计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.250200120

基于 MOBSF_rule 的安卓恶意软件检测方法

陈维国[1]　张高峰[2]　贾　晟[1]　徐本柱[2]　郑利平[1]
1 合肥工业大学计算机与信息学院　合肥 230601
2 合肥工业大学软件学院　合肥 230601
(2022111134@mail.hfut.edu.cn)

摘　要　在安卓应用安全研究领域的静态分析中,一种有效的方式是使用逆向工程工具对应用程序进行反编译,并从反编译后的代码文件中提取函数调用图(Function Call Graph,FCG)作为恶意软件识别的主要特征,特别是基于敏感 API 的 FCG 调用子图已经得到广泛验证。然而,现有的此类基于敏感 API 的研究工作大多依赖较早的敏感 API 集,没有随着系统 API 迭代继续更新。通过实验可以发现,使用传统的敏感 API 集从应用的函数调用图(FCG)中提取特征节点时,很多情况下无法获取到所需的特征节点。例如随着安卓系统的迭代更新,出现显著的 API 调整和更换,或者使用反射机制(Reflect)相关技术可以动态隐式调用系统 API。对此,文中根据最新的安卓应用综合研究框架,提出了一种基于 MOBSF_rule 提取 FCG 子图的安卓恶意软件检测方法。该方法首先从应用程序反编译的代码文件中生成函数调用图(FCG);然后利用 MOBSF_rule 规则集提取特征节点,生成包含这些特征节点的五点图、六点图和七点图,统计不同构型子图的出现频率;最后把频率矩阵输入机器学习方法中进行训练、推理。相比现有的敏感 API 集,所提方法有如下优势。1)MOBSF_rule 过滤规则集在提取特征节点方面表现出色,能够有效提取包括反射机制、组件交互、签名验证、网络通信和客户端/服务器(C/S)架构通信等关键 API 特征,对比传统敏感 API 集,在最新恶意软件数据集中特征提取有效率提升了 69.765%。2)MOBSF_rule 规则集在不同时间标签下提取特征节点的能力表现出色,具有较强的稳定性。它不仅能够适应安卓系统的持续更新,还能在不同版本之间保持高度一致的特征提取能力。2012—2022 年期间,相比传统敏感 API 集,MOBSF_rule 规则集的特征提取有效率的多年总体方差降低了 98.747%。3)采用了 Stacking 集成学习方法,对比随机森林集成学习方法和多层感知机方法,准确率提升了 4.32%。

关键词:安卓;函数调用图;敏感 API;反射机制;集成学习
中图分类号　TP181

MOBSF_rule Based Android Malware Detection Method

CHEN Weiguo[1],ZHANG Gaofeng[2],JIA Sheng[1],XU Benzhu[2] and ZHENG Liping[1]
1 School of Computer Science and Information Engineering,Hefei University of Technology,Hefei 230601,China
2 School of Software,Hefei University of Technology,Hefei 230601,China

Abstract　In the field of Android application security research,a highly effective method within static analysis involves utilizing reverse engineering tools to decompile the application and subsequently extracting the Function Call Graph(FCG) from the decompiled code files,which serves as a primary feature for malware identification. Notably,FCG subgraphs based on sensitive APIs have been widely validated. However,the majority of existing research efforts in this area rely on older sets of sensitive APIs and have not continued to update with the iteration of system APIs. Through experimentation,it has been discovered that when using traditional sensitive API sets to extract feature nodes from the application's FCG,many cases fail to obtain the desired feature nodes. For instance,with the iterative updates of the Android system,there are significant API adjustments and replacements,or dynamic implicit invocation of system APIs can be achieved using reflection mechanism(Reflect)-related technologies. In response to this,based on the latest comprehensive research framework for Android applications,this paper proposes an Android malware detection method that extracts FCG subgraphs using MOBSF_rule. The method first generates the FCG from the decompiled code files of the application. Then,it utilizes the MOBSF_rule set to extract feature nodes,generating five-node,six-node,and seven-node graphs containing these feature nodes,and statistically analyzing the occurrence frequency of different configuration subgraphs. Finally,the frequency matrix is input into the machine learning method for training and inference. Compared to existing sensitive API sets,the proposed method has the following advantages. 1)The MOBSF_rule filtering rule set demonstrates outstanding performance in extracting feature nodes,effectively extracting key API features including reflection mechanisms,component interactions,signature verification,network communication,and client/server(C/S) architecture communication. Compared to traditional sensitive API sets,the effective rate of feature extraction in the latest malware datasets has increased by 69.765%.

