计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220297
60(8): 1875−1894, 2023

# Android 恶意应用的静态检测方法综述

潘建文[1] 崔展齐[1] 林高毅[1] 陈 翔[2] 郑丽伟[1]
[1](北京信息科技大学计算机学院 北京 100101)
[2](南通大学信息科学技术学院 江苏南通 226019)
(panjianwen@bistu.edu.cn)

# A Review of Static Detection Methods for Android Malicious Application

Pan Jianwen[1], Cui Zhanqi[1], Lin Gaoyi[1], Chen Xiang[2], and Zheng Liwei[1]
[1](*Computer School*, *Beijing Information Science and Technology University*, *Beijing* 100101)
[2](*School of Information Science and Technology*, *Nantong University*, *Nantong*, *Jiangsu* 226019)

**Abstract** Due to the openness of the Android system and the diversity of the third-party application markets, Android system has achieved a high market share while brought huge risks. As a result, Android malware emerge endlessly and spread widely, which seriously threaten users' privacy and economic security. How to effectively detect Android malware has been widely concerned by researchers. According to whether the application is executed or not, the existing malware detection methods are divided into static detection and dynamic detection. Between the two, the static detection methods outperform the dynamic detection methods in terms of efficiency and code coverage, Drebin and other static detection tools have been widely used. We systematically review the research progress in the field of static Android malware detection. First, the static features of Android applications are introduced. Then, according to different static features used for detecting Android malware, the static Android malware detection methods are classified into three categories: permissions, application programming interface (API), and opcode based approaches, and the Android application data sets and indicators commonly used to evaluate the detection performance of Android malware are summarized. Finally, potential research directions of static Android malware detection techniques in the future are discussed, which provides references for researchers in related directions.

**Key words** Android malware; static detection; permission; application programming interface (API); opcode

**摘 要** Android 系统的开放性和第三方应用市场的多样性,使其在取得高市场占有率的同时也带来了巨大的风险,导致 Android 恶意应用层出不穷并广泛传播,严重威胁了用户的隐私和经济安全.如何有效检测 Android 恶意应用受到了研究人员的广泛关注.根据是否运行应用程序,将现有的恶意应用检测方法分为静态检测和动态检测.其中,静态检测的效率和代码覆盖率均优于动态检测,Drebin 等静态检测工具取得了广泛应用.为此,系统调研了 Android 恶意应用静态检测领域的研究进展,并进行了分析和总结.首先,介绍了 Android 应用静态特征;然后,根据静态特征的不同,分别对基于权限、应用程序编程接口(application programming interface, API)和操作码等不同静态特征的 Android 恶意应用检测方法进行了分

收稿日期: 2022-04-12; 修回日期: 2022-10-10
基金项目: 江苏省前沿引领技术基础研究专项 (BK202002001); 国家自然科学基金项目 (61702041); 北京信息科技大学"勤信人才"培育计划项目 (QXTCP C201906)
This work was supported by the Jiangsu Provincial Frontier Leading Technology Fundamental Research Project (BK20202001), the National Natural Science Foundation of China (61702041), and the Beijing Information Science and Technology University "Qin-Xin Talent" Cultivation Plan Project (QXTCP C201906).
通信作者: 崔展齐(czq@bistu.edu.cn)

析,并总结了常用的 Android 应用数据集和评价 Android 恶意应用检测性能的常用指标;最后,对 Android 恶意应用静态检测技术的发展进行了总结和展望,以期为该领域的研究人员提供参考.

**关键词** Android 恶意应用;静态检测;权限;应用编程接口;操作码

**中图法分类号** TP311.5

近年来,随着移动互联网的发展,社交通信、金融证券、游戏娱乐和电子商务等应用全面地从 PC 转向移动互联网,移动互联网用户和设备数量呈指数级增长. 据 2021 年中国互联网网络安全报告[1] 显示,截至 2021 年 12 月,我国手机用户规模达 10.32 亿,我国使用手机上网的用户比例达 99.7%. 根据国际数据公司(IDC)调查报告[2] 显示,2021 年 Android 手机的市场份额占 84%,并不断增长. 开源的 Android 系统和多样的应用分发平台在带来巨大的市场份额的同时,也使其成为了恶意应用滋生的温床和传播的主要平台. 国家互联网应急中心捕获和通过厂商交换获得的移动恶意应用也主要集中在 Android 平台,仅在 2020 年就多达 303 万个[3]. 部分 Android 恶意应用的广泛传播造成了严重后果. 例如,2019 年出现的零日漏洞 Bad Binder 被 Android 恶意应用利用以控制用户设备[4],影响到 Android 9 及之前的版本,给大量用户带来了巨大经济损失.

因此,如何有效检测 Android 恶意应用受到了研究人员的广泛关注. 根据其是否运行应用程序,将现有的检测技术分为动态检测和静态检测 2 类,其中,动态检测[5-6] 是在真实设备或者模拟器中运行应用,使用测试脚本模拟用户操作,记录应用的运行情况,通过分析应用行为特征,如应用程序编程接口(application programming interface, API)调用、网络流量等特征来检测恶意应用. 动态检测的优点是可以克服代码加密、混淆和动态代码加载的障碍,其缺点是在真实设备或模拟器中运行测试脚本时间开销大、难以覆盖应用全部功能,且耗时较长. 静态检测则是通过反编译应用提取权限、API 和操作码等特征,采用机器学习算法构造分类器以检测 Android 恶意应用. 静态检测不依赖运行环境、代码覆盖率较高,检测效率高于动态检测方法,例如,Drebin[7] 检测工具在学术界和工业届受到了广泛应用和高度重视. 本文重点对 Android 恶意应用静态检测方法的已有研究成果进行综述.

本文搜集了 2013 年至今发表过的关于 Android 恶意应用静态检测相关论文并进行了梳理. 首先,使用“Android(安卓)”组合“malware/malicious(恶意应用)”、“detection(检测)”和“classification(分类)”等关键字,在国内外重要的论文数据库(例如 IEEE, ACM, Springer, Elsevier, CNKI)中检索相关论文. 由于中国计算机学会(CCF)推荐的期刊和会议影响力较大,具有一定的权威性,我们通过人工筛选的方式,检索出发表在 CCF 评级为 A, B 类的国际会议或国际期刊,以及《软件学报》《计算机学报》《计算机研究与发展》等权威中文期刊上的论文. 然后,分析论文的标题和摘要,去除与本综述关注问题无关的论文. 最终,选择出与本研究问题直接相关的高质量论文 89 篇,包括国际期刊 77 篇、中文期刊 12 篇. 其中,TIFS8 篇、TDSC6 篇、CCS2 篇、Computers&Security25 篇、《软件学报》3 篇、《计算机学报》1 篇、《计算机研究与发展》3 篇. 检索到的论文在不同年份的数量分布如图 1 所示,可以看出相关论文发表数量持续增长,这表明 Android 恶意应用检测是近年来的热点研究问题. 发表于 CCF 的 A, B 类国际期刊和国际会议的论文在不同领域的分布情况如图 2 所示,本研究问题

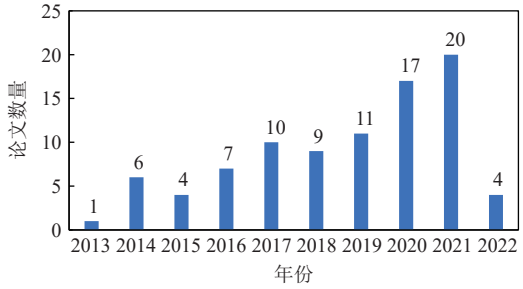


Fig. 1 Distribution of the number of research papers published in different years

图 1 在不同年份发表的研究论文数量分布

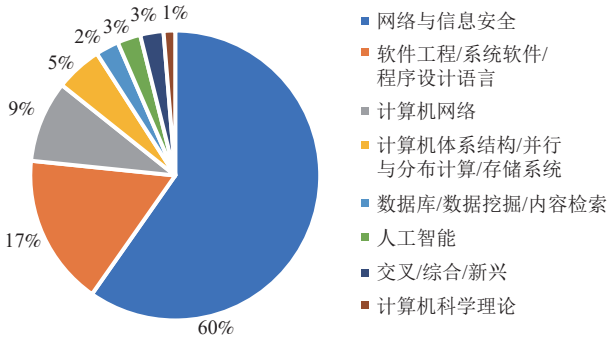


Fig. 2 Field distribution of papers published in international journals and conferences ranked as CCF A and B

图 2 发表于 CCF A, B 类国际期刊和会议的论文领域分布

的论文大多数分布在网络与信息安全领域(46 篇).

目前，在 Android 恶意应用静态检测的综述中，已有的工作包括：Samra 等人[8]在 2019 年对 Android 恶意应用静态检测进行综述，但是仅介绍了基于权限和基于签名的检测方法，对于其他静态特征方法内容略为缺失；Bayazit 等人[9]在 2020 年对传统的机器学习检测 Android 恶意应用进行了综述，他们考虑的是恶意应用分类，并对动态和静态检测方法进行了介绍；在此之后，有 41 篇重要论文发表，尤其是基于深度学习的 Android 恶意应用检测技术取得进一步的发展.

本文主要关注 Android 恶意应用静态检测方法，首先，介绍了 Android 应用的静态特征；然后，分别对基于权限、API、基于操作码和其他静态特征的检测方法进行分析和总结，并论述数据集和评价指标；最后，对现有的研究工作总结，并对未来的研究方向进行了展望.

## 1 Android 应用静态特征

APK(Android application package)是一种 Android 操作系统使用的应用程序包文件格式，用于应用分发和系统安装. Android 静态特征提取是根据对 Android 隐私安全和开发过程的了解，在不运行应用程序情况下分析 APK 中的文件内容，从中收集和提取各类特征. 基于所提取到的静态特征，可采用不同的算法应用于 Android 恶意应用检测.

常见的提取特征方法包括：采用 Androguard[10]等逆向工具直接提取权限、API 等部分特征；使用 Apktool[11]等工具进行反编译分析，这种提取特征方法所提取到的特征更为全面. 如图 3 所示，Apktool 反编译后的文件结构包括 Android 系统可执行(Dalvik executable, DEX)文件、AndroidManifest.xml 文件、res 文件夹、META-INF 文件夹和 lib 文件夹. DEX 文件是 Dalvik[12]和 ART 虚拟机可执行的字节码文件，可被反编译为 Smali 或 Java 文件集合. Smali 文件中包含应用程序的全部操作指令以及运行时数据，每个 Smali 文件包含类，类中包含多个方法，可在其中提取 API 调用以及由操作码(opcode)和操作数所组成的指令. AndroidManifest.xml 是清单配置文件，其包含了 Android 系统为运行应用程序所需要的参数，如权限、软件包名称、应用组件、应用所需要的硬件和软件功能等信息. res 文件夹用于存放资源文件，包括应用执行所需的布局、图像、动画、颜色和界面字符串等. META-INF 目录下存放的是签名信息，用来保证 APK 包的完整性和系统的安全，Android 系统要求所有 APK 必须先使用证书进行数字签名，然后才能安装、更新和分发. lib 目录存放应用程序依赖库的 so 文件. so 文件是在 Android 系统上直接运行的二进制代码，可在应用运行时访问设备实体组件，例如传感器和触摸输入等，是 Android 的动态链接库.

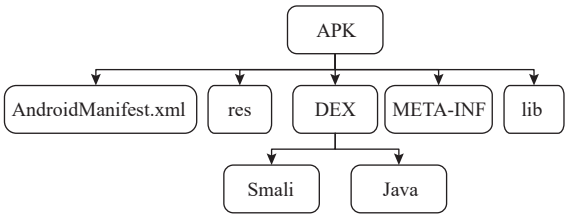


Fig. 3 Decompiling structure of APK

图 3 APK 反编译结构

APK 文件中的内容可被提取作为静态特征. 根据所使用静态特征类型的不同，衍生出不同的检测方法，本文中将其分为基于权限特征、基于 API 特征、基于操作码特征和基于其他特征 4 类 Android 恶意应用检测方法. 本文将在第 2～5 节中分别对这 4 类静态检测方法进行详细介绍.

## 2 基于权限的 Android 恶意应用检测方法

### 2.1 简 述

权限机制用于限制 Android 应用程序访问文件、数据和资源等安全敏感项. 为保护用户隐私数据，Google 为 Android 10 版本设置了 155 个权限，开发人员可根据应用功能的需要向用户申请权限. 然而，调查数据表明，大多数用户不够了解应用所需的权限，导致过度授权的情况频发[13]. 一旦恶意应用申请的敏感权限获得通过，将可能导致用户遭受隐私泄露、恶意扣费和资费消耗等严重后果，侵害用户利益. 良性应用和恶意应用所申请的权限在数量、类别和关系上存在一定的差异，这使得基于权限分析检测 Android 恶意应用具有可行性. 基于权限的 Android 恶意应用检测方法通过清单配置文件 AndroidMenifest.xml 提取权限特征，然后分析其使用情况或挖掘权限间相关性，以检测恶意应用. 由于权限特征数量较少且提取便捷，使得此类方法出现较早，根据使用权限特征的方式不同，本节将其分为基于权限使用情况和基于权限相关性挖掘的 Android 恶意应用检测方法.

