分 类 号：_____ 密级：_____

学　　号：19208208047

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

硕士学位论文

Thesis for Master's Degree

基于生成对抗样本的应用程序恶意性检测和家族分类方法研究

申请人姓名：***

指 导 教 师：***（校内）***（校外）

类　　别：全日制专业型硕士

工 程 领 域：软 件 工 程

研 究 方 向：软 件 安 全

2022　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：　　日期：

学 位 论 文 知 识 产 权 声 明 书

本人完全了解学校有关保护知识产权的规定，即：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：　　指导教师签名：

年　　月　　日

论文题目：基于生成对抗样本的应用程序恶意性检测和家族分类方法研究

专　　业：软件工程

硕 士 生：***　　（签名）_____

指导教师：***（校内）　　（签名）_____

　　***（校外）　　（签名）_____

摘 要

近年来机器学习技术已经广泛应用于恶意应用检测中，基于机器学习的静态检测方法通过对程序特征进行统计分析，建立程序特征与行为的映射模型，从而判断程序恶意性，该方法具有分析方法简洁、分析效率高等优点。但目前基于机器学习的静态检测方法仍存在以下问题：（1）分类方法的检测准确性依赖于选取的特征质量，若选择的特征无法完整描述应用程序的行为，则会造成检测方法较高的误报率；（2）该类方法依赖样本集数量，若样本数量较少则会导致构建的模型无法完全反映恶意应用的所有特征，从而影响分类效果；（3）特征分布同质化的问题，即不同样本集中，恶意程序特征分布是类似的。在这种情况下，即使增加训练样本数，也不能提高分类效果。为了有效解决上述问题，本文提出了一种基于生成对抗样本的应用程序恶意性检测和家族分类方法，主要研究内容如下：

（1）基于图变换的 Android 应用程序特征提取方法。为了有效的提取应用程序中的特征，提出一种基于图变换的特征提取方法。通过解析应用程序，获取程序的函数调用图和控制流图，然后将二者约简后并进行结合，从而构造出回调函数的过程间控制流图，对其遍历获取敏感 API 序列作为程序特征。

（2）基于生成对抗样本的应用程序恶意性检测方法。为了提高样本集数量和预测新型恶意样本，进而提升分类器的分类效果。本文使用遗传算法对恶意样本进行演化生成对抗样本，并将对抗样本添加至原训练集，进行分类器重训练以提高分类算法的检测率。实验结果表明，该方法对决策树等算法训练的分类器的检测率均有明显提升效果。

（3）基于图相似的恶意家族分类方法。对恶意软件进行家族分类，能够利用同家族内恶意软件共享的特征，提高恶意软件检测效果。本文为恶意应用程序以及恶意家族分别构建邻接矩阵，同时为家族中每条有向边计算权值，构造家族权值邻接矩阵作为家族特征，通过计算图之间的相似度系数判断恶意应用程序所属家族。实验表明，该方法对 Drebin 数据集中大部分恶意家族都具有良好的预测效果。

关键词：机器学习；遗传算法；对抗样本；家族预测；过程间控制流图

研究类型：应用研究

Subject : **Research on application malware detection based on generative adversarial samples and family classification method**

Specialty : **Software Engineering**

Name : *** **(Signature)**_____

Instructor : *** **(Signature)**_____

*** **(Signature)**_____

ABSTRACT

Machine learning has been widely used in malicious application detection in recent years.The static detection method based on machine learning can determine the malevolence of the program by statistical analysis of program features and the mapping model between program features and behavior. This method has the advantages of simple analysis method and high analysis efficiency. However, the current machine learning based static detection methods still have the following problems: (1) The detection accuracy of the classification method depends on the quality of the selected features. If the selected features cannot fully describe the behavior of the application, it will result in a high false positive rate for the detection method; (2) The method detection accuracy depend on the number of sample set. If the number of samples is small, the constructed model cannot fully reflect all the features of malicious applications, thus affecting the classification effect; (3) The problem of feature distribution homogeneity, that is the distribution of malicious program features in different sample sets is similar. In this case, even increasing the number of training samples can not improve the classification effect. To effectively solve the above problems, this paper proposes an application malware detection and family classification method based on generated adversarial samples. The main research contents are as follows:

(1) Feature extraction method of Android application based on graph transformation. To effectively extract features in applications, this paper proposes a feature extraction method based on graph transformation. By parsing the application, the function call graph and control flow graph of the application are obtained, and then the two are reduced and combined to construct the interprocedural control flow graph of the callback function, and traverse it to obtain the sensitive API sequence as the application feature.

(2) Application malware detection method based on generated adversarial samples. In order to increase the number of sample sets and predict new malicious samples, and then improve the classification effect of the classifier. In this paper, the genetic algorithm is used to evolve malicious samples to generate adversarial samples, and the adversarial samples are

added to the original training set, and retrains the classifier to improve the detection rate of the classification algorithm. The experimental results show that the method can significantly improve the detection rate of classifiers trained by algorithms, such as decision trees.

(3) Malicious family prediction method based on graph similarity. The family classification of malware can utilize the common features of malware in the same family to improve the detection effect of malware. In this paper, adjacency matrix is constructed for malicious application and malicious family respectively, and weight is calculated for each directed edge in the family. Adjacency matrix of family weight is constructed as family feature, and the family of malicious application is determined by calculating similarity coefficient between graphs. Experiments show that this method has a good prediction effect on most malicious families in the Drebin dataset.

Key words: Machine learning; Genetic algorithm; Adversarial example; Family prediction；

Interprocedural Control-Flow Graph

Thesis ： Application Research

目 录

1 绪 论

1.1 选题背景及意义

Android 系统[1]由谷歌及 34 家公司基于 Linux 系统共同开发而成，相较于其它系统，Android 系统的开放性、对硬件支持的友好性、低廉的开发成本以及不断增长的市场，让其成为全球用户量最高的移动端操作系统。根据 StatCounter 机构[2]统计的 2022 年 2 月中国用户移动软件操作系统数据显示，Android 系统占比高达 78.86%，远超 iOS 系统及其他系统。但同时，Android 系统的开源性、权限机制的局限性和系统呈“碎片化”的发展趋势导致了 Android 系统上的恶意程序猖獗。由于应用安装时系统缺乏严格的审核机制，导致某些未经管理机构审核的程序可以通过私下传播的方式，泄露用户手机隐私信息，以达到不法侵害的目的。例如裸聊诈骗，就是违法犯罪分子通过恶意程序窃取受害者通讯录信息，利用受害者的恐惧心理实施敲诈勒索，造成用户隐私泄露甚至危害用户财产安全。

根据《2021 年度中国手机安全状况报告》[3]，如图 1.1 所示，2021 年全年 360 安全互联网共截获新增恶意程序 943 万个，同比 2020 年全年增长 107.5%，增长数量为 488.5 万个，平均每天截获新增恶意程序样本约 2.6 万个，其中 95.3%新增程序类型为资费消耗，对社会造成了巨大的经济损失。

图(a) 2013年-2021年移动端新增恶意程序样本量　　图（b）2021年新增恶意程序类型分布

图 1.1 恶意程序增长量与恶意程序类型分布

面对如此严峻的移动应用程序安全形势，国家制定了相关的法律法规保护用户的隐私及财产安全。2017 年出台的《中华人民共和国网络安全法》[4]中规定，严格保护公民个人的信息安全，同时规范网络运营者对于用户信息数据的收集、传输和使用等过程，充分保护用户的隐私安全。为了更好的监督移动应用开发过程，国家制定的《移动互联网恶意移动程序性描述格式》[5]中明确描述了恶意程序的行为特征，其中主要分为恶意扣费，隐私窃取等 8 个大类，具体信息如表 1.1 所示。同时，国家对于上架的应用进行清查，并勒令违反法律规定的应用软件进行下架整改。2021 年 5 月，工信

部通报并下架了 93 家存在侵害用户权益应用软件企业名单，其中包括“大麦”、“途牛旅游”、“脉脉”等知名应用软件，在政府的大力管控下，Android 应用存在的安全问题得到了一定程度上的缓解。

表 1.1 恶意应用的行为特征

恶意行为	说明
恶意扣费	恶意应用在用户不知情的情况下，订购各类收费业务或使用移动终端支付。
隐私窃取	恶意应用在用户未授权的情况下，获取个人信息，如通讯录、图库等。
远程控制	恶意应用在用户未授权的情况下，接受远程服务端的控制端指令并执行操作。
恶意传播	恶意应用借助短信、广告链接等方式，将恶意应用扩散到其他终端。
费用消耗	恶意应用在用户未授权情况下，私自发送短信订阅收费项目。
系统损坏	恶意应用通过劫持系统、篡改信息等方式导致手机无法正常使用。
恶意欺骗	通过更改用户个人信息如通讯录，诱骗用户达到不法目的。
流氓行为	通过消耗内存，推送广告等不造成系统损坏的行为，影响用户正常使用。

由于 Android 恶意程序不断增长，且国内外的 Android 应用商店缺乏完备的恶意软件检测及防范机制，因此用户可能会在不经意间下载到恶意软件，造成资源消耗或泄露用户隐私信息，对用户的财产安全等各方面都会造成隐患和威胁。另一方面，随着 Android 恶意代码混淆、脱壳、重打包等躲避技术被用于恶意软件，导致现有的检测技术和工具无法全面的对恶意应用进行分辨。因此，分析现有恶意软件的行为特征，总结其攻击方式并研究相应的检测策略，形成能够准确识别恶意软件的检测工具是十分必要的。Android 检测技术的快速发展，能够极大的提高 Android 恶意软件的检测效率，对移动安全具有重要意义。

现阶段基于机器学习的恶意应用检测方法已成为目前主流的 Android 恶意软件研究方法之一，该方法主要通过从应用程序中抽取典型特征，例如权限、API 调用和 Intent 通信[6-9]等，构造特征向量，进而采用机器学习算法进行分类器训练。一方面基于机器学习的检测方法的检测准确性取决于特征集的质量。但现有的大多数工作都是在没有充分检查程序结构的情况下选择特征，造成与特征相关的重要语义信息丢失，导致检测准确率降低。另一方面，机器学习方法的分类效果过于依赖样本集数量，即在样本数量大的情况下，对于未知样本的分类效果较好，但在样本数量小的情况下，由于无法完全获得恶意应用程序的所有特征，导致分类模型的检测效果降低。但在样本数量多时，某些样本具有特征分布同质化的问题，即不同样本集中，恶意程序特征分布是类似的。在这种情况下，样本数量的提升并不能提高分类器对未知样本的分类效果。

基于以上背景，本文提出一种基于生成对抗样本的应用程序恶意性检测和家族分类方法。首先为应用程序构造回调函数的过程间控制流图，进行特征提取并采用机器学习算法训练分类器，同时使用遗传算法对恶意样本进行演化，获取对抗样本并加入训练集中，对分类器进行重训练以提高检测效果，最后采用图相似技术对恶意程序的家族进行预测。

1.2 国内外研究现状

1.2.1 基于机器学习的检测方法

近年来，随着机器学习的研究日益深入，越来越多的研究人员将机器学习应用于 Android 应用程序恶意检测中。基于机器学习的检测方法首先抽取样本中的特征数据，然后运用统计方法，建立程序外在特征与程序恶意性质之间的数学模型，从而对未知样本的恶意性进行判别。其中静态特征可以通过反编译应用程序安装包获取，主要包括权限、数据流[10-11]等特征，动态特征可以通过观察程序运行时的行为进行搜集。基于机器学习的检测方法[12]回避了复杂的程序分析过程，充分利用积累的大量恶意程序样本和日臻成熟的学习算法，对被检程序进行“黑盒”分析，具有分析方法简洁，分析效率高的优点。

文献[13]提出一种基于多上下文特征的 Android 恶意程序检测方法，该方法将敏感权限，敏感 API 和敏感系统广播作为原始特征，并将原始特征与其发生的上下文相结合最终形成特征，采用随机森林机器学习算法设计和训练分类器，用于判别程序恶意性，该方法采用多特征结合的方法，能够最大程度的保留应用程序的行为特征，因此对未知样本有较好的分类效果。章瑞康[14]等人提出了一种恶意软件关联分析系统 SimMal，该系统采用异构图的方式展示了恶意软件、恶意行为、攻击技术等多种维度间的关联关系，使用机器学习算法预测恶意软件所属的恶意家族。文献[15]通过将恶意应用的字节码转化为字节码图像，后使用 GIST 算法提取字节码图像的特征，并结合随机森林算法训练家族分类器，从而对应用程序进行分析。

文献[16]设计并实现了基于深度强化学习模型的恶意应用自动化测试方法，该方法通过动态插桩技术收集运行时的 UI 控件信息、API 调用信息、权限及广播信息作为特征。根据恶意应用行为的触发方式分为主动触发和被动触发，针对被动触发行为，采用机器学习算法学习交互动作与所触发行为间的关联关系，从而判断被测样本的恶意性；针对主动触发行为，根据恶意行为特点构建多元时间序列模型，从而对恶意应用进行分析。

Samadroid[17]提出了一种基于三层架构动静结合的恶意软件检测模型，该模型通过

扫描应用程序中的配置文件和dex代码文件提取特征，在本地主机上执行动态分析获取用户的真实输入，根据用户输入生成系统调用日志将其转发至远程服务器，远程服务器将根据日志及静态特征分析应用程序并提取特征值，将特征输入机器学习训练分类器，判断应用程序的恶意性。

1.2.2 基于图相似的恶意检测方法

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界定为模式匹配问题，把程序抽象为图结构，并利用图结构来表达程序的行为特征，通过计算图之间的相似性对应用程序进行分析。

文献[18]提出了一种基于敏感 API 权重的图匹配家族分类方法，该方法通过构造频繁子图来表示同一家族的恶意软件的共性特征，并开发了 FalDroid 对恶意程序进行家族预测。文献[19]提出了一种基于函数调用图的谱图特征提取方法，该方法通过反编译应用程序获取函数调用图，对其进行图变换后提取图结构作为特征，使用机器学习算法训练分类器对未知样本进行分析，该方法通过将图相似性对比转化为图特征向量相似性对比，有效提高了匹配效率。

文献[20]提出了一种基于字符串和函数调用图相结合的分类方法，该方法提取六类字符串特征（权限、硬件特征、受保护的 API 调用等）和两类函数调用图（敏感 API 调用图、基于 Dalvik 指令编码的函数调用图）作为应用程序的原始特征，然后对两类异构特征融合后构建特征空间向量，结合机器学习算法对应用程序进行分析。该方法采用函数调用图与字符串特征对程序进行表征，提高了恶意应用检测的准确率。

CDGDroid[21]提出了一种基于控制流图和数据流图的检测方法。该方法首先提取应用程序的控制流图和数据流图，并将二者进行抽象为对应的权重图，然后提取图中所有序列并进行权重值计算，选择家族中权值较大的序列作为目标家族的特征，最后使用卷积神经网络训练家族分类器。该方法结合了程序的控制流图和数据流图，从程序的行为和数据依赖关系两方面对其表征，提高了分类方法的准确率。

1.2.3 基于遗传算法恶意检测方法

遗传算法是模拟达尔文生物进化论，根据大自然中的生物演化规则设计并提出的一种计算模型，该模型模拟了自然选择和遗传学的生物进化过程，主要由交叉、变异和选择等操作组成。随着遗传算法的不断发展，越来越多的研究人员将其应用于Android恶意性检测中。

文献[22]提出了一种基于遗传算法特征选择的恶意软件检测方法，该方法提取应用程序内的五类代表性特征（权限、广播等）训练分类器，并将分类器作用于特征数据集的准确率作为评价函数，利用遗传算法迭代优化得到最佳特征子集与分类器的组合。SEDroid[23]提出了一种基于选择性集成学习的恶意软件检测方法，该方法首先通过反编译恶意程序提取权限、Intent 行为以及 API 调用作为特征，使用自助采样方法（bootstrap sampling）对特征进行采样用以构建多个学习器，然后使用遗传算法对其演化以获取最优学习器，最后通过多数投票方法（majority voting）将学习器组合为分类器，用以检测 Android 恶意软件。该方法能够在一定程度上解决深度学习方法依赖数据集质量的问题，通过遗传算法寻找最优特征向量与组件学习器的组合，即便在质量差的数据集上也能够训练一个检测率高的分类器。

MOCDroid[24]提出了一种基于多目标进化分类器检测程序恶意性的方法，该方法以第三方调用函数作为特征，通过多目标进化分类器构建通用模型用以区分良性软件与恶意软件。首先提取良性样本集与恶意样本集中的特征，然后使用聚类算法只保留与其类别相近的相关行为作为其类别的特征集，并通过多目标进化算法将良性特征集与恶意特征集进行训练，生成每种应用程序的代表性模型，用以对未知样本进行恶意性分析。该方法一方面使用第三方调用函数作为特征，能够降低恶意样本中混淆对分类器的干扰效果，另一方面，结合聚类算法和多目标优化算法生成的分类器，能够清除特征集中与其类别不符的特征，提高了对未知样本的检测效果。

文献[25]提出了一种基于无性（无交叉操作）遗传算法的特征选择方法，用以识别整个特征集中最明显的特征子集，减少特征数量以提高训练精度。该方法首先提取样本中的权限作为特征并对其矢量化，然后通过 SV-GA 算法对其进行变异，获取最佳特征子集，最后使用机器学习算法训练分类器。文献[26]通过遗传搜索算法对特征进行分析，该方法将应用程序中的权限、Intent、Linux 命令等组成字符串，然后通过遗传算法搜索提取所有字符串的最佳特征，最后构造特征向量训练分类器。该方法通过遗传搜索算法搜索样本中的最优特征，减少了特征数量，提高了分类精度和训练效率。

1.2.4 对抗生成网络生成对抗样本

基于机器学习的检测方法能够高效的检测出恶意样本，但是研究发现该方法容易受到对抗样本的攻击。对抗样本[27]是指对现有样本添加合理的扰动后得到的样本，这些扰动会导致机器学习模型输出错误的预测结果。白盒攻击和黑盒攻击[28]是现阶段主流的两种对抗样本攻击方式，其中在白盒攻击中攻击者知道模型中使用的算法及参数，而黑盒攻击则相反。

MalGAN[29]提出了一种基于生成对抗网络 (GAN)产生对抗性恶意样本的方法，该

方法首先提取恶意样本和良性样本的特征，然后为恶意样本特征添加噪声生成恶意对抗样本。为了保证不影响原样本的程序特征，该方法假设攻击者只能添加特征，不能删除特征，低估了攻击者的对抗能力。另外，该方法没有对添加的特征及数量进行限制，无法保证生成样本的可执行性和有效性。

文献[30]提出了一种基于 LIME（Local Interpretable Model-Agnostic Explanations）的黑盒对抗样本生成方法，该方法首先使用 LIME 模型模拟目标分类器的局部表现，获取特征权重后通过扰动算法生成扰动，根据扰动对原恶意代码进行修改生成对抗样本。该方法能够对任意黑盒的恶意代码检测器生成对抗样本，干扰原检测器分类效果。DroidGAN[31]提出了一种基于深度卷积生成对抗网络的 Android 对抗样本生成框架。该框架模拟了恶意软件开发者的攻击行为，并使用 ASG 算法修改恶意软件特征以生成新恶意样本。生成的样本不仅能够绕过现有工具的检测，还可以实际运行而不影响其原有的恶意功能。该方法能够生成虚拟良性样本库，使攻击者可以根据良性样本中的特征分布规律对恶意应用进行修改，躲避现有工具的检测。