基金项目:国家重点研发计划项目(2022YFC3900800)
This work was supported by the National Key R&D Program of China(2022YFC3900800).
通信作者:张高峰(g.zhang@hfut.edu.cn)

2) The MOBSF_rule set shows excellent capability in extracting feature nodes across different time tags, exhibiting strong stability. It can not only adapt to the continuous updates of the Android system but also maintain a highly consistent feature extraction capability across different versions. Between 2012 and 2022, compared to traditional sensitive API sets, the overall variance in feature extraction effectiveness over multiple years decreases by 98.747%. 3) The method employs the Stacking ensemble learning approach, and compared to the random forest ensemble learning method and the multilayer perceptron method, the accuracy rate has increased by 4.32%.

Keywords Android, Function call graph, Sensitive API, Reflection mechanisms, Ensemble Learning

1 引言

随着移动设备的计算能力不断提升，安卓智能设备已深度融入日常生活的各个场景。在硬件层面，处理器性能、电池续航、相机成像等核心组件的持续升级，推动着安卓设备不断迭代更新。特别是移动端神经网络处理器（NPU）的集成，显著增强了设备的 AI 计算能力。以最新推出的 YOYO 智能体[1)]为例，其能够自动拆解任务步骤，通过语音指令完成复杂操作，展现了智能化应用的广阔前景。然而，根据工信部 2024 年公示的 10 批存在问题的 APP（SDK）通报[2)]，监管部门基于用户投诉反映的具体问题，对多个安卓应用市场的应用程序进行了抽样检测。图 1 给出了当前安卓应用在用户个人信息保护、权限管理和使用体验等方面存在的主要问题。

传统智能手机主要承担日程管理、理财投资、资讯获取、社交娱乐等功能，而现代智能应用已实现更深度的场景融合。例如，在获得用户授权后，相关软件可主动读取并展示预定的列车班次和座位信息，并在发车前提供智能提醒服务。然而，随着用户对智能设备依赖程度的加深，这给大量存储在设备中的个人隐私和敏感数据带来了新的安全隐患。恶意软件不仅影响用户体验，还可能窃取其他应用数据，在用户不知情时进行非法数据传输，严重威胁用户信息安全。

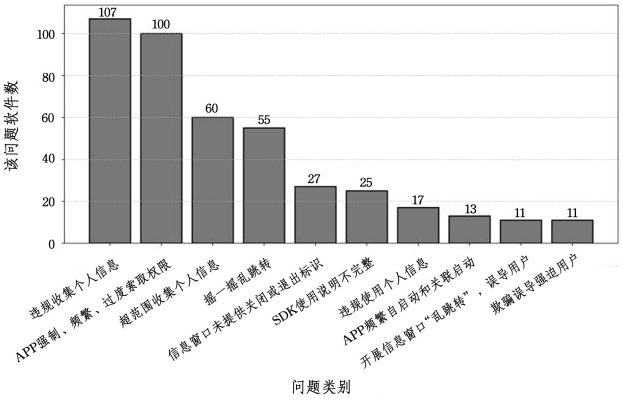


图 1 软件恶意行为直方图

Fig. 1 Histogram of software malicious behavior

为应对这一挑战，业界已采取多项技术措施。主流手机厂商通常都会提供官方的安全监管软件，用于拦截和防范第三方应用的潜在风险。然而，作为开源项目的 AOSP（Android Open Source Project）允许用户通过侧载（SideLoading）方式安装来自其他渠道的应用程序，这为恶意软件的检测带来了独特挑战。基于此，恶意软件检测主要存在以下 3 个典型应用场景。1）手机厂商自营的应用商店建立了严格的应用上架审核机制，在应用软件发布前实施多层级安全检测。该机制结合自动化机器检测与人工审核，通过静态代码分析、动态行为监测以及试安装运行等多维度验证，确保上架应用的安全性。2）在近年来开展的 APP 侵害用户权益专项整治行动中，工信部组织专业技术团队，依托第三方检测机构对 APP 及其 SDK 进行系统性技术检测。同时，监管部门重点加强对应用分发平台的主体责任落实情况监督检查，建立常态化监管机制。3）随着神经网络架构的优化和深度学习算法的突破，机器学习技术在恶意软件检测领域取得了显著进展。基于海量样本训练的检测模型能够有效识别新型恶意软件变种，显著提升了检测准确率和效率。

在当前的安卓恶意软件检测研究中，基于反编译技术的特征提取方法已成为主流技术路线。该方法通过对应用安装包（APK）进行反编译，提取已编译的代码文件（Classes. dex）和 AndroidManifest. xml 文件中的关键数据作为机器学习模型的输入特征。在这一研究方向上，PScout[1] 采用敏流感（Flow-sensitive）技术对权限到 API 的函数调用图（Function Call Graph）进行深入分析，并生成完整的 Map〈权限：API〉数据映射关系。SuSi[2] 借鉴了污点分析（Taint Analysis）和信息流分析（Information-Flow Analysis）中 SOURCES 和 SINKS 的概念，并结合安卓系统 API 特性进行了适应性重构。使用支持向量机（SVM）方法，将相关 API 分类为 SOURCES API 和 SINKS API。这两项研究成果为后续的安卓恶意软件检测工作提供了重要的理论基础和技术支撑。

