基于敏感组件函数调用图的安卓重打包恶意软件检测方法

杜瑞颖[1],陈 晶[2,3,4],吴 聪[5*],闫晰渝[2,3]

(1. 河南师范大学计算机与信息工程学院,河南新乡 453007;2. 武汉大学国家网络安全学院,湖北武汉 430072;
3. 武汉大学空天信息安全与可信计算教育部重点实验室,湖北武汉 430072;4. 武汉大学日照信息技术研究院,山东日照 276800;
5. 香港大学工学院,电机电子工程学系,中国香港 999077)

摘 要: 安卓系统占据移动操作系统七成以上的市场份额,成为许多不法分子传播恶意软件的平台. 其中,重打包恶意软件通过嵌入少量恶意代码到良性软件中,利用大量良性行为掩盖恶意行为,从而绕过普通恶意软件检测方法. 然而,当前学术界对重打包恶意软件的研究相对较少,现有基于函数调用图分区的检测方法存在通用性不足的问题,且在敏感API(Application Programming Interface)中心性特征方面未充分考虑恶意行为的语义特征. 本文提出了一种安卓重打包恶意软件检测方法Partdroid,该方法通过分析清单文件和smali代码,提取应用程序的组件信息并生成组件函数调用图,将所有含有敏感API的组件的函数调用图合并,利用污点分析的方法发掘组件间调用关系,形成敏感组件函数调用图来避免函数调用图分区的局限性. 同时,该方法通过挖掘敏感API与入口函数、交互函数的关系突出恶意行为的特征,并结合中心性算法综合计算敏感API的重要性,避免直接使用中心性算法提取特征的局限性. 实验结果表明,本方法对安卓重打包恶意软件检测的综合性能优于其他同类工具,随机森林分类器的F1值和准确率分别达到91.34%和91.93%,投票算法则为91.63%和92.15%. 此外,Partdroid在新恶意软件检测中表现突出,从谷歌应用商店随机选取的2 000个应用中检出3个可疑软件.

关键词: 安卓重打包恶意软件;函数调用图;敏感API;恶意行为;机器学习

基金项目: 国家重点研发计划(No.2021YFB2700200,No.2022YFB3103300);国家自然科学基金(No.62206203,No.62076187);湖北省重点研发计划(No.2022BAA039,No.2021BAA190);山东省重点研发计划(No.2022CXPT055)

中图分类号: TP309.5 **文献标识码:** A **文章编号:** 0372-2112(2025)07-2372-17

电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250075

A Detection Method for Android Repackaged Malware Based on Sensitive Component Function Call Graph

DU Rui-ying[1], CHEN Jing[2,3,4], WU Cong[5*], YAN Xi-yu[2,3]

(1. *School of Computer and Information Engineering*, *Henan Normal University*, *Xinxiang*, *Henan* 453007, *China*;
2. *School of Cyber Science and Engineering*, *Wuhan University*, *Wuhan*, *Hubei* 430072, *China*;
3. *Key Laboratory of Aerospace Information Security and Trusted Computing*, *Ministry of Education*, *Wuhan University*, *Wuhan*, *Hubei* 430072, *China*;
4. *Rizhao Institute of Information Technology*, *Wuhan University*, *Rizhao*, *Shandong* 276800, *China*;
5. *Department of Electrical and Electronic Engineering*, *Faculty of Engineering*, *The University of Hong Kong*, *Hong Kong* 999077, *China*)

Abstract: The Android system occupies over 70% of the market share of mobile operating systems, making it a key platform for malicious actors to distribute malware. Repackaged malware embeds a small amount of malicious code into legitimate software, masking malicious activities with a majority of benign behaviors to evade traditional malware detection methods. However, academic research on repackaged malware remains relatively limited. Existing detection methods based on partitioning function call graphs often lack generalizability and fail to fully capture the semantic features of malicious behavior associated with sensitive API(Application Programming Interface) centrality. To solve these problems, we propose

收稿日期:2025-01-22;录用日期:2025-05-09;责任编辑:覃怀银
*通讯作者:吴聪

Partdroid, a detection method for Android repackaged malware. The method analyzes manifest files and smali code to extract application component information and generate component function call graphs. It combines graphs of components with sensitive APIs and uses taint analysis to uncover inter-component relationships, forming a sensitive component function call graph to overcome partitioning limitations. Additionally, Partdroid highlights malicious behavior by exploring the relationships between sensitive APIs, entry functions, and interaction functions. It also integrates centrality algorithms to calculate the importance of sensitive APIs comprehensively, addressing the limitations of directly using centrality algorithms for feature extraction. Experimental results demonstrate that Partdroid outperforms other tools in detecting Android repackaged malware, achieving an F1 score of 91.34% and accuracy of 91.93% with a random forest classifier, and 91.63% and 92.15% with a voting algorithm. Moreover, Partdroid performs outstandingly in detecting new malware, identifying 3 suspicious software among 2 000 randomly selected applications from the Google Play Store.

Key words: android repackaged malware; function call graph; sensitive API; malicious behavior; machine learning

Foundation Item(s): National Key Research and Development Program of China (No.2021YFB2700200, No.2022YFB3103300); National Natural Science Foundation of China (No.62206203, No.62076187); The Key Research and Development Program of Hubei Province (No.2022BAA039, No.2021BAA190); The Key Research and Development Program of Shandong Province (No.2022CXPT055)

1　引言

自2007年底谷歌公司携手84家企业创立开放手持设备联盟并推出安卓系统以来,该系统凭借其开源特性迅速吸引了众多智能手机厂商,逐步占据了市场的主导地位. 截至2010年,安卓系统已超越塞班系统,成为全球第一大移动智能终端操作系统;至2017年3月,其设备数量更是超越了微软的Windows系统,进一步巩固了其作为全球领先操作系统的地位. 据国际权威调研机构International Data Corporation报告[1]显示,2023年安卓系统的市场占有率约为79.9%,远超苹果操作系统,成为市场上不可或缺的一部分. 随着移动智能终端设备如智能手机、平板电脑等的普及,这些设备已成为人们日常生活中不可或缺的工具,极大地丰富了人们的生活方式并提升了社会生产效率.

尽管智能手机功能多样,但仍难以满足所有用户群体的特定需求,应用程序(Application,App,也称软件)作为智能手机的扩展,为用户提供更加丰富和个性化的功能. 安卓系统不仅拥有谷歌官方应用市场,还有众多第三方应用市场,如小米应用商店、腾讯应用宝等. 然而,应用程序在方便用户的同时,也引入了安全风险,各种恶意软件随之产生,安卓系统也因其高额的市场占有率成为了移动恶意软件的主要攻击目标. 据卡巴斯基报告[2]显示,2023年第三季度拦截移动恶意软件攻击共834.6万次,相比第二季度上升了46.3%. 一般恶意软件因其构造相对简陋,易于被用户察觉,然而在实际应用中,恶意软件制作者利用良性软件作为载体,将其反编译后写入恶意代码,然后重打包为恶意软件,并借助多种途径广泛传播. 这种重打包恶意软件制作成本低,且通过大量的良性行为掩盖其恶意本质,给检测工具带来了极大的辨识难度,使之难以区分重打包恶意软件与其良性载体应用程序,从而对用户安全构成了重大威胁.

尽管国内外研究者对安卓恶意软件检测进行了大量研究并取得了一定成果[3~16],但现有方法难以有效检测安卓重打包恶意软件,适用性较差. 针对安卓重打包恶意软件的检测研究仍相对匮乏,在函数调用图(call graph)提取方面,现有的重打包恶意软件检测方法大多通过对函数调用图分区的方式来识别和检测恶意软件,但随着软件功能逐渐丰富,难以找到足够数量的单区软件来训练分类器,导致该方法存在通用性不足的问题. 在特征提取方面,虽已有学者通过中心性算法提取函数调用图中敏感应用程序编程接口(Application Programming Interface,API)的中心性特征[8],但未考虑到恶意行为的语义特征,而重打包恶意软件对原始程序嵌入恶意代码后,功能或行为可能与其他良性软件类似,仅靠中心性特征难以准确区分,从而影响了检测准确率.

针对上述问题,本文提出一种安卓重打包恶意软件检测方法Partdroid,该方法一共由3个模块组成. 在基于敏感组件的函数调用图提取模块,Partdroid通过分析清单文件和smali代码提取安卓应用程序的活动(activity)、服务(service)、广播接收器(broadcast receiver)和内容提供者(content provider)[17]等四大组件信息,为每个组件生成组件函数调用图(Component Function Call Graph,CFCG),并通过搜集常用回调函数补充因安卓系统特性导致缺失的调用边,然后将调用敏感API的组件函数调用图(component function call graph for invoking sensitive APIs)合并为敏感组件函数调用图(Sensitive Component Function Call Graph,SCFCG),并利用污点分析方法挖掘组件间调用关系来补充完善敏感组件函数调用图. 其中,调用敏感API的组件函数调用图表示调用敏感API的组件对应的函数调用图,只包含某个敏感组件自身的调用关系,而敏感组件函数调用图

则表示整合了所有调用敏感API的组件函数调用图及其组件间调用边的整体函数调用图. 在基于恶意行为的特征提取模块,Partdroid通过挖掘敏感API与入口函数、交互函数的关系,结合中心性算法综合计算敏感API重要性来缓解仅使用中心性算法丢失语义信息的问题,并提取了不易被混淆的权限特征来避免特征过于单一影响准确性. 在基于机器学习的分类模块,将提取的特征用于训练多种机器学习分类器. 此外,本文通过实验评估证明了Partdroid在安卓重打包恶意软件和普通恶意软件检测上取得了良好的效果,同时能够在谷歌商店上检测出可疑软件.

综上,本文的主要贡献如下:

(1)设计了一种提取敏感组件函数调用图的方法,通过收集应用程序的组件信息并生成组件函数调用图,将所有含有敏感API的组件函数调用图合并为敏感组件函数调用图,并通过收集回调函数调用和分析组件调用关系补充调用图的连接边. 该方法提取的函数调用图恶意行为更明显,且具有唯一性,解决了现有方法存在难以找到足量的单区域样本训练机器学习分类器的问题.

(2)设计了一种将恶意行为特征与中心性算法结合的方法,通过发掘敏感API与入口函数、交互函数的关系凸显恶意行为的特征,并将该关系结合网络分析中常用的中心性算法来综合计算敏感API的重要性,缓解了现有工作中直接利用中心性算法提取敏感API特征时丢失恶意行为语义的问题.