### 2.2 已有工作分析

#### 2.2.1 基于权限使用情况分析的方法

基于权限使用情况分析的方法先提取权限特征，然后将机器学习方法用于检测 Android 恶意应用. 例

如, Sanz 等人[14]提出基于权限的方法来检测 Android 恶意应用, Ilham 等人[15]在此基础上采用信息增益、信息增益率和皮尔逊相关系数等方法选择出权限的特征子集, 然后分别构建随机森林(random forest)、支持向量机(support vector machine, SVM)和决策树(decision tree)模型检测 Android 恶意应用.

Intent[16]是包含目标组件地址和数据的消息对象, 主要用于 Android 应用程序内和应用程序间通信, 可通过发送 Intent 消息对象请求应用内的组件或其他应用执行特定操作, 还可在操作结束后接收返回的数据. Intent 与权限相比语义信息更丰富, 因此除单独分析权限使用情况外, 研究人员还对权限结合 Intent 检测 Android 恶意应用的方法进行了研究. 其中, 杨宏宇等人[17]采用加权投票方法改进随机森林算法, 使用权限和 Intent 作为特征值检测 Android 恶意应用. AndroDialysis[18]首先采用 Intent 作为特征, 取得较好的检测效果, 然后将权限与 Intent 特征结合, 并使用贝叶斯网络来检测 Android 恶意应用, 进一步提升了检测性能. Idrees 等人[19]提出结合权限和 Intent 的集成学习方法 PIndroid, 构造了决策表、多层感知机(multilayer perceptron, MLP)和决策树等 6 种基分类器, 并用 Boosting, Bagging, Stacking 集成学习方法检测 Android 恶意应用, 取得 99.8% 的准确率. Zhang 等人[20]提出基于 N-Gram 分析和在线增量分类器结合的方法, 提取权限、Intent 等多种特征用于 N-Gram 分析, 采用被动攻击(passive-aggressive)算法进行增量学习, 该框架不仅可以检测恶意应用, 还可对恶意应用所属家族进行分类.

#### 2.2.2 基于权限相关性挖掘的方法

Android 有多个权限组, 每个权限组中含有若干个权限. Android 恶意应用进行恶意行为时, 通常需要多个权限的协作和配合, 因此可挖掘权限组间和权限组内的权限相关性, 并将其用来检测恶意应用. 例如, 为修改接收到的短信, 恶意应用需要使用权限组合: 接收短信(RECEIVE_SMS)、读取短信(READ_SMS)和写入短信(WRITE_SMS). 2014 年, Wang 等人[21]通过挖掘单个权限或一组协作权限风险的方式来检测恶意应用. 该方法首先采用互信息、皮尔逊相关系数和 T 检验 3 种特征排序算法对权限进行风险排序; 然后使用序列前向选择(sequential forward selection)方法和主成分分析(principal component analysis)方法选择权限子集, 挖掘出多个权限间协作的风险; 最后构造支持向量机、决策树以及随机森林模型检测恶意应用.

以往大多数研究工作主要关注权限使用情况, 而忽视了挖掘权限之间的关系, 杨欢等人[22]提出了基于频繁模式挖掘算法 PApriori, 挖掘恶意应用家族中权限使用的相互依赖关系, 通过匹配待检测应用的权限特征来检测 Android 恶意应用. 相似地, Arora 等人[23]提出了基于权限对的 Android 恶意应用检测方法 PermPair, 首先分别提取恶意应用集和良性应用集的权限对, 构建恶意应用权限对图(Malicious-Graph)和良性应用权限对图(Normal-Graph), 并根据权限对在不同应用集中出现的频率, 对图中每条边分配权重, 然后通过比较待检测应用权限对在 Malicious-Graph 和 Normal-Graph 中的权重和, 确定其是否为恶意应用. 该方法在权重计算过程中采用了图边缘优化算法, 删除不重要的权限对, 有效减少了检测恶意应用的时间消耗.

### 2.3 已有工作对比和点评

表 1 对基于权限的方法进行了对比. 其中, 数据集来源和数量列采用来源(数量)的形式给出了论文使用的良性和恶意应用来源和数量, 例如, 良性: Google Play(674)表示使用了 Google Play 数据集中的 674 个良性应用. 从表 1 可以看出, 基于权限相关性挖掘的 Android 恶意应用检测方法的准确率整体低于基于权限使用情况分析的方法, 这主要是由于 Android 应用中过度和随意申请敏感权限现象极为常见, 使得噪声更多, 增加了挖掘权限相关性的难度.

基于权限的 Android 恶意应用检测方法的优点是权限特征提取方式简单快捷、特征数量较少、检测效率较高, 其缺点是恶意应用容易申请和良性应用相同的权限, 以伪装成良性应用, 导致检测准确率较低、误报率较高. 常见的改进方法是将权限特征和其他静态特征结合, 以提高 Android 恶意应用检测的精度.

## 3 基于 API 的 Android 恶意应用检测方法

### 3.1 简 述

Android API[24]是 Android 系统提供的函数接口, 恶意应用在获得敏感权限后, 可能通过 API 调用访问和获取手机中的短信、通讯录、定位、相机和相册等敏感数据, 也可能通过恶意操作执行系统破坏、远程控制和诱骗欺诈等恶意行为, 导致用户的隐私和经济安全受到侵害. 良性应用和恶意应用在 API 使用

**Table 1 Comparison of Android Malware Detection Methods Based on Permission**

**表 1 基于权限的 Android 恶意应用检测方法对比**

| 检测方法 | 相关文献 | 发表年份 | 方法/使用算法 | 使用特征 | 数据集来源（数量） | 优劣势分析 |
|---|---|---|---|---|---|---|
| 基于权限使用情况分析 | 文献 [17] | 2017 | IRFCM | 权限、Intent | 良性：Google Play（674）<br>恶意：Genome（1 260） | 使用加权投票方法改进随机森林算法，但未与其他方法进行实验对比. |
| | 文献 [18] | 2017 | AndroDialysis | 权限、Intent | 良性：Google Play（1 846）<br>恶意：Drebin（5 560） | 结合 Intent 和权限特征，考虑了样本类别不平衡问题，但较难抵抗对抗性攻击. |
| | 文献 [19] | 2017 | PIndroid | 权限、Intent | 良性：Google Play（445）<br>恶意：Genome（1 000），Drebin（100） | 结合权限和 Intent 特征，采用集成学习方法优化检测结果，但样本数量较少. |
| | 文献 [15] | 2018 | SVM、随机森林、J48 | 权限 | 良性：Google Play（58）<br>恶意：AMD（673） | 使用特征选择和多种分类器，但特征类型单一且样本数量较少. |
| | 文献 [20] | 2019 | Passive-Aggressive | 权限、Intent、DEX、APK | 良性：Google Play（37 224）<br>恶意：Genome（928），Drebin（5 560），其他（33 259） | 使用 N-Gram 结合在线增量学习检测恶意应用和恶意家族分类，但抗代码混淆能力有待增强. |
| 基于权限相关性挖掘 | 文献 [22] | 2013 | PApriori | 权限 | 良性：Google Play（2 000）<br>恶意：Genome（1 260） | 挖掘恶意应用家族的频繁权限模式，但检测准确率较低. |
| | 文献 [21] | 2014 | SVM、决策树、随机森林 | 权限 | 良性：Google Play（310 926）<br>恶意：Genome（1 260），VirusShare（3 207） | 分析单个权限和协作权限组的风险，采用多种特征选择算法和多种分类器，但恶意应用和良性应用申请相同权限时，会降低该方法准确率. |
| | 文献 [23] | 2019 | PermPair | 权限 | 良性：Google Play（5 993）<br>恶意：Genome（1 264），Drebin（1 744），Koodus（2 975），Contagio（250），PwnZen（300） | 通过构建权限对图检测恶意应用，但检测申请权限较少的恶意应用时准确率较低. |

和调用模式上存在差异，因此可通过分析 API 特征的方式来检测恶意应用. 权限特征设置粒度较粗，导致基于权限的 Android 恶意应用检测方法存在较大的不确定性，而分析 API 调用情况能更全面细致地理解应用特征. 因此，相较于基于权限的方法，基于 API 的 Android 恶意应用检测方法取得了更高的准确性和鲁棒性，是目前使用最广泛的一类方法. 此类方法通过分析反编译后的 Smali 文件，提取 API 调用序列或 API 调用图，以检测恶意应用. 根据所使用 API 信息和方式的不同，本节将其分为基于 API 使用情况、基于 API 调用依赖图和基于 API 与其他特征结合 3 类 Android 恶意应用检测方法.

### 3.2 已有工作分析

#### 3.2.1 基于 API 使用情况的方法

API 使用情况能在一定程度反映应用的特点. 基于 API 使用情况的方法通过提取和分析应用的 API 调用序列和关系，并使用各类机器学习算法检测恶意应用，这类方法取得了较好效果. 例如，Scalas 等人[25]针对勒索应用对 Android 用户的威胁，提出基于 API 检测策略的方法 R-PackDroid，该方法可用于检测具有勒索行为的恶意应用和其他类别的恶意应用. 由于勒索应用攻击严重依赖特定的 API 来执行其操作，可通过包、类和函数 3 类粒度的 API 特征信息进行检测，实验结果表明，基于函数粒度特征的检测方法相比其他 2 种有更高的准确率.

隐私窃取是恶意应用的典型行为之一，常使用数据流分析和污点分析来检测这类恶意行为. 其中，数据流分析方法记录和跟踪敏感数据流路径，包括源点、数据传播路径和交汇点，并分析 API 在数据流路径上的调用序列，以检测隐私数据是否被泄露. Wu 等人[26]提取与数据流相关的 API，并根据每个 API 在恶意应用中的使用情况计算其恶意权重值，在计算出恶意权重值的基础上，使用改进的马氏距离（Mahalanobis distance）方法计算 $K$ 最邻近（K-nearest neighbor, KNN）分类算法中相邻节点的距离，以提高敏感数据分析的效率和模型的检测精度. 随着恶意应用的不断演化，其收集敏感数据的行为更加复杂，仅由源点和交汇点构成的简单信息流难以捕获此类行为. 为此，Shen 等人[27]通过提取由简单信息流集合所组成的复杂信息流路径上的 API 序列，使用 N-Gram 模型生成不同长度的 API 特征向量，然后使用 SVM 算法构造模型以检测恶意应用. 此外，Dexteroid[28]从 Android 生命周期函数调用中提取组件的行为特征、事件调用和状态转换，以检测存在信息泄露和发送恶意短信隐患的恶意应用.

污点分析方法能够追踪应用程序中的隐私数据从获取到泄露的整个传播过程. Feng 等人[29]提出基于 API 语义的方法 Apposcopy，该方法将静态污点分析

和组件间调用图相结合,以检测具有控制流和数据流属性的恶意应用.为提高污点分析效率,张捷等人[30]提出基于污染变量关系图的污点分析方法 FastDroid.除了关注污点分析效率外,大量恶意应用采用代码保护技术,导致污点分析技术存在误报率较高的问题.为此,缪小川等人[31]使用敏感路径识别方法分析 Android 应用的安全性,该方法首先提取组件间函数调用关系图,在图中寻找由敏感行为以及触发该行为的 API 组成的敏感路径,然后对敏感路径进行特征抽象化,以使用决策树模型检测恶意应用.王蕾等人[32]提出面向 Android 应用隐私泄露检测的多源污点分析技术可有效区分出分支互斥路径,提升了检测效率,并且降低了多源污点问题计算开销.

也有研究工作关注到用户输入和行为等敏感操作与 API 调用序列之间存在一定关联关系.2015 年,Elish 等人[33]使用 TriggerMetric 元组来表示用户输入和行为触发的敏感 API 调用特征,TriggerMetric 元组包括用户操作、触发器、依赖路径和 API 调用,可根据应用操作行为的不同区分良性应用和恶意应用.Alam 等人[34]提出了基于优势树(dominance tree)的 API 调用序列挖掘方法 DroidDomTree,改进 TF-IDF 方法为优势树中的节点分配权重,根据权重选择重要的 API 特征,从而提高了检测效率.

