软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(6):2526−2542 [doi: 10.13328/j.cnki.jos.006848]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

图卷积网络的抗混淆安卓恶意软件检测*

吴月明[1,2,3], 齐 蒙[1,2,3], 邹德清[1,2,3], 金 海[1,4]

[1](大数据技术与系统国家地方联合工程研究中心 (服务计算技术与系统教育部重点实验室 华中科技大学), 湖北 武汉 430074)

[2](分布式系统安全湖北省重点实验室, 湖北 武汉 430074)

[3](华中科技大学 网络空间安全学院, 湖北 武汉 430074)

[4](华中科技大学 计算机科学与技术学院, 湖北 武汉 430074)

通信作者: 邹德清, E-mail: deqingzou@hust.edu.cn

摘 要: 自安卓系统发布以来, 由于其开源、硬件丰富和应用市场多样等优势, 该系统已成为全球使用最广泛的手机操作系统. 同时, 安卓设备和安卓应用的爆炸式增长也使其成为 96%移动恶意软件的攻击目标. 在现有的安卓恶意软件检测方法中, 忽视程序语义而直接提取简单程序特征的方法, 其检测速度快但精确度不够理想, 将程序语义转换为图模型并采用图分析的方法, 其精确度虽高但开销大且扩展性低. 为了解决上述挑战, 将应用的程序语义提取为函数调用图, 在保留语义信息的同时, 采用抽象 API 技术将调用图转换为抽象图, 以减少运行开销并增强鲁棒性. 基于得到的抽象图, 以 Triplet Loss 损失训练构建基于图卷积网络的抗混淆安卓恶意软件分类器 SriDroid. 对 20 246 个安卓应用进行实验分析后发现: SriDroid 可以达到 99.17%的恶意软件检测精确度, 并具有良好的鲁棒性.

关键词: 安卓恶意软件; 抗混淆; 函数调用图; 抽象 API; 图卷积网络
中图法分类号: TP311

中文引用格式: 吴月明, 齐蒙, 邹德清, 金海. 图卷积网络的抗混淆安卓恶意软件检测. 软件学报, 2023, 34(6): 2526–2542. http://www.jos.org.cn/1000-9825/6848.htm

英文引用格式: Wu YM, Qi M, Zou DQ, Jin H. Obfuscation-resilient Android Malware Detection Based on Graph Convolution Neural Networks. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2526–2542 (in Chinese). http://www.jos.org.cn/1000-9825/6848.htm

Obfuscation-resilient Android Malware Detection Based on Graph Convolutional Networks

WU Yue-Ming[1,2,3], QI Meng[1,2,3], ZOU De-Qing[1,2,3], JIN Hai[1,4]

[1](National Engineering Research Center for Big Data Technology and System (Key Laboratory of Services Computing Technology and System, Ministry of Education, Huazhong University of Science and Technology), Wuhan 430074, China)

[2](Hubei Key Laboratory of Distributed System Security, Wuhan 430074, China)

[3](School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China)

[4](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: Since the release of Android, it has become the most widely used mobile phone operating system in the world due to its advantages such as open source, rich hardware, and diverse application markets. At the same time, the explosive growth of Android devices and Android applications (app for short) has made it a target of 96% of mobile malware. Among current detection methods, the

* 基金项目: 国家自然科学基金(62172168); 湖北省重点研发计划(2021BAA032)
本文由“软件可信性与供应链安全前沿进展”专题特约编辑向剑文教授、郑征教授、申文博研究员、常瑞副教授、田聪教授推荐.
收稿时间: 2022-09-05; 修改时间: 2022-10-10; 采用时间: 2022-12-14; jos 在线出版时间: 2023-01-13

direct extraction of simple program features, ignoring the program semantics is fast but less accurate, and the conversion of semantic information of programs into graph models for analysis improves accuracy but has high runtime overhead and is not very scalable. To address these challenges, the program semantics of an App is distilled into a function call graph and the API call is abstracted to convert the call graph into a simpler graph. Finally, these vectors are fed into a graph convolution network (GCN) model to train a classifier with triplet loss (i.e., SriDroid). After conducting experimental analysis on 20 246 Android apps, it is found that SriDroid can achieve 99.17% malware detection accuracy with sound robustness.

Key words: Android malware; obfuscation-resilient; function call graph; abstract API; graph convolutional network (GCN)

随着移动智能手机的快速发展, 移动应用程序凭借其功能多样性以及能够提供便捷服务等优点, 已成为人们日常生活中不可或缺的一部分. 据国家相关部门统计结果显示, 截至 2022 年 6 月, 有 99.6%的网民在日常生活中使用手机上网[1]. 根据 360 安全实验室发布的 2022 年第 1 季度中国手机安全状况报告[2], 2022 年第 1 季度, 360 安全大脑在网络上共截获移动端新增恶意程序样本约 499.0 万个, 平均每天截获新增手机恶意程序样本约 5.5 万个. 为了保护安卓系统的生态和提高用户体验, 设计高效的安卓恶意软件检测工具具有十分重要的意义.

迄今为止, 已有多种安卓恶意软件检测方法被提出. 传统上的基于权限[3]和 API[4]的检测方法具有分析速度快的优势, 但是由于未考虑程序的语义信息, 其检测效果往往不够理想, 并且容易被攻击者通过代码混淆技术加以攻击. 为了获得更精确的检测效果, 也有研究者将程序语义抽象成不同的图模型[5,6], 并采用图分析方法来检测恶意软件. 然而, 图的相似性度量属于 NP 难问题, 随着安卓应用越来越复杂, 图相似性的计算会产生难以接受的运行开销. 整体而言, 现有的恶意软件检测系统主要存在如下 3 个缺点.

- 缺乏语义信息: 基于字符串的特征不足以区分正常应用和恶意应用. 例如: 用于获取位置信息的敏感 *APIgetLastKnownLocation*(·)在恶意应用和正常应用中均被广泛使用, 在不考虑完整语义信息的情况下, 检测准确率较低[7];
- 鲁棒性差: 基于权限、字符串、API 的检测方法由于仅在应用中检索是否存在相关特征, 导致简单的代码混淆都可以使其失效. 例如: 经过研究者的设计, 通过对恶意软件样本的源文件进行很小的扰动, 即可让检测器的检测效果大幅下降[8];
- 可扩展性低: 基于图分析的方法在检测恶意软件方面效果较为理想, 然而高开销的程序分析以及复杂的图分析使得这些方法的扩展性较低. 例如: MaMaDroid 基于函数调用图进行恶意软件检测, 但是由于高额的运行开销和内存占用, 使其可扩展性较低[9].

为了解决上述挑战, 本文提出了基于图卷积网络的抗混淆安卓恶意软件分类器 SriDroid. SriDroid 首先采用静态分析技术提取应用的函数调用图, 从而保留应用的程序语义; 然后, 对函数调用图中的函数进行抽象处理, 将函数转化成对应的类(class)、包(package)或者家族(family), 以此构建出更为简单的抽象图; 最后,以 Triple Loss 对比损失训练图卷积网络分类器. 为了考虑更全面的语义信息, SriDroid 使用包含结构信息的函数调用图来分析应用的行为, 并进一步通过图卷积网络聚合邻居特征, 从而提升检测的准确性. 而抽象图的提取和 Triple Loss 对比损失的应用, 使得 SriDroid 在抵抗代码混淆方面具有更强的鲁棒性, 同时还降低了图的复杂度, 使得 SriDroid 的检测流程有着更低的运行开销和更好的扩展性.

为了测试 SriDroid 在安卓恶意软件检测上的实际效果, 实验从 AndroZoo[10]以及 Virushare[11]中一共下载了 10 757 个正常应用和 9 489 个恶意应用, 并从有效性、鲁棒性以及扩展性这 3 个方面分别对 SriDroid 进行了详细的实验分析. 在有效性方面, SriDroid 实现了 99.17%的恶意软件检测精确度; 在鲁棒性方面, SriDroid 可以抵抗大部分的代码混淆; 在扩展性方面, SriDroid 分析一个应用的平均时间为 4.6 s.

本文的主要贡献如下:

(1) 首先, 本文提出了基于抽象图分析的安卓恶意软件检测方法, 采用图卷积网络作为分类器;

(2) 其次, 本文设计了 SriDroid, 一个包含程序语义、抗混淆和可扩展的安卓恶意软件检测系统;

(3) 最后, 本文测试了 SriDroid 的检测效果, 通过实验分析 10 757 个正常应用和 9 489 个恶意应用, 发

现 SriDroid 可以获得 99.17%的恶意软件检测精确度, 而其他对比系统: PerDroid、Drebin、MaMaDroid 和 IntDroid 的检测精度分别为 94.01%、95.81%、96.13%和 96.72%.

本文第 1 节介绍安卓恶意软件检测的相关方法和研究现状. 第 2 节具体介绍恶意软件检测器 SriDroid. 第 3 节从有效性、鲁棒性和扩展性这 3 个方面介绍 SriDroid 的实验结果. 第 4 节总结全文.

1 安卓恶意软件相关工作

根据提取特征的类型, 现有的安卓恶意软件检测方法主要分为两大类: 基于语法特征的方法和基于语义特征的方法.

1.1 基于语法特征的方法

为了实现快速的恶意软件分析, 一些研究者只考虑程序的语法特征, 这些特征主要包括应用程序的权限、API、组件和元数据等.