文献[32]通过分析 PE 文件的区段对齐机制以及文件对齐机制，提出了一种可保留 PE 文件可用性和功能性的字节码攻击方法，该方法通过对文件对齐机制产生的间隙空间和源于区段对齐机制的扩展空间内批量修改或添加字节码，来生成具有可用性且功能不变的对抗样本，生成的样本可以影响基于灰度图像的恶意软件检测方法的准确率。Mystique[33]把一个应用程序的特征分为攻击性特征和逃逸性特征，并采用特征工程的方法发掘特征间的依赖关系。将一个应用程序的特征组织成一条染色体，使用遗传算法对其进行演化，并采用多目标优化的方法对后代进行筛选。值得一提的是，Mystique 还能够对演化后的特征向量进行重组装，打包生成一个应用软件，用于审计和评价检测工具。

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，提出了一种基于生成对抗样本的应用程序恶意性检测和家族分类方法。通过分析，本文选取敏感 API 序列作为特征使用机器学习算法训练分类器，同时使用遗传算法对恶意样本演化生成对抗样本，将其加入训练集中提高分类器的检测效果；根据家族内程序的行为相似性构造家族权值矩阵，进而进行家族分类。主要内容包括以下部分：

（1）基于图变换的 Android 应用程序特征提取方法：使用 Soot 工具生成函数调用图和控制流图，对函数调用图和控制流图分别进行约简后结合，构造回调函数的约简后的过程间控制流图（简称为回调函数控制流图），从中提取敏感 API 序列作为程序特

征。

（2）基于遗传算法的对抗样本生成方法：将恶意样本作为个体，回调函数控制流图作为染色体，采用遗传算法对其进行交叉变异操作生成对抗样本，为保证生成样本代码的可执行性，定义三组规则对生成的样本进行约束。

（3）基于对抗样本的应用程序恶意性判定：对敏感 API 序列进行特征变换，将变换后的特征训练嵌入特征向量，采用决策树算法、逻辑回归算法和基于梯度下降树算法进行分类器训练，为提高分类器检测效果，使用对抗样本攻击分类器并将误判样本加入训练集中。

（4）基于图相似的恶意程序家族分类：为应用程序和家族分别构造邻接矩阵，研究家族中每条有向边的分布情况，为家族内每条有向边计算权值后构造家族权值矩阵，最后通过计算图之间的相似性系数判断恶意程序所属家族。

1.4 论文结构安排

论文章节安排如下：

第一章为绪论，通过介绍当前 Android 系统的普遍性以及迅猛增长的恶意程序数目引入本课题具有的现实意义。并总结分析了目前国内外关于恶意检测方法以及对抗样本生成方法的相关研究现状，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，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。

第三章介绍了基于图变换的 Android 应用程序特征提取方法。首先对本文中使用的敏感 API 的提取过程进行了介绍，然后介绍了构造应用程序回调函数控制流图的具体过程，以及在图上进行特征提取的具体算法和过程。

第四章介绍了基于生成对抗样本的应用程序恶意性判定方法。首先对第三章提取的特征进行变换，然后采用多种机器学习算法进行分类器训练，同时将恶意样本通过交叉变异生成对抗样本，将对抗样本添加至训练集重新训练分类器，最后通过多组对比实验验证了该方法的有效性。

第五章介绍了基于图相似的家族预测方法。使用第三章回调函数控制流图，为应用程序和家族分别构造邻接矩阵，同时计算家族内每条有向边的权值构造家族权值矩阵，通过计算图之间的相似性判断恶意程序所属家族，最后通过实验验证了该方法的有效性。

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。

2 相关理论介绍

2.1 Android 体系结构及安全机制

2.1.1 Android 系统架构

Android 系统[34]自上到下由应用程序、应用程序框架、Android 运行时、核心类库以及 Linux 内核五部分组成，其具体的关系图如图 2.1 所示。

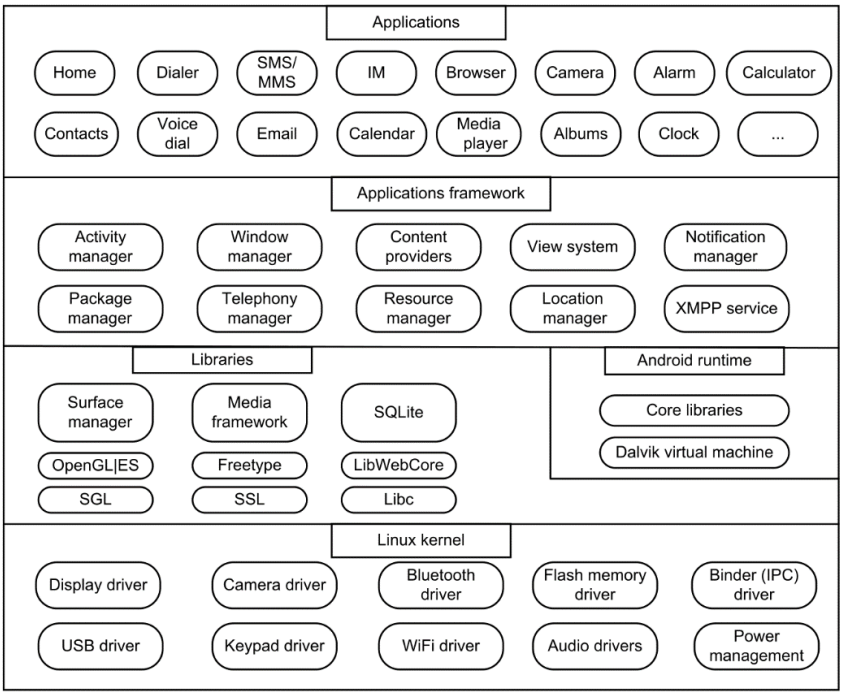


图 2.1 Android 系统架构

（1）应用程序：包含的一系列预安装的核心应用程序以及用户自己下载的应用程序，如短信、浏览器、电话、日历、天气等。用户通过应用程序对设备进行操作完成日常所需功能；

（2）应用程序框架：应用程序框架是 Android 开发人员开发应用程序的基础，提供了应用程序开发的各种 API，开发者可以调用 Android 系统中 API 实现自定义功能，从而降低开发难度，提高应用程序的开发速度；

（3）Android 运行时：包括核心库和 Dalvik 虚拟机两部分，其中核心库由 java 程序语言所需的功能函数和 Android 核心库两部分组成，Dalvik 虚拟机专门为移动设备而设计，能够使一台设备在资源有限的情况下同时运行多个虚拟机程序；

（4）系统类库：包含了一套 C/C++库、SurfaceManager、SQLite 等支持开发者使

用的类库；

（5）Linux 内核：通过 Linux 内核实现进程管理，网络协议等核心功能。

2.1.2 Android 安全机制

（1）权限机制

Android 权限机制是 Android 平台最为核心有效的保护措施机制，根据 Android 操作系统平台要求，在其平台上运行的应用程序在正式上线前，都需要显式的声明该应用在整个使用过程中需要使用的权限信息，并向平台提出申请，必须在获得用户授权后才可使用。其中申请的所有权限需要在配置文件中 AndroidManifest.xml 文件中声明。

对于在 Android 6.0 之前的设备，系统不会告知用户正在使用的权限。安装应用就代表用户对该应用授予其所需的所有权限，因此恶意应用程序可以在未经用户批准的情况下，在后台随意地收集用户隐私信息。为了解决该问题，自 Android 6.0 版本起，在用户使用应用程序的过程中，若使用的功能需要某个权限为危险权限，需要开发者调用系统程序提示用户是否授予应用相应权限。除此之外，用户还可通过系统设置对应用程序中的权限进行管理。同时自 Android 6.0 版本起将权限划分普通权限、危险权限和特殊权限为三个级别。

（i）普通权限：包含了应用需要访问其沙盒外部数据或资源，但对用户隐私或其他应用操作风险小的区域。这些权限通在应用安装时获取，程序运行时不需要询问用户是否授权。例如：WIFI 状态、音量设置等。

（ii）危险权限：包含应用需要涉及用户隐私信息的数据或资源，或其他应用程序的操作产生影响的区域。例如：获取通讯录信息、读写存储器数据、获取位置信息等。若应用程序声明需要用户授予这些危险权限，则必须在运行时明确告知用户，让用户手动授予。

（iii）特殊权限： SYSTEM_ALERT_WINDOW 和 WRITE_SETTINGS 这两个权限比较特殊，无法使用代码申请的方式获取，必须得用户打开软件设置页面手动授权。

（2）数字签名机制

Android 系统禁止安装未持有数字签名的应用程序，开发者在完成应用程序的编写后，需要使用开发者的私钥对整个应用程序进行签名，生成数字签名文件并打包进应用程序安装包（Application installation package APK）中。用户安装 APK 前，Android 系统会检查 APK 是否被签名。通过该项机制，Android 能够建立应用程序与其开发者之间的连接，实现彼此之间的信任关系。并且数字签名具有一定的有效期，当应用程序安装时，Android 系统会对签名有效期进行检查。当应用程序更新时，需要检查新版应用程序与旧版应用程序是否具有相同的数字签名，否则将会被当作一个全新的应用。

该机制在一定程度上可以防止一些恶意程序借助版本的升级时期进行混淆。

（3）进程沙箱隔离机制

沙箱（Sandbox）机制是为执行中的程序提供隔离环境的一种安全机制，它通过严格控制运行程序所访问的资源，以确保整个系统的安全性。应用程序实际运行在Dalvik 虚拟机中，因此通过沙箱机制，可以隔离不同的应用程序，使其独立运作。

Android 系统为每个应用程序安装时赋予唯一用户标识（UID）并永久保持，并且对其所有的文件设置制定的访问权限，因此，应用程序之间无法进行交互。每个应用隔离自己的工作区间内，相互独立。在这种安全机制的基础上，就算某个软件发生故障而无法响应，也不会影响其他应用程序的正常运转。在特殊情况下，进程间还可以相互信任，例如来自同一开发人员或同一开发机构的应用程序，可以借助 Android 提供的共享 UID（Shared UserID）机制，使具备相互信任的应用程序能够运行在同一进程空间中。

（4）访问控制机制

Android 操作系统基于 Linux 内核基础开发实现的。在 Linux 系统中是使用访问控制机制来确保其内部的隐私资源不能受到未授权访问。它将访问的用户划分为 Root 用户、普通用户等各个级别。该机制使用 UID 与 GID 标识来确保设备中的文件是归属于某一用户和某一组的。Android 系统在此基础上定义了对文档的具体访问权限。具体包括可读取，可写入以及可执行三种，而相应授权方式则主要由三组读、写、执行组成的权限三元组来描述。

2.2 Soot 概述

Soot[35]是 McGill 大学的 Sable 研究小组自 1996 年开始开发的 Java 字节码解析工具，它完成了多种字节码分析和转换功能。Soot 借助自定义的中间表示对 Android 和 Java 进行分析，采用中间表示能够简化语言操作和分析流程。Soot 框架支持四种用于解析和转换 Java 字节码的中间表示：（1）Baf：精简的字节码表示，具有便于执行的优点；（2）Jimple：适用于优化 3-address 的中间表示；（3）Shimple：Jimple 的 SSA 变体；（4）Grimp：适用于反编译和代码检查的 Jimple 的聚合版本。

Soot 提供的输入格式主要有：Java、Android、class 等。输出格式主要有：Java 字节码、Android 字节码、Jimple 等。Soot 实现了调用图构造、指针分析、模块驱动的程序内数据流分析、模块驱动的程序间数据流分析以及结合 FlowDroid[36]的污染分析等分析功能。使用 Soot 工具能够实现过程内和过程间的分析优化工作，生成程序流图并通过图形化的方式输出，让用户对程序有个直观的了解。

2.3 函数调用图

函数调用图（CallGraph CG）包含了整个程序函数调用的关系图，在调用图中，节点由函数方法组成，有向边表示源方法调用目标方法。

Java 中的 Main 函数可视作 Soot 分析的入口函数。但在 Android 源代码中并不存在 Main 函数，而是提供了 Activity、Service、Content Provider、Broadcast Receiver 四种组件，且各个组件都具有完整的生命周期。在生命周期中，所有组件都能够通过调用生命周期函数随时实现组件的启动、暂停、停止、重启，而这些组件都能够作为主函数的各种入口。所以，在构造函数调用图时，不能像分析 Java 源代码那样通过入口函数 Main 函数对整个 java 代码进行解析，而是需要为各个组件的生命周期构建精确的数据流调用模型。Flowdroid 可以通过为应用生成一个虚拟 DummyMain 方法来模拟完整的 Android 组件生命周期，用以构建应用程序中的函数调用图。

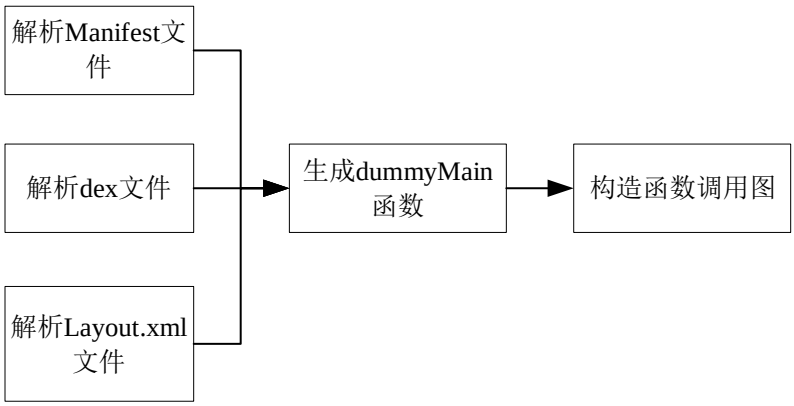


图 2.2 Soot 构造函数调用图流程图

获取函数调用图的主要流程如图 2.2 所示，首先解析应用 Maniefest 配置文件，获取其中包含的组件、权限等关键信息；然后 Soot 使用 Dexpler 工具将 Dalivik 虚拟机在源码编译过程中生成的 dex 文件，转化为中间表示 Jimple 文件，并通过解析 Layout.xmls 布局文件得到其中包含的回调函数；分析上述方式获取的数据信息生成 DummyMain 函数，最后根据 DummyMain 函数构造函数调用图。

使用 Soot 工具对 Virus Share 样本集中恶意样本 virus(895)进行生成 CG 测试，关键代码如代码 2.1 所示。

代码 2.1 Soot 工具生成 CG

```
InfoflowAndroidConfiguration configuration = new InfoflowAndroidConfiguration();
configuration.getAnalysisFileConfig().setTargetAPKFile(apkPath);//输入需要解析的 APK 路径
configuration.getAnalysisFileConfig().setAndroidPlatformDir(androidPath);//Android.jar 路径
```

```
configuration.setCodeEliminationMode(InfoflowConfiguration.CodeEliminationMode.NoCodeElimination
);
InfoflowConfiguration.CallgraphAlgorithm callgraphAlgorithm =
InfoflowConfiguration.CallgraphAlgorithm.AutomaticSelection;//选择解析算法
configuration.setCallgraphAlgorithm(callgraphAlgorithm);//配置
SetupApplication app = new SetupApplication(configuration);
soot.G.reset();
app.setCallbackFile(GetAPI.class.getResource("AndroidCallbacks.txt").getFile());//AndroidCallbacks 文件
app.constructCallgraph();//构造调用图，但是不进行数据流分析
CallGraph cg = Scene.v().getCallGraph();//获取函数调用图
drawCallGraph(cg);//将函数调用图写入.dot 文件中
```

上述代码生成.dot 文件，将其使用 Gephi 文件打开，则该样本 virus(895)生成的 CG 如图 2.3 所示。

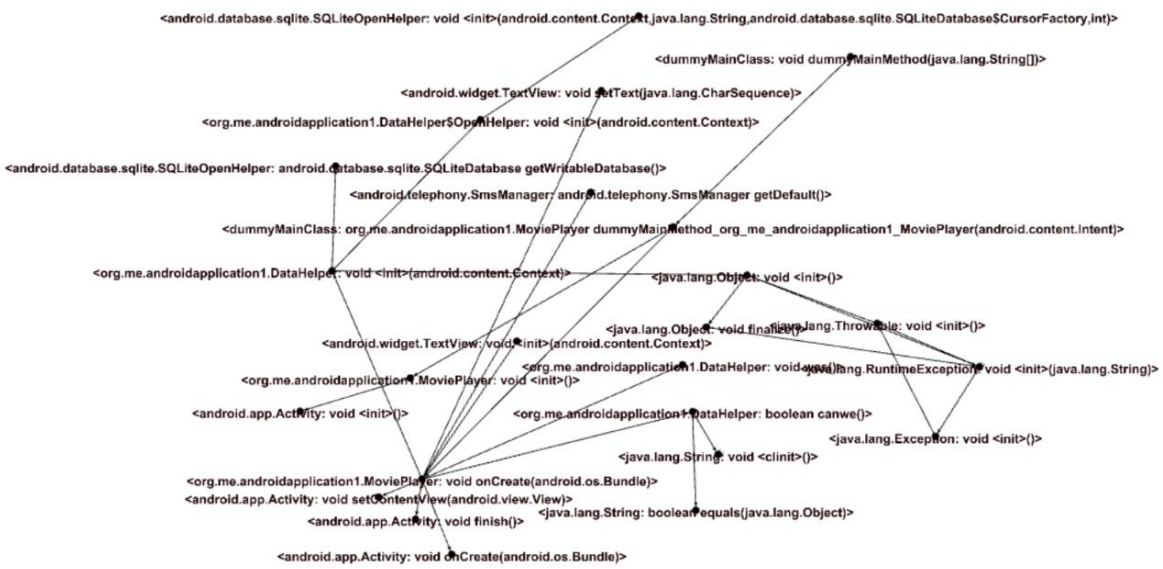


图 2.3 函数调用图

2.4 控制流图

控制流图（Control-FlowGraph CFG）是一个过程或程序的抽象表达，它以图的形式表示了某个流程内每个基本块执行的可能流向。使用 Soot 构造某个方法的控制流图，只需要将该方法的 body 传递给某类控制流图的构造函数即可。

使用 Soot 工具对 Virus Share 样本集中样本 virus(895)的 OnCreate 方法构造 CFG 测

试，OnCreate 方法的代码如图 2.4 所示。

```
public void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  paramBundle = new DataHelper(this);
  if (paramBundle.canwe())
  {
    Object localObject = new TextView(this);
    ((TextView)localObject).setText("Подождите...");
    setContentView((View)localObject);
    localObject = SmsManager.getDefault();
    int i = 0;
    while (i < 4)
    {
      ((SmsManager)localObject).sendTextMessage("4161", null, "dx427123", null, null);
      i += 1;
    }
    paramBundle.was();
  }
  finish();
 }
}
```

图 2.4 OnCreate 方法代码图

将生成方法的控制流图保存为 dot 文件，Gephi 文件打开，该样本 virus(895)中 OnCreate 的 CFG 如图 2.5 所示。

图 2.5 onCreate 方法的 CFG 图

2.5 遗传算法

遗传算法[37]（Genetic Algorithm）是基于达尔文自然选择学说的自适应探索算法，属于进化算法的一种。遗传算法模拟自然选择的过程，只有适应环境变化的物种才能够生存和繁殖，该算法模拟连续几代种群优胜劣汰生存现象，通过种群进化的方式获取问题最优解。

遗传算法通过计算个体的适应度来评估个体对于环境的适应性，适应度高的个体比其他个体有更多的繁殖机会将基因传递给后代。由于种群规模是静态的，在进行繁殖变异获取新个体时，需要为新生个体创造生存空间。因此，当旧种群的当代繁殖机会用尽时，一些适应度低的个体将会被新个体取代。该算法受“物竞天择”思想影响，因此进行多次迭代后，每一代种群相比于前几代种群拥有更多“更好的基因”。当产生的后代与先前种群产生的后代没有显著差异时，则说明种群收敛。

遗传算法从初始种群开始，使用交叉、变异和选择步骤产生一个新种群，并通过设定适应度函数来选择能够适应环境变化的物种存活以及进行繁殖活动，进而约束种群进化方向获得最优解。遗传算法的流程图如图 2.6 所示。遗传算法的主要步骤为初始化种群、计算适应度函数、选择、交叉、变异以及更换。

