个系统广播特征值（表3.1给出了部分系统广播事件）。



图4.1 特征向量的构成

设*i*对应的原始特征为，它构成的特征为，则Vector[*i*]的取值由函数计算得出，即：

Vector[*i*]=

其中， *f*为特征变换函数，它根据原始特征的类别有不同的定义方式。我们将一个特征经过*f*变换后得到的值，称为该特征的**特征值**。特征值的计算如下面4.2.2节介绍。

4.2.2 特征值的计算

（1）广义敏感API的特征值

根据敏感API的上下文计算其特征值，将其上下文分为UI相关和无关两类，其特征变换函数定义如下：

**定义4.1** 设*api*为一个广义敏感API，其构成的程序特征为<*api*, *context\_set*>，则其特征值通过如下特征变换函数求得：

*f*(<*api*, *context\_set*>)=

其中，

* UI和Non\_UI表示集合*context\_set*中与界面相关和无关的回调函数子集，且满足。|UI|和|Non\_UI|分别表示两个集合的元素个数。
* *w*1i（）表示回调函数的权值。其中*w*11为界面相关回调函数的权值，*w*12为界面无关回调函数的权值。

通过统计方法计算*w*1i的值。以300个恶意程序和300个良性程序作为样本，分别统计在界面相关和界面不相关的回调函数中广义敏感API的平均调用频次，统计结果如表4.1所示。

表4.1 以回调函数界面相关性为粒度的API出现平均频次

|  |  |  |
| --- | --- | --- |
| 回调函数的UI相关性 | 良性程序/（个） | 恶意程序/（个） |
| UI相关 | 18.5 | 13.6 |
| UI不相关 | 9.3 | 29.6 |

根据API特征在不同上下文中出现频次的差异性为其设置权值。差异性计算方法如公式（4.1）所示：

** （4.1）

其中*C*m(*p*)为恶意程序中上下文为*p*的API特征出现的频次，*C*n(*p*) 为良性程序中上下文为*p*的API特征出现的频次。

表4.2给出了每种上下文中API对应的*φ*(*p*)的值，为了方便处理，对*φ*(*p*)值进行归一化处理，处理后的结果为。将作为每类上下文中API特征对应的权值。

表4.2 不同API上下文对应的*φ*(*p*)值

|  |  |  |
| --- | --- | --- |
| 上下文 | UI | Non\_UI |
| *φ*(*p*) | 9.9 | 94.2 |
|  | 0.1 | 0.9 |

（2）敏感权限的特征值

根据敏感权限的上下文计算其特征值，本文将敏感权限的上下文（即组件）按照组件类型分为四类：Activity、Service、Receiver和Provider，特征变换函数定义如下：

**定义4.2** 设*permission*为一个敏感权限，其构成的程序特征为<*permission*, *context\_set*>，则其特征值通过如下特征变换函数求得：



其中，

* *w*2i（）为每类组件类型的权值；
* setActivity、setService、setReceiver和setProvider为*context\_set*中四类组件构成的互斥子集，|set|表示集合中元素的个数。

同样，*w*2i通过统计方法求得。在恶意样本和良性样本中以组件为粒度对图3.2中20个敏感权限出现的平均频次做了统计，统计结果如表4.3所示。利用公式（4.1）计算差异值*φ*(*p*)，并进行归一化处理得到，如表4.4所示。同样将作为每类组件中权限特征对应的权值。

表4.3 以组件为粒度的权限平均频次

|  |  |  |
| --- | --- | --- |
| 组件 | 良性程序 /（个） | 恶意程序/（个） |
| Activity | 4.142 | 6.147 |
| Service | 0.204 | 1.879 |
| Receiver | 0.283 | 2.954 |
| Provider | 0.042 | 0.290 |

表4.4 组件对应的*φ*(*p*)值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 组件 | Activity | Service | Receiver | Provider |
| *φ*(*p*) | 9 | 17 | 30 | 2 |
|  | 0.15 | 0.30 | 0.52 | 0.03 |

（3）敏感系统广播的特征值

由于敏感系统广播的上下文为应用程序，本文对该类特征赋予的权值为0或1。即在一个待检测的应用程序中，若某敏感系统广播被该应用程序注册，则为其赋予权值1，反之则为0。

综合以上每类特征值的求解方法，对于图4.1所示的特征向量Vector，如果不存在某特征，则特征向量的相应特征元素值为0，否则，特征元素值的计算公式（4.2）为：  （4.2）

以恶意程序Opfake为例，其特征向量为：

Vector={0,...1.8,...2,...0.1,...0.9,....2.7,...2.7,...3.6,...1.8,...0.9,...0.15,..0.3,...0.52,...1...0}

省略号部分的元素值都为0。

## 4.3 机器学习算法

本文拟采用朴素贝叶斯、K邻近算法（K-NearestNeighbor，KNN）、随机森林（Random Forest，RF）、逻辑回归（Logistic Regression，LR）、支持向量机（Support Vector Machines，SVM）等机器学习算法进行分类器训练。下面着重对随机森林、朴素贝叶斯、SVM这三种算法原理进行详细介绍。

4.3.1 随机森林算法

随机森林是一种以决策树为基本分类器的集成学习模型，该算法包含了由Bagging集成学习技术训练的多个决策树，待分类样本的分类结果由多个决策树通过某种结合策略决定。该算法的执行步骤如下：

输入：训练集，测试样本***x***

输出：样本***x***所属的类别

（1）从原始的训练集通过有放回的采样方式随机选择*N*个样本，构造采样集，如此执行*T*次，得到*T*个采样集；

（2）采用*T*个采样集训练*T*个决策树，在训练每个决策树时，从*n*个样本特征中随机选取*m*（*m*<*n*）个特征，然后从*m*个特征中选取分类效果最佳的特征进行左右子树的划分。最终获得*T*个 CART（Classification and Regression tree）决策树。

（3）输入样本***x***，该样本的分类结果由T个决策树的输出结果综合决定。

随机森林算法采用CART决策树作为弱分类器，使用基尼系数代替熵，从而最大程度地最小化不纯度，基尼系数公式为：



其中*D*是数据集，*k*是样本类别，基尼系数的意义是从数据集中随机选择两个样本，其类别不一致的可能性。

对于分类任务，随机森林算法中最常用结合策略是投票法，设为随机森林分类器，则其在样本上的预测输出表示为N维向量，其中是分类器在类别上的输出。



即如果某类别支持率超过50%，则可以将样本预测为该类别，否则该预测将被拒绝。

4.3.2 朴素贝叶斯算法

朴素贝叶斯算法是一种基于贝叶斯理论和特征条件独立假设的分类方法。对于给定的训练数据集，首先基于特征条件独立的假设来学习输入/输出的联合概率分布，然后基于该模型，对于给定的输入***x***，使用贝叶斯定理找到后验概率最大的输出*y*，该算法的执行步骤如下：

（1）输入样本集和测试样本*x*，样本集表示为：



其中*m*为样本集中的样本个数，*n*为特征数。表示样本类别，样本的类别集合为

（2）计算先验概率：计算样本类别的个数*K*，对于每一个样本Y=C*k*，计算总样本中样本类别为C*k*的概率。

（3）计算条件概率：首先将样本集划分为*K*个子集，然后分别对属于C*k*类别的子样本集进行计算，计算特征的概率，即，其为该子集中特征值为*ajl*的样本数与该子集样本数的比值。

（4）对于待测样本*x*，计算其对于每一个样本类别C*k*的后验概率：，概率值最大的类别C*k*即为样本的预测类别。

4.3.3 支持向量机

支持向量机的原理是找出能够满足要求的最佳分类超平面，并在确保分类性能的同时最大化平面两侧的空白空间。此超平面的方程公式为：



其最佳权值和最佳偏置计算公式为：



其中，为支持向量，。本文所采用的核方法为径向基核函数（RBF），其对应的解析式为：



相当于核函数将之前的分类函数映射成：

|  |  |
| --- | --- |
|  |  |

其中，为支持向量，为未知的输入向量；

## 4.4 方法实现与实验评估

4.4.1 方法实现

本文方法的实现基于Soot和Sklearn库。在本文中，我们仅使用Soot为每个实现的类方法生成调用图和控制流程图。Sklearn库是基于python的第三方机器学习库，它通过提供各种有效的机器学习算法来帮助我们训练恶意程序分类器。实现过程如图4.2所示。