(3)基于提出的敏感组件函数调用图提取方法和敏感API重要性计算方法,实现了一种安卓重打包恶意软件检测系统Partdroid,并选取安卓重打包恶意样本集合和普通恶意样本集合对其进行评估. 实验结果表明,Partdroid在检测重打包恶意样本时F1和准确率分别为91.34%和91.93%,综合性能优于同类工具;同时在检测普通恶意软件上也表现良好,虽综合性能略逊于Intdroid,但精确率高达99.14%. 此外,在新恶意软件检测中,Partdroid能有效发现潜在威胁,从谷歌应用商店随机选取的2 000个应用中成功检测出3个可疑软件.

2 相关工作

2.1 普通恶意软件检测

目前,普通安卓恶意软件检测方法主要分为2类:基于语法的检测方法[4~6]和基于语义的检测方法[7~16]. 基于语法的检测方法往往从安卓软件中直接提取特征,因为忽略了代码的语义,所以检测较为高效. 北京交通大学团队提出了一种基于语法的安卓恶意软件检测方法[4],通过提取权限信息并对权限进行风险排序,将排序靠前的权限作为特征结合机器学习算法,实现对恶意软件的高效检测. Kim等人[5]进一步优化特征提取方式,从软件中提取多维特征向量,如方法操作码特征、方法特征、权限特征和组件特征等,提升检测精度. 北京信息科技大学团队[6]提取应用权限、API和操作码3类特征,并计算选择Droid-TF-IDF值较高的特征子集训练分类器. 上述研究工作虽然使用了多种语法特征,在某些情况下展现出良好的性能,但收集的一些特征容易通过混淆手段进行更改从而影响检测有效性.

基于语义的检测方法[7~16]在检测恶意软件的准确率上往往要优于基于语法的检测方法,Mamadroid[7]通过分析数据流和上下文信息等语义特征提取函数调用图,结合马尔可夫链表示函数转换概率,并作为特征向量训练分类器. 为提升运行效率,Malscan利用Androguard[18]从安卓软件中快速构建函数调用图,使用基于社交网络的不同中心性算法计算敏感API的中心性,并将其作为特征训练分类模型. 该方法运行效率比Mamadroid高出数十倍,但良性软件和恶意软件存在类似行为时提取的特征相近,容易产生误报. Intdroid[10]在此基础上对API中心性进行排序并选取靠前的API作为中心节点,计算敏感API与中心节点的亲密度作为特征训练分类器. Whgdroid[11]从应用程序中提取5个实体,包括意图、特定权限、特定危险权限、API、API簇,通过丰富语义元路径来建立应用程序节点之间的隐式关联,并生成仅包含应用程序节点的同构图,最后使用神经网络学习这些节点的嵌入. 然而上述方法易受图对抗攻击,为了解决这个问题,Rgdroid[12]通过图神经网络学习函数调用图中的行为特征,并结合安卓文档中提取的API信息. 为抵御图对抗性攻击,它减少了不同功能部分的连接,以最大限度减少对抗性扰动. 虽然基于语义的检测方法准确率较高,但效率往往比基于语法的检测方法低.

2.2 重打包恶意软件检测

尽管国内外研究者已对安卓恶意软件检测进行了大量研究,并取得了一定成果,但对安卓重打包恶意软件并不适用. 针对安卓重打包恶意软件的检测研究也分为基于语法的检测方法[19~21]和基于语义的检测方法[22,23].

Singh等人[19]提出了一种基于语法的重打包恶意软件检测技术,从重打包恶意软件和良性软件中提取权限特征、包、意图、硬件信息、类、泄露信息情况、源代码嵌入等7种特征,并用这些特征训练分类器. Liu等人[20]提出了类似的重打包恶意软件检测技术,提取传感器、权限、包、硬件、意图、类和泄露信息情况等7种特征,并用这些特征训练分类器. 基于语法的特征容易受到混淆干扰,为此He等人[21]使用明文内容和流量统计特征来计算应用程序之间的相似度,并对相似度值进行聚类,以区分原始软件和重打包的恶意软件,然而恶意软件的恶意行为可能只在特定情况下触发,动态收

集流量特征可能遗漏恶意行为特征.

基于语义的检测[22,23]可以缓解混淆带来的影响,Migdroid[22]在smali代码基础上构建函数调用图,将其划分为多个弱连接子图,并根据调用的敏感API计算子图的威胁分数以检测恶意软件,但由于子图划分缺乏语义依据,可能存在事件和依赖关系不完整,且难以确定通用阈值,存在误报和漏报问题. Tian等人[23]提出了新的基于类依赖图的划分区域方法,首先对应用程序生成类依赖图,图中完全没有连接的子图即为划分的区域,并对每个子图生成函数调用图提取特征,利用单区域数据集训练分类器以缓解事件和依赖关系不完整问题. 然而,随着安卓软件功能复杂化,单区域样本数量不足,限制了检测方法的通用性. 随着安卓重打包恶意软件形式的日益多样化,还有一些针对特定类型重打包恶意软件的检测方法,如VAHunt[24]针对应用虚拟化重打包恶意软件,通过有限状态机模型识别应用虚拟化引擎的存在,进而通过数据流分析提取隐蔽加载策略特征,以区分恶意和良性应用虚拟化应用. 但有限状态机模型对复杂虚拟化引擎的识别准确率有待提高,数据流分析提取的特征可能不够全面,导致部分恶意行为无法被有效检测. 此外,Salem等人[25]开发了一种能够自动重打包安卓应用并注入任意载荷的工具,来按需生成重打包应用,帮助评估反重打包技术的效果.

综上,尽管国内外研究者已对安卓恶意软件检测进行了大量研究,并取得了一定成果,但针对安卓重打包恶意软件的检测研究仍相对匮乏,且存在划分依据不足、通用性不强等问题. 因此,开发一种通用且高效的安卓重打包恶意软件检测方法,对于保障用户安全、应对当前威胁具有重要意义.

3　设计与实现

3.1　研究动机

3.1.1　函数调用图缺陷

重打包恶意软件指将合法的应用程序反编译后,将恶意代码嵌入原本的代码中,再通过重新编译和打包,制作出的看似正常但含有恶意行为的软件. 其中合法的应用程序为实现多种功能,调用函数较多,而嵌入的恶意代码往往功能较少,调用的函数也相对较少. 重打包恶意软件的函数调用图如图1所示,其中黑色节点为该应用程序的合法代码,红色节点为嵌入的恶意代码,灰色边为函数调用边,该应用程序共含有16 010个节点,其中777个节点属于恶意代码,比例仅占4.8%. 这导致恶意代码的特征被大量良性代码掩盖,普通恶意软件检测方法难以有效提取恶意代码的特征,增加了假阴性的风险.

现有针对安卓重打包恶意软件的检测方案采用函数调用图分区策略,将整个软件函数调用图分成多个子区域. 由于样本标签仅区分恶意与良性,单一区域的恶意软件可轻易标记,但多区域应用却难以对每个区域单独标签化. 因此分区方法需要找到足够数量的仅含单一区域的应用程序才能有效训练机器学习分类器. 然而,随着应用功能复杂化,难以找到足够多的单区域样本,导致分区策略通用性受限.

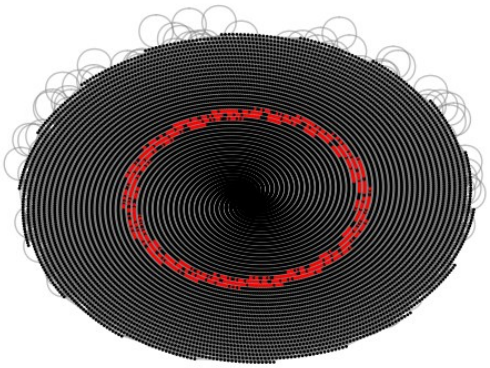

图1　重打包恶意软件中嵌入恶意代码的节点占比

因此如何在不使用分区的方法且尽可能地保留应用程序语义的情况下,凸显安卓重打包恶意软件的恶意行为特征成为了一个具有挑战性的问题.

3.1.2　敏感API中心性特征缺陷

Wu等人[8]的研究证明了用于分析网络结构的中心性算法可以用于安卓恶意软件检测,但中心性算法考虑的是图中节点之间的关系,比如节点到其他节点的距离、节点的直接连接数等,所以只使用中心性算法计算敏感API的中心性作为特征,将导致函数调用图丢失恶意行为的语义. 图2为良性软件和重打包恶意软件调用相同敏感API的示意图,其中蓝色代表正常系统API,红色代表敏感API,可以清晰地看到它们的节点结构非常相似,如果忽略了恶意行为语义,仅靠敏感API的中心性无法很好地区分良性行为和恶意行为.

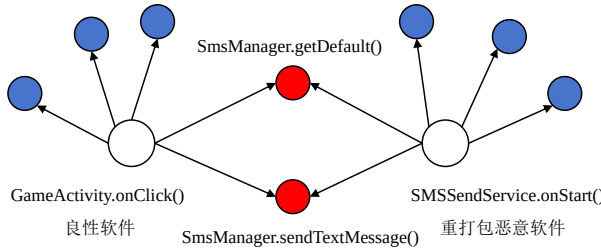

图2　良性软件与重打包恶意软件调用敏感API示意图

因此如何发掘恶意行为特征,并利用这些特征与中心性算法结合从而提取更准确的特征成为了一个具有创新性和挑战性的问题.

3.2 方案设计

为解决上述研究问题，本文设计并实现了安卓重打包恶意软件检测系统 Partdroid. 不同于现有方法多采用分区策略来解决重打包恶意软件中恶意代码被大量良性代码掩盖的函数调用图缺陷问题，本文提出了一种基于敏感组件的函数调用图提取方法，通过去除未调用敏感 API 的良性组件对应的函数调用图，并整合所有调用敏感 API 的敏感组件对应的函数调用图来构建该软件对应的敏感组件函数调用图，提高恶意组件的可见性，捕获更丰富的恶意代码特征，在不使用分区策略的情况下解决重打包恶意软件的函数调用图缺陷问题，且提取得到的敏感组件函数调用图涵盖了软件的敏感组件中包含的所有敏感 API 及其调用关系，极大地保留了应用程序的语义信息. 如图 3 所示，Partdroid 主要分为以下 3 个模块.

