软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(4):1715−1739 [doi: 10.13328/j.cnki.jos.007499] [CSTR: 32375.14.jos.007499]

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

RMDroid: 基于多模态融合学习的安卓恶意软件鲁棒检测*

凌 祥[1,2], 周伯霖[1,2], 王时予[1,2], 罗天悦[1], 尹 鹏[2,3], 吴春明[4], 王 滨[5], 吴敬征[1,2]

[1](中国科学院 软件研究所, 北京 100190)

[2](中国科学院大学, 北京 100049)

[3](军工保密资格审查认证中心, 北京 100089)

[4](浙江大学 计算机科学与技术学院, 浙江 杭州 310027)

[5](杭州海康威视网络与信息安全实验室, 浙江 杭州 310052)

通信作者: 吴敬征, E-mail: jingzheng08@iscas.ac.cn

摘 要: 随着人工智能技术的蓬勃发展和广泛应用, 越来越多的恶意软件检测方法和工具利用深度学习的强大学习能力来检测安卓平台上新出现的恶意软件. 然而, 深度学习模型已经被证明容易受到对抗攻击的威胁. 与此同时, 攻击者已经开始提出多种针对安卓恶意软件检测方法的对抗攻击方法, 即生成对抗性安卓恶意软件, 从而达到绕过恶意软件检测的目的. 现有安卓恶意软件检测方法容易受到对抗攻击威胁的主要原因在于, 这些恶意软件检测方法都建立在单一模态特征之上, 而以单一模态存在的特征却很容易被攻击者恶意性地操控. 因此, 为了提高当前安卓恶意软件检测方法可以抵御对抗攻击的鲁棒性, 提出一种基于多模态融合学习的安卓恶意软件鲁棒检测方法 RMDroid, 可以在不影响针对一般性安卓恶意软件检测准确性的基础上, 显著提高其抵御对抗攻击的鲁棒性. 具体而言, RMDroid 首先会从待测安卓软件中提取多种模态的特征信息, 然后分别利用相应的深度学习模型学习表征相应模态深层语义信息的特征向量, 最后利用异类识别网络降低甚至消除多模态特征中受到对抗攻击干扰的模态特征对最终恶意软件预测的影响, 从而提高其抵御对抗攻击的鲁棒性. 实验结果表明, 所提出的 RMDroid 在 5 项有效性指标和 1 项鲁棒性指标上均优于所有基线检测方法. 特别的, 在误报率 *FPR* 相同的情况下, RMDroid 的检出率 *TPR* 比最好的基线检测方法的 *TPR* 高出 10% 以上; 并且针对最先进的 HRAT 攻击, RMDroid 的鲁棒性值高达 96% 以上, 显著高于 MaMaDroid 和 MalScan 基线检测方法的鲁棒性值.

关键词: 安卓恶意软件; 鲁棒检测; 对抗攻击; 多模态学习; 融合学习

中图法分类号: TP311

中文引用格式: 凌祥, 周伯霖, 王时予, 罗天悦, 尹鹏, 吴春明, 王滨, 吴敬征. RMDroid: 基于多模态融合学习的安卓恶意软件鲁棒检测. 软件学报, 2026, 37(4): 1715–1739. http://www.jos.org.cn/1000-9825/7499.htm

英文引用格式: Ling X, Zhou BL, Wang SY, Luo TY, Yin P, Wu CM, Wang B, Wu JZ. RMDroid: Android Malware Robust Detection Based on Multi-modal Fusion Learning. Ruan Jian Xue Bao/Journal of Software, 2026, 37(4): 1715–1739 (in Chinese). http://www.jos.org.cn/1000-9825/7499.htm

RMDroid: Android Malware Robust Detection Based on Multi-modal Fusion Learning

LING Xiang[1,2], ZHOU Bo-Lin[1,2], WANG Shi-Yu[1,2], LUO Tian-Yue[1], YIN Peng[2,3], WU Chun-Ming[4], WANG Bin[5], WU Jing-Zheng[1,2]

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](Defense Industry Secrecy Examination and Certification Center, Beijing 100089, China)

* 基金项目: 国家自然科学基金 (62202457); 中国博士后科学基金 (2022M713253); 国防基础科研计划 (JCKY2021906B002)
收稿时间: 2024-01-11; 修改时间: 2025-06-03; 采用时间: 2025-06-27; jos 在线出版时间: 2025-12-10
CNKI 网络首发时间: 2025-12-11

[4](College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)
[5](Network and Information Security Laboratory of Hangzhou Hikvision Digital Technology Co. Ltd., Hangzhou 310052, China)

Abstract: With the booming development and wide application of artificial intelligence, more and more deep learning-based Android malware detection methods and tools have been developed to detect newly emerged Android malware. However, deep learning models have been extensively proven to be vulnerable to adversarial attacks. Meanwhile, attacker shave started to propose adversarial attacks against Android malware detection methods to generate adversarial Android malware that can bypass detection. This study argues that the main reason current Android malware detection methods are vulnerable to such adversarial attacks is that these detectors are mostly built on single-modal features, which can be easily manipulated by attackers. Therefore, to improve the robustness of Android malware detection against adversarial attacks, the study proposes a robust Android malware detection method based on multi-modal fusion learning, namely RMDroid. RMDroid improves robustness in identifying adversarial malware without sacrificing accuracy in general Android malware detection. Specifically, RMDroid first extracts feature information from different modalities of Android APPs and then uses the corresponding deep learning models to sufficiently learn feature vectors that characterize the deep semantics of each modality. Finally, an odd-one-out network is employed to reduce or even eliminate the influence of interfered modal features on the final malware prediction, thus improving robustness against adversarial attacks. The experimental results show that RMDroid achieves higher performance across five effectiveness metrics and one robustness metric compared to all baseline detection methods. In particular, given the same *FPR*, the *TPR* value of RMDroid is more than 10% higher than that of the best baseline detection method. In the case of the state-of-the-art adversarial attack of HRAT, RMDroid achieves over 96% in robustness, which is significantly higher than the robustness of both MaMaDroid and MalScan.

Key words: Android malware; robust detection; adversarial attack; multi-modal learning; fusion learning

安卓 (Android) 是一款基于 Linux 内核开发且面向移动设备的开源移动操作系统. 相比于其他移动操作系统, 安卓系统具有高度的开放性和自由性, 开发人员可以自由地定制、修改和新增不同功能用以满足不同应用场景的需求 (例如, 智能手机、平板电脑、智能电视、智能手表、智能家居、可穿戴设备等), 从而可以给大量移动终端用户带来便捷、智能和人性化的用户体验. 根据全球知名市场研究咨询公司 Strategy Analytics 的调研, 2022 年全球销售的所有移动终端中安卓的份额占比最高, 占有率达到 81%, 持续占据了全球市场中移动操作系统霸主地位[1,2].

随着安卓操作系统在人们日常工作、生活和娱乐中的广泛使用和普及, 大量针对安卓操作系统的恶意软件也随之应运而生, 并且成为当前移动互联网上最大的安全威胁之一. 具体而言, 安卓恶意软件主要指在未经用户允许的情况下被安装到安卓操作系统并且执行恶意行为 (例如, 破坏移动设备功能、收集用户隐私信息、恶意地消耗资源或者扣费等) 的应用程序. 常见的安卓恶意软件包括广告软件、手机木马、勒索软件等. 根据全球知名网络安全公司卡巴斯基的统计, 2022 年总共检测出 1 661 734 个新的恶意软件安装包, 并且相关安卓恶意软件在官方应用商店 Google Play 和其他第三方应用商店内广泛传播, 对大量的用户造成严重的安全威胁[3]. 特别地, 2022 年卡巴斯基公司新检测到安卓木马程序 Harly, 仅在官方应用商店 Google Play 上的下载量就超过 480 万, 不仅可以窃取用户隐私数据, 而且恶意订阅用户不需要的付费服务[4]. 此外, 在统计所有受到恶意软件攻击影响的国家之中, 中国的移动终端用户被攻击的比例最高, 达到 17.70%[3]. 上述统计都表明安卓恶意软件已经严重威胁终端用户乃至国家安全.

为此, 如何及时且准确地检测安卓操作系统中存在的恶意软件一直是学术界的研究热点[5], 同时工业界中大量知名的安全公司也持续推出针对安卓平台的恶意软件检测工具, 包括 Google Play Protect[6]、Norton Mobile Security for Android[7]、腾讯手机管家[8]、360 手机卫士[9]等. 一般而言, 这些安卓恶意软件检测方法和工具主要可以分成基于应用程序静态特征的静态检测方法和基于应用程序运行时动态特征的动态检测方法. 静态检测方法通常不需要实际运行应用程序, 直接对应用程序的代码或文件本身进行分析, 而动态检测方法通过使用沙箱[10]、模拟器等工具来监测和分析应用程序运行时的行为来检测恶意软件. 传统的基于签名或特征匹配的恶意软件检测方法属于静态检测方法的范畴, 主要通过分析待测软件与恶意软件库的签名或者特征之间的相似度来判断待测软件是否为恶意软件[11−13]. 显然, 这类检测方法不仅需要安全专家手工挖掘和更新大量安全规则, 而且无法检测新出现的恶意软件库之外的恶意软件[13].

进一步地, 随着人工智能技术在图像、语音、自然语言处理等领域的蓬勃发展, 越来越多的安卓恶意软件检测方法和工具开始利用机器学习或者深度学习模型的强大学习能力来检测新出现的恶意软件, 称之为基于学习的安卓恶意软件检测方法, 从而弥补传统安卓恶意软件检测方法的不足[14]. 具体而言, 基于学习的安卓恶意软件检测方法首先从安卓软件 (即 APK 文件) 中提取不同类型的特征 (例如, 权限、组件、API 调用、控制流、字节码等) 并进一步将其表示成相应的特征向量, 然后输入到特定的机器学习模型或者深度学习模型中进行训练, 从而使得训练后的模型具备识别安卓恶意软件的能力. 这种最新的基于学习的安卓恶意软件检测方法可以基于软件的静态特征或动态特征来学习恶意软件和良性软件之间的潜在差异, 在部署后获得突出的恶意软件检测效果.

然而, 近年来机器学习或者深度学习在计算机视觉、语音识别、自然语言处理等领域已经被广泛证明是容易受到对抗样本攻击威胁的[15]. 在恶意软件检测领域中, 随着基于学习的安卓恶意软件检测方法的广泛使用, 攻击者开始考虑如何利用对抗攻击方法绕过这些基于学习的安卓恶意软件检测方法[16], 从而达到恶意攻击安卓操作系统的目的. 例如, 针对基于函数调用图的安卓恶意软件检测模型, Zhao 等人[17]提出的一种基于启发式优化强化学习的结构化对抗攻击方法 HART, 在保证安卓恶意软件语义不变的情况下, 通过修改函数调用图生成对抗性安卓恶意软件, 从而可以以超过 90% 的攻击成功率绕过 MalScan[18]、MaMaDroid[19]等当前先进的安卓恶意软件检测方法.

综上, 本文认为当前基于学习的安卓恶意软件检测方法[20]主要建立在单一模态 (即某一特定类型的特征, 例如, 函数调用图等) 之上, 而以单一模态存在的特征是非常容易被攻击者所操控的 (例如, 修改函数间的调用关系, 增加空函数等), 这是当前基于学习的安卓恶意软件检测方法容易受到对抗攻击威胁的根本原因[21].

因此, 研究人员在利用多模态软件特征进行恶意软件检测方面作出了一些探索工作, 相比单一模态取得了更高的鲁棒性. 例如, Kim 等人[22]从安卓软件的 Manifest 文件、Dalvik 字节码以及共享库中构造丰富的静态特征并生成相应的特征向量, 最终结合多模态深度学习模型进行检测. 2025 年, Zhang 等人[23]从安卓软件中提取由函数调用图表示的静态特征和 API 调用图表示的动态特征进行安卓恶意软件检测. Li 等人[24]利用预训练的语言模型和视觉模型分别从安卓软件的源代码以及二进制代码中提取特征进行训练并预测恶意软件. Chen 等人[25]提出了一种类集合调用图的新表征来融合结构和语义特征, 并使用多模态特征融合网络来进行恶意软件识别. 然而, 尽管这些方法从不同角度融合安卓软件的多模态特征, 但本质上是采用冗余的思想提高模型检测的鲁棒性, 单一模态下的对抗性扰动产生的影响随着特征融合被传递到上层的分类预测网络中, 导致恶意软件检测模型仍然不可避免地受到对抗性攻击的影响导致鲁棒性降低.

因此, 为了进一步提高当前基于学习的安卓恶意软件检测方法的鲁棒性, 本文提出了一种基于多模态融合学习的安卓恶意软件鲁棒检测方法 RMDroid, 通过主动识别可能被扰动的模态从而自适应加权削弱被扰动的模态特征给模型预测结果带来的负面影响, 进而在保证不降低甚至提高当前恶意软件检测准确性的情况下, 提高其抵御对抗样本攻击的鲁棒性. 具体而言, RMDroid 首先从待测安卓软件中提取与挖掘不同模态的特征信息, 具体包括灰度图模态特征、API 调用序列模态特征和控制流图 (control flow graph, CFG) 模态特征; 其次, 基于所提取的 3 种模态特征, RMDroid 分别利用相应的深度学习模型, 包括卷积神经网络 (convolutional neural network, CNN)、循环神经网络 (recurrent neural network, RNN) 和图神经网络 (graph neural network, GNN), 充分地学习灰度图模态、API 调用序列模态和控制流图模态的深层语义信息, 从而得到相应模态的特征向量; 最后, 针对所有学习得到的多模态特征向量, 提出基于鲁棒融合学习的恶意软件鲁棒检测模型, 通过降低甚至消除多模态特征中受到对抗攻击干扰的模态特征对最终恶意软件预测的影响, 从而提升其抵御对抗攻击的鲁棒性.

为了充分地评估与验证 RMDroid 检测一般性恶意软件的检测有效性和检测对抗性恶意软件的鲁棒性, 本文在包含 28 456 个恶意安卓软件和善意安卓软件的标准数据集上, 系统地对比了 RMDroid 与 3 个现有最好的安卓恶意软件基线检测方法的有效性指标和鲁棒性指标. 实验结果表明, 在有效性评估方面, 本文所提出的 RMDroid 的 5 个有效性指标均高于所有基线检测方法的有效性指标, 特别是在误报率 (false positive rate, *FPR*) 相同的情况下检出率 (true positive rate, *TPR*) 比最好的基线检测方法的 *TPR* 值还要高出 10% 以上; 在鲁棒性评估方面, 设置了 3 种不同的对抗攻击情况, 并选择了 3 种最具代表性的基线恶意软件检测方法与 RMDRroid 进行比较, 最终结

果显示 RMDroid 鲁棒性指标 Robust 高于所有基线检测方法, 特别在最先进的 HRAT 攻击的情况下, RMDroid 的 *Robust* 值高达 96% 以上, 远远高于 MaMaDroid 和 MalScan 基线检测方法的鲁棒性.

综上所述, 本文的主要贡献总结如下.

(1) 针对当前安卓恶意软件检测方法面临对抗攻击的严重安全威胁, 通过实验证明基于单一模态特征的恶意软件检测方法很容易被攻击者所绕过, 并据此提出一种基于多模态融合学习的安卓恶意软件鲁棒检测方法 RMDroid, 可以极大地提高攻击者同时操控多模态特征所需要的攻击代价, 从而提高恶意软件检测的鲁棒性.

(2) 在提取并学习安卓恶意软件多模态特征的基础上, RMDroid 设计并实现一种异类识别网络, 可以识别多模态特征信息中存在的不一致模态甚至定位受到对抗攻击干扰的模态, 从而为最终的安卓恶意软件鲁棒检测任务提供一致性且不受对抗攻击干扰的模态特征信息.

(3) 在包含 28 456 个恶意安卓软件和善意安卓软件的标准数据集上进行了系统的实验评估, 结果表明, RMDroid 相比于所有基线检测方法表现出更优的有效性和鲁棒性, 即不仅可以有效地检测一般性安卓恶意软件, 而且可以检测出攻击者恶意生成的对抗性安卓恶意软件.