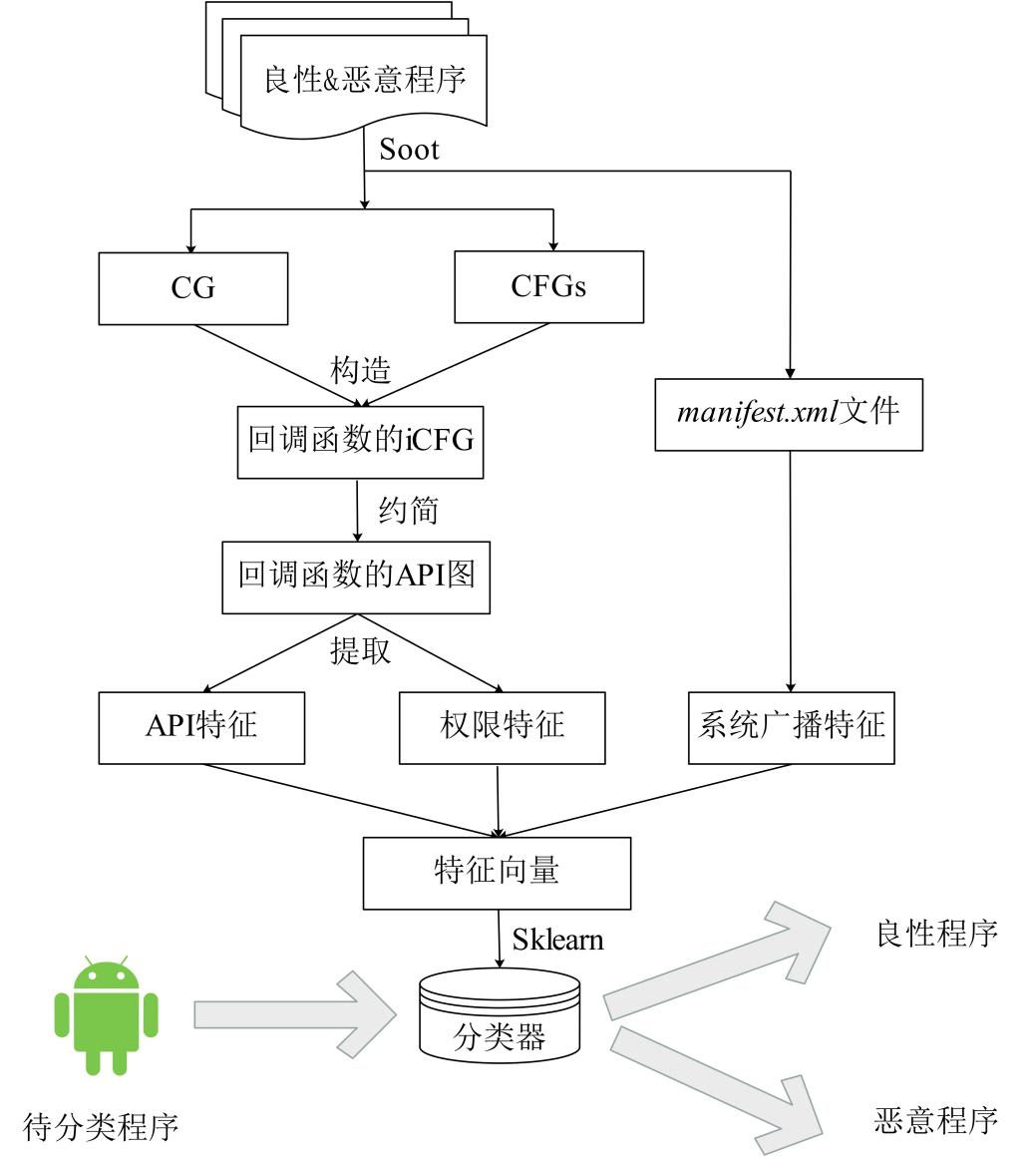


图4.2 方法实现

4.4.1 数据集和度量指标

恶意样本集分别选自于Drebin提供的样本库、Virus Share样本库、FlowDroid提供的DroidBench测试集。为了使样本数据更具代表性，所选恶意样本集覆盖了每个样本库中的所有恶意软件家族。由于Soot在处理apk时，其时间效率会随着apk的增大而骤减，所以在选取样本时，考虑到时间成本，我们对应用程序的大小也做了必要的限制。良性样本来自于Android应用官方市场Google Play，其大小与恶意样本的大小范围保持一致，并预先通过电脑管家和360安全卫士进行检测，以确保它们是良性程序。表4.1给出了本文采用的样本集的属性信息。最后将样本集的75%作为训练集，25%作为测试集。每个样本的处理时间不超过3分钟。

表4.1 样本集的属性信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 程序类别 | 样本来源 | 样本数（个） | 平均大小(MB) | 家族数（个） |
| 恶意程序 | Drebin | 358 | 1.4 | 179 |
| virusShare | 160 | 1.7 | 80 |
| FlowDroid | 42 | 1.8 | 未分类 |
| 良心程序 | googleplay | 312 | 1.7 | 26 |

为了评价算法的有效性，我们采用混淆矩阵来表示常用的度量指标，如表4.2所示。

表4.2 混淆矩阵

|  |  |  |
| --- | --- | --- |
| 真实情况 | 预测结果 | |
| 恶意程序 | 良性程序 |
| 恶意程序 | *TP*（真正例） | *FN*（假反例） |
| 良性程序 | *FP*（假正例） | *TN*（真反例） |

根据以上参数，用如下几个指标来评估本文算法的有效性。

误报率定义为：

 （4.3）

即良性程序被检测为恶意程序的个数占良性程序总数的百分比。

准确率定义为：

 （4.4）

即样本被正确分类的个数占总的样本的百分比。

精确率定义为：

 （4.5）

即恶意程序被检测为恶意程序占所有被检测为恶意程序的百分比。

召回率定义为：

 （4.6）

即恶意程序被检测为恶意程序的个数占恶意程序总数的百分比。

4.4.2 约简转换的有效性验证

约简转换的主要目的是提高特征提取阶段的时间效率，本实验旨在验证将iCFG约简后生成API图的必要性和有效性。

（1） iCFG与API图的规模对比

本文在生成iCFG后，对iCFG进行了无用节点的过滤，生成了API图。由于API图只保留了我们所关注的广义敏感API调用节点，删除了无用节点，因此，图的规模被大大缩减，算法效率得到了很大程度上的提高。图4.3对iCFG与API图的规模进行了比较。本文以图的节点总数和边总数两个元素进行对比，从图中可以看出，经过过滤生成的API图的节点数和边数比原始iCFG小得多。通过计算可知，本文对于图的压缩率可达：

*P*comp==75.4%



图4.3 iCFG与API图规模的比较

（2） 约简转换的有效性

在前面的实验中，我们对约简前后的图规模进行了对比，从而验证了对iCFG进行约简的必要性。为了说明对图的约简确实提高了本文特征提取算法的时间效率，我们进行了对比实验，并将结果展示在图4.4。从图中可以看出，不论对于多大的apk，在经过约简的API图上进行特征提取的时间效率都要高于在iCFG上进行特征提取。其平均时间效率可提高：

=26.3%



图4.4 算法时间效率对比

4.4.3 基于多种分类算法的分类器训练

将本文的特征集应用于朴素贝叶斯（Naive Bayes，NB）、K邻近算法（K-NearestNeighbor，KNN）、随机森林（Random Forest，RF）、逻辑回归（Logistic Regression，LR）、梯度提升树（Gradient Boosting Decison Tree，GDBT）这5种分类器进行训练并测试，每类算法的检测效果如表4.3所示。

表4.3 不同机器学习算法的检测效果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 分类器 | NB | KNN | RF | LR | GBDT |
| *Accuracy* | 89.2% | 93.8% | 95.4% | 93.1% | 88.4% |
| *FPR* | 12.5% | 7.9% | 5.6% | 9.1% | 13.1% |
| *Recall* | 90.7% | 95.2% | 96.5% | 95% | 89.7% |
| *Precision* | 89.2% | 93.2% | 95.1% | 92.3% | 88.7% |

从表4.3中可以看出，本文的特征集在RF分类算法上的表现效果最好，获得了96.5%的召回率、95.4%的准确率，误报率仅为5.6%。而在GBDT算法上的表现效果最差。

接下来来分析一下为什么RF算法的分类性能最好。我们知道，在构建决策树时，随机森林算法首先从所有特征中随机选择若干特征，然后逐步从若干特征子集中选择最优属性进行划分，因此在处理高维数据时表现良好。此外，RF采用了集成学习方法的思想，并结合多个基本分类器的结果来确定分类。因此，RF具有较好的分类性能和泛化能力。

为了评估本文所选取的特征集的有效性，本文从以下三方面做了对比实验。

（1）广义敏感API特征的有效性

为了验证本文自定义的广义敏感API合乎逻辑，将敏感API分为A1、A2、A3、A4四部分。其中A1表示受权限保护的API集合，A2表示Sources&sinks函数集合，A3表示动态加载相关函数集合，A4表示恶意应用常用函数集合。广义敏感API集合G=A1∪A2∪A3∪A4。然后分别以G、A1、A1∪A2、A1∪A2∪A3这四个API集合作为API特征，权限特征和系统广播特征不变。然后将新的三类特征集应用于随机森林算法。每种特征集构造的分类器的准确率、精确率以及召回率如图4.5所示。图示结果表明本文所定义的广义敏感API是合理的，能有效提高分类效果。



图4.5 不同API特征集在RF下的检测效果

（2）基于组件上下文的权限特征的有效性

为了验证申请权限集合、基于应用程序粒度的实际使用权限集合和基于组件粒度的实际使用权限集合这三类特征集对检测效果的影响，我们使用Android Asset Packaging Tool（AAPT）工具从Manifest.xml文件中提取应用申请的权限集合，记为Request-Permission，然后基于应用程序粒度，提取整个应用使用的权限集合，记为Coarse-Permission，将基于组件层次的实际使用权限集合记为Fine-Permission，广义敏感API特征和系统广播特征不变，将新的两类特征集应用于随机森林算法，其构造的分类器的准确率、召回率以及精确率如图4.6所示。图示结果表明从源码中提取的权限集合要比直接从Manifest.xml文件提取权限信息要准确得多，而且对权限集合进行的细粒度划分也能有效提高分类器的性能。



图4.6 不同权限集合在RF下的检测效果

（3）系统广播事件特征的有效性

为了验证系统广播事件对于Android恶意软件检测的有效性，我们将不包含系统广播事件特征的特征集应用于随机森林算法，与原实验结果进行对比，对比结果如图4.7所示。图示结果表明，包含系统广播特征的特征集的效果明显更好，因此验证了该特征的有效性。



图4.7 系统广播事件特征的有效性验证

为了验证本文方法的有效性，将本文提出的检测模型与近5年的典型相关工作进行对比，结果如表4.4所示。从表4.4中可以看出，本文方法的准确率好于大部分其他工作，仅次于DroidMat。虽然DroidMat的准确率最高，但其召回率较低，本文方法在保持良好准确率的同时，召回率比DroidMat提高了9.1%，证明了本文方法的有效性。

表4.4 与现有的其他工作对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 检测工具 | 数据集 | | 特征集 | 准确率 | 召回率 |
| 良性样本 | 恶意样本 |
| DroidMat[12] | 1500 | 238 | 组件、系统调用、函数调用等特征 | 97.8% | 87.4% |
| Drebin[4] | 123453 | 5560 | 组件、权限、intent过滤器、敏感API、网络地址等特征 | 93.9% | 95.9% |
| Androdect[6] | 2000 | 1126 | 组件、函数调用和系统调用 | 94.2% | 87.4% |
| 文献[3] | 100 | 500 | API、权限 | 90.0%左右 | 85.0%左右 |
| 本文方法 | 2600 | 2130 | 权限、广义敏感API和系统广播 | 95.4% | 96.5% |