3.2.2 基于 API 调用依赖图的方法

随着 Android 版本的升级,以及恶意应用的演化,常使用升级后的 API 实现相似的恶意功能,导致恶意应用检测模型逐渐老化.面对新出现的恶意应用时检测准确率不断下降,需要定期重新收集数据集再次训练,构建 API 调用关系图是减缓模型老化的有效方法.为此,Mariconti 等人[35]提出了基于行为模型马尔可夫链(Markov chain)的恶意应用检测方法 MaMaDroid,将 API 调用图抽象成包和类序列以构建马尔可夫链形式的行为模型,以从中提取特征检测恶意应用.实验结果表明,将使用旧数据集训练的模型用于检测 1 年和 2 年后出现的新恶意应用时,$F1$值分别达到了 86% 和 75%.为进一步缓解模型老化问题,Zhang 等人[36]提出了 APIGraph 框架,通过构建 API 关系图分析 API 升级前后的语义相似性,以减缓模型老化.首先,采用 NLP 语义解析不同版本的 Android API 文档,从中提取 API、权限等实体并构建 API 关系图;然后使用图嵌入方法将图中的实体编码为向量,通过向量之间的差异表示实体间的语义相似性,以将语义相似的实体聚类到不同簇中;最后使用实体簇训练模型以缓解 Drebin 和 MaMaDroid 等 Android 恶意应用检测工具老化速度.相似地,Xu 等人[37]提出了 SDAC 方法,根据现有 API 的贡献来评估新增 API 对恶意应用检测模型的贡献,具体为根据 API 的语义距离对所有 API 进行聚类,在训练阶段创建一个可扩展的特征集,以通过添加检测阶段新增 API 特征的方式来适应 Android 版本差异所带来的 API 变化,实验结果表明,SDAC 的性能相较 MaMaDroid 取得了明显提升.

为提升检测效率,相关研究使用社交网络分析技术检测 Android 恶意应用.例如,Wu 等人[38]提出了基于图的轻量级检测方法 MalScan,将函数调用图视为社交网络,采用社交网络中心性分析获取图的语义特征,相比 Drebin 和 MaMaDroid 大幅度提升了检测速度.但是当恶意应用行为与良性应用行为相似时,MalScan 会出现漏报,为此 Zou 等人[39]提出了 IntDroid,在社交网络中心性分析的基础上,通过计算社交网络中敏感 API 和中心节点的平均亲密度来表示图的语义特征,以适当降低扩展性为代价取得了比 MalScan 更高的准确率.Wu 等人[40]将函数调用图中的恶意部分节点仅占一小部分的恶意应用称为隐蔽恶意应用.隐蔽恶意应用的函数调用图中良性部分和恶意部分具有强相关性且所占比例较小,导致 IntDroid 面对隐蔽恶意应用时检测性能下降.为此,他们提出了基于社交网络同质性分析的方法 HomDroid 检测隐蔽 Android 恶意应用,实验结果表明 HomDroid 检测隐蔽恶意应用性能优于 Drebin, MaMaDroid, IntDroid 等方法.随着对 Android 应用研究的深入,恶意应用检测方法取得较大进展,基于 API 调用依赖图的方法取得了较好的检测性能.然而,研究发现通过扰乱特征向量等方法可生成成功绕过检测的恶意应用.例如,Zhao 等人[41]提出了 Android 恶意应用对抗性攻击的方法,使用强化学习修改对函数调用依赖图,通过插入节点、删除节点、添加边和重写的方式适应程序的操作(插入方法、删除方法、添加调用关系和重写),提高对抗性攻击的有效性.针对精心设计的对抗性攻击可能会绕过恶意应用检测模型的问题,Demontis 等人[42]提出了对抗规避攻击的恶意应用检测方法,通过对不同特征分配不同权重的方式来提高模型的鲁棒性.

随着深度学习技术在各个领域取得良好成绩,研究人员也开始尝试使用深度学习检测 Android 恶意应用[43-44].卷积神经网络(convolutional neural network, CNN)仅能处理欧氏空间数据如图像、文本等,不支持非欧空间数据,如图数据[45].而图数据能够更准确

地表达 Android 应用和 API 之间调用关系，Gao 等人[46]提出使用图卷积神经网络（graph convolutional network, GCN）检测 Android 恶意应用的方法 GDroid，将应用和 API 映射为异构图，恶意应用检测任务转换为节点分类任务，该方法以应用（APP）、API 间的调用关系和 API 的使用模式构建 APP-API 和 API-API 为边的异构图，并将异构图输入图卷积网络模型以检测恶意应用. 相似地，Li 等人[47]也提出使用图卷积网络检测恶意应用的方法，该方法首先提取 API 调用序列生成有向循环图，然后使用马尔可夫链和主成分分析法提取图的特征，并基于图卷积网络检测恶意应用. 除此之外，AMalNet[48]还采用 GCN 结合独立循环网络（independently recurrent neural network, IndRNN）的方式检测 Android 恶意应用.

为了减少对图的分析，提升检测效率，研究人员将 API 调用图化分为多个子图集合. $S^3$Feature[49]在函数调用图中标记敏感节点形成敏感函数调用图，然后从中挖掘出敏感子图（SSG）及其相邻子图（NSG），最后将 SSG 和 NSG 去重后编码为特征向量用于恶意应用检测. Fan 等人[50]提出了基于频繁子图的 Android 恶意应用检测方法 FalDroid，通过构建同一家族代表性应用的频繁子图的方式，将敏感 API 调用关系图划分为子图集合，以减少图分析复杂度，提升检测恶意应用效率. 类似地，Lu 等人[51]在提取函数调用图后，从图中删除敏感 API 节点距离大于 3 的节点以简化函数调用图，构建去噪图卷积神经网络（denoising graph convolutional neural network），提升了恶意应用检测性能.

除了文献 [49−51] 构建 API 调用关系图的方法外，研究人员[52]还使用 API 调用上下文信息来提高检测精度. Allen 等人[53]对 API 调用上下文信息的有效性进行了全面研究，发现 API 调用的入口点对分类正确性影响较大，并提出了轻量级上下文感知系统 PIKADROID，以用于检测恶意应用. Zhang 等人[54]提出基于 API 语义依赖图的方法来检测 Android 恶意应用，并实现了原型系统 DroidSIFT，该方法首先提取上下文加权 API 依赖图作为程序语义来构造特征集合，然后根据图相似性来检测恶意应用. 另外，MKLDroid[55]从应用程序的依赖图中捕获结构和上下文信息，以在依赖图中定位细粒度的恶意代码.

同一家族应用程序具有类似的功能，可通过分析它们共同特征的相似性检测恶意应用. 受推荐系统的启发，Frenklach 等人[56]提出了利用应用程序相似性图来检测 Android 恶意应用的方法，在该方法中，应用程序对应被推荐的项目，功能对应用户. 该方法首先提取并处理 API 调用图，从应用程序和功能的二分网络中生成应用相似性图（APP similarity graph, ASG）；然后采用 Node2Vec 图嵌入方法将 ASG 转为特征向量，以使用随机森林模型检测恶意应用. 具有共同特征的恶意应用可能属于同一个恶意应用家族，Karbab 等人[57]据此在 Cypider[58]的基础上提出基于应用程序相似性网络的检测方法，该方法首先构建应用程序的相似性网络，从相似性网络中提取具有高连通性的子图作为社区，然后为每个社区生成单个社区指纹，最后使用 SVM 对社区指纹进行分类，以将恶意应用归类为不同家族.

#### 3.2.3 基于 API 与其他特征相结合的方法

研究人员还采用 API 与其他特征相结合的方式，进一步提升检测 Android 恶意应用的性能. 典型工作如 Arp 等人[7]提出的 Drebin 在 API 的基础上，结合了权限、Intent、各类组件、硬件组件和网络地址等特征以检测恶意应用，该项工作还共享了恶意应用数据集，Drebin 方法成为了后续工作的常用比较对象. 2019 年，Badhani 等人[59]提出 CENDroid 方法，该方法首先提取 API 和权限特征；然后分别创建 API、权限及 API 和权限的 3 组特征集；其次采用决策树、极限学习机（extreme learning machine, ELM）、逻辑回归（logistic regression, LR）、RIPPER 和 SVM 这 5 种分类器分别使用 3 组特征集进行恶意应用检测，实验结果表明 API 与权限组合的特征有更高的检测性能；最后，通过集成学习将 5 种分类器结合，其各项评估指标均优于单个的分类器，提高了检测 Android 恶意应用的性能. HybriDroid[60]同样使用多数投票等集成学习方法结合径向基函数（radical basis function network, RBF）网络、人工神经网络（artificial neural network, ANN）和逻辑回归等多个分类器，并结合 API、权限和元数据等特征，使得检测 Android 恶意应用准确率达到了 98.8%. 针对如何学习有效的特征表示，Zhu 等人[61]通过无监督特征学习算法从数据集中学习有效的特征，以降低对 Android 先验知识和人工选择特征的依赖.

由于 API 和其他特征结合后会导致特征数量增多，样本在特征空间分布更加稀疏，容易造成过拟合等问题，影响模型的检测时间和精度，因此，研究人员尝试通过特征选择方法优化模型性能. Cen 等人[62]将 API 与权限相结合作为特征，采用信息增益（information gain, IG）和卡方检验（chi-square test, CHI）进行特征选择，采用基于正则化 Logistic 回归的概率判别模型来检测 Android 恶意应用. Peynirci 等人[63]提出基于 IDF

值的特征选择算法以检测恶意应用,该方法首先提取权限、API 和字符串 3 种特征,根据恶意应用出现最多和良性应用出现最少的特征,计算特征 IDF 差值并选择前 5% 或 10% 的特征,然后使用 KNN、SVM、Native Bayes、J48、随机森林等模型检测 Android 恶意应用. Kong 等人[64]将孪生卷积神经网络(siamese convolutional neural network, SCNN)用于检测恶意应用,该方法首先使用特征分组(feature grouping)策略对 API 和权限进行特征选择,然后构建和训练 SCNN 模型,通过相似度距离方法计算出良性应用和恶意应用的均值并将其作为中心点,最后计算待检测应用与 2 个中心点的距离以确定应用类别. 注意到现有部分工作忽视了特征间的重要性差异, Cai 等人[65]提出了基于特征加权联合优化的 Android 恶意应用检测方法 JOWMDroid, 该方法提取系统资源、权限、组件、Intent、受限 API、敏感 API、权限调用 API 和敏感 Shell 命令 8 类特征,然后使用 IG 对其进行特征选择,并使用 5 种权重映射函数计算特征权重,最后通过集成学习结合 SVM, LR, MLP 这 3 类模型提升恶意应用检测精度. 文献 [62−65] 采用了特征选择的方法,避免了特征“维度爆炸”,在减少模型训练时间和检测时间的同时,提高了模型准确率.

除此之外, Feng 等人[66]提出了 MobiTive 检测系统,该方法结合了 API、权限、Intent 和硬件特征,并比较了 CNN、长短期记忆(long short-term memory, LSTM)和门控循环单元(gated recurrent unit, GRU)这 3 种模型的准确率,其中 GRU 模型取得了最好效果, MobiTive 还可对 Android 恶意应用家族进行归类. 类似地, Garcia 等人[67]提出了 RevealDroid 方法,该方法提取 API、反射特征和 Native 代码作为特征,不仅提升了检测 Android 恶意应用的准确性和效率,还可适用于经过代码混淆处理的应用. 另外, Hei 等人[68]提出了快速检测新应用的框架 HAWK,将实体(包含 API、权限、类、接口和 so 文件等)和行为关系建模为异构信息网络,并结合改进的图注意力网络模型检测新应用.

### 3.3 已有工作的对比和点评

表 2 对部分具有代表性的基于 API 的恶意应用检测方法进行了对比. 从表 2 的分析可以看出,基于 API 使用情况的方法假设各特征间相互独立,或仅考虑数据流在 API 之间的传播路径,不考虑 API 间复杂的相关性,虽然具有方便快捷的优点,但是大量冗余的 API 特征会增加算法时间消耗.

通过 API 调用依赖图来抽象 API 之间的调用关系,可利用图结构进行分析以识别恶意应用,但构建 API 调用图存在较大挑战. 首先, API 调用基于事件监听与回调,可能无法正确表示调用方法的顺序;然后,应用的不同部分通过组件间通信来进行交互,这部分行为在代码层面没有直接调用关系,因此构建的应用 API 调用依赖图可能不够完整;最后, API 调用依赖关系的复杂性导致所生成的图较为复杂,使得基于图分类的方法时间消耗较多.

API 与其他特征结合的方法通过结合多种特征提升了恶意应用检测精度,但通常需要使用有效的特征选择算法以选取更有代表性的特征子集.

## 4 基于操作码的 Android 恶意应用检测方法

### 4.1 简 述

研究发现,部分恶意应用采用代码混淆技术以规避检测,降低了基于权限和 API 检测 Android 恶意应用方法的有效性[69]. 代码混淆技术在一定程度上改变了应用的程序结构,会引起程序的信息流等变化,但应用的核心操作逻辑通常不会发生较大变化. 作为程序执行过程中的指令,操作码可表征应用的行为特征,针对方法名、变量名、字符串和权限的混淆技术对操作码特性影响较小. 因此,可将操作码信息作为特征来检测恶意应用. 基于操作码的 Android 恶意应用检测方法首先通过反编译提取 Smali 文件中的操作码序列,然后对操作码序列进行分析处理,以检测恶意应用. 根据所使用操作码信息的方式不同,本节将介绍基于操作码使用情况、基于操作码序列转化图像、基于操作码和其他特征的 3 类 Android 恶意应用检测方法.