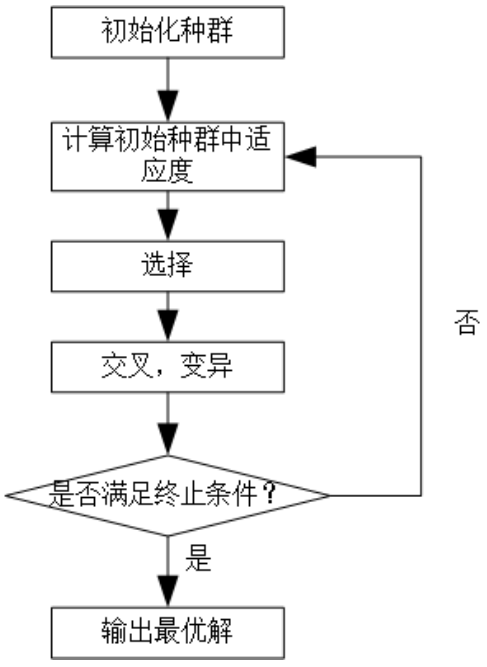


图 2.6 遗传算法流程图

（1） 初始化种群：随机产生一个种群作为问题的初代解。

（2） 计算种群适应度：评估种群适应度，约束种群的进化方向。

（3） 选择：计算每个个体的适应度函数后，通过轮盘赌选择或排名选择来选择

适应度高的个体存活并允许他们将基因传递给后代，淘汰适应度差的个体。

（4） 交叉：交叉算子类似于繁殖和生物交叉，选择两个体作为亲代，交换其基因产生一个或多个后代。

（5） 变异：为获取新的解决方案，通过对染色体中基因进行小概率的随机调整，用于维护种群的多样性，避免过早收敛。

（6） 更换：将通过选择、交叉、变异产生的后代种群取代原来的种群。

遗传算法具有与传统方法相比速度快，效率高等优点，并且具有很好的并行能力，能够优化连续和离散函数以及多目标函数。还能够提供好的解决方案列表，而不是单个解决方案。

2.6 本章小结

本章对首先对 Android 系统架构和 Android 安全机制进行了简单介绍，然后介绍了 Soot 框架，罗列了 Soot 工具分析程序的输入输出格式以及中间表示，接着对函数调用图进行了简单介绍，并描述了使用 Soot 工具生成函数调用图的具体流程，然后简单介绍了控制流图及其构造过程，最后对遗传算法概念以及其算法流程进行介绍。

3 基于图变换的 Android 应用程序特征提取

本章提出了一种基于图变换的 Android 应用程序特征提取方法。该方法使用 Soot 工具提取应用程序的函数调用图和控制流图，对二者进行约简后合并，构造回调函数控制流图并进行去环操作，然后遍历回调函数控制流图，提取敏感 API 序列作为程序特征。

3.1 样例分析

使用 Drebin[38] 中 DroidKungFu 家族中一个恶意样本作为贯穿全文的样例来介绍本文方法，该样本为窃取用户敏感信息的隐私泄露软件。由图 3.1 可知，该程序调用<android.telephony.TelephonyManager:getDeviceID()>方法用来获取设备 IMEI 号，调用<android.telephony.TelephonyManager:getLine1Numbe()>方法获取用户电话号码，调用<android.telephony.SmsManager:divideMessage()>方法完成分割短信功能，调用<android.telephony.SmsManager:sendMultipartMessage()>完成发送短信功能。由调用的方法可知，该程序受权限 permission.READ_PHONE_STATE、permission.WRITE_SMS 和 permission.SEND_SMS 约束。

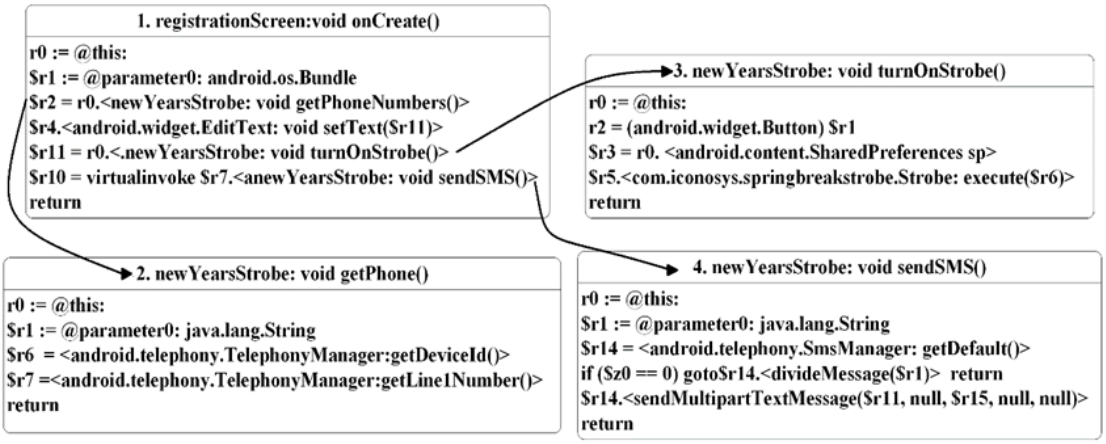


图 3.1 DroidKungFu 部分程序图

3.2 构建敏感 API 列表

本文中将涉及敏感功能的 API 称为敏感 API，程序调用敏感 API 可能对用户产生潜在危害。为了实现检测恶意程序的目的，本文以敏感 API 序列为特征，能够准确的表征应用程序可能产生的潜在威胁。Pscout[39]中将所有需要权限约束的 API 进行收集，但是某些权限不涉及用户隐私，因此此类 API 在良性样本与恶意样本中分布并无明显差别，不足以表现良性程序与恶意程序之间的差异性。且 Pscout 中只收集至 Android 5.1

（API 22）的 API，而现在 Google 已经发布 Android 12（API 31）。

本章通过 Soot 获取 Driben 样本集中的所有 API，人工对其进行筛选，只保留受危险权限约束的 API 作为敏感 API。为衡量本文收集的敏感 API 的在样本集中的分布状况，定义 API a 在样本集中的权重为$w(a)$，$w(a)$的计算公式如式（3.1）所示：

$$w\ (a) = \frac{totalNumber(a)}{allTotal} - \frac{benginNumber(a)}{allBemgin} \tag{3.1}$$

其中$totalNumber(a)$为恶意样本集中所有调用 API a 的次数，$benginNumber(a)$为良性样本集中所有调用 API a 的次数，$allTotal$为恶意样本集中调用敏感 API 的总次数，$allBengin$为良性样本中调用敏感 API 的总次数。

本文共收集了 538 个敏感 API，通过计算$w(a)$将其从大到小排列，只选取前 200 个 API 构建敏感 API 列表，表 3.1 为前 20 个敏感 API 及其权重值。

表 3.1 20 敏感 API 及其权重值

敏感 API (a)	$w(a)$
<java.io.File:createTempFile()>	0.0364
<android.telephony.TelephonyManager:getPhoneType()>	0.0308
<java.io.FileOutputStream:write()>	0.0285
<android.telephony.TelephonyManager:getNetworkType()>	0.0283
<java.io.File:getAbsolutePath()>	0.0260
<android.location.Location:getAccuracy()>	0.0252
<java.io.File:getName()>	0.0251
<android.location.Location:getTime()>	0.0241
<java.io.File:delete()>	0.0228
<android.net.NetworkInfo:getDetailedState()>	0.0185
<android.net.NetworkInfo:getType()>	0.0177
<android.location.Location:getLatitude()>	0.0163
<android.location.Location:getLongitude()>	0.0164
<android.provider.ContactsContract$Contacts:openContactPhotoInputStream()>	0.0078
<android.hardware.Camera:autoFocus()>	0.0057
<android.hardware.Camera:stopPreview()>	0.0053
<android.hardware.Camera:open()>	0.0048
<android.hardware.Camera:cancelAutoFocus()>	0.0047
<android.hardware.Camera:getParameters()>	0.0039

3.3 特征获取

3.3.1 构造约简后函数调用图

使用 Soot 框架为每个应用程序生成函数调用图，然后进行约简操作，只保留能够到达敏感 API 节点的节点和敏感 API 节点。为表示约简过程，我们给出函数调用图

（Call Graph）的定义。

定义 3.1（**Call Graph**）一个程序的调用图 ***CG*** 定义为n个方法 *SootMethod* 的有限集：

$$\boldsymbol{CG} = \{Sootmethod_1, Sootmethod_2, \cdots, Sootmethod_n\}$$

其中方法 *SootMethod* 定义为：

$$SootMethod =< edgeInto, edgeOutOf >$$

- *edgeInto* 为方法 *SootMethod* 的前驱节点集；
- *edgeOutOf* 为方法 *SootMethod* 的后继节点集。

由于我们只关心敏感 API 的调用关系，因此对 CG 进行约简操作。约简 CG 的具体过程为：遍历函数调用图中的所有方法并将其与敏感 API 列表中敏感 API 进行比对，只保留敏感 API 列表中包含的方法；接着获取能够到达敏感 API 的方法，迭代循环直到最终方法为程序的入口点 DummyMain 方法为止。

算法 3.1 CG 的约简算法

输入：函数调用图 CG= $\{SootMethod_1, SootMethod_2, \cdots, SootMethod_n\}$
输出：约简后函数调用图 CG′ = $\{SootMethod'_1, SootMethod'_2, \cdots, SootMethod'_k\}$

```
1   for each SootMethod ∈ CG
2     if sootMethod ∈ 敏感 API and  sootMethod∉ CG′
3         将 sootMethod 加入 CG′
4   end
5     for each  SootMethod ∈ CG′
6         for each m ∈ CG.edgeInto(sootMethod)
7             if m ∉  CG′
8                 将m加入 CG′
9         end
10    end
```

在算法 3.1 中，第 1-4 行为获取函数调用图中敏感 API 节点操作，对于 CG 中的所有$sootMethod$，如果$sootMethod$属于敏感 API 且CG′中不包含$sootMethod$，则将$sootMethod$加入CG′中，此时CG′中包含该函数调用图中所有敏感 API 节点；第 5-10 行获取能够到达敏感 API 节点的所有节点，将CG′中所有$sootMethod$的$edgeInto$集加入CG′中，直到$sootMethod$的$edgeInto$集合为空。

3.3.2 构造约简后控制流程图

通过 Soot 框架为约简后的 CG 上的所有节点（敏感 API 节点除外）构造控制流图并约简，约简后的控制流图只保留约简后的 CG 中存在的节点。为表示控制流图的约简过程，我们给出控制流图（Control-Flow Graph）的定义。

定义 3.2（**Control-Flow Graph**）一个类方法$sootMethod$的控制流图 **CFG** 定义为

一个有向图：

$$\mathbf{CFG} =< Units, Edges, Head, Tail >$$

其中：

- $Units$为节点的有限集；
- $Edges \subseteq Units \times Units$ 为边的有限集；
- $Head \in Units$为程序的入口节点；
- $Tail \in Units$为程序的出口节点。

定义两个辅助函数$predOf(u)$和$succsOf(u)$分别为节点u的前驱节点集合和后继节点集合。对于有向图$G = (Unit, Edges)$。

- $\forall u \in Units\ predOf(u) = \{v|(v,u) \in Edges\}$，$v$节点为$u$节点的前驱节点；
- $\forall u \in Units\ succsOf(u) = \{v|(u,v) \in Edges\}$，$v$节点为$u$节点的后继节点。

CFG 的约简操作为：遍历$Units$中除程序入口节点和程序出口节点以外的所有节点，若该节点不为约简后的 CG 中的节点，则删除当前节点，并将该节点的所有后继节点添加到，该节点的前驱结点的后继节点集中。

算法 3.2 CFG 的约简算法

输入：控制流图 CFG，CG'
输出：约简后控制流图 CFG′

```
1   for each Unit ∈ Units
2   if Unit ≠ Head or Tail
3      if Unit 中不包含 CG′中方法 or Unit 包含的方法是调用自身的方法
4         for each u_suc ∈ CFG.SuccsOf(Unit)
5            for each u_pre ∈ CFG.PresOf(Unit)
6               将u_suc添加至u_pre的 SuccsOf集合中
7               将u_pre添加至u_suc的PresOf集合中
8            end
9         end
10        CFG.SuccsOf(Unit).delete(Unit)
11        CFG.PresOf(Unit).delete(Unit)
12        从Units中删除Unit
11  end
```

在算法 3.2 中，第 2 行为判断$Unit$节点是否为入口节点或者出口节点，如果不是则判断$Unit$节点是否为CG′中节点，如果不是则进行删除$Unit$节点操作；第 4-10 行为删除操作，从$Unit$的$SuccsOf$集合和$PresOf$集合中删除$Unit$节点，对于$Unit$的$SuccsOf$集合中所有节点u_{suc}，以及$PresOf$集合中所有节点u_{pre}。将u_{suc}添加到u_{pre}的$SuccsOf$集合中，将u_{pre}添加到u_{suc}的$PresOf$集合，最后从$Units$中删除$Unit$节点。

如图 3.2 为 DroidKungFu 中回调函数 onCreate 方法的 CFG 约简过程，其中删除了不可到达敏感 API 的 trunOnStrobe 节点，与非敏感 API 的$r1:=@parameter 节点和 setText

节点。约简后 CFG 如图（b）所示，只保留出口节点、入口节点以及可达敏感 API 节点的 getPhoneNumber 节点和 sendSMS 节点 。

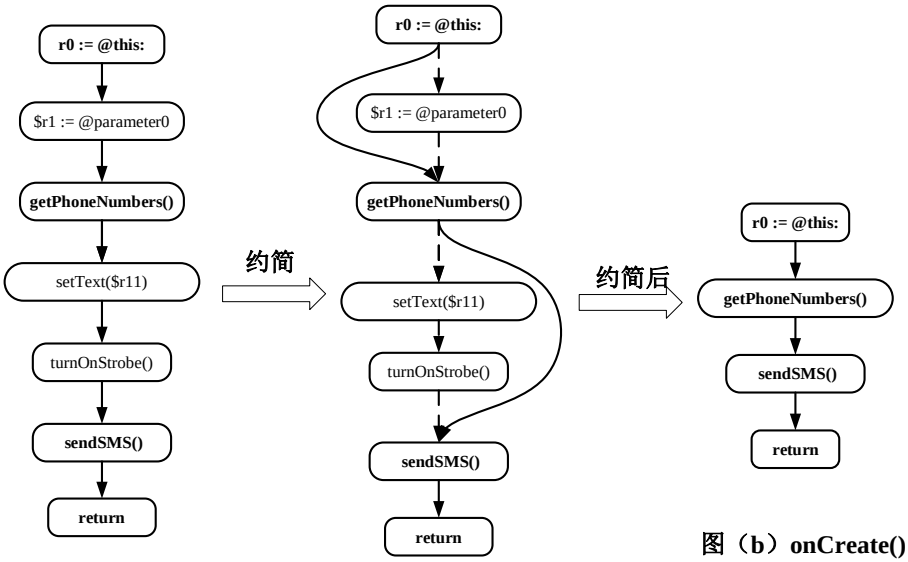


图 3.2 onCreate()的 CFG 的约简过程

3.3.3 构造回调函数的过程间控制流图

给定一个回调函数m_0，$\{m_0, m_1, \cdots, m_k\}$是回调函数$m_0$调用的所有类方法的集合，$CFG_i =< Units_i, Edges_i, Head_i, Tail_i >$是方法$m_i(i \in \{0,1,2,\cdots,k\})$对应的控制流图。回调函数$m_0$的过程间控制流图（Interprocedural CFG，iCFG）为回调函数 m_0 调用的所有类方法的 CFG 结合而成。

合成回调函数m_0的 iCFG 的操作为：遍历回调函数m约简后的 CFG，对于回调函数m_0约简后的 CFG 中的方法节点，将其替换为该方法约简后的 CFG。具体操作为：为方法节点的前驱节点的后继节点集中，添加该方法构造的约简后的 CFG 中入口节点的后继节点，并删除改方法节点。

算法 3.3 回调函数 iCFG 的构造算法

输入：回调函数m_0的 CFG_{m0} ,CFG_{m0}中所有方法的CFG_{mi}，其中 $i \in \{1,2,\cdots,k\}$;
输出：回调函数m_0的$iCFG_{m0}$;

```
for Unit ∈ Units_m0
    if Unit ≠ Head or Tail
        获取 Unit中包含的m1方法的CFG_m1
        if CFG_m1 ≠ null
            for each Unit_1 ∈ Units_m1
                if Unit_1 = Head
                    for each u_pre ∈ CFG_m0.PresOf(Unit)
                        for each u_suc ∈ CFG_m1.SuccsOf(Unit_1)
```

```
9                    将$u_{pre}$添加至$u_{suc}$的 $PresOf$集合中，并删除 $Unit$
10                   将$u_{suc}$添加至$u_{pre}$的 $SuccsOf$集合中，并删除 $Unit_1$
13               if $Unit_1 = Tail$
11                   for each $u_{pre} \in CFG_{m1}.PresOf(Unit_1)$
12                       for each $u_{suc} \in CFG_{m0}.SuccsOf(Unit)$
13                           将$u_{pre}$添加至$u_{suc}$的 $PresOf$集合中，并删除 $Unit_1$
14                           将$u_{suc}$添加至$u_{pre}$的 $SuccsOf$集合中，并删除 $Unit$
15                       end
16                   end
14                   $Unit_1$ 加入 $Units_{m0}$
15               end
16           end
17   end
```

算法 3.3 中可知，第 5-12 行为将m_1方法的CFG_{m1}组合到回调函数m_0的CFG_{m_0}操作，将所有m_1中（不为出入口节点）节点$Unit_1$添加到CFG_{m_0}中，对于CFG_{m1}程序入口节点，为m_1方法所在的$Unit$节点的$PresOf$集合中的所有节点u_{pre}的$SuccsOf$集合中添加CFG_{m1}入口节点的$SuccsOf$集合中的所有节点，同时为CFG_{m1}入口节点的$SuccsOf$集合中的所有节点u_{suc}的$PresOf$集合中添加$Unit$节点的$SuccsOf$集合中的所有节点。CFG_{m1}程序出口节点操作类似。

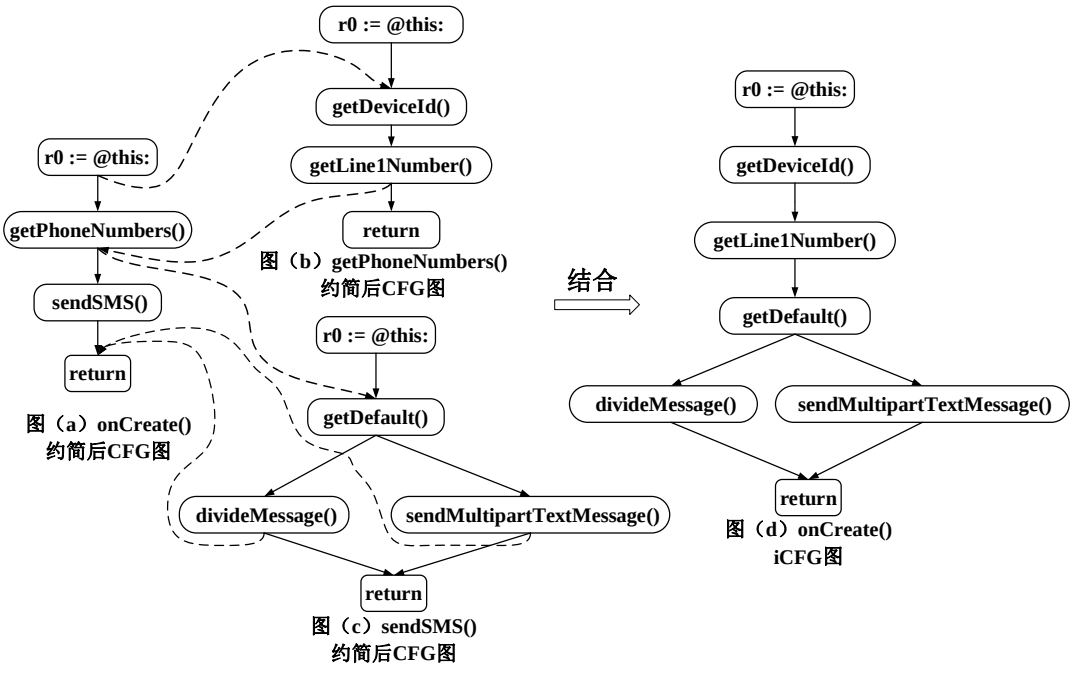


图 3.3 onCreate 方法 iCFG 的构造过程

如图 3.3 所示，遍历 onCreate 方法 CFG 图，对于 getPhoneNumber 节点，首先获取该节点的前驱节点 r0 := @this:和该节点的后继节点 sendSMS，将 r0 := @this:节点指向 getPhoneNumber 方法约简后 CFG 内入口节点的后继节点 getDeviceId 节点，并将出口节