VirusTotal是一个免费的恶意软件检测网站，共收集了69种恶意软件检测工具，我们将样本集集应用于该网站，采用这些工具对样本进行分类。检测结果表明，本文所以构造的分类器的检测效果比大部分VirusTotal中的工具效果好。图4.8展示其中6种检测工具与本文分类器的对比效果。



图4.8 与现有的主流工具对比

第 5 章 基于文本分析技术的家族预测和安全摘要生成

## 5.1 引言

Android恶意程序家族具有衍生性，即同一家族中的恶意程序在行为上具有一定的相似性。本章首先以文本分析技术为手段，以行为块为基本分析对象，通过分析统计同一恶意软件家族中各程序行为块的相关性，不同恶意软件家族中程序行为块之间的差异性，进而以行为块为特征，为每个家族训练出能够表征家族的特征向量，然后采用特征向量进行1-NN（1-Nearest Neighbor）分类器构造，并从恶意家族中提取家族内程序的共同行为块，对其进行人工分析，为每个家族生成安全摘要。最终采用1-NN分类器对第四章判定为恶意，但未知家族的程序进一步进行家族预测，然后根据样本的所属家族给出样本的安全摘要。其工作流程图如图5.1所示。

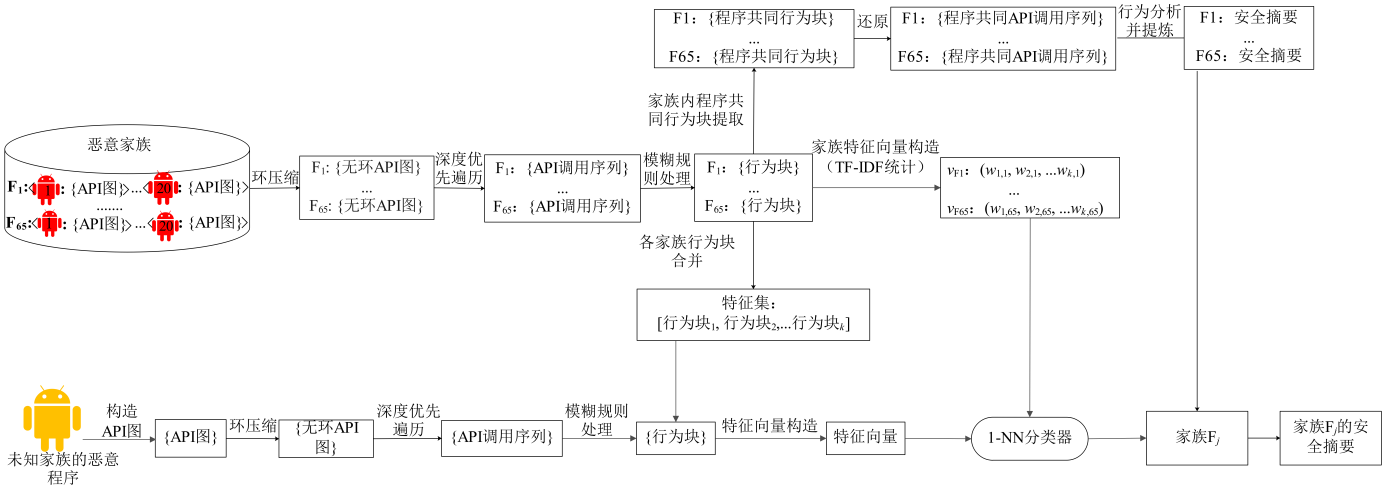


图5.1 工作流程图

从图5.1中可以看出，本章工作基于第3章构造的API图。由于Android应用程序由Java语言编写，而Java语言中的for循环，while循环等语法结构会造成API图中存在大量的环，因此，首先对恶意软件家族库中所有应用程序的API图进行环压缩处理，然后对去环后的API图进行API调用序列提取，进而对提取出的API调用序列采用模糊规则处理，生成行为块。接下来的工作分为两部分。1）家族预测：将所有家族中的行为块合并形成特征集合，并训练家族特征向量。在进行未知家族恶意样本的家族预测时，同样需对待测样本做类似于训练样本的处理，构造特征向量，最后采用1-NN分类器进行家族预测。2）安全摘要生成：提取家族中应用程序的共同行为块，将其作为家族产生的恶意类别，然后将行为块还原到API调用序列，并对API调用序列进行人工分析并提炼，为每个家族构造一个安全摘要。最终根据1）预测的恶意程序的所属家族给出恶意程序的安全摘要。

## 5.2 API图环压缩处理

5.2.1 环的识别

不同于一般图中的环，API图中的环通常称作natural loop[45]（自然循环），它是一个具有唯一入口的强连通子图，而且从循环外进入循环内，必须首先经过循环的入口节点，即入口节点支配（dominate）循环中的每一个节点。为了识别API图中的natural loop，先给出以下几个定义。

**定义5.1 (*paths*)** API调用路径。API图中任意两节点*n*0，*nk*之间的路径集合*paths*定义为：

 （5.1）

其中为API图中的边集合，为API图的节点集合。

**定义5.2 (*dom*)** 支配节点。在API图中，对于节点*n*，其支配者节点集合*dom*(*n*)定义为：

 （5.2）

其中代表API图中节点*n*的所有前驱节点集合。

**定义5.3 (*back edge*)** 回边。API图中的回边*back edges*定义为：

 （5.3）

根据以上定义，下面给出natural loop的定义。

**定义5.4 (*natural loop*)** 自然循环。设API图中存在一条回边*l=*(*n*,*h*)，则该条回边对应的*natural loop*为：



其中：

* *head=h*，是*natural loop*的头节点，即循环入口点；
* ，是*natural loop*的尾节点，即返回到循环入口节点的最后节点，也就是回边的头节点；
* ，即不经过节点*h*能够到达节点*n*的所有节点集合。

5.2.2 环的压缩

由于本课题在API图需进行路径搜索运算，natural loop的存在会在很大程度上降低搜索效率，且文献[45]指出，通常情况下程序中的循环结构满足90/10规则，即程序中10%的代码是循环，因此为了提高搜索效率，需对API图上的环进行压缩处理。

环的压缩必须满足的条件是图是可归约的。由文献[45]可知，如果程序中仅使用到while循环、for循环、if-then(else)、break和continue这几种语言结构时，则其对应的流程图是可归约的。因此在本文中，对API图中natural loop的压缩是可实现的。算法5.1展示了natural loop压缩的具体过程。为了简化算法描述，给出以下几个定义。

**定义5.5 (*idom*)** 直接支配节点。给定一个API图节点*n*的直接支配节点是从API图的入口节点到节点*n*的所有路径必须经过的最后一个节点，即：



**定义5.6（*Dominator Tree*）**支配树。对于一个API图，其对应的*Dominator Tree*定义为：



其中：

* ，即支配树的根节点为API图的入口节点；
* ，即支配树的节点集合为API图的节点集合；
* ，即直接支配节点与被支配节点的边集合；

|  |
| --- |
| 算法5.1 环压缩算法 |
| **输入**：API图 APIG  **输出**：压缩后的API图，压缩节点与natural loop的映射表Map<LoopID, natural loop> maps;   1. 根据定义5.6构造一个API图对应的支配树Dominator Tree; 2. 根据定义5.3识别API图中的所有回边; 3. for each node *h* in Dominator tree 4. for each node *n* for which there exists a back edge *n*->*h* 5. set=所有不经过节点*h*能到达节点*n*的节点集合； 6. head=h; 7. tail=n; 8. loops.add(*natural loop*(*n*->*h*)=(head,tail,set)); 9. end; 10. end; 11. for each loop in loops 12. Node loopID=New Node(natural loop的关键信息); 13. APIG.predsOf(loop.head).removeSucc(loop.head); 14. APIG.predsOf(loop.head).addSucc(loopID); 15. APIG.succsOf(loop.tail).removePred(loop.tail); 16. APIG.succsOf(loop.tail).addPred(loopID); 17. maps.add(loopID,loop); 18. end; 19. return APIG, maps; |

在算法5.1中，给出了对API图进行环压缩的具体过程。首先在第1行和第2行分别计算API图的支配树和回边集合，这是进行natural loop识别的基础。然后算法3-10行是进行natural loop识别过程，并把得到的所有natural loop存入loops集合中。算法11-18行是将识别到的natural loop进行压缩，采用能够唯一标识natural loop的节点代替natural loop，最后将唯一标识节点与natural loop存入映射表。

## 5.3 API调用序列提取

在5.2节通过对API图中环的压缩，构造了无环API图，本节在此基础上提取回调函数的API调用序列。算法5.2展示了对某样本中所有API图进行API调用序列提取的过程。

|  |
| --- |
| 算法5.2 API调用序列提取算法 |
| **输入**：无环API图集合: Map<NLAPIG, 回调函数> NLAPIGS;  压缩节点与natural loop的映射表Map<LoopID, natural loop> map;  **输出**：Map<回调函数，List<API调用序列>> maps;   1. for each NLAPIG in NLAPIGS 2. path=*dfn*(NLAPIG); 3. for each node in path 4. if(node is loopID) 5. natural loop=maps.get(node); 6. path.replaceWith(node, “(”+natural loop+”)\*”); 7. end 8. end 9. if(maps.keySet().contains(NLAPIGS.get(NLAPIG))) 10. maps.get(NLAPIGS.get(NLAPIG)).add(path); 11. else 12. maps.put(NLAPIGS.get(NLAPIG),path); 13. end; 14. end; |

在算法中，对某样本的若干个API图进行API调用序列提取。首先，对于每个无环API图，即NLAPIG进行深度优先遍历，获取API调用序列，然后在第3-8行，对API调用序列中环的唯一标识节点采用“(环中的API调用序列)\*”进行替换，最后将此API调用序列与其对应的回调函数存入map映射表。由于每个样本中可能会存在多个相同的回调函数调用了不同的API调用序列，因此该算法获得的最终结果为回调函数与API调用序列集合的映射表。