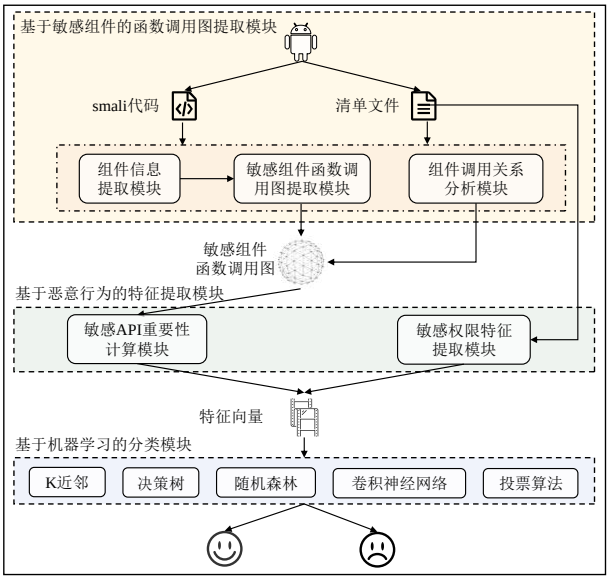

图 3 Partdroid 系统流程

(1)基于敏感组件的函数调用图提取模块. 提取待检测应用程序的敏感组件函数调用图，用于后续的特征提取. 具体而言，首先分析清单文件和 smali 代码，提取应用程序中所有的组件信息. 然后通过广度优先遍历的方式并补充回调函数为每个组件生成函数调用图. 接着，将包含敏感 API 的组件函数调用图合并生成敏感组件函数调用图. 最后，通过污点分析技术识别组件间的调用关系，并基于此生成调用边补充完善敏感组件函数调用图(详见 3.3.1 节).

(2)基于恶意行为的特征提取模块. 基于已生成的敏感组件函数调用图和无法混淆的清单文件提取特征，用于后续的恶意软件分类. 具体而言，首先利用 Networkx[26] 提供的中心性算法计算敏感组件函数调用图中每个节点的中心性，然后挖掘敏感 API 与入口函数、交互函数的关系，并结合敏感 API 的中心性综合计算敏感 API 的重要性. 同时，反编译清单文件并提取敏感权限特征. 最后，合并敏感 API 重要性和敏感权限特征得到特征向量(详见 3.3.2 节).

(3)基于机器学习的分类模块. 基于不同的机器学习分类算法，利用提取生成的特征向量训练分类器并实现恶意软件分类. 具体而言，使用上阶段生成的特征向量分别训练基于 K 近邻算法、决策树算法、随机森林算法、卷积神经网络和投票算法的分类器，并比较分类器的综合性能. 最后使用分类器将应用程序分类为良性软件或恶意软件(详见 3.3.3 节).

3.3 系统实现

3.3.1 基于敏感组件的函数调用图提取模块

四大组件是安卓应用程序的基本功能组成单位，活动负责前台用户交互，服务用于执行后台耗时操作，广播接收器负责响应系统或用户自定义事件，而内容提供者用于管理应用程序的数据. 每个组件都是一个基本功能的实现单元，因此组件是应用程序语义的重要载体，可将其视作软件的基本构成单位. 每个组件对应各自的组件函数调用图，因此本文设计收集应用程序的组件信息，生成所有组件的函数调用图，并去除未调用敏感 API 的良性组件，将剩余敏感组件整合生成敏感组件函数调用图，提高恶意组件的可见性且极大地保留了应用程序的语义信息. 此外，一个应用程序只对应一个敏感组件函数调用图，所有样本都可以用于训练机器学习分类器，从而避免了划分区域策略导致的通用性不足问题.

为实现上述方案，基于敏感组件的函数调用图提取模块可分为组件信息提取、敏感组件函数调用图提取和组件调用关系分析等 3 个子模块，如图 4 所示.

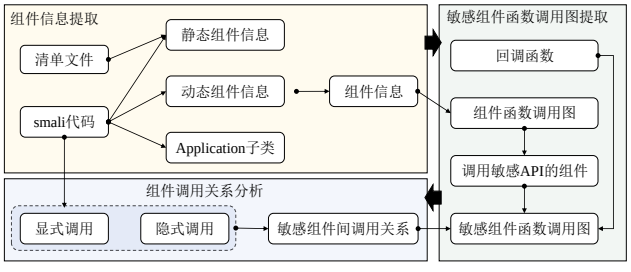

图 4 基于敏感组件的函数调用图提取模块框架图

(1)组件信息提取

四大组件信息一般存储于清单文件，但由于 Java 语言的特性和安卓系统的多样性，仅关注清单文件提取组件信息会有所遗漏. 针对此问题，本文首先通过清单文件和 smali 代码提取静态组件信息，然后分析 smali 代码提取动态组件信息，最后通过 smali 代码获取 Application 子类信息. 具体而言，Partdroid 首先利用 An-

droguard反编译清单文件,过滤组件标签提取已声明的组件列表,获取类列表,并在此基础上分析smali代码,剔除被重打包恶意软件移除但仍有清单文件相关信息的组件类. 本文将用户自定义并在清单文件中声明的类称为用户组件类,未声明的自定义类称为用户组件基类,系统定义的组件称为系统组件类. 为实现特定功能,用户组件类必须继承系统组件类,这一继承关系可能是直接的,也可能是通过用户组件基类间接实现的. 基于Java语言特性,若用户组件类继承自用户组件基类,Partdroid将两者视为一个整体. 遍历类列表,对每个组件类进行检查,若组件类不属于系统类,则递归查询其父类信息,直到找到系统类为止. 在查询过程中,将沿途经过的类名添加到一个列表中,完成查询后,将该列表视为一个整体组件,并将其添加到组件信息集合中. 然后继续分析下一个组件类至所有组件检查完毕,得到的组件信息集合即为提取的静态组件信息.

安卓应用程序中活动、内容提供者组件均在清单文件中声明,但广播接收器或服务可以使用静态注册或动态注册,而动态组件是软件在运行时注册的,清单文件中没有相关信息,因此本文利用动态广播接收器调用registerReceiver函数,动态服务调用startService函数或bindService函数的特点,采用污点分析的方式分析smali代码来提取动态注册组件信息. 以动态广播接收器为例,本文通过寻找从广播接收器类(污点源)到registerReceiver函数(污点槽)的可达路径,识别并存储动态注册的广播接收器信息. 具体而言,算法首先从应用程序中提取名为registerReceiver的函数,并收集所有调用该函数的函数列表. 然后,对于每个调用registerReceiver的函数,获取其对应的smali代码,并按顺序逐行分析. 在分析过程中,算法检查每个操作码和操作数,以确定是否存在与动态注册相关的操作并采取相应措施,进而提取动态组件信息.

如果一个安卓软件不包含四大组件时,它一定存在Application类的子类. 当软件无需与用户直接交互,而仅需执行某些操作时,可以利用Application子类来实现这一功能. 这种用法在恶意软件中较为常见,然而,Androguard工具无法直接从清单文件中提取此类信息,因此本文遍历应用程序所有的类,并检查其父类是否为“Landroid/app/Application;”. 若满足条件,则将该类视作一个组件.

最后,将上述提取得到的静态组件信息、动态组件信息和Application子类汇总得到应用程序的所有组件信息.

(2)敏感组件函数调用图提取

函数调用具有明确的方向性,本文用有向图$G=(V,E)$表示函数调用图,其中V为节点集合,表示函数,$E\subseteq\{(x,y)|(x,y)\in V^2\}$为节点间有向边集合,表示函数之间的调用关系,其中$(x,y)$表示$x$指向$y$. 则敏感组件函数调用图SCFCG$=(V',E')$,其中$V'$为所有调用敏感API的组件所提取的函数集合的并集,E'包括敏感组件内部的函数调用边及通过组件间关系分析识别出的组件间调用边. 为精确表示,函数调用图的节点包含包名、类名和函数名以避免函数重名干扰. 本文将敏感组件函数调用图提取分为3步:首先为每个组件生成组件函数调用图;然后判断组件属性,删除良性组件;最后合并剩余组件生成敏感组件函数调用图. 具体而言,Partdroid采用广度优先遍历算法分析每个函数及其调用的其他函数的调用关系. 它定义已分析列表和待分析列表,首先将组件的入口函数加入待分析列表,然后采用广度优先遍历算法分析函数调用关系,直到待分析列表为空. 在这个过程中,算法从待分析列表取出当前函数并检查该函数是否已分析,若未分析则处理其调用的函数,包括类的回调函数,并记录其调用关系,然后将非系统函数加入待分析列表以继续分析,而分析过的函数则加入已分析列表以避免重复,从而提高提取效率.

其中,回调函数在安卓应用程序中十分常见,且存在未加入回调函数时的组件函数调用图没有调用敏感API而回调函数调用了敏感API的可能,缺少分析回调函数可能导致敏感组件误判为良性组件,从而影响最终提取特征. 然而回调函数一般由事件驱动,不会被其他函数直接调用,从入口函数出发通过广度优先遍历无法处理回调函数. 针对此问题,本文根据文献[27]提取回调函数,当分析到某个函数时,遍历该函数所属类内所有函数并识别回调函数,将其加入组件函数调用图并以此为起点继续分析. 在这个过程中,为了提高组件函数调用图的提取效率,本文首先去除在smali代码中可以被其他函数直接调用的回调函数[27],然后根据从AndroZoo[28]随机选取的1 000个良性软件和1 000个恶意软件统计剩余6 476个回调函数的调用频率,并最终选取占总次数的95%以上的前380个出现次数最多的回调函数进行分析.

然后,在提取得到所有的组件函数调用图后,以该组件是否调用敏感API为标准,判断每个组件函数调用图是否属于敏感组件. 具体而言,若组件函数调用图中的节点含有敏感API,则该组件函数调用图对应的组件为敏感组件,反之为良性组件. 删除良性组件及其函数调用图,将所有敏感组件的函数调用图合并为一个敏感组件函数调用图,该图是整个软件函数调用图的子图,每个应用程序只能提取一个敏感组件函数调用图,从而解决了分区方法无法对每个区域添加标签的问题. 敏感组件函数调用图的生成示例如图5所示,步骤①中组件a、组件b、组件c和组件d的函数调用图分别对应图(a)、

图(b)、图(c)和图(d),步骤②中判断组件属性,将未调用敏感API的组件c及其函数调用图(c)删除,步骤③则将剩余的图(a)、图(b)、图(d)进行合并,生成敏感组件函数调用图,对应图(e). 从图(e)可以看出,图(a)与图(b)通过共同函数调用相关联,但均与图(d)无直接调用边,看似无关联. 然而,实际软件中组件间存在众多调用以确保功能运行,因此图(e)并不完整. 这是由于组件调用是事件驱动的,Androguard难以直接分析组件调用关系,导致当前敏感组件函数调用图缺失组件间的调用边.