### 4.2 已有工作分析

#### 4.2.1 基于操作码使用情况的方法

操作码可用于表征应用程序的行为模式,很多研究工作在提取操作码序列后,使用统计模型或机器学习方法来检测 Android 恶意应用. 例如,陈铁明等人[70]提出了基于 N-Gram 模型的 Android 恶意应用静态检测方法 Maldetect, 该方法首先使用逆向工程提取操作码指令并将其抽象为指令符号,然后使用 N-Gram 模型将指令符号序列编码为特征,最后构造随机森林、SVM、KNN、Naive Bayes 模型来检测恶意应用. 基于 N-Gram 模型的方法对操作码数量较为敏感,如果 N-Gram 模型中操作码较少,无法涵盖足够信息来检测恶意应用;而如果 N-Gram 模型中操作码较多,特征维度过大,则会引入过多噪声,导致更高的训练时间成本. 与统计模型相比,深度学习方法可自动提

**Table 2 Comparison of Android Malware Detection Methods Based on API**

**表 2 基于 API 的 Android 恶意应用检测方法对比**

| 检测方法 | 相关文献 | 发表年份 | 方法/使用算法 | 使用特征 | 数据集来源（数量） | 优劣势分析 |
|---|---|---|---|---|---|---|
| 基于 API 使用情况 | 文献 [33] | 2017 | TriggerMetric, DPVC | API | 良性：Google Play（2 684）<br>恶意：Genome（1 433） | 分析了用户输入/行为和敏感 API 调用之间的静态依赖关系，但是忽视了部分无需用户特殊操作就能触发的恶意应用. |
| | 文献 [27] | 2018 | N-Gram, SVM | API | 良性：Google Play（4 699）<br>恶意：Genome（800），Intelligence Company（3 899） | 分析了复杂信息流路径上的 API 调用序列，但是忽视了部分不在复杂信息流路径中的 API 调用. |
| | 文献 [25] | 2019 | R-PackDroid | API | 良性：Google Play（18 396）<br>恶意：VirusTotal（17 744），Drebin（5 560） | 通过多粒度特征检测勒索应用，但依赖于特定的 API 调用，难以应对版本升级导致的 API 变化. |
| | 文献 [35] | 2019 | MaMaDroid | API | 良性：PlayDrone（5 879），Google Play（2 568）<br>恶意：Drebin（5 560），VirusShare（29 933） | 根据应用程序执行的抽象 API 调用图构建马尔可夫链形式的行为模型，但未考虑运行环境和用户输入等影响. |
| | 文献 [37] | 2020 | SDAC | API | 良性：Androzoo（35 646）<br>恶意：Androzoo（34 496） | 根据 API 语义距离进行聚类，创建可扩展特征集，以适应版本差异所带来的 API 变化，但聚类会损失部分语义信息. |
| 基于 API 调用依赖图 | 文献 [36] | 2020 | APIGraph | API、权限 | 良性：Google Play（290 505）<br>恶意：VirusShare（32 089） | 分析恶意应用演化过程中的语义相似性，减缓模型老化速度，但是未考虑代码混淆影响. |
| | 文献 [48] | 2020 | AMalNet | API、权限 | 良性：Androzoo（5 000）<br>恶意：Drebin，AMD，Lab-built（5 000） | 提取多种特征，并结合 GCN 和 IndRNN 构建模型，但图神经网络的计算效率还有待提高. |
| | 文献 [46] | 2021 | GDroid | API | 良性：Google Play（2 100）<br>恶意：AMD（1 200） | 将应用和 API 构建为异构图，使用 GCN 检测恶意应用，但是实验数据集较小且未考虑模型的可解释性. |
| 基于 API 和其他特征结合 | 文献 [7] | 2014 | Drebin | API、权限、硬件组件、Intent、网络地址 | 良性：Google Play（96 150），Chinese Market（19 545），Russian Market（2 810），其他（13 106）<br>恶意：Drebin（5 560） | 结合多种特征的轻量级检测方法，但使用特征数量较多，未进行特征选择. |
| | 文献 [63] | 2020 | Delta_IDF | API、权限、字符串 | 良性：Google play（5 900）<br>恶意：Genome，AndroZoo，Drebin（8 900） | 通过 IDF 差值选择特征，但未考虑混淆技术带来的影响. |
| | 文献 [66] | 2020 | MobiTive | API、权限、Intent、硬件特征 | 良性：Google Play（18 000）<br>恶意：Drebin（3 561），Genome（1 009），Contagio（198），Pwnzen（572），VirusShare（12 660） | 采用定制化深度神经网络，可部署在移动设备中，但易受移动设备性能影响. |
| | 文献 [61] | 2021 | SHLMD | API、权限 | 良性：Google Play（15 098）<br>恶意：VirusShare（2 799） | 设计无监督算法学习有效特征，降低了对人工或先验知识的依赖，但检测准确率较低. |
| | 文献 [65] | 2021 | JOWMDroid | 权限、组件、Intent、敏感 API | 良性：Google Play（3 000）<br>恶意：VirusShare（5 000） | 结合多类特征并进行特征选择，通过集成学习提升检测精度，但未考虑特征间的关联性. |

取特征并组织特征之间的关系. 例如, McLaughlin 等人[71] 提取由 218 个操作码组成的多个序列作为文本集, 采用自然语言处理的方法, 构建卷积神经网络模型检测恶意应用, 并和基于 N-Gram 模型的方法进行比较, 实验结果表明, 卷积神经网络模型在效率和性能均优于基于 N-Gram 的检测方法.

李挺等人[72] 提出了基于 Dalvik 指令的 Android 恶意代码检测方法, 该方法首先对恶意应用中操作码指令集进行精简以保留反映程序语义的指令, 并对其进行形式化描述以建立恶意特征库, 然后使用软件相似度度量算法(MOSS)和闵可夫斯基距离算法(Minkowski distance)计算待检测应用恶意应用特征库的相似度结果来判定其是否为恶意应用.

机器学习方法通常需要类别较为平衡的训练样本集才能生成性能较好的检测模型. 然而在真实场景下样本类别存在严重不平衡, 导致模型检测性能较低. 为此, Mercaldo 等人[73] 采用进程代数模型表示 Android 应用, 将字节码转换为进程, 并通过等价性检验(equivalence checking)来分析所生成进程代数模型之间的关系, 该方法不需要样本标签, 即可检测恶意应用及其所属家族, 从而解决样本类别不平衡的问题.

鉴于 Windows 平台上的应用同样由操作码序列

组成,有研究人员将 Windows 平台基于操作码序列检测恶意应用的方法移植到 Android 平台上. 例如, Canfora 等人[74]评估了使用基于操作码特征的隐马尔可夫模型(hidden Markov model, HMM)和结构熵 2 种方法检测 Android 恶意应用的性能. 该研究表明基于 HMM 的方法可有效检测 Android 恶意应用,而基于结构熵的方法更适合对 Android 恶意应用家族分类.

4.2.2 基于操作码序列转化图像的方法

大多数恶意应用的变种往往通过自动化技术重用携带恶意代码模块的方法生成,因此它们之间的操作码序列具有一定的相似性. 部分研究人员将操作码序列转化为图像使其可视化,然后采用图像识别的方法以识别原始样本的变体来检测 Android 恶意应用. 典型工作为 Xiao 等人[75]将 Dalvik 操作码和操作数组成的字节码文件转换为 RGB 图像,该方法首先以十六进制格式读取字节码,然后将 3 个连续的十六进制数分别转换为 RGB 通道的 3 个值,将字节码序列转换为颜色矩阵,形成图像文件,最后将图像输入 CNN 模型进行分析,检测是否为恶意应用. 在此工作基础上, Yadav 等人[76]对多种 CNN 模型进行测试,结果表明采用 EfficientNetB4 结构的 CNN 模型优于 MobileNetV2 和 ResNet50 等结构的模型.

代码混淆技术会绕过部分 Android 恶意应用检测系统,降低现有方法的有效性. 为此, Tang 等人[69]提出了基于多粒度操作码特征的 Android 恶意应用检测方法 MGOPDroid,首先针对不同的混淆技术提取不同粒度的操作码特征,通过 TF-IDF 方法和混淆前后操作码特征的差异计算特征权重,以选择有效的抗混淆操作码特征,然后将操作码序列特征转为灰度图像,作为 ResNet 模型的输入,最后把训练好的模型,通过 TensorFlow Lite 部署在移动设备,实现本地检测.

注意到已有基于深度学习的恶意应用检测方法缺乏可解释性, Iadarola 等人[77]对检测恶意应用的 CNN 模型进行了可解释性分析. 首先,应用反编译,并将 DEX 字节码转换为灰度图,以训练 CNN 模型;然后,使用 Grad-CAM 生成热图,对 CNN 模型的输出进行解释,并按恶意应用家族分类热图,以生成代表家族特性的累积热图.

部分研究工作借鉴了检测 Windows 平台恶意应用的方法来检测 Android 应用,并取得了良好的效果. 例如, Yuan 等人[78]提出了基于马尔可夫图像(Markov image)和深度学习的方法 MDMC,将提取的字节码经概率转移矩阵转换成 Markov 图像,通过深度卷积神经网络的方法对图像进行恶意应用检测. 该方法不仅适用于检测 Android 恶意应用,也适用于检测 Windows 平台恶意应用.

4.2.3 基于操作码和其他特征相结合的方法

为进一步提升 Android 恶意应用检测性能,研究人员还结合了操作码、权限和 API 等多种特征来检测恶意应用. 张炳等人[79]提出了面向概念漂移的可解释性 Android 恶意应用检测方法 InterDroid,首先根据 Android 恶意应用报告分析和提取了 API、权限、Intent 和字节码,然后使用自动化机器学习算法 TPOT(tree-based tipeline optimization tool)筛选最佳分类模型集合,并使用集合中的模型和特征构建特征解释器,最后基于联合分布适配算法提高检测准确率. Qiu 等人[80]提出的多视图特征智能检测框架 MFI,从已知恶意应用家族中学习功能表示,以识别具有相同功能的新恶意应用. 具体而言,提取权限、API 调用图和操作码等异构特征,通过特征分析、选择、聚合和编码等过程,构建深度神经网络(deep neural network, DNN)检测模型检测恶意应用. 实验结果表明, MFI 优于 Drebin 和 MaMaDroid 等方法. 陈波等人[81]提出基于多层 SimHash 的相似度检测方法,首先从 APK 文件中提取的 AndroidManifest.xml、Smali 代码集、操作码指令集、Java 代码集、Java 指令集 5 类特征表征应用,然后采用 SimHash 算法进行相似度分析,并分别用于构建一层数据模型,在检测过程中使用投票感知器(voted perceptron)算法,为每层分别赋予信任权重值,最后将每层结果结合权重进行投票,根据投票结果判断是否为恶意应用. 实验结果表明该方法有较好的检测效果. Kim 等人[82]提出多模态深度学习方法检测恶意应用,通过提取操作码、API、权限、组件、字符串、共享库函数和环境 7 类特征,以构建深度神经网络 DNN 模型检测恶意应用.

Lu 等人[83]设计了 DLAMD 框架,使用预检测和深度检测 2 个阶段检测 Android 恶意应用,预检测阶段提取权限特征并使用 BP 神经网络模型初步筛选具有明显特征的恶意应用,以降低检测时间和计算成本,提高整个框架的检测效率. 深度检测阶段提取应用的操作码结合 CNN 和 LSTM 模型,以进一步检测疑似恶意应用,提升检测准确性.