## 5.4 模糊处理

5.4.1 API调用序列的局限性

API调用序列是程序行为的最直接表现，但同一家族中不同程序的API调用序列并非完全相同，具体原因如下：

（1）Android框架层提供的API接口众多，不同应用程序为了实现同一操作可调用不同的API，如sendTextMessage()和sendDataMessage()都可实现发送短信的操作；

（2）某些恶意行为的实现对API的调用顺序没有具体要求，如getDeviceId()和getSubscriberId()都用来获取手机相关的隐私信息，而这两个API对调用顺序没有要求。

因此，如果将API调用序列直接作为行为块去挖掘，会造成拟合度过高，无法识别家族中应用程序之间的变化点。因此，在本文中，我们在5.3节提取出API调用序列的基础上，对该序列进行模糊处理。其优点主要有以下几点：

（1）相比于提取程序中每个方法的API调用序列[24]，提取基于回调函数的API调用序列能够很大程度上降低特征维度，提高算法效率。

（2）由于回调函数是Android应用程序执行的基本单位，因此由基于回调函数的API调用序列转化而来的文本序列也能够在一定程度上反映各种事件（用户界面事件或系统事件）所触发的程序行为。

（3）经过模糊处理的文本序列在保留了程序行为特征的基础上，消除了程序之间存在的差异性，能够更好反映家族存在的共性特征。

5.4.2 模糊处理规则

为了描述恶意程序家族中所有应用程序的共性行为，则需要消除其个性差异，保留共性特征，因此，本文定义了一组用于处理API调用序列，构造行为块的模糊处理规则，本文采用的模糊处理方法为：

* 对于受权限保护的API：采用权限信息替代API调用。由于API调用是程序的具体操作，而权限是该操作涉及到的敏感资源，相比于API调用，权限是将具体操作进行抽象，映射到对设备资源进行操作的一组特征，因此更能消除行为存在的个性，保留其本质属性；
* 对于Source & Sink API：用SOURCE替代SUSI中定义的所有source API，用SINK替代所有sink API。SUSI中定义的所有source API预示了该API是数据进入程序的源点，由于我们并不关注数据是如何进入程序的，只关心数据进入程序这一动作，所以将所有的source API使用SOURCE字符串代替，同理用SINK替代SUSI中定义的所有sink API。
* 对于与动态加载相关的API：采用DYNAMIC\_LOAD替代与动态加载相关的API，该操作也是为了保留操作属性，消除行为的个性特征。
* 对于其它恶意程序常用API：采用SENSITIVE\_API替代所有其他恶意程序常用的API，表示该操作为一个恶意程序经常进行的一个敏感操作，而并不关注该操作本身。

下面给出模糊处理的具体定义。

**定义5.7**（模糊处理）假设某API调用序列为，则模糊处理规则可定义为一个转换函数***O***，将转换成：



其中：

 在本节中，我们对API调用序列进行了模糊处理。很显然API调用序列可以继承API图对应的回调函数信息，因此，为了更好描述程序的语义信息，本文将由回调函数与经过模糊处理的API调用序列组成的二元组称为行为块。如某经过模糊处理的API调用序列为*sequence=*(SOURCE,SEND\_SMS,SINK)，该API调用序列对应的回调函数为onReceive()，设行为块为AC，则

AC=<onReceive(), (SOURCE,SEND\_SMS,SINK)>

AC很显然可用表5.1中的文法表示。

表5.1 AC的语法描述

|  |  |  |
| --- | --- | --- |
| <AC> | ::= | ‘<’<callback>’, (‘<word>\*’)>’ |
| <word> | ::= | |<permission>|<permissionList>|’SOURCE’|’SINK’|’DYNAMIC\_ LOAD’|’SENSITIVE\_API’|’,’ |
| <permission> | ::= | ‘SEND\_SMS’|.......|’INTERNET’ |
| <permissionList> | ::= | <permission>|AND <permission> |
| <callback> | ::= | ‘onClick()’|......|’onReceive()’ |

由于篇幅有限，本文给出部分广义敏感API与替代字符串之间的对应关系，如表5.2所示。

表5.2 广义敏感API与替代字符串的对应关系

|  |  |
| --- | --- |
| **API** | **替代字符串** |
| android/telephony/TelephonyManager;->getSubscriberId | READ\_PHONE\_STATE |
| android/provider/Contacts$People;->markAsContacted | WRITE\_CONTACTS AND READ\_SOCIA L\_ STREAM |
| android/net/wifi/WifiManager;->getConnectionInfo | ACCESS\_WIFI\_STATE, DUMP |
| android/location/LocationManager;->setTestProviderStatus | ACCESS\_MOCK\_LOCATION |
| android/provider/CallLog$Calls;->getLastOutgoingCall | ADD\_VOICEMAIL AND READ\_CALL\_L OG AND DUMP |
| android/bluetooth/BluetoothA2dp;->disconnect | BLUETOOTH\_ADMIN AND DUMP |
| android/app/DownloadManager;->addCompletedDownload | VIBRATE AND INTERNET AND WRITE\_E XTERNAL\_STORAGE |
| android/net/ConnectivityManager;->requestRouteToHost | CHANGE\_NETWORK\_STATE AND DUMP |
| android/provider/Browser;->addSearchUrl | READ\_HISTORY\_BOOKMARKS AND W RITE\_HISTORY\_BOOKMARKS AND GE T\_ ACCOUNTS, DUMP |
| android/provider/Settings$System;->setShowGTalkServiceStatus | WRITE\_SETTINGS AND DUMP |
| java/net/URL;->openConnection | INTERNET |
| android/accounts/AccountManager;->getAccountsByType | GET\_ACCOUNTS |
| com/android/internal/telephony/ITelephony;->call | CALL\_PHONE |
| android/telephony/SmsManager;->getAllMessagesFromIcc | RECEIVE\_SMS |
| com/android/mms/util/PduLoaderManager;->getPdu | READ\_SMS |
| javax.crypto.SecretKeyFactory->generateSecret | SENSITIVE\_API |
| android/net/wifi/WifiInfo;->getMacAddress | SOURCE |
| android/content/ContentResolver;->insert | SINK |
| dalvik.system.DexClassLoader;->loadClass | DYNAMIC\_LOAD |

如某回调函数onCreate()对应的API调用序列为：

path=(getSystemService("phone"),getSystemService("location"),getAction())

则其对应的行为块为：

AC=<onCreate(), (READ\_PHONE\_STATE,ACCESS\_COARSE\_LOCATION,DUMP)>

## 5.3 Android恶意程序家族中行为块分析

5.3.1 分析指标

为了探索家族内部行为块的分布情况，应用程序中行为块分布与恶意家族中行为块分布的差异性，以及能否直接从恶意家族的所有行为块中找出一组能够有效表征家族的子集。我们首先给出了一组能够执行此分析的度量指标：

* AC\_set(*a*)：表示样本*a*中拥有的行为块集合，对于重复的行为块，只记录一次；
* ：表示家族F*i*中的所有应用程序行为块的集合；
*  表示家族F*i*中每个应用程序中都存在的行为块集合；
* ：表示对于家族集合M={F1，F2，...，F*m*}，在家族F*i*中的所有应用程序中都存在，并且没有在其他家族的应用程序中出现过的行为块集合，该集合很显然原则上能够作为从家族集合M中唯一标识家族F*i*的特征。

如某家族F*j*中存在3个恶意样本，每个样本的行为块集合AC\_set如表5.3所示。从表中我们可得出以下指标的度量结果：

* FAC(F*j*)={<onCreate(android.os.Bundle), (DUMP,SINK,DUMP,READ\_PHONE\_ST ATE,INTERNET,SOURCE,INTERNET,SINK)>, <onCreateDialog(int), (INTERAC T\_ACROSS\_USERS,INTERACT\_ACROSS\_USERS\_FULL)>}，即样本1，样本2和样本3共有的行为块集合，该集合可在一定程度上描述恶意软件家族实现的恶意类别；
* CAC(F*j*)={<onCreate(android.os.Bundle), (DUMP,SINK,DUMP,READ\_PHONE\_S TATE,INTERNET,SOURCE,INTERNET,SINK)>, <onCreateDialog(int), (INTERA CT\_ACROSS\_USERS,INTERACT\_ACROSS\_USERS\_FULL)>}，即每个样本中所包含的行为块的并集；
* 假设并未在除过F*j*的所有其他家族的应用程序中出现过行为块<onCreate(androi d.os.Bundle), (DUMP,SINK,DUMP,READ\_PHONE \_STATE,INT ERNET,SOURCE, INTERNET,SINK)>，则DFAC(F*j*) ={<onCreate(androi d.os.Bundle), (DUMP,SINK,D UMP,READ\_PHONE \_STATE,INTERNET,SOURCE,INTERNET, SINK)>}，原则上，该行为块可作为独立判别家族F*j*的特征。

表5.3 家族F*j*中样本的行为块分布

|  |  |
| --- | --- |
| 样本 | 行为块 |
| 样本1 | <onCreate(android.os.Bundle),(DUMP,SINK,DUMP,READ\_PHONE\_STATE,INTERNET,SOUCE,INTERNET,SINK)>  <onCreateDialog(int),(INTERACT\_ACROSS\_USERS,INTERACT\_ACROSS\_USERS\_FULL)> |
| 样本2 | <onCreate(android.os.Bundle),(DUMP,SINK,DUMP,READ\_PHONE\_STATE,INTERNET,SOUCE,INTERNET,SINK)>  <onCreateDialog(int),(INTERACT\_ACROSS\_USERS,INTERACT\_ACROSS\_USERS\_FULL)> |
| 样本3 | <onCreate(android.os.Bundle),(DUMP,SINK,DUMP,READ\_PHONE\_STATE,INTERNET,SOUCE,INTERNET,SINK)>  <onCreateDialog(int),(INTERACT\_ACROSS\_USERS,INTERACT\_ACROSS\_USERS\_FULL)><onDestory, (DUMP)> |