本文第 1 节详细介绍背景知识及相关工作. 第 2 节介绍基于多模态融合学习的安卓恶意软件鲁棒检测方法 RMDroid 设计与实现. 第 3 节是对应的实验评估及结果分析. 最后第 4 节总结本文并展望未来.

1 基础知识与相关工作

1.1 安卓软件介绍

安卓是一个由谷歌公司领衔开发的基于 Linux 内核的移动操作系统. 得益于安卓的高度开放性和自由性, 开发人员可以自由地定制、修改和新增不同的功能以满足不同场景的需求, 因此安卓持续占据全球移动操作系统的霸主地位. 为了规范安卓系统中软件 (或称应用程序) 的开发、分发、安装和运行过程中的文件格式, 开发人员必须将安卓软件编译打包成一个统一文件格式——APK[26], 其中包含安卓软件安装和运行过程中所有文件的压缩文件, 通常由以下几类文件构成.

● 代码文件: 主要定义了安卓软件的功能和行为, 包括逻辑处理、数据管理、用户界面等. 通常而言, 一个安卓软件中的 Java 代码会被编译并整合成一个或多个 Dalvik 可执行文件, 而使用 NDK (native development kit) 编写的本地代码 (通常以 C/C++语言编写) 被编译为.so 共享库文件, 最终打包成 APK 文件形式.

● 资源文件: 主要包含安卓软件中安装运行过程中所需的各种非代码的多媒体数据, 包括图标、图片、音视频、布局文件等, 并以资源标识符的形式在应用程序中被引用.

● 清单文件: 是描述安卓软件整体结构和属性的元数据文件, 具体包括包名、版本号、所需权限等, 通常是以 XML 文件格式存在.

● 其他文件: 指除了上述主要文件之外的其他辅助文件. 例如, 签名文件、库文件、配置文件等, 用以支撑安卓软件的安装运行.

1.2 安卓恶意软件检测方法

随着安卓操作系统的广泛普及和移动应用的快速增长, 安卓中的恶意软件也日益剧增, 已经成为威胁移动终端用户及其移动设备安全的重要因素. 因此, 为了识别并阻止安卓恶意软件的传播, 学术界和工业界已经提出并实现大量的安卓恶意软件检测方法和工具, 包括传统的基于签名或特征匹配的安卓恶意软件检测方法和最新的基于学习的安卓恶意软件检测方法. 具体而言, 传统的基于签名或特征匹配的安卓恶意软件检测方法, 首先利用安全领域专家知识提取相关特征码来构建相应的恶意软件签名库或者特征库, 然后将待测软件与恶意软件特征库进行匹配, 从而判断待测软件是否为恶意软件[27]. 例如, Aafer 等人[28]利用静态代码分析方法从恶意软件库中挖掘相应的抽象恶意模式, 并通过模式匹配来识别安卓恶意软件. Grace 等人[29]首先将安卓恶意软件风险分成高、中、低这 3 个级别, 然后利用基于指纹和控制流图等特征匹配方法分别检测不同级别的恶意软件. 然而, 由于这类方法严重依赖领域专家知识和有限恶意软件样本所构建的特征库, 因此显然无法检测出特征库中不包含的恶意软件, 更无法检测最新出现的恶意软件.

为了提高安卓恶意软件检测方法针对新出现恶意软件的检测能力, 越来越多的研究人员将各种人工智能技术应用到安卓恶意软件检测任务之中, 一般称这类检测方法为基于学习的安卓恶意软件检测方法. 这类检测方法首先会从安卓软件中提取相应的特征, 包括静态特征 (例如, 原始字节、操作码、API 调用等) 或者动态特征 (例如, 文件操作、网络流量、硬件状态等), 然后输入到合适的传统机器学习模型或者深度学习模型中进行训练, 从而使得训练后的模型具备检测恶意软件的能力. 例如, Droid-Sec[30]是首个尝试使用深度学习方法来检测安卓恶意软件的, 其具体流程是, 首先利用静态分析和动态分析相结合的方式提取 200 多种特征, 然后训练基于深度神经网络的安卓恶意软件检测模型. Arp 等人[31]提出了一种高效且可解释的安卓恶意软件检测方法 Drebin, 具体采用轻量级静态分析方法从代码和清单文件中提取特征, 然后训练支持向量机模型进行安卓恶意软件检测. Onwuzurike 等人[19]提出的 MaMaDroid 恶意软件检测方法, 首先利用静态分析方法获得 API 调用图, 然后利用提取的 API 调用序列构建相应的马尔可夫链作为相应的行为特征, 最后使用随机森林模型等机器学习模型进行安卓恶意软件检测. 此外, Yang 等人[32]提出了一种基于图像和卷积神经网络的安卓恶意软件检测方法, 将软件的 Dalvik 字节码转换为 RGB 图像, 直接用卷积神经网络模型对图像进行训练和分类, 从而识别恶意软件. 类似地, Tang 等人[33]提出了一种基于新型混合字节码图像与注意力机制的安卓恶意软件分类方法. 该方法分别生成可执行文件的灰度图像与马尔可夫图像, 利用状态转移概率对灰度图像与马尔可夫图像进行融合, 构建出新的纹理特征空间, 并将其映射为二维图像, 最终利用混合图像特征进行安卓恶意软件预测.

1.3 对抗样本攻击及防御方法

尽管以深度学习为代表的人工智能模型已经成功大量地应用在图像识别、自然语言处理、语音识别等多个领域中, 但是近年来相关研究已经证明人工智能模型非常容易受到对抗攻击[34,35], 即攻击者通过在原始输入数据中添加微小但精心构造的扰动来生成对抗样本, 从而误导目标模型产生如攻击者所期望的错误结果. 根据攻击者对目标模型 (例如, 模型架构、参数、特征表示等信息) 的掌握程度, 对抗样本攻击可以分成白盒攻击、灰盒攻击与黑盒攻击. 其中, 白盒攻击指的是攻击者掌握目标模型的所有信息, 是攻击者利用对抗样本攻击所能达到攻击效果的上限; 黑盒攻击指的是攻击者除了目标模型的输出外, 不了解目标模型的其他任何信息; 灰盒攻击介于白盒和黑盒攻击之间, 指的是攻击者只能掌握目标模型的部分信息.

自从 2013 年对抗样本攻击方法首次在图像识别领域被提出以来[36], 大量的研究人员不仅在计算机视觉领域提出大量的对抗样本攻击方法[37−39], 而且逐渐扩展至自然语言处理、语音识别、代码分析、恶意软件检测等领域[16,40−42]. 在恶意软件检测领域中, 已经有大量的研究发现攻击者利用提出的对抗样本攻击方法生成相应的对抗性恶意软件, 可以绕过恶意软件检测模型甚至是商业杀毒软件的检测. 具体而言, 根据恶意软件文件格式的不同, 研究人员针对 Windows PE 恶意软件[16]、恶意 PDF 文件[43]、安卓恶意软件[17]等不同格式的恶意软件的特征, 提出大量不同的对抗样本攻击方法, 已经严重威胁到计算机系统的网络安全.

为了充分探索基于各种特征的恶意软件检测模型的鲁棒性, 研究人员提出了许多不同的对抗攻击方法. 针对基于图像模态的安卓恶意软件检测模型的对抗性攻击, 现有研究从两个方面进行了探索, 即问题空间和特征空间: (1) 问题空间的攻击方法通过在实际的恶意软件上添加字节或修改不影响软件功能的字节, 首先生成实际的恶意样本, 然后通过预处理流程映射到图像模态. 然而, 问题空间受限于软件功能一致性的要求以及软件内部组织结构规范的限制, 无法在特征空间中映射连续的对抗样本进行精细的搜索[16]; (2) 得益于图像数据的连续性, 特征空间的攻击方法能够直接生成连续的、精心构造的对抗性图像数据, 从而对恶意软件检测模型发起攻击, 评估模型的鲁棒性上界. 例如, Vi 等人[44]提出了一种基于传统图像领域对抗攻击 FGSM 的对抗性恶意软件生成方法, 通过在 PE 文件的资源部分引入扰动生成对抗样本使其转换成的图像被恶意软件检测模型错误识别. Darwaish 等人[45]则通过在恶意软件转换形成的图像之后拼接良性软件的对应图像, 构造对抗样本以误导恶意软件分类器. Naeem 等人[46]直接使用 FGSM、PGD 以及 Carlini-Wagner 等图像领域的对抗性攻击生成对抗性恶意软件对应的 RGB 图像, 用于评估其所提出的恶意软件检测系统的鲁棒性. 针对图像模态的攻击方式虽然利用图像领域的对抗性攻击方法, 但其原理实质上是对底层特征空间的扰动, 这种扰动在编码过程中仍可能传递到最终模型输入, 因此能够影

响模型的检测结果.

针对基于序列的深度学习恶意软件分类器, Tan 等人[47]提出基于深度 Q 网络 (deep Q-network, DQN) 和启发式回溯搜索的对抗攻击方法, 构造针对序列的扰动动作集, 映射修改到源代码, 生成满足实际问题空间约束的扰动序列. 此外, Zhang 等人[48]提出了 SeqAdver 攻击方法, 从良性 APK 中提取 Smali 代码, 过滤用户可见 API 和 Intent, 归一化处理后, 按序列位置或启动类注入目标文件, 实现自动有效负载构造与注入, 生成对抗性安卓恶意软件. 针对基于图结构模态 (例如控制流图、函数调用图等) 进行恶意软件检测的模型, Zhao 等人[17]提出了一种基于结构化扰动生成对抗性函数调用图的攻击方法, 能够有效地绕过基于函数调用图的恶意软件检测模型. Li 等人[49]利用通过向安卓恶意软件中插入不会被执行的函数调用, 生成对抗性函数调用图使恶意软件检测模型产生错误分类. 除此之外, He 等人[50]提出了一种通用的对抗性攻击方法, 通过扰动搜索树构建预定义的扰动操作, 并基于模型的预测置信度变化优化搜索过程, 通过向恶意软件插入良性的组件、权限等元素构造对抗性恶意软件绕过恶意软件检测器.

随着对抗样本攻击的快速发展, 研究人员也提出一系列的防御方法, 旨在提高人工智能模型抵御对抗样本攻击的鲁棒性[51,52]. 特别在图像分类、语音识别等传统领域中, 针对对抗样本攻击的防御方法主要分为数据级别的防御方法和模型级别的防御方法两大类[53]. 其中, 数据级别的防御方法主要通过对输入样本进行一定的处理进而消除其中可能被添加的对抗扰动, 常见的数据处理方法包括去噪、滤波、特征预处理、特征编码[24]等; 而模型级别的防御方法主要通过改进或者加强人工智能模型及其训练过程来提高针对对抗样本攻击的鲁棒性, 常见的方法包括蒸馏防御[54]、梯度正则化[55]、对抗训练[56]等. 然而, 由于恶意软件通常是以二进制的形式存在, 并且具备高度复杂的代码结构和行为逻辑, 因此上述图像分类、语音识别等领域中的防御方法或者无法直接应用到恶意软件检测领域中, 或者应用后所取得的防御效果极其有限[16]. 因此, 当前亟需针对恶意软件检测领域, 特别是针对安卓恶意软件检测模型的特点, 设计相对应的可以抵御对抗性恶意软件的防御方法, 进而提高安卓恶意软件检测方法的鲁棒性.

2 RMDroid 设计与实现

2.1 RMDroid 总览

与一般安卓恶意软件检测任务不同的是, 安卓恶意软件鲁棒检测任务不仅要求能够正确地识别善意软件和一般性恶意软件, 而且需要能够抵御对抗攻击的干扰, 即能够正确地识别攻击者生成的对抗性恶意软件. 形式上, 给定一个待测安卓软件 z, 恶意软件鲁棒检测任务的目的是预测 z 为恶意软件的概率 $p_z \in [0,1]$, 并进一步检测其是否为恶意软件 $y_z \in \{0,1\}$, 其中 $y_z = 0$ 代表为善意软件, 而 $y_z = 1$ 代表 z 为恶意软件, 既包括一般性恶意软件, 也包括对抗性恶意软件.

为此, 如图 1 所示, 本文提出一种基于多模态融合学习的安卓恶意软件鲁棒检测方法 RMDroid, 在保证其具备准确识别一般性恶意软件的有效性基础上, 大幅度提高其识别对抗性恶意软件的鲁棒性. 具体而言, 本文所提出的 RMDroid 鲁棒检测方法主要包括 3 个核心部分.

(1) 面向安卓恶意软件的多模态特征提取: 首先, RMDroid 从待测安卓软件中提取与挖掘不同模态的特征信息, 具体包括灰度图模态特征、API 调用序列模态特征和控制流图模态特征. 详见第 2.2 节.

(2) 基于深度学习模型的多模态特征学习: 其次, 基于上述所提取的安卓软件多种模态特征信息, RMDroid 分别利用相应的深度学习模型, 包括卷积神经网络、循环神经网络和图神经网络, 充分地学习灰度图模态、API 调用序列模态和控制流图模态的深层语义信息, 从而得到相应模态的特征向量. 详见第 2.3 节.

(3) 基于鲁棒融合学习的恶意软件鲁棒检测: 最后, 针对所有学习得到的多模态特征向量, 进一步提出基于鲁棒融合学习的恶意软件鲁棒检测模型, 通过降低甚至消除多模态特征中受到对抗攻击干扰的模态特征对最终恶意软件预测的影响, 从而提升 RMDroid 抵御对抗攻击的鲁棒性. 详见第 2.4 节.

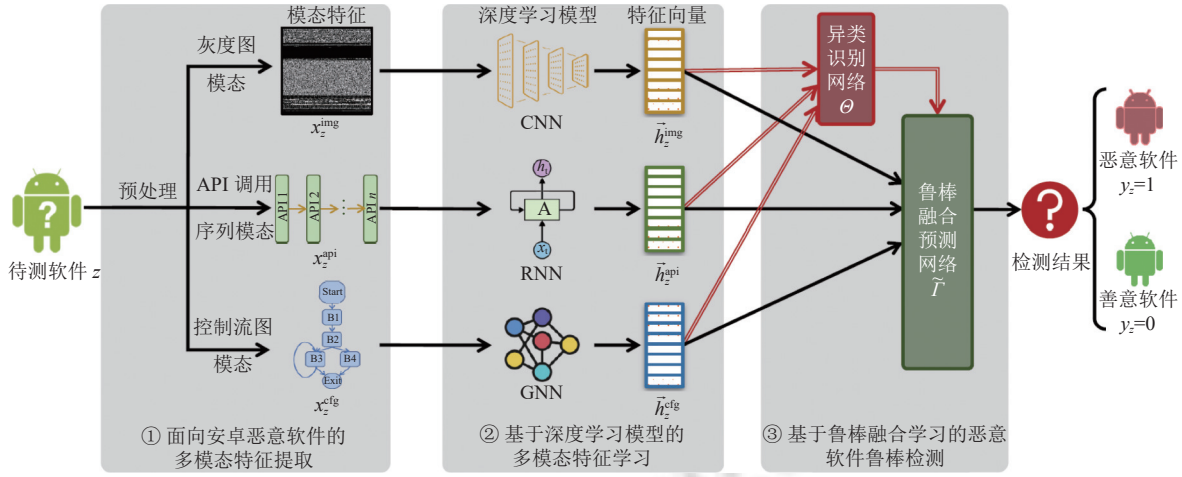

图 1 基于多模态融合学习的安卓恶意软件检测方法 RMDroid 的模型架构图

下面将在第 2.2–2.4 节分别详细介绍 RMDroid 的 3 个核心部分, 最后在第 2.5 节介绍整个 RMDroid 的训练方法. 本文使用的重要符号如表 1 所示.