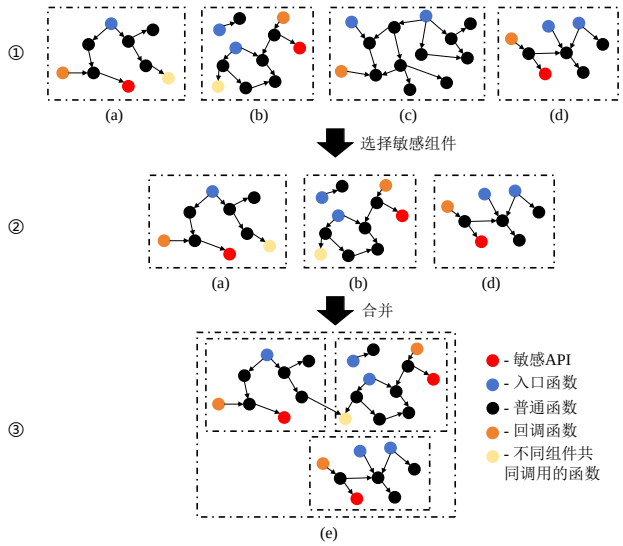

图5 敏感组件函数调用图生成示意图

(3)组件调用关系分析

对于上述问题,Partdroid通过污点分析方法分析组件间的显式调用和隐式调用[29],得到组件间的调用关系,并向敏感组件函数调用图补充相关的调用边. 具体而言,对于显式调用,以活动组件为例分析方法如算法1所示,Partdroid提取所有名为startActivity的函数,收集调用这些函数的函数列表并分析其smali代码,标记污点源和污点槽. 当代码操作码为new-instance,且创建的类为Intent,则记为污点源并存储于寄存器字典;当代码操作码为const-class,则表示可能是即将调用的组件类,记录在存储器字典中;当操作码为invoke-direct、调用函数为初始化函数且寄存器字典存在操作数1和操作数3时,表示该意图进行初始化,生成的意图对象存储于寄存器字典;当操作码为invoke-virtual、调用函数为startActivity时,则为污点槽. 此时污点源即为被调用的组件,生成当前函数与组件的调用边来补充完善敏感组件函数调用图.

在Partdroid中,隐式调用的分析是通过安卓系统中的已注册意图过滤器来实现的. 隐式调用通常发生在应用程序未明确指定调用目标组件,而是通过发送

算法1 活动组件显示调用关系分析

输入: 安卓应用程序apkPath

输出: 调用关系字典ret

1. 初始化空寄存器字典reg,通过apkPath获取调用startActivity的函数列表
2. FOR 函数列表的每个函数 DO
3. FOR 函数smali代码列表的每行代码 DO
4. IF (操作码 =="new-instance" and "Intent;" ∈ 代码) or 操作码 == "const-class" THEN
5. (reg.key, reg.value) ← (操作数1, 操作数2)
6. END IF
7. IF 操作码 == "invoke-direct" and "<init>" ∈ 代码 and 操作数3 ∈ reg and 操作数1 ∈ reg THEN
8. (reg.key, reg.value) ← (操作数1, reg[操作数3]初始化意图)
9. END IF
10. IF 操作码 == "invoke-virtual" and "startActivity" ∈ 代码 THEN
11. (ret.key, ret.value) ← (当前函数, reg[操作数2])
12. END IF
13. END FOR
14. END FOR

Intent与目标组件进行通信的场景. 为了捕获这类调用,Partdroid首先需要收集应用程序中注册的意图过滤器. 对于静态注册的组件,如四大核心组件(活动、服务、广播接收器和内容提供者),这些信息可以直接从应用程序的清单文件(AndroidManifest. xml)中提取. 这些组件的注册信息提供了关于应用如何与其他组件交互的重要线索. 另一方面,动态注册的组件,特别是广播接收器的注册,则需要通过分析应用程序的代码来提取. Partdroid使用污点分析技术,尤其是分析registerReceiver函数的IntentFilter参数,来捕捉动态注册的广播接收器.

在捕捉隐式调用的过程中,Partdroid利用污点分析确定隐式意图中的关键元素,Action和Category. 具体来说,当分析到操作码const-string定义的字符串时,Partdroid会将其视为污点源. 如果这个污点源出现在setAction或IntentFilter初始化函数的参数中,系统会将这些字符串与Action相关联;当出现addCategory参数时,系统则将其与Category相关联. 通过这种方式,Partdroid能够精确地追踪Intent的Action和Category信息,这对于正确匹配隐式意图至关重要. 每当系统分析到一个registerReceiver调用,Partdroid会把广播接收器类和对应的意图过滤器记录下来,为后续的调用边合并做准备.

一旦隐式意图中的Action和Category信息被提取并解析,Partdroid接着通过意图过滤器来匹配隐式意图. 在这个过程中,应用程序中声明的所有组件都会与收集到的意图过滤器进行比对. 如果某个组件的注册意

图与当前的隐式意图匹配,说明该组件在特定的操作或事件下被触发. 此时,Partdroid 会根据匹配的结果,将相关的组件与隐式意图之间的调用关系添加到敏感组件函数调用图中. 这样,尽管这些调用关系是隐式的,没有显式地指定目标组件,Partdroid 依然能够捕捉到这些跨组件的互动并在最终的调用图中呈现出来,确保在动态组件间的交互中,恶意行为不会被遗漏. 通过这一过程,Partdroid 显著增强了对复杂应用程序中组件间交互的捕捉能力,并确保了合并后的敏感组件函数调用图的完整性.

其中,本文用到的污点源和污点槽如表 1 所示,污点槽操作数后的数字表示 smali 代码对应的寄存器标号,如“Intent. init-1”表示 smali 代码调用 Intent 类的 init 函数,且污点源在第 1 个寄存器中.

表 1　污点分析中使用的污点源和污点槽

污点源操作码	污点源操作数	污点槽操作码	污点槽操作数
new-instance	Intent 类	invoke-direct	Intent.init-1
	IntentFilter 类	invoke-direct	IntentFilter.init-1
	广播接收器类	invoke-direct	该类 init-2
const-class	某个组件类	invoke-direct	Intent.init-3
		invoke-virtual	Intent.setClass-3
const-string	字符串	invoke-virtual	Intent.setPackage-2
			Intent.setAction-2
			Intent.addCategory-2
		invoke-direct	IntentFilter.init-2
-	Intent 对象	invoke-virtual	startActivity-2
			startActivityForResult-2
			startService-2
			bindService-2
			sendBroadcast-2
			sendStickyBroadcast-2
			sendOrderedBroadcast-2
-	IntentFilter 对象	invoke-virtual	registerReceiver-3

最后,根据分析得到的组件间调用关系,向敏感组件函数调用图中添加相关的调用边生成完整的敏感组件函数调用图. 一个完整的敏感组件函数调用图如图 6 所示,对应于图 5(e),其中黑色箭头表示函数之间的直接调用,蓝色箭头表示通过分析组件调用关系补充的调用边,图中一共添加了 4 条调用边.

3.3.2　基于恶意行为的特征提取模块

敏感 API 是恶意软件实施恶意行为的重要方式,也是检测恶意软件经常使用的特征. 此外,安卓系统中最重要的安全机制之一是权限管理机制,安卓应用程序调用敏感 API 前通常需要先取得相应权限的授权. 因此本文提取应用程序的 2 个特征,即敏感 API 重要性和敏感权限特征,以避免仅使用中心性算法提取特征

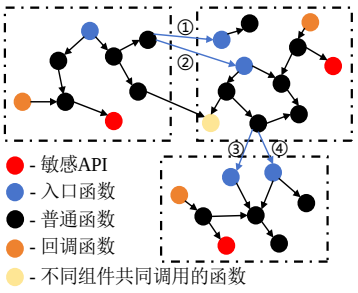

图 6　添加组件调用边后的敏感组件函数调用图示意图

导致的恶意行为语义丢失. 特征提取的流程如图 7 所示.

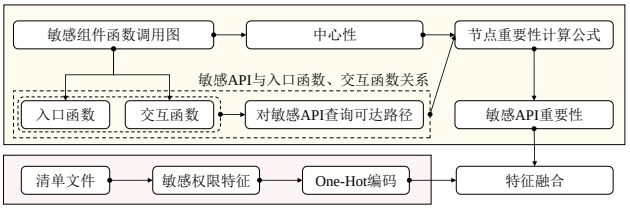

图 7　特征提取流程

(1)敏感 API 重要性计算

本文使用 Gong 等人[30]总结的 426 个敏感 API 进行计算,其中包括 260 个恶意行为强相关的敏感 API、112 个需要得到危险权限授权的受限 API 和 54 个可以触发敏感操作进行攻击的敏感操作 API,因此提取的特征向量为 426 维. 本文将组件的生命周期函数视为入口函数,入口函数不需要用户交互,可以自动触发,此外本文收集了 35 个 UI 组件的交互函数,它们需要用户操作后才能触发,具体函数如表 2 所示.

本文根据从 AndroZoo[28]随机选取的 1 000 个良性软件和 1 000 个恶意软件统计恶意软件与良性软件自动触发敏感 API 频率的比例,结果显示恶意软件自动触发敏感 API 的概率要大于良性软件,利用这一特点与中心性算法结合,来综合计算敏感 API 的重要性. 具体而言,Partdroid 首先在敏感组件函数调用图中从所有节点筛选出入口函数和交互函数,然后利用中心性算法计算每个敏感 API 的中心性,接着对每个敏感 API 查询入口函数或交互函数之间是否存在可达路径,最后根据下列公式综合计算敏感 API 的重要性.

$$\text{importance(API)}=\text{centrality(API)}\times\frac{E^{*}\alpha+I}{E+I}\times\left(\frac{E}{N}+1\right)\quad(1)$$

$$(\alpha>1)$$

其中,N 表示入口函数总数;centrality(API) 为敏感 API 的中心性;当一个入口函数与该敏感 API 在函数调用图中存在可达路径时,记为 E_i=(entrypoint, sensitiveAPI),总数为 E;当一个交互函数与该敏感 API 在函数调用图中存在可达路径时,记为 I_i=(Interaction, sensitiveAPI),

表2 入口函数和交互函数

入口函数	交互函数		
onCreate	onMenuItemClick	onKeyDown	onClick
onStart	onPreferenceClick	onPictureTaken	onKey
onResume	onOptionsItemSelected	onLongClick	onTouch
onPause	onContextItemSelected	onRatingChanged	onTabChanged
onStop	onCheckedChanged	onItemSelected	onEditorAction
onDestory	onItemLongClick	performLongClick	onTabSelected
onRestart	onRadioButtonClicked	performItemClick	onDrag
onBind	onQueryTextSubmit	onTabReselected	performClick
onUnbind	onCheckboxClicked	onKeyLongPress	onClose
onRebind	onLocationChanged	onToggleClicked	onItemClick
onStartCommnd	onProgressChanged	onListItemClick	onTextChanged
onReceive	onQueryTextChange	afterTextChanged	

总数为I;α为重要因子且α值大于1,交互函数与敏感API含有交互通路的重要因子为1. 本文通过实验测试发现当α取值为6时大多数分类算法表现较好,因此将α的初始值取为6.