### 4.3 已有工作对比和点评

表 3 将部分基于操作码的代表性方法进行了对比. 其中,基于操作码使用情况方法的优点是可以有效避免因代码混淆而提取不到重要特征的问题,也无需人工选择具体特征. 基于操作码转化图像的方法将 Android 恶意应用检测的问题转为图像分类问题,可

**Table 3 Comparison of Android Malware Detection Methods Based on Opcode**

**表 3 基于操作码的 Android 恶意应用检测方法对比**

| 检测方法 | 相关文献 | 发表年份 | 方法/使用算法 | 使用特征 | 数据集来源（数量） | 优劣势分析 |
|---|---|---|---|---|---|---|
| 基于操作码使用情况 | 文献 [72] | 2014 | MOSS | 操作码 | 良性：Google Play（7 700）<br>恶意：Genome（300） | 对操作码指令集进行形式化描述以简化操作码特征，但提取的特征数量和涵盖的恶意代码种类较少. |
| | 文献 [70] | 2016 | Maldetect | 操作码 | 良性：Google Play（2 000）<br>恶意：Drebin（2 000） | 将操作码简化为指令符号，使用 N-Gram 进行特征编码，但对操作码数量较为敏感. |
| | 文献 [74] | 2016 | Random Forest，NBTree | 操作码 | 良性：Google Play（5 560）<br>恶意：Drebin（5 560） | 使用 HMM 和结构熵处理操作码特征，构建多种模型检测恶意应用，但对已知的恶意应用家族依赖性较强. 且 HMM 容易过时，检测新应用性能会下降. |
| | 文献 [71] | 2017 | CNN | 操作码 | 良性：Google Play（9 268）<br>恶意：McAfee Labs（9 902） | 基于操作码序列构建 CNN，但检测新样本准确率较低. |
| 基于操作码序列转图像 | 文献 [75] | 2019 | CNN | 操作码和操作数组成的字节码 | 良性：Google Play（4 406）<br>恶意：AMD（6 134） | 将字节码序列转换为图像，以将恶意应用检测转化为图像分类问题，但对通过重打包注入恶意组件的应用检测准确率较低. |
| | 文献 [78] | 2020 | MDMC | 操作码和操作数组成的字节码 | 良性：未使用<br>恶意：Drebin（4 020） | 将字节码生成 Markov 图像使用 CNN 模型检测恶意应用，但未考虑代码混淆技术的影响. |
| | 文献 [69] | 2021 | MGOPDroid | 操作码和操作数组成的字节码 | 良性：Market（4 621）<br>恶意：Drebin（5 560） | 采用特征差异计算选择抗混淆操作码特征，并将操作码序列转为灰度图像构建 ResNet 模型，但未考虑加壳和动态加载技术的影响. |
| | 文献 [77] | 2021 | CNN | 操作码和操作数组成的字节码 | 良性：Google Play（1 060）<br>恶意：AMD （7 386） | 使用字节码生成灰度图训练 CNN 检测恶意应用，并通过 Grad-CAM 生成热力图进行解释，但无法检测到不在训练集中的恶意家族，且检测性能依赖于数据集数量. |
| 基于操作码和其他特征结合 | 文献 [82] | 2018 | DNN | API、权限、操作码、组件、字符串、共享库函数和环境 | 良性：Google Play（20 000）<br>恶意：VirusShare（20 000），Genome（1 260） | 提取操作码等多类特征，使用多模态深度学习检测恶意应用，但未考虑混淆技术的影响. |
| | 文献 [79] | 2021 | InterDroid | 操作码、API、权限、Intent | 良性：OmniDroid（5 900）<br>恶意：OmniDroid（5 900） | 提取操作码等多类别特征，使用 TPOT 算法筛选模型，并构建特征解释器，但自动化程度较弱，在特征选择阶段需要人工参与. |
| | 文献 [83] | 2021 | DLAMD | 权限、操作码 | 良性：360 Application Market（3 900）<br>恶意：VirusShare，Drebin（3 900） | 结合 CNN 和 LSTM 模型，通过预检测和深度检测 2 个阶段检测恶意应用，但未考虑动态特征. |
| | 文献 [80] | 2022 | MFI | 操作码、API 调用图、字符串 | 良性：未使用<br>恶意：Drebin（5 560），AMD（24 553） | 使用操作码及 API 调用图等多种异构特征，构建 DNN 模型检测恶意应用，但检测性能依赖于已知的恶意应用家族数据. |

利用已有的图像分类算法，由于其缺点是可解释性较差，且对计算机硬件和算力要求较高；基于操作码和其他特征结合的方法结合多种特征，提升了恶意应用检测精度，其缺点是模型设计较为复杂且检测时间较长.

## 5 基于其他特征的 Android 恶意应用检测方法

### 5.1 简 述

除了上述权限、API 和操作码等常用的静态特征方法外，为应对代码混淆、恶意代码注入所带来的挑战，有研究通过结合动静态特征和相似代码片段等方法来检测 Android 恶意应用.

### 5.2 已有工作分析

#### 5.2.1 基于动静态特征结合的方法

Android 恶意应用动态检测指分析 Android 应用是在严格控制的环境（真实设备或模拟器）中利用自动化脚本模拟用户操作，执行如 API 调用、网络通信、资源访问、进程控制等操作，并通过对应用行为分析来判断其是否存在隐私窃取、恶意扣费、远程控制等恶意行为. 由于 Android 应用是通过事件交互驱动的，恶意应用行为往往与交互方式相关，在动态检测中如果没有发现恶意行为，并不能确保之后不会发生，而静态分析可以分析更多的可执行路径，但难以应对代码混淆加密的问题. 因此，有研究工作通过将动态特征与静态特征相结合的方式来解决上述问题.

杨欢等人[84]提出了静动态结合的方法 Androdect，其使用的静态特征包括权限、各类组件、API 信息和 Native 代码，在模拟器中使用 Monkey[85]模拟用户行为，以收集动态特征，并设计了 3 层混合系综算法（triple hybrid ensemble algorithm, THEA），使用决策树、

朴素贝叶斯和 KNN 等多种机器学习模型的最优分类器来检测恶意应用.

Alzaylaee 等人[86]提出了基于深度学习的恶意应用检测方法 DL-Droid, 可适用于真实设备. DL-Droid 使用权限、API 和 Intent 等静态特征和运行日志等动态特征来训练检测恶意应用的深度学习模型, 并与 SVM, SVM RBF, NB, LR, PART, RF, J48 等主流算法进行比较, 实验结果表明 DL-Droid 具有更好的性能.

网络流量特征可用于检测具有恶意代码更新功能和泄露隐私的应用, 可与静态特征相结合检测恶意应用. Arora 等人[87]提出了基于权限特征和网络流量特征的 Android 恶意应用检测方法, 该方法将 KNN 算法和 K-Medoids 算法相结合, 实验结果表明, 该方法优于仅使用动态流量特征或静态权限特征的检测方法. 大多数恶意应用窃取隐私数据后将其发送到服务端, Tong 等人[88]先通过动态运行应用提取文件调用和网络访问相关的特征, 分别构建良性应用和恶意应用模式集, 以基于模式匹配检测恶意应用. Wang 等人[89]通过挖掘应用访问的 URL 以检测恶意应用, 首先将收集到的应用在模拟器中运行, 并收集应用的网络访问数据, 然后提取其中的 URL 信息并将其向量化, 以训练用于检测恶意应用的神经网络模型.

此外, 权限、组件和 API 调用等静态特征也可以和动态行为特征组合以检测恶意应用[90-92]. 例如, 李鹏伟等人[93]对 APK 进行脱壳重打包后提取动静态特征, 其中静态特征包括应用的结构信息、权限信息、加固情况、反射、Native 代码、权限和类信息, 动态特征为通过插装获取的行为序列. Costa 等人[94]提出了污点分析和动态特征结合的检测方法, 主要对敏感 API 调用特征进行分析. 另外, Yuan 等人[95]提出了深度学习检测的方法 Droid-Sec, 该方法提取静动态特征采用深度置信网络检测恶意应用. 在动态运行过程中, Android 应用在调用分配内存、访问文件等 API 函数时, 会转换为相应的系统调用, Amer 等人[96]通过结合 API 调用、系统调用和权限序列特征, 构建 LSTM 模型, 捕获应用运行时 API 调用和系统调用间的依赖关系, 以检测恶意应用.

### 5.2.2 基于相似代码片段的方法

为规避检测, Android 恶意应用开发人员可对良性应用反编译, 注入恶意组件(例如存在恶意代码的第三方库), 修改控制流以确保其正常运行后, 再重新打包上架至第三方应用市场. 通过这种方式构造的恶意应用与其对应的良性应用行为相似, 难以检测. 由于这些恶意应用中注入的恶意代码存在相似性, 有研究通过基于相似代码片段的方法检测 Android 恶意应用.

重打包检测方法主要通过各种相似度度量指标找到高度相似的应用. 例如, 为消除第三方库带来的影响, Zhang 等人[97]在特征提取阶段根据应用白名单移除通用第三方库后, 采用 Jaccard 指数分析应用剩余部分的代码相似度, 然后根据代码相似度、应用程序的元数据和商业防病毒软件提供的标签 3 类异构信息构建网络表示学习以对弱标签恶意应用聚类划分为已知和未知的家族, 最后构建 3 层 DNN 来进一步分类已知家族的应用. Tian 等人[98]提出了代码重打包检测方法 DR-Droid, 该方法将应用代码的结构根据其行为依赖关系划分为多个代码子集, 在不同的代码子集中提取了粗粒度的类级依赖图和细粒度的方法调用图来生成应用的特征向量. 除常见的 Java 库隐藏恶意代码外, 部分恶意应用将恶意行为隐藏在 Native 代码中, 因此, Alam 等人[99]提出了 DroidNative 以检测字节码或 Native 代码中的恶意行为. Zhan 等人[100]对 Android 第三方库的检测工具进行了实证研究, 从恶意应用的检测性能、检测效率、代码混淆恢复能力和易用性 4 方面分析和评估了现有的检测工具. 结果表明, LibScout[101]有更好的检测性能, LibRadar[102]时间消耗少且更易用, Libeccio[103]抗代码混淆效果最好.

He 等人[104]提出了基于恶意代码片段识别的检测方法 MsDroid, 使用 GNN 检测 Android 恶意应用. MsDroid 使用调用图、操作码和权限信息, 把 Android 应用中的敏感行为建模为行为子图集(behavior subgraph set), 通过查找相似恶意应用代码片段检测恶意应用. Chen 等人[105]使用克隆代码检测 Android 恶意应用, 该方法分为签名生成和签名匹配 2 个阶段. 签名生成阶段使用代码检测工具 NiCad[106]查找恶意应用集中的克隆类, 通过基于相似子集的方法将不同的恶意应用聚类成不同的集合, 并取每个克隆类的代表性示例作为这个类的签名, 构成签名集合. 签名匹配阶段以增量模式使用 NiCad 在签名集合中匹配良性应用和恶意应用克隆示例. Meng 等人[107]提出了基于确定性符号自动机的检测模型, 通过总结恶意应用家族的代表性行为特征来检测恶意应用的变种.

### 5.2.3 其他静态检测方法

此外, 还有一些利用签名、Intent、APK 文件等其他静态特征的 Android 恶意应用检测方法.

应用签名相当于程序的唯一标识, 用于应用编译打包后的身份验证, 刘新宇等人[108]提出了基于 APK 签名信息反馈的检测方法, 根据对 APK 签名的验证

和筛选来检测恶意应用. 现有部分方法捕获应用程序和 Android 系统的交互, 而 Xu 等人[109] 关注应用程序内部或应用程序间的通信, 提出 Intent 组件检测方法 ICCDetector.

研究人员除了将操作码序列转图像外, 还可将 Android 可执行文件 APK 转为图像进行检测, Pinhero 等人[110] 提出了基于恶意应用可视化和深度学习的恶意应用检测方法, 将 APK 转为灰度、RGB 和 Markov Image, 然后使用 Gabor 滤波器进行纹理分析, 最后将图像作为输入提供给 12 个不同的神经网络进行检测.

### 5.3 已有工作对比和点评

表 4 对部分具有代表性的基于其他特征的 Android 恶意应用检测方法进行了对比. 由于部分论文未提供数据集来源, 在数据集来源和数量列仅给出了应用数量. 动静态结合方法的优点是可检测代码混淆加密等代码保护技术和代码热更新技术的恶意应用, 提升了检测性能; 缺点是由于应用在模拟器或真实设备中运行以收集动态特征, 导致时间消耗增大且鲁棒性较差. 基于相似代码片段检测的方法可用于识别重复和相似的代码, 优点是可识别具有相似恶意代码模块的应用, 可有效对恶意应用所属家族进行归类, 缺点是检测未知类型的恶意应用误报率较高.

## 6 数据集与评价指标

在评估 Android 恶意应用检测方法的有效性时, 需要采用合理评价指标, 并使用有代表性的 Android 应用数据集. 本节从 2 方面出发, 总结常用的 Android 应用数据集和评价指标.

### 6.1 数据集

Android 恶意应用检测研究离不开有效的数据集支撑. 在表 1～4 给出了此前研究中常用的良性应用和恶意应用数据集. 其中, 大多数研究人员使用的良性应用数据集来自 Google Play 应用商店. Google Play 中的应用在上架发布前会经过 Google Play Protect 的安全检测, 虽然可能仍会含有恶意应用, 但是数量极少, 通常可认为是良性应用. 由于 Google Play 应用商店的应用无法直接下载, 爬取也较为困难, Androzoo[111] 平台收集了大量的应用, 包括 Google Play 应用商店的应用, 可作为良性应用数据集备选方案.