5.3.2 分析与讨论

对本文收集到的所有恶意样本家族中的样本进行行为块提取，并根据以上各项指标进行统计。在统计过程中，以回调函数作为行为块比较的前置条件，即当两个行为块的回调函数相同时，才进一步比较。统计结果如表5.4所示。

表5.4 度量指标统计

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 家族F*i* | |F*i*| | 程序指标 | 家族指标 | | | |
| |AVG(AC(*a*)| | |FAC(F*i*)| | |CAC(F*i*)| | |CAC(*a*)(90%)| | |DFAC(F*i*)| |
| AccuTrack | 24 | 12 | 96 | 4 | 7 | 6 |
| Adrd | 36 | 21 | 291 | 0 | 4 | 1 |
| AnserverBot | 22 | 3 | 4 | 2 | 2 | 2 |
| Aks | 24 | 2 | 17 | 1 | 1 | 0 |
| BaseBridge | 139 | 19 | 584 | 0 | 5 | 1 |
| BeanBot | 23 | 14 | 276 | 3 | 3 | 3 |
| Biige | 23 | 22 | 236 | 8 | 13 | 4 |
| Boxer | 34 | 11 | 172 | 0 | 2 | 2 |
| Coogos | 24 | 34 | 475 | 6 | 15 | 10 |
| Copycat | 25 | 44 | 716 | 5 | 13 | 2 |
| Cosha | 26 | 29 | 375 | 1 | 11 | 1 |
| Dougalek | 25 | 2 | 2 | 2 | 2 | 2 |
| DroidDream | 39 | 20 | 238 | 2 | 7 | 1 |
| DroidKungFu | 311 | 20 | 3175 | 0 | 8 | 4 |
| DroidSheep | 25 | 22 | 244 | 7 | 12 | 1 |
| EICAR-Test-File | 23 | 1 | 3 | 0 | 0 | 0 |
| ExploitLinuxLotoor | 27 | 28 | 513 | 0 | 11 | 3 |
| FaceNiff | 24 | 24 | 284 | 4 | 9 | 3 |
| FakeDoc | 65 | 34 | 551 | 7 | 12 | 6 |
| FakeInstaller | 398 | 8 | 945 | 0 | 3 | 1 |
| Fakelogo | 27 | 3 | 13 | 2 | 2 | 2 |
| Fakengry | 25 | 73 | 912 | 4 | 21 | 1 |
| FakePlayer | 29 | 1 | 7 | 0 | 1 | 0 |
| FakeRun | 29 | 33 | 101 | 19 | 23 | 11 |
| FakeTimer | 28 | 4 | 52 | 0 | 2 | 0 |
| FoCobers | 28 | 39 | 257 | 24 | 26 | 15 |
| Gappusin | 26 | 28 | 179 | 1 | 15 | 1 |
| Geinimi | 42 | 12 | 269 | 0 | 3 | 0 |
| GinMaster | 157 | 21 | 954 | 0 | 7 | 2 |
| Glodream | 28 | 20 | 297 | 1 | 3 | 1 |
| Hamob | 35 | 21 | 113 | 17 | 18 | 12 |
| Iconosys | 67 | 5 | 151 | 0 | 3 | 1 |
| Imlog | 38 | 15 | 352 | 1 | 6 | 3 |
| Jifake | 30 | 3 | 53 | 0 | 1 | 0 |
| Kidlogger | 25 | 20 | 361 | 0 | 6 | 2 |
| Kmin | 37 | 21 | 232 | 0 | 13 | 7 |
| Lemon | 24 | 51 | 690 | 12 | 19 | 11 |
| Mania | 25 | 2 | 2 | 2 | 2 | 2 |
| Mobilespy | 25 | 29 | 376 | 7 | 14 | 3 |
| MobileTx | 31 | 8 | 25 | 1 | 6 | 3 |
| Mobinauten | 25 | 8 | 75 | 1 | 3 | 1 |
| Nandrobox | 28 | 29 | 365 | 4 | 12 | 4 |
| Nickspy | 24 | 18 | 257 | 0 | 7 | 3 |
| Nyleaker | 30 | 42 | 691 | 5 | 16 | 3 |
| Opfake | 283 | 4 | 557 | 0 | 1 | 1 |
| Penetho | 27 | 8 | 97 | 2 | 3 | 2 |
| Placms | 25 | 12 | 133 | 2 | 3 | 2 |
| Plankton | 291 | 28 | 2957 | 2 | 10 | 4 |
| RootSmart | 24 | 23 | 361 | 1 | 7 | 5 |
| SeaWeth | 25 | 56 | 745 | 3 | 19 | 9 |
| SendPay | 23 | 36 | 272 | 2 | 11 | 5 |
| SerBG | 26 | 43 | 634 | 3 | 14 | 7 |
| SMSreg | 36 | 32 | 678 | 0 | 13 | 5 |
| SMSZombie | 23 | 5 | 5 | 5 | 5 | 4 |
| Spitmo | 23 | 2 | 1 | 1 | 1 | 1 |
| SpyHasb | 22 | 16 | 16 | 16 | 16 | 14 |
| Stealer | 29 | 4 | 36 | 0 | 2 | 0 |
| Stiniter | 23 | 1 | 7 | 0 | 1 | 0 |
| Trackplus | 25 | 14 | 176 | 3 | 7 | 2 |
| TrojanSMS.Hippo | 29 | 20 | 248 | 0 | 5 | 1 |
| Typstu | 24 | 9 | 58 | 5 | 5 | 3 |
| Vdloader | 25 | 34 | 357 | 0 | 6 | 2 |
| Xsider | 27 | 8 | 113 | 0 | 1 | 1 |
| Yzhc | 35 | 8 | 87 | 1 | 1 | 1 |
| Zitmo | 25 | 3 | 49 | 0 | 1 | 1 |

整个数据集共包含恶意程序3270个，包含的行为块共23568个。图5.2展示了每个恶意软件家族内应用程序行为块个数的变化区间。从该图可以看出，某些家族中应用程序行为块个数存在较大差异，如家族Fakengry中，应用程序中包含的行为块个数最少为37个，最多为106个，家族内应用程序的平均行为块个数也高达73个。而在另一些家族中，应用程序包含的行为块个数变化区间较小，且家族内应用程序的平均行为块个数也非常少，如家族Spitmo中应用程序的行为块平均只有2个。我们分析产生这种情况的原因是：Spitmo家族是SpyEye木马的最早版本之一，该家族中的应用程序从受感染的手机中窃取隐私信息，行为单一，且传播范围有限。根据调查，此类恶意代码目前只存在于银行相关的应用程序中，因此该家族中应用程序的行为块数目较少。而Fakengry存在大量变种，其恶意行为众多，如在程序安装的过程中向付费号码发送短信，后门执行远程服务器发送的指令等，因此通常出现在各类常用的应用程序中，所以该家族中应用程序的行为块数目较多，且变化幅度较大。



图5.2 各家族中应用程序包含行为块个数的变化范围

接下来分析家族指标的统计数据。FAC描述了一个恶意软件家族内所有应用程序存在的行为块集合，图5.3展示了不同恶意软件家族的FAC指标值。从图中可以看出，各个家族的FAC指标差异较大，而这似乎与家族中应用程序的平均行为块数无关，因此本文猜想这可能与恶意家族的扩散与流行程度有关，如恶意家族DroidKungFu，Plankton等经常出现在许多非常常见的重打包应用程序中，因此在不同的软件中找到相同的恶意程序会造成该恶意程序家族行为块总数的急剧增加。



图5.3 各家族的FAC指标统计

CAC描述了整个恶意软件家族中所有应用程序共有的行为块，该集合的大小在不同家族中也表现出较大的差异性。在经历过重大变化的家族中该集合甚至为空，如家族FakeInstaller中398个应用程序无共同行为块。而那些不是非常流行的恶意家族的CAC集合比较大，如家族SpyHasb中22个应用程序拥有16个相同的行为块。从统计结果可以看出，在65个恶意程序家族中，有24个家族的CAC为空。很显然，如果仅采用简单的文本对比寻找家族内应用程序的共同行为块，则有1/3的恶意家族内的应用程序不存在共同行为块。

通过人工分析某些家族内应用程序行为块之间的关系，发现部分家族内应用程序之间虽然不存在完全相同的行为块，但行为块之间存在非常高的相似度，本文猜想这可能就是应用程序之间存在的变化点。因此，因此本文采用一种相似度计算方法度量两个行为块之间的相似度，方法如定义5.8所示。

**定义5.8** (Similarity\_Ratio)假设存在两个文本序列分别为，，则这两个文本序列的相似度计算方式为：

 （5.5）

其中*dif* (*s*1,*s*2)指两个文本序列中不同的单词个数，其计算方式如下：



如给定两个文本序列*p*1=“早上好，很高兴见到你！”，*p*2=“晚上好，很高兴见到你！”，则*p*1，*p*2的相似度为：



本文将两个回调函数相同，模糊API调用序列相似度不低于90%的行为块认定为属于同一行为块，从而包容程序之间的变化点。图5.4展示了基于简单文本对比方法提取的CAC集合个数与基于文本相似度方法提取的CAC集合个数的对比图，从该图可以看出，基于文本相似度方法能够很大程度上增加CAC集合中的元素。



图5.4 各家族的CAC/CAC(90%)/DFAC指标统计

最终对基于文本相似度方法提取的CAC集合进一步处理，提取出家族内程序共享，并且未在其他家族中出现过的行为块集合，即DFAC集合。图5.4同样展示了每个家族的DFAC集合元素个数。结果显示，在所有的恶意软件家族中，只有8个家族的DFAC集合为空，剩下的57个家族的DFAC集合都不为空，原则上这些行为块可以作为家族预测的判别特征，但这种判别特征将会非常脆弱，当某家族出现一个新的恶意程序，该程序与该家族中其他所有的应用程序不存在相同的行为块时，则会导致CAC集合为空，从而导致DFAC为空，即不存在能够识别家族的特征。但此应用程序可能并不真正属于该家族，而是被误判，或者是由某些恶意软件开发者经过精心构造，从而避免与家族中的其他应用程序拥有相同的行为块。无论是哪种情况，都会导致DFAC不可用。接下来，为了构造更强大的家族特征表示，我们将分析所有的行为块在各个家族中的分布情况。