(2)敏感权限特征提取

恶意软件执行恶意行为需要软件声明且用户授予相应的敏感权限,因此重打包恶意软件往往比良性软件申请更多的敏感权限. 本文从北京交通大学团队的数据集[4]中收集88个与恶意软件关联较大的敏感权限特征,并通过解析应用程序清单文件提取敏感权限特征. 具体而言,Partdroid通过Androguard工具反编译二进制XML格式的清单文件,提取应用程序的权限特征,并使用One-Hot编码形式表示. 对于应用程序声明的权限,将特征向量赋值为1,未声明的权限特征向量赋值为0,生成88维特征向量.

最后将第一阶段提取的426维敏感API特征与第二阶段提取的88维敏感权限特征合并,构成每个样本的514维特征向量,并标注标签来区分良性和恶意样本,作为模型训练数据,其中标签为1表示恶意样本,0表示良性样本.

3.3.3 基于机器学习的分类模块

完成上述特征提取后,本文使用不同的分类算法来训练安卓恶意软件分类器,包括K近邻、决策树、随机森林、卷积神经网络和投票算法等. 模型的选择综合考虑了特征类型、模型适配性与任务特点,具体而言,Partdroid提取的特征向量包含426维的敏感API连续特征与88维的离散敏感权限特征,整体呈现稀疏、异构、部分维度不相关等特征结构. 考虑到特征之间既具有明确语义分割,又存在潜在组合模式,分类模型应具备对连续变量的鲁棒建模能力、对离散变量的分裂处理能力以及一定的特征冗余容忍性. K近邻算法作为轻量级惰性学习方法,适用于特征空间分布清晰的初步评估;决策树具有可解释性强、适应离散特征的优势,适合权限类特征分类;随机森林通过集成多棵树提升泛化能力,天然适应本文的特征类型;卷积神经网络则可通过1D卷积捕捉特征之间的局部关系,辅助挖掘深层组合模式;而投票算法集成学习策略整合多模型优势,提高整体鲁棒性. 相比如XGBoost等复杂集成模型,本文优先采用结构简单、易部署、结果可解释的方法作为对比基线,并为后续强化模型能力提供可扩展空间.

本文基于提取的软件特征,使用sklearn库分别构建K近邻、决策树、随机森林这3个分类模型,并通过调节模型关键参数得到最优的分类模型. 卷积神经网络是深度学习中常用的模型,本文提取的514维特征均为数字,没有空间结构,因此本文使用一维卷积神经网络. 本文使用二元交叉熵作为损失函数,该函数常用于二分类问题,能有效地刻画数据分布,优化算法并提高预测精度;使用ReLU激活函数,该函数计算速度快,参数数量少,在训练过程中能自动学习数据的非线性特征. 另外,影响卷积神经网络性能的主要参数有卷积核大小、卷积核数量和卷积层数,因此本文调节这些参数,并衡量准确率、参数总量、每epoch平均训练时间来得到综合性能最优的卷积神经网络模型.

此外,投票算法综合多个分类器的预测结果来提高整体的分类准确率. 因为sklearn中无法将卷积神经网络作为投票的基算法,本文选择K近邻、决策树和随机森林算法作为投票算法的基算法,其流程如图8所示. 本文在训练数据时采用十折交叉验证来验证测试的结果. 具体而言,十折交叉验证将数据集平均分为10份,轮流将其中9份作为训练数据,剩余1份作为测试数据,依次进行10次实验,并对这10次的实验结果取平均值. 通过这种方式,能够充分利用数据集,减少

因数据划分不合理而导致的模型性能偏差.

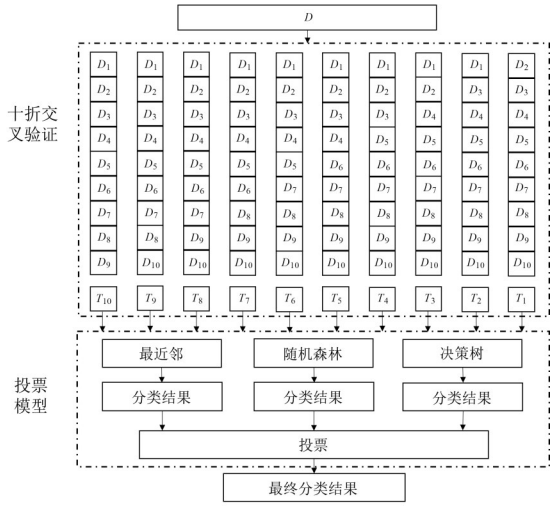

图8　投票算法流程图

最后,利用训练好的分类器对待测应用程序进行分类,即将提取的待测软件特征输入分类器,进而识别其是否为安卓重打包恶意软件.

4　实验与分析

4.1　实验设置

4.1.1　数据集和评估指标

本文采用2个数据集评估Partdroid综合性能,分别是重打包恶意软件数据集和普通恶意软件数据集.

重打包恶意软件数据集:本文基于文献[31]收集的1 497对良性软件和重打包恶意软件,爬取VirusTotal[32]检测报告,并设置将被2个以上反病毒扫描程序标记为恶意的应用程序归为恶意样本,没有被标记恶意的应用程序则为良性样本,最终选取了1 220个良性样本和1 160个恶意样本. 该数据集的基本信息如表3所示,其中良性样本最大为126 426.42 kb,最小为16.52 kb,dex最大为8 580.57 kb,最小为5.83 kb;恶意样本最大为79 943.08 kb,最小为52.79 kb,dex最大为6 040.03 kb,最小为1.24 kb.

表3　重打包恶意软件数据集基本信息

分类	数量/个	平均总大小/kb	平均dex大小/kb
良性样本	1 220	11 177.45	1 131.49
恶意样本	1 160	11 448.58	1 235.61
综合	2 380	11 309.60	1 182.23

普通恶意软件数据集:本文从AndroZoo[28]2012—2022年份的安卓软件中随机选取2 449个恶意软件和2 362个良性软件,该数据集的基本信息如表4所示,其中良性样本最大为45 309.56 kb,最小为23.11 kb,dex最大为17 608.08 kb,最小为3.79 kb;恶意样本最大为88 974.26 kb,最小为112.79 kb,dex最大为7 094.05 kb,最小为52.79 kb.

表4　普通恶意软件数据集基本信息

分类	数量/个	平均总大小/kb	平均dex大小/kb
良性样本	2 362	5 558.24	3 033.65
恶意样本	2 449	6 526.04	1 254.93
综合	4 811	6 050.89	2 128.20

评估指标:本文采用准确率(Accuracy)、精确率(Precision)、召回率(Recall)、F1值(F_1-score)、假阳率(False Positive Rate)和假阴率(False Negatives Rate)等6个指标对实验效果进行评估.

4.1.2　实验细节和参数设置

本文实现了重打包恶意软件检测工具Partdroid,使用的编程语言为Python3.6.9,且在实验过程中使用了Networkx、Androguard等python库和Jadx[33]、Apktool[34]等分析工具. 在基于机器学习的分类模块中,根据sklearn构建K近邻、决策树、投票算法和随机森林等分类模型,以完成训练和检测任务,其中训练集与测试集比例为3∶1. 实验机器配置为Intel(R) core(TM) i7-8700 CPU 3.20 GHz、1 T机械硬盘、16 GB内存和Ubuntu 18.04.

为获得最优参数,本文采用F1值和准确率对K最近邻、决策树、随机森林、投票算法的不同参数进行效果比较,采用准确率、参数总量、每epoch平均训练时间对卷积神经网络的不同参数进行效果比较. 最终,根据模型综合性能,本文将K近邻算法中的邻居数k值设为5;决策树算法中的max_depth参数设为6,criterion参数设为“entropy”,splitter参数设为“best”;随机森林算法中的n_estimators参数设为90,max_depth参数设为10;卷积神经网络中的卷积层数设为3,卷积核大小设为(7,5,3),卷积核数量设为(32,32,32);投票算法中的voting参数设为“hard”. 此外,本文采用度中心性算法、接近中心性算法、介数中心性算法进行实验,实验结果表明度中心性算法在不同分类模型下F1值均优于其他中心性算法,因此本文采用度中心性算法.

4.2　检测结果

为了测试Partdroid的检测效果并选取最优分类器,本文分别采用K近邻、决策树、随机森林、卷积神经网络和以K近邻、决策树、随机森林算法为基模型的投票算法训练分类器,其检测结果如表5所示.

F1值和准确率衡量Partdroid在分类安卓良性软件和安卓重打包恶意软件的综合性能,值越高说明分类器综合性能越好;召回率衡量分类器对恶意软件检测的覆盖程度,值越高越不会遗漏恶意样本;假阳率和假阴率衡量分类器对恶意软件检测的误报率和漏报率,值越低表明分类器识别效果越高. 实验表明,随机森林

表5 Partdroid在不同分类算法下的实验效果 单位:%

指标	F1值	准确率	精确率	召回率	假阳率	假阴率
K近邻算法	85.44	85.92	86.25	84.84	12.95	15.16
决策树算法	89.32	90.13	94.89	84.48	4.47	15.52
随机森林算法	91.34	91.93	96.74	86.56	2.80	13.44
卷积神经网络	89.88	90.29	94.73	85.42	4.57	14.58
投票算法	91.63	92.15	94.86	88.65	4.49	11.35

的F1值、准确率召回率等指标在单一分类算法中最优,其F1值和准确率达到了91.34%和91.93%,优于K最近邻、决策树和卷积神经网络,因此随机森林在单一分类算法中的综合性能最好;而投票算法虽然精确率和假阳率略差于随机森林,但在F1值、召回率、准确率和假阴率上优于随机森林,因此投票算法的整体性能要优于随机森林,这说明综合多种分类算法要优于单一分类算法.

4.3 与现有工作对比

为进一步评估本文方法的有效性,本文选择了3种同类工具与Partdroid的检测效果进行对比,分别为基于语法的检测方法Perdroid[4],基于语义的检测方法Malscan[8]和Intdroid[10]. 表6给出了Partdroid与其他恶意软件检测方法在重打包恶意软件数据集上的检测结果.