早期对 Android 恶意应用的研究主要通过研究人员自行收集数据集, 以对所提出方法的有效性进行评估. Genome, Drebin, AMD 是 Android 恶意应用检

**Table 4 Comparison of Android Malware Detection Methods Based on Other Features**

**表 4 基于其他特征的 Android 恶意应用检测方法对比**

| 检测方法 | 相关文献 | 发表年份 | 方法/使用算法 | 使用特征 | 数据集来源（数量） | 优劣势分析 |
|---|---|---|---|---|---|---|
| 基于动静态结合 | 文献 [84] | 2014 | Androdect | API、组件、动态 | 良性:（1 126）<br>恶意:（2 000） | 提取多类静态特征，并使用 Monkey 收集动态特征，结合 THEA 检测恶意应用，但动态特征较为单一. |
| | 文献 [88] | 2017 | Patterns Set | 网络流量、API | 良性：Xi'an JiaoTong University（1 000）<br>恶意：Genome（1 000） | 结合文件访问调用和网络流量特征建立良性和恶意应用模式集，但模式集易过时且检测准确率较低. |
| | 文献 [87] | 2018 | KNN，K-Medoids | 网络流量、权限 | 良性:（271）<br>恶意:（180） | 结合静态权限和动态网络流量特征，但检测准确率较低，且实验数据集较小. |
| | 文献 [86] | 2020 | DL-Droid | 权限、API、动态 | 良性：McAfee Labs（19 620）<br>恶意:（11 505） | 可在真实设备上运行，检测性能较高，但动态分析所覆盖的行为有限. |
| 基于相似代码片段 | 文献 [105] | 2015 | 代码克隆检测 | 代码片段 | 良性：AppChina（484）<br>恶意：Genome（1 260） | 使用 NiCad 查找恶意应用中的克隆类，选择并生成签名集以检测恶意应用，但只能检测已知恶意家族的应用且对克隆检测结果的依赖性较强. |
| | 文献 [98] | 2017 | DR-Droid | 代码片段 | 良性：Google Play（1 617）<br>恶意：VirusShare(（1979） | 可检测重打包恶意应用，但难以应对使用代码混淆、动态加载等技术的恶意应用. |
| | 文献 [97] | 2019 | ANDRE | 代码片段 | 良性：未使用<br>恶意：Drebin（5 416），VirusShare（9 000） | 移除通用第三方库，对恶意应用聚类，并使用 DNN 对恶意应用进一步分类，但需要较大规模的恶意应用数据集以进行准确聚类. |
| | 文献 [104] | 2022 | MsDroid | 代码片段 | 良性：Androzoo（51 580）<br>恶意：Drebin（5 560），AMD，Androzoo（24 650） | 基于代码片段使用 GNN 检测恶意应用，但用于检测新增样本时精度下降较为明显. |
| 其他特征检测方法 | 文献 [109] | 2016 | ICCDetector | Intent、组件 | 良性：Google Play（12 026）<br>恶意：Drebin（5 264） | 根据组件间通信模式检测恶意应用，但难以应对使用反射或代码混淆技术的恶意应用. |
| | 文献 [108] | 2017 | SigFeedback | 签名 | 良性：AppChina（1 407）<br>恶意：VirusShare（1 214） | 通过对 APK 签名信息的验证和筛选来检测恶意应用和筛选来检测恶意应用，但实验数据集较小. |

测问题研究中使用最多的恶意数据集. Zhou 等人[112]最早共享出 Genome 数据集, Genome 里面包含 49 个家族, 共 1 260 个恶意应用, 应用分布年份在 2010—2011 年间. Arp 等人[7]收集了 Drebin 数据集, 该数据集包含 179 个家族, 共 5 560 个恶意应用. Wei 等人[113]共享了 AMD (Android malware dataset)数据集, AMD 数据集包含 24 650 个恶意应用. Kadir 等人[114]针对 Android 僵尸网络的研究工作收集了数据集 Botnet, 其共包括 1 929 个僵尸网络恶意应用. Wang 等人[115]根据 McAfee, FireEye, Kaspersky 等 10 家安全公司发布的移动安全报告, 收集创建了包含 148 个恶意应用家族, 共 4 534 个恶意应用的数据集 MalRadar, 并从分发渠道、安装方法、激活方法、恶意行为和反分析技术等方面进行了描述. 关注到近年来 COVID-19 相关的恶意应用较为泛滥的问题, Wang 等人[116]系统地对 COVID-19 相关的应用进行分析, 构建了含有 4 322 个应用的 COVID-19 相关应用数据集, 其中 611 个为恶意应用. 这些恶意应用数据集有力推动了 Android 恶意应用检测技术的发展.

随后出现的恶意应用共享平台, 收集了用户提交的大量恶意应用, 可供研究人员下载使用. 例如 OmniDroid[117], Koodous, Contagio, Pwnzen, VirusShare 等. 表 5 给出了常用的恶意应用数据集和恶意应用共享平台, 其中, 应用分布年份代表该数据集所收集应用的年份. 从表 5 中可以看出 Koodous, MalRadar, Androzoo 提供了一些近年来出现的恶意应用.

**Table 5 Datasets of Malware Application**

**表 5 恶意应用数据集**

| 名称 | 应用分布年份 | 恶意应用数量 | 访问链接 |
|---|---|---|---|
| Genome[112] | 2010—2011 | 1 260 | http://www.malgenomeproject.org/ |
| Drebin[7] | 2010—2012 | 5 560 | https://www.sec.tu-bs.de/~danarp/drebin/index.html |
| AMD[113] | 2010—2016 | 24 650 | http://amd.arguslab.org/ |
| Botnet[114] | 2010—2014 | 1929 | https://www.unb.ca/cic/datasets/android-botnet.html |
| OmniDroid[117] | 2018 | 22 000 | http://aida.etsisi.upm.es/datasets/ |
| MalRadar[115] | 2014—2021 | 4 534 | https://malradar.github.io/ |
| Covid19apps[116] | 2020 | 611 | https://covid19apps.github.io/ |
| Koodous | 2015—2021 | 347 079 | https://koodous.com/ |
| Contagio | 2015—2018 | 360 | http://contagiominidump.blogspot.com/ |
| Pwnzen | 2020 | 1 830 | http://www.pwnzen.com/ |
| VirusShare | 2012—2019 | 182 923 | https://virusshare.com/ |
| Androzoo | 2016—2022 | 1 302 969 | https://androzoo.uni.lu/ |

### 6.2 评价指标

在 Android 恶意应用检测过程中, 研究人员最常使用的评估分类模型指标有准确率(*Accuracy*)、误报率(*ErrorRate*)、精确率(*Precision*)、召回率(*Recall*)、*F*1值、ROC 曲线和 AUC 值等. 在表 1～4 展示了各参考文献中采用的评价指标及最优性能. 除 ROC 曲线外, 其余评价指标都需要通过混淆矩阵计算, Android 恶意应用检测的混淆矩阵如表 6 所示.

**Table 6 Confusion Matrix**

**表 6 混淆矩阵**

| 预测分类 | 实际分类 | |
|---|---|---|
| | 恶意应用 | 良性应用 |
| 恶意应用 | *TP* | *FP* |
| 良性应用 | *FN* | *TN* |

*Accuracy*是系统对整个样本的检测能力, 即将恶意应用检测为恶意应用, 将良性应用检测为良性应用的能力, 计算为

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}. \tag{1}$$

*ErrorRate*与*Accuracy*相反, 表示预测错误的样本(*FP*和*FN*)在所有样本中占的比例, 计算为

$$ErrorRate = \frac{FP+FN}{TP+TN+FP+FN}. \tag{2}$$

*Precision*是实际为恶意应用的个数占检测为恶意应用的个数之比, 计算为

$$Precision = \frac{TP}{TP+FP}. \tag{3}$$

*Recall*是被正确识别出来的恶意应用个数与测试集中所有恶意应用个数的比值, 计算为

$$Recall = \frac{TP}{TP + FN}. \quad (4)$$

$F1$值（$F1$-Measure）是 *Precision* 和 *Recall* 的调和平均值，计算为

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}. \quad (5)$$

受试者工作特征曲线（receiver operating characteristic, ROC），ROC 曲线横坐标为假正率（false positive rate, FPR），纵坐标为真正率（true positive rate, TPR）. 根据 ROC 曲线面积（area under the curve, AUC）值判断分类器的优劣，AUC 取值范围是 0～1，越靠近 1，性能越好.

## 7 结束语

本文对 Android 恶意应用静态检测方法的最新研究技术成果进行了回顾和总结. 首先，对 Android 应用静态特征进行了分析；然后，将现有恶意 Android 检测静态方法分为基于权限的方法、基于 API 的方法、基于操作码的方法、基于其他静态特征的方法 4 类，分别对各类方法进行了归纳总结，并比较了不同方法的优缺点；最后，介绍了在 Android 恶意应用检测研究中常用的数据集和评价指标，以方便对后续提出的检测方法进行全面地评估. 虽然研究人员针对 Android 恶意应用静态检测方法取得了大量研究成果，但是还存在缺少统一的 Android 实验数据集，难以提取经加壳、混淆等处理后应用的静态特征，未能充分利用多类特征的互补性提升检测精度和检测新类型的恶意应用能力较弱等局限性. 基于上述分析，本文提出 Android 恶意应用静态检测 4 个值得进一步研究的方向：

1）构建持续更新的 Android 应用开放数据集. Android 应用数据集对实验结果有着至关重要的影响，尽管研究人员已共享了大量 Android 恶意应用数据集，其中一些恶意应用数据集[7,112-113]被使用的次数也较多，但是其中存在部分发布时间较久的恶意应用，可能影响方法评估的有效性，且缺少代表性强、认可度高的开放数据集. 另外，Android 恶意应用的更新换代较为频繁，需要提供开放接口以面向用户收集最新出现的恶意应用. 构建持续更新的 Android 应用开放数据集有助于将研究人员更公平和全面地评估所提出的恶意应用检测方法.

2）提升自动化提取特殊 Android 应用静态特征能力. Android 恶意应用静态检测方法中使用了权限、API 和操作码等多种特征，快速准确地提取特征是决定恶意应用检测性能的关键环节. 研究人员在实验中所使用的 Android 恶意应用数据集通常经过筛选和预处理，确保能够从中提取静态特征. 然而，在真实场景下，Android 恶意应用常会使用代码混淆、加密、加壳和反编译对抗等代码保护技术，以及代码热更新的方法，导致直接提取静态特征失败，而人工脱壳反汇编提取特征则效率较低. 为提升自动化提取特殊 Android 应用静态特征能力，需研制更为成熟有效的 Android 应用反编译脱壳等工具.

3）融合多类特征综合提升 Android 恶意应用检测性能. 权限、API 和操作码等不同类型的信息可从不同角度表征 Android 应用的行为，可通过融合多类特征以提升 Android 恶意应用检测的性能. 例如，可针对每种静态特征类型分别搭建恶意应用检测模型，再通过集成学习结合多类模型，以提高检测准确率. 除静态特征外，还可尝试加入资源申请、交互行为等动态特征以进一步提升 Android 恶意应用检测性能.

4）提供有效的自适应方法检测新增 Android 恶意应用. 随着移动互联网技术的发展，Android 系统版本不断升级，恶意应用出现的场景也在不断变化，例如，在人们焦虑 COVID-19 的时候，部分恶意应用伪造 COVID-19 疫苗预约和注册的功能来进行网络钓鱼和诈骗[116]. 由于静态特征和新业务场景变化，导致基于已有恶意应用构建的模型在检测新出现的恶意应用时面临着检测模型退化、检测准确率下降等问题[36]，这通常需要收集新数据集再次进行模型训练，并且增加了时间和经济成本. 未来可使用增量学习方法构建可持续更新的 Android 恶意应用检测模型，有效学习新增 Android 应用样本特征，避免重新训练模型带来的开销，并保持相对较高的准确率，以更好地适用于移动应用市场等需要处理大量新增待检测应用的场景.

**作者贡献声明：**潘建文负责文献资料的整理和分析、论文主体撰写及修订等工作；崔展齐指导论文选题、论文整体思路框架设计并修改论文；林高毅整理资料并修改论文；陈翔和郑丽伟提出指导意见并修改论文.

## 参 考 文 献

[1] China Internet Network Information Center. Statistical report on the development of Internet in China [EB/OL]. 2022[2022-07-10]. https://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/202202/P02022-

0407403488048001.pdf (in Chinese)
(中国互联网络信息中心. 中国互联网络发展状况统计报告[EB/OL]. 2022[2022-07-10].https://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/ 202202/P020220407403488048001.pdf)

[2] International Data Corporation. Smartphone market share [EB/OL]. 2021[2021-12-09].https://www.idc.com/promo/smartphone-market-share

[3] The National Computer Network Emergency Response Technical Team/Coordination Center of China. 2020 China Internet network security report[EB/OL]. 2021[2022-07-08].https://www.cert.org.cn/p ublish/main/upload/File/2020%20Annual%20Report.pdf
(中国国家计算机网络应急响应技术团队/协调中心. 2020年中国互联网网络安全报告[EB/OL]. 2021[2022-07-08].https://www.cert.or g.cn/publish/main/upload/File/2020%20Annual%20Report.pdf)

[4] Project Zero Team at Google. Bad binder: Android in the wild exploit [EB/OL]. 2019[2021-11-05].https://googleprojectzero.blogspot.com/20 19/11/bad-binder-android-in-wild-exploit.html

[5] Hasan H, Ladani B T, Zamani B. MEGDroid: A model-driven event generation framework for dynamic Android malware analysis[J]. Information and Software Technology, 2021, 135: 106569

[6] Feng Pingbin, Ma Jianfen, Sun Cong, et al. A novel dynamic Android malware detection system with ensemble learning[J]. IEEE Access, 2018, 6: 30996−31011

[7] Arp D, Spreitzenbarth M, Hubner M, et al. Drebin: Effective and explainable detection of Android malware in your pocket[C] //Proc of the 14th Symp of Network & Distributed System Security Symp. San Diego, CA: ISOC, 2014: 23−26

[8] Samra A A A, Qunoo H N, Al-Rubaie F, et al. A survey of static Android malware detection techniques[C/OL] //Proc of the 7th IEEE Palestinian Int Conf on Electrical and Computer Engineering (PICECE). Piscataway, NJ: IEEE, 2019[2022-09-03].https://ieeexplore.ieee.org/ document/8747224