(1) 基于权限、API 和组件的安卓恶意软件检测

权限是一种安全机制, 针对应用自身而言, 它可以限制程序内部某些具有限制性特性的功能使用; 针对不同的应用, 它可以限制不同应用之间的组件访问. 例如: 一个应用软件如果要使用网络, 则需要申请网络相关权限 INTERNET. 基于权限分析的恶意软件检测方法会从安卓应用的配置文件 AndroidManifest 中提取权限信息, 并将它们作为特征来训练分类器.

Wang 等人[12]应用 3 种不同的特征排序算法(互信息、相关系数和 T 检验)将安卓应用的权限进行了风险评估; 同时, 采用前向选择和主成分分析来识别更有风险的权限子集, 并利用这些权限子集构建特征向量以训练机器学习模型. 此外, 为了获得更高效的实验结果, Li 等人[13]分别针对提取的权限进行模式挖掘, 并采用这些权限组成的模式来检测恶意软件.

Zhou 等人[4]提出了一种可扩展的恶意软件检测方法 DroidRanger. 对于已知的恶意软件, DroidRanger 开发了一种基于权限信息的行为足迹方案进行检测. 对于未知的恶意软件, DroidRanger 结合远程代码的加载情况和软件的动态执行信息, 采用启发式方法定义恶意软件的可疑行为, 并最后使用这些预定义的行为规则来识别未知恶意家族的恶意行为.

为了使应用程序与开发人员更方便地访问安卓系统的特定例程, 而无需访问源码或理解系统内部的实现细节, 安卓定义了一些函数并提供外部接口, 这些函数就是安卓 API. API 相较于权限而言是一种粒度更细的特征. 因此, 为了实现更精确的恶意软件检测, 一些研究者从应用中提取不同类型的 API, 并以此作为特征来训练机器学习模型. 例如, Asfer 等人[14]通过静态分析提取了 5 种不同类型的 API: 应用相关 API、安卓框架 API、DVM (Dalvik virtual machine)相关 API、Linux 系统调用 API 和工具 API, 并利用这些 API 训练模型, 并以此检测恶意软件.

Zhao 等人[15]通过动态分析记录应用执行的系统调用 API, 并以此作为特征训练机器学习模型. 此外, 为了进一步提升恶意软件检测的效果, 一些研究者[16]将权限和 API 进行组合. Kouliaridis 等人[17]通过提取权限和 Intent 信息作为安卓恶意软件的有效特征, 并将 PCA 和 t-SNE 集成于基本模型, 发现两种转换都能显著提高每个基本模型以及所构建的集成的性能, 最终在 AndroZoo 数据集上分别获得了 95.1%和 91.7%的 AUC 和准确率.

Cai 等人[18]提出了一种基于特征权重的 Android 恶意软件检测方法 JOWMDroid, 通过权重映射和分类器参数的联合优化, 从 Android 应用中提取 8 个特征类别, 然后利用 IG 算法筛选出最重要的特征. JOWMDroid 采用 3 个基础模型计算每个特征的权重, 然后设计了 5 个权重映射模型, 将初始权重映射到最终权重, 并最终利用差分进化算法对权重映射模型和基础模型的参数进行联合优化.

除了权限和 API 之外, 缪小川等人[19]还发现, 应用配置文件 AndroidManifest 中的 Intent 和组件信息以及应用代码中的一些字符串也可以作为检测恶意软件的有效特征. 例如: Arp 等人[20]和 Daoudi 等人[21]不仅从配置文件中提取权限、组件、Intent 信息(例如重启手机后触发恶意行为的 BOOT_COMPLETED)和硬件信息(例

如 GPS 和网络模块), 还从应用代码中提取关键 API、使用的权限、可疑 API 和网络地址信息(例如 IP 地址和 URL), 并将这 8 种不同类型的特征集合在一起来训练分类器.

刘玮[22]通过对比安卓应用权限, 发现安卓应用 Intent 信息是识别安卓恶意应用程序的一个有效特征. 超凡等人[23]融合权限、API 调用、Intent Filter 的 action 属性以及数据流等多种风险因素, 并采用层次聚类分析对因素子集进行权重分配, 以此评估安卓应用的真实安全风险.

(2) 基于元数据的安卓恶意软件检测

除了权限、API 和组件特征之外, 一些研究者还会提取和应用程序代码无关的其他信息(下载量、功能描述、大小、版本号等)来检测恶意软件.

Teufl 等人[24]将爬取应用的多种元数据信息作为特征, 包括最后一次的修改时间、应用价格、详细的描述信息、所属类别、应用下载量、用户评级、安装包大小、截图数量、包名信息、版本号、开发者 ID 以及联系方式等.

Corla 等人[7]采用聚类分析对应用的描述信息进行主题分类, 一共划分为 29 个主题, 包括游戏、TV、音乐、语言等, 然后针对不同的主题进行不同的分析. 例如: 如果一个社交应用程序调用了 API 函数 *URL*.*openConnection*(·)来连接网络, 这属于一个正常现象; 而一个相机应用程序如果调用了这个 API, 则会被判定为可疑应用. 该方法将应用的描述信息和本身行为相结合进行了分析, 以此在不同主题下挖掘可疑应用.

Fan 等人[25]提出了基于安全技术博客分析的恶意软件检测方法 CTDroid, 从大量与恶意软件相关的技术博客中抽取出与恶意行为相关的敏感特征, 并以此来检测恶意软件.

Pandita 等人[26]提出的 WHYPER 是第一个采用自然语言处理技术的安卓应用危险等级评估工具. WHYPER 会分析应用程序中的 3 种权限(READ CONTACTS、READ CALENDAR 和 RECORD AUDIO)是否被申请, 并借助自然语言处理的技术对应用的描述进行分析. 通过判断描述中是否包含读取通信录、读取日程表和录音功能的相关语句, WHYPER 可以发现是否存在行为不一致的现象.

1.2 基于语义特征的方法

为了实现更为高效的恶意软件分析, 研究人员会执行程序分析以提取相关的语义特征, 这类特征一部分以图的方式加以展示, 一部分以数据流的方式进行分析.

(1) 基于图分析的安卓恶意软件检测

为了在保留程序语义的基础上提高特征的鲁棒性, 越来越多的研究者将程序语义表示为图, 并采用图分析来检测恶意软件. 常用的图模型根据节点属性的不同可以分为不同的类型. 例如: 节点为函数的函数调用图、节点为基本块的控制流程图和节点为语句的数据依赖图等. 根据节点类型的不同, 这些图大致上可以分为 3 个粒度: 粗、中等和细. 粒度较粗的图主要以视图、包和类作为分析对象, 粒度中等的图主要以 API 和函数作为分析对象, 粒度更细的图主要以基本块和语句作为分析对象.

Chen 等人[27]提出的 MassVet 分析的是应用程序的视图, 它将所有视图组合成一个有向加权的 UI 图. 一个视图到另一个视图的过渡关系可以表示为视图之间的导航关系. UI 图中的每个节点代表应用的一个视图, 节点之间边的权重由活动小部件(具有适当事件响应操作的控件)的数量决定, 数量越多, 权重越大.

Feng 等人[5]提出的 Apposcopy 通过对软件进行语义签名分析来判断其是否为恶意软件, 用于识别会窃取用户隐私数据的恶意软件. 为了保留应用的语义信息, Apposcopy 采用静态分析将应用的组件类型信息、控制流信息和数据流信息表示为一种新型的调用图形式, 并开发了一种用于表征恶意软件家族语义行为的高级语言, 最后, 以此来检测已知恶意软件家族中的恶意软件.

Feng 等人[6]在 2017 年将 Apposcopy 进行了改进, 并提出了 Astroid. Astroid 可以从恶意软件家族的极少数样本中自动学习到该家族的语义签名, 其关键思想是: 将恶意软件的语义表示为一个组件间调用图, 并在恶意软件家族的所有已知实例之间寻找最大的可疑公共子图, 最后采用近似子图匹配算法来判断应用程序是否与某些恶意软件家族的语义签名相符合: 如果符合, 则认为是属于该家族的恶意软件.

Wu 等人[28]提出的 HomDroid 将应用的程序语义抽象成对应的函数调用图, 视其为社交网络, 并分析网络

中正常子图和敏感子图之间的同质性, 以挖掘恶意子图, 基于聚合的恶意子图, 采用网络三元组分析技术, 提取相关的语义特征并以此训练机器学习模型, 实现对隐蔽型恶意软件的检测.

He 等人[29]提出的 MsDroid 首先通过静态分析获取应用的函数调用图, 然后针对敏感 API 分析相关的代码片段, 并以此构建行为子图集. 由于恶意代码一般是在围绕敏感 API 的几个片段中得以实现, 因此, 子图集的片段表示方法能够对应用程序的敏感行为进行建模. 获得子图集之后, MsDroid 会训练一个图神经网络来训练分类器, 并以此识别恶意行为.

Zhang 等人[30]提出的 DroidSIFT 通过分析敏感 API 之间的上下文语义来构建 API 依赖图, 上下文语义信息通过入口点发现、调用图分析和前向后向数据流分析的方式进行提取. 构建完 API 依赖图之后, DroidSIFT 采用图相似性匹配算法来进行恶意软件检测.

Hou 等人[31]提出的 HinDroid 从反编译得到的 smali 代码中提取 API 调用, 并将这些 API 调用转换成一组全局整数 ID 来表示相应 API 调用的静态执行顺序. 获取整数 ID 后, HinDroid 会进一步分析它们之间的关系, 例如是否属于相同的 smali 代码块、是否具有相同的包名称和是否使用了相同的调用方法等. 通过集合整数 ID 间的关系, HinDroid 构建了一个异构信息网络, 用于分析网络中的元路径, 从而检测恶意软件.