表6 不同检测方法在重打包恶意软件数据集上的实验效果

单位:%

指标	F1值	准确率	精确率	召回率	假阳率	假阴率
Perdroid	84.11	84.24	82.76	85.64	17.06	14.36
Malscan	88.63	89.33	92.23	85.45	6.81	14.55
Intdroid	85.95	87.31	93.28	79.75	5.50	20.25
Partdroid	91.34	91.93	96.74	86.56	2.80	13.44

实验结果显示,Partdroid的F1值、准确率、召回率等指标均优于其他方法,证明了本文方法的有效性. 其中,Perdroid提取权限信息并对权限进行排序,选择88个危险权限特征构建分类器,但因其仅依赖语法特征导致较高的假阳率和较低的综合性能;Malscan基于语义特征,通过提取21 986个敏感API的中心性特征训练分类器,但因其提取整个软件的函数调用图,恶意行为易被大量良性行为掩盖而导致假阴率高于Partdroid,且因其仅使用中心性特征,易因相似的中心性造成误报导致假阳率高于Partdroid;而Intdroid选择中心性靠前的节点作为中心节点,根据敏感API与中心节点间的距离计算亲密度并以此作为特征训练分类器,但其综合性能低于Malscan和Partdroid,且假阴率最高,这可能是因为有些安卓重打包恶意软件嵌入的恶意代码与原本良性软件的代码耦合性很弱,Intdroid选择的中心性靠前的节点大多是良性软件的节点,恶意代码调用的敏感API可能与所有中心节点都不存在通路,导致敏感API的特征为0而造成漏报.

相比同类工具,Partdroid提取敏感组件函数调用图来凸显恶意样本特征,并考虑敏感API与入口函数、交互函数之间的关系和权限特征来提取函数调用图恶意行为的语义信息,缓解中心性相似造成的误报问题. 因此,Partdroid的综合性能优于同类工具,可以更精准地检测到更多的安卓重打包恶意软件.

4.4 消融实验

为了评估Partdroid中各个模块对于重打包恶意软件检测的有效性,本文在重打包恶意软件数据集上设置了一系列消融实验. 实验结果如图9所示,其中Partdroid-G用于评估敏感组件函数调用图提取模块的有效性,该方法提取整个应用程序的函数调用图,并使用本文特征提取模块提取特征;Partdroid-A用于评估敏感API重要性特征的有效性,该方法使用本文的函数调用图提取模块提取敏感组件函数调用图,并提取权限特征和直接使用中心性算法计算敏感API的中心性.

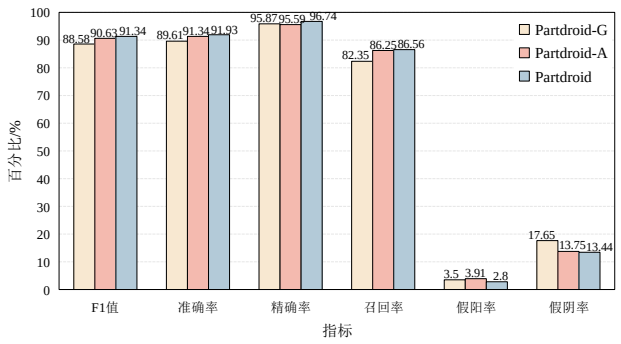

图9 消融实验结果对比

实验结果显示,Partdroid的F1值和准确率均达到最高,综合性能最好,Partdroid-A次之,Partdroid-G的综合性能最差. 这是因为Partdroid-G提取的是整个应用程序的函数调用图,而Partdroid和Partdroid-A提取的是敏感组件函数调用图,受良性行为干扰较小. 而Partdroid的综合性能优于Partdroid-A是因为Partdroid-A直接采用中心性算法计算敏感API的中心性作为特征,忽略了函数调用图中恶意行为的语义.

综上所述,本文设计的2个模块均有助于重打包恶意软件检测,其中敏感组件函数调用图对检测安卓重打包恶意软件的影响较大,提高分类器的综合性能较为明显;而敏感API和入口函数、交互函数的关系与中心性算法的结合计算敏感API重要性虽然也改善了分类器的综合性能,但提升幅度较小.

4.5 时间开销

为了评估Partdroid的检测效率,本文对实验的时间开销进行分析. 本文选择4.1.1节中的2个数据集作

为测试对象,总共包含了3 582个良性软件和3 609个恶意软件,且Partdroid的整体执行流程分为3个环节:敏感组件函数调用图提取、敏感API重要性计算和敏感权限特征提取和分类检测. 样本dex大小与提取敏感组件函数调用图时间的关系如图10所示,大多数样本的dex大小与提取时间成正比,而一些dex较小却需要较长时间的样本,经分析发现这些样本的组件数量较多,而Partdroid需要对每个组件生成组件函数调用图并验证其是否调用敏感API,因此当应用程序dex大小相等的情况下,组件数量越多,Partdroid提取敏感组件函数调用图的效率越低. 对于95%的样本,Partdroid可以在29.5 s内获取对应的敏感组件函数调用图,如图11所示,平均11.25 s可以完成一个样本分析.

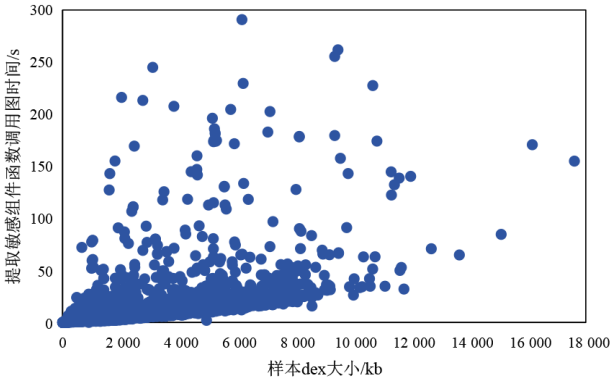

图10　提取敏感组件函数调用图时间开销与dex大小的关系

对于敏感API重要性计算和敏感权限特征提取,单个样本的提取时间最慢为13.26 s,最快为0.36 ms,敏感组件函数调用图越大,提取特征的时间越长. 95%的应用程序可以在1.4 s内完成特征向量提取. 如图11所示,平均0.48 s可以提取一个应用程序的特征.

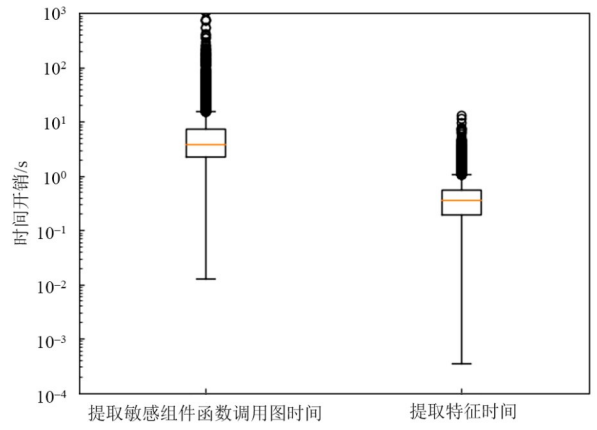

图11　提取敏感组件函数调用图和提取特征的时间开销

表7给出了Partdroid在不同分类算法下的运行时间开销,其中K近邻算法的训练时间最短,训练一个样本平均只需0.001 ms,卷积神经网络的平均训练时间最长,因为它训练时需迭代50轮,而综合性能最好的随机森林则需要0.112 ms. 对于分类时间,决策树算法的平均耗时最短,卷积神经网络的分类时间最长,随机森林的分类时间适中.

表7　不同分类算法下一个样本的平均训练和平均分类时间

单位:ms

分类算法	平均训练时间	平均分类时间
K近邻	0.001	0.103
决策树	0.025	0.001
随机森林	0.112	0.034
卷积神经网络	21.429	0.199
投票算法	0.134	0.193

在表8中,展示了Partdroid与现有同类工具(如Perdroid、Malscan和Intdroid)在静态分析、特征提取和模型分类3个阶段的时间开销对比. 可以看出,Partdroid在静态分析阶段的时间开销显著高于其他工具,这主要是因为Partdroid的分析流程包括组件信息的提取、敏感组件函数调用图的生成以及组件间调用关系的分析. 相比之下,其他工具如Perdroid并不进行这些复杂的步骤,因此其静态分析时间相对较短. 在特征提取阶段,由于Perdroid的特征维度较低,且采用了相对简单的特征提取方法,因此耗时最短. Partdroid需要处理更多的特征维度,并进行更复杂的特征提取和计算,这使得其特征提取过程的时间开销较高. 尽管如此,Partdroid在检测精度和综合性能上的优势明显,能够弥补其在特征提取阶段较长的时间开销.

表8　不同工具检测一个软件的时间开销

工具	Partdroid	Perdroid	Malscan	Intdroid
静态分析/s	11.259 663	4.171 157	4.867 797	4.867 797
特征提取/s	0.480 665	0.000 023	0.956 045	29.542 671
模型训练/ms	0.112 031	0.041 707	0.010 190	0.010 190
预测分类/ms	0.033 509	0.020 319	1.118 711	1.118 711
总时长/s	11.740 339	4.171 200	5.824 961	34.411 587

在模型分类阶段,Perdroid和Partdroid都采用了随机森林算法作为分类器. 由于Perdroid的特征维度较低,训练时间明显短于Partdroid. 后者在特征维度上进行了更复杂的计算,因此训练和分类所需的时间更长. 而Malscan和Intdroid使用了K近邻算法,虽然这两者的训练和分类时间一致,但由于K近邻算法的计算复杂度较高,其整体运行时间也超过了使用随机森林的Partdroid. 尽管Partdroid在时间开销方面存在一定劣势,但在论文中已经提出了优化策略. 针对静态分析阶段的时间开销问题,未来将考虑对组件分析过程进行并行化处理,通过模块化将不同组件的调用图生成和

分析过程并行执行,从而显著减少静态分析阶段的时间开销. 此外,特征提取过程中,也可以探索更高效的算法,优化特征提取方法,减少计算复杂度,以提高整体运行效率. 综上所述,虽然Partdroid在时间开销上存在一定劣势,但其在准确性、精确度以及综合性能方面的优势十分显著. 未来,将通过并行化分析和特征提取优化等手段,进一步提升其运行效率,同时保持其高检测精度和全面性.

综上所述,分析应用程序的平均总时长从短到长排序为Perdroid、Malscan、Partdroid、Intdroid,检测安卓重打包恶意软件的综合性能从高到低排序为Partdroid、Malscan、Intdroid、Perdroid. Partdroid在检测安卓重打包恶意软件上综合性能最好,时间开销适中.

为进一步提升Partdroid在实际场景中的检测效率,本文对当前系统中的时间开销问题进行了深入分析,并提出了3项具有可操作性的优化策略. 首先,在静态分析阶段,Partdroid对每个组件执行了组件函数调用图构建、回调函数识别以及跨组件调用边分析等操作,由于现阶段这些流程是串行执行,无法充分利用多核处理器资源,导致整体分析耗时较高. 为此,建议将组件分析过程进行模块化重构,使每个组件的分析任务作为独立子任务进行并发处理. 通过引入线程池并行调度机制,在多核环境下可以显著降低整体分析时长,尤其是在分析包含大量组件的大型应用时,该策略预期可获得近线性的加速比.