点的前驱节点 getLineNumber 节点指向 sendSMS 节点，删除 getPhoneNumber 节点和 getPhoneNumber 方法约简后 CFG 的入口节点和出口节点；对 onCreate 方法 CFG 图中的 sendSMS 节点做相同操作，得到图（d）所示 onCreate 方法的 iCFG 图。

3.3.4 约简后 iCFG 的去环

在 Soot 对应用程序构造控制流图时，循环操作由图中的环来表示。但环的存在会降低路径搜索算法提取敏感 API 序列的效率，因此我们为约简后 iCFG 图进行去环操作。

算法 3.4 iCFG 的去环操作算法

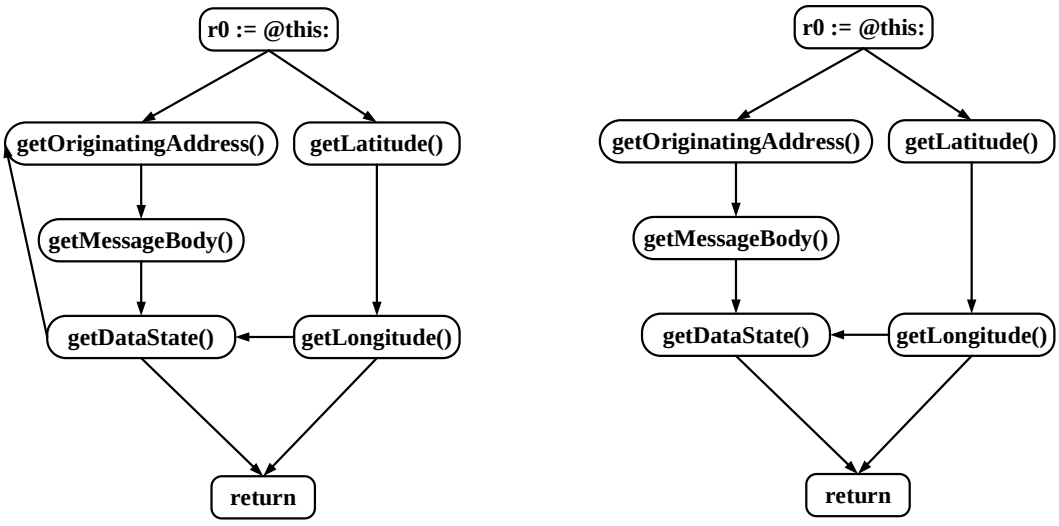
输入：约简后$iCFG_{m0}$;
输出：去环后$iCFG_{m0}$;

```
1   List<Unit> path //路径，List <Unit> loopStart //环起点集合；List<Unit> loopEnd；//环终点集合
2   void findLoop (Unit unit)
3         if (unit != Tail)
4            path.add(unit)
5            for each unitNext ∈ iCFG_m0.SuccsOf(unit)
6                 if path.contains(unitNext)  & unitNext≠ Tail
7                    loopStart.add(unitNext)
8                    loopEnd.add(unit);
9                 else if (path.!contains(unitNext))
10                     findLoop (unitNext);//递归
11                else if unitNext = Tail
12                     path.remove(unit)
13   for each  Unit start  ∈ loopStart ,end ∈ loopend
14       iCFG_m0.SuccsOf(end).removePred(start)
15       iCFG_m0.SuccsOf(start).removePred(end)
16  end
```

在算法 3.4 中，首先定义了两点之间的路径集合、环起点集合以及环终点集合。第 2-14 行进行环识别操作，首先从入口节点开始遍历全图，寻找入口节点到出口节点的所有路径，将该路径上的所有节点存入 path 集合中。如果节点$unit$的下一个节点$unitNext$为该路径出现过的节点，则$unit$节点为一个环的终点，$unitNext$为这个环上的起点，将$unit$和$unitNext$分别存入 loopStart 集合和 loopEnd 集合中；14-19 行进行去环操作，对所有的环起点的$PredsOf$集合进行删除对应的环终点操作，对所有的环终点的$SuccsOf$进行删除对应的环起点操作。

如图 3.4 所示，图（a）为去环前 onRusue CFG 图，遍历该图，发现序列{getOriginatingAdress()；getMessageBody()；getDataState()；getOriginatingAdress()}为一个循环，其中 getOriginatingAdress()为该环起点，getDataState 为环终点。对其进行去环操作，将 getDataState()的后继节点集合中删除 getOriginatingAdress()，将 getOriginatingAdress()的前驱集合中删除 getDataState()。图（b）为去环后 onRusue CFG

图。



图（a）约简后onRusue()的iCFG图　　**图（b）去环约简后onRusue()的iCFG图**

图 3.4 iCFG 去环操作图

3.4 敏感 API 序列提取

通过对 iCFG 图的遍历提取敏感 API 序列作为特征值，算法 3.5 展示了对某样本中的 iCFG 图进行 API 调用序列提取的过程。

算法 3.5 API 调用序列提取算法

输入：约简后 iCFG
输出：List < List<Unit >> allPath;//所有序列集合

```
1    List<Unit> path //序列节点集合;Unit unit//iCFG 的入口节点
2   void findPath (Unit unit)
3         if (unit != Tail)
4             path.add(unit)
5             for each unitNext ∈ CFG.SuccsOf(unit)
6                 if (!path. contains(unitNext))
7                     findPath (unitNext);//递归
8                 else  path.remove(unit)
9               end
10          else allpath.add(new List<Unit>(path.add(unit)))
11    end
```

在算法 3.5 中，第一行定义了节点集合 path 来保存某条序列上的所有节点。第 2-11 行为提取敏感 API 序列操作，首先从图的入口节点开始进行深度优先遍历，提取入口节点到出口节点的所有序列。若当前节点$unit$为出口节点则将该条序列存入 allPath 中，返回$unit$后继节点。若不为出口节点，则遍历$unit$节点的$SuccsOf$集合，如果$unit$节点指向的节点$unitNext$不在序列中，则将$unitNext$作为$unit$节点，重复上述操作，直到遍历图中所有节点。

用该算法对图 3.3 中图（d）中 onCreate 方法 iCFG 图进行提取敏感 API 序列操作后得到{r0 := @this:；getDeviceId()；getLine1Number()；getDefault()；sendMultipartTextMessage()；return}和{ r0 := @this:；getDeviceId()；getLine1Number()；getDefault()；divideMessage()；return }两条敏感 API 序列。

3.5 本章小结

本章介绍了获取敏感 API 序列的详细过程，首先分析了案例样本中包含的敏感 API 及权限。接着介绍了本文中使用的敏感 API，并定义了一组计算公式衡量敏感 API 在样本集中的分布情况。然后详细描述了函数调用图和控制流图的约简操作，介绍了回调函数控制流图的构造方法，并分析了环对提取敏感 API 序列产生的影响，并详细描述了回调函数控制流图的去环过程，最后介绍了敏感 API 序列的提取过程。

4 基于生成对抗样本的应用程序恶意性检测

本章提出一种基于生成对抗样本的应用程序恶意性检测方法，方法流程图如图 4.1 所示。本章方法的总体思路是：以第三章提取的敏感 API 序列作为应用程序的特征，使用不同机器学习分类方法对良性和恶意样本的特征向量进行学习，构造分类器。在构造分类器的过程中，为了提高分类器的准确率、召回率和精确度，采用遗传算法对训练集中恶意样本进行演化，生成新的恶意样本作为对抗样本，输入到分类器中进一步训练，这一过程不断迭代，直到检测效果达到最佳时停止。

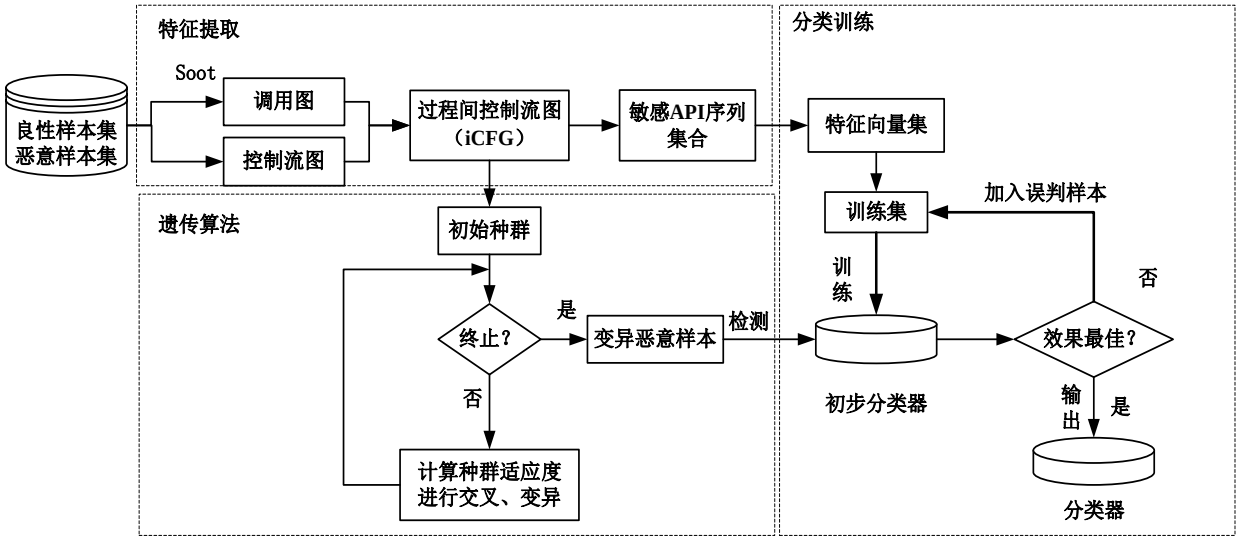


图 4.1 方法流程图

4.1 基于遗传算法生成对抗样本

采用遗传算法将恶意样本集作为初始种群进行演化以得到新型的恶意样本，将其作为对抗样本。我们将一个恶意应用程序作为个体，约简后一个回调函数的 iCFG 作为一条染色体，每个应用程序中包含多个回调函数的约简后 iCFG，即每个个体具有多条染色体，iCFG 上的敏感 API 作为基因，进行交叉变异。定义三个目标函数约束其演化方向，同时规定三种约束规则以保证生成的恶意样本的代码具有可执行性。

4.1.1 基因编码

首先对个体（即一个恶意应用程序）进行编码。设该 app 有 k 个回调函数，每个回调函数对应的约简后的 iCFG 为C_i，则该 app 的染色体序列为

$$I = \{C_1, C_2 \cdots, C_k\}$$

为了便于后面讨论染色体的变异，这里给出染色体，即 iCFG 的定义。

定义 4.1 (Interprocedural CFG) 由于 iCFG 图也是一个控制流图，为简化起见，我们仍用 CFG 来定义 iCFG，因此一个 iCFG 可以定义为一个五元组：

$$iCFG =< Units, Edges, Head, Tail, L >$$

- $Units$是节点的有限集；
- $Edges \subseteq Units \times Units$ 是边的有限集；
- $Head, Tail \in Units$分别是图的入口点和出口点；
- $L : Units \rightarrow APIs$是一个标记函数，它把每个节点（除去$Head, Tail$节点）映射为一个敏感 API 函数。

4.1.2 种群适应度

将定义的n个敏感 API 进行编码，从 1 到n连续编号，每个敏感 API 被分配一个从 1 开始的索引，则本文使用的敏感 API 表示为$\{a_1, a_2, a_3, \cdots, a_n\}$，其中$a_n$表示编号为$n$的敏感 API。

适应度函数是评价个体对于环境的适应程度，为了全面衡量一个程序的恶意性，攻击性和规避性，我们规定三个目标函数分别对应不同的演化方向，定义如下：

定义 4.2（**攻击性**）攻击性代表程序x可能对用户产生的损坏程度，通过个体中包含的敏感 API 的数量来表示，即：

$$F_{ac}(x) = \sum_{i=1}^{n} \|a_i\| \tag{4.1}$$

如果个体x中包含敏感 API a_i，则$\|a_i\|$的值为 1，不包含则为 0。

定义 4.3（**规避性**）代表了程序x被检测出的概率，使用样本是否能够通过给定的多个检测工具的检测来表示，即：

$$F_{ec}(x) = \frac{T}{C_{num}} \tag{4.2}$$

C_{num}为给定的检测工具的个数，T 为检测工具将该程序标记为恶意的个数。

定义 4.4（**恶意性**）代表程序x可能为恶意样本的概率，通过包含的敏感 API 概率来表示，定义为：

$$F_{weight}(x) = \frac{\sum_{i=0}^{n} w(a_i)}{n} \tag{4.3}$$

$w(a_i)$为敏感 API a_i的权重，由式（3.1）计算而来，n为程序中敏感 API 的总数。

我们给予三个目标函数与其权重相乘，三者相加作为程序x最终的适应度函数，适应度函数公式如式（4.4）所示。

$$F(x) = \omega_1 F_{ac}(x) + \omega_2 F_{ec}(x) + \omega_3 F_{weight}(x) \tag{4.4}$$

式（4.4）中，ω_1，ω_2，ω_3分别为$F_{ac}(x)$，$F_{ec}(x)$，$F_{weight}(x)$的权重值在本文方法中权重值分别为 0.3，0.2 和 0.5，权重值由作者根据实验效果进行调整。

4.1.3 选择

选择算子模拟自然界中适者生存的生存规则，保留适应度高的个体，淘汰适应度低的个体。本章中通过轮盘赌选择法对种群中个体进行选择，轮盘赌选择法的思想是种群中个体被选择的概率与其适应度成正比。

定义 4.5（**选择概率**）个体x_i被选中的概率$P(x_i)$，如式（4.5）所示：

$$P(x_i) = \frac{F(x_i)}{\sum_{i=1}^{n} F(x_i)} \tag{4.5}$$

其中$F(x_i)$为个体x_i适应度函数，n为种群中所有样本的数量。

定义 4.6（**累积概率**）表示每个个体之前所有个体的选择概率之和，如式（4.6）所示：

$$q(x_i) = \sum_{j=1}^{i} P(x_j) \tag{4.6}$$

选择算子的具体步骤为：首先对种群中每个个体计算适应度函数并对其按照适应度从大到小排列，然后通过选择算法选取个体组成新种群替代就种群。

算法 4.1 选择算法

```
输入： LinkedList<Individual> group //种群，int maxsize //种群中最大个体数；
输出： LinkedList<Individual> newGroup//新种群；
1   LinkedList<Individual> ScoreGroup，double[] f, double[] p_set, double[] q_set
2   for each i in group
3       根据公式（4.1）、(4.2)、(4.3)、(4.4)分别计算F_ac(i)、F_ec(i)、F_weight(i)、F(i)
4       ScoreGroup.set (index,i);//根据 F(i)值由大到小插入 ScoreGroup 集合中
5       f.set(index. F(i));
6   for each i in ScoreGroup
7       根据公式（4.5）计算 P(x_i) 加入 p_set 中
8       根据公式（4.6）计算 q(x_i)加入 q_set 中
9    for j in (0,maxsize):
10          double r = math.random(0,1);
11          for each q in q_set
12            if r<q && r > q_set(index(q)-1);
13              newGroup.add (ScoreGroup.get(j))//将个体加入新种群中
14          end
15  return newGroup;
```

算法 4.1 为选择操作过程，其中 2-5 行为为种群中所有个体计算适应度函数，并根据适应度函数大小对其进行排序；6-8 行为计算每个个体的个体选择概率和累积概率；

9-14 行为选择个体添加新种群中，首先随机生成一个 0 到 1 之间的数，然后将该随机数与所有个体的累积概率比较，如果随机数小于某个体的积累概率并且大于该个体前一个个体的累积概率，则将该个体加入新种群中。循环该过程，直到新种群中样本数量达到最大数量。

4.1.4 交叉

交叉算子模拟自然界中个体的交配过程，从而增加种群多样性。在交叉过程中，首先使用轮盘赌随机选取两个适应度高的个体i_1、i_2 作为父代，随机抽取双方一半的染色体进行结合，得到具有双方基因的子代i_{child}，确保在迭代过程中保留上一代的优秀基因。

算法 4.2 交叉算法

输入：个体 $i_1=\{C_1^1,C_2^1,\cdots,C_n^1\}$，$i_2=\{C_1^2,C_2^2,\cdots,C_n^2\}$
输出：个体 $i_{child}=\{C_1^{child},C_2^{child},\cdots,C_k^{child}\}$
1 for $n<1/2*size(i_1)$
2 int r_1 =Ramdom($size(i_1)$)
3 int r_2 =Ramdom($size(i_2)$)
4 for each $C_k^{child}\in i_{child}$
5 if $C_{r_1}^1$，$C_{r2}^1\notin i_{child}$
6 将$C_{r_1}^1$，C_{r2}^1作为C_n^{child}，C_{n+1}^{child}加入i_{child}中
7 end
8 将$C_{r_1}^1$，C_{r2}^1从 i_1，i_2中删除
9 end

算法 4.2 为交叉操作过程，第 2-3 行分别为i_1，i_2生成范围为 1 至i_1的长度的随机数；将生成的随机数作为下标提取染色体$C_{r_1}^1$，C_{r2}^1，如果i_{child}中没有$C_{r_1}^1$，C_{r2}^1，则将其作为C_n^{child}，C_{n+1}^{child}加入i_{child}中，然后将$C_{r_1}^1$，C_{r2}^1分别从i_1，i_2中删除，到n的值为i_1长度的一半为止。

4.1.5 变异

交叉过程只是基于原有基因集进行选择，交换其组合顺序。与交叉算子不同的是，变异算子通过对个体自身（染色体）的基因位置进行变异来模拟种群中个体的基因变异。在本研究中，变异主要由增添基因，更改基因以及交换次序变量三种变异模式组成。首先将变异概率设置为 0.01，然后遍历整个种群，并随机为每条染色体产生一个数值，如果数值小于变异概率，则随机选择一种变异模式对染色体进行变异操作。