Mariconti 等人[9]提出的 MaMaDroid 首先通过静态分析获取应用程序的函数调用图, 然后提取图中所有函数序列, 并将函数抽象到对应的包或家族. MaMaDroid 通过抽象后的序列建立一个马尔可夫链模型来表示函数之间的转换概率, 并进一步将转换概率作为特征输入机器学习模型以训练分类器. 类似地, Wu 等人[32]提出的 MalScan 从提取到的函数调用图上分析敏感 API 的不同中心性, 并以此训练机器学习模型来检测恶意软件.

Yumlembam 等人[33]提出的 VGAE-MalGAN 首先提取应用的 API 调用图并分析图中 API 的中心性, 然后采用图神经网络对这些 API 进行图嵌入以获取对应的特征向量, 最后, 结合 Permission 和 Intent 训练用于安卓恶意软件分类的多种机器学习模型.

Crussell 等人[34]提出的 DNADroid 通过分析程序代码之间的数据流关系来构建数据流图, 从而进行细粒度的恶意软件分析. 其构建的数据流图由程序的语句代码组成, 语句之间的边表示两者之间的数据依赖关系.

(2) 基于数据流分析的安卓恶意软件检测

恶意软件的一大典型恶意行为是获取用户的隐私信息, 导致用户隐私泄露. 为了检测这一类行为, 研究者采用数据流分析技术来记录敏感数据在应用程序中的流转: 如果发生异常流转, 则判定为恶意程序. 主流的数据流分析技术分为静态和动态两种.

Arzt 等人[35]提出的 FlowDroid 是第一个采用静态数据流分析技术来检测安卓应用隐私泄露的工具. FlowDroid 的分析对象主要包括应用的字节码以及配置文件, 具有上下文敏感、流敏感、域敏感和对象敏感等特性. 因此, FlowDroid 可以精确地构建完整的安卓生命周期, 包括在应用程序中正确处理回调和用户定义的 UI 窗口小部件.

王蕾等人[36]提出的 FlowDroidSP 利用稀疏优化方法对静态污点分析中无关联的传播进行消除, 以达到时间和空间的开销优化. 实验结果表明: FlowDroidSP 与原 FlowDroid 相比, 在非剪枝模式下, 具有平均 4.8 倍的时间加速和 61.5%的内存降低; 在剪枝模式下, 具有平均 18.1 倍的时间加速和 76.1%的内存降低. 此外, 为了分析安卓生态系统中第三方 SDK 的安全漏洞, 马凯等人[37]集合了不同的工具(如 FlowDroid), 一共分析了市场上流行的 129 个第三方 SDK, 通过实验, 发现超过 60%的 SDK 中含有各种安全漏洞.

Li 等人[38]提出的 IccTA 可以在两个组件之间进行数据流分析, 并充分建模生命周期和回调方法以检测基于组件间通信的隐私泄漏. IccTA 利用 IC3[39]等分析 Intent 相关的字符串并找到 Intent 的目标之后, 会利用 FlowDroid 进行污点追踪以检测隐私泄露状况.

Nan 等人[40]提出的 UIPicker 和 Huang 等人[41]提出的 SUPOR 均通过解析 UI 界面的相关文件来识别用户输入是否为敏感数据, 并会对判定为敏感数据的数据流进行追踪, 只要发现这些数据以明文形式或不安全的 SSL 形式保存或者发送至网络, 就认为发生了隐私泄露.

Enck 等人[42]提出的 TaintDroid 采用动态数据流分析技术实时监控应用的敏感数据. 当敏感数据被发送至

网络时, TaintDroid 会标记数据并记录传输的应用程序以及数据被传输的目的地, 从而为用户提供实时监控与反馈. 这样, 用户可以更深入地了解应用程序正在做什么, 从而识别出行为异常的应用程序. 为了方便用户自定义敏感数据, Hornyack 等人[43]改进了 TaintDroid, 并提出了 AppFence, AppFence 实现了对用户自定义隐私数据的实时拦截, 以达到隐私保护的目的.

总体而言, 基于语法特征的恶意软件检测方法分析速度快, 但是由于未考虑程序的语义信息, 并且大部分都是以字符串的形式存在, 因此容易被攻击者通过混淆技术攻击. 基于语义特征的恶意软件检测方法中, 基于图分析的检测方法效果好、鲁棒性高, 可以检测更多的恶意软件, 但是由于程序分析和图分析的开销都大, 扩展性并不理想; 基于数据流分析的恶意软件检测方法在针对特定恶意行为时效果最好, 但是由于恶意软件的恶意行为多样化, 会出现误报和漏报现象. SriDroid 属于语义特征中基于图分析的检测方法, 但不同于传统检测技术之处在于: SriDroid 通过对节点进行抽象化以减少图的大小, 并基于 Triplet Loss 对比损失训练一个图卷积网络分类器, 从而具有了更高的分析准确性、更强的鲁棒性和更好的扩展性.

2 系统介绍

这一节介绍本文提出的安卓恶意软件检测方法 SriDroid.

2.1 系统总览

如算法 1、算法 2 和图 1 所示, SriDroid 分为两个阶段: 训练阶段和预测阶段. 每个阶段包含 4 个步骤: 静态分析、图抽象化、特征向量提取和模型的训练/预测.

- 静态分析: 这一阶段的目的是保留安卓应用的程序语义. 主要步骤是, 采用静态分析技术提取应用的函数调用图. 输入是一个安卓应用, 输出是应用对应的函数调用图;
- 图抽象化: 这一阶段的目的是将图变得更具有鲁棒性, 以抵抗代码混淆. 主要步骤是: 将调用图中的节点抽象成对应的类, 并构建成一个类图. 输入是一个函数调用图, 输出是对应的抽象图(类图);
- 特征向量初始化: 这一阶段的目的是获取图中节点的初始特征向量. 主要步骤是: 采用 One-Hot 编码方式, 按照节点的不同类型为节点赋予初始特征向量. 输入是一个抽象图, 输出是节点的初始特征向量;
- 模型的训练/预测: 这一阶段的目的在训练阶段是训练一个恶意软件分类器, 在预测阶段是应用训练好的模型检测应用是否为恶意软件. 主要步骤是: 基于抽象图和节点的初始特征向量, 基于 Triplet Loss 对比损失训练一个图卷积网络分类器, 并以训练好的分类器检测恶意软件. 输入是抽象图和节点初始特征向量, 在训练阶段的输出是训练好的图卷积网络模型, 在预测阶段的输出是 0 或 1 的预测标签, 0 表示是正常软件, 1 表示是恶意软件.

算法 1. 训练阶段模型训练算法.

输入: A: 安卓应用程序, L: 程序对应的标签;

输出: M_{GCN}: 训练好的 GCN 模型.

1. G=*GetFunctionCallGraph*(A) //获取应用程序的函数调用图 G
2. G_{abs}=*GetAbstractGraphh*(G) //按照 class, package 或者 family 层级进行图抽象, 获得抽象图 G_{abs}
3. V=*GetOneHotVector*(G_{abs}) //对所有图进行遍历, 获取节点的 One-Hot 向量编码 V
4. A=*GetAdjustDict*(G_{abs}) //将图结构表示为邻接矩阵 A
5. **for each** *Batch* in *GCN*(A,V,L):
6. $Batch_{pos}$=*GetPositiveSample*(*Batch*) //从训练集获取与输入样本相同标签的数据作为正样本
7. $Batch_{neg}$=*GetNegtiveSample*(*Batch*) //从训练集获取与输入样本不同标签的数据作为负样本
8. *Loss*=*GetTripleLoss*(*Batch*,$Batch_{pos}$,$Batch_{neg}$) //带入计算 Triple Loss, 并反向传播优化模型
9. **end for**
10. 结束训练返回模型 M_{GCN}

算法 2. 测试阶段安卓恶意软件检测算法.

输入: A: 待检测的安卓应用程序, M_{GCN}: 训练好的 GCN 模型;

输出: O: 对待检测应用的预测标签.

1. G=*GetFunctionCallGraph*(A) //获取应用程序的函数调用图 G
2. G_{abs}=*GetAbstractGraphh*(G) //按照 class, package 或者 family 层级进行图抽象, 获得抽象图 G_{abs}
3. V=*GetOneHotVector*(G_{abs}) //对所有图进行遍历, 获取节点的 One-Hot 向量编码 V
4. A=*GetAdjustDict*(G_{abs}) //将图结构表示为邻接矩阵 A
5. $O=M_{GCN}(A,V)$
6. 结束测试返回预测标签 O

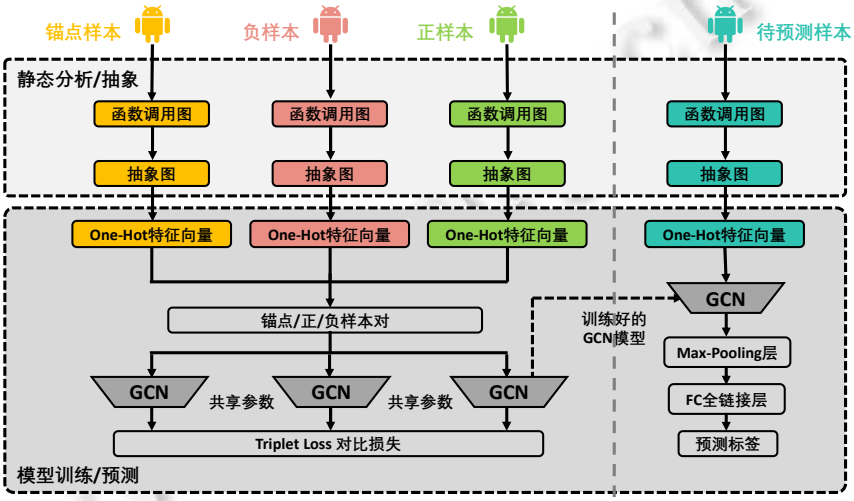

图 1 系统总览

2.2 静态分析和图抽象化

静态分析的目的是提取安卓应用的函数调用图. 函数调用图中, 每个节点都是一个函数, 可以是 API 调用或用户定义的函数. SriDroid 的静态分析过程是基于一个安卓逆向工具(Androguard[44])实现的, Androguard 在不考虑程序上下文和流敏感的情况下, 可以提取一个更加简洁的函数调用图.