FalDroid[3] 提出的 Frequent Subgraph 方法基于 SuSi 项目定义的 SOURCES API 和 SINKS API 分类体系。该方法通过 Apktool 反编译 APK 文件，获取 Smali 代码并构建函数调用图（FCG），其中 API 方法作为节点，调用关系作为有向边。在此基础上，提取包含 SOURCES API、SINKS API 及其连通节点的子图，形成 Frequent Subgraph。

Ou 等[4] 提出的 SFCG（Sensitive Function-Call Graph）生成方法整合了 PScout 的权限-API 映射数据和 SuSi 的 SOURCES API 与 SINKS API 分类，构建了候选 API 集。通过设置阈值筛选，构建所需的 SensitiveSet，基于该集合对 APK 生成的 FCG 进行图裁剪，生成子图 SFCG。

SeGDroid[5] 提出的基于敏感 API 的函数调用图（FCG）裁剪工作中，共使用了 21 986 个敏感 API，这些敏感 API 均源自 PScout 的数据组。

基于上述研究分析，后续基于敏感 API 的安卓恶意软件检测方法主要建立在 PScout 和 SuSi 两项基础性研究之上，本质上是对其理论框架的延伸与方法论的改良。尽管最新的安卓 API 级别已经更新至 35[8]，但 PScout 的结果只涵盖到

1) https://developer.honor.com/cn/doc/guides/101368

2) https://wap.miit.gov.cn/jgsj/xgj/gzdt/index.html

API_22 版本，SuSi 项目仅提供了 API_17 版本的结果。这导致基于这两项研究的规则集无法提取部分函数调用图（应用程序经过反编译后生成）的特征节点。

如图 2 所示，使用 API 级别为 22，21，19 和 18 的敏感 API 集，从 MalwareBazaar[19] 和 MalRadar[18] 数据集中提取特征节点时，出现了很多应用无法提取到特征节点的现象，这在 MalwareBazaar 开放平台上尤为明显，传统的敏感 API 集在提取最新上传的恶意软件时表现不佳，无法根据这些敏感 API 集获得函数调用图（FCG）的特征节点。这给基于敏感 API 的安卓恶意软件检测工作带来了实际挑战。

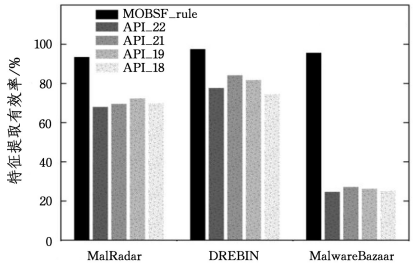


图 2　MalRadar，DREBIN，MalwareBazaar 3 个数据集上的特征提取有效率

Fig. 2　Feature extraction efficiency on the MalRadar，DREBIN，and MalwareBazaar datasets

本文提出特征提取有效率这一指标旨在探索一个新方案替代传统敏感 API 集，提取更有针对性、更有价值的特征信息。其计算式如式（1）所示：

$$\text{特征提取有效率}=\frac{\text{有特征节点的软件数}}{\text{软件总数}} \tag{1}$$

经过大量实验证明，本文方法可总结出以下两点优势：

1）使用 MOBSF[11] 提供的规则集，相比传统的敏感 API 集，显著降低了恶意软件和良性软件的丢失率，并且在不同时间标签下表现出较强的稳定性。

2）特征提取有效率提升表明编码后的特征矩阵规模更完整。在更大规模的特征矩阵下，使用原始方法的随机森林在 Accuracy（准确率）、F1 Score（F1 分数）和 AUC（曲线下面积）这 3 个指标上的二分类性能保持了相同的水平。通过采用集成学习中的 Stacking 架构，并选择随机森林和多层感知机（MLP）作为基学习器，与原始方法的单一随机森林相比，该方法的这 3 个指标均有所提升。

本文第 2 章介绍了关于安卓逆向分析工具和安卓恶意软件检测的相关工作；第 3 章介绍了实验中涉及到的所有技术和方法；第 4 章介绍了实验过程及结果分析；最后总结全文。