表 1 本文重要符号表

符号	定义或描述
z	安卓软件样本
$y_z \in \{0,1\}$	样本 z 的预测结果
$\hat{y}_{z_i} \in \{0,1\}$	样本 z 的真实结果
$\mathbf{x}_z^m$	样本 z 在模态 m 下的初始化特征向量
$\bar{\mathbf{x}}_z^m$	样本 z 在模态 m 下的标准化特征向量
$\vec{h}_z^m$	样本 z 在模态 m 下学习得到的特征向量
$h_{v_i}^{(t)}$	节点 v_i 通过第 t 层图神经网络的特征向量
$\mathcal{N}(v_i)$	节点 v_i 的所有邻居节点的集合
n_z^{cfg}	安卓软件 z 的控制流图中的节点数量
Θ	异类识别网络
$\tilde{\Gamma}$	鲁棒融合预测网络
F_{robust}	基于鲁棒融合学习的恶意软件鲁棒检测方法
$p_\Theta^{(k)}$	异类识别网络 Θ 预测的第 k 个模态被扰动的概率
$\mathbf{H}_z = \{\vec{h}^{(k)}\}_{k=1}^{k=K}$	K 个不同模态构成的特征向量集合
$\Theta(\mathbf{H}_z) = \{p_\Theta^{(k)}\}_{k=1}^{k=K}$	K 个不同模态被扰动的概率
$\mathbf{S}(\mathbf{H}_z)$	特征向量 $\mathbf{H}_z$ 中所有特征通过 $\mathbf{S}$ 融合得到的向量
$\vec{e}_z = \mathbf{S}(\mathbf{H}_z)\Theta(\mathbf{H}_z)$	$\Theta(\mathbf{H}_z)$对特征融合向量 $\mathbf{S}(\mathbf{H}_z)$ 进行加权求和得到的鲁棒融合特征向量 $\vec{e}_z$

2.2 面向安卓恶意软件的多模态特征提取

随着软件工程技术的不断发展和软件功能需求的不断扩张, 包括恶意软件在内的各类软件的代码规模也在急剧增加, 因此很难全面地利用恶意软件中代码的所有特征信息来进行恶意软件检测. 为此, 大量的研究人员尝试从不同的角度来提取恶意软件不同层面的特征信息[18,57−59]. 例如, 恶意软件的灰度图特征信息可以展示其二进制代码的整体布局和结构信息[57], API 调用序列特征信息可以展示恶意软件中的潜在行为信息[18], 而控制流图相关的特征信息可以全面地展示恶意软件中代码的控制流结构及其语义信息[58,59].

为了全面捕获和充分挖掘安卓恶意软件中与恶意行为相关的语义特征信息, 本文主要通过提取待测安卓软件 z 中灰度图、API 调用序列和控制流图这 3 种模态的特征信息, 为后面的安卓恶意软件鲁棒检测提供多模态特征信息的支持. 下面将分别介绍针对灰度图模态、API 调用序列模态和控制流图模态的特征提取方法.

2.2.1 针对灰度图模态的特征提取

对于安卓应用程序而言, Java 源代码被编译为 Java 字节码后转换为 Dalvik 字节码, 以便在安卓操作系统上运行, 因此 Dalvik 字节码是应用程序功能最直接的表示形式. 通过将二进制 Dalvik 字节码转换成相应的图像形式, 不仅能够直观且可视化地展示二进制字节码序列, 而且能够充分体现二进制代码的整体布局和结构信息. 事实上, 当前已经有大量恶意软件检测相关研究直接将恶意软件表示成灰度图的形式并取得相当不错的恶意软件检测效果[57,60,61]. 恶意软件检测中灰度图特征有效的原因之一, 是由于恶意软件的开发人员通常会以某个原始恶意软件代码作为基础来生成新的恶意软件, 这往往使得新生成的恶意软件中大部分二进制代码保持不变, 因此防御方可以利用这种特点来识别恶意软件[60]. 下面将逐步介绍本文所提出的 RMDroid 如何提取待测安卓软件 z 的灰度图特征 $\mathbf{x}_z^{\text{img}}$, 具体步骤如下.

(1) 直接以二进制形式读取待测安卓软件 z, 并将连续的 8 位二进制数据转换成 0–255 之间的数字, 即形成一个包含十进制数字的数组.

(2) 根据基于灰度图的恶意软件检测方法的相关经验[57,61]和待测安卓软件 z 的实际大小, 确定相应的灰度图的宽度.

(3) 基于灰度图的宽度, 将待测安卓软件的十进制数组重新组织成二维灰度图形式, 即得到最终的灰度图模态特征 $\mathbf{x}_z^{\text{img}}$.

2.2.2 针对 API 调用序列模态的特征提取

一般而言, 安卓恶意软件需要通过调用大量的安卓 API 接口函数来执行一些安全敏感操作或者恶意行为. 因此可以利用待测安卓软件中的 API 调用信息来检测安卓恶意软件. 事实上, 这种基于 API 调用信息的恶意软件检测方法已经被大量研究人员广泛地应用于安卓恶意软件检测任务之中[18,19], 不仅取得了不错的检测效果, 而且具备更好的可解释性, 例如, 某些家族的安卓恶意软件中会更容易存在某几种相对固定的 API 调用模式. 进一步地, RMDroid 提取待测安卓软件 z 中的 API 调用序列特征 $\mathbf{x}_z^{\text{api}}$, 具体步骤如下.

(1) 利用逆向分析工具 Androguard 提取待测安卓软件 z 的函数调用序列, 并获得每个函数的调用信息, 包括函数名、参数类别、返回值等.

(2) 根据安卓系统函数列表识别函数调用序列中的每个函数是安卓系统函数还是应用开发人员自定义的本地函数. 由于逆向分析过程中系统函数名称通常保持不变, 而本地函数名称大部分会因为编译混淆而丢失, 因此本文只保留 API 调用序列中的安卓系统 API 接口函数.

(3) 为了向量化 API 调用序列特征, 直接将每个 API 调用函数名称的独热编码作为该 API 调用函数的初始特征向量, 从而得到待测安卓软件 z 的 API 调用序列特征 $\mathbf{x}_z^{\text{api}}$.

2.2.3 针对控制流图模态的特征提取

作为静态代码分析中非常重要的数据结构和分析工具, 控制流图常用来表示软件代码的逻辑结构和语义信息, 其具体定义如下所示.

定义 1. 控制流图. 任意一个软件代码的控制流图 $\mathcal{G}=(\mathcal{V},\mathcal{E})$ 是一个由节点集 $\mathcal{V}$ 和边集 $\mathcal{E}\subseteq\mathcal{V}\times\mathcal{V}$ 共同构成的有向图. 其中, 每个节点 $v\in\mathcal{V}$ 代表一个基本块, 即除了最后一条指令外不包含任何跳转指令的一组指令序列; 而每条有向边 $(v_1,v_2)\in\mathcal{E}$ 代表两个基本块之间的控制流路径, 即从 v_1 所代表的基本块跳转到 v_2 所代表的基本块.

由于控制流图的表征方法不仅与软件的程序语言和运行硬件环境无关, 而且可以准确地表达程序的内在执行逻辑, 因此近年来研究人员也提出大量的基于控制流图的恶意软件检测方法[58,59]. 具体而言, 首先, 利用逆向分析工具 Androguard 生成待测安卓软件 z 对应的控制流图, 图中每个节点是一个包含若干条指令的基本块. 由于上述控制流图信息无法输入到后面的深度学习模型中, 进一步对控制流图中每个节点 (即每个基本块) 进行向量化, 即

为每一个节点初始化一个特征向量. 如表 2 所示, 本文从 Dalvik 指令的操作码和操作数两方面统计控制流图中每个节点的数值统计特征, 最终得到向量化之后的控制流图特征 $\mathbf{x}_z^{\text{cfg}}=\left\{\{h_{v_i}^{(0)}\}_{i=0}^{i=n_z^{\text{cfg}}},\mathcal{E}_z\right\}$, 其中 $h_{v_i}^{(0)}$ 表示控制流图中节点 v_i 的初始化特征向量, n_z^{cfg} 表示控制流图中节点的总数量, $\{h_{v_i}^{(0)}\}_{i=0}^{i=n_z^{\text{cfg}}}$ 表示控制流图中所有节点初始化特征向量的集合, $\mathcal{E}_z$ 则表示控制流图中表示节点邻居关系的边集.

表 2 控制流图中每个节点的初始化特征

种类	基于操作码的节点特征	特征描述
操作码特征	move	数据操作指令总数量
	ret	返回或终止指令总数量
	monitor	锁指令总数量
	inst	类实例操作指令总数量
	array	数组操作指令总数量
	jmp	跳转指令总数量
	comp	比较指令总数量
	field	字段操作总数量
	call	方法调用指令总数量
	trans	数据转换指令总数量
	arith	数据运算指令总数量
	logic	逻辑运算指令总数量
操作数特征	string	字符串常量总数量
	const	数字常量总数量

2.3 基于深度学习模型的多模态特征学习

在提取到待测安卓软件 z 的多模态初始化特征信息 (灰度图模态特征 $\mathbf{x}_z^{\text{img}}$、API 调用序列模态特征 $\mathbf{x}_z^{\text{api}}$、控制流图模态特征 $\mathbf{x}_z^{\text{cfg}}$) 的基础上, 为了充分地学习待测安卓软件 z 中每一种模态特征的深层语义信息, 本文利用深度学习模型来学习相应多模态特征向量集 $\mathbf{H}_z=\left\{\vec{h}_z^{\text{img}},\vec{h}_z^{\text{api}},\vec{h}_z^{\text{cfg}}\right\}$, 其中 $\vec{h}_z^{\text{img}}$、$\vec{h}_z^{\text{api}}$ 和 $\vec{h}_z^{\text{cfg}}$ 分别代表所学习得到的灰度图模态特征向量、API 调用序列模态特征向量和控制流图模态特征向量. 下面将分别介绍利用深度学习模型学习灰度图、API 调用序列和控制流图这 3 种模态特征向量的具体实现方法.

2.3.1 基于卷积神经网络的灰度图特征学习方法

为了充分地学习待测安卓软件 z 在灰度图模态 $\mathbf{x}_z^{\text{img}}$ 上的深层语义信息, 本文采用图像识别领域最常用且最经典的深度残差网络 ResNet 来学习相应的灰度图特征向量 $\vec{h}_z^{\text{img}}$. 深度残差网络 ResNet 是由微软研究院提出的一种基于卷积神经网络的残差神经网络架构, 不仅通过增加深度学习网络深度来提高准确性, 而且可以缓解深度神经网络中增加网络深度带来的梯度消失和梯度爆炸问题, 在多项计算机视觉竞赛任务中取得优异的成绩[62].

此外, 安卓软件的代码规模会影响所提取灰度图的尺寸, 因此, 为了避免灰度图尺寸过大而导致的训练效率下降以及尺寸过小导致特征信息不足的问题, 如公式 (1) 所示, 本文首先利用 *ZeroPS* 函数将待测安卓软件 z 的灰度图模态特征 $\mathbf{x}_z^{\text{img}}$ 进行标准化补零和归一化处理, 并将最大维度设置为 512 (即 $max=512$), 从而得到维度为 $\mathbb{R}^{512\times512}$ 的标准灰度图形式 $\bar{\mathbf{x}}_z^{\text{img}}$.

$$\bar{\mathbf{x}}_z^{\text{img}}=ZeroPS(\mathbf{x}_z^{\text{img}};max=512)\in\mathbb{R}^{512\times512} \tag{1}$$

$$\vec{h}_z^{\text{img}}=ResNet34(\bar{\mathbf{x}}_z^{\text{img}};c=1,odim=d)\in\mathbb{R}^{d} \tag{2}$$

接着, 如公式 (2) 所示, 进一步将标准灰度图 $\bar{\mathbf{x}}_z^{\text{img}}$ 输入到相应的 ResNet34 网络中, 从而能够学习得到最终的灰度图特征向量 $\vec{h}_z^{\text{img}}$. 值得注意的是, 由于所提取的是灰度图而非彩色图, 因此将 ResNet34 网络中卷积层的通道数设置为 1 (即设置 $c=1$), 并进一步修改 ResNet34 网络中最后一层全连接层的输出维度为 d (即设置 $odim=d$), 从而使得学习得到的灰度图特征向量 $\vec{h}_z^{\text{img}}$ 的维度为 $\mathbb{R}^d$.

2.3.2 基于循环神经网络的 API 调用序列特征学习方法

同样地, 为了充分地学习待测安卓软件 z 在 API 调用序列特征 $\mathbf{x}_z^{\text{api}}$ 上的深层语义信息, 本文采用自然语言处理领域中处理序列数据最常用的循环神经网络来学习 API 调用序列特征向量 $\vec{h}_z^{\text{api}}$. 特别地, 作为一种特殊且应用范围最广的循环神经网络, 长短期记忆网络 (LSTM) 能够在一定程度上缓解传统循环神经网络处理长序列数据时存在的梯度消失和梯度爆炸问题, 例如, 长文本处理等[63]. 因此, 由于第 2.2.2 节中所提取的 API 调用序列一般比较长 (大约 3 500), 属于长序列数据, 本文因而采用 LSTM 来学习相应的 API 调用序列特征向量 $\vec{h}_z^{\text{api}}$.

具体而言, 首先, 如公式 (3) 所示, 根据安卓操作系统中所有常用的 API 调用的函数名称构建相应的 API 调用词典 $Vocab_{\text{api}}$, 并利用该 API 调用词典将待测安卓软件 z 的 API 调用序列特征 $\mathbf{x}_z^{\text{api}}$ 进行独热编码 (one-hot encoding), 从而得到初始化的 API 调用序列特征向量 $\bar{\mathbf{x}}_z^{\text{api}}$.

$$\bar{\mathbf{x}}_z^{\text{api}} = OneHotEncoding(\mathbf{x}_z^{\text{api}}, Vocab_{\text{api}}) \tag{3}$$

$$\vec{h}_z^{\text{api}} = LSTM(\bar{\mathbf{x}}_z^{\text{api}})_{[-1,\cdots]} \in \mathbb{R}^d \tag{4}$$

接着, 如公式 (4) 所示, 将初始化的 API 调用序列特征向量 $\bar{\mathbf{x}}_z^{\text{api}}$ 输入到相应的 LSTM 网络中, 从而学习到每个时间步的隐向量. 特别地, 为了保证最终的 API 调用序列特征向量 $\vec{h}_z^{\text{api}}$ 的特征维度与灰度图特征向量 $\vec{h}_z^{\text{img}}$ 的特征维度一致, 本文将 LSTM 网络中隐向量维度设置为 d, 并且将最后一个时间步的隐向量 $LSTM(\bar{\mathbf{x}}_z^{\text{api}})_{[-1,\cdots]}$ 作为最终学习得到的 API 调用序列特征向量 $\vec{h}_z^{\text{api}} \in \mathbb{R}^d$.

2.3.3 基于图神经网络的控制流图特征学习方法

近年来, 随着深度神经网络的不断发展, 图神经网络已经被证明可以成功地应用于各类需要处理图数据的相关任务中, 包括节点分类、图分类、图相似性计算等任务[64]. 为了学习待测安卓软件 z 的控制流图模态特征 $\mathbf{x}_z^{\text{cfg}}$ 的深层语义信息, 本文首先利用图神经网络 GraphSAGE[65]来逐层学习控制流图中每个节点丰富的特征表示, 即控制流图中每个节点的特征向量. 特别地, 假设 GraphSAGE 网络的层数是 T, 那么对控制流图中任意一个节点 v_i 而言, 第 $t \in [1,T]$ 层 GraphSAGE 网络生成的特征向量表示为 $h_{v_i}^{(t)}$. 具体而言, 如公式 (5) 所示, 第 t 层 GraphSAGE 网络会分别通过节点自传递函数 f_{node} 和邻居消息传递函数 f_{neighbor} 从上一层中该节点 v_i 本身的特征向量 $h_{v_i}^{(t-1)}$ 和所有邻居节点的特征向量 $\{h_u^{(t-1)}, u \in \mathcal{N}(v_i)\}$ 来共同学习和更新本层中该节点的特征向量, 即 $h_{v_i}^{(t)}$.

$$h_{v_i}^{(t)} = \sigma\left(f_{\text{node}}(h_{v_i}^{(t-1)}) + \sum_{u\in\mathcal{N}(v_i)} f_{\text{neighbor}}(h_u^{(t-1)})\right) = \sigma\left(W_1^{(t-1)} \cdot h_{v_i}^{(t-1)} + \frac{1}{|\mathcal{N}(v_i)|}\sum_{u\in\mathcal{N}(v_i)} W_2^{(t-1)} \cdot h_u^{(t-1)}\right) \tag{5}$$

其中, σ 表示任意一个激活函数, 例如线性整流激活函数 $ReLU(x) = \max(0,x)$; $\mathcal{N}(v_i)$ 代表控制流图中节点 v_i 的所有邻居节点的集合; $W_1^{(t_1-1)}$ 和 $W_2^{(t_1-1)}$ 分别是自传递函数 f_{node} 和邻居消息传递函数 f_{neighbor} 的模型参数. 特别地, 第 0 层每个节点的特征向量 $h_{v_i}^{(0)}$ 是在第 2.2.3 节中提取控制流图并初始化每个基本块节点而得到的.