[9] Bayazit E C, Sahingoz O K, Dogan B. Malware detection in Android systems with traditional machine learning models: A survey[C/OL] // Proc of the 2020 Int Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA). Piscataway, NJ: IEEE,2020[2022-09-03].https://ieeexplore.ieee.org/abstract/document/9152840

[10] Desnos A, Gueguen G, Bachmann S. Androguard [EB/OL]. 2018[2022-09-03].https://androguard.readthedocs.io/

[11] Tumbleson C. Apktool [EB/OL]. 2022[2022-09-03].https://ibotpeache s.github.io/Apktool/

[12] Android Developers. Android runtime (ART) and Dalvik [EB/OL]. (2022-08-11)[2022-09-03].https://source.android.com/docs/core/dalvik

[13] Peruma A, Palmerino J, Krutz D E. Investigating user perception and comprehension of Android permission models[C] //Proc of the 5th Int Conf on Mobile Software Engineering and Systems. New York: ACM, 2018: 56−66

[14] Sanz B, Santos I, Laorden C, et al. Puma: Permission usage to detect malware in Android[C] //Proc of the 5th the Int Joint Conf Computational Intelligence in Security for Information Systems. Berlin: Springer, 2012: 289–298

[15] Ilham S, Abderrahim G, Abdelhakim B A. Permission based malware detection in Android devices[C/OL] //Proc of the 3rd Int Conf on Smart City Applications. New York: ACM, 2018[2022-09-03]. https://dl.ac m.org/doi/abs/10.1145/3286606.3286860

[16] Android Developers. Intent [EB/OL]. (2022-06-08)[2022-09-03].https://developer.android.com/reference/android/content/Intent

[17] Yang Hongyu, Xu Jin. Android malware detection based on improved random forest[J]. Journal on Communications, 2017, 38(4): 8−16 (in Chinese)
(杨宏宇, 徐晋. 基于改进随机森林算法的Android恶意软件检测[J]. 通信学报, 2017, 38(4): 8−16)

[18] Feizollah A, Nor B A, Rosli S, et al. AndroDialysis: Analysis of Android intent effectiveness in malware detection[J]. Computers & Security, 2017, 65: 121−134

[19] Idrees F, Rajarajan M, Conti M, et al. PIndroid: A novel Android malware detection system using ensemble learning methods[J]. Computers & Security, 2017, 68: 36−46

[20] Zhang Li, Thing V L L, Cheng Yao. A scalable and extensible framework for Android malware detection and family attribution[J]. Computers & Security, 2019, 80: 120−133

[21] Wang Wei, Wang Xing, Feng Dawei, et al. Exploring permission-induced risk in Android applications for malicious application detection[J]. IEEE Transactions on Information Forensics and Security, 2014, 9(11): 1869−1882

[22] Yang Huan, Zhang Yuqing, Hu Yupu, et al. Android malware detection method based on permission sequential pattern mining algorithm[J] Journal on Communications, 2013, 34(S1): 106−115 (in Chinese)
(杨欢, 张玉清, 胡予濮, 等. 基于权限频繁模式挖掘算法的Android恶意应用检测方法[J]. 通信学报, 2013, 34(S1): 106−115)

[23] Arora A, Peddoju S K, Conti M. PermPair: Android malware detection using permission pairs[J]. IEEE Transactions on Information Forensics and Security, 2019, 15: 1968−1982

[24] Android Developers. Android API[EB/OL]. (2022-05-11)[2022-09-03].https://developer.android.com/reference/packages

[25] Scalas M, Maiorca D, Mercaldo F, et al. On the effectiveness of system API-related information for Android ransomware detection[J]. Computers & Security, 2019, 86: 168−182

[26] Wu Songyang, Wang Pan, Li Xun, et al. Effective detection of Android malware based on the usage of data flow APIs and machine learning[J]. Information and Software Technology, 2016, 75: 17−25

[27] Shen Feng, Del Vecchio J, Mohaisen A, et al. Android malware detection using complex-flows[J]. IEEE Transactions on Mobile Computing, 2018, 18(6): 1231−1245

[28] Junaid M, Liu Donggang, Kung D. Dexteroid: Detecting malicious behaviors in Android apps using reverse-engineered life cycle models[J]. Computers & Security, 2016, 59: 92−117

[29] Feng Yu, Anand S, Dillig I, et al. Apposcopy: Semantics-based detection of Android malware through static analysis[C] //Proc of

the 22nd ACM SIGSOFT Int Symp on Foundations of Software Engineering. New York: ACM, 2014: 576−587

[30] Zhang Jie, Tian Cong, Duan Zhenhua. Taint analysis tool of Android applications based on tainted value graph[J]. Journal of Software, 2021, 32(6): 1701−1716 (in Chinese)
(张捷, 田聪, 段振华. 基于污染变量关系图的Android应用污点分析工具[J]. 软件学报, 2021, 32(6): 1701−1716)

[31] Miao Xiaochuan, Wang Rui, Xu Lei, et al. Security analysis for Android applications using sensitive path identification[J]. Journal of Software, 2017, 28(9): 2248−2263 (in Chinese)
(缪小川, 汪睿, 许蕾, 等. 使用敏感路径识别方法分析安卓应用安全性[J]. 软件学报, 2017, 28(9): 2248−2263)

[32] Wang Lei, Zhou Qin, He Dongjie, et al. Multi-source taint analysis technique for privacy leak detection of Android APPs[J]. Journal of Software, 2019, 30(2): 211−230 (in Chinese)
(王蕾, 周卿, 何冬杰, 等. 面向Android应用隐私泄露检测的多源污点分析技术[J]. 软件学报, 2019, 30(2): 211−230)

[33] Elish K O, Shu Xiaokui, Yao Dangfen, et al. Profiling user-trigger dependence for Android malware detection[J]. Computers & Security, 2015, 49: 255−273

[34] Alam S, Alharbi S A, Yildirim S. Mining nested flow of dominant APIs for detecting Android malware[J]. Computer Networks, 2020, 167: 107026

[35] Mariconti E, Onwuzurike L, Andriotis P, et al. MaMaDroid: Metecting Android malware by building Markov chains of behavioral models[C/OL] //Proc of the 23rd Annual Network and Distributed System Security Symp(NDSS). San Diego, CA: ISOC, 2017[2022-07-03].https://www.ndss-symposium.org/ndss2017/ndss-2017-progr-amme/mamadroid-detecting-android-malware-building-markov-chains-behavioral-models/

[36] Zhang Xiaohan, Zhang Yuan, Zhong Ming, et al. Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware[C] //Proc of the 2020 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2020: 757−770

[37] Xu Jiayun, Li Yingjiu, Deng R, et al. SDAC: A slow-aging solution for Android malware detection using semantic distance based API clustering[J]. IEEE Transactions on Dependable and Secure Computing, 2020, 19(2): 1149−1163

[38] Wu Yueming, Li Xiaodi, Zou Deqing, et al. MalScan: Fast market-wide mobile malware scanning by social-network centrality analysis[C] //Proc of the 34th IEEE/ACM Int Conf on Automated Software Engineering (ASE). Piscataway, NJ: IEEE, 2019: 139−150

[39] Zou Deqing, Wu Yueming, Yang Siru, et al. IntDroid: Android malware detection based on API intimacy analysis[J]. ACM Transactions on Software Engineering and Methodology, 2021, 30(3): 1−32

[40] Wu Yueming, Zou Deqing, Yang Wei, et al. HomDroid: Detecting Android covert malware by social-network homophily analysis[C] // Proc of the 30th ACM SIGSOFT Int Symp on Software Testing and Analysis. New York: ACM, 2021: 216−229

[41] Zhao Kaifa, Zhou Hao, Zhu Yulin, et al. Structural attack against graph based Android malware detection[C] //Proc of the 9th ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2021: 3218−3235

[42] Demontis A, Melis M, Biggio B, et al. Yes, machine learning can be more secure! A case study on Android malware detection[J]. IEEE Transactions on Dependable and Secure Computing, 2017, 16(4): 711−724

[43] D'Angelo G, Ficco M, Palmieri F. Malware detection in mobile environments based on autoencoders and API-images[J]. Journal of Parallel and Distributed Computing, 2020, 137: 26−33

[44] Gao Tianchong, Peng Wei, Sisodia D, et al. Android malware detection via graphlet sampling[J]. IEEE Transactions on Mobile Computing, 2018, 18(12): 2754−2767

[45] Xu Bingbing, Cen Keting, Huang Junjie, et al. A survey on graph convolutional neural network[J]. Chinese Journal of Computers, 2020, 43(5): 755−780 (in Chinese)
(徐冰冰, 岑科廷, 黄俊杰, 等. 图卷积神经网络综述[J]. 计算机学报, 2020, 43(5): 755−780)

[46] Gao Han, Cheng Shaoyin, Zhang Weiming. GDroid: Android malware detection and classification with graph convolutional network[J]. Computers & Security, 2021, 106: 102264

[47] Li Shanxi, Zhou Qinguo, Zhou Rui, et al. Intelligent malware detection based on graph convolutional network[J]. The Journal of Supercomputing, 2021, 78: 4182−4198

[48] Pei Xinjun, Yu Long, Tian Shengwei. AMalNet: A deep learning framework based on graph convolutional networks for malware detection[J]. Computers & Security, 2020, 93: 101792

[49] Ou Fan, Xu Jian. $S^3$Feature: A static sensitive subgraph-based feature for Android malware detection[J]. Computers & Security, 2021, 112: 102513

[50] Fan Ming, Liu Jun, Luo Xiapu, et al. Android malware familial classification and representative sample selection via frequent subgraph analysis[J]. IEEE Transactions on Information Forensics and Security, 2018, 13(8): 1890−1905

[51] Lu Xiaofeng, Zhao Jinglun, Lio P. Robust Android malware detection based on subgraph network and denoising GCN network[C] //Proc of the 20th Annual Int Conf on Mobile Systems, Applications and Services. New York: ACM, 2022: 549−550

[52] Amer E, Zelinka I, El-Sappagh S. A multi-perspective malware detection approach through behavioral fusion of API call sequence[J]. Computers & Security, 2021, 110: 102449

[53] Allen J, Landen M, Chaba S, et al. Improving accuracy of Android malware detection with lightweight contextual awareness[C] //Proc of the 34th Annual Computer Security Applications Conf. New York: ACM, 2018: 210−221

[54] Zhang Mu, Duan Yue, Yin Heng, et al. Semantics-aware Android malware classification using weighted contextual API dependency graphs[C] //Proc of the 2014 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2014: 1105−1116

[55] Narayanan A, Chandramohan M, Chen Lihui, et al. A multi-view context-aware approach to Android malware detection and malicious

code localization[J]. Empirical Software Engineering, 2018, 23(3): 1222−1274

[56] Frenklach T, Cohen D, Shabtai A, et al. Android malware detection via an app similarity graph[J]. Computers & Security, 2021, 109: 102386

[57] Karbab E M B, Debbabi M, Derhab A, et al. Scalable and robust unsupervised Android malware fingerprinting using community-based network partitioning[J]. Computers & Security, 2020, 97: 101965

[58] Karbab E M B, Debbabi M, Derhab A, et al. Cypider: Building community-based cyber-defense infrastructure for Android malware detection[C] //Proc of the 32nd Annual Conf on Computer Security Applications. New York: ACM, 2016: 348−362

[59] Badhani S, Muttoo S K. CENDroid—A cluster-ensemble classifier for detecting malicious Android applications[J]. Computers & Security, 2019, 85: 25−40

[60] Mahindru A, Sangal A L. HybriDroid: An empirical analysis on effective malware detection model developed using ensemble methods[J]. The Journal of Supercomputing, 2021, 77: 8209−8251

[61] Zhu Huijuan, Wang Liangmin, Zhong Sheng, et al. A hybrid deep network framework for Android malware detection[J/OL]. IEEE Transactions on Knowledge and Data Engineering, 2021[2022-09-03].https://ieeexplore.ieee.org/abstract/document/9387579

[62] Cen Lei, Gates C S, Si Luo, et al. A probabilistic discriminative model for Android malware detection with decompiled source code[J]. IEEE Transactions on Dependable and Secure Computing, 2014, 12(4): 400−412

[63] Peynirci G, Eminağaoğlu M, Karabulut K. Feature selection for malware detection on the Android platform based on differences of IDF values[J]. Journal of Computer Science and Technology, 2020, 35(4): 946−962

[64] Kong Ke, Zhang Zhichao, Yang Ziyuan, et al. FCSCNN: Feature centralized Siamese CNN-based Android malware identification[J]. Computers & Security, 2021, 112: 102514

[65] Cai Lingru, Li Yao, Xiong Zhi. JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters[J]. Computers & Security, 2021, 100: 102086

[66] Feng Rui, Chen Sen, Xie Xiaofei, et al. A performance-sensitive malware detection system using deep learning on mobile devices[J]. IEEE Transactions on Information Forensics and Security, 2020, 16: 1563−1578

[67] Garcia J, Hammad M, Malek S. Lightweight, obfuscation-resilient detection and family identification of Android malware[J]. ACM Transactions on Software Engineering and Methodology, 2018, 26(3): 1−29