2　相关工作

本章介绍安卓逆向工程领域中静态分析与动态分析工具、机器学习技术和安卓恶意软件检测的相关工作。

2.1　安卓逆向工具

Androguard[9] 是一个强大的安卓应用静态分析工具库，基于 Python 编写并开源于 GitHub，最早由安全研究员 Anthony Desnos 开发。其核心功能包括：读取 APK 文件，提取 AndroidManifest. xml 和资源文件结构，解析 DEX 字节码，并支持将 DEX 文件中的类、方法、字段等信息直接映射到 Python 对象。通过解析 DEX 字节码，Androguard 可以帮助用户理解应用的代码逻辑和架构，分析数据流和控制流，追踪敏感 API 调用和权限使用情况。

Mobile Security Framework（MobSF）[11] 是一款功能丰富的开源工具，用于自动化分析移动应用的安全性，支持安卓和 iOS 平台。首先检查应用的 AndroidManifest. xml 文件（对于安卓应用）或应用的 Info. plist 文件（对于 IOS 应用），这两个文件分别包含了应用的配置、权限请求和组件声明等关键信息。MobSF 会分析这些文件中声明的权限，检测是否存在潜在的权限滥用或过度请求的情况。MobSF 的静态分析模块还会扫描应用中使用的敏感 API 调用，这些 API 通常与数据存储、网络传输、密码学操作和硬件访问相关。并且它可以检测应用中使用的第三方库，并检查这些库是否存在已知漏洞或更新滞后的问题。最后，MobSF 的静态分析结果会生成一份详尽的报告，包括权限使用情况、敏感 API 调用、代码中的潜在漏洞、使用的第三方库及其安全状态等信息。

在 APK 反编译上，本文使用 Androguard 工具生成 GML 格式的方法调用图（FCG），每个节点代表一个 API 方法。在提取敏感 API 的过程中，本文采用 MobSF 开源项目中提供的 android_apis. yaml 规则集，进行 FCG 的图裁剪。android_apis. yaml 规则集列举了 54 个应用行为和对应的 API 正则范式。

2.2　机器学习技术

随机森林（Random Forest，RF）[21] 是一种集成学习方法，主要用于分类、回归以及其他任务，其核心思想是通过集成多个决策树来提升预测性能和泛化能力。

使用 Bagging（Bootstrap Aggregating）方法，对样本进行随机有放回的抽样。

$$H(x)=\underset{c}{\operatorname{argmax}}\left(\sum_{m=1}^{M}G(h_m(x)=c)\right) \tag{2}$$

其中，$H(x)$ 是最终预测类别，$G(x)$ 表示当 $h_m(x)=c$ 时，$G(h_m(x)=c)=1$，否则 $G(h_m(x)=c)=0$。c 是类别标签，m 是对应的决策树，M 是决策树的总数。随机森林即使用投票法集成决策树的结果，得到模型最终的结果。

多层感知机（Multilayer Perceptron，MLP）[22] 是一种前馈人工神经网络，是深度学习领域中最基础的模型之一。MLP 由输入层、隐藏层和输出层组成，各层之间是完全连接的，能够通过层层传递信息来完成从输入到输出的映射。输入层接收原始特征，隐藏层通过权重和偏置对输入进行线性变换，并应用激活函数（如 ReLU，Sigmoid 或 Tanh）引入非线性，使得模型能够拟合复杂的非线性关系。

$$a^{(i-1)}=\varphi(z^{(i-1)}) \tag{3}$$

$\varphi(z^{(i-1)})$ 是输入层和隐藏层的激活函数，本文使用的激活函数为 ReLU。

$$z^{(i)}=W^{(i)}a^{(i-1)}+b^{(i)} \tag{4}$$

其中，$W^{(i)}$ 和 $b^{(i)}$ 分别是权重矩阵和偏置向量，该式代表 $i-1$ 层到 i 层之间的传递。

2.3　基于图（Digraph）的安卓恶意软件检测

图 $G=(V,E)$，在 Flow-sensitive 分析的工作中通过建模 CFG（Control Flow Graph）用于追踪程序中每个基本块（basic block）的执行路径，实现细粒度的动态跟踪。DroidMiner[12] 构建 CDG（节点为安卓组件）和 CBG（节点为 API 方法）挖掘并识别出恶意软件的多模态信息。AppContext[13] 着重于敏感 API 的触发路径和敏感 API 节点的所有前驱因素。早期的信息流分析工作已构建用于识别恶意软件行为的数据库。

然而，恶意软件开发者开始采用代码混淆技术，如插入无用代码、使用无意义的方法命名等，提高了恶意软件检测的难度，并且从大量应用样本中构建提取复杂的异构图特征也带来了高昂的计算开销。