其次,在组件间调用边的补全过程中,现有实现中对所有组件一视同仁地执行污点分析,虽然提升了图的完整性,但带来了较高的计算开销. 为优化此阶段的效率,可引入轻量级的敏感性预判机制,在提取组件函数调用图后,通过敏感API初筛或调用模板匹配方式,快速识别出需要进行污点分析的组件,从而避免对无风险组件执行冗余分析. 该方法可有效降低污点传播分析的频次,缩短静态分析时间,同时不会影响恶意行为路径的识别完整性.

最后,在特征提取阶段,Partdroid当前提取的特征包括426维敏感API特征与88维权限特征,总计514维,虽然提供了丰富的行为信息,但也在一定程度上增加了模型训练与推理的时间开销. 为提升整体效率,可引入特征选择机制,例如通过信息增益、Gini系数等评价指标评估各特征的重要性,提前筛除对分类贡献度较低的特征维度. 同时,可结合主成分分析等降维算法进一步压缩特征空间,从而减少后续分类器的计算负担,提高训练和检测效率.

综上,尽管Partdroid在静态分析阶段存在一定时间开销劣势,但通过上述优化策略,在不牺牲检测准确率和泛化能力的前提下,有望实现运行效率的显著提升. 这些优化方案在理论上具备良好的可行性,未来工作将重点推进系统重构与并发机制实现,以使Partdroid更适应大规模应用环境中的实际部署需求.

4.6 普通恶意软件检测

为了评估Partdroid检测普通恶意软件的能力,本文按照上述实验流程采用普通恶意软件数据集进行了相同实验,采用最优参数的不同分类算法的实验结果如表9所示.

表9 Partdroid检测普通恶意软件的实验效果 单位:%

指标	F1值	准确率	精确率	召回率	假阳率	假阴率
K近邻算法	95.83	95.70	94.40	97.30	5.93	2.70
决策树算法	95.76	95.67	95.41	96.11	4.74	3.89
随机森林算法	97.12	97.13	99.14	95.18	0.84	4.82
卷积神经网络	96.30	96.16	95.53	97.08	4.81	2.92
投票算法	96.97	96.97	98.46	95.54	1.57	4.46

实验结果显示,在单一算法中,随机森林算法的F1值、准确率和精确率最高,这说明随机森林在5种分类算法中综合性能最好,且随机森林算法的假阳率最低,表示不易产生误报. 而投票算法的F1值、准确率和精确率略低于随机森林,这可能是因为K近邻和决策树算法的综合性能较低,综合后的投票算法性能略差. 但K近邻和决策树算法的假阴率较低,因此综合后投票算法的假阴率要优于随机森林算法. 因此,使用Partdroid检测安卓普通恶意软件时,可以根据需求选择随机森林算法或投票算法.

表10给出了Partdroid与3种其他恶意软件检测方法在普通恶意软件数据集上的检测结果. 可以看出与重打包恶意软件检测相似,Partdroid的综合性能优于Perdroid和Malscan. 但与之前不同的是,因为普通恶意软件不是基于良性软件嵌入,Intdroid选取中心节点与敏感API的距离作为亲密度,更容易发现伪装的恶意软件,因此Intdroid的F1值和准确率达到了97.47%和97.42%,优于Partdroid. 尽管如此,Intdroid的时间开销比其他工具要大,且Partdroid的精确率达到了99.14%,这说明Partdroid在普通恶意软件检测方面也表现出不错的能力.

4.7 新恶意软件检测

为了评估Partdroid在检测新型恶意软件方面的能

表10 不同检测方法对普通恶意软件的实验效果 单位:%

指标	F1值	准确率	精确率	召回率	假阳率	假阴率
Perdroid	93.66	93.52	93.08	94.28	7.26	5.72
Malscan	96.65	96.51	95.45	97.88	4.92	2.12
Intdroid	97.47	97.42	98.13	96.83	1.94	3.17
Partdroid	97.12	97.13	99.14	95.18	0.84	4.82

力,本文利用重打包恶意软件样本和普通恶意软件样本,训练了基于随机森林算法的恶意软件分类器,并从谷歌应用商店中随机爬取了涵盖不同类别(如财务、游戏、工具等)应用程序的2 000个安卓应用. 所选取的应用程序发布时间范围为2022年初至2023年下半年,覆盖了近年来主流的安卓应用版本,确保样本具有一定的时效性,并反映了当前安卓生态中的应用发展趋势和特征. 由于这些应用程序在谷歌应用商店中并未标注为恶意软件,本文通过Partdroid模型在这些样本上的高置信度输出进一步筛选出了潜在的可疑软件. 具体而言,根据分类器的输出结果,选取了具有高风险提示的应用进行手动分析,并通过第三方平台(如VirusTotal)进行验证.

这一过程中,部分应用表现出典型的恶意行为特征,进一步证明了所提方法对新型恶意软件的检测能力和泛化能力. 在这2 000个应用程序中,Partdroid共触发了21条警报,提示存在潜在恶意软件. 通过与VirusTotal的对比分析,结果显示其中有4个应用被标记为恶意软件. 这些恶意软件的特征包括但不限于异常权限请求、对敏感API的调用、恶意代码的嵌入等,进一步验证了本方法在识别新型恶意软件方面的有效性. 虽然银行木马、勒索软件等特定恶意软件家族在应用商店中较少出现,且这些类型的恶意软件往往具有高度的隐蔽性,传播途径更加狭窄,因此本研究中的样本未能涵盖这些特定恶意软件家族. 然而,通过本次实验的结果可以证明,Partdroid在检测广泛应用样式和不同类别的恶意软件时具有很好的时效性和泛化能力. 为了进一步加强方法的适应性,未来的研究将继续扩展样本集,尤其是引入更多具代表性的恶意软件家族(如银行木马、勒索软件等)进行检测,全面评估Partdroid在各种恶意软件中的表现. 综上所述,本实验通过对2 000个最新发布的安卓应用程序进行检测,不仅验证了Partdroid在实际环境中的时效性,还证明了其在检测新型恶意软件方面的有效性.

com.maaxhdc.maaxhdc是一款视频播放器,被Virustotal的2个安全厂商标记为恶意. 它调用多个敏感API获取用户的敏感信息,如MAC地址、SIM卡序列号等,风险代码如图12所示,且它多次使用反射调用函数. Falcon沙箱检测该应用程序的威胁分数为100,因此它是有潜在风险的应用程序.

com.ebe.mobilebanking是一款手机银行软件,Virustotal中有6个安全厂商报告该应用程序含有恶意行为,Hybrid的Falcon沙箱报告该应用程序的威胁分数为50,经分析发现它存在手机重启后自动执行代码的能力,如图13所示,因此该应用程序具有潜在的风险.

```
if (connectionInfo != null) {
    String macAddress = connectionInfo.getMacAddress();
    if (macAddress == null) {
        macAddress = "";
    }
    networkInterface.hardwareAddress = macAddress;
}

if (((ApplicationContext) fREContext).controller().hasAccess()) {
    Logger.d(TAG, "Phone: Has permission", new Object[0]);
    String line1Number = ((TelephonyManager) fREContext.getActivity().getSystemService("phone")).getLine1Number();
```

图12　com.maaxhdc.maaxhdc存在风险的代码

```
<uses-permission android:name="android.permission.USE_FINGERPRINT"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-feature android:name="android.hardware.telephony" android:required="false"/>
<application android:label="EBank Mobile" android:icon="@drawable/icon" android:name="and
```

图13　com.ebe.mobilebanking存在手机重启自动执行代码能力

com.boltrend.floen是一款3D角色扮演类游戏,被Virustotal的4个安全厂商标记为恶意. 通过分析代码发现该应用会监听用户SIM卡状态,并获取用户位置、SIM卡序列号等敏感信息,且申请了可以修改系统设置的android.permission.WRITE_SETTINGS权限,如图14所示. Falcon沙箱还将libcri_ware_unity.so标记为蠕虫和僵尸网络,因此它也是潜在的恶意软件.

此外,Partdroid检测到的air.com.maidmarian.sherwood虽被Virustotal的3个安全厂商标记为恶意,但Falcon沙箱检测该样本的恶意指标为0,且分析代码发现它并没有获取用户的敏感信息,调用了一些操作文件的敏感API也只用于适配低版本的安卓系统,没有明显的恶意行为,因此可能是Partdroid的误报.

从谷歌应用商店随机下载的2 000个安卓应用程序中,Partdroid检测出了3个疑似的恶意软件. 当前工作已初步评估了Partdroid方法在近2年发布的安卓应

```
if (simId == 0) {
    TelephonyManager tm = (TelephonyManager) getApplicationContext().getSystemService("phone");
    return tm.getSimState();
}

String networkOperatorName = ((TelephonyManager) systemService).getNetworkOperatorName();
Intrinsics.checkNotNullExpressionValue(networkOperatorName, "telephonyManager.networkOperatorName");
carrierName = networkOperatorName;

Location location = loctionManager.getLastKnownLocation(currentProvider);
if (location == null) {
    return null;
}
GPSLocation gpsLocation = new GPSLocation();
gpsLocation.longitude = (long) (location.getLongitude() * 1000000.0d);
gpsLocation.latitude = (long) (location.getLatitude() * 1000000.0d);
return gpsLocation;

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
```

图 14 com.boltrend.floen存在的风险代码

用样本中的泛化能力,通过随机选取的2 000个应用程序,展示了该方法在常见应用场景下的有效性. 虽然本研究聚焦于安卓重打包恶意软件和普通恶意软件的检测,并在实际环境中证明了Partdroid的时效性和泛化能力,但本文未特别集中于银行木马和勒索软件等特定新型恶意软件家族. 原因在于,银行木马和勒索软件等类型的恶意软件通常具备高度的隐蔽性和针对性,其分布较为局限,且往往通过定向攻击或在特定平台上传播,而非通过主流的应用商店发布. 因此,这类恶意软件的样本较为稀缺,不是当前实验的重点关注对象. 未来的工作将进一步拓展恶意软件样本库,计划引入更多代表性的恶意软件家族,特别是银行木马、勒索软件等,以验证Partdroid在面对这些新型、高度隐匿的恶意软件时的适应性与时效性. 通过对这些恶意软件的引入,能够更全面地评估该方法在不同类型恶意软件检测中的表现,进一步证明其广泛适用性和强大的泛化能力.