在提取函数调用图之后, SriDroid 会对调用图进行抽象化以构建鲁棒性更强的抽象图. 具体而言, SriDroid 会将调用图中的节点抽象成对应的 3 个不同级别的抽象标签 class、package 或者 family. 如图 2 所示: 一个 API 由冒号左边的 class 和冒号右边的 method 组成, 不同的 method 代表着不同的行为. class 又分为 3 部分: 第 1 部分表示 family 类别, 前两部分一起表示 package 类别, 3 个部分合在一起表示一个 class.

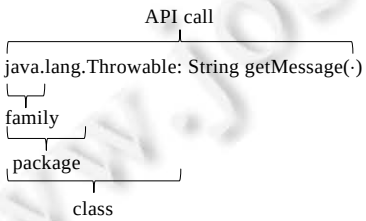

图 2 API 调用

为了获得抽象标签的集合, 本文以安卓官方给出的 class 列表[45]作为分析对象, 最终获得 5 971 个 class、387 个 package 和 9 个 family, 并将这些抽象标签作为白名单. 只有当函数调用图中函数的抽象标签出现在白名单中时, 程序才会将该函数抽象成对应的 class、package 或者 family, 否则会抽象成“other”. 在该过程中, 一

方面, 将 API 进行抽象化会带来一些误差, 导致不同功能的函数会被抽象至同一个标签; 另一方面, 恶意软件在执行恶意行为时, 一般会调用敏感 API. 因此, 为了减小针对恶意行为表征的误差, SriDroid 没有对敏感 API 进行抽象处理. 从实验结果上来看, SriDroid 在鲁棒性上得到了增强; 同时, 在对恶意行为的表征上没有出现较大的误差, 进一步验证了方法的有效性.

整体而言, 函数的抽象化主要分为两个步骤.

(1) 首先, 判断函数是否为敏感 API: 若不是敏感 API, 则进行下一步; 否则, 不进行处理;

(2) 然后, 根据所需抽象层级, 通过正则处理, 将函数的 class、package 或者 family 提取出来. 抽象出的标签如果在白名单中, 则将其抽象为对应的标签; 否则, 抽象为“other”.

本文一共选择了 3 个不同的敏感 API 子集[46]: 第 1 个子集是与恶意软件相关性最高的 260 个敏感 API, 第 2 个子集是 112 个与危险权限相关的敏感 API, 第 3 个子集是 70 个与敏感操作相关的敏感 API. 通过计算这 3 个 API 子集的并集, 可以获得 426 个敏感 API. 如图 3 所示: 以 class 抽象层级为例, 由于节点 4 和节点 5 是敏感 API, 所以不对其进行抽象处理; 节点 1 和节点 3 的 class 不在白名单之中, 所以将其转换成 other; 节点 2 和节点 6 的 class 相同, 均为 java.lang.Throwable, 由于该 class 在白名单之中, 所以将它们抽象成 java.lang.Throwable.

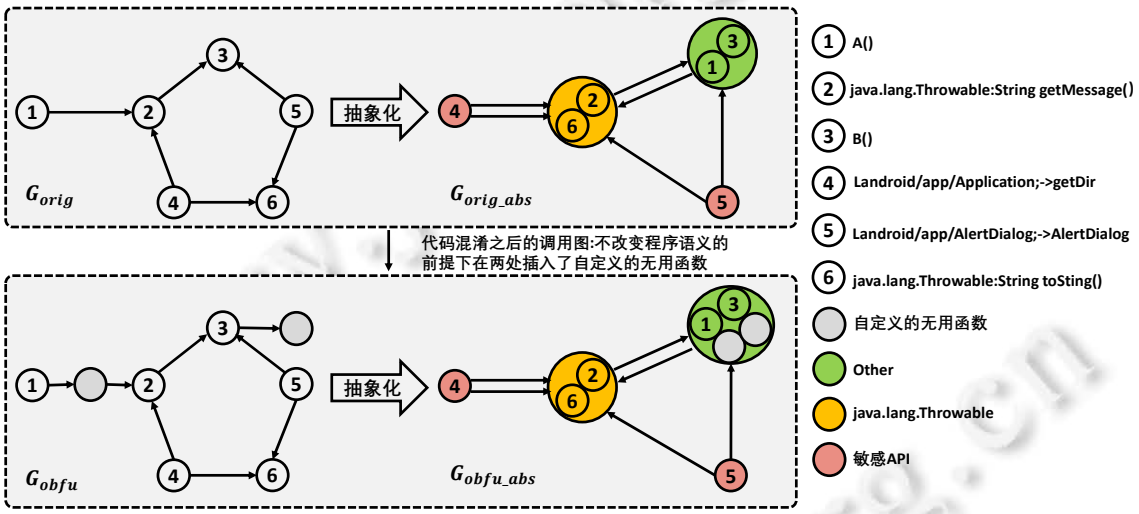

图 3 图抽象化和特征提取

此外, 为了阐明抽象之后的图具有更高的鲁棒性, 图 3 中还展示了一个简单的例子. 假设程序被混淆之后, 混淆工具主要的操作是在代码中插入了无用代码. 如图 3 所示, 主要在函数 A(·)和函数 B(·)之后分别插入了两个没有意义的函数. 在计算比较原始调用图和混淆调用图以及原始抽象图和混淆抽象图之间的图编辑距离之后, 发现抽象图的距离更小, 具体结果见表 1. 对于原始调用图, 混淆前、后图之间的编辑距离为 18, 而抽象化之后, 图编辑距离减少至 2. 这个结果也说明, 抽象图在对抗混淆上具有更强的鲁棒性.

表 1 混淆前、后图之间的编辑距离

调用图	原始图 G_{orig}	混淆图 G_{obfu}	原始抽象图 G_{orig_abs}	混淆抽象图 G_{obfu_abs}
节点数	6	8	4	4
边数	6	8	6	8
图编辑距离	G_{orig} 和 G_{obfu} 的编辑距离为 18		G_{orig_abs} 和 G_{obfu_abs} 的编辑距离为 2	

2.3 特征向量初始化

在获得函数调用图的抽象图之后, SriDroid 没有采用传统复杂的图分析方式对其进行处理, 而是采用

One-Hot 编码方式为图中所有节点赋予一个初始化的特征向量，这样可以规避高开销的图分析，使得后续的恶意软件检测更为高效.

具体而言，针对抽象图的集合，首先对所有图中的节点进行一次遍历，并将它们按照预先设定的编号加以保存，从而构建一个节点字典. 字典的长度即所有节点的种类数，表示为 *Num_node*. 在初始化节点特征向量的过程中，对于每个节点，先初始化一个长度为 *Num_node* 的全 0 特征向量，然后根据其在节点字典中的索引，在相应的特征向量位上赋 1，从而生成了维度均为 *Num_node* 的节点初始特征向量.

2.4 模型训练和预测

构建特征向量之后，SriDroid 会首先基于 Triplet Loss 损失训练一个分类器，然后利用该分类器来检测恶意软件. 因为图卷积网络可以聚合邻居特征并自动学习高效的特征表示，所以本文选择图卷积网络作为 SriDroid 的分类器，并在损失函数上选择了 Triplet loss. Triplet loss 是深度学习的一种损失函数，通过对比样本的方式，可以提升模型检测的精确度和鲁棒性. 具体如图 1 所示：针对一个输入样本 *a* 作为锚点样本，首先获取一个正样本 *p* 和一个负样本 *n*，然后通过损失函数优化使得样本到正样本的距离小于到负样本的距离. 令 margin 是一个大于 0 的常数，对比损失最终的目标是：使锚点样本 *a* 和正样本 *p* 之间的距离尽可能地靠近，与负样本 *n* 之间的距离尽可能地拉远. 原理公式如下:

$$Loss_{triplet}=\max(distance(a,p)-distance(a,n)+\text{margin},0).$$

图卷积网络[47]由三大层次构成：第 1 层次为输入层；第 2 层次为隐藏层，隐藏层内部由多个卷积层构成；第 3 层次为输出层，输出层由池化层和一个可以生成全局信息的全连接多层感知分类器构成. 隐藏层中，卷积层的主要作用是抽取数据的主要特征，减小数据的大小，以达到降维的目的. 池化层则可以按照设定的规则实现最大池化或者平均池化，从而起到降低数据维度的作用；同时，也可以使网络模型获得对抗小位移和抗形变的能力. 卷积与池化方法大大降低了模型的复杂性，减少了模型的各项参数，同时保留了原始数据的关键特征.