在总共经过 T 层 GraphSAGE 网络之后, 对待测安卓软件 z 的控制流图而言, 可以学习到控制流图中所有节点的特征向量的集合, 即 $\{h_{v_i}^{(T)}\}_{i=1}^{i=n_z^{\text{cfg}}}$, 其中 n_z^{cfg} 代表待测安卓软件 z 的控制流图中所有节点的数量. 紧接着, 为了学习针对待测安卓软件 z 所提取控制流图的全局图向量, 如公式 (6) 所示, 采用一种基于最大池化操作的图聚合模型, 并在最大池化操作的前面增加一层全连接网络来增加全局图聚合模型的学习能力.

$$\vec{h}_z^{\text{cfg}} = MaxPooling\left(FC\left(\{h_{v_i}^{(T)}\}_{i=1}^{i=n_z^{\text{cfg}}}\right)\right) \in \mathbb{R}^d \tag{6}$$

其中, $MaxPooling$ 和 FC 分别表示最大池化操作和一层全连接网络. 为了保证所学习的控制流图特征向量 $\vec{h}_z^{\text{cfg}}$ 的维度与其他两种模态特征向量 ($\vec{h}_z^{\text{img}}$ 和 $\vec{h}_z^{\text{api}}$) 的维度一致, 本文将全连接网络 $FC(\cdot)$ 的输出维度设置为 d, 最终学习得到的控制流图特征向量 $\vec{h}_z^{\text{cfg}}$ 的维度为 $\mathbb{R}^d$.

2.4 基于鲁棒融合学习的恶意软件鲁棒检测

给定待测安卓软件 z, 本文中安卓恶意软件鲁棒检测任务的目标是, 不仅需要准确地识别善意软件和一般性恶意软件, 而且需要准确地识别攻击者生成的对抗性恶意软件, 即准确地预测 z 是否为恶意软件 $y_z \in \{0,1\}$, 其中

$y_z=0$ 代表善意软件, 而 $y_z=1$ 代表一般性恶意软件或者对抗性恶意软件. 为此, 在第 2.2 节提取待测安卓软件 z 的多模态特征 ($\mathbf{x}_z^{\text{img}}$、$\mathbf{x}_z^{\text{api}}$、$\mathbf{x}_z^{\text{cfg}}$) 和第 2.3 节学习到相应的多模态特征向量集 $\mathbf{H}_z=\{\vec{h}_z^{\text{img}},\vec{h}_z^{\text{api}},\vec{h}_z^{\text{cfg}}\}$ 的基础上, 本文采用面向多模态信息的特征融合学习方法, 全面融合 z 的多个模态信息, 进一步扩宽和补充安卓恶意软件的深层语义信息, 从而提升安卓恶意软件检测方法的准确性和鲁棒性.

本质上, 多模态信息融合学习[66]的核心是充分地利用样本中多个模态特征之间的互补性, 将多个模态的特征信息进行融合学习并进一步映射到一个统一的向量空间中, 从而为下游任务提供丰富且稳定的特征表示. 不失一般性地, 假设任一样本总共有 K 个模态特征向量 $\mathbf{H}=\{\vec{h}^{(k)}\}_{k=1}^{k=K}$, 那么标准的多模态信息融合学习方法可以表示成 $\mathrm{F}(\mathbf{H})=\Gamma\left(\{\vec{h}^{(k)}\}_{k=1}^{k=K}\right)$, 其中 Γ 代表某种融合方法, 例如, 特征向量拼接、加权求和、线性投影等.

然而, 尽管多模态信息融合学习方法已经在各类计算机视觉相关任务中表现出极出色的性能, 例如场景理解、情感分析、语音识别等[66], 但是相关研究发现这类方法在单个模态受到对抗攻击时也会引发多模态信息融合学习方法失效[67]. 因此, 受到计算机视觉领域中鲁棒融合学习相关工作的启发[67], 本文在安卓恶意软件鲁棒检测任务中引入面向多模态信息的鲁棒融合学习方法, 提出基于鲁棒融合学习的恶意软件检测方法 $F_{\text{robust}}(\mathbf{H})$, 不仅可以准确地识别善意软件和一般性恶意软件, 而且能够识别攻击者生成的对抗性恶意软件. 现有的多模态恶意软件检测模型主要是通过不同模态的特征冗余提供稳定性, 但受到扰动的特征被融合时对抗性扰动带来的影响仍然不可避免地引入最终的预测网络中. 针对现有工作的不足, 本文利用异类识别网络 Θ 来主动识别可能被扰动的恶意软件模态, 从而对该模态对应的特征进行加权处理, 弱化该扰动的模态对最终预测网络的影响, 有效地提高恶意软件检测的鲁棒性.

如图 2 和公式 (7) 所示, 本文所提出的基于鲁棒融合学习的恶意软件检测方法 $F_{\text{robust}}(\mathbf{H})$ 具体由异类识别网络 Θ 和鲁棒融合预测网络 $\tilde{\Gamma}$ 共同组成.

$$F_{\text{robust}}(\mathbf{H})=\tilde{\Gamma}(\mathbf{H};\Theta(\mathbf{H})) \tag{7}$$

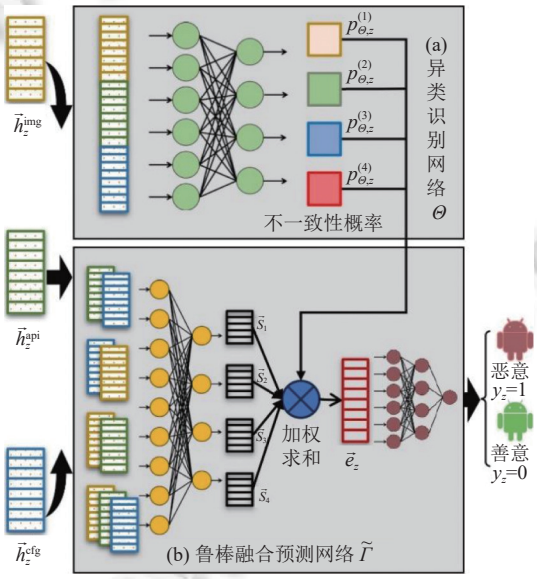

图 2 基于鲁棒融合学习的安卓恶意软件检测方法 F_{robust} 示意图

这表明, 针对任一样本的多模态特征向量 $\mathbf{H}=\{\vec{h}^{(k)}\}_{k=1}^{k=K}$, 异类识别网络 Θ 会预测每个模态特征可能受到对抗样本攻击的概率; 而鲁棒融合预测网络 $\tilde{\Gamma}$ 则会利用异类识别网络 Θ 预测的概率, 降低甚至消除受到对抗攻击的模态特征对最终恶意软件检测结果的影响, 从而提高其抵御对抗攻击的鲁棒性. 下面将在第 2.4.1 节和第 2.4.2 节分别介绍异类识别网络 Θ 和鲁棒融合预测网络 $\tilde{\Gamma}$.

2.4.1 异类识别网络

为了向恶意软件鲁棒检测任务提供一致性且不受对抗攻击干扰的多模态特征信息, 本文具体利用异类识别网络 Θ 充分地学习和比较来自不同模态特征之间的异同点, 从而识别多模态特征信息中存在的不一致模态甚至是受到对抗攻击干扰的模态. 具体而言, 假设样本总共有 K 个模态特征向量 $\mathbf{H}=\{\vec{h}^{(k)}\}_{k=1}^{k=K}$, 异类识别网络 Θ 的输出是一个维度为 $K+1$ 的预测向量 $\Theta(\mathbf{H})=\{p_{\theta}^{(k)}\}_{k=1}^{k=K+1}$, 并且其中所有 $K+1$ 个预测概率值相加等于 1. 特别地, 当 $k\in\{1,2,\dots,K\}$ 时, $p_{\theta}^{(k)}$ 代表第 k 个模态特征 $\vec{h}^{(k)}$ 与其他模态特征不一致的概率, 或者表示其受到对抗攻击扰动的概率; 而当 $k=K+1$ 时, $p_{\theta}^{(k+1)}$ 则代表没有任何一个模态特征受到对抗攻击扰动的概率.

如图 2(a) 和公式 (8) 所示, 对本文所关注的安卓恶意软件鲁棒检测任务而言, 在学习到待测安卓软件 z 中 $K=3$ 个模态特征向量 $\mathbf{H}_z=\{\vec{h}_z^{\text{img}},\vec{h}_z^{\text{api}},\vec{h}_z^{\text{cfg}}\}$ 的基础上, 本文所提出的异类识别网络 Θ 首先对 $K=3$ 个模态特征向量进行向量拼接, 然后将拼接后的特征向量输入到单层的全连接网络中, 从而预测待测安卓软件 z 中每个模态可能受到对抗攻击扰动的概率 $\Theta(\mathbf{H}_z)=\{p_{\theta,z}^{(1)},p_{\theta,z}^{(2)},p_{\theta,z}^{(3)},p_{\theta,z}^{(4)}\}$.

$$\Theta(\mathbf{H}_z)=\Theta\left(\{\vec{h}_z^{\text{img}},\vec{h}_z^{\text{api}},\vec{h}_z^{\text{cfg}}\}\right)=FC\left(Concat\left(\{\vec{h}_z^{\text{img}},\vec{h}_z^{\text{api}},\vec{h}_z^{\text{cfg}}\}\right)\right) \tag{8}$$

其中, $Concat$ 是向量拼接函数, 而 FC 表示一层全连接神经网络, 其输入维度是 $K\times d=3d$, 而输出维度是 $K+1=4$.

2.4.2 鲁棒融合预测网络

如图 2(b) 所示, 首先, 为了充分利用异类识别网络所预测的不一致概率向量, 本文利用基于混合专家系统的集成学习模型[67,68]来针对不同模态特征向量 $\mathbf{H}$ 学习相应的特征融合向量 $\mathbf{S}(\mathbf{H})$. 如公式 (9) 所示, 当 $k\in\{1,2,\dots,K\}$ 时, 特征融合向量 $\vec{s}_k(\mathbf{H})$ 利用向量拼接函数 $Concat$ 和全连接神经网络 FC 融合了所有模态特征向量集 $\mathbf{H}$ 中除了第 k 个模态特征向量之外的所有模态特征向量 $\mathbf{H}_{-k}$; 而当 $k=K+1$ 时, 特征融合向量 $\vec{s}_{K+1}(\mathbf{H})$ 则利用 $Concat$ 和 FC 融合了所有模态特征向量 $\mathbf{H}$.

$$\mathbf{S}(\mathbf{H})=\begin{cases}\vec{s}_k(\mathbf{H}_{-k})=FC(Concat(\mathbf{H}_{-k})),\ \forall k\in\{1,2,\dots,K\}\\ \vec{s}_{K+1}(\mathbf{H})=FC(Concat(\mathbf{H}))\end{cases} \tag{9}$$

其次, 针对待测安卓软件 z, 如公式 (10) 所示, 进一步将异类识别网络输出的结果 $\Theta(\mathbf{H}_z)=\{p_{\theta,z}^{(1)},p_{\theta,z}^{(2)},p_{\theta,z}^{(3)},p_{\theta,z}^{(4)}\}$ 作为权重, 对特征融合向量 $\mathbf{S}(\mathbf{H}_z)$ 进行加权求和, 从而得到不受对抗攻击干扰的鲁棒融合特征向量 $\vec{e}_z$.

$$\vec{e}_z=\mathbf{S}(\mathbf{H}_z)\Theta(\mathbf{H}_z)=\sum_{k=1}^{k=K+1}\vec{s}_k(\mathbf{H}_z)p_{\theta}^{(k)} \tag{10}$$

特别地, 假设第 $k\in\{1,2,\dots,K\}$ 个模态特征受到对抗攻击的扰动, 那么异类识别网络判断该模态特征与其他模态不一致的概率较大, 即 $p_{\theta}^{(k)}$ 较大, 而特征融合向量 $\vec{s}_k(\mathbf{H}_{-k})$ 中排除了第 k 个模态特征向量, 因此可以降低甚至消除受到对抗攻击的模态特征对鲁棒融合特征向量 $\vec{e}_z$ 的影响.

紧接着, 为了预测待测安卓软件 z 为恶意软件的概率 p_z, 如公式 (11) 所示, 首先利用 3 个标准的全连接神经网络将得到的鲁棒融合特征向量 $\vec{e}_z$ 逐步地线性投影到维度为 1 的标量上, 然后利用标准的 $Sigmoid$ 函数将预测为恶意软件的概率 p_z 限制在 $(0,1)$ 的范围内.

$$p_z=Sigmoid(FC_3(FC_2(FC_1(\vec{e}_z)))) \tag{11}$$

$$y_z=\begin{cases}1, & \text{if } p_z>\zeta\\ 0, & \text{otherwise}\end{cases} \tag{12}$$

最后, 如公式 (12) 所示, 假设最终的恶意软件鲁棒检测模型的二分类超参数阈值为 $\zeta\in(0,1)$, 那么可以预测给定待测安卓软件 z 的二分类标签 $y_z\in\{0,1\}$, 即是否为恶意软件, 其中 $y_z=0$ 代表 z 为善意软件, 而 $y_z=1$ 代表 z 为恶意软件, 既包括一般恶意软件, 也包含由对抗攻击生成的对抗性恶意软件.

2.5 模型训练

如公式 (13) 所示, 本文通过最小化二元交叉熵 (binary cross entropy) 损失函数 $\mathcal{L}$ 来训练基于多模态融合学习的安卓恶意软件鲁棒检测模型.

$$\mathcal{L}=-\frac{1}{N}\sum_{i=1}^{i=N}\left(\hat{y}_{z_i}\log(p_{z_i})+(1-\hat{y}_{z_i})\log(1-p_{z_i})\right) \tag{13}$$

其中, N 为训练集中所有安卓软件的总数量, p_{z_i} 代表第 i 个安卓软件被预测为恶意软件的概率, 而 $\hat{y}_{z_i}$ 代表第 i 个安卓软件的真实标签值, 即 0 代表善意软件, 1 代表恶意软件.

具体而言, 为了充分地训练本文所提出的中多模态特征学习中的深度学习子模型, 本文首先单独训练只针对单个模态的子模型参数, 即灰度图模态特征学习模型 RMDroid(img)、API 调用序列模态特征学习模型 RMDroid(api)、控制流图模态特征学习模型 RMDroid(cfg); 然后, 将针对单个模态的子模型参数对完整的鲁棒检测模型进行初始化, 并进一步利用公式 (13) 训练包含 F_{robust} 在内的整个基于多模态融合学习的安卓恶意软件鲁棒检测模型 RMDroid.

3 实验评估

首先在第 3.1 节中对本文实验评估中所用的实验数据、评价指标、对比的基线检测方法、实验环境及参数设置等实验设置进行详细介绍, 然后在第 3.2 节中对相关实验结果进行分析.