Concept Drift 是机器学习领域中的常见问题：输入特征与标签之间的映射关系发生变化。该问题可以分为两种情况：1）存在某些作用因素没有发现，或者是需要捕捉更多有价值的映射关系，导致相同的特征输入在不同条件下出现不同的标签。例如，恶意软件开发者可能会采用新的技术手段来隐藏恶意行为，或者针对不同的安卓系统版本进行优化。即使在同样的输入特征下（例如应用程序的权限请求、API 调用序列），恶意软件的检测结果可能会在不同条件下发生变化，这种情况下可能出现 Concept Drift。2）模型出现老化的情况。随着时间的推移，安卓系统不断更新，新的恶意软件家族不断出现，原有的检测模型可能无法适应新的数据集，导致检测性能下降。这两种情况的区别在于：前者倾向于现有的，后者倾向于未来的。

TESSERACT[14] 提出了一个新的指标 AUT，用于衡量不同分类器面对时间衰减问题的稳定性。APIGraph[15] 构建了一种知识图谱，采用包、类、方法和权限作为节点，形成异构图结构。使用 K-Means 算法对 API 进行聚类，聚类结果被输入到现有的分类器（如 MAMADROID，DROIDEvolver，DREBIN，DREBIN-DL）中，从而提升模型的性能，应对模型老化问题。

E-FCG[16] 从函数调用图（FCG）中生成多个调用序列，并采用 CBOW（Continuous Bag of Words）词向量生成方法，将具有相似属性的节点编码到距离较近的空间中。Lo 等[17] 设计了 6 层相同的图神经网络，采用 Jumping Knowledge 技术防止过平滑问题。在节点特征编码时，使用了 PageRank、入度/出度（In/out degree）和节点介数中心性（Node betweenness centrality）3 种算法来初始化节点特征向量。该研究在 Malnet-Tiny（Windows 应用数据集）和 DREBIN[6]（安卓应用数据集）上均取得了优异的性能指标。

3 方法和技术

本章首先介绍了 MOBSF_rule 规则集相较于传统敏感 API 集的优势，并通过实验结果展示了基于该规则集提取特征节点的有效性和实用性。MOBSF_rule 规则集通过更精准的筛选机制，能够更高效地识别出与恶意软件行为密切相关的敏感 API。然后详细展示了该方法的完整流程，包括基于特征节点生成的五点图、六点图和七点图，以及最终特征矩阵的构建过程。

3.1 MOBSF_rule 规则集

本小节首先展示 MOBSF_rule 规则集相较于 PScout 提供的 API_22，API_21，API_19，API_18 这 4 个 API 级别的 MAP〈权限，API〉集中筛选出的敏感 API 集，在特征提取有效率指标上的显著优势。然后根据从 DREBIN[6]，MalRadar[18] 和 MalwareBazaar[22] 数据集中得到的高频率出现的特征节点 API，详细介绍 MOBSF_rule 规则集提取的特征节点类型。

根据官方开发者文档，安卓权限机制设有多种保护级别，大致可分为普通（normal）、危险（dangerous）、签名（signature）等类别。在 Ou 等的研究中，将保护级别为普通的 API 定义为非敏感 API，而将其他保护级别相关的 API 定义为敏感 API。本文采用这种方式，从 PScout 提供的 API_22，API_21，API_19，API_18 这 4 个 API 级别的 MAP〈权限，API〉集中筛选出敏感 API 集，并与 MOBSF（Mobile Security Framework）提供的规则集（以下简称 MOBSF_rule）进行对比。

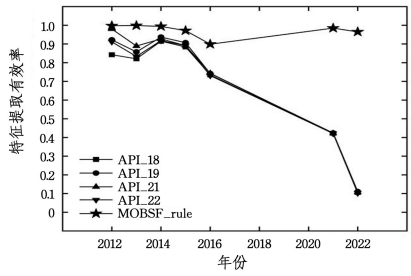


图 3 2012－2022 年恶意特征提取有效折线图

Fig. 3 Line chart of malware feature extraction efficiency from 2012 to 2022

实验结果显示，通过对图 3－图 6 进行对比分析可知，在恶意软件检测领域，MOBSF_rule 相较于传统敏感 API 集展现出了极为显著的性能优势，具体体现在以下两个方面。1）特征提取稳定性：在处理 2021－2022 年恶意软件数据集时，MOBSF_rule 的特征提取有效率在所有对比方案中最高，说明该规则集在面对新型恶意软件样本时，能够保持良好的性能，充分体现出其在特征提取方面具有很强的鲁棒性。2）函数调用图（FCG）特征节点数：在函数调用图（FCG）的特征节点提取上，MOBSF_rule 的表现同样出色。将 MOBSF_rule 提取的每个应用程序反编译后的函数调用图特征节点数，与传统敏感 API 集进行对比，结果清晰地显示出 MOBSF_rule 的优势。从图 5 和图 6 的箱线图能够直观地看到，MOBSF_rule 所对应数据分布的均值和上限，均远远高于传统敏感 API 集的数据分布。这表明 MOBSF_rule 能够提取到更为丰富、全面的特征节点，有助于更精准地识别恶意软件。