SriDroid 中，图卷积网络的结构图如图 4 所示，其输入层接受的数据大小是(*batch_size*,*node_num*, *feature_dim*). *batch_size* 为一个训练批次中图的个数，*node_num* 为图中节点的个数，*feature_dim* 为节点的初始特征向量维度.

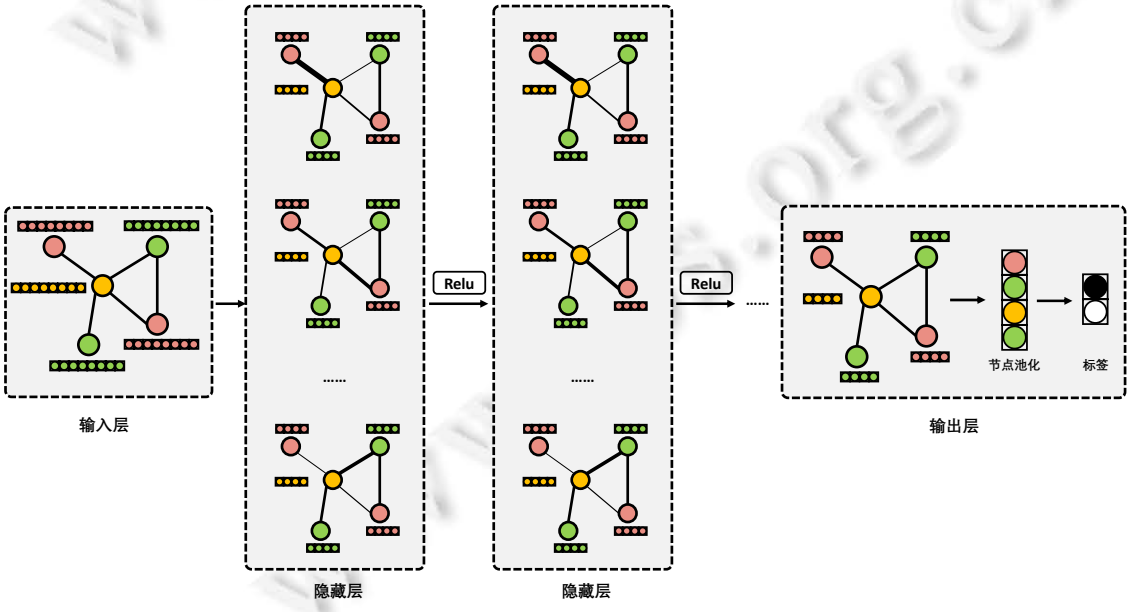

图 4 SriDroid 中的图卷积网络

具体实验中, 由于采用 One-Hot 编码, *feature_dim* 等于抽象图数据库中所有节点总的种类数. 经过 3 层卷积操作后, 输出的数据大小为(*batch_size*,*node_num*,*hidden_dim*), *hidden_dim* 指的是隐藏层单元数. 如图 1 所示: 在模型训练过程中, 首先将输入样本确定为锚点样本, 从训练集中随机获取与输入样本相同标签的数据作为正样本, 不同标签的数据作为负样本, 并将锚点样本、正样本和负样本的该输出层数据组合成样本对. 样本对的数据带入 Triplet Loss 损失函数计算损失后, 将会通过反向传播调整优化模型. 在得到训练好的模型后, 在预测阶段, 将该输出层的数据送入池化层. 池化层采用最大池化策略保留节点向量每个维度的最大值, 从而得到大小为(*batch_size*,*hidden_dim*)的数据. 最后, 全连接层会将 *hidden_dim* 的数据映射至分类的种类数, 从而获得模型预测的标签以用于判断应用是正常还是恶意.

3 实验分析

本节从以下 4 个方面对 SriDroid 进行实验测试, 包括有效性、鲁棒性和扩展性.

RQ1: SriDroid 在检测恶意软件方面效果如何?

RQ2: SriDroid 是否可以对抗代码混淆?

RQ3: SriDroid 的运行开销是多少?

3.1 实验设置

实验的硬件环境采用 AMD EPYC 7742 64-Core 处理器, 256 GB 内存, NVIDIA GeForce RTX 3090 GPU, 软件环境采用 Ubuntu 20.04.4 LTS, PyTorch-1.12.1 工具库.

为了测试 SriDroid 的效果, 实验从 Virusshare[11]上随机下载了 9 489 个恶意应用, 同时从 AndroZoo[10]上随机下载了 10 757 个正常应用, 从而构成了实验的数据集. 所有从 AndroZoo 下载的安卓应用会被送入 VirusTotal[48]中, 经过扫描器进行扫描并返回扫描结果. 只有当所有扫描结果均显示应用为正常时, 应用才被认为是正常软件. 在训练和测试过程中, 实验采用十折交叉验证法来进行实验和测试, 通过多次重复利用数据集以得到准确的评估结果. 具体而言, 将数据集分为 10 份, 其中 9 份用来训练模型, 剩下 1 份用作测试进行模型评估, 重复 10 次直至每一份数据都经过测试评估. 最后, 实验对 10 次测试结果取平均值, 从而得到更为准确的结果. 由于在十折交叉验证法的过程中会随机生成多组训练集和测试集进行训练, 实验可以充分利用样本数据集以得到更为准确和稳定的评估结果. 此外, 实验采用了研究者广泛使用的一些指标来衡量 SriDroid 的检测效果, 分别为精确度(*P*, precision)、召回率(*R*, recall)、*F*-分数(*F*1)和准确率(*A*, accuracy). 为了实验的完整性, 本文从基于语法特征和基于语义特征的相关工作中分别选择了对比工具. 对于基于语法特征的工具, 本文主要选择了两个被使用得最为广泛的恶意软件检测系统; 对于基于语义特征的工具, 由于 SriDroid 是基于函数调用图的恶意软件检测系统, 因此在对比分析工具中也选择了两个基于函数调用图的恶意软件检测方法. 具体而言, 一共选择了 4 个安卓恶意软件检测系统作为对比系统, 具体为:

- PerDroid[12]: 一个基于权限分析的安卓恶意软件检测系统, 它通过分析安卓应用要求的权限来构建特征向量, 并以此来检测恶意软件;
- Drebin[20]: 一个基于权限、API、Intent 等的安卓恶意软件检测系统, 它执行广泛的静态分析, 以便从应用程序中提取尽可能多的特征, 并将它们嵌入到联合向量空间中, 以检测恶意软件;
- MaMaDroid[9]: 一个基于图分析的安卓恶意软件检测系统, 它利用从函数调用图中获得的抽象函数调用序列来构建行为模型, 并以此进行恶意软件检测;
- IntDroid[49]: 一个基于亲密度分析的安卓恶意软件检测系统, 它通过分析应用函数调用图中敏感 API 和中心节点之间的亲密度来提取高效的语义特征, 并以此来检测恶意软件.

在参数选择方面, SriDroid 在十折交叉验证的数据中选用前 3 组的数据进行超参数调试, 选取最佳结果的参数冻结后进行后续完整的实验, 具体的参数选择见表 2. 图卷积网络采用 ReLU 作为激活函数, Triplet Loss 作为损失函数, Adma 作为优化算法, 在 0.005 的学习率之下进行训练. 训练得到分类器之后, 在测试阶段, 实验使用训练好的分类器来检测新的样本.

表 2 图卷积网络的训练参数

参数	值
输入节点向量	One-Hot 编码向量
隐藏层数	3
隐藏层单元数	64
激活函数	ReLU
池化策略	MaxPooling
Dropout rate	0.5
Batch size	32
训练轮次	50
学习率	0.005
优化算法	Adma
损失函数	Triplet Loss

3.2 RQ1: 有效性

为了对比分析不同工具的检测能力, SriDroid 与对比工具在同样的数据集上进行了实验分析. SriDroid 在实验过程中, 分别从 class、package 和 family 这 3 个抽象层级进行了实验, 结果分别记为 SriDroid-class、SriDroid-package 和 SriDroid-family. 在 SriDroid 每一个层级的实验中, 都将 Triplet Loss 替换为交叉熵损失, 进行了对比实验, 记为 SriDroid(CE), 具体的对比实验结果见表 3.

表 3 不同工具的对比实验结果

对比系统	*P*	*R*	*F*1	*A*
PerDroid	94.12	93.59	93.85	94.01
Drebin	95.86	95.66	95.76	95.81
MaMaDroid	96.11	95.78	95.95	96.13
IntDroid	97.12	96.36	96.74	96.72
SriDroid (CE)-family	96.84	96.82	96.83	96.83
SriDroid-family	97.83	97.85	97.84	97.86
SriDroid (CE)-package	97.32	97.54	97.40	97.42
SriDroid-package	98.53	98.64	98.58	98.59
SriDroid (CE)-class	98.38	98.41	98.39	98.39
SriDroid-class	**99.21**	**99.13**	**99.16**	**99.17**

PerDroid 是一个基于危险权限分析的安卓恶意软件检测方法, 它应用 3 种不同的特征排序算法(互信息、相关系数和 T-test)将现有安卓应用的权限进行了风险排序, 并将排序靠前的权限选为特征, 以此来检测恶意软件. 在本节中, PerDroid 相关文献中公布的 88 个危险权限被选为特征来构建对比分析分类器.