3.1 实验设置

3.1.1 实验数据

AndroZoo 是由 Allix 等人[69]在 2016 年开始从谷歌官方 Google Play 应用程序市场收集并构建的包含大量安卓软件的数据集, 每个安卓软件不仅包括应用程序 APK 本身, 而且还包括相应的哈希值、APK 大小、被 VirusTotal 杀毒引擎检测的结果列表, 被 VirusTotal 杀毒引擎检测的日期等. 由于 AndroZoo 数据集不仅数据规模巨大, 而且包含权威杀毒引擎 VirusTotal 的检测结果, 因此 AndroZoo 及其相关数据集已经被广泛地用于评估安卓恶意软件检测任务中[17,18,70,71].

为了充分地评估本文所提出的 RMDroid 检测有效性和鲁棒性, 本文采用了与经典安卓恶意软件检测方法 MalScan[18]和最新安卓恶意软件对抗攻击 HRAT[17]相同的实验数据, 并且这些实验数据全部来源于 AndroZoo 数据集, 记为 $\mathbb{D}$. 与此同时, 考虑到恶意软件检测方法的有效性评估存在概念漂移 (concept drift) 的问题[72], 即随着时间推移, 之前训练好的恶意软件检测方法很可能无法检测最新出现的恶意软件. 因此, 为了充分对比并进一步消除安卓软件的时间对有效性和鲁棒性的影响, 本文按照如下两种不同的数据划分方式对实验数据 $\mathbb{D}$ 进行划分, 从而分别形成随机划分的数据集 $\mathbb{D}_{\text{rand}}$ 和时间划分的数据集 $\mathbb{D}_{\text{time}}$, 两者具体的数据量统计如表 3 所示.

表 3 实验数据集的统计信息

数据集	类型	训练集	验证集	测试集	总量
$\mathbb{D}_{\text{rand}}$	恶意软件	7093	1424	5674	14191
	善意软件	7129	1433	5703	14265
	总量	14222	2857	11377	28456
$\mathbb{D}_{\text{time}}$	恶意软件	7668	1307	5216	14191
	善意软件	7409	1372	5484	14265
	总量	15077	2679	10700	28456

● 随机划分的数据集 $\mathbb{D}_{\text{rand}}$: 针对数据集 $\mathbb{D}$ 中 2011–2018 年时间窗口内所有安卓软件, 按照训练集、验证集和测试集为 5:1:4 的划分比例, 对 $\mathbb{D}$ 中的恶意软件和善意软件进行随机划分, 从而得到随机划分的数据集 $\mathbb{D}_{\text{rand}}$. 由于 $\mathbb{D}_{\text{rand}}$ 中训练集的时间窗口和验证集/测试集完全重合, 因此该数据集划分方式并不符合实际场景中训练阶段数据与部署应用后测试样本数据分布不同的现实情况.

● 时间划分的数据集 $\mathbb{D}_{\text{time}}$: 针对数据集 $\mathbb{D}$ 中 2011–2018 年时间窗口内所有安卓应用软件, 进一步按照时间窗口划分, 将训练集限制在 2011–2014 年的时间窗口内, 将验证集/测试集限制在 2015–2018 年的时间窗口内, 从而得到时间划分的数据集 $\mathbb{D}_{\text{time}}$. 由于 $\mathbb{D}_{\text{time}}$ 中训练集的时间窗口全部小于验证集/测试集的时间窗口, 很大程度上符合

实际场景中安卓恶意软件检测训练数据与测试数据的分布情况, 因此可以用来充分地评估安卓恶意软件检测方法中时间迁移对检测有效性的影响.

3.1.2 评价指标

为了全面地评估本文所提出的安卓恶意软件鲁棒检测方法, 本文主要从恶意软件检测的有效性和鲁棒性两大类指标进行实验评估.

(1) 有效性指标: 恶意软件检测方法的有效性通常指的是能够准确地分类出善意软件和一般恶意软件的能力. 由于恶意软件检测方法本质上可以看作一种特殊的二分类模型, 本文主要采用二分类模型的评价指标, 即 AUC、*TPR*、*FPR*、*bACC*, 来评估所提出检测方法和基线检测方法在同一数据集上的有效性. 具体而言, AUC (area under the ROC curve) 指的是接受者操作特性 ROC 曲线下的面积, AUC 值越大, 说明恶意软件检测方法的有效性越高. 此外, 本文进一步评估在假阳率 *FPR* (false positive rate) 相同的情况下本文所提出检测方法和基线检测方法的真阳率 *TPR* (true positive rate) 和平衡准确率 *bACC* (balanced accuracy), 具体计算公式如下所示:

$$FPR = \frac{fp}{fp + tn} \tag{14}$$

$$TPR = \frac{tp}{tp + fn} \tag{15}$$

$$bACC = \frac{TPR + TNR}{2}, \quad TNR = \frac{tn}{fp + tn} \tag{16}$$

其中, tp、fp、tn、fn 分别表示二分类模型中真阳性、假阳性、真阴性、假阴性的数量.

(2) 鲁棒性指标: 在对抗攻击环境下, 恶意软件检测方法的鲁棒性指的是该方法可以抵御对抗样本攻击的能力, 即是否能够正确检测出对抗样本攻击方法所生成的对抗性恶意软件. 特别地, 对于实际的恶意软件检测方法以及杀毒软件而言, 攻击者关注的核心是误导其将恶意软件错误地分类成善意软件, 而不是将善意软件错误地分类成恶意软件. 因此, 如公式 (17) 所示, 本文首先计算恶意软件检测方法被攻击者生成的对抗性恶意软件成功绕过的比例, 即攻击成功率 (attack success rate, *ASR*), 然后利用公式 (18) 计算该恶意软件检测方法针对该对抗攻击的鲁棒性 *Robust*. 直观上, 对任一恶意软件检测方法而言, 攻击者生成的对抗性恶意软件越容易绕过该方法的检测, 即攻击成功率 *ASR* 则越高, 那么相应的该方法的鲁棒性 *Robust* 则越低.

$$ASR = \frac{\sum_{i=1}^{M} \left| y_{z_i} = 1 \wedge y_{z_i^*} = 0 \right|}{\sum_{i=1}^{M} \left| y_{z_i} = 1 \right|} \tag{17}$$

$$Robust = 1 - ASR \tag{18}$$

其中, M 表示所有鲁棒性评估的恶意软件样本数量; z_i 表示第 i 个恶意软件样本, 而 z_i^* 则表示 z_i 被攻击后生成的对抗性恶意软件; y_{z_i} 表示恶意软件检测方法预测第 i 个样本的标签值 (1 代表恶意软件, 0 代表善意软件).

3.1.3 对比的基线检测方法

为了验证本文所提出安卓恶意软件鲁棒检测方法的有效性和鲁棒性, 本文将 RMDroid 与当前 3 种最典型的安卓恶意软件检测方法进行对比, 下面简单介绍 3 种进行对比的基线检测方法.

(1) IMCFN[57]是当前最具代表性的基于图像的安卓恶意软件检测方法. 具体而言, IMCFN 首先利用一种特殊的图像生成算法将待测安卓软件转换成三维的彩色图像, 然后通过微调常用的基于卷积神经网络的图像分类模型, 例如 VGG、ResNet、Inception V3 等, 来达到检测安卓恶意软件的目的.

(2) MaMaDroid[19]是一种基于应用程序行为的安卓恶意软件检测方法. MaMaDroid 首先会提取安卓软件的抽象 API 调用序列, 并构建相应的行为模型, 然后从该行为模型中提取相应的语义特征作为恶意软件检测分类的依据, 最后利用随机森林或者 SVM 等分类方法进行安卓恶意软件检测.

(3) MalScan[18]是一种基于 API 调用图的安卓恶意软件快速检测方法. 具体而言, MalScan 首先提取安卓软件的 API 调用图, 并利用社交网络分析方法挖掘其中敏感 API 调用的中心性, 最后依次将其作为该安卓软件的语义特征来进行恶意软件检测.

3.1.4 对抗性恶意软件生成方法

为了充分地评估鲁棒性, 本文采用当前最先进的 3 种对抗攻击生成相应的对抗性安卓恶意软件.

(1) COPYCAT[73]是当前最具代表性的针对 Windows PE 恶意软件中图像特征的对抗攻击方法. COPYCAT 首先提取 Windows PE 恶意软件的图像特征, 然后利用图像领域的对抗攻击方法生成对抗图像样本, 接着将对抗图像样本附加到原始图像样本末尾, 再逐字节还原为二进制文件从而生成相应的对抗性 PE 恶意软件. 由于 COPYCAT 原本针对的是 Windows PE 恶意软件, 因此本文进一步将 COPYCAT 适配到可以攻击安卓恶意软件, 并采用图像领域当前最先进的 C&W[38]作为核心的对抗攻击方法.

(2) GADGET[74]是一种专门针对恶意软件中 API 调用序列特征而设计的对抗攻击方法, 该攻击方法在梯度的引导下主要通过添加额外的 API 调用来生成相应的对抗性恶意软件.

(3) HRAT[17]是 2021 年发表于 ACM CCS 上的一种专门针对恶意软件中 API 调用图特征而设计的对抗攻击方法, 也是当前针对安卓恶意软件最先进的对抗攻击方法. 具体而言, HRAT 设计了 4 种 (插入节点、删除节点、添加边、重新布线) 针对 API 调用图的操作方式, 并采用强化学习作为启发式优化算法, 从而生成相应的对抗性安卓恶意软件.

3.1.5 实验环境及参数设置

在实验环境方面, 本文中所有实验均在一台使用 Ubuntu 20.04.1 LTS 操作系统的服务器上进行, 处理器为 Intel(R) Xeon(R) Gold 5218R, 内存为 64 GB, GPU 为 4 张 NVIDIA GeForce RTX 3090. 本文中代码实现所用的编程语言为 Python 3.8.16, 其中所有神经网络的代码实现均采用 PyTorch 2.0.1 和 PyTorch_Geometric 2.3.1 深度学习框架. 特别地, 本文利用 Androguard 逆向工具[75]对所有的实验数据集即安卓软件进行预处理, 从而得到相应的 API 调用序列信息和控制流图信息.

对于 IMCFN 而言, 在数据预处理阶段, 本文将安卓软件的 Dalvik 字节码以 8 bit 为单位连续编码为 0–255 之间灰度值, 并根据表 4 中的规则确定灰度图的宽度, 将灰度值序列按照固定宽度排列为二维灰度矩阵, 并在不满足宽度的最后一行填充 0 构造完整的灰度图像. 接着使用预训练的 VGG16 图像分类模型提取输入图像的特征, 然后通过 1 层卷积层和 3 层全连接层逐层提取特征进行最终的恶意性预测. 本文使用初始学习率 1E–4、权重衰减系数 1E–4 的 Adam 优化器训练 IMCFN, 训练 epoch 为 20.

表 4 灰度图像宽度与 Dalvik 字节码文件大小的对应关系

Dalvik文件大小 (KB)	图像宽度 (pixel)
<5	64
5–10	128
10–20	256
20–50	512
50–100	1 024
>100	2 048

对于 MaMaDroid 而言, 本文遵循原文的方法流程来提取安卓软件的调用图, 并从家族 (family) 和包 (package) 两个粒度抽象调用关系, 构建 family 和 package 粒度的特征, 使用决策树数量为 512 的随机森林进行拟合学习, 并在实验中评估对比不同粒度特征下 MaMaDroid 的有效性. 对于 MalScan 而言, 本文同样遵循原文的方法, 首先提取安卓软件的调用图特征, 然后计算调用图的 Harmonic 中心性, 最后基于预定义的敏感 API 列表获取敏感 API 的中心性分数, 所有敏感 API 的中心性分数构成安卓软件的特征向量. 模型选择方面, 本文首先使用少量训练集和验证集对 MalScan 所使用的 KNN-1、KNN-3 和随机森林这 3 种模型进行预实验评估模型的性能, 最终本文选择效果最稳定、性能最高的随机森林, 决策树数量设置为 100.

对于本文的 RMDroid 而言, 在参数设置方面, 默认情况下, 本文所实现及评估的各个子模态 (灰度图、API 调用序列和控制流图) 所学习的特征向量的维度 d 均设置成 300, 即 $d = 300$, 并且训练周期 epoch 设为 20, mini-batch 设为 16, 优化算法选择 AdamW 优化器, 默认初始学习率为 1E–5, 默认权重衰减系数为 0.01. 其中, RMDroid 使用

与 IMCFN 相同的预处理方法生成安卓软件的图像模态数据用于训练和测试. 特别地, 为了更好地且针对性地训练中各个子模态的神经网络, 对于第 2.3.1 节中基于卷积神经网络的灰度图特征学习方法而言, 具体采用典型的 ResNet34 网络[62], AdamW 优化器的初始学习率为 1E–5; 对于第 2.3.2 节中基于 LSTM 的 API 调用序列特征学习方法而言, LSTM 模型的隐藏特征维度设置为 200, AdamW 优化器的初始学习率为 0.01; 对于第 2.3.3 节中基于图神经网络的控制流图特征学习方法而言, GraphSAGE 网络的层数 T 为 2 且输出维度均为 300, 并且在每层 GraphSAGE 网络之后都添加一层概率为 0.3 的 Dropout 网络以及 *ReLU* 激活函数避免模型过拟合, 此外, 其 AdamW 优化器的初始学习率为 0.001.

对于第 3.1.4 节中的对抗性攻击基线方法, 本文选择图像领域经典的对抗性攻击方法 FGSM、PGD 以及 Carlini-Wagner 作为 COPYCAT 中集成的对抗样本生成方法. 其中 FGSM 的扰动参数 ϵ 设置为 0.05; PGD 的扰动参数 ϵ 设置为 0.033 3, 最大迭代次数为 250 次, 学习率设置为 0.01; Carlini-Wagner 的迭代次数设置为 100, 学习率设置为 0.1. GADGET 攻击遵循论文设置, 采用大小为 140 的滑动窗口, 短序列用 0 填充, 长序列按窗口分割后逐段处理. 每次窗口处理尝试插入 API 的最大次数为 250 次, 插入时保持窗口长度不变, 将末尾 API 移至剩余序列. 对于 HRAT 而言, 本文使用具有 3 个隐藏层的全连接神经网络和 *ReLU* 激活函数作为 DQN 代理近似 $Q(s,a)$ 函数进行行为决策, 隐藏层神经元数量自底向上分别为 1 000 和 50, 输入为状态向量, 输出层包含 4 个神经元, 对应 HRAT 所设计的 4 种动作的 Q 值. DQN 使用学习率为 1E–3 的 Adam 优化器进行训练, 折扣因子 γ 设置为 0.9. 此外, 本文采用大小为 32 的经验回放缓冲区, 以及批大小为 16 的 mini-batch 进行训练, 目标网络每 100 步更新一次. Episode 设置为 30, 每个 episode 的最大步数设置为 500.

3.2 实验结果与分析

为了充分地评估本文所提出 RMDroid 在安卓恶意软件鲁棒检测任务上的实际效果, 本节首先通过对比 RMDroid 与 3 个基线检测方法在同一数据集上的有效性指标, 对在恶意软件上的有效性进行系统的分析; 然后, 使用相同设置分析 RMDroid 的鲁棒性.

3.2.1 有效性实验结果与分析

(1) 对比实验

为了评估本文所提出 RMDroid 能够准确地检测出一般性恶意软件的能力, 即有效性实验, 本文选取 IMCFN、MaMaDroid 与 MalScan 这 3 种最新的安卓恶意软件检测方法作为基线, 与本方法一起作为实验对象进行对比实验. 具体而言, IMCFN 是当前最具代表性的基于图像的安卓恶意软件检测方法, 而 MaMaDroid 和 MalScan 是当前最先进的基于 API 调用信息的安卓恶意软件检测方法. 特别地, 由于 MaMaDroid 中所采用的 API 调用信息的特征粒度对安卓恶意软件检测的性能影响较大, 因此本文进一步训练两个 MaMaDroid 变体模型, 即 MaMaDroid(f) 和 MaMaDroid(p), 分别代表利用家族 (family) 信息和包 (package) 信息的 MaMaDroid 检测方法.