[68] Hei Yiming, Yang Renyu, Peng Hao, et al. HAWK: Rapid Android malware detection through heterogeneous graph attention networks[J/OL]. IEEE Transactions on Neural Networks and Learning Systems, 2021[2022-09-03].https://ieeexplore.ieee.org/abstract/docu ment/9524453

[69] Tang Junwei, Li Ruixuan, Jiang Yu, et al. Android malware obfuscation variants detection method based on multi-granularity opcode features[J]. Future Generation Computer Systems, 2022, 129: 141−151

[70] Chen Tieming, Yang Yimin, Chen Bo. Maldetect: An Android malware detection system based on abstraction of dalvik instructions[J]. Journal of Computer Research and Development, 2016, 53(10): 2299−2306 (in Chinese)
(陈铁明, 杨益敏, 陈波. Maldetect: 基于Dalvik指令抽象的Android恶意代码检测系统[J]. 计算机研究与发展, 2016, 53(10): 2299−2306)

[71] McLaughlin N, Martinez del Rincon J, Kang B J, et al. Deep Android malware detection[C] //Proc of the 7th ACM on Conf on Data and Application Security and Privacy. New York: ACM, 2017: 301−308

[72] Li Tine, Dong Hang, Yuan Chunyang, et al. Description of Android malware feature based on Dalvik instructions[J]. Journal of Computer Research and Development, 2014, 51(7): 1458−1466 (in Chinese)
(李挺, 董航, 袁春阳, 等. 基于Dalvik指令的Android恶意代码特征描述及验证[J]. 计算机研究与发展, 2014, 51(7): 1458−1466)

[73] Mercaldo F, Santone A. Formal equivalence checking for mobile malware detection and family classification[J]. IEEE Transactions on Software Engineering, 2021, 48(7): 2643−2657

[74] Canfora G, Mercaldo F, Visaggio C A. An HMM and structural entropy based detector for Android malware: An empirical study[J]. Computers & Security, 2016, 61: 1−18

[75] Xiao Xusheng, Yang Shao. An image-inspired and CNN-based Android malware detection approach[C] //Proc of the 34th IEEE/ACM Int Conf on Automated Software Engineering (ASE). Piscataway, NJ: IEEE, 2019: 1259−1261

[76] Yadav P, Menon N, Ravi V, et al. EfficientNet convolutional neural networks-based Android malware detection[J]. Computers & Security, 2022, 115: 102622

[77] Iadarola G, Martinelli F, Mercaldo F, et al. Towards an interpretable deep learning model for mobile malware detection and family identification[J]. Computers & Security, 2021, 105: 102198

[78] Yuan Baoguo, Wang Junfeng, Liu Dong, et al. Byte-level malware classification based on Markov images and deep learning[J]. Computers & Security, 2020, 92: 101740

[79] Zhang Bing, Wen Zheng, Wei Xiaoyu, et al. InterDroid: An interpretable Android malware detection method for conceptual drift[J]. Journal of Computer Research and Development, 2021, 58(11): 2456−2474 (in Chinese)
(张炳, 文峥, 魏筱瑜, 等. InterDroid: 面向概念漂移的可解释性Android恶意软件检测方法[J]. 计算机研究与发展, 2021, 58(11): 2456−2474)

[80] Qiu Junyang, Han Qinglong, Luo Wei, et al. Cyber code intelligence for Android malware detection[J/OL]. IEEE Transactions on Cybernetics, 2022[2022-09-03].https://ieeexplore.ieee.org/abstract/document/9764650

[81] Chen Bo, Pan Yongtao, Chen Tieming. Android malware detection method based on SimHash[J]. Journal on Communications, 2017, 38(S2): 30−36 (in Chinese)
(陈波, 潘永涛, 陈铁明. 基于多层SimHash的Android恶意应用程

序检测方法[J]. 通信学报, 2017, 38(S2): 30−36)

[82] Kim T G, Kang B J, Rho M, et al. A multimodal deep learning method for Android malware detection using various features[J]. IEEE Transactions on Information Forensics and Security, 2018, 14(3): 773−788

[83] Lu Ning, Li Dan, Shi Wenbo, et al. An efficient combined deep neural network based malware detection framework in 5G environment[J]. Computer Networks, 2021, 189: 107932

[84] Yang Huan, Zhang Yuqing, Hu Yupu, et al. A malware behavior detection system of Android applications based on multi-class features[J]. Chinese Journal of Computers, 2014, 37(1): 15−27 (in Chinese)
(杨欢, 张玉清, 胡予濮, 等. 基于多类特征的Android应用恶意行为检测系统[J]. 计算机学报, 2014, 37(1): 15−27)

[85] Android Developers. UI/application exerciser Monkey[EB/OL]. (2022-03-11)[2022-09-03].https://developer.android.com/studio/test/monkey.html

[86] Alzaylaee M K, Yerima S Y, Sezer S. DL-Droid: Deep learning based Android malware detection using real devices[J]. Computers & Security, 2020, 89: 101663

[87] Arora A, Peddoju S K, Chouhan V, et al. Hybrid Android malware detection by combining supervised and unsupervised learning[C] // Proc of the 24th Annual Int Conf on Mobile Computing and Networking. New York: ACM, 2018: 798−800

[88] Tong Fei, Yan Zheng. A hybrid approach of mobile malware detection in Android[J]. Journal of Parallel and Distributed Computing, 2017, 103: 22−31

[89] Wang Shanshan, Chen Zhenxiang, Yan Qiben, et al. Deep and broad URL feature mining for Android malware detection[J]. Information Sciences, 2020, 513: 600−613

[90] Han Qian, Subrahmanian V S, Xiong Yanhai. Android malware detection via (somewhat) robust irreversible feature transformations[J]. IEEE Transactions on Information Forensics and Security, 2020, 15: 3511−3525

[91] Chakraborty T, Pierazzi F, Subrahmanian V S. Ec2: Ensemble clustering and classification for predicting Android malware families[J]. IEEE Transactions on Dependable and Secure Computing, 2017, 17(2): 262−277

[92] Jerbi M, Dagdia Z C, Bechikh S, et al. On the use of artificial malicious patterns for Android malware detection[J]. Computers & Security, 2020, 92: 101743

[93] Li Pengwei, Jiang Yuqian, Xue Feiyang, et al. A robust approach for Android malware detection based on deep learning[J]. Acta Electronica Sinica, 2020, 48(8): 1502−1508 (in Chinese)
(李鹏伟, 姜宇谦, 薛飞扬, 等. 一种基于深度学习的强对抗性Android恶意代码检测方法[J]. 电子学报, 2020, 48(8): 1502−1508)

[94] Costa F H, Medeiros I, Menezes T, et al. Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for Android malware identification[J]. Journal of Systems and Software, 2022, 183: 111092

[95] Yuan Zhenlong, Lu Yongqiang, Wang Zhaoguo, et al. Droid-Sec: Deep learning in Android malware detection[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 371−372

[96] Amer E, El-Sappagh S. Robust deep learning early alarm prediction model based on the behavioural smell for Android malware[J]. Computers & Security, 2022, 116: 102670

[97] Zhang Yanxin, Sui Yulei, Pan Shirui, et al. Familial clustering for weakly-labeled Android malware using hybrid representation learning[J]. IEEE Transactions on Information Forensics and Security, 2019, 15: 3401−3414

[98] Tian Ke, Yao Danfeng, Ryder B G, et al. Detection of repackaged Android malware with code-heterogeneity features[J]. IEEE Transactions on Dependable and Secure Computing, 2017, 17(1): 64−77

[99] Alam S, Qu Zhengyang, Riley R, et al. DroidNative: Automating and optimizing detection of Android native code malware variants[J]. Computers & Security, 2017, 65: 230−246

[100] Zhan Xian, Fan Lingling, Liu Tianming, et al. Automated third-party library detection for Android applications: Are we there yet?[C] // Proc of the 35th IEEE/ACM Int Conf on Automated Software Engineering (ASE). Piscataway, NJ: IEEE, 2020: 919−930

[101] Backes M , Bugiel S , Derr E . Reliable third-party library detection in Android and its security application[C] //Proc of the 23rd ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2016: 356−367

[102] Ma Ziang , Wang Haoyu , Yao Guo , et al. LibRadar: Fast and accurate detection of third-party libraries in Android apps[C] //Proc of the 38th Int Conf on Software Engineering Companion (ICSE-C). New York: ACM, 2017: 653−656

[103] Yuan Zhang , Dai Jiarun , Zhang Xiaohan , et al. Detecting third-party libraries in Android applications with high precision and recall[C] //Proc of the 25th IEEE Int Conf on Software Analysis, Evolution and Reengineering (SANER). Piscataway, NJ: IEEE, 2018: 141−152

[104] He Yiling, Liu Yiping, Wu Lei, et al. MsDroid: Identifying malicious snippets for Android malware detection[J/OL]. IEEE Transactions on Dependable and Secure Computing, 2022[2022-09-03].https://ieeexplore.ieee.org/abstract/document/9762803

[105] Chen Jian, Alalfi M H, Dean T R, et al. Detecting Android malware using clone detection[J]. Journal of Computer Science and Technology, 2015, 30(5): 942−956

[106] Cordy J R, Roy C K. The NiCad clone detector[C] //Proc of the 19th Int Conf on Program Comprehension. Piscataway, NJ: IEEE, 2011: 219−220

[107] Meng Guozhu, Xue Yinxing, Xu Zhengzi, et al. Semantic modelling of Android malware for effective malware comprehension, detection, and classification[C] //Proc of the 25th Int Symp on Software Testing and Analysis. New York: ACM, 2016: 306−317

[108] Liu Xinyu, Weng Jian, Zhang Yue, et al. Android malware detection based on APK signature information feedback[J]. Journal on Communications, 2017, 38(5): 190−198 (in Chinese)
(刘新宇, 翁健, 张悦, 等. 基于APK签名信息反馈的Android恶意应用检测[J]. 通信学报, 2017, 38(5): 190−198)

[109] Xu Ke, Li Yingjiu, Deng R H. ICCdetector: ICC-based malware detection on Android[J]. IEEE Transactions on Information

Forensics and Security, 2016, 11(6): 1252−1264

[110] Pinhero A, Anupama M L, Vinod P, et al. Malware detection employed by visualization and deep neural network[J]. Computers & Security, 2021, 105: 102247

[111] Allix K, Bissyandé T F, Klein J, et al. Androzoo: Collecting millions of Android apps for the research community[C] //Proc of the IEEE/ACM 13th Working Conf on Mining Software Repositories (MSR). Piscataway, NJ: IEEE, 2016: 468−471

[112] Zhou Yajin, Jiang Xuxian. Dissecting Android malware: Characterization and evolution[C] //Proc of the 2012 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2012: 95−109

[113] Wei Fengguo, Li Yuping, Roy S, et al. Deep ground truth analysis of current Android malware[C] //Proc of the 14th Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2017: 252−276

[114] Kadir A F A, Stakhanova N, Ghorbani A A. Android botnets: What URLs are telling us[C] //Proc of the 9th Int Conf on Network and System Security. Berlin: Springer, 2015: 78−91

[115] Wang Liu, Wang Haoyu, He Ren, et al. MalRadar: Demystifying Android malware in the new era[J]. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2022, 6(2): 1−27

[116] Wang Liu, He Ren, Wang Haoyu, et al. Beyond the virus: A first look at coronavirus-themed Android malware[J]. Empirical Software Engineering, 2021, 26(4): 1−38

[117] Martín A, Lara-Cabrera R, Camacho D. Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset[J]. Information Fusion, 2019, 52: 128−142

**Pan Jianwen**, born in 1994. Master candidate. Student member of CCF. His main research interest includes intelligent software analysis and testing technology.

**潘建文**, 1994 年生. 硕士研究生. CCF 学生会员. 主要研究方向为智能软件分析及测试技术.

**Cui Zhanqi**, born in 1984. PhD, professor, master supervisor. Senior member of CCF. His main research interestincludes intelligent software analysis and testing technology.

**崔展齐**, 1984 年生. 博士, 教授, 硕士生导师, CCF 高级会员. 主要研究方向为智能软件分析及测试技术.

**Lin Gaoyi**, born in 1998. Master candidate. Student member of CCF. His main research interest includes intelligent software analysis and testing technology.

**林高毅**, 1998 年生. 硕士研究生. CCF 学生会员. 主要研究方向为智能软件分析及测试技术.

**Chen Xiang**, born in 1980. PhD, associate professor, master supervisor. Senior member of CCF. His main research interests include software testing, software maintenance, empirical software engineering, and software repository mining.

**陈 翔**, 1980 年生. 博士, 副教授, 硕士生导师. CCF 高级会员. 主要研究方向为软件测试、软件维护、实证软件工程和软件仓库挖掘.

**Zheng Liwei**, born in 1979. PhD, associate professor, master supervisor. Member of CCF. His main research interests include requirement engineering and trusted computing.

**郑丽伟**, 1979 年生. 博士, 副教授, 硕士生导师. CCF 会员, 硕士生导师. 主要研究方向为需求工程和可信计算.