表 3 描述了 PerDroid 和 SriDroid 的对比实验结果, 从结果可以得知, SriDroid 比 PerDroid 在恶意软件检测上的表现更优. 例如: PerDroid 的 *F*1 和 *Accuracy* 只有 93.85%和 94.01%, 而 SriDroid 可以达到 99.16%和 99.17%. 出现这个现象是因为 PerDroid 未考虑安卓应用的语义信息而 SriDroid 保留了程序语义. Drebin 是一个基于多特征分析的安卓恶意软件检测方法, 它通过静态分析从 Manifest 文件和反汇编代码中尽可能地提取足够多的特征, 并以此来检测恶意软件. 尽管 Drebin 在 *F*1 和 *Accuracy* 上的表现比 PerDroid 更优, 但在各个评价标准上都不如 SriDroid. 这是因为 Drebin 虽然提取了足够多的特征, 但其中包含了很多冗余和无效的特征; 而且 Drebin 没有考虑应用的图信息, 因此在程序语义信息的获取上是不够完整的.

MaMaDroid 是一个基于图分析的安卓恶意软件检测方法, 它从函数调用图中提取对应的抽象函数调用序列, 并以此作为特征进行分类. 这些抽象序列被用于构建马尔可夫链模型, 以表示函数之间的转移概率. 从表 3 可以得知, MaMaDroid 比 PerDroid 和 Drebin 表现得更好. 这主要是因为 MaMaDroid 将应用的语义信息转换成了对应的图模型, 考虑了程序的语义信息, 也因此有更好的检测效果. IntDroid 在提取应用的函数调用图之后, 会视其为社交网络并采用中心性分析挖掘出其中的中心节点, 最后计算敏感 API 和中心节点之间的亲密度, 并以此为依据检测恶意软件. IntDroid 与 MaMaDroid 类似, 都是基于函数调用图的恶意软件检测方法, 由于考虑了程序的语义信息, 其检测能力要优于 PerDroid 和 Drebin. 但是由于 MaMaDroid 和 IntDroid 在计算特征时都只考虑了部分图结构信息, 损失了一些程序的语义特征, 因此仍有改进空间. 而 SriDroid 采用的图卷

积网络通过聚合邻居节点的特征, 可以更全面地考虑图的结构信息和程序的语义信息, 并自动学习高效的特征表示, 从而更进一步地提升了检测效果.

通过对比 SriDroid 在不同抽象层级中的表现可以发现, 从 family 到 packag 再到 class, SriDroid 表现出的检测性能依次逐渐递增, 对应的抽象层级依次递减, 抽象图中包含的信息依次增多. 这一结果也与我们的预测相吻合, 训练数据的图中包含的信息越多, 训练出的分类模型也就会更精确. 通过对比 SriDroid 使用 Triplet Loss 和交叉熵损失下的表现可以发现, Triplet Loss 通过与正负样本对的组合和对比进行训练, 能够进一步提升模型检测的准确性. 在不同的抽象层级中, 使用 Triplet Loss 进行训练的模型均表现出了更好的检测效果. 因此, 在后续实验中, 我们均采用 class 层级的抽象和 Triple Loss 作为训练 loss 而进行实验测试.

总体而言, 在有效性问题上, 得益于对图结构的提取和图卷积网络的使用, SriDroid 能够更全面地考虑程序的语义信息, 在各个评价指标中均优于现有工作. 同时, 以 class 为图抽象层级和对 Triple Loss 的使用, 可以进一步提升检测效果.

3.3 RQ2: 鲁棒性

为了验证模型的鲁棒性, 即对抗代码混淆的能力, 本节使用一个代码混淆工具 Obfuscapk[50]自动生成混淆应用, 并用这些混淆应用来测试 SriDroid 和对比工具. 具体而言, 实验一共选择了 12 种代码混淆方法, 分别是 ClassRename、FieldRename、MethodRename、ConstStringEncryption、AssetEncryption、LibEncryption、ResStringEncryption、ArithmeticBranch、CallIndirection、Goto、Nop 和 Reorder. 表 4 一一给出了这些代码混淆方法的功能描述.

表 4 实验选择的代码混淆方法

混淆方法	功能描述
ClassRename	重命名包、类
FieldRename	重命名域
MethodRename	重命名方法
ConstStringEncryption	加密常量字符串
AssetEncryption	加密资产文件
LibEncryption	加密本地库
ResStringEncryption	加密资源中的字符串
ArithmeticBranch	插入由算术计算和分支指令组成的垃圾代码
CallIndirection	在不改变代码语义的情况下修改控制流图
Goto	通过添加两个新节点来修改控制流图
Nop	在每个方法实现中插入随机 nop 指令
Recorder	更改控制流图的基本块的顺序

具体而言, 实验首先从数据集中的 9 489 个恶意应用中随机选择 1 000 个样本作为分析对象, 然后用 Obfuscapk 混淆工具将这些应用进行混淆, 选择的代码混淆方法见表 4, 最后使用已训练好的 SriDroid 以及其他对比分析工具来检测 1 000 个混淆之后的应用. 测试结果见表 5, 从表中可以明显看出: PerDroid 的鲁棒性最差, 只有 72.6%的 *Recall*. 这是因为它完全没有考虑任何程序的语义信息, 导致其在对抗代码混淆上效果很差. 此外, MaMaDroid 和 IntDroid 的效果要更加理想一些. 这是因为它们是基于传统图分析的恶意软件检测方法, 将程序语义转换成调用图, 通过分析调用图来检测恶意软件, 使得它们具有更高的鲁棒性.

表 5 不同工具的抗混淆对比实验结果

对比系统	*R*
PerDroid	72.6
Drebin	86.1
MaMaDroid	87.2
IntDroid	90.1
SriDroid(CE)	95.7
SriDroid	**97.4**

此外, 从表 5 的结果还可以看到: SriDroid 在对抗代码混淆方面, 比 PerDroid、Drebin、MaMaDroid 和 IntDroid 都要更优, 这主要归功于 SriDroid 的抽象化和 Triplet Loss 的使用. 对比交叉熵损失的训练结果可以

发现, Triplet Loss 的应用提升了系统的抗混淆能力. 这是因为 Triple Loss 在训练过程中会更加关注样本之间的差距, 训练过程中, 在拉近相同类型样本之间距离的同时, 拉远了不同标签样本的距离, 并且引入更多的对比数据可以增强模型的泛化能力, 从而整体上提升系统的抗混淆能力.

为了验证抽象化在鲁棒性方面的表现, 实验从测试的数据集中随机选择了 100 个样本, 然后计算其原始调用图和混淆调用图以及原始抽象图和混淆抽象图之间的相似度. 相似度的计算步骤为:

(1) 采用 Node2Vec[51]技术将调用图的节点转换为对应的向量表示;

(2) 将所有节点的向量表示相加, 得到的向量作为调用图的向量表示.

没有采用图编辑距离的原因是: 因为调用图的节点数量过多, 计算图编辑距离时时间复杂度过大, 长达 10 个多小时的计算也无法得到一对样本的距离. 图 5 展示了这 100 个样本的原始调用图和抽象调用图混淆前后的相似度, 可以很明显地看到: 将函数调用图进行抽象化之后, 混淆前后的图相似度要更高, 其平均值为 0.827; 而对于原始调用图, 混淆前后的平均相似度为 0.722. 这个结果也验证了抽象图比原始图具有更高的抗混淆能力.

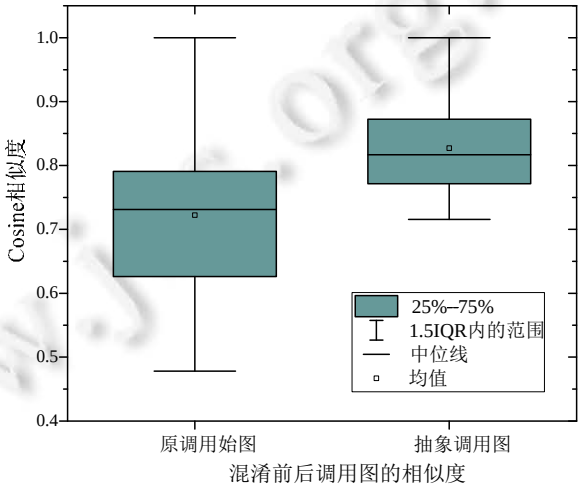

图 5 原始调用图和抽象调用图混淆前后的 Cosine 相似度

总体而言, 在鲁棒性问题上, 得益于图抽象化和 Triplet Loss 的使用, SriDroid 相较于对比工具具有更强的抗混淆能力. 同时, 原始图及抽象图混淆前后的相似度对比结果, 也进一步验证了抽象化操作在提升鲁棒性上所发挥的作用.

3.4 RQ3: 扩展性

为了检测 SriDroid 的扩展性, 本文将实验数据集中的所有应用都选为测试对象, 包括 10 757 个正常软件和 9 486 个恶意软件. 给定一个新的安卓应用, SriDroid 需要执行 4 个步骤来对其进行分析: (1) 静态分析提取函数调用图; (2) 图抽象化生成抽象图; (3) One-Hot 编码初始化节点特征向量; (4) 恶意软件分类.

SriDroid 的第 1 步是提取安卓应用的函数调用图. 图 6 展示了 SriDroid 在 20 243 个安卓应用(10 757 个正常软件和 9 486 个恶意软件)上提取调用图的运行开销. 对于 95%以上的安卓应用, SriDroid 可以在 10 s 内获得对应的函数调用图. 平均而言, SriDroid 需花费大约 3.02 s 完成一个安卓应用的静态分析. SriDroid 的第 2 步是将调用图进行抽象化. 如图 6 所示: SriDroid 在这一阶段花费的时间较少, 平均只需 1.32 s 即可完成一个函数调用图的抽象化阶段. SriDroid 的第 3 步是采用 One-Hot 编码初始化节点特征向量, 这一阶段的运行开销最小, 几乎可以忽略不计. SriDroid 的最后一步是利用图卷积网络训练一个分类器, 并对测试集进行结果分类. 如图 6 所示: SriDroid 完成分类所需时间也很快, 平均 0.25 s 即可完成一个抽象图的分类.