综上, 为了公平地对比与所有基线方法的有效性, 本文采用第 3.1.2 节中定义的 AUC、*TPR* 与 *bACC* 这 3 类有效性评价指标. 特别地, 由于只有在保证 *FPR* 相同的情况下比较 *TPR* 和 *bACC* 才有意义, 因此对于每个待评估的安卓恶意软件检测方法, 本文在保证 *FPR* 分别等于 1% 和 0.1% 的情况下计算相应的 *TPR* 和 *bACC*. 显然地, *TPR* 和 *bACC* 的值越高, 说明相应安卓恶意软件检测方法的有效性越好. 表 5 展示了与所有基线方法分别在 $\mathbb{D}_{\text{rand}}$ 和 $\mathbb{D}_{\text{time}}$ 两项数据集上的有效性实验结果.

首先, 分析所有基线检测方法之间相互对比的实验结果. 通过对表 5 中包括 IMCFN、MaMaDroid(f)、MaMaDroid(p) 和 MalScan 在内所有基线安卓恶意软件检测方法的实验结果进行对比分析. 本文发现在 $\mathbb{D}_{\text{rand}}$ 数据集上基于 API 调用特征的 MaMaDroid(p) 的所有有效性评价指标全部优于其他基线检测方法. 然而, 在更符合实际场景的数据集 $\mathbb{D}_{\text{time}}$ 上, 基于图像特征的 IMCFN 的有效性评价指标明显高于其他基线检测方法. 本文推测出现该结果可能的原因之一是: 随着时间推移, 恶意软件内调用的 API 函数也会随着安卓版本更迭而变化 (例如新的 API 出现, 旧的 API 被弃用), 因此会影响基于 API 调用特征的 MaMaDroid 检测方法检测 $\mathbb{D}_{\text{time}}$ 数据集中的测试集中新出现的恶意软件; 然而, 由于 IMCFN 采用的是建立在二进制字节码上的图像特征, 受到 API 函数更迭的影响

不大, 因此在 $\mathbb{D}_{rime}$ 数据集上表现出更好的检测有效性, 这一点也可以通过对比后续消融实验中数据集 RMDroid 与单模态子模型中, $\mathbb{D}_{time}$ 上 RMDroid(img) 和 RMDroid(api) 的有效性实验结果得到进一步验证.

表 5　RMDroid 与基线方法的有效性实验结果 (%)

数据集	检测方法	AUC	*FPR*=1%		*FPR*=0.1%	
			TPR	*bACC*	*TPR*	*bACC*
$\mathbb{D}_{rand}$	IMCFN	97.78	63.80	81.40	29.70	64.80
	MaMaDroid(f)	97.53	73.72	86.37	51.29	75.61
	MaMaDroid(p)	98.18	81.26	90.13	60.14	80.03
	MalScan	97.40	71.48	85.24	46.40	73.16
	RMDroid	**99.52**	**93.55**	**96.27**	**75.84**	**87.87**
$\mathbb{D}_{time}$	IMCFN	80.96	31.23	65.12	13.09	56.55
	MaMaDroid(f)	76.23	21.66	60.35	5.61	52.76
	MaMaDroid(p)	68.30	24.48	61.75	9.89	54.90
	MalScan	66.54	17.66	58.37	6.98	53.44
	RMDroid	**83.96**	**41.76**	**70.38**	**23.10**	**61.51**

注: 加粗数据表示结果最佳

接着将 RMDroid 与所有基线检测方法的有效性实验结果进行对比分析. 从表 5 中可以观察到, 尽管在更符合实际场景的数据集 $\mathbb{D}_{time}$ 上的有效性评价指标有所降低, 但是分别在 $\mathbb{D}_{rand}$ 和 $\mathbb{D}_{time}$ 两个数据集上的所有 5 个有效性评价指标都远高于 4 个基线检测方法. 具体而言, 在 $\mathbb{D}_{rand}$ 数据集上的 AUC 值为 99.52%, 在 *FPR*=1% 时相应 *TPR* 值和 *bACC* 值分别为 93.55% 和 96.27%, 表现出最优且稳定的有效性实验结果. 即使对于更具挑战的 *FPR*=0.1% 而言, *TPR* 值和 *bACC* 值也分别高达 75.84% 和 87.87%, 比最好的基线模型 MaMaDroid(p) 的 *TPR* 值和 *bACC* 值还要高出 15.70% 和 7.84%. 同样地, 在 $\mathbb{D}_{rand}$ 数据集上, 即使 RMDroid 的 *TPR* 值和 *bACC* 值有所下降, 但是仍然远高于最好的基线模型 IMCFN 的 *TPR* 值和 *bACC* 值. 上述的实验结果与分析都表明, 本文所提出的安卓恶意软件鲁棒检测方法在两个数据集上都表现出比所有基线检测方法更优的有效性.

为了进一步评估 RMDroid 和基线方法在最新安卓恶意软件数据集上的检测性能, 对比不同方法的有效性随着时间衰减的幅度, 本文进一步限定时间窗口为 2024 年 1 月–2025 年 6 月, 并从 AndroZoo 数据集[69]中随机采集了 1 650 个良性安卓软件和 1 678 个恶意安卓软件进行有效性评估, 记为新的测试数据集 $\mathbb{D}_{new}$. 本文直接利用原来在 $\mathbb{D}_{rand}$ 和 $\mathbb{D}_{time}$ 数据集上训练的恶意软件检测模型在 $\mathbb{D}_{new}$ 进行测试, 测试结果如表 6 所示.

表 6　不同检测方法在新数据集 $\mathbb{D}_{new}$ 上的检测性能 (%)

数据集	检测方法	AUC	*FPR*=1%		*FPR*=0.1%	
			TPR	*bACC*	*TPR*	*bACC*
训练: $\mathbb{D}_{rand}$ 测试: $\mathbb{D}_{new}$	IMCFN	71.59	18.89	59.25	0.00	50.00
	MaMaDroid(f)	58.59	12.90	55.98	11.47	55.69
	MaMaDroid(p)	25.21	1.57	50.31	0.00	50.00
	MalScan	**91.17**	12.87	56.37	0.83	50.42
	RMDroid	<u>81.03</u>	**24.55**	**61.85**	**20.41**	**60.21**
训练: $\mathbb{D}_{time}$ 测试: $\mathbb{D}_{new}$	IMCFN	67.36	**15.37**	**57.30**	0.00	50.00
	MaMaDroid(f)	39.74	3.94	51.50	1.43	50.67
	MaMaDroid(p)	60.05	11.64	55.35	1.26	50.59
	MalScan	**90.85**	1.72	50.74	0.49	50.25
	RMDroid	<u>72.13</u>	<u>14.27</u>	<u>56.69</u>	**6.66**	**53.33**

注: 加粗数据表示结果最佳, 下划线表示结果次优

通过对比不同检测方法的有效性指标, 本文观察到 MalScan 在新数据集 $\mathbb{D}_{new}$ 上的 AUC 高于其他方法, 这表明 MalScan 在所有可能的阈值下整体性能较好. 然而, MalScan 的其他 4 项有效性指标 *TPR* 和 *bACC* 较低, 这反映出其无法在较低的 *FPR* 条件下达到较高的 *TPR*. 对比在 $\mathbb{D}_{rand}$ 上训练的模型的测试结果, 本文发现 RMDroid 在

FPR=1% 和 *FPR*=0.1% 时, *TPR* 分别达到 24.55% 和 20.41%, 显著优于其他基线模型. 然而, 在 $\mathbb{D}_{time}$ 上 RMDroid 在 *FPR*=1% 时的 *TPR* 和 *bACC* 略低于 IMCFN, 但在更加严格的 *FPR*=0.1% 下 RMDroid 的检测性能优于基线方法. 因此, RMDroid 的整体检测性能最好, 这表明 RMDroid 在适应恶意样本演化方面具有更强的泛化能力.

(2) 消融实验

首先, 为了充分地评估本文所提出 RMDroid 中各个模态特征 (即灰度图、API 调用序列与控制流图共 3 种模态特征) 对有效性实验结果的影响, 本文进一步设计并实现 3 个相应的仅包含单模态特征的子模型, 即 RMDroid(img)、RMDroid(api) 与 RMDroid(cfg). 具体而言, RMDroid(img) 单模态子模型中只利用灰度图模态特征, 不包含其他模态特征, 因而也不包含相应的异类识别网络和鲁棒融合预测网络, 取而代之的是包含两层全连接网络和 *Sigmoid* 激活函数的预测网络. 以此类推, 本文同样可以构建另外两个单模态子模型 RMDroid(api) 和 RMDroid(cfg).

表 7 显示了 RMDroid 及其 3 个单模态子模型分别在 $\mathbb{D}_{rand}$ 和 $\mathbb{D}_{time}$ 两项数据集上的有效性实验结果. 一方面, 通过对表 7 中包括 RMDroid(img)、RMDroid(api) 与 RMDroid(cfg) 在内的 3 个模型变体的实验结果进行对比分析, 可以观察到, 在 $\mathbb{D}_{rand}$ 数据集上 RMDroid(cfg) 的所有有效性评价指标是最高的, 然而在 $\mathbb{D}_{time}$ 数据集上, RMDroid(img) 的 AUC 值以及当 *FPR*=0.1% 时相应 *TPR* 值和 *bACC* 值都略高于 RMDroid(cfg). 这表明在安卓恶意软件检测任务中, 没有任何一种模态特征是完全优于其他模态特征的, 每种模态特征都有自己的优点和缺点. 另一方面, 如果将完整的 RMDroid 与 3 个模型变体的实验结果进行对比分析, 可以观察到, 在 $\mathbb{D}_{rand}$ 和 $\mathbb{D}_{time}$ 两个数据集上 RMDroid 的所有 5 个有效性评价指标都显著地高于 RMDroid(img)、RMDroid(api) 与 RMDroid(cfg). 基于上述实验结果与分析, 本文发现, 同时利用并学习安卓恶意软件的灰度图、API 调用序列和控制流图这 3 种模态特征可以显著地提高安卓恶意软件检测方法的有效性, 因此进一步证明了本文所提出多模态特征融合学习的重要性.

表 7 RMDroid 与单模态子模型的有效性实验结果 (%)

数据集	检测方法	AUC	*FPR*=1%		*FPR*=0.1%	
			TPR	*bACC*	*TPR*	*bACC*
$\mathbb{D}_{rand}$	RMDroid(img)	95.63	60.22	79.61	17.50	58.71
	RMDroid(api)	94.83	59.32	79.16	15.42	57.67
	RMDroid(cfg)	98.60	89.27	94.13	66.83	83.37
	RMDroid	**99.52**	**93.55**	**96.27**	**75.84**	**87.87**
$\mathbb{D}_{time}$	RMDroid(img)	77.84	33.40	66.21	12.54	56.22
	RMDroid(api)	73.76	15.84	57.44	0.40	50.19
	RMDroid(cfg)	75.71	35.53	67.27	9.70	54.80
	RMDroid	**83.96**	**41.76**	**70.38**	**23.10**	**61.51**

注: 加粗数据表示结果最佳

其次, 为了充分评估本文所提出的 RMDroid 方法中各模态学习的特征向量维度 d 对有效性实验结果的影响, 本文在 $\mathbb{D}_{rand}$ 和 $\mathbb{D}_{time}$ 两个数据集上将 RMDroid 的特征向量维度 d 分别设置为 200、250、300、350 和 400 进行训练和评估, 从而详细地对比 RMDroid 在使用不同特征向量维度时对应的恶意软件检测性能, 实验结果如表 8 所示. 从表 8 中观察到, 整体上 RMDroid 的检测性能随着特征维度的增加呈现增长的趋势. 具体而言, 对数据集 $\mathbb{D}_{rand}$ 而言, 特征向量维度设置为 d = 350 时, RMDroid 的检测性能最优, 特征维度 200 和 300 的检测性能大体上与 d = 350 相近. 另一方面, 对更符合现实场景的 $\mathbb{D}_{time}$ 数据集而言, 在特征维度 300 下训练的 RMDroid 的有效性显著优于 250 和 350 对应的模型性能, 尤其在 *FPR*=0.1% 的更严格的场景下, 其 *TPR* 和 *bACC* 都优于其他模型, 表现出更高的稳定性. 因此, 综合考虑不同特征维度下 RMDroid 的检测性能以及维度增长带来的计算资源和训练时间开销, 最终本文将特征维度默认值设置为 300.

最后, 如本文第 2.4 节所述, 本文提出的鲁棒融合学习使用异类识别网络来识别被扰动的模态, 从而避免被扰动的模态对应的特征向量对最终预测结果造成较大的影响. 为了评估不同特征融合策略对 RMDroid 有效性的影响, 本文将异类识别网络和鲁棒融合预测网络替换为不同的特征融合策略构造 RMDroid 变体进行对比, 即 RMDroid(vote)、RMDroid(pool) 和 RMDroid(conc). 具体而言, RMDroid(vote) 使用投票方式综合多个单模态子模型

RMDroid(img)、RMDroid(api) 和 RMDroid(cfg) 的预测结果进行决策; RMDroid(pool) 中各模态学习的特征向量 $\{\vec{h}_z^{\text{img}},\vec{h}_z^{\text{api}},\vec{h}_z^{\text{cfg}}\}$ 通过最大池化层 (MaxPooling) 降维并提取显著特征, 然后经过具有 150 个神经元的隐藏层以及最终的输出层预测结果; RMDroid(conc) 则将各模态的特征向量 $\{\vec{h}_z^{\text{img}},\vec{h}_z^{\text{api}},\vec{h}_z^{\text{cfg}}\}$ 进行拼接, 并经过 300 个神经元的隐藏层以及输出层进行预测.

表 8 不同特征维度下 RMDroid 的检测性能 (%)

数据集	特征维度	AUC	*FPR*=1%		*FPR*=0.1%	
			TPR	*bACC*	*TPR*	*bACC*
$\mathbb{D}_{\text{rand}}$	200	99.51	94.29	96.65	80.68	90.30
	250	99.19	90.24	94.62	73.35	86.63
	300	99.52	93.55	96.27	75.84	87.87
	350	**99.68**	**94.94**	**96.97**	**82.36**	**91.14**
	400	99.54	93.83	96.42	65.35	82.63
$\mathbb{D}_{\text{time}}$	200	82.80	31.69	65.35	15.38	57.64
	250	79.16	32.02	65.52	9.32	54.61
	300	83.96	**41.76**	**70.38**	**23.10**	**61.51**
	350	83.89	33.70	66.36	13.04	56.47
	400	**87.16**	40.74	69.88	19.10	59.50

注: 加粗数据表示结果最佳, 下划线表示结果次优

从表 9 中实验结果可以观察到, 对 $\mathbb{D}_{\text{rand}}$ 数据集而言, RMDroid 的检测性能显著优于使用其他特征融合策略的 RMDroid 变体. 特别地, 在 *FPR*=0.1% 时 *TPR* 和 *bACC* 分别达到 68.73% 和 84.32%. 另一方面, 对 $\mathbb{D}_{\text{time}}$ 数据集而言, RMDroid 在 *FPR*=1% 时检测性能与最高的 RMDroid(pool) 相差较小, 但是在更有挑战性的条件下 (误报率 *FPR*=0.1%), RMDroid 的检测性能达到最高, 反映出 RMDroid 在保持较低误报率时优秀的检测性能. 综上所述, 使用投票、最大池化以及拼接等常见特征融合方法的检测性能 (特别在更有挑战性、在恶意软件检测场景下更重要的误报率 *FPR*=0.1% 的条件下) 整体上低于 RMDroid, 证明了本文所提出的基于鲁棒融合学习的恶意软件鲁棒检测模块 F_{robust} 的有效性.