随着安卓平台应用的持续演进，其功能日益多样化，导致应用代码体量逐渐臃肿。特别是如图 4 所示，良性软件的特征提取有效率随着年份呈上升趋势，这在一定程度上反映了安卓应用对系统功能调用的需求在不断增加。在 Ou 等的研究框架内，通过计算传统敏感 API 集在数据集中的出现频率，不仅有效降低了时间复杂度，还成功挖掘出了与恶意行为更为相关（即出现频率更高）的敏感 API。然而，在基于敏感 API 的安卓恶意软件检测工作中，仍需进一步提升针对性，以更精准地识别恶意软件的行为特征。

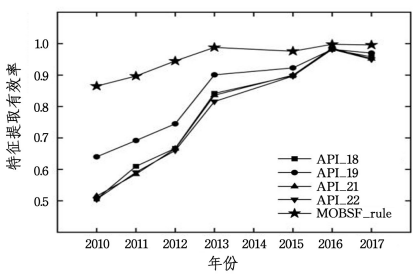


图 4 2010－2017 年良性软件特征提取有效率折线图

Fig. 4 Line chart of benign feature extraction efficiency from 2010 to 2017

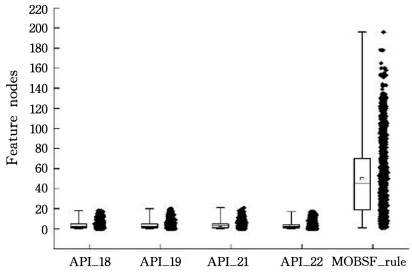


图 5　DREBIN 特征节点数分布箱线图

Fig. 5　Boxplot of DREBIN feature node distribution

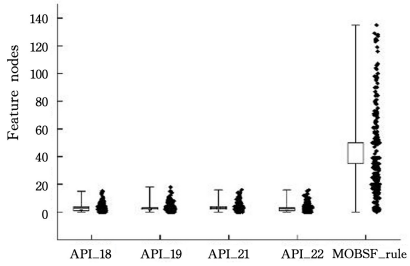


图 6　VirusShare 特征节点数分布箱线图

Fig. 6　Boxplot of VirusShare feature node distribution

通过对大规模 APK 样本进行静态逆向工程处理和特征节点提取，得出了以下高频 API 节点的实验结果。表 1 中，Method 类和 Field 类的分布频率较高，同样是规则集 MOBSF_rule 关注的 API 类型之一。这两类 API 均与反射机制密切相关，是应用程序在 JVM 或 ART 环境中动态加载类的主要实现方式。反射机制 API 的调用并不依赖于安卓权限的设置，这与编译好的静态代码有所不同。虽然动态加载机制提供了灵活性，但同时也增加了应用软件的运行开销。

表 1　反射机制相关 API

Table 1　Reflection mechanism related API

完整类名－＞方法名/方法名
java. lang. reflect. Method－＞invoke/getName/getParameterTypes
java. lang. reflect. Field－＞set/get/getType

但是，在使用 PScout 中与权限相关的 API 时，难以精准地识别和提取与反射机制相关的系统 API。

图 7 是从一个函数调用图(FCG)中提取的子图，其中 invoke，setAccessible，getDedaredMethod 方法所在的节点均为系统 API 节点，子图提取过程中保留了特征节点在完整 FCG 图中的出入度信息。上述这 3 个节点均为出度为零的节点。