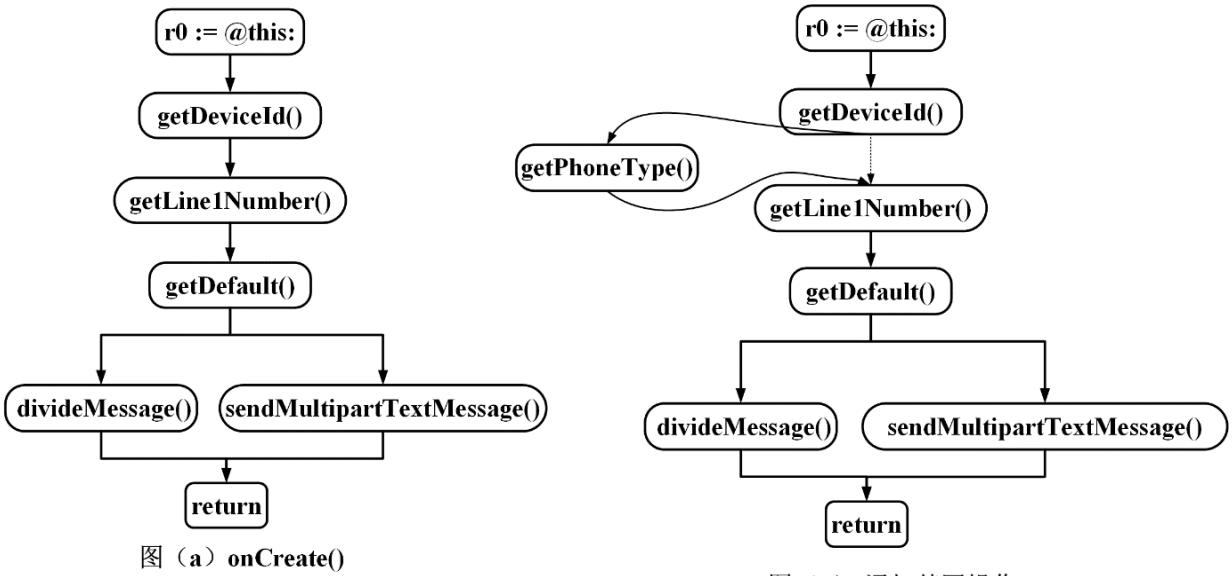
（1）增添基因，遍历敏感 API 集，随机选择一个 API 作为基因插入染色体随机选择的基因之后。

定义 4.7（**增添基因**）设随机选择的 API 为a，在 iCFG 上随机选择的插入位置节点

为$n \in Units$，则增添基因就是形成一个标记为a的节点n'，然后将该节点插入位置节点n之后。记添加基因后的$iCFG$为$iCFG' =< Units', Edges', Head', Tail', L' >$，则添加基因操作可以定义为一个$iCFG$到$iCFG'$的图变换过程$g: iCFG \to iCFG'$，该变换满足如下条件：

- $Units' = Units \cup \{n'\}$，$n'$为新增节点；
- $Edges' = (Edges | Edges \neq n \times succsOf(n)) \cup \{(n, n')\} \cup n' \times succsOf(n)$；
- $Head' = Head$，$Tail' = Tail$；
- $L' = L \cup (n', a)$。

如图 4.2 中图（b）所示，在图（a）中插入节点 getPhoneType 时，将其连接在节点 getDeviceId 和节点 getLine1Number 之间。



图（a）onCreate() iCFG图　　图（b） 添加基因操作

图 4.2 增添基因操作图

（2）更改基因，随机选取染色体上一基因，将其更改为染色体上未出现的基因。

定义 4.8（**更改基因**）设随机选择的 API 为a'，在 iCFG 上随机选择的更改节点为$n \in Units$，更改节点的 API 为a，则更改基因就是形成一个标记为a'的节点n'，然后将节点n更改为该节点。记更改基因后的$iCFG$为$iCFG' =< Units', Edges', Head', Tail', L' >$，则更改基因操作可以定义为一个$iCFG$到$iCFG'$的图变换过程$g: iCFG \to iCFG'$，该变换满足如下条件：

- $Units' = \{Units | Units \neq n\} \cup \{n'\}$，$n'$为更改节点；
- $Edges' = (Edges | Edges \neq n \times succsOf(n) \cup predOf(n) \times n) \cup n' \times succsOf(n) \cup predOf(n) \times n'$；

- $Head' = Head$，$Tail' = Tail$；
- $L' = \{L|L \neq (n,a)\} \cup (n',a')$。

如图 4.3 中图（b）所示，将图（a）中节点 getDeviceId 替换为节点 getPhoneType。

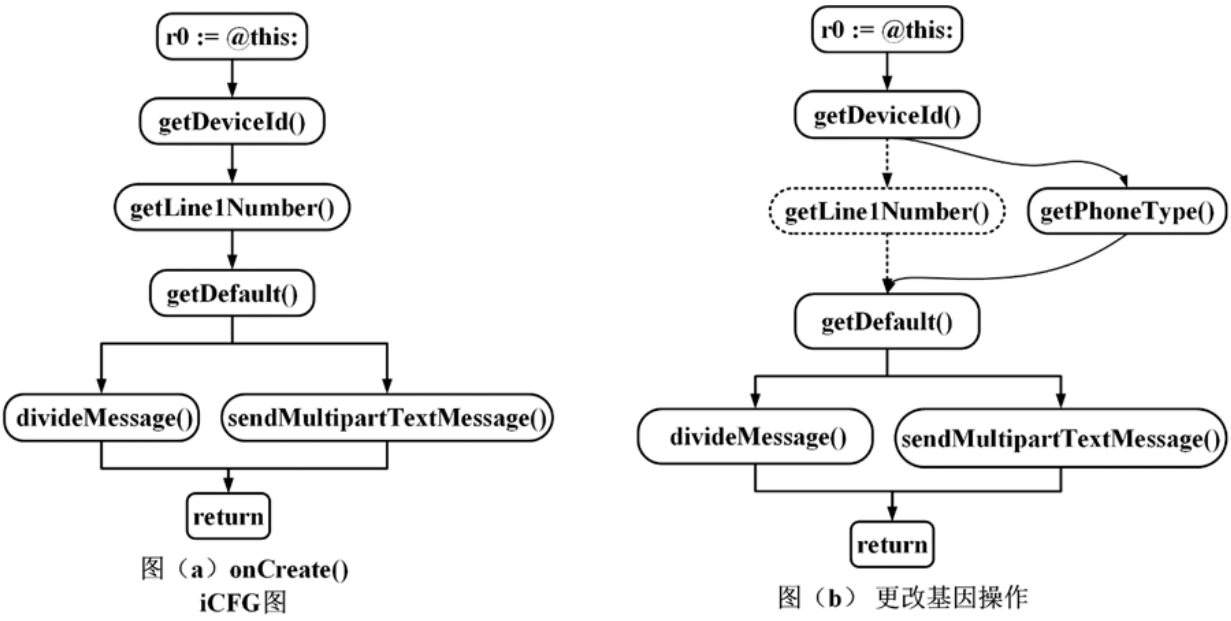


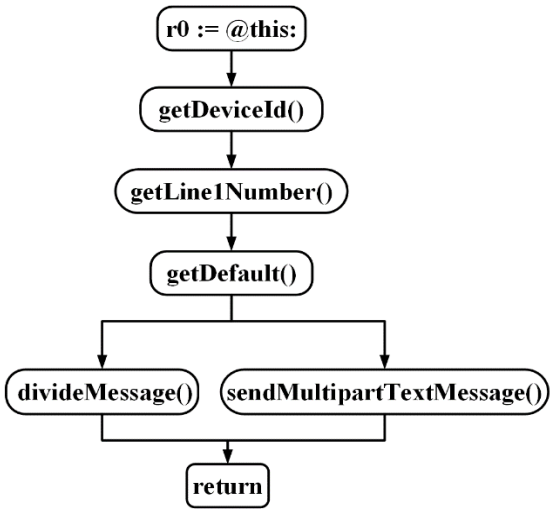
图 4.3 更改基因操作图

（3）交换位置，随机选取染色体上两个基因进行位置交换。

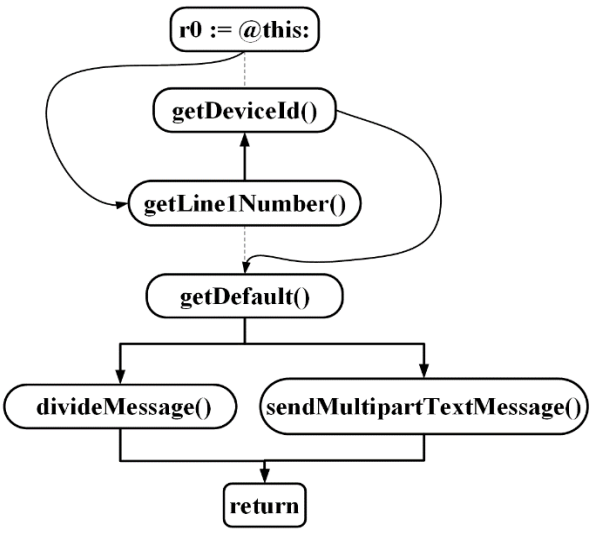
定义 4.9（**交换次序**）在$iCFG$上随机选择两个交换位置的节点$n,n' \in Units$，则交换次序就是将节点n与节点n'的前驱结点集和后续节点集合分别进行交换。记交换次序后$iCFG$为$iCFG' =< Units', Edges', Head', Tail', L' >$，则交换位置操作可以定义为一个$iCFG$到$iCFG'$的图变换过程$g: iCFG \to iCFG'$，该变换满足如下条件：

- $Units' = Units$；
- $Edges' = \left(Edges \middle| \begin{matrix} Edges \neq n \times succsOf(n) \cup predOf(n) \times n \\ \cup\, n' \times succsOf(n') \cup predOf(n') \times n' \end{matrix}\right) \cup n' \times succsOf(n) \cup predOf(n) \times n' \cup n \times succsOf(n') \cup predOf(n') \times n$，$n,n'$为交换位置的两节点；
- $Head' = Head$，$Tail' = Tail$；
- $L' = L$。

过程如图 4.4 中图（b）所示，将图（a）中节点 getDeviceId 与节点 getLine1Number 的位置进行交换，即将两节点的前驱节点集合和后继节点集合分别进行交换。



图（a）onCreate() iCFG图



图（b） 交换次序操作

图 4.4 交换次序操作图

4.1.6 约束规则

为保证生成代码的正确性和有效性，在进行变异操作时，需要满足特定的约束，我们定义的约束如下：

（1）权限约束：某些 API 在执行过程中，需要用户赋予相应权限。如果不满足权限约束，则会导致生成的恶意样本无法实现该 API 对应的功能。例如<android.telephony.SmsManager:sendDataMessage()>发送短信 API，需要 android.permission.SEND_SMS 权限才能运行，因此在进行增添基因以及替换基因操作时，需要查验原染色体是否已赋予该权限，即原染色体是否出现需要该权限约束的 API。

（2）次序约束：在应用程序运行某些功能时，对于 API 的次序有严格要求，如果生成样本不满足次序约束，会导致某些功能无法实现。例如执行录音功能时，实现开始录音功能的 API <android.media.MediaRecorder:start()>必须在实现结束录音 API <android.media.MediaRecorder:stop()>之前出现，因此在进行交换次序操作需要查验交换次序的 API 是否有次序要求。

（3）API 之间的依赖约束：在实现某些功能时，有时需要两个及以上 API 相互配合完成，例如获取用户详细位置信息时，需要获取纬度的 API，<android.location.Location:getLatitude()>和获取经度 API，<android.location.Location:getLongitude()>同时出现，因此在进行更改基因和增加基因操作时，需要查验该节点是否与其他 API 有依赖关系。

4.2 特征变换

针对良性样本集以及恶意样本集生成敏感 API 调用序列集，由于生成的序列可能在样本内或样本间发生重复，对其进行去重操作，建立敏感 API 调用序列特征集合并进行编号。使用机器学习方法进行恶意程序检测，需要将提取的敏感 API 序列进行特征值编码，通过特征向量表示程序样本。

假设样本集的敏感 API 调用序列集数量为m，则样本的特征向量表示为一个$m+1$维的位向量$malicious=\{flag,0,1,\cdots 0\}$，其中$flag$为标志位，取值为 1 时为恶意样本，为 0 时为良性样本。当样本中的某一个敏感 API 序列出现在恶意特征样本集合中时，该特征向量的对应位置标注为 1，否则为 0。

例如，遍历 DroidKungFu 约简后 iCFG 图得到{r0 := @this:；getDeviceId()；getLine1Number()；getDefault()；sendMultipartTextMessage()；return}和{ r0 := @this:；getDeviceId()；getLine1Number()；getDefault()；divideMessage()；return }两条敏感 API 序列，若这两条序列在恶意样本特征集合中的序号分别为 1 和 2，则 DroidKungFu 的特征向量可以表示为$f=\{1,1,1,0,\cdots,0,0\}$。良性特征向量得到编码过程与此类似。

4.3 实验评估

4.3.1 实验环境、数据集和度量标准

本章实验在 windows10(64 位)操作系统上完成，CPU 为 Intel(R) Core(TM) i5-10210U，16G 内存。相关软件工具包括 Python3.9、Java1.8、IDEA2020、Pycharm2020。

恶意样本集分别选自于 Drebin 样本库，Virus Share 样本库和 Maldroid[40] 样本库。为了使样本数据具有代表性，我们选择的样本覆盖所有样本库中的所有家族。良性样本来自 Benign_2015 样本库和 Benign_2016 样本库，该样本库样本均预先通过检测确保为良性样本。随着 apk 增大，Soot 处理 apk 的效率降低；但 apk 过小时， apk 中能够提取的特征较少；为了提高时间效率的同时保证提取的特征能完整描述样本集特性，我们选择大小范围在 100k 到 10M 之间的 apk 作为实验样本。

表 4.1 样本集的属性信息

程序类别	样本来源	样本数（个）	家族数（个）
恶意程序	Drebin	600	49
	virusShare	600	35
	Maldroid	600	27
良性程序	Benign_2015	600	未分类
	Benign_2015	600	未分类

为了能够全面的评估本检测方法的可行性与有效性，采取 ACC（准确率），PR（精确度）与 RC（召回率）三种指标为评价标准，着重对加入对抗样本前后分类器的检测能力进行测试。TP（TruePositive）表示被正确归类为恶意程序的样本数量，FP(FalsePositive)表示把良性样本错误识别为恶意样本的数量，TN（TrueNegative）表示正确归类为良性样本的数量，FN（FalseNegative）为被错误归类的恶意样本的数量。

ACC：在所有样本中，被正确检测出的样本（包括正确检测为恶意样本和正确检测为良性样本）所占的比例，如式（4.5）所示：

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.5}$$

PR：在被检测为恶意样本的数量中，确实为恶意样本的比例，如式（4.6）所示：

$$PR = \frac{TP}{(TP+FP)} \tag{4.6}$$

RC：在真实恶意样本中被成功检测为恶意样本的比例，如式（4.7）所示：

$$RC = \frac{TP}{TP+FN} \tag{4.7}$$

4.3.2 敏感 API 序列分析

（1） 敏感 API 序列集合性质

为了更清晰表达敏感 API 序列集合性质，本定义了敏感 API 序列重复率[41]（Sequence Repeat Rate，SRR），公式如式（4.8）所示：

$$SRR = \frac{\text{allRepeatSquence}}{\text{noRepeatSquence}} \tag{4.8}$$

其中 allRepeatSquence 为样本集内所有重复的序列数，noRepeatSquence 为样本集内无重复的序列数目。

表 4.2 敏感 API 序列集合统计

样本类别	样本数量	敏感 API 序列总数	样本内去重	样本间去重	SRR
恶意样本	1620	167532	15408	117972	390.5%
良性样本	779	87450	13695	43699	190.1%

根据 SRR 公式，恶意样本 SRR 为 390.5%，良性样本为的 SRR 为 190.1%。数据表明，虽然恶意样本数量是良性样本数量的一倍，但是经过去重操作后生成的敏感 API 序列数量相差不大。因为良性样本的代码量往往比恶意样本代码量大，良性样本需要实现更多功能，而恶意样本为了快速的实施破坏活动，代码量小。恶意样本的 SRR 超过良性样本的 SRR 两倍，说明了恶意样本相较于良性样本具有更多的重复敏感 API 序

列，意味着恶意软件在运行时会频繁的触发某些敏感 API 序列，在恶意行为的表现上更聚焦。

（2）敏感 API 序列长度分布情况

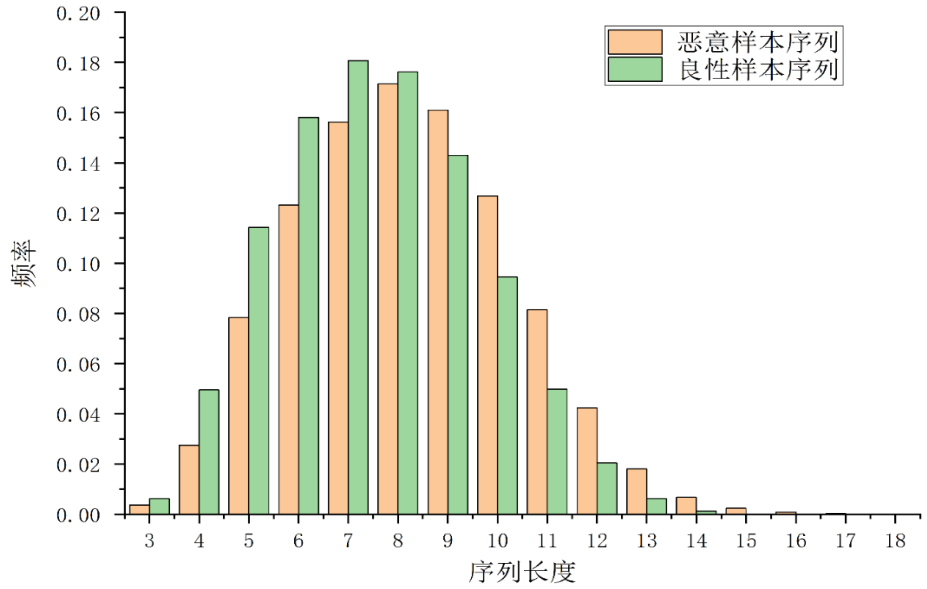


图 4.5 敏感 API 序列分布情况

图 4.5 为敏感 API 序列长度分布情况。由图可知，敏感 API 序列长度范围主要在 6-8 之间。相较于良性样本来说，恶意样本在序列长度在 9-18 之间的频率较高。而在敏感序列长度范围在 3-8 之间时，良性样本的频率高于恶意样本的频率。由图可以看出良性样本和恶意样本的敏感 API 序列的长度有差异，因此使用敏感 API 序列能够更好的区分样本的恶意性。

4.3.3 初步分类器训练

将本文的三个样本库提取特征集分别应用于决策树算法、逻辑回归（Logistic Regression，LR）、梯度提升树（Gradient Boosting Decison Tree，GDBT）这 3 类分类算法进行分类器训练并测试。其中训练集的恶意样本数量均为 200 个，良性样本为 100 个；测试样本为该样本库下恶意样本 400 个，良性样本 200 个。

（1） Drebin 样本集初步分类器训练

实验结果如表 4.3 所示：

表 4.3 Drebin 样本集不同机器学习算法的检测结果

度量标准	决策树	GDBT	LR
ACC	89.97 %	86.57%	88.58%
RC	93.96%	84.55%	89.95%
PR	89.50%	95.75%	91.75%

从表 4.3 可以看出，由 Drebin 样本集提取的特征集在决策树分类算法的效果最好，准确率有 89.97%，93.96%的召回率和 89.50%的精确率。GDBT 的准确率最低，但精确度最高，达到 95.75%。

（2） Virus Share 样本库初步分类器训练

实验结果如表 4.4 所示：

表 4.4 Virus Share 样本库不同机器学习算法的检测结果

度量标准	决策树	GDBT	LR
ACC	85.43%	94.17%	95.08%
RC	80.40%	94.80%	96.07%
PR	96.64%	97.20%	96.35%

由表 4.4 可知，由 VirusShare 样本集提取的特征集在 LR 分类算法的效果最好，获得了 95.08%的准确率、96.07%的召回率和 96.35%的精确率。而在决策树算法上的表现效果最差，只有 80.40%的召回率。

（3） Maldroid 样本库初步分类器训练

实验结果如表 4.5 所示：

表 4.5 Maldroid 样本库不同机器学习算法的检测结果

度量标准	决策树	GDBT	LR
ACC	89.78%	91.00%	92.02%
RC	88.72%	88.72%	89.77%
PR	96.14%	96.79%	98.03%