表 9 不同特征融合策略下的检测性能 (%)

数据集	检测模型	AUC	*FPR*=1%		*FPR*=0.1%	
			TPR	*bACC*	*TPR*	*bACC*
$\mathbb{D}_{\text{rand}}$	RMDroid(vote)	98.91	83.43	91.22	16.80	58.35
	RMDroid(pool)	99.36	92.16	95.58	59.08	79.49
	RMDroid(conc)	**99.48**	93.62	96.31	65.49	82.70
	RMDroid	**99.48**	**93.88**	**96.45**	**68.73**	**84.32**
$\mathbb{D}_{\text{time}}$	RMDroid(vote)	**89.56**	38.32	68.67	14.46	57.18
	RMDroid(pool)	85.46	**44.98**	**72.00**	12.58	56.24
	RMDroid(conc)	82.03	37.98	68.50	20.19	60.05
	RMDroid	83.96	41.76	70.38	**23.10**	**61.51**

注: 加粗数据表示结果最佳, 下划线表示结果次优

3.2.2 鲁棒性实验结果与分析

为了评估本文所提出检测方法能够准确地检测出对抗性恶意软件的能力, 即鲁棒性实验, 本文选择第 3.1.4 节中介绍的 3 种最新的且最具有代表性的对抗性恶意软件生成方法 (即 COPYCAT、GADGET 与 HRAT) 来评估包括与所有基线检测方法在内所有目标检测方法的鲁棒性评价指标 *Robust*.

值得注意的是, 由于现有的对抗性恶意软件生成方法只能攻击部分基于特定模态特征的恶意软件检测方法, 因此本实验中 COPYCAT、GADGET 与 HRAT 这 3 种对抗性恶意软件生成方法只能分别评估特定目标检测方法的鲁棒性指标. 具体而言, 由于 COPYCAT 只会改变恶意软件的图像特征, 因此只能用于评估 IMCFN、RMDroid(img)

和 RMDroid 这 3 种检测方法的鲁棒性; GADGET 是专门针对 API 调用序列特征设计的攻击方法, 无法攻击基于图像特征和控制流图特征的恶意软件检测方法, 因而只能用于评估 RMDroid(api) 和 RMDroid 两种检测方法的鲁棒性; HRAT 是当前先进的针对 API 调用图的对抗攻击方法, 不仅可以直接攻击基于 API 调用图的恶意软件检测方法 MaMaDroid 和 MalScan, 而且可以间接地攻击 RMDroid(api) 和 RMDroid. 此外, 由于没有任何一个针对基于控制流图特征的安卓恶意软件检测方法而设计并实现的对抗攻击方法, 因此本文中无法评估 RMDroid(cfg) 单模态变体方法的鲁棒性.

特别地, 对于任意一个目标恶意软件检测方法 (IMCFN、MaMaDroid(f)、MaMaDroid(p)、MalScan 以及相应的单模态子模型方法) 而言, 综合考虑 COPYCAT、GADGET 与 HRAT 这 3 种对抗攻击方法的攻击效率和实验资源限制, 本文首先从测试集中 $\mathbb{D}_{\text{rand}}$ 和 $\mathbb{D}_{\text{time}}$ 两个数据集中分别随机选取 500 个已经被目标检测方法分配为恶意软件的样本作为候选集; 然后, 对于每个候选集中每个样本, 本文遵循每种对抗攻击方法的默认参数设置, 生成相应地针对目标检测方法的对抗性恶意软件; 最后, 利用公式 (17) 和公式 (18) 计算所有恶意软件检测方法在相应攻击方法下的鲁棒性指标 *Robust*. 表 10 展示了与所有基线方法分别在 $\mathbb{D}_{\text{rand}}$ 和 $\mathbb{D}_{\text{time}}$ 两项数据集上的鲁棒性实验结果.

表 10 RMDroid 与基线检测方法的鲁棒性实验结果 (%)

数据集	检测方法		鲁棒性 *Robust* 值		
	设置	名称	COPYCAT	GADGET	HRAT
$\mathbb{D}_{\text{rand}}$	*FPR*=1%	IMCFN	8.51	—	—
		MaMaDroid(f)	—	—	2.12
		MaMaDroid(p)	—	—	1.20
		MalScan	—	—	0
		RMDroid	**90.70**	**100**	**99.81**
	FPR=0.1%	IMCFN	5.72	—	—
		MaMaDroid(f)	—	—	0
		MaMaDroid(p)	—	—	2.99
		MalScan	—	—	0
		RMDroid	**81.00**	**100**	**100**
$\mathbb{D}_{\text{time}}$	*FPR*=1%	IMCFN	59.00	—	—
		MaMaDroid(f)	—	—	0
		MaMaDroid(p)	—	—	0
		MalScan	—	—	0
		RMDroid	**65.40**	**91.24**	**96.70**
	FPR=0.1%	IMCFN	17.00	—	—
		MaMaDroid(f)	—	—	0
		MaMaDroid(p)	—	—	0
		MalScan	—	—	0
		RMDroid	**59.00**	**93.25**	**97.65**

注: “—”代表该攻击方法无法被用于攻击相应的目标检测方法, 加粗数据表示结果最佳

首先, 对比分析所有基线检测方法在相应对抗攻击方法攻击情况下鲁棒性 *Robust* 的实验结果. 从表 10 中可以观察到, 对 $\mathbb{D}_{\text{rand}}$ 数据集上 *FPR*=1% 和 *FPR*=0.1% 两种情况下, 所有的基线检测方法都表现出极低的鲁棒性, *Robust* 值不超过 9%. 其中, MaMaDroid(f)、MaMaDroid(p) 和 MalScan 的鲁棒性 *Robust* 值更是低至 2.99%, 这意味着 HRAT 生成的对抗性安卓恶意软件中有高达 97% 的样本可以成功地绕过这 3 个基线检测方法. 另一方面, $\mathbb{D}_{\text{time}}$ 数据集上 *FPR*=1% 和 *FPR*=0.1% 两种情况下, 尽管 IMCFN 的鲁棒性 *Robust* 值有所提高, 但是 MaMaDroid(f)、MaMaDroid(p) 和 MalScan 仍然表现极差的鲁棒性, *Robust* 值全为 0%, 这意味着 HRAT 生成的所有对抗性安卓恶意软件都能成功地绕过这 3 个基线检测方法. 上述的实验结果与分析表明, 由于这些基线检测方法都主要建立在单一模态特征 (例如 API 调用信息等) 之上, 而以单一模态存在的特征是非常容易被攻击者所操控的 (例如, 修改

API 调用图等), 因此大部分基线检测方法的鲁棒性都非常低, 特别是针对 API 模态设计实现的 HRAT 对抗攻击方法几乎可以 100% 地绕过 MaMaDroid 和 MalScan 两类最新基于 API 的安卓恶意软件检测方法.

接着将与所有基线检测方法的鲁棒性实验结果进行对比分析. 从表 10 中可以观察到, 相比于 $\mathbb{D}_{time}$ 数据集, 尽管在更符合实际场景的数据集 $\mathbb{D}_{time}$ 上, 在 COPYCAT、GADGET 和 HRAT 这 3 种对抗攻击方法攻击情况下的鲁棒性 *Robust* 值分别有不同程度的降低, 但是在所有实验情况下的鲁棒性 *Robust* 值都远高于其他基线检测方法. 特别地, 在利用 HRAT 进行攻击情况下, MaMaDroid(f)、MaMaDroid(p) 和 MalScan 这 3 个基线检测方法的最高鲁棒性 *Robust* 值只有 2.99%, 然而 RMDroid 却表现出较高的鲁棒性, *Robust* 值高达 96.70%. 上述的实验结果与分析表明, 由于本文所提出的 RMDroid 融合学习了 3 种模态特征, 并利用异类识别网络降低甚至消除受到对抗攻击的模态特征对最终恶意软件检测结果的影响, 从而在所有实验情况下都表现出比基线检测方法更高的鲁棒性.

最后, 为了分析各个模态特征对鲁棒性实验结果的影响, 本文进一步分析与各个单模态检测方法的对比实验结果. 表 11 展示了 RMDroid 及其 3 个单模态子模型分别在 $\mathbb{D}_{rand}$ 和 $\mathbb{D}_{time}$ 两项数据集上的鲁棒性实验结果. 一方面, 通过对表 11 中 RMDroid(img) 和 RMDroid(api) 两个变体的单模态检测方法的实验结果进行对比, 可以明显地观察到 RMDroid(img) 的鲁棒性 *Robust* 值在各种实验条件下均高于 RMDroid(api), 这表明相较于 API 调用模态特征, 灰度图模态特征更不容易被对抗攻击所干扰. 另一方面, 如果将完整的与变体的单模态检测方法进行对比, 本文发现, 不论是在 $\mathbb{D}_{rand}$ 和 $\mathbb{D}_{time}$ 两个数据集上, 还是在 *FPR*=1% 和 *FPR*=0.1% 两种情况下, RMDroid 的鲁棒性评价指标都显著地高于 RMDroid(img) 与 RMDroid(api). 因此, 基于上述实验结果与分析, 本文发现同时利用并学习安卓恶意软件的灰度图、API 调用序列和控制流图这 3 种模态特征也可以显著地提高安卓恶意软件检测方法的抵御对抗攻击的鲁棒性, 因此进一步证明了本文所提出多模态特征融合学习的重要性.

表 11 RMDroid 与单模态子模型的鲁棒性实验结果 (%)

数据集	检测方法		鲁棒性 *Robust* 值		
	设置	名称	COPYCAT	GADGET	HRAT
$\mathbb{D}_{rand}$	*FPR*=1%	RMDroid(img)	49.94	—	—
		RMDroid(api)	—	0.21	3.31
		RMDroid(cfg)	—	—	—
		RMDroid	**90.70**	**100**	**99.81**
	FPR=0.1%	RMDroid(img)	45.45	—	—
		RMDroid(api)	—	0	4.35
		RMDroid(cfg)	—	—	—
		RMDroid	**81.00**	**100**	**100**
$\mathbb{D}_{time}$	*FPR*=1%	RMDroid(img)	47.00	—	—
		RMDroid(api)	—	0.13	0
		RMDroid(cfg)	—	—	—
		RMDroid	**65.40**	**91.24**	**96.70**
	FPR=0.1%	RMDroid(img)	45.06	—	—
		RMDroid(api)	—	0	0
		RMDroid(cfg)	—	—	—
		RMDroid	**59.00**	**93.25**	**97.65**

注: “—”代表该攻击方法无法被用于攻击相应的目标检测方法, 加粗数据表示结果最佳

4 总结与展望

针对当前安卓恶意软件检测方法面临对抗样本攻击的严重安全威胁, 本文认为基于单一模态特征的恶意软件检测方法很容易被攻击者所操控, 并据此提出一种基于多模态融合学习的安卓恶意软件鲁棒检测方法, 可以在不影响针对一般性安卓恶意软件检测准确率的基础上, 大幅度提高其识别对抗性恶意软件的鲁棒性. 具体而言, 首先从待测安卓软件中提取与挖掘不同模态的特征信息, 然后分别利用相应的深度学习模型充分地学习相应模态的特

征向量, 最后利用异类识别网络降低甚至消除多模态特征中受到对抗攻击干扰的模态特征对最终恶意软件预测的影响, 从而提高其抵御对抗攻击的鲁棒性. 实验结果表明, 相较于现有基线检测方法展现了更高的有效性和鲁棒性, 即不仅可以有效地检测一般性的安卓恶意软件, 而且可以检测出攻击者恶意生成的对抗性安卓恶意软件.

在下一步研究工作中, 本文计划从以下两方面进行研究探索. 首先, 尽管本文已经选取当前最先进的 3 种对抗攻击方法来评估的鲁棒性, 但是由于目前没有任何针对安卓恶意软件中控制流图特征的对抗攻击, 更没有任何针对安卓恶意软件中多模态特征的对抗攻击, 因此无法评估控制流图模态特征以及多模态特征的鲁棒性. 后续的研究工作将重点探索如何设计并提出针对基于控制流图的安卓恶意软件检测的对抗攻击方法, 并进一步提出针对多模态特征的安卓恶意软件检测的对抗攻击方法, 从而进一步评估最新安卓恶意软件检测方法及工具的鲁棒性. 其次, 尽管本文利用已有的对抗攻击方法评估安卓恶意软件检测方法的经验鲁棒性, 但是无法准确地评估该检测方法针对任何对抗攻击方法的理论鲁棒性. 事实上, 可验证鲁棒性是对抗机器学习领域的一个重要研究方向, 旨在利用形式化验证等方法来证明机器学习或者深度学习模型在一定范围内对对抗攻击是鲁棒的. 然而, 现有的可验证鲁棒性研究大量集中在计算机视觉和自然语言处理领域, 因此亟需针对基于学习的安卓恶意软件检测方法进行可验证鲁棒性研究, 构建可信的安卓恶意软件检测系统, 对保障智能移动终端用户的信息安全具有重要意义.

References

[1] TechInsights. Global smartphone sales forecast by operating system for 88 countries: 2007 to 2029. 2025. https://www.techinsights.com/blog/global-smartphone-sales-forecast-operating-system-88-countries-2007-2029

[2] XuebaoCaijingshe. Institution: The global smartphone sales are expected to decline by 5% YoY in 2023, with an approximate Android market share of 80%. 2023 (in Chinese). https://www.speed-daily.com/newsflashes.html#23981

[3] Kaspersky. The mobile malware threat landscape in 2022. 2023. https://securelist.com/mobile-threat-report-2022/108844/

[4] Kaspersky. Harly: Another Trojan subscriber on Google Play. 2023. https://www.kaspersky.com/blog/harly-trojan-subscriber/45573/

[5] Liu Y, Tantithamthavorn C, Li L, Liu YP. Deep learning for Android malware defenses: A systematic literature review. ACM Computing Surveys, 2022, 55(8): 153. [doi: 10.1145/3544968]

[6] Google for Developers. Google Play protect. 2025. https://developers.google.com/android/play-protect

[7] Norton Inc. Norton mobile security for Android. 2025. https://uk.norton.com/products/mobile-security-for-android?srsltid=AfmBOopv_qRZtkOMHkt4fsIWHlAqGcEOrJ4gLJ5GCkl1-PykY5fl6Gv7

[8] Tencent Inc. Tencent mobile manager. 2025 (in Chinese). https://m.qq.com/

[9] Qihoo 360 Inc. 360 Security for Mobile Devices 9.0. 2025 (in Chinese). https://shouji.360.cn/v6/index.html

[10] Guarnieri C, Tanasi A, Bremer J, Schloesser M, Houtman K, van Zutphen R, Graaff B. Cuckoo sandbox. 2019. https://cuckoosandbox.org/community-sub/index

[11] Leavitt N. Malicious code moves to mobile devices. Computer, 2000, 33(12): 16–19.

[12] Feng Y, Anand S, Dillig I, Aiken 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. Hong Kong: ACM, 2014. 576–587. [doi: 10.1145/2635868.2635869]

[13] Faruki P, Bharmal A, Laxmi V, Ganmoor V, Gaur MS, Conti M, Rajarajan M. Android security: A survey of issues, malware penetration, and defenses. IEEE Communications Surveys & Tutorials, 2015, 17(2): 998–1022. [doi: 10.1109/COMST.2014.2386139]

[14] Ye YF, Li T, Adjeroh D, Iyengar SS. A survey on malware detection using data mining techniques. ACM Computing Surveys, 2017, 50(3): 41. [doi: 10.1145/3073559]

[15] Pan WW, Wang XY, Song ML, Chen C. Survey on generating adversarial examples. Ruan Jian Xue Bao/Journal of Software, 2020, 31(1): 67–81 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5884.htm [doi: 10.13328/j.cnki.jos.005884]

[16] Ling X, Wu LF, Zhang JY, Qu ZQ, Deng W, Chen X, Qian YG, Wu CM, Ji SL, Luo TY, Wu JZ, Wu YJ. Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art. Computers & Security, 2023, 128: 103134. [doi: 10.1016/j.cose.2023.103134]

[17] Zhao KF, Zhou H, Zhu YL, Zhan X, Zhou K, Li JF, Yu L, Yuan W, Luo XP. Structural attack against graph based Android malware detection. In: Proc. of the 2021 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2021. 3218–3235. [doi: 10.1145/3460120.3485387]

[18] Wu YM, Li XD, Zou DQ, 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). San Diego: IEEE, 2019. 139–150. [doi: 10.1109/ASE.2019.00023]

[19] Onwuzurike L, Mariconti E, Andriotis P, de Cristofaro E, Ross G, Stringhini G. MaMaDroid: Detecting Android malware by building Markov chains of behavioral models (extended version). ACM Trans. on Privacy and Security, 2019, 22(2): 14. [doi: 10.1145/3313391]

[20] Wang JL, Zhang C, Qi XY, Rong Y. A survey of intelligent malware detection on Windows platform. Journal of Computer Research and Development, 2021, 58(5): 977–994 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2021.20200964]

[21] Wang SY. Research on robust detection method of Android malware based on multimodal fusion learning [MS. Thesis]. Beijing: University of Chinese Academy of Sciences, 2023 (in Chinese with English abstract).