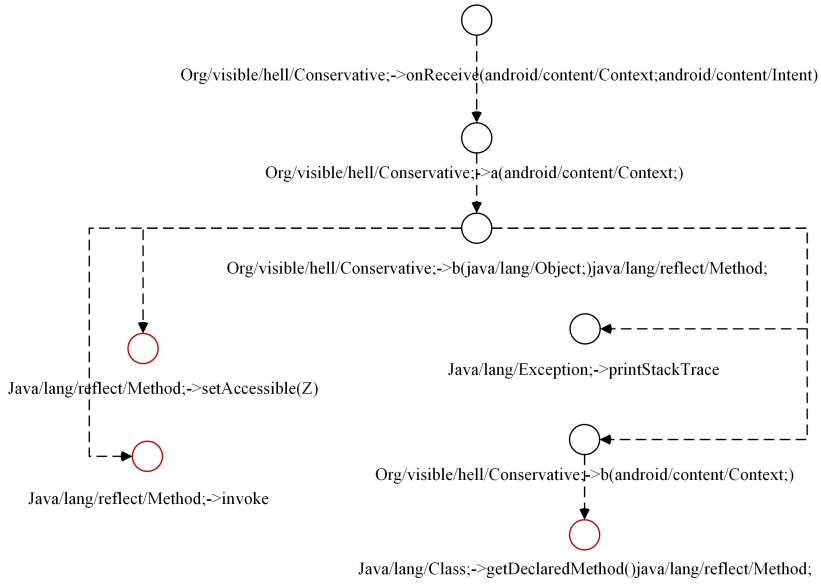


图 7　反射机制相关的 FCG 图

Fig. 7　FCG diagram related to reflection mechanism

表 2 列出了与 android. content 相关的 API，这些 API 的分布频率也较高。android. content. Context 类是安卓系统中的一个核心类，用于管理资源访问，启动 Activity，Service，BBroadcastReceiver 等组件，使用 Context. getSystemService()获取系统服务。每个安卓应用组件都继承自 Context 类。

表 2　组件相关的 API

Table 2　Component-related API

完整类名－＞方法名/方法名
android/content/Context－＞getSystemService
android/content/Context－＞getSharedPreferences
android/content/pm/PackageManager－＞queryIntentActivities
android/content/ContentProvider－＞⟨init⟩/attachInfo
android/content/Intent－＞getIntExtras/resolveActivity
android/content/IntentFilter－＞addAction

安卓应用组件(Component)是安卓应用开发的核心构建模块，它们是构成安卓应用程序的基本单元，负责实现应用程序的各种功能和交互逻辑。每个组件都有其特定的角色和生命周期，开发者通过合理组织和协同这些组件来构建完整的应用程序。

如图 8 所示，Component 相关的部分函数调用图(FCG)呈现了组件之间的交互情况。图中的 setComponentEnabledSetting() 方法用于设置组件的状态，它接受 PackageManager 的成员常量作为参数，能够将组件的状态设置为缺省(DEFAULT)、取消(DISABLED)或激活(ENABLED)等。

除了上述提到的反射机制和组件相关的 API 外，还有一些重要的工具类 API。例如：MessageDigest 用于签名验证；URL 用于网络通信，以及 IBinder 用于客户端/服务器(C/S)架构通信。这些 API 至今未被弃用，且沿用时间较长，因此它们的特征分布对识别恶意软件和良性软件具有重要价值。

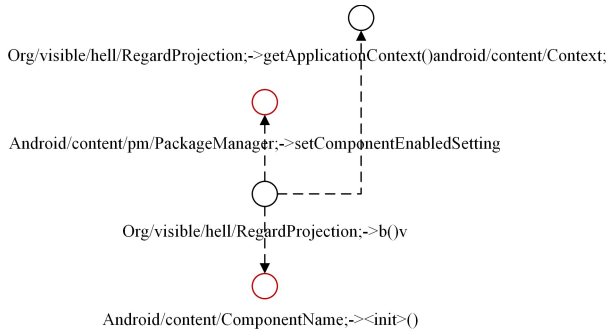


图 8 Component 相关的 FCG

Fig. 8 FCG diagram related to Component

3.2 API 节点调用模式挖掘

本小节致力于从提取的特征节点中挖掘出在恶意软件集中普遍存在且频繁出现的 API 调用模式，把这些调用模式在良性软件和恶意软件中的数据分布差异作为监督分类任务的关键输入。

为了提高挖掘效率，采用了一种创新的方法：通过遍历特征节点的邻节点，生成包含特征节点的五点图、六点图和七点图。这种方法不仅能够捕捉到特征节点与周围节点之间的复杂关系，还能显著降低特征处理过程中的时间复杂度。

图 9、图 10 为包含敏感 API 节点的 5 点图，都是从图 7 中得到的特征子图。在反射机制中，使用 invoke()方法执行目标方法，使用 getDeclaredMethod()方法获取目标方法实例，并通过 setAccessible()方法设置访问权限，以便能够访问和修改私有(private)或缺省(protected)的字段或方法。这些系统 API 在实现具体功能时通常会同时出现。在图 11 中，用 printStackTrace()节点替换了图 10 中的 onReceive()节点，从而得到了两个不同构型的五点图。基于这些不同构型的特征子图的出现频率，来挖掘良性软件和恶意软件在数据分布上的差异，通过这种方法，使用机器学习技术识别和区分良性软件与恶意软件。