我们首先统计了23568个行为块在65个恶意程序家族中的分布情况，如图5.5所示。该图展示了行为块数目与其所分布的家族数目之间的函数变化关系（我们认为如果某行为块出现在某家族的至少一个应用程序中，则该行为块分布在该家族）。从图中可以看出，有81.49%的行为块只出现在一个恶意软件家族中，这表明如果一个行为块出现在一个恶意软件家族，则不太可能在其他恶意软件家族中找到相同的行为块。同时出现在两个家族中的行为块比例骤降至5.94%，并且总共只有不到5%的行为块同时出现在九个及以上的家族。该分布表明，每个恶意软件家族可以表现为几个行为块，并伴随着一些额外的信息，如行为块在家族中出现的频率等。



图5.5 行为块数目与家族数目的变化关系

## 5.4 基于文本挖掘技术的家族预测器构造

基于以上讨论结果，我们采用文本挖掘技术分析了恶意程序样本和家族，并构造了能够对未知家族的恶意样本进行家族分类的1-NN分类器。本节首先介绍向量空间模型，然后详细介绍家族分析器的构造过程。

5.4.1 向量空间模型

由于我们的行为块实质为普通文本，因此能够很好地适应文本挖掘领域中的各项数字度量指标，而文本挖掘领域中的一个重要概念为向量空间模型（Vector Space Model）。此概念是由Salton于1975年提出，它将对文本的处理简化成向量空间中的向量运算，采用空间上的相似度表达语义相似度。首先采用文档集合中的所有文档构造文档库，则文档库中的第*j*个文档可被表示为文档空间中的一个向量：

t*j*=(*m*1,*j*，...，*mk*,*j*)

其中*k*为文档库中词的总个数。元素表示第*i*个词*wi*在文档t*j*中所具有的权重，权重的计算是向量空间模型的关键所在。

TF-IDF（Term Frequency-Inverse Document Frequency）是向量空间模型中最常用的统计指标，该指标用于评估文档库中一个词对于一个文档的重要性，涉及两个最重要的概念：（1）词频：描述了一个词在文档中出现的频率。（2）逆文档频率：用于描述一个词在文档库中的所有文档中是常见的还是罕见的。通常由文档库中的总文档数除以包含该词的文档数，然后对结果取以10为底的对数得到。因此很明显我们可以看出，如果一个词在某文档中出现的频率高，而在文档库中的其他文档中很少出现，则该词对应的TF-IDF值会比较高，因此具有很好的类别区分能力，这与行为块对于家族的类别区分如出一辙。

我们将该模型应用于本文的家族预测问题。首先需构造恶意软件家族的文档库，具体构造过程为：

1. 新建包含65个文档的文档库D；
2. 选择一个未处理的家族F*j*，首先对家族内每个应用程序的行为块进行去重操作，即每个应用程序中，重复的行为块只能包含一次，然后将所有应用程序中剩下的行为块集合进行合并，存入文档D*j*中；
3. 判断所有家族是否全部处理，没有的话进入（2），否则结束。

经过以上过程，得到了一个以行为块为基本单词，以恶意家族为文档类别的文档库。一个家族对应一个文档，因此一个家族可被表示为一个向量。如家族F*j*可被表示为*v*j=(I1,*j*, ..., I*k*,*j*)，其中I*i*,*j*=I(*ci*, F*j*, M)，计算公式为：

 （5.6）

下面给出TF和IDF的具体定义和计算公式。

**定义5.9** (**TF**) TF(*c*, F*j*)表示行为块*c*在家族F*j*中出现的频率，其计算公式为：

 （5.7）

其中freq(*c*, *a*)为行为块*c*在应用程序*a*中出现的次数。

**定义5.10** (**IDF**) IDF(*c*, M)表示行为块*c*在家族集合M中的逆文档频率，计算公式为：

 （5.8）

当某行为块在所有的家族中都不存在时，则会导致分母为零。因此通过给分母加1来应对这种情况。

5.4.2 案例说明

为了更直观说明空间向量模型对于家族分类预测具有一定的有效性，本节给出一个案例。假设存在一个恶意软件家族库M1={F1,F2,F3,...,F100}，其中每个家族中应用程序的行为块*ci*分布情况如表5.5所示。从表5.5可以看出，行为块*ci*在家族F1具有的权值最高，因为该家族中的所有应用程序都包含该行为块，且该行为块在其他家族中出现的频率很低。对于那些不包含该行为块的家族，其对应的权值为0。

表5.5 某恶意家族库中行为块*ci*的权值计算

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M1 | F1 | | | F2 | | | ... | F100 |
| 样本 | App1 | App2 | App3 | App4 | App5 | App6 | ... | App*n* |
| 是否包含行为块*ci* | **√** | **√** | **√** | **×** | **×** | **√** | **×** | **×** |
| TF | 1 | | | 1/3 | | | 0 | 0 |
| IDF |  | | | | | | | |
| TF-IDF | 1.523 | | | 0.508 | | | 0 | 0 |

5.4.3 预测器构造

根据5.4.2节的案例可知，文本空间向量模型可以很好适应本课题的家族分类问题，该模型可以根据行为块在家族中的分布情况，计算出每个行为块在每个家族中的TF-IDF权值，该权值在一定程度上反映了行为块对于家族类别的区分能力。因此，本节将采用文本向量模型进行预测器的构造。

（1）家族特征向量构造

首先，将数据集分为训练集和测试集两部分，由于本文所采用的数据集中共包含65个恶意软件家族，且每个家族的恶意样本数量不尽相同，有的家族中样本数量高达近400个，而有的仅有20多个。因此为了保持每个家族的训练样本数量相同并且最大程度上训练出效果良好的分类器，本文从每个家族中随机选取20个样本，共1300个样本组成训练集，而将剩下的所有数据集作为测试样本。

本文以TF-IDF作为统计指标，将文本空间向量模型应用于训练集，为每一个家族构造一个特征向量，构造过程如算法5.3所示。该算法的处理过程主要分为三个步骤：

1）对输入的数据集进行处理，统计每个家族的FAC集合；

2）统计所有家族的行为块集合，对每一个家族，计算每个行为块的IDF及TF值；

3）根据TF和IDF值，计算行为块的权值，从而为每一个家族构造行为特征向量。

|  |
| --- |
| 算法5.3 家族特征向量构造 |
| **输入**：带有家族标识的应用程序行为块序列集合：{(*a*1,F*a*1)，(*a*2,F*a*2)，...，(*a*1300,F*am*)}  家族集合  **输出**：向量集，如家族F*j*对应的特征向量为*vj*=(I1,*j*，I2,*j*，...，I*k*,*j*)   1. for j=1 to 65 3. for each 4. if (F*ai*=Fj) 6. end if 7. end for 8. end for 10. k=|C(M)|; 11. for i=1 to k 12. 根据公式（5.8）计算IDF(*c*i, M); 13. end for 14. for each Fj do 15. for i=1 to k 16. 根据公式（5.7）计算TF(*c*i, Fj); 17. *v*j[i]=Ii,j=TF(*c*i, Fj)·IDF(*c*i, M); 18. end for 19. end for 20. return{*v*1, *v*2, ..., *v*65}; |

（2）1-NN分类器构造

在构造家族特征向量时，我们从所有的训练集中提取了23568个行为块，即*k*=23568，因此每个家族的行为特征向量为23568维。通过观察发现，我们所构造的家族特征向量非常稀疏，平均而言，每个特征向量中只有约9%包含能够进行家族判别的信息。这种情况正如我们在5.4节分析的结果那样。

采用与算法5.3相似的方式对测试集中的样本进行处理，将待测样本处理成23568维的矢量表示。为了能够判别待测样本所属的恶意家族，我们构造了1-NN分类器，即计算待测样本与每个家族特征向量之间的相似度，然后将其归类到与其距离最近的一个家族。在本课题中我们采用余弦相似性衡量两向量之间的相似度，关于与余弦相似性的定义如定义5.11所示。余弦相似性越大，相似度越高，因此待测样本最终会被判定为属于与其余弦相似性最高的家族。具体过程如算法5.4所示。

**定义5.11**（**余弦相似度**）向量d=(*d*1，*d*2，...，*dk*)与向量e=(*e*1，*e*2，...，*ek*)之间的余弦相似度计算公式为：

similarity(d,e)=cos(d,e)= （5.9）

|  |
| --- |
| 算法5.4 1-NN分类器构造 |
| **输入**：待测样本*a*的行为块集合;  行为块集合C(M)，IDF(*c*i, M);  **输出**：预测家族Fj   1. for each *c*i in C(M) 2. *u*[i]=TF(ci, *a*)·IDF(ci, M); 3. end for 4. j=max(similarity(*u*, *v*j)); 5. return Fj |

5.5 实验及结果分析

5.5.1 环压缩的有效性

循环识别是反编译中控制流分析的关键步骤，现阶段对于循环的识别主要应用于代码优化领域。本文则是通过识别环并在API图中进行环的压缩，从而提高特征提取算法在图上的搜索效率。为了验证对循环处理的必要性，我们对300个apk文件进行了循环的统计，并将统计结果展示在了图5.6中。从图中我们得知，循环结构在代码中的存在是一个普遍现象，并且随着apk的增大，循环的数目也随之增大，在apk大小为1.5~2MB时，程序中环的数目平均可达162个，这会对图上路径的搜索带来非常大的阻碍。因此，本文对于API图上循环的压缩是有必要的。



图5.6 Android应用中循环的数量统计

5.5.2 1-NN分类器的有效性

将测试集中的所有样本应用于1-NN分类器进行家族预测，其对于每个家族的预测结果如表5.6所示。从表中可以看出，本文的1-NN分类器几乎对于所有恶意软件家族都具有较好的预测效果。