[22] Kim T, Kang B, Rho M, Sezer S, Im EG. A multimodal deep learning method for Android malware detection using various features. IEEE Trans. on Information Forensics and Security, 2019, 14(3): 773–788. [doi: 10.1109/TIFS.2018.2866319]

[23] Zhang SF, Su H, Liu HX, Yang W. MPDroid: A multimodal pre-training Android malware detection method with static and dynamic features. Computers & Security, 2025, 150: 104262. [doi: 10.1016/j.cose.2024.104262]

[24] Li X, Liu L, Liu YZ, Zhao Y, Zhang P, Liu HX. Multimodal fusion for Android malware detection based on large pre-trained models. IEEE Trans. on Software Engineering, 2025, 51(5): 1569–1590. [doi: 10.1109/TSE.2025.3557577]

[25] Chen SJ, Lang B, Liu HY, Chen YK, Song YC. Android malware detection method based on graph attention networks and deep fusion of multimodal features. Expert Systems with Applications, 2024, 237: 121617. [doi: 10.1016/j.eswa.2023.121617]

[26] Wikipedia. APK. 2025 (in Chinese). https://zh.wikipedia.org/wiki/APK

[27] Li YS. Malware detection on mobile endpoints. 2022 (in Chinese). https://www.51cto.com/article/720959.html

[28] Aafer Y, Du WL, Yin H. DroidAPIMiner: Mining API-level features for robust malware detection in Android. In: Zia T, Zomaya A, Varadharajan V, Mao M, eds. Proc. of the 9th Int'l ICST Conf. on Security and Privacy in Communication Networks. Sydney: Springer, 2013. 86–103. [doi: 10.1007/978-3-319-04283-1_6]

[29] Grace M, Zhou YJ, Zhang Q, Zou SH, Jiang XX. RiskRanker: Scalable and accurate zero-day Android malware detection. In: Proc. of the 10th Int'l Conf. on Mobile Systems, Applications, and Services. Low Wood Bay: ACM, 2012. 281–294. [doi: 10.1145/2307636.2307663]

[30] Yuan ZL, Lu YQ, Wang ZG, Xue YB. Droid-Sec: Deep learning in Android malware detection. In: Proc. of the 2014 ACM Conf. on SIGCOMM. Chicago: ACM, 2014. 371–372. [doi: 10.1145/2619239.2631434]

[31] Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K. DREBIN: Effective and explainable detection of Android malware in your pocket. In: Proc. of the 2014 NDSS Symp. San Diego: Internet Society, 2014. [doi: 10.14722/ndss.2014.23247]

[32] Xiao XS, Yang S. An image-inspired and CNN-based Android malware detection approach. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 1259–1261. [doi: 10.1109/ASE.2019.00155]

[33] Tang JW, Xu W, Peng T, Zhou SJ, Pi QS, He RH, Hu XR. Android malware detection based on a novel mixed bytecode image combined with attention mechanism. Journal of Information Security and Applications, 2024, 82: 103721. [doi: 10.1016/j.jisa.2024.103721]

[34] Ling X, Ji SL, Ren K. Adversarial example attacks and defenses of deep learning systems. Communications of the CCF, 2018, 14(6): 11–17 (in Chinese).

[35] Ling X, Ji SL, Zou JX, Wang JN, Wu CM, Li B, Wang T. DEEPSEC: A uniform platform for security analysis of deep learning model. In: Proc. of the 2019 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2019. 673–690. [doi: 10.1109/SP.2019.00023]

[36] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R. Intriguing properties of neural networks. arXiv:1312.6199, 2014.

[37] Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. arXiv:1412.6572, 2015.

[38] Carlini N, Wagner D. Towards evaluating the robustness of neural networks. In: Proc. of the 2017 IEEE Symp. on Security and Privacy. San Jose: IEEE, 2017. 39–57. [doi: 10.1109/SP.2017.49]

[39] Zhang JL, Li C. Adversarial examples: Opportunities and challenges. IEEE Trans. on Neural Networks and Learning Systems, 2020, 31(7): 2578–2593. [doi: 10.1109/TNNLS.2019.2933524]

[40] Zhang WE, Sheng QZ, Alhazmi A, Li CL. Adversarial attacks on deep-learning models in natural language processing: A survey. ACM Trans. on Intelligent Systems and Technology (TIST), 2020, 11(3): 24. [doi: 10.1145/3374217]

[41] Zheng BL, Jiang PP, Wang Q, Li Q, Shen C, Wang C, Ge YJ, Teng QY, Zhang SY. Black-box adversarial attacks on commercial speech platforms with minimal information. In: Proc. of the 2021 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2021. 86–107. [doi: 10.1145/3460120.3485383]

[42] Wan Y, Zhang SJ, Zhang HY, Sui YL, Xu GD, Yao DZ, Jin H, Sun LC. You see what I want you to see: Poisoning vulnerabilities in neural code search. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 1233–1245. [doi: 10.1145/3540250.3549153]

[43] Xu WL, Qi YJ, Evans D. Automatically evading classifiers: A case study on PDF malware classifiers. In: Proc. of the 2016 NDSS. San Diego: Internet Society, 2016. [doi: 10.14722/ndss.2016.23115]

[44] Vi BN, Nguyen HN, Nguyen NT, Tran CT. Adversarial examples against image-based malware classification systems. In: Proc. of the 11th Int'l Conf. on Knowledge and Systems Engineering (KSE). Da Nang: IEEE, 2019. 1–5. [doi: 10.1109/KSE.2019.8919481]

[45] Darwaish A, Naït-Abdesselam F, Titouna C, Sattar S. Robustness of image-based Android malware detection under adversarial attacks. In: Proc. of the 2021 IEEE Int'l Conf. on Communications. Montreal: IEEE, 2021. 1–6. [doi: 10.1109/ICC42927.2021.9500425]

[46] Naeem H, Ullah F, Krejcar O, Alsirhani A, Zhao Y. Adversarial malware detection on consumer devices using optimized image-based ensembles. IEEE Consumer Electronics Magazine, 2024. 1–10. [doi: 10.1109/MCE.2024.3507282]

[47] Tan K, Zhan DY, Ye L, Zhang HL, Fang BX. A practical adversarial attack against sequence-based deep learning malware classifiers. IEEE Trans. on Computers, 2024, 73(3): 708–721. [doi: 10.1109/TC.2023.3339955]

[48] Zhang F, Feng RT, Xie XF, Li XH, Shi LS. SeqAdver: Automatic payload construction and injection in sequence-based Android adversarial attack. In: Proc. of the 2023 IEEE Int'l Conf. on Data Mining Workshops (ICDMW). Shanghai: IEEE, 2023. 1342–1351. [doi: 10.1109/ICDMW60847.2023.00172]

[49] Li H, Cheng Z, Wu B, Yuan LH, Gao CY, Yuan W, Luo XP. Black-box adversarial example attack towards FCG based Android malware detection under incomplete feature information. In: Proc. of the 32nd USENIX Security Symp. Anaheim: USENIX Association, 2023. 1181–1198.

[50] He P, Xia YF, Zhang XH, Ji SL. Efficient query-based attack against ML-based Android malware detection under zero knowledge setting. In: Proc. of the 2023 ACM SIGSAC Conf. on Computer and Communications Security. Copenhagen: ACM, 2023. 90–104. [doi: 10.1145/3576915.3623117]

[51] Machado GR, Silva E, Goldschmidt RR. Adversarial machine learning in image classification: A survey toward the defender's perspective. ACM Computing Surveys, 2021, 55(1): 8. [doi: 10.1145/3485133]

[52] Xu H, Ma Y, Liu HC, Deb D, Liu H, Tang JL, Jain AK. Adversarial attacks and defenses in images, graphs and text: A review. Int'l Journal of Automation and Computing, 2020, 17(2): 151–178. [doi: 10.1007/s11633-019-1211-x]

[53] Akhtar N, Mian A. Threat of adversarial attacks on deep learning in computer vision: A survey. IEEE Access, 2018, 6: 14410–14430. [doi: 10.1109/ACCESS.2018.2807385]

[54] Papernot N, McDaniel P, Wu X, Jha S, Swami A. Distillation as a defense to adversarial perturbations against deep neural networks. In: Proc. of the 2016 IEEE Symp. on Security and Privacy (SP). San Jose: IEEE, 2016. 582–597. [doi: 10.1109/SP.2016.41]

[55] Ross A, Doshi-Velez F. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In: Proc. of the 32nd AAAI Conf. on Artificial Intelligence. New Orleans: AAAI, 2018. 1660–1669. [doi: 10.1609/aaai.v32i1.11504]

[56] Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A. Towards deep learning models resistant to adversarial attacks. arXiv:1706.06083, 2019.

[57] Vasan D, Alazab M, Wassan S, Naeem H, Safaei B, Zheng Q. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. Computer Networks, 2020, 171: 107138. [doi: 10.1016/j.comnet.2020.107138]

[58] Yan JQ, Yan GH, Jin D. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In: Proc. of the 49th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN). Portland: IEEE, 2019. 52–63. [doi: 10.1109/DSN.2019.00020]

[59] Ling X, Wu LF, Deng W, Qu ZQ, Zhang JY, Zhang S, Ma TF, Wang B, Wu CM, Ji SL. MalGraph: Hierarchical graph neural networks for robust windows malware detection. In: Proc. of the 2022 IEEE Conf. on Computer Communications. London: IEEE, 2022. 1998–2007. [doi: 10.1109/INFOCOM48880.2022.9796786]

[60] Belguendouz H, Guerid H, Kaddour M. Static classification of IoT malware using grayscale image representation and lightweight convolutional neural networks. In: Proc. of the 5th Int'l Conf. on Advanced Communication Technologies and Networking (CommNet). Marrakech: IEEE, 2022. 1–8. [doi: 10.1109/CommNet56067.2022.9993956]

[61] Jung J, Choi J, Cho S, Han S, Park M, Hwang Y. Android malware detection using convolutional neural networks and data section images. In: Proc. of the 2018 Conf. on Research in Adaptive and Convergent Systems. Honolulu: ACM, 2018. 149–153. [doi: 10.1145/3264746.3264780]

[62] He KM, Zhang XY, Ren SQ, Sun J. Deep residual learning for image recognition. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). Las Vegas: IEEE, 2016. 770–778. [doi: 10.1109/CVPR.2016.90]

[63] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735–1780. [doi: 10.1162/neco.1997.9.8.1735]

[64] Ling X, Wu LF, Wu CM, Ji SL. Graph neural networks: Graph matching. In: Wu LF, Cui P, Pei J, Zhao L, eds. Graph Neural Networks:

Foundations, Frontiers, and Applications. Singapore: Springer, 2022. 277–295. [doi: 10.1007/978-981-16-6054-2_13]

[65] Hamilton WL, Ying R, Leskovec J. Inductive representation learning on large graphs. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 1025–1035.

[66] Konstantinos P. Multimodal deep learning: Definition, examples, applications. 2022. https://www.v7labs.com/blog/multimodal-deep-learning-guide

[67] Yang K, Lin WY, Barman M, Condessa F, Kolter Z. Defending multimodal fusion models against single-source adversaries. In: Proc. of the 2021 IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR). Nashville: IEEE, 2021. 3339–3348. [doi: 10.1109/CVPR46437.2021.00335]

[68] Shazeer N, Mirhoseini A, Maziarz K, Davis A, Le Q, Hinton G, Dean J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv:1701.06538, 2017.

[69] Allix K, Bissyandé TF, Klein J, Le Traon Y. AndroZoo: Collecting millions of Android apps for the research community. In: Proc. of the 13th Int'l Conf. on Mining Software Repositories. Austin: ACM, 2016. 468–471. [doi: 10.1145/2901739.2903508]

[70] Daoudi N, Allix K, Bissyandé TF, Klein J. Assessing the opportunity of combining state-of-the-art Android malware detectors. Empirical Software Engineering, 2023, 28(2): 22. [doi: 10.1007/S10664-022-10249-9]

[71] Wu YF, Shi J, Wang PC, Zeng DR, Sun C. DeepCatra: Learning flow- and graph-based behaviours for Android malware detection. IET Information Security, 2023, 17(1): 118–130. [doi: 10.1049/ise2.12082]

[72] Lu J, Liu AJ, Dong F, Gu F, Gama J, Zhang GQ. Learning under concept drift: A review. IEEE Trans. on Knowledge and Data Engineering, 2019, 31(12): 2346–2363. [doi: 10.1109/TKDE.2018.2876857]

[73] Khormali A, Abusnaina A, Chen SQ, Nyang D, Mohaisen A. COPYCAT: Practical adversarial attacks on visualization-based malware detection. arXiv:1909.09735, 2019.

[74] Rosenberg I, Shabtai A, Rokach L, Elovici Y. Generic black-box end-to-end attack against state of the art API call based malware classifiers. In: Bailey M, Holz T, Stamatogiannakis M, Ioannidis S, eds. Proc. of the 21st Int'l Symp. on Research in Attacks, Intrusions, and Defenses. Heraklion: Springer, 2018. 490–510. [doi: 10.1007/978-3-030-00470-5_23]

[75] Desnos A. Androguard documentation. 2019. https://androguard.readthedocs.io/en/latest/index.html

附中文参考文献

[2] 雪豹财经社. 机构: 2023 年全球智能手机销量将同比下降 5%, 安卓份额约为 80%. 2023. https://www.speed-daily.com/newsflashes.html#23981

[8] Tencent Inc. 腾讯手机管家. 2025. https://m.qq.com/

[9] Qihoo 360 Inc. 360 手机卫士 9.0. 2025. https://shouji.360.cn/v6/index.html

[15] 潘文雯, 王新宇, 宋明黎, 陈纯. 对抗样本生成技术综述. 软件学报, 2020, 31(1): 67–81. http://www.jos.org.cn/1000-9825/5884.htm [doi: 10.13328/j.cnki.jos.005884]

[20] 汪嘉来, 张超, 戚旭衍, 荣易. Windows 平台恶意软件智能检测综述. 计算机研究与发展, 2021, 58(5): 977–994. [doi: 10.7544/issn1000-1239.2021.20200964]

[21] 王时予. 基于多模态融合学习的安卓恶意软件鲁棒检测方法研究 [硕士学位论文]. 北京: 中国科学院大学, 2023.

[26] 维基百科. APK. 2025. https://zh.wikipedia.org/wiki/APK

[27] 李嫣娴. 移动终端侧的恶意软件检测. 2022. https://www.51cto.com/article/720959.html

[34] 凌祥, 纪守领, 任奎. 面向深度学习系统的对抗样本攻击与防御. 中国计算机学会通讯, 2018, 14(6): 11–17.

作者简介

凌祥, 博士, 副研究员, CCF 高级会员, 主要研究领域为智能软硬件安全, 人工智能安全.

周伯霖, 硕士, 主要研究领域为智能软件安全, 软件供应链安全.

王时予, 硕士, 主要研究领域为机器学习, 软件安全, 恶意软件检测.

罗天悦, 工程师, CCF 专业会员, 主要研究领域为操作系统安全分析, 代码漏洞挖掘, 人工智能安全.

尹鹏, 博士, 副研究员, 主要研究领域为网络安全, 设备安全检测.

吴春明, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为互联网体系架构, 可编程网络, 网络空间内生安全.

王滨, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为人工智能安全, 物联网安全, 密码学.

吴敬征, 博士, 研究员, 博士生导师, CCF 杰出会员, 主要研究领域为系统安全, 漏洞挖掘, 操作系统安全.