从表 4.5 可以看出，由 Maldroid 样本集提取的特征集在 LR 分类算法的效果最好，准确度为 92.02%、召回率为 89.77%、精确度为 98.03%。而决策树算法的效果最差，准确率为 89.78%。

综合以上实验可知，在三个数据集中使用三种不同机器学习算法训练的分类器对未知样本的检测准确率均达到 85%以上，精确率大部分在 96%以上。验证了本文将敏感 API 序列作为特征能够较为准确的区分良性样本和恶意样本。

4.3.4 对抗样本的干扰性实验分析

本实验旨在验证通过本文提出的对抗样本生成方法生成的样本是否具有干扰性。

（1） 对抗样本对现有工具的干扰性

为了验证本文生成对抗样本的方法是否对现有安全工具有干扰性，本文基于 Drebin 样本集的 20 个恶意样本作为原始样本集，同时对其进行交叉变异生成对抗样本，进行重打包后生成 APK 文件后作为对抗样本集，使用 VirusTotal 网站中收集的多种恶意软件检测工具，分别对原始样本集和对抗样本集进行检测。检测结果如图 4.6 所示。

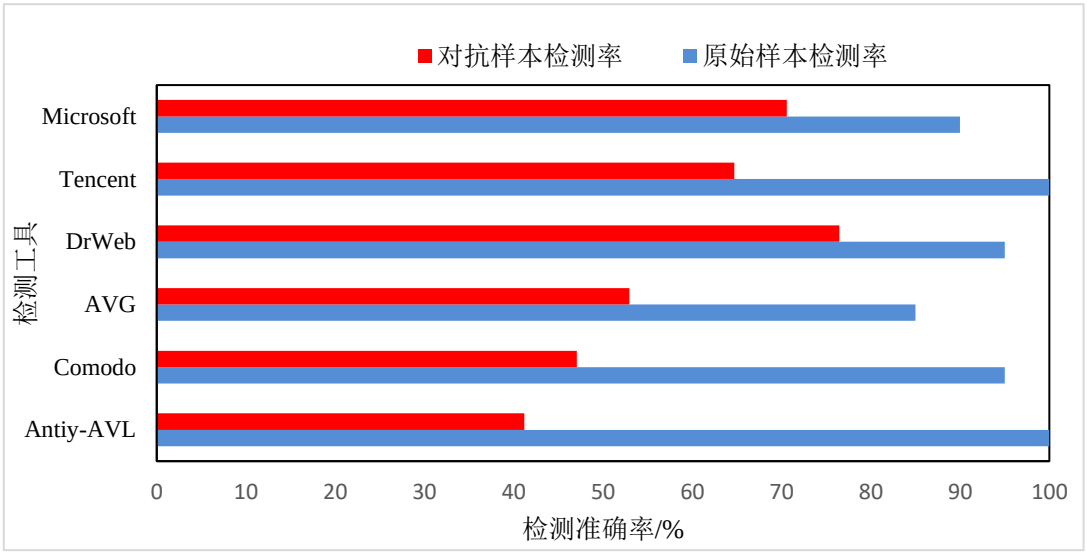


图 4.6 对抗样本及原始样本检测对比

由图 4.6 可知，本文方法对于 Antiy-AVL 工具干扰性最强，检测率下降了 59.83%，对于 DrWeb 工具干扰性较弱，检测率下降了 8.53%。实验结果表明采用本文方法生成的恶意样本能够躲避现有工具的检测。

（2） 对抗样本对初步分类器的干扰性

为了验证本文生成对抗样本对本文训练的分类器的干扰效果，选取 Drebin 样本集中 200 个恶意样本作为初始种群进行交叉变异，生成对抗样本后输入初步分类器进行检测。不同样本集生成干扰性实验结果如图 4.7。

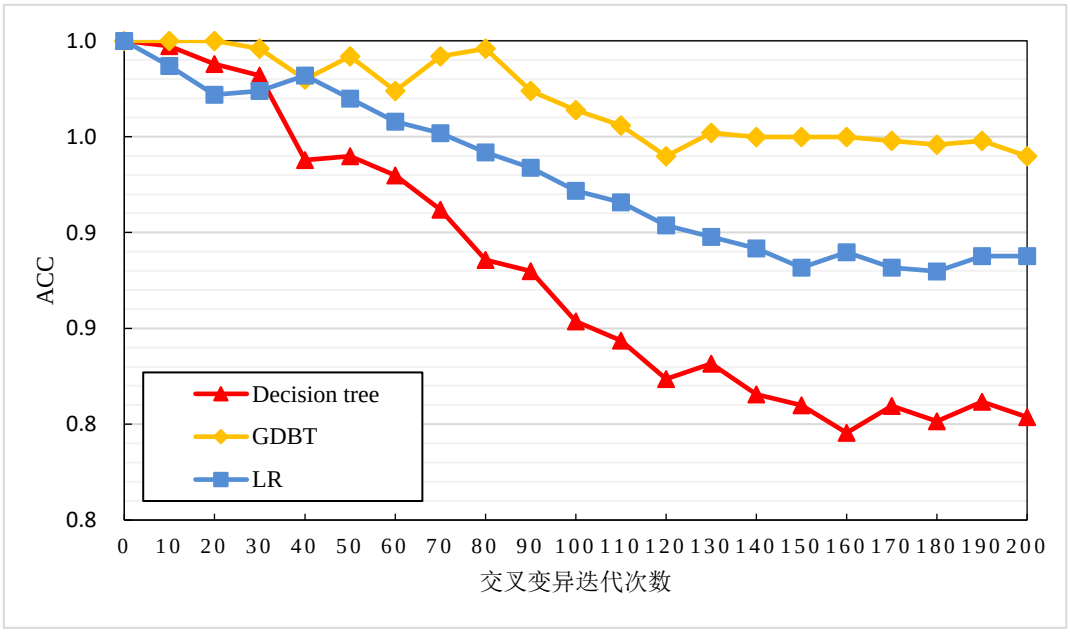


图 4.7 Drebin 数据集中对抗样本准确率与迭代次数关系图

由图 4.7 可知，随着迭代次数的增加，新生成的对抗样本对初步分类器的干扰性逐步增强，在迭代160次之后，干扰效果趋于稳定。具体而言，新生成的对抗样本对决策树算法的干扰最大，准确率降低了 19%，对 GDBT 算法的干扰性最小，准确率降低了 5%，实验结果表明采用本文方法所生成的对抗样本能够躲避初步分类器的检测。

4.3.5 分类器重训练实验分析

将训练集中恶意样本作为初始种群，进行演化生成新的恶意样本作为测试集测试 4.3.3 节中训练的分类器，并将误判样本加入训练集中，重新训练分类器。验证在少量样本作为训练集的情况下，加入对抗样本集前后，分类器的分类性能是否有提高。训练集由数据集下 200 个恶意样本和 100 个良性样本构成，测试集由同数据集下 400 个恶意样本，200 个良性样本构成。实验采用三个数据集进行对比验证。

（1） Drebin 数据集分类结果

将 Drebin 数据集作为恶意数据集进行实验，实验结果如图 4.8 和表 4.6 所示。

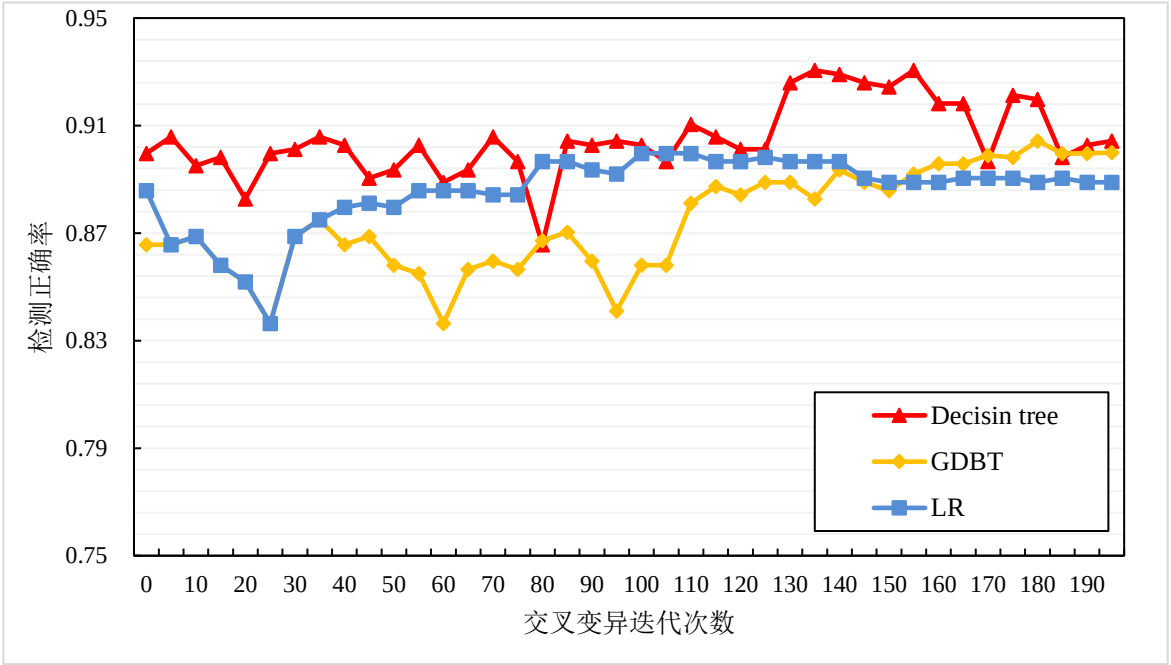


图 4.8 Drebin 数据集分类算法准确率与交叉变异迭代次数关系

图 4.8 表示三种机器学习算法加入对抗样本前后准确率与迭代数关系，其中决策树算法和 GDBT 算法的准确率在加入对抗样本前后提升效果比较明显，决策树算法在 130 代时，准确率达到最高 92.75%，GDBT 在 180 代准确率达到最高 90.23%，LR 算法的提升效果较弱，在 110 代达到最高值 89.81%。

表 4.6 中实验结果表明，在加入对抗样本集后，分类器的准确率，召回率和精确度有了明显提高。具体来说，决策树算法和 GDBT 算法的准确率在加入对抗样本前后提升效果比较明显提高了 2%-4%，其中 LR 算法的精确度提高了 4.75%。

表 4.6 Drebin 数据集加入对抗样本前后分类算法的检测结果

采用方法	加入对抗样本前后	ACC/%	RC/%	PR/%
决策树	前	89.97	93.96	89.50
	后	**92.75**	**95.14**	**93.00**
GDBT	前	86.57	84.55	95.75
	后	**90.28**	**87.36**	**98.50**
LR	前	88.58	89.95	91.75
	后	**89.81**	**88.13**	**96.50**

（2） Maldroid 数据集分类结果

使用 Maldroid 数据集作为恶意数据集进行实验。实验结果如图 4.9 及表 4.7 所示。

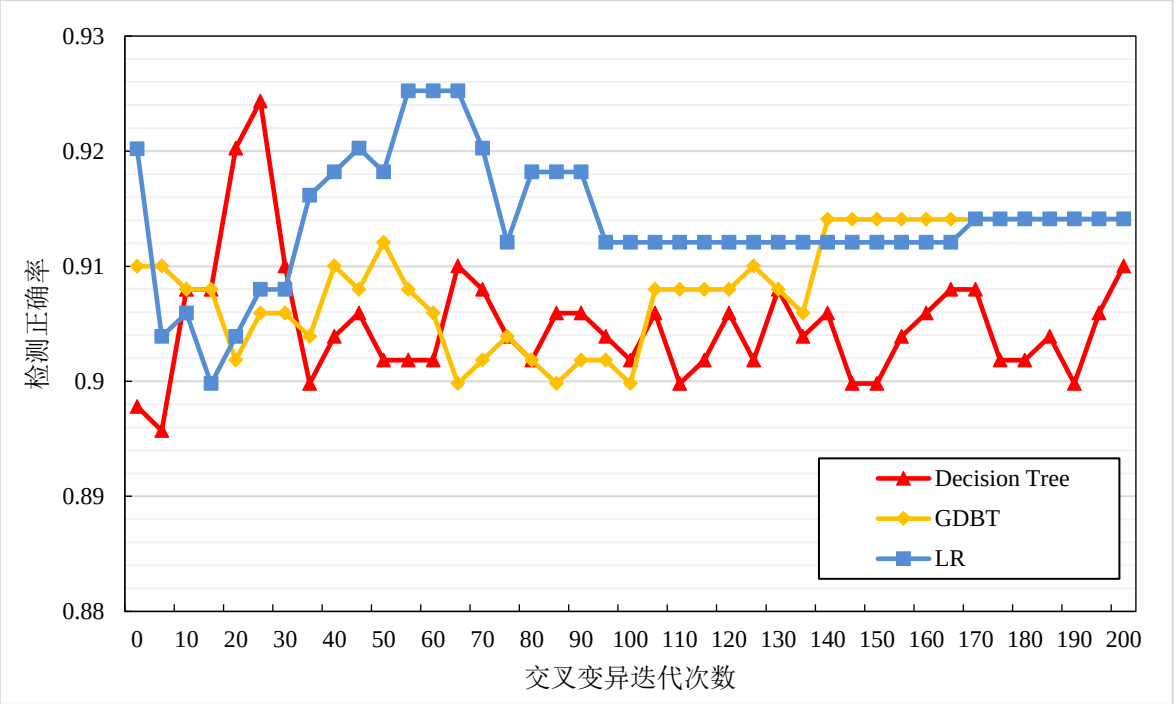


图 4.9 Maldroid 数据集分类算法准确率与交叉变异迭代次数关系

由图 4.9 可知，其中决策树算法在 25 代时，准确率达到最高 92.43%，提升了 2.7%。而 LR 算法和 GDBT 算法的准确率在加入对抗样本前后检测准确率并无明显变化，提升了不到 1%。

表 4.7 Maldroid 数据集加入对抗样本前后分类算法的检测结果

采用方法	加入对抗样本前后	ACC/%	RC/%	PR/%
决策树	前	89.78	88.72	96.14
	后	**92.44**	**91.21**	**96.78**
GDBT	前	91.00	88.72	96.79
	后	**91.41**	**89.44**	**98.07**
LR	前	92.02	89.77	98.71
	后	**92.54**	**90.53**	**99.03**

由表 4.7 可知，使用本方法后，分类效果提升效果并不明显，其中三种算法的准确

率提高范围在 0.4%-4%，其中加入对抗样本后 GDBT 算法准确率为 90.28%，较之前提升 3.71%。

（3） VirusShare 数据集分类结果

使用 VirusShare 数据集作为恶意数据集进行实验。实验结果见图 4.10 和表 4.8。

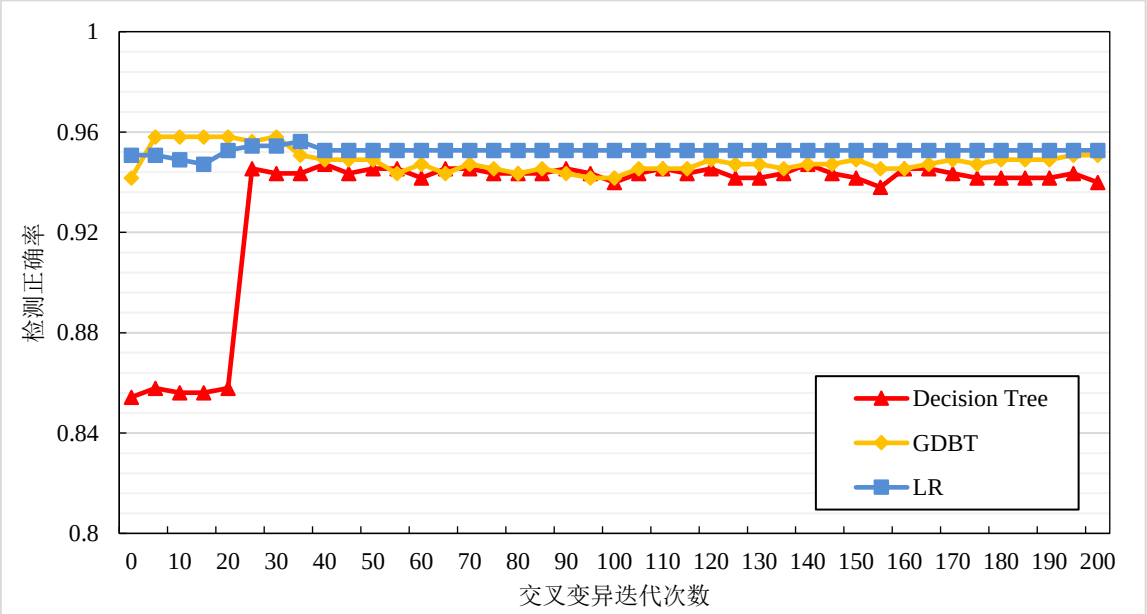


图 4.10 VirusShare 数据集分类算法准确率与交叉变异迭代次数关系

如图 4.10 所示，决策树算法准确率在加入对抗样本前后提升效果比较明显，该算法在第 25 代时，准确率达到最高 94.71%，提升了 9.7%，而 GDBT 在第 5 代准确率达到最高 95.81%，LR 算法的提升效果较弱，在第 35 代达到最高值 95.62%。

表 4.8 VirusShare 数据集加入对抗样本前后分类算法的检测结果

采用方法	加入对抗样本前后	ACC/%	RC/%	PR/%
决策树	前	85.43	80.40	96.64
	后	**94.71**	**95.89**	**96.06**
GDBT	前	94.17	94.80	97.20
	后	**95.81**	**96.91**	**96.63**
LR	前	95.08	96.07	96.35
	后	**95.62**	**95.86**	**97.47**

由表 4.8 可知，在当前数据集下，决策树算法的提升效果最为明显，其中加入对抗样本前后决策树的准确率分别为 85.43%和 94.71%，准确率提升 9.7%，召回率提升 5.49%。

综合以上实验可知，本章方法在不同数据集中，针对不同机器学习算法均有不同程度的提升效果，其中在 VirusShare 数据集中，决策树算法的提升效果最为明显，准确率提升 9.7%，召回率提升 5.49%，证明了本文方法的有效性。

4.4 本章小结

本章介绍了基于生成对抗样本的应用程序恶意性检测方法。该方法首先使用恶意样本的回调函数控制流图作为染色体进行演化，详细介绍了交叉算子和变异算子的具体操作以及三种规则约束。同时将良性样本和恶意样本中提取的特征进行特征变换采用机器学习算法训练分类器，使用演化生成的对抗样本测试分类器，将误判样本加入训练集中重新训练分类器。通过实验，分析了对抗样本对现有工具以及本章分类器的干扰效果，并分析了该方法对三个不同数据集上三个机器学习算法构造的分类器检测率的提升效果，验证本方法的有效性。

5 基于图相似的家族预测方法

本章介绍了基于图相似的家族预测方法，方法整体流程如图 5.1 所示。该方法基于第三章介绍的回调函数控制流图，首先将所有获取样本内所有回调函数控制流图的节点和有向边，构造应用程序的邻接矩阵和家族邻接矩阵，并对家族中每条有向边的进行权值计算用来构造家族权值邻接矩阵，最后计算图之间的相似性判断样本所属家族，并通过实验验证本章方法的有效性。