表5.6 1-NN分类器预测结果

|  |  |  |  |
| --- | --- | --- | --- |
| **错误率：未正确预测样本数/总样本数（%）** | | | |
| AccuTrack | 0/4（0.00%） | Jifake | 0/10（00.00%） |
| Adrd | 3/16（18.75%） | Kidlogger | 0/5（00.00%） |
| AnserverBot | 0/2（00.00%） | Kmin | 1/17（5.88%） |
| Aks | 0/4（00.00%） | Lemon | 0/4（00.00%） |
| BaseBridge | 2/119（1.68%） | Mania | 0/5（00.00%） |
| BeanBot | 0/3（00.00%） | Mobilespy | 0/5（00.00%） |
| Biige | 0/3（00.00%） | MobileTx | 0/11（00.00%） |
| Boxer | 0/14（00.00%） | Mobinauten | 0/5（00.00%） |
| Coogos | 0/4（00.00%） | Nandrobox | 0/8（00.00%） |
| Copycat | 0/5（00.00%） | Nickspy | 0/4（00.00%） |
| Cosha | 0/6（00.00%） | Nyleaker | 0/10（00.00%） |
| Dougalek | 0/5（00.00%） | Opfake | 9/263（3.42%） |
| DroidDream | 1/19（5.26%） | Penetho | 0/7（00.00%） |
| DroidKungFu | 3/291（1.03%） | Placms | 0/5（00.00%） |
| DroidSheep | 0/5（00.00%） | Plankton | 7/271（2.58%） |
| EICAR-Test-File | 0/3（00.00%） | RootSmart | 0/4（00.00%） |
| ExploitLinuxLotoor | 1/7（14.28%） | SeaWeth | 0/5（00.00%） |
| FaceNiff | 0/4（00.00%） | SendPay | 1/3（33.33%） |
| FakeDoc | 2/45（4.44%） | SerBG | 0/6（00.00%） |
| FakeInstaller | 5/378（1.32%） | SMSreg | 0/16（00.00%） |
| Fakelogo | 0/7（00.00%） | SMSZombie | 0/3（00.00%） |
| Fakengry | 0/5（00.00%） | Spitmo | 0/3（00.00%） |
| FakePlayer | 0/9（00.00%） | SpyHasb | 0/2（00.00%） |
| FakeRun | 0/9（00.00%） | Stealer | 0/9（00.00%） |
| FakeTimer | 0/8（00.00%） | Stiniter | 0/3（00.00%） |
| FoCobers | 0/8（00.00%） | Trackplus | 0/5（00.00%） |
| Gappusin | 0/6（00.00%） | TrojanSMS.Hippo | 0/9（00.00%） |
| Geinimi | 2/22（9.09%） | Typstu | 0/4（00.00%） |
| GinMaster | 7/137（5.11%） | Vdloader | 0/5（00.00%） |
| Glodream | 0/8（00.00%） | Xsider | 0/7（00.00%） |
| Hamob | 0/15（00.00%） | Yzhc | 0/15（00.00%） |
| Iconosys | 0/47（00.00%） | Zitmo | 0/5（00.00%） |
| Imlog | 0/18（00.00%） | 总数 | 44/1970（2.23%） |

5.5.3 与相关工作对比

文献[24]同样采用文本挖掘技术进行恶意软件家族的分析及家族预测，不同之处在于文献[24]将由类方法的CFG转化而来的代码结构作为家族特征进行挖掘，因此本文将本部分实验结果与文献[24]的部分实验结果进行对比。如图5.7所示。对比结果显示，本文方法对恶意软件家族预测的准确率在整体水平上要优于文献[24]，我们分析产生这种情况的原因是，相比于文献[24]采用以类方法为单位的代码结构进行文本挖掘，本文的行为块以回调函数为单位，并在行为块中附加了回调函数信息，即在考虑程序行为的同时考虑了行为的触发方式，也就是行为的上下文。在恶意程序中，行为的触发方式对于行为的执行效率至关重要，恶意软件家族为了实现特定目的在其恶意行为的触发方式上具有一定的规律，同一家族的恶意程序在恶意行为的实现方式上存在极高的相似性，因此将行为上下文考虑在内会提高家族特征向量的强壮性。



图5.7 与文献[24]的部分实验结果对比

5.6 安全摘要生成

5.6.1 家族行为提取

现有的Android恶意程序检测方法层出不穷，为了躲避检测，延长其工作寿命从而获得最大程度的经济利益，许多恶意软件通常是嵌入了恶意代码片段的良性程序，因此如何从恶意软件的众多行为中区分出恶意行为成为关键。

在5.3.2节，我们统计的CDAC指标描述了整个恶意软件家族中所有应用程序共有，并且该家族特有的行为块集合。在本节我们将CDAC集合中的行为块作为家族行为，但表5.4结果显示，在65个恶意软件家族中，有58个家族的CDAC不为空，而剩余的7个家族所对应的CDAC集合为空，然而除过EICAR-Test-File家族之外的6个家族对应的CAC不为空，因此对于这6个家族，采用CAC作为家族行为。而对于EICAR-Test-File家族，该家族是由欧洲计算机防病毒协会开发的一款仅用于测试反病毒软件是否安装的检测代码，该病毒本身不存在威胁，因此对于属于该家族的恶意软件，本文给出的安全摘要为无病毒恶意软件。

5.6.2 家族安全摘要生成

CDAC集合中的行为块是由API调用路径经过模糊处理得到的，其忽略了行为的具体操作信息。为了给出行为产生的安全摘要，我们必须保留这种具体操作，因此需对行为块对应的API调用路径进行人工分析。表5.7给出部分家族对应的安全摘要。

表5.7 恶意程序家族安全摘要

|  |  |  |  |
| --- | --- | --- | --- |
| 家族 | API调用序列 | 行为分析 | 安全摘要 |
| Aks | getExternalStorageDirectory(),mkdirs(),getResources(),openRawResource(2130968576),getSystemService("phone"),getSimCountryIso()>(),getDeviceId()>(),getExternalStorageDirectory()>(),append(java.lang.String)>("/Apps/"),append(java.lang.String)>("file.apk"),openConnection()>(),setRequestMethod(java.lang.String)>("GET"),connect()>(),getContentLength()>(),getInputStream()>(),read(byte[])>($r3), close()>(). | 获取sd卡的根目录，并在根目录下创建/Apps目录，然后获取系统存在的资源，并以数据流的方式读取资源存在/Apps目录下的文件中。然后获取手机系统服务，获取sim卡提供的国家代码及设备id存入文件。最后后台下载某些数据。 | 获取设备信息并发送给远程服务器，并在后台进行数据下载造成用户流量损失 |
| AccuTrack | getAction(),equals("android.intent.action.BOOT\_COMPLETED"),startService(TrackingService). | 监听系统开机广播，待手机开机，启动后台服务TrackingService。 | 通过监听系统广播事件BOOT\_COMPLETED，从而启动位置跟踪服务，将被感染设备变成一个GPS跟踪器 |
| getSystemService(java.lang.String)>("power"),newWakeLock(int,java.lang.String)>(1,"GPS"),getSystemService(java.lang.String)>("phone"),listen(android.telephony.PhoneStateListener,int)>($r1,2),getSystemService(java.lang.String)>("location"),isProviderEnabled(java.lang.String)>("gps"),registerReceiver(android.content.BroadcastReceiver,android.content.IntentFilter)>($r7,$r8),getSystemService(java.lang.String)>("alarm"),getBroadcast(android.content.Context,int,android.content.Intent,int)>($r0, 0, $r1, 134217728) | 在TrackingService服务中，获得电源管理服务，并获得唤醒锁以保持CPU运转防止设备休眠。然后监听设备状态，获得系统的位置服务，注册监听手机设置更新，GPS更新请求，闹钟以及电量发生变化的广播，当发生此类事件，继续唤醒TrackingService服务内部intentBroadcastReceiver。 |

因此，当采用1-NN分类器将某恶意程序的家族预测为家族F*j*时，该恶意程序的安全摘要即为家族F*j*对应的安全摘要。



第 6 章 总结与展望

6.1 工作总结

现有恶意程序通过模仿良性程序的安全敏感行为和后台服务隐藏恶意负载的发生，使得良性程序和恶意程序的区分成为一项艰巨的任务。在本课题中，通过分析现有Android恶意程序的行为模式，以及传统Android恶意程序静态检测方法存在的不足，提出了一种基于多层次上下文特征的Android恶意程序静态检测方法，该方能够在检测程序恶意性方面存在着较高准确率，而且能够进一步对恶意程序进行家族预测，并给出程序的安全摘要。主要的工作内容如下：

1. 了解目前国内外Android恶意程序检测方法的研究现状，总结现有研究工作的不足之处，然后通过对恶意样本实例进行分析，深入了解良性程序与恶意程序在行为模式上的区别，最后通过将程序特征与特征所在的上下文相结合形成新的特征，从而对传统的基于机器学习算法的Android恶意程序检测方法进行改进；
2. 为了有效提取程序特征，本课题首先借助Soot对apk文件进行解析，为每个程序生成Call Graph，为程序中的每个类方法生成CFG，然后通过Call Graph提供的方法之间的调用关系信息，将多个CFG进行集成，最终生成基于回调函数的iCFG，并且，为了压缩图规模，提高特征提取效率，本文还定义了一组应用于iCFG的约简规则，从而过滤掉无用节点，生成简化版的API图；
3. 首先对大量样本进行统计分析，计算原始特征在各类应用程序的不同上下文中出现的频率差异，从而计算每个上下文对于特征的权值，最终通过权值对提取出的程序特征进行特征变换，变换后的特征嵌入特征向量，形成特征空间；
4. 收集多个恶意样本库提供的恶意样本，并从Google Play上下载不同类型的良性程序，组成样本集。将样本集分割为训练集与测试集，将训练集应用于多种机器学习算法，进行分类器训练，将测试集应用于分类器，通过实验验证了本文方法在随机森林算法上具有良好的类别区分能力；
5. 从API图中提取API调用序列，并对提取出的序列进行模糊处理，将处理后的文本序列作为家族的行为块特征，采用文本分析技术对恶意软件家族的行为块特征进行挖掘，以TF-IDF作为统计指标，采用将向量空间模型进行家族向量特征构造，最终构造1-NN分类器进行恶意软件家族的预测，并通过实验验证了该预测器具有良好的预测效果；
6. 提取家族中所有应用程序共有的行为块特征，将其作为恶意软件家族产生的恶意类别，然后将行为块还原到API调用序列，对API调用序列进行人工分析，从而给出家族的安全摘要。