5 结论

本文提出一种安卓重打包恶意软件检测系统Partdroid,该方法提出了一种基于敏感组件的函数调用图提取模块,通过分析清单文件和smali代码,提取应用程序的组件信息,并为每个组件生成组件函数调用图,将调用敏感API的组件的函数调用图合并生成敏感组件函数调用图,利用污点分析的方法发掘组件间调用关系来补充敏感组件函数调用图,以此来凸显恶意行为特征,缓解恶意行为被大量良性行为掩盖的影响,并避免函数调用图分区的局限性. 同时,本文提出了基于恶意行为的特征提取模块,通过挖掘敏感API与入口函数、交互函数的关系,结合中心性算法综合计算敏感API的重要性,并提取不易被混淆的权限特征,避免直接使用中心性算法提取特征导致丢失恶意行为语义特征的问题.

实验结果表明,Partdroid在检测安卓重打包恶意样本时F1和准确率分别为91.34%和91.93%,综合性能优于同类工具. 消融实验证明了敏感组件函数调用图和计算敏感API重要性对检测系统综合性能提升的有效性. 同时在检测普通恶意软件上也表现良好,虽综合性能略逊于Intdroid,但其精确率高达99.14%. 此外,Partdroid在新恶意软件检测中表现突出,从谷歌应用商店随机选取的2 000个应用中成功识别出3个可疑软件. 此外,本文还有一定提升空间,例如考虑现有代码混淆中的虚拟化混淆、动态加载代码技术等[35]可能被恶意软件开发者利用的代码保护措施和第三方代码库的影响,以及结合反射调用[36]等进一步完善敏感组件函数调用图. 未来工作可以考虑抗代码混淆的检测方法,结合第三方代码库检测[37~39]等来减少代码保护措施和第三方代码库带来的影响,同时可以在平衡函数调用图的准确性和时间开销的情况下添加更多调用关系,来获取更完整的函数调用图,进而提取更准确的特征.

参考文献

[1] IDC. Smartphone market share[EB/OL]. (2024-5-14)[2024-12-30]. https://www. idc. com/promo/smartphone-market-share.

[2] KASPERSKY. IT threat evolution in Q3 2023. Mobile statistics [EB/OL]. (2023-11-1) [2024-12-30]. https://securelist.com/it-threat-evolution-q3-2023-mobile-statistics.

[3] DAHIYA A, SINGH S, SHRIVASTAVA G. Android malware analysis and detection: A systematic review[J]. Expert Systems, 2025, 42: e13488.

[4] WANG W, WANG X, FENG D W, 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.

[5] KIM T, KANG B, RHO M, et al. A multimodal deep learning method for Android malware detection using various features[J]. IEEE Transactions on Information Forensics

and Security, 2019, 14(3): 773-788.

[6] 潘建文, 张志华, 林高毅, 等. 基于特征选择的恶意Android应用检测方法[J]. 计算机工程与应用, 2023, 59(21): 287-295.
PAN J W, ZHANG Z H, LIN G Y, et al. Android malware detection based on feature selection[J]. Computer Engineering and Applications, 2023, 59(21): 287-295. (in Chinese)

[7] MARICONTI E, ONWUZURIKE L, ANDRIOTIS P, et al. MaMaDroid: Detecting Android malware by building Markov chains of behavioral models[C]//Proceedings 2017 Network and Distributed System Security Symposium. San Diego: NDSS, 2017: 3313391.

[8] WU Y M, LI X D, ZOU D Q, et al. MalScan: Fast market-wide mobile malware scanning by social-network centrality analysis[C]//2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE, 2019: 139-150.

[9] MAFAKHERI A, SULAIMANY S. Android malware detection through centrality analysis of applications network[J]. Applied Soft Computing, 2024, 165: 112058.

[10] ZOU D Q, WU Y M, YANG S R, et al. Intdroid: Android malware detection based on API intimacy analysis[J]. ACM Transactions on Software Engineering and Methodology(TOSEM), 2021, 30: 3442588.

[11] HUANG L, XUE J F, WANG Y, et al. WHGDroid: Effective Android malware detection based on weighted heterogeneous graph[J]. Journal of Information Security and Applications, 2023, 77: 103556.

[12] LI Y K, HU Y K, WANG Y Z, et al. RGDroid: Detecting Android malware with graph convolutional networks against structural attacks[C]//2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Piscataway: IEEE, 2023: 639-650.

[13] YANG H Y, WANG Y W, ZHANG L, et al. A novel Android malware detection method with API semantics extraction[J]. Computers & Security, 2024, 137: 103651.

[14] 郭燕慧, 王东, 王晓煊, 等. 一种面向图神经网络安卓恶意代码检测的通用解释定位方法[J]. 软件学报, 2024, 35(8): 1.
GUO Y H, WANG D, WANG X X, et al. A generic explaining & locating method for malware detection based on graph neural networks[J]. Journal of Software, 2024, 35(8): 1. (in Chinese)

[15] GU J T, ZHU H L, HAN Z W, et al. GSEDroid: GNN-based Android malware detection framework using lightweight semantic embedding[J]. Computers & Security, 2024, 140. DOI: 10.1016/j.cose.2024.103807.

[16] ZHAO W X, WU J T, MENG Z Y. AppPoet: Large language model based Android malware detection *via* multi-view prompt engineering[J]. Expert Systems with Applications, 2025, 262: 125546.

[17] ZHAN Z X, JI S, ZHEGN W Y, et al. DroidExaminer: An Android malware hybrid detection system based on ensemble learning[J]. Journal of Internet Technology, 2024, 25(1): 105-116.

[18] DESNOS A. Androguard: Reverse engineering and pentesting for Android applications[EB/OL]. (2013-01-01)[2024-12-30]. https://gitcode.com/gh_mirrors/an/androguard.

[19] SINGH S, CHATURVEDY K, MISHRA B. Multi-view learning for repackaged malware detection[C]//Proceedings of the 16th International Conference on Availability, Reliability and Security. New York: ACM, 2021: 1-9.

[20] LIU B Y, YUN D Y, GUO X, et al. Detecting sensor-based repackaged malware[C]//2020 IEEE International Conference on Big Data (Big Data). Piscataway: IEEE, 2020: 5759-5761.

[21] HE G F, XU B F, ZHANG L, et al. On-device detection of repackaged Android malware *via* traffic clustering[J]. Security and Communication Networks, 2020, 2020: 8630748.

[22] HU W J, TAO J, MA X B, et al. MIGDroid: Detecting APP-Repackaging Android malware *via* method invocation graph[C]//2014 23rd International Conference on Computer Communication and Networks (ICCCN). Piscataway: IEEE, 2014: 1-7.

[23] TIAN K, YAO D F, RYDER B G, et al. Detection of repackaged Android malware with code-heterogeneity features[J]. IEEE Transactions on Dependable and Secure Computing, 2020, 17(1): 64-77.

[24] SHI L M, MING J, FU J M, et al. VAHunt: Warding off new repackaged Android malware in app-virtualization's clothing[C]//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2020: 535-549.

[25] SALEM A, PAULUS F F, PRETSCHNER A. Repackman: A tool for automatic repackaging of Android apps[C]//Proceedings of the 1st International Workshop on Advances in Mobile App Analysis. New York: ACM, 2018: 25-28.

[26] HAGBERG A, CONWAY D. Networkx: Network analysis with python[EB/OL]. (2020-01-01) [2024-12-30]. https://networkx. github. Io.

[27] CAO Y Z, FRATANTONIO Y, BIANCHI A, et al. EdgeMiner: Automatically detecting implicit control flow transitions through the Android framework[C]//Proceedings 2015 Network and Distributed System Security Symposium. San Diego: NDSS, 2015.
[28] ALLIX K, BISSYANDÉ T F, KLEIN J, et al. AndroZoo: Collecting millions of Android apps for the research community[C]//2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). Piscataway: IEEE, 2016: 468-471.
[29] CAI H P, RYDER B. A longitudinal study of application structure and behaviors in Android[J]. IEEE Transactions on Software Engineering, 2021, 47(12): 2934-2955.
[30] GONG L Y, LI Z H, QIAN F, et al. Experiences of landing machine learning onto market-scale mobile malware detection[C]//Proceedings of the Fifteenth European Conference on Computer Systems. New York: ACM, 2020: 1-14.
[31] LI L, LI D Y, BISSYANDÉ T F, et al. Understanding Android app piggybacking: A systematic study of malicious code grafting[J]. IEEE Transactions on Information Forensics and Security, 2017, 12(6): 1269-1284.
[32] virustotal. VirusTotal[EB/OL]. (2024-11-30)[2024-12-30]. https://www.virustotal.com/gui/home/uploadÎ.
[33] SKYLOT. Jadx[EB/OL]. (2024-01-01)[2024-12-30]. https://github. com/skylot/jadx.
[34] WINSNIEWSKI R. Apktool: A tool for reverse engineering android apk files[EB/OL]. (2012-01-01)[2024-12-30]. https://ibotpeaches.github.io/Apktool.
[35] GAO C Y, HUANG G Z, LI H, et al. A comprehensive study of learning-based Android malware detectors under challenging environments[C]//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. New York: ACM, 2024: 1-13.
[36] SUN X Y, LI L, BISSYANDÉ T F, et al. Taming reflection: An essential step toward whole-program analysis of android apps[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2021, 30(3): 1-36.
[37] HUANG J J, XUE B, JIANG J S, et al. Scalably detecting third-party Android libraries with two-stage bloom filtering[J]. IEEE Transactions on Software Engineering, 2022, 49(4): 2272-2284.
[38] ZHAN X, FAN L L, CHEN S, et al. ATVHunter: Reliable version detection of third-party libraries for vulnerability identification in Android applications[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). Piscataway: IEEE, 2021: 1695-1707.
[39] ZHAN X, LIU T M, FAN L L, et al. Research on third-party libraries in Android apps: A taxonomy and systematic literature review[J]. IEEE Transactions on Software Engineering, 2022, 48(10): 4181-4213.

作者简介

杜瑞颖 女,1964年10月出生于河南省新乡市. 现为武汉大学国家网络安全学院教授、博士生导师. 主要研究方向为网络空间安全、大模型安全、系统安全、云安全.
E-mail: duraying@126.com

陈 晶 男,1981年3月出生于湖北省武汉市. 现为武汉大学国家网络安全学院副院长、教授、博士生导师. 主要研究方向为系统安全、移动安全、云安全.
E-mail: chenjing@whu.edu.cn

吴 聪 男,1995年10月出生于江西省丰城市. 现为香港大学博士后研究员. 主要研究方向为分布式智能系统安全.
E-mail: congwu@hku.hk

闫晰渝 女,2000年12月出生于河南省商丘市. 现为武汉大学国家网络安全学院硕士研究生. 主要研究方向为web3链上异常交易风险检测.
E-mail: xiyuyan@whu.edu.cn