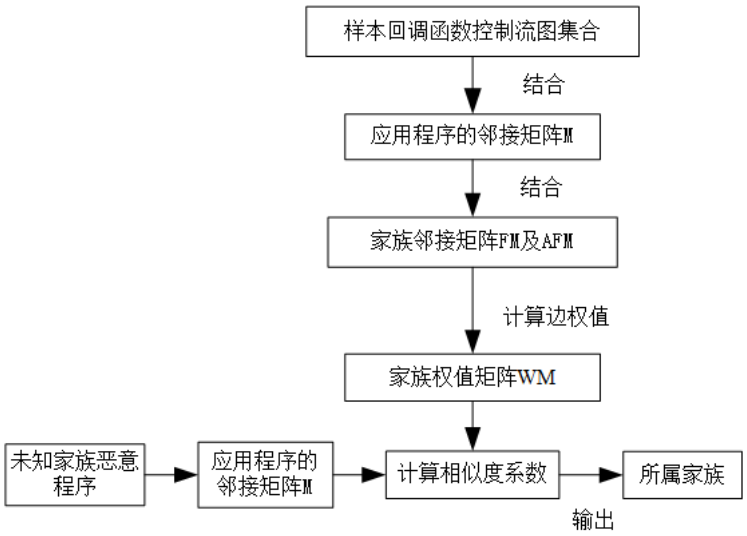


图 5.1 家族分类流程图

5.1 构造应用程序的邻接矩阵

在进行有向图的特征计算时，使用邻接矩阵来表示有向图，能够有效提高我们对有向图中特征的提取效率。

给定一个应用程序A，$\{m_0, m_1, \cdots, m_k\}$是应用程序$A$中所有回调函数的集合，$iCFG_i =< Units_i, Edges_i, Head_i, Tail_i, L_i >$是应用程序$A$中回调函数$m_i(i \in \{0,1,2,\cdots,k\})$对应的控制流图。设$Head_i \subseteq Units_i$，$Tail_i \subseteq Units_i$分别为$iCFG_i$的入口节点和出口节点，则$Heads = \bigcup_{i\in\{0,1,2,\cdots,k\}} Head_i, Tails = \bigcup_{i\in\{0,1,2,\cdots,k\}} Tail_i$分别为应用程序$A$中所有回调函数控制流图的入口节点集合和出口节点集合。

我们定义应用程序A的邻接矩阵M为一个$N \times N$阶矩阵，其中N为应用程序A的节点集中节点数，其行和列对应于应用程序A中的节点集$Units_A^*$，其中$Units_A^* = (\bigcup_{i\in\{0,1,2,\cdots,k\}} Units_i) - (Heads \cup Tails)$。对于任意一回调函数边集$Edges_i$中的每条有向边$(u, v)$，在邻接矩阵$M$的$M_{u,v}$元素取值为 1。

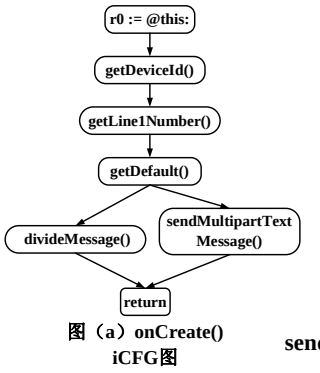
一个应用程序有多个回调函数控制流图，因此首先遍历应用程序内所有回调函数控制流图，获取图中所有节点之后，构造邻接矩阵。接着提取所有回调函数控制流图的有向边，将邻接矩阵内相应位置的值改为 1。

算法 5.1 应用程序的 iCFG 图邻接矩阵构造算法

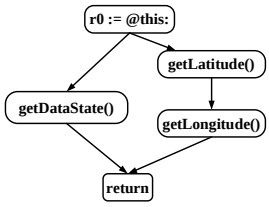
输入：应用程序A内所有回调函数控制流图集合$A_Set = \{iCFG_1, iCFG_2, \cdots, iCFG_m\}$
输出：应用程序A邻接矩阵M

```
1   for each iCFG_m in A_i_Set
2       for each u in Units_m
3           if u ∉ Units_A^*
4               Units_A^* add u
5       end
6   for each iCFG_m in A_i_Set
7       for each (u, v) in Edges_m
8           M_{u,v} = 1
9       end
10  end
11  end
```

如算法 5.1 所示，第 2-5 行为遍历$iCFG_m$图中所有节点u，如果节点u不在节点集$Units_A^*$中，则将节点u添加至节点集$Units_A^*$中。在第 6-10 行中，对 $Edges_m$中每条有向边(u, v),，$M_{u,v}$值为 1。



图（a）onCreate() iCFG图



图（b）onRusue() iCFG图

	r0 := @this:	getDeviceId()	getLine1Number()	getDefault()	divideMessage()	sendMultipartTextMessage()	getDataState()	getLatitude()	getLongitude()	return
r0 := @this:	0	1	0	0	0	0	1	1	0	0
getDeviceId()	0	0	1	0	0	0	0	0	0	0
getLine1Number()	0	0	0	1	0	0	0	0	0	0
getDefault()	0	0	0	0	1	1	0	0	0	0
divideMessage()	0	0	0	0	0	0	0	0	0	1
sendMultipartTextMessage()	0	0	0	0	0	0	0	0	0	1
getDataState()	0	0	0	0	0	0	0	0	0	1
getLatitude()	0	0	0	0	0	0	0	0	1	0
getLongitude()	0	0	0	0	0	0	0	0	0	1
return	0	0	0	0	0	0	0	0	0	0

图（c）应用程序 的邻接矩阵

图 5.2 应用程序的邻接矩阵构造过程

如图 5.2 所示，图（a）与图（b）分别为某应用程序的两个回调函数控制流图，首先遍历图（a）中所有节点以及图（b）中所有节点，并去掉重复的节点后构造10×10阶矩阵M，然后获取两图中所有的有向边，将矩阵中相应位置的值记为 1。例如，对于

有向边（getDeviceId()，getLineNumber()），则矩阵中$M_{2,3}$的值为 1。图（c）为合并了 onCreate()控制流图和 onRusue()控制流图后，构造的应用程序的邻接矩阵。

5.2 构造家族权值邻接矩阵

给定一个家族 F，$\{M^1, M^2, \cdots, M^m\}$是家族 F 中所有应用程序的邻接矩阵的集合，其中$\{Units_0^*, Units_1^*, \cdots, Units_m^*\}$是家族 F 中$M^i(i \in \{0,1,2,\cdots,m\})$对应应用程序的邻接矩阵的节点集集合。

定义家族 F 的邻接矩阵为FM是一个$N \times N$阶矩阵，其中 N 为家族 F 中的节点集中节点数，矩阵的行和列对应于家族中的节点集 $Units'$，其中 $Units' = \bigcup_{i\in\{0,1,2,\cdots,m\}} Units_i^*$。假设$M_{u,v}^m$值为 1，其中$u, v \in Units'$，则$FM_{u,v}$的值为 1。

为计算边在家族中的频率，定义家族F中边关系矩阵AFM是一个$N \times N$阶矩阵，其中 N 为家族 F 中的节点集中节点数，矩阵的行和列对应于家族中的节点集$Units'$其中 $Units' = \bigcup_{i\in\{0,1,2,\cdots,m\}} Units_i^*$。假设$M_{u,v}^m$值为 1，其中$u, v \in Units'$，则$AFM_{u,v}$的值加 1。

定义 5.1（**TP**）描述某边在该条边的源节点中的所有边中所占的比例，在家族F_i中边(u,v)的概率计算公式为：

$$TP_{Fi}(u,v) = \frac{AFM_{u,v}^i}{\sum_{j=1}^{n}\left|AFM_{u,j}^i\right|} \tag{5.1}$$

其中n为家族F_i的节点数。

定义 5.2（**IDF**）描述某边在所有家族中的存在的频率，在k个家族中边(u,v)的 IDF 计算公式为：

$$IDF(u,v) = \log\left(\frac{k}{\sum_{i=1}^{k}|FM_{u,v}^i|}\right) \tag{5.2}$$

则在家族F_i中边(u,v)权值计算公式为：

$$W_{Fi}(u,v) = IDF(u,v) * TP_{Fi}(u,v) \tag{5.3}$$

计算F_i权值矩阵$FM_i \rightarrow WM_i$。

算法 5.2 家族权值矩阵构造算法

输入：家族集合$F = \{F_1, F_2, \cdots, F_k\}$,家族$F_i$中应用程序邻接矩阵集合$M^{Fi}_set = \{M^1, M^2, \cdots, M^n\}$,其中$i = \{1,2,3,\cdots,k\}$;

输出：家族权值矩阵集合WM_set=$\{WM^1, WM^2, \cdots, WM^k\}$

```
F_Units_set = { Units'_1, Units'_2, ..., Units'_k }, FM_set = {FM^1, FM^2, ..., FM^k}, AFM_set = {AFM^1, AFM^2, ..., AFM^k};
for i in 1 to k
    for each M^n ∈ M^Fi_set
        Units'_i = Units'_i ∪ Units*_n
    end
```

```
6       F_Units_set.add Units_i'
7   end
8   for i in 1 to k
9       for each M^n ∈ M^Fi_set
10          if M_{u,v}^n = 1  FM_{u,v}^i = 1, AFM_{u,v}^i = AFM_{u,v}^i + 1
11      end
12      FM_set.add FM^i, AFM_set.add AFM^i
13  end
14  for i in 1 to k
15     for each FM_{u,v}^i in FM^i
16              调用计算公式 5.1 计算TP_Fi(u,v)
17              调用计算公式 5.2 计算IDF(u,v);
18              调用计算公式 5.3 计算W_Fi(u,v)
19              WM_{u,v}^i = W_Fi(u,v)
20      end
21      WM_set.add WM^i
22  end
23  rerturn WM_set
```

算法 5.2 中，第 2-6 行为获取各家族内节点集的过程，家族的节点集为该家族内所有应用程序节点集的并集；第 8-13 行为构造家族邻接矩阵及家族边关系矩阵的过程，对于家族内每一应用程序的邻接矩阵取值为 1 的元素，则该家族邻接矩阵的相应元素取值为 1，家族边关系矩阵的相应元素值加 1；第 14-20 行描述了构造家族权值矩阵的过程，对于家族F_i中每条边(u,v)，使用式（5.1）计算$TP_{Fi}(u,v)$，式（5.2）计算$IDF\ (u,v)$，最后使用式（5.3）计算$W_{Fi}(u,v)$。

5.3 计算相似性系数

本章方法通过计算待测样本与每个家族权值矩阵之间的相似度系数，将待测样本归类到与其相似度最高的家族，对待测样本进行家族分类。基于杰卡德相似系数[42]算法思想，本节定义一组用于衡量两个有向图之间的相似系数计算公式。关于杰卡德相似性的公式如式（5.4）所示，本节相似度系数计算公式为式（5.5）。

定义 5.3 (杰卡德相似系数) 集合 A 与集合 B 的交集元素的个数在集合 A 与集合 B 的并集元素个数中所占的比例，称之为集合 A 与集合 B 的杰卡德相似系数。公式如下：

$$J(A,B) = \frac{A \cap B}{A \cup B} \tag{5.4}$$

本节结合杰卡德相似系数思想，定义应用程序 A 与家族F_i的相似系数的计算公式为：

$$similarity(A,F_i) = \sum_{u \in Units_i'} (\sum_{v \in Units_i'} M_{u,v} \cdot WM_{u,v}^i) \tag{5.5}$$

其中相似系数越大，相似度越高，因此最终会将待测样本判定属于与其相似系数

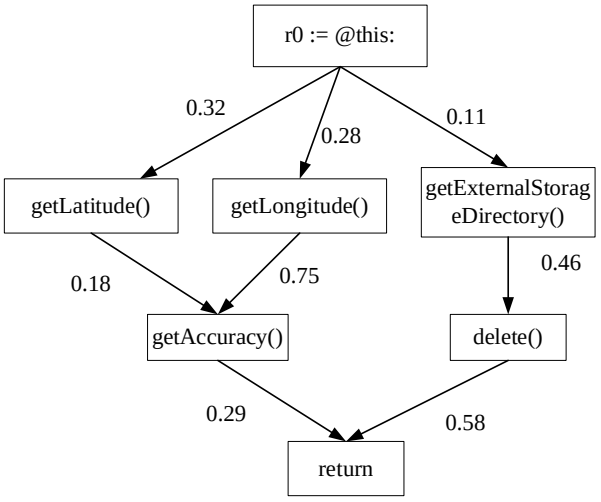
最大的家族。具体过程如算法 5.3 所示。

算法 5.3 家族预测算法

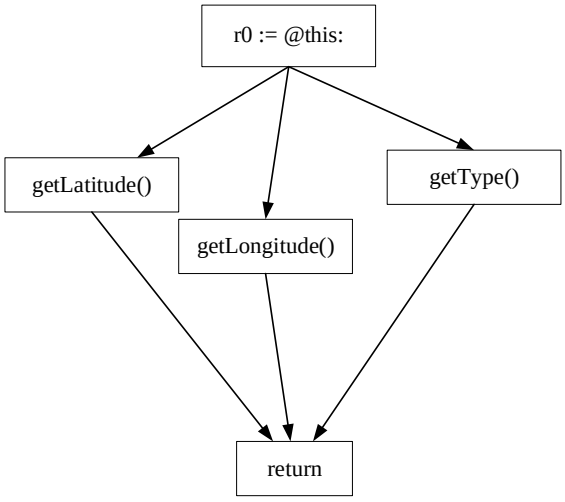
输入：待测样本 A(G), 家族权值矩阵集合WM_set={$WM^1, WM^2, \cdots, WM^k$}
输出：所属家族 F

```
1   double sim , String F;
2   for each WM^k ∈ WM_set
3       调用公式 5.5 计算similarity(A, F_k)
4       if  similarity(A, F_k) >sim
5           sim = similarity(A, F_k);
6           F = F_k;
7    end
8   return F
```

如图 5.3 所示，（a）为 F_i 加权家族控制流图，图（b）为待测样本应用程序的控制流图。首先计算图（a）与图（b）相同节点的相似度，即计算 r0 := @this 节点的相似度，该节点在图 b 中的边集合为{getLatitude(),getLongtitude(),getType()}，在图（a）中的边集合为{getLatitude(),getLongitude(),getExternalStorageDirectory()}，则该点在两图中的相似系数为 0.32+0.28，结果为 0.6，而 getLatitude()节点和 getLongitude()节点两图中调用边的交集为空，则这两点的相似系数为 0，将所有节点的相似系数相加得到两个图的相似度，即 0.6。



图(a) F_i家族加权控制流图



图(b) 待测应用程序控制流图

图 5.3 F_i家族加权控制流图与待测应用程序控制流图

5.4 实验结果与分析

5.4.1 数据集信息

在本文中所用的数据集来自 Drebin 数据集中，其中共 49 个恶意应用家族，2319 个

恶意应用程序，具体信息如表 5.1 所示。

表 5.1 数据集中各家族及样本数量信息

家族名	数量	家族名	数量	家族名	数量	家族名	数量
Adrd	44	FakeRun	8	Kmin	63	SendPay	26
BaseBridge	140	FakeTimer	8	Mania	5	SerBG	6
Boxer	14	Fatakr	8	Mobilespy	5	SMSreg	16
Copycat	5	FoCobers	8	MobileTx	31	Stealer	9
Cosha	6	Gappusin	26	Mobinauten	5	Steek	8
Dougalek	5	Geinimi	42	Nandrobox	8	Trackplus	5
DroidDream	40	GinMaster	163	Nickspy	5	TrojanSMS.Hippo	11
DroidKungFu	317	Glodream	35	Nyleaker	10	Vdloader	5
ExploitLinux Lotoor	30	Hamob	15	Opfake	292	Xsider	7
FakeInstaller	401	Iconosys	67	Penetho	7	Yzhc	17
Fakelogo	7	Imlog	18	Placms	5		
Fakengry	49	Jifake	15	Plankton	287	总计	2319
FakePlayer	5	Kidlogger	5	SeaWeth	5		

5.4.2 家族邻接矩阵分析

对本文收集到的所有恶意软件家族构建家族矩阵及边关系矩阵，并对家族中所有边进行分析。图 5.4 展示了不同家族中有向边的总数。由图 5.4 可知，DroidKungFu 家族的边数量最多，高达 14147 条，这是因为 DroidKungFu 家族中包含 317 个应用程序，恶意应用程序数量较多，且 DroidKungFu 家族包含的恶意行为包括删除文件、打开网页、下载安装 APK 和启动其他程序等。我们推测该家族有向边数量众多的原因是，该家族恶意行为众多，拥有大量变种以及传播范围广。

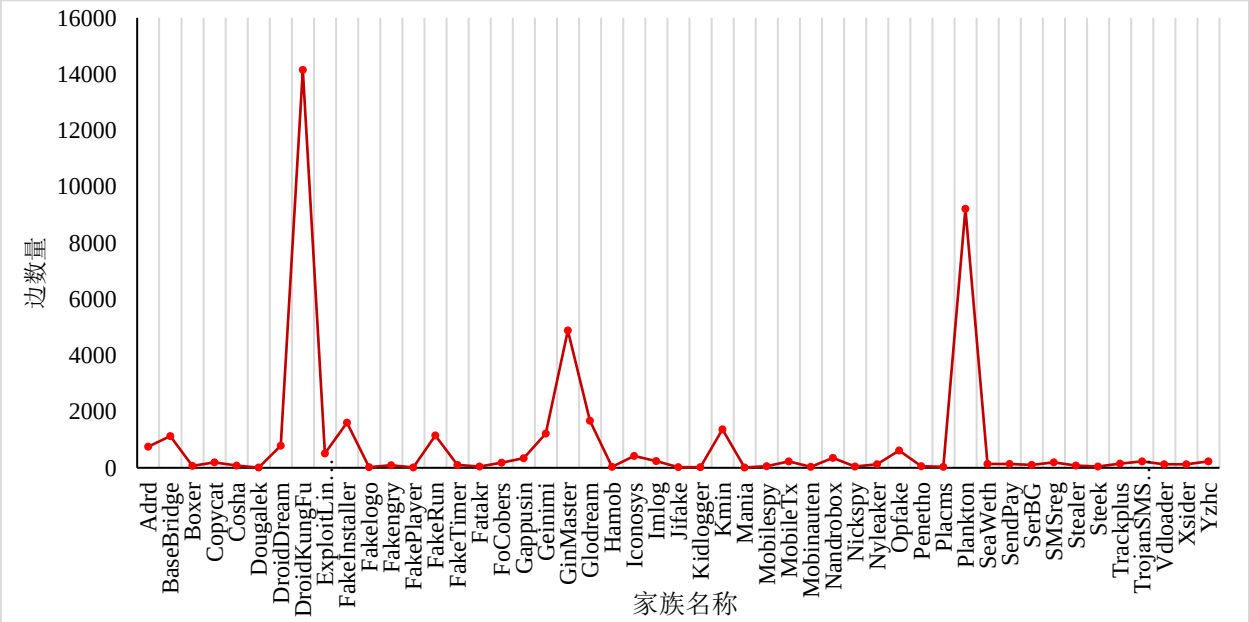


图 5.4 各家族中有向边数量

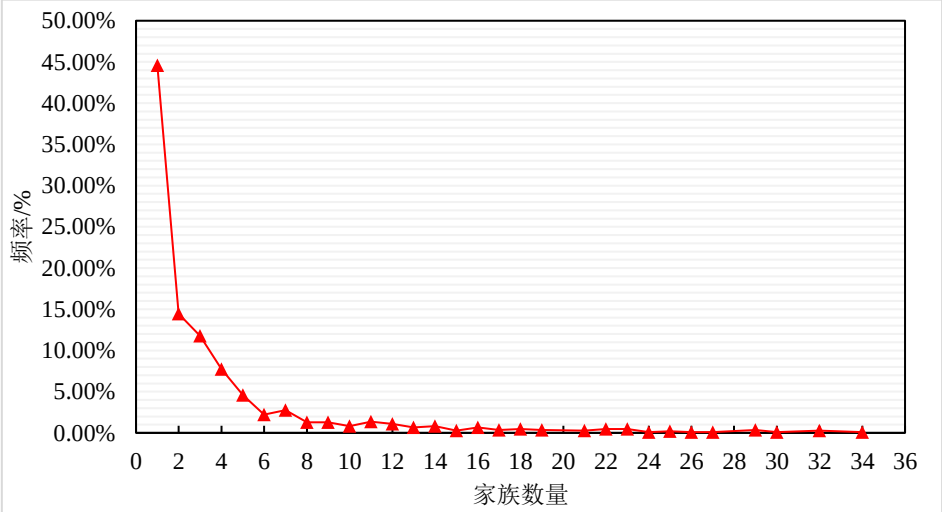


图 5.5 有向边分布家族数量关系

图 5.5 为有向边与其分布的家族数目关系。我们统计了各家族提取的 43383 个有向边在各家族中分布情况。由图 5.5 可知，有 44.58%的有向边只分布在一个家族中，说明这些有向边只出现在一个家族中，因此这些边在其所在家族中的权值较大。而 14.43%的有向边分布在两个家族中，有 4.5%左右的有向边同时分布在 15 个及以上家族。