6.2 工作展望

在本课题中，分别将回调函数，组件，应用程序作为广义敏感API调用，敏感权限和敏感广播的上下文，构造新的程序特征，然后采用机器学习算法进行分类器训练，并且在对程序进行恶意性判断的基础上预测了恶意程序所属的恶意家族，根据家族摘要给出了程序的安全摘要，但本文方法仍存在许多不足之处，我们将在后续的工作中不断进行完善，未来的工作计划安排如下：

1. 虽然本文在考虑程序特征的同时将特征所处的上下文这一语义信息也考虑在内，从而提高了本课题中程序恶意性检测方法的准确率。但对于这些原始特征，其在程序中的上下文不止于此，如API调用的触发条件，相关的数据依赖关系，方法参数等也包含了许多有用的信息，因此，如果能全面地将这些上下文都考虑在内，将会很大程度上提高检测准确率；
2. 在构造1-NN家族预测器时，所采用的行为特征是通过对基于回调函数的API图进行API调用序列提取，然后对该API调用序列进行模糊处理而来的，该特征虽然能够反映程序的行为信息，但实际上在从API图中提取API调用序列时忽略了API图中的代码结构信息，而代码结构实际上包含了程序行为模式等有用信息。因此，在后续工作中，将会将API图作为行为特征，然后进行家族行为特征挖掘；
3. 在进行安全摘要分析时，采用人工分析的方式对API调用序列进行分析，从而给出家族的安全摘要，然后将家族的安全摘要直接作为恶意程序的安全摘要。但在进行人工分析的时候，发现同一家族中应用程序在行为模式上具有很高的相似性，也就是说，我们可以总结每一家族的行为模式，将行为模式采用一种自然语言语法描述出来，最后直接通过将恶意程序的API调用序列采用自然语言语法进行解释，从而生成程序的安全摘要。

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\*\*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。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403实验室的杨\*、李\*、胡\*\*等同学和杜\*师妹、郭\*\*师妹、南\*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 2017 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] | 2019年上半年中国手机安全状况报告[R].360互联网安全中心,2019. |
| [2] | Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification[C]. Proceeding. of the 16th ACM Conference On Computer and Communications Security (CCS 2009). 2009:235-245. |
| [3] | 邵舒迪, 虞慧群, 范贵生. 基于权限和API特征结合的Android恶意软件检测方法[J]. 计算机科学, 2017, 044(004):135-139. |
| [4] | Arp D, Spreitzenbarth M, Hubner M, et al. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.[C]. Network and distributed system security symposium, 2014. |
| [5] | A deep learning approach to Android malware feature learning and detection |
| [6] | 杨欢, 张玉清, 胡予濮. 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. |
| [7] | 张家旺, 李燕伟. 基于N-gram算法的恶意程序检测系统研究与设计[J]. 信息网络安全,2016(8):74-80. |
| [8] | 陈铁明, 徐志威. 基于API调用序列的Android恶意代码检测方法研究[J].浙江工业大学学报,2018,46(2):148-154. |
| [9] | 荣俸萍, 方勇. MACSPMD: 基于恶意API调用序列模式挖掘的恶意代码检测[J]. 计算机科学,2018,45(5):132-138. |
| [10] | 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. |
| [11] | 彭凌. Android应用权限检测技术[D]. 安徽:中国科学技术大学,2015. |
| [12] | Wu D J, Mao C H, Wei T E, et al. DroidMat: Android Malware Detection through Manifest and API Calls Tracing[C]. Conference on Information Security. IEEE, 2012. |
| [13] | Xie N, Zeng F, Qin X, et al. RepassDroid: Automatic Detection of Android Malware Based on Essential Permissions and Semantic Features of Sensitive APIs[C]. Theoretical aspects of software engineering, 2018. |
| [14] | Enck W, Gilbert P, Han S, et al. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones[J]. ACM Transactions on Computer Systems, 2014, 32(2). |
| [15] | Sun M, Wei T, Lui J C, et al. TaintART: A Practical Multi-level Information-Flow Tracking System for Android RunTime[C]. Computer and communications security, 2016:331-342. |
| [16] | Backes M, Bugiel S, Schranz O, et al. ARTist: The Android Runtime Instrumentation and Security Toolkit[C]. IEEE european symposium on security and privacy, 2017:481-495. |
| [17] | Wei L, Luo W, Weng J, et al. Machine Learning-Based Malicious Application Detection of Android[J]. IEEE Access, 2017:25591-25601. |
| [18] | Li J, Ye Y, Zhou Y, et al. CodeTracker: A Lightweight Approach to Track and Protect Authorization Codes in SMS Messages[J]. IEEE Access, 2018: 10107-10120. |
| [19] | Fan M, Liu J, Luo X, et al. Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis[J]. IEEE Transactions on Information Forensics and Security, 2018:1890-1905. |
| [20] | Cui Z, Xue F, Cai X, et al. Detection of Malicious Code Variants Based on Deep Learning[J]. IEEE Transactions on Industrial Informatics, 2018, 14(7): 3187-3196. |
| [21] | Wagner M, Fischer F, Luh R, et al. A Survey of Visualization Systems for Malware Analysis[C]. Eurographics, 2015:105-125. |
| [22] | Silvio C, Yang X. Classification of Malware Using Structured Control Flow[C]. Australasian Symposium on Parallel and Distributed Computing, 2010:61-70. |
| [23] | Chen K, Liu P, Zhang Y, et al. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets[C]. International conference on software engineering, 2014: 175-186. |
| [24] | Suarez-Tangil G , Tapiador J E , Peris-Lopez P , et al. Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families[J]. Expert Systems with Applications, 2014, 41(4):1104-1117. |
| [25] | Ryder B G. Dimensions of precision in reference analysis of object-oriented programming languages[J]. Compiler construction, 2003: 126-137. |
| [26] | Arzt S, Rasthofer S, Fritz C, et al. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps[J]. Programming language design and implementation, 2014, 49(6):259-269. |
| [27] | Gordon M I, Kim D, Perkins J H, et al. Information-Flow Analysis of Android Applications in DroidSafe[C]. Network and distributed system security symposium, 2015. |
| [28] | Wei F, Roy S, Ou X, et al. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps[J]. ACM Transactions on Privacy and Security, 2018, 21(3). |
| [29] | Soot. <https://github.com/Sable/soot>. |
| [30] | Rasthofer S, Arzt S, Bodden E, et al. A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks[C]. Network and distributed system security symposium, 2014. |
| [31] | Huang J, Li Z, Xiao X, et al. SUPOR: precise and scalable sensitive user input detection for android apps[C]. Usenix security symposium, 2015:977-992. |
| [32] | Li L, Bartel A, Bissyande T F, et al. IccTA: detecting inter-component privacy leaks in Android apps[C]. International conference on software engineering, 2015:280-291. |
| [33] | Christensen A S, Moller A, Schwartzbach M I, et al. Precise analysis of string expressions[C]. Static analysis symposium, 2003:1-18. |
| [34] | Octeau D, Luchaup D, Dering M L, et al. Composite constant propagation: application to Android inter-component communication analysis[C]. International conference on software engineering, 2015:77-88. |
| [35] | Yang W, Xiao X, Andow B, et al. AppContext: differentiating malicious and benign mobile app behaviors using context[C]. International conference on software engineering, 2015:303-313. |
| [36] | Zhang M, Duan Y, Yin H, et al. Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs[C]. Computer and communications security, 2014:1105-1116. |
| [37] | Zhang M, Duan Y, Feng Q, et al. Towards Automatic Generation of Security-Centric Descriptions for Android Apps[C]. Computer and communications security, 2015:518-529. |
| [38] | Huang J, Zhang X, Tan L, et al. AsDroid: detecting stealthy behaviors in Android applications by user interface and program behavior contradiction[C]. international conference on software engineering, 2014:1036-1046. |
| [39] | Wang Z, Li C, Yuan Z, et al. DroidChain: A novel Android malware detection method based on behavior chains[J]. Pervasive and Mobile Computing, 2016: 3-14. |
| [40] | Feng Y, Anand S, Dillig I, et al. Apposcopy: semantics-based detection of Android malware through static analysis[C]. Foundations of software engineering, 2014:576-587. |
| [41] | BartelA KleinJ. Dexpler:converting Android Dalvik bytecode to Jimple for static analysis with Soot[C]. ACM Sigplan International Workshop on State of the Artin Java Program Analysis. ACM,2013:27-38. |
| [42] | Ondrej Lhoták, Hendren L J. Scaling Java points-to analysis using SPARK[C]. [International Conference on Compiler Construction](https://link.springer.com/conference/cc), 2003. |
| [43] | Au K W Y, Yi F Z, Zhen H, et al. PScout:analyzing the Android permission specification[C]. ACM Conference on Computer & Communications Security. 2012. |
| [44] | Zhou Y, Jiang X. Dissecting Android Malware: Characterization and Evolution[J]. Proceedings - IEEE Symposium on Security and Privacy, 2012. |
| [45] | Mohammad Ali Ghodrat, Tony Givargis, Alex Nicolau. Optimizing control flow in loops using interval and dependence analysis[J]. design automation for embedded systems. 13(3):193-221. |

附 录

硕士期间发表的论文:

1. 刘晓建,雷倩.多层次特征的Android恶意程序静态检测方法[J].华中科技大学学报（自然科学版）(已录用)
2. Xiaojian Liu, Qian Lei.A Graph-based Feature Generation Approach in Android Malware Detection with Machine Learning Techniques[J].Mathematical Problems in Engineering.

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》