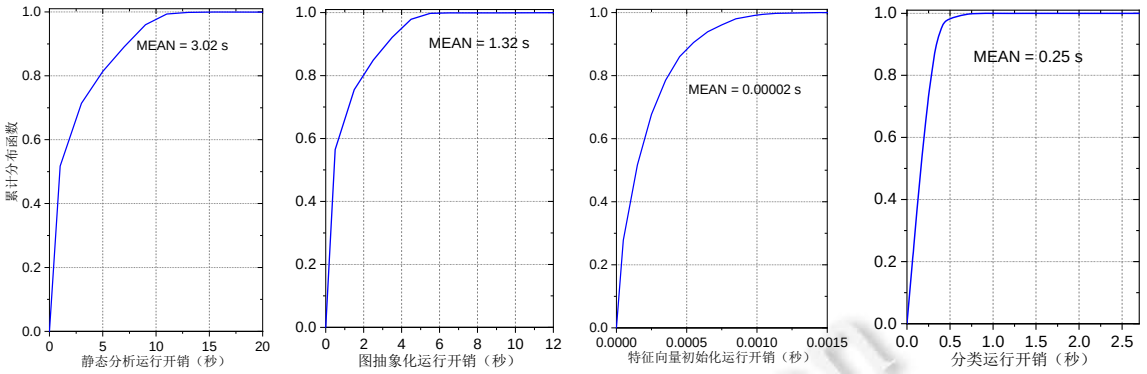

图 6 不同阶段的运行开销

此外, 实验针对 PerDroid、MaMaDroid、Drebin 和 IntDroid 也进行了同样的运行开销评估. 如图 7 所示: 对于 PerDroid, 由于它只需要从安卓应用的 Manifest 中提取相关权限, 其运行开销在所对比的工具中最低, 只需大约 6.27 s 就可以完成所有的分析; 对于 Drebin, 由于它不仅要从 Manifest 中而且还要从反汇编代码中提取特征, 因此其运行开销较高, 大约需要 33.9 s 才能完成一个安卓应用的分析, 但是 Drebin 分类时所需内存是 SriDroid 的百倍以上, 这是因为它的特征向量维度超过了 90 000; 对于 MaMaDroid, 它平均需要约 68.2 s 完成整个分析, 同时, 由于其特征维度很大(115 600 个特征), 故分类时要求的内存也很大; 对于 IntDroid, 完成一次分析的平均开销为 42.7 s, 相比之下, SriDroid 只需 4.6 s 即可完成所有分析.

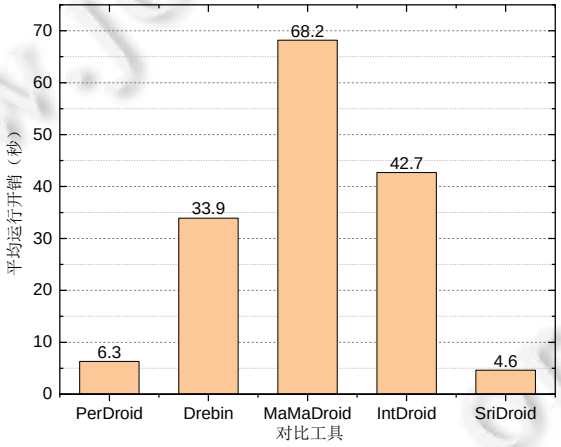

图 7 不同工具的运行开销

总体而言, 在扩展性问题上, 通过分析对比各个步骤的运行时间和所需内存可以发现, SriDroid 相较于现有工作具有最低的运行开销, 也因此具有最好的扩展性能.

4 总 结

在现有安卓恶意软件检测技术中, 未考虑语义信息的方法鲁棒性差、效果不理想; 考虑语义信息的方法效果理想, 但扩展性低. 为了解决这些问题, 本文提出了基于图卷积网络的抗混淆安卓恶意软件检测方法 SriDroid. 通过分析 10 757 个正常应用和 9 489 个恶意应用, SriDroid 可达到 99.17%的 *Accuracy*, 而 PerDroid、Drebin、MaMaDroid 和 IntDroid 的 *Accuracy* 分别为 94.01%、95.81%、96.13%和 96.72%. 此外, 在对抗代码混淆的实验测试中, SriDroid 可以达到 97.4%的 *Recall*, 而对比系统的 *Recall* 分别为 72.6%、86.1%、87.2%和 90.1%. 总体而言, SriDroid 相较于现有的安卓恶意软件检测方法具有最高的检测精度、最快的检测速度和最

好的抗混淆能力.

在未来的工作中, 我们将对敏感 API 的筛选进行更细粒度的划分, 以优化抽象图的过程; 同时, 提取更多的语义特征(例如数据流)来构建更丰富的图模型, 并以此设计支持恶意软件家族分类的新工具; 最后, 考虑采用新的可解释图卷积网络来开发具有解释功能的恶意软件检测工具.

References:

[1] CNNIC. 《中国互联网络发展状况统计报告》 (第 50 次). http://www.cnnic.net.cn/

[2] 360 互联网安全中心. 2021 年度中国手机安全状况报告. https://pop.shouji.360.cn/safe_report/Mobile-Security-Report-202206.pdf

[3] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification. In: Proc. of the 2009 ACM Conf. on Computer and Communications Security (CCS 2009). 2009. 235−245. [doi: 10.1145/1653662.1653691]

[4] Zhou YJ, Wang Z, Zhou W, Jiang X. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. In: Proc. of the 19th Annual Network and Distributed System Security Symp. (NDSS 2012). 2012. 1−13.

[5] Feng Y, Anand S, Dillig I, Saswat A. Apposcopy: Semantics-based detection of android malware through static analysis. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE 2014). 2014. 576−587. [doi: 10.1145/2635868.2635869]

[6] Feng Y, Bastani O, Martins R, Dillig I, Aiken A. Automated synthesis of semantic malware signatures using maximum satisfiability. In: Proc. of the 24th Annual Network and Distributed System Security Symp. (NDSS 2017). 2017. 1−15. [doi: 10.48550/arXiv.1608.06254]

[7] Gorla A, Tavecchia I, Gross F, Zeller A. Checking App behavior against App descriptions. In: Proc. of the 36th Int'l Conf. on Software Engineering (ICSE 2014). 2014. 1025−1035. [doi: 10.1145/2568225.2568276]

[8] Chen X, Li C, Wang D, Wen S, Zhang J, Nepal S, Xiang Y, Ren K. Android HIV: A study of repackaging malware for evading machine-learning detection. IEEE Trans. on Information Forensics and Security, 2019, 15: 987−1001.

[9] Mariconti E, Onwuzurike L, Andriotis P, Cristofaroy ED, Rossy G, Stringhiniy G. MaMaDroid: Detecting Android malware by building Markov chains of behavioral models. In: Proc. of the 24th Annual Network and Distributed System Security Symp. (NDSS 2017). 2017. 1−16.

[10] Allix K, Bissyandé TF, Klein J, Traon YL. Androzoo: Collecting millions of Android Apps for the research community. In: Proc. of the 2016 IEEE/ACM Working Conf. on Mining Software Repositories (MSR 2016). 2016. 468−471.

[11] VirusShare.com—Because sharing is caring. https://virusshare.com/

[12] Wang W, Wang X, Feng DW, *et al*. Exploring permission-induced risk in Android applications for malicious application detection. IEEE Trans. on Information Forensics and Security, 2014, 9(11): 1869−1882. [doi: 10.1109/TIFS.2014.2353996]

[13] Li J, Sun LC, Yan QB, Li ZQ, Srisa-an W, Ye H. Significant permission identification for machine-learning-based Android malware detection. IEEE Trans. on Industrial Informatics, 2018, 7(14): 3216−3225.

[14] Aafer Y, Du W L, Yin H. Droidapiminer: Mining API-level features for robust malware detection in Android. In: Proc. of the 2013 Int'l Conf. on Security and Privacy in Communication Networks (SecCom 2013). 2013. 86−103.

[15] Zhao M, Ge FB, Zhang T, Yuan ZJ. AntiMalDroid: An efficient SVM-based malware detection framework for Android. In: Proc. of the 2nd Int'l Conf. on Information Computing and Applications (ICICA 2011). 2011. 158−166. [doi: 10.1007/978-3-642-27503-6_22]

[16] Zhu ZY, Dumitras T. FeatureSmith: Automatically engineering features for malware detection by mining the security literature. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2016). 2016. 767−778. [doi: 10.1145/2976749.2978304]

[17] Kouliaridis V, Potha N, Kambourakis G. Improving Android malware detection through dimensionality reduction techniques. In: Proc. of the 2020 Int'l Conf. on Machine Learning for Networking (ICMLN). 2020. 57−72. [doi: 10.1007/978-3-030-70866-5_4]

[18] Cai L, Li Y, Xiong Z. JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. Computers & Security, 2021, 100: 102086. [doi: 10.1016/j.cose.2020.102086]

[19] Miao XC, Wang R, Xu L, Zhang WF, Xu BW. Security analysis for Android applications using sensitive pathidentification. Ruan Jian Xue Bao/Journal of Software, 2017, 28(9): 2248−2263 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5177.htm [doi: 10.13328/j.cnki.jos.005177]

[20] Arp D, Spreitzenbarth M, Hubner M, Gascon1 H, Rieck K. DREBIN: Effective and explainable detection of Android malware in your pocket. In: Proc. of the 21st Annual Network and Distributed System Security Symp. (NDSS 2014). 2014. 1−15.