5.4.3 家族分类实验结果分析

选择各家族中样本数量的 20%作为测试样本，80%作为训练集，通过训练集中的样本构建各家族权值邻接矩阵，计算测试样本与各家族之间的相似度进行家族分类。对于每个家族的预测结果如图 5.6 所示。

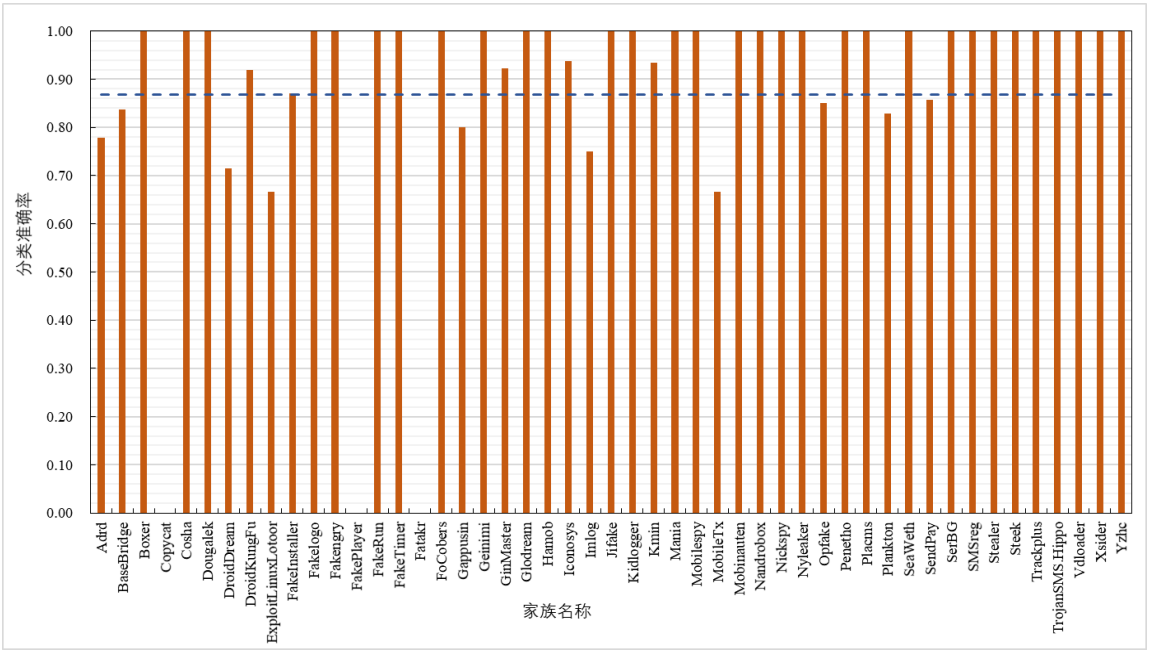


图 5.6 各家族分类实验结果

从图 5.6 中可以看出，本文方法对于大部分恶意软件家族具有良好的分类效果，其中 Kmin 家族、Inconosys 家族、Ginimi 家族和 DroidKungFu 家族的分类正确率均达到 90%以上。而对于 FakePlayer 家族的预测正确率为零，我们对该家族进行分析，发现该家族邻接矩阵的规模较小，只有三个节点以及两条边，且这两条边在所有家族内均有分布，因此 FakePlayer 家族在本文方法中效果较差。

5.4.4 与相关工作比对分析

文献[15]通过将恶意应用的字节码转化为字节码图像，后利用 GIST 算法提取字节码图像的特征，并结合随机森林算法训练家族分类器。图 5.7 为本文方法与文献[15]方法在相同数据集上不同家族检测率的对比情况，两种检测方法在不同的恶意家族上检测效果互有优劣，总体检测效果接近。在 Gappusin 家族中，文献[15]准确率只有 53.4%，本文方法的检测准确率达到 80%，比文献[16]中方法准确率高出 27%，因此本文方法在对某些家族检测率要优于文献[15]方法。

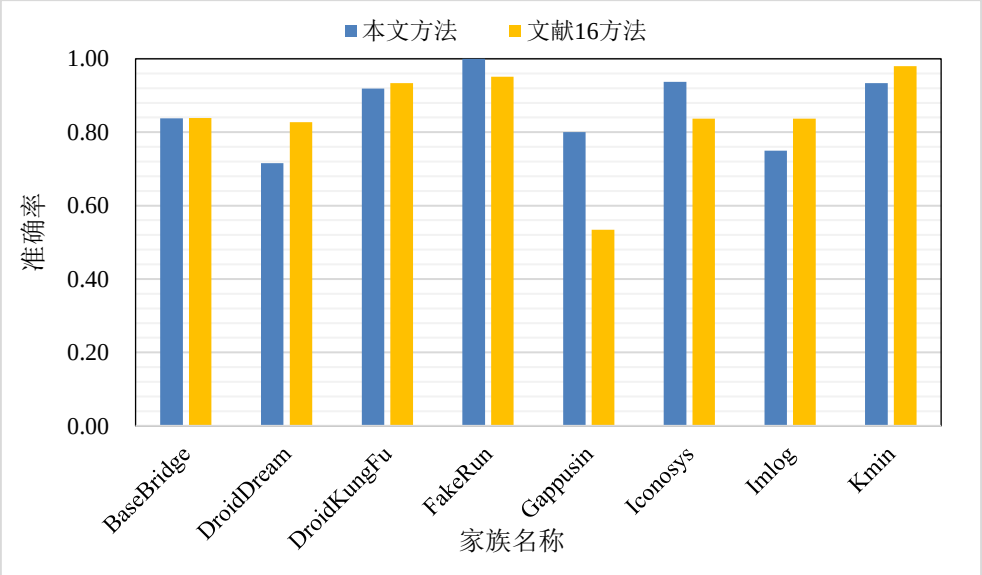


图 5.7 与相关工作对比

5.5 本章小结

本章介绍了基于图相似的家族检测方法，首先介绍了应用程序邻接矩阵的构造过程，接着描述了家族邻接矩阵和家族权值矩阵的构造方法，并定义了边权值的计算公式。基于杰卡德相似度系数思想，本章提出一种计算有向图相似度的计算公式，通过计算待测样本与各家族权值矩阵的相似度，判断其所属家族。最后通过实验分析了该方法的有效性。

6 总结与展望

6.1 工作总结

随着网络和移动终端技术的不断发展，大量 Android 应用程序应运而生，与此同时 Android 恶意应用程序的数量也在不断增加，这严重的威胁着人们的财产及隐私安全。随着无用代码注入、加壳、重打包等混淆技术的发展，使得现阶段如何更高效的区分应用程序的恶意性成为一个艰难的课题。在本文中，通过分析现有 Android 恶意程序的行为模式，以及传统基于机器学习检测方法的不足，提出了一种基于生成对抗样本的应用程序恶意性检测和家族分类方法，该方法不仅能够生成具有干扰性的对抗样本，将对抗样本加入训练集后，还能扩大样本集数量，预测新型恶意样本，提高分类器识别恶意样本的准确率，并对恶意程序进行家族预测。主要的工作内容如下：

（1）为了有效提取应用程序特征，本文使用 Soot 工具对应用程序安装包进行解析，并为每个程序生成函数调用图并约简，接着对约简后的函数调用图中方法，构造控制流图后对其进行约简，将约简的控制流图进行集成生成回调函数控制流图，遍历回调函数控制流图获取敏感 API 序列作为特征。

（2）将恶意样本作为个体，样本内的回调函数控制流图作为染色体，敏感 API 节点作为基因，通过遗传算法对其进行演化生成对抗样本，同时定义了三个种群适应度函数以控制种群演化方向，并定义了三种约束来保证生成对抗样本代码的可行性。

（3）收集多个数据集提供的恶意样本及良性样本，分别抽取特征值构造分类器，同时将对抗样本作为测试集，输入分类器进行恶意性判定，将分类器误判为良性的对抗样本作为误判样本，加入训练集后重新训练分类器。为验证方法的有效性，特选取三个不同的恶意样本库进行了实验，通过三组对照实验验证了本文方法能够有效提高机器学习的分类效果，其中针对 ViusShare 数据集决策树算法的准确率提升了 9.3%，召回率提升了 4.49%。

（4）将恶意应用程序内回调函数控制流图进行合并，构造应用程序邻接矩阵，同时为各家族构造家族邻接矩阵，计算家族中所有边的权值并构建家族权值邻接矩阵，通过计算图之间的相似性系数判断样本所属家族，最后通过实验表明了该方法对恶意程序具有良好的分类效果。

6.2 展望

在本课题中，从回调函数控制流图中提取敏感 API 序列作为程序特征，采用机器

学习进行分类器训练，同时使用遗传算法对恶意样本交叉变异生成对抗样本，将分类器误判的对抗样本加入训练集进行重训练获取最终分类器，并且在对程序恶意性判断的基础上预测了恶意程序所属的恶意家族，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）虽然本文中使用敏感 API 序列作为程序特征，但是敏感 API 序列为细粒度特征，因此本课题中构造的分类器具有较高的精确度。但过细粒度的特征会降低检测准确率，且训练时间及检测时间较长，因此未来会对敏感 API 序列进行模糊化处理，在提高检测率的同时降低程序运行时间。

（2）本文中只使用了回调函数控制流图作为染色体进行交叉变异，但是，单个特征很难完全描述应用程序的恶意行为，因此我们后续将提取应用程序中的其他特征，如权限、广播、组件等，然后选取多个特征同时作为染色体进行交叉演化，并组合生成基于多特征的对抗样本，用该方法生成的对抗样本将更具有恶意样本的特性。

致 谢

三年的学习时光转瞬而逝，随着毕业论文的完成，研究生学习生活即将结束。于我来说，这是一段难忘又珍贵的回忆。它不仅仅是“路漫漫其修远兮，吾将上下而求索”的三年，也是增长阅历、挫折与荣誉同在的三年。这期间的点滴收获，都离不开老师的教诲，同学的帮助以及家人的支持。

我由衷的感谢我敬爱的导师***老师，感谢***老师对我的教导和栽培。三年来，***老师对我的学习和研究都提供了细心的指导，使我受益匪浅。***老师渊博的专业知识，严谨的工作态度以及精益求精的工作作风都是我毕生学习的楷模。本论文从开始选题，中期考核到最终成型，***老师都给予了我大量的指导意见。感谢老师在学术上对我的严格要求，让我在学习中增长了见识，拓宽了眼界。在此，对导师三年来对我学术上的指导以及生活中的关怀表示崇高的敬意和最衷心的感谢。

感谢我的同学朋友以及***师姐，***师姐，***和***师弟。在我遇到困难时给予我帮助，不断开导我并帮我走出困境。感谢师姐师弟们对我的支持和鼓励，同门的情谊让我终身难忘。感谢室友对我生活上小缺点的容忍，我们一起生活一起成长，一起彼此陪伴度过一个又一个的夜晚。酒逢知己千杯少，难得在漫漫人生路上认识你们，一起走过的日子，非常感谢你们对我的帮助。

感谢一直陪在我身边的男朋友***，你的出现让我拥有了爱与被爱的能力。因为有你的帮助，我才能够克服撰写代码中的重重困难，才可以顺利的完成研究内容。感谢你陪我走过的风风雨雨，感谢你无限包容我的小脾气。希望未来路上我们能够相守相知，一路同行。

感谢呵护我成长的父母，回顾走过的路，每一步都浸满他们无私的关爱和谆谆教诲，十多年的求学路，寄托着父母对我的殷切希望，他们在物质上和精神上的无私支持，坚定了我追求人生理想的信念。

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] 曾健平, 邵艳洁. Android 系统架构及应用程序开发研究[J]. 微计算机信息, 2011, 27(9): 3.

[2] StatCounter 中国 Android 系统统计情况[EB/OL]. https://gs.statcounter.com/. 2022-3-10:

[3] 360 互联网安全中心 2021 年中国手机安全状况报告[EB/OL]. https://pop.shouji.360.cn/safe_report/Mobile-Security-Report-202112.pdf . 2022-2-23:

[4] 赵淑钰.《中华人民共和国网络安全法》实施三年取得显著成绩 以法治化构筑网络安全坚实屏障[J]. 网络传播, 2020(06): 34-37.

[5] 舒敏, “移动互联网恶意程序描述格式”等 6 项行业标准. 北京市, 国家计算机网络应急技术处理协调中心, 2015-11-05.

[6] Romli R N, Zolkipli M F, Osman M Z. Efficient feature selection analysis for accuracy malware classification[C]//Journal of Physics: Conference Series. IOP Publishing, 2021, 1918(4): 042140.

[7] Kim J, Lee S. Malicious Behavior Detection Method Using API Sequence in Binary Execution Path[J]. Tehnički vjesnik, 2021, 28(3): 810-818.

[8] 赵赛, 刘昊, 王雨峰, 苏航, 燕季薇. Android 组件间通信的模糊测试方法[J]. 计算机科学, 2020, 47(S2): 303-309+315.

[9] Chen H, Tiu A, Xu Z, et al. A permission-dependent type system for secure information flow analysis[C]//2018 IEEE 31st Computer Security Foundations Symposium (CSF). IEEE, 2018: 218-232.

[10] Ndibanje B, Kim K H, Kang Y J, et al. Cross-method-based analysis and classification of malicious behavior by api calls extraction[J]. Applied Sciences, 2019, 9(2): 239.

[11] 王佳扬. 基于静态代码分析的 Android 恶意应用检测技术研究[D]. 北京邮电大学, 2020.

[12] 鄢然. 基于机器学习的 Android 恶意代码检测研究[D]. 重庆大学, 2019.

[13] 雷倩. 基于多上下文特征的 Android 恶意程序检测和家族分类方法研究[D]. 西安科技大学, 2020.

[14] 章瑞康, 周娟, 袁军, 李文瑾, 顾杜娟. SimMal：基于异构图学习的恶意软件关联分析系统[J]. 信息技术与网络安全, 2021, 40(11): 8-15.

[15] 杨益敏, 陈铁明. 基于字节码图像的 Android 恶意代码家族分类方法[J]. 网络与信息

安全学报, 2016, 2(06): 38-43.

[16] 王淞鹤. 基于运行时特征的 Android 恶意代码探查方法的研究与实现[D]. 北京邮电大学, 2021.

[17] Arshad S, Shah M A, Wahid A, et al. SAMADroid: a novel 3-level hybrid malware detection model for android operating system[J]. IEEE Access, 2018, 6: 4321-4339.

[18] 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, 13(8): 1890-1905.

[19] 林舒婕. 基于函数调用图的 Android 恶意程序检测技术研究与实现[D]. 北京邮电大学, 2017.

[20] 高珍祯. 基于字符串和函数调用图特征的安卓恶意应用检测方法[D]. 北京交通大学, 2019.

[21] Zhi X U, Ren K, Song F. Android malware family classification and characterization using CFG and DFG[C]//2019 International Symposium on Theoretical Aspects of Software Engineering (TASE). IEEE, 2019: 49-56.

[22] 韩潇宁. 基于遗传算法的安卓恶意软件检测技术[D]. 大连理工大学, 2021.

[23] Wang J, Jing Q, Gao J, et al. SEdroid: A robust Android malware detector using selective ensemble learning[C]//2020 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, 2020: 1-5.

[24] Martín A, Menéndez H D, Camacho D. MOCDroid: multi-objective evolutionary classifier for Android malware detection[J]. Soft Computing, 2017, 21(24): 7405-7415.

[25] Wang L, Gao Y, Gao S, et al. A New Feature Selection Method Based on a Self-Variant Genetic Algorithm Applied to Android Malware Detection[J]. Symmetry, 2021, 13(7): 1290.

[26] Firdaus A, Anuar N B, Karim A, et al. Discovering optimal features using static analysis and a genetic search based method for Android malware detection[J]. Frontiers of Information Technology & Electronic Engineering, 2018, 19(6): 712-736.

[27] Zhang L, Wang X, Lu K, et al. An efficient framework for generating robust adversarial examples[J]. International Journal of Intelligent Systems, 2020, 35(9): 1433-1449.

[28] 陈梦轩, 张振永, 纪守领, 魏贵义, 邵俊.图像对抗样本研究综述[J]. 计算机科学, 2022, 49(02): 92-106. 6.

[29] Kawai M, Ota K, Dong M. Improved malgan: Avoiding malware detector by leaning

cleanware features[C]//2019 international conference on artificial intelligence in information and communication (ICAIIC). IEEE, 2019: 040-045.

[30] 黄天波, 李成扬, 刘永志, 李燈辉, 文伟平. 基于 LIME 的恶意代码对抗样本生成技术[J]. 北京航空航天大学学报, 2022, 48(02): 331-338.

[31] 唐川, 张义, 杨岳湘, 施江勇. DroidGAN: 基于 DCGAN 的 Android 对抗样本生成框架[J]. 通信学报, 2018, 39(S1): 64-69.

[32] 肖茂, 郭春, 申国伟, 蒋朝惠. 可保留可用性和功能性的对抗样本[J/OL]. 计算机科学与探索: 1-13 [2022-04-22]. http://kns.cnki.net/kcms/detail/11.5602.TP.20210518.1536.011.html.

[33] Meng G, Xue Y, Mahinthan C, et al. Mystique: Evolving android malware for auditing anti-malware tools[C]//Proceedings of the 11th ACM on Asia conference on computer and communications security. 2016: 365-376.

[34] Khokhlov I, Reznik L. Android system security evaluation[C]//2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE, 2018: 1-2.

[35] Arzt S, Rasthofer S, Bodden E. The soot-based toolchain for analyzing android apps[C]//2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft). IEEE, 2017: 13-24.

[36] Arzt S, Rasthofer S, Fritz C, et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps[J]. Acm Sigplan Notices, 2014, 49(6): 259-269.

[37] Mirjalili S. Genetic algorithm[M]//Evolutionary algorithms and neural networks. Springer, Cham, 2019: 43-55.

[38] Arp D, Spreitzenbarth M, Hubner M, et al. Drebin: Effective and explainable detection of android malware in your pocket[C]//Ndss. 2014, 14: 23-26.

[39] Au K W Y, Zhou Y F, Huang Z, et al. Pscout: analyzing the android permission specification[C]//Proceedings of the 2012 ACM conference on Computer and communications security. 2012: 217-228.

[40] Mahdavifar S, Kadir A F A, Fatemi R, et al. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning[C]//2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE, 2020: 515-522.

[41]陈均. 调用路径驱动的 Android 应用程序恶意行为检测方法研究[D]. 西安电子科技大学, 2019.

[42]Niwattanakul S, Singthongchai J, Naenudorn E, et al. Using of Jaccard coefficient for keywords similarity[C]//Proceedings of the international multiconference of engineers and computer scientists. 2013, 1(6): 380-384.

附 录

硕士攻读学位期间所发表的学术论文:

[1] ***, ***. Research on Family Classification Based on Graph Similarity [C]// 2022 International Conference on Internet and Cyber Security Technology(ICICST 2022).2022(已录用)