[21] Daoudi N, Allix K, Bissyandé TF, Klein J. A deep dive inside Drebin: An explorative analysis beyond Android malware detection scores. ACM Trans. on Privacy and Security, 2022, 25(2): 1−28.

[22] Liu W. Research on a method of security detection for Android based on intent. Computer Technology and Development, 2019, 29(5): 102−106 (in Chinese with English abstract).

[23] Chao F, Yang Z, Du XH, Han B. Classified risk assessment method of Android application based on multi-factor clustering selection. Chinese Journal of Network and Information Security, 2021, 7(2): 161−173 (in Chinese with English abstract).

[24] Teufl P, Ferk M, Fitzek A, Hein D, Kraxberger S, Orthacker C. Malware detection by applying knowledge discovery processes to application metadata on the Android market (Google Play). Security and Communication Networks, 2016, 9(5): 389−419. [doi: 10.1002/sec.675]

[25] Fan M, Luo XP, Liu J, Nong C, Zheng QH, Liu T. CTDroid: Leveraging a corpus of technical blogs for Android malware analysis. IEEE Trans. on Reliability, 2019, 69(1): 124−138. [doi: 10.1109/TR.2019.2926129]

[26] Pandita R, Xiao XS, Yang W, Enck W, Xie T. WHYPER: Towards automating risk assessment of mobile applications. In: Proc. of the 2013 USENIX Security Symp. (USENIX Security 2013). 2013. 527−542.

[27] Chen K, Wang P, Lee Y, Wang XF, Zhang N, Huang HQ, Zou W, LiuP. Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-play scale. In: Proc. of the 24th USENIX Security Symp. (USENIX Security 2015). 2015. 659−674.

[28] Zhang M, Duan Y, Yin H, Zhao Z. Semantics-aware Android malware classification using weighted contextual API dependency graphs. In: Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2014). 2014. 1105−1116. [doi: 10.1145/2660267.2660359]

[29] Wu Y, Zou D, Yang W, Li X, Jin H. HomDroid: Detecting Android covert malware by social-network homophily analysis. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA 2021). 2021. 216−229. [doi: 10.1145/3460319.3464833]

[30] He Y, Liu Y, Wu L, Yang ZQ, Ren K, Qin Z. MsDroid: Identifying malicious snippets for Android malware detection. IEEE Trans. on Dependable and Secure Computing, 2022, 1−16. [doi: 10.1109/TDSC.2022.3168285]

[31] Hou SF, Ye YF, Song YQ, Abdulhayoglu M, *et al*. Hindroid: An intelligent Android malware detection system based on structured heterogeneous information network. In: Proc. of the 25th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining (KDD 2017). 2017. 1507−1515. [doi: 10.1145/3097983.3098026]

[32] Wu Y, Li X, Zou D, Yang W, Zhang X, Jin H. MalScan: Fast market-wide mobile malware scanning by social-network centrality analysis. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2019). 2019. 139–150.

[33] Yumlembam R, Issac B, Jacob SM, Yang LZ. IoT-based Android malware detection using graph neural network with adversarial defense. IEEE Internet of Things Journal, 2022, 1−13. [doi: 10.1109/JIOT.2022.3188583]

[34] Crussell J, Gibler C, Chen H. Attack of the clones: Detecting cloned applications on Android markets. In: Proc. the 17th European Symp. on Research in Computer Security (ESORICS 2012). 2012. 37−54. [doi: 10.1007/978-3-642-33167-1_3]

[35] Arzt S, Rasthofer S, Fritz C, Bodden E, Bartel A, Klein J, Traon YL, Octeau D, McDaniel P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taintanalysis for Android Apps. In: Proc. of the 2014 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2014). 2014. 259−269. [doi: 10.1145/2666356.2594299]

[36] Wang L, He DJ, Li L, Feng XB. Sparse framework based static taint analysis optimization. Journal of Computer Research and Development, 2019, 56(3): 480−495 (in Chinese with English abstract).

[37] Ma K, Guo SQ. Security analysis of the third-party SDKs in the Android ecosystem. Ruan Jian Xue Bao/Journal of Software, 2018, 29(5): 1379−1391 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5497.htm [doi: 10.13328/j.cnki.jos.005497]

[38] Li L, Bartel A, Bissyande T, Klein J, Traon YL, Arzt S, Rasthofer S, Bodden E, Octeau D, McDanielP. ICCTA: Detecting inter-component privacy leaks in Android Apps. In: Proc. the 37th Int'l Conf. on Software Engineering (ICSE 2015). 2015. 280−291.

[39] Octeau D, Luchaup D, Dering M, Jha S, McDaniel P. Composite constant propagation: Application to Android inter-component communication analysis. In: Proc. of the 37th Int'l Conf. on Software Engineering (ICSE 2015). 2015. 77−88.

[40] Nan YZ, Yang M, Yang ZM, Zhou SF, Gu GF, Wang XF. UIPicker: User-input privacy identification in mobile applications. In: Proc. of the 24th USENIX Security Symp. (USENIX Security 2015). 2015. 993–1008.

[41] Huang JJ, Li ZC, Xiao XS, Wu ZY, Lu KJ, Zhang XY, Jiang GF. SUPOR: Precise and scalable sensitive user input detection for Android Apps. In: Proc. of the 24th USENIX Security Symp. (USENIX Security 2015). 2015. 977–992.

[42] Enck W, Gilbert P, Han S, Tendulkar V, Chun BG, Cox LP, Jung J, Msdaniel P, Sheth A. TaintDroid: An information-flow tracking system for realtime privacy monitoringon smartphones. ACM Trans. on Computer Systems, 2014, 32(2): 1–29. [doi: 10.1145/2619091]

[43] Hornyack P, Han S, Jung J, Schechter S, Wetherall D. These aren't the droids you're looking for: Retrofitting Android to protect data from imperious applications. In: Proc. of the 2011 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2011). 2011. 639–652. [doi: 10.1145/2046707.2046780]

[44] Desnos A. Androguard: Reverse engineering, malware and goodware analysis of Android applications. BlackHat, 2013.

[45] https://developer.android.com/reference/classes

[46] Gong L, Li Z, Qian F, Zhang ZF, Chen QA, Qian ZY, Lin H, Liu YH. Experiences of landing machine learning onto market-scale mobile malware detection. In: Proc. of the 2020 European Conf. on Computer Systems (EuroSys 2020). 2020. 1–14. [doi: 10.1145/3342195.3387530]

[47] O'Shea K, Nash R. An introduction to convolutional neural networks. arXiv:1511.08458, 2015.

[48] VirusTotal: Analyze suspicious files and URLs to detect types of malware, automatically share them with the security community. https://www.virustotal.com/gui/home/upload

[49] Zou D, Wu Y, Yang S, Chauhan A, Yang W, Zhong JY, Dou SH, Jin H. IntDroid: Android malware detection based on API intimacy analysis. ACM Trans. on Software Engineering and Methodology, 2021, 30(3): 1–32.

[50] Aonzo S, Georgiu GC, Verderame L, Merlo A. Obfuscapk: An open-source black-box obfuscation tool for Android Apps. SoftwareX, 2020, 11: 100403. [doi: 10.1016/j.softx.2020.100403]

[51] Grover A, Leskovec J. Node2Vec: Scalable feature learning for networks. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD 2016). 2016. 855–864. [doi: 10.1145/2939672.2939754]

附中文参考文献:

[19] 缪小川, 汪睿, 许蕾, 张卫丰, 徐宝文. 使用敏感路径识别方法分析安卓应用安全性. 软件学报, 2017, 28(9): 2248–2263. http://www.jos.org.cn/1000-9825/5177.htm [doi: 10.13328/j.cnki.jos.005177]

[22] 刘玮. 一种基于意图的安卓应用安全检测方法研究. 计算机技术与发展, 2019, 29(5): 102–106.

[23] 超凡, 杨智, 杜学绘, 韩冰. 基于多因素聚类选择的 Android 应用程序分类风险评估方法. 网络与信息安全学报, 2021, 7(2): 161–173.

[36] 王蕾, 何冬杰, 李炼, 冯晓兵. 基于稀疏框架的静态污点分析优化技术. 计算机研究与发展, 2019, 56(3): 480–495.

[37] 马凯, 郭山清. 面向 Android 生态系统中的第三方 SDK 安全性分析. 软件学报, 2018, 29(5): 1379–1391. http://www.jos.org.cn/1000-9825/5497.htm [doi: 10.13328/j.cnki.jos.005497]

吴月明(1993－), 男, 博士, CCF 学生会员, 主要研究领域为移动安全, 软件供应链安全, 人工智能安全, 恶意软件分析, 漏洞分析, 克隆代码审计.

齐蒙(1998－), 男, 硕士, 主要研究领域为机器学习, 漏洞检测, 安卓恶意软件检测.

邹德清(1975－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为云计算安全, 网络攻防与漏洞检测, 软件定义安全与主动防御, 大数据安全与人工智能安全, 容错计算.

金海(1966－), 男, 博士, 教授, 博士生导师, CCF 会士, IEEE 会士, ACM 终身会员, 主要研究领域为计算机系统结构, 虚拟化技术, 集群计算, 网格计算, 并行与分布式计算, 对等计算普适计算, 语义网, 存储与安全.