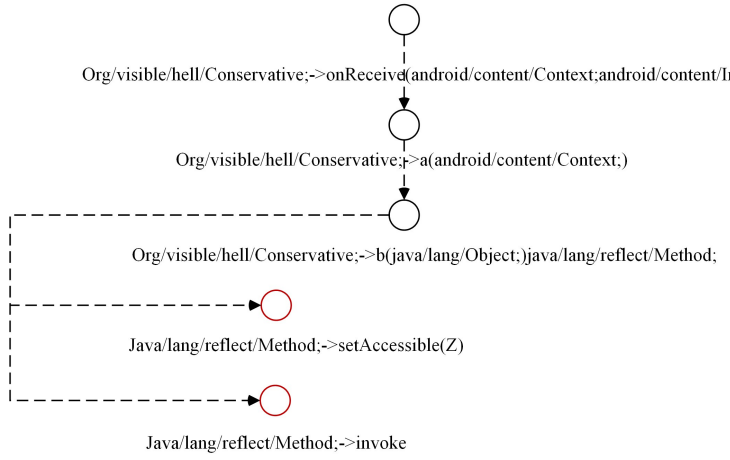


图 9 特征子图构型一

Fig. 9 Feature subgraph configuration 1

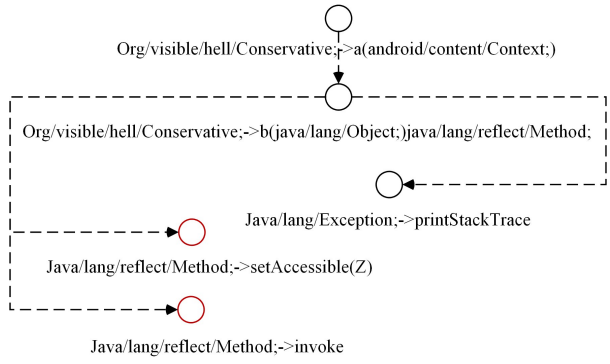


图 10 特征子图构型二

Fig. 10 Feature subgraph configuration 2

为了提取包含特征节点的五点图、六点图和七点图，本文方法采用深度优先搜索(DFS)策略，从规则集 MOBSF_rule 提取特征节点开始遍历其邻节点。如算法 1 中获取特征节点(Node)的后继节点(successors)和前驱节点(predecessors)。node_num 是子图的节点数上限，即五点图、六点图和七点图，Subgraph_node_set 是新生成子图的节点集，Judge_iso 方法用于判断生成的子图构型是否已存在，新构型就会保存为 GML 文件。

算法 1 对特征节点的前驱进行 DFS 搜索

输入：Node，node_num，Subgraph_node_set

输出：包含特征节点的特征子图 GML_Graph

1. begin{algorithm}[! ht]
2. SetAlgoLined
3. KwIn{敏感 API 节点 v，函数调用图 G，子图目标节点数 k，当前子图节点集合 S，邻接搜索上限 1，APK 对象 texttt{apk}}
4. KwOut{满足大小为 k 的子图(如有)被识别并判断是否为新模式}
5. If {|S| > k}{
6. Returntcp * {超过节点数限制，停止搜索}
7. }
8. If {|S| = k}{
   构建诱导子图 G′ leftarrow G[S]；
   调用子图模式判定过程：texttt{judge_is_Iso}(G′, texttt{apk})；
   Return
   }
9. 获取当前节点 v 的所有邻接节点列表 N_v(包括前驱和后继)；
10. 初始化已遍历邻居计数器 c leftarrow 0；
11. ForEach{u in N_v}{
12. If{u in S}{

```
13.        textbf{continue}tcp*{避免重复访问}
14.      }
15.      将 u 加入 S;
16.      递归调用 texttt{dfs}(u,G,k,S,l,texttt{apk});
17.      将 u 从 S 移除 tcp*{回溯}
18.      c leftarrow c + 1;
19.      If{c > l}{
20.        textbf{break}tcp*{达到邻接遍历上限}
21.      }
22.    }
23.    caption{深度优先搜索敏感 API 调用子图 texttt{dfs}(v,G,k,
       S,l,texttt{apk})}
24.    label{algo:dfs_sensitive_subgraph}
25. end{algorithm}
```

3.3 构建特征矩阵

本小节给出了构建特征矩阵的完整流程，步骤如下。1)提取特征节点与生成子图(图裁剪)：从函数调用图中提取特征节点，遍历其邻节点，生成包含特征节点的五点图、六点图和七点图。统计每种子图的出现频率，这些频率将作为特征矩阵的基础数据。2)构建特征维度：收集所有出现的子图构型，将这些构型作为特征矩阵的维度。子图的构型数量决定了特征矩阵的维度大小，每个构型对应矩阵中的一个特征维度。3)嵌入特征向量：针对每个应用程序的函数调用图，将其编码为特征向量。通过将子图频率映射到对应的特征维度，完成对函数调用图的向量化表示，从而为后续的机器学习分析提供结构化的特征输入。

图 11 给出了特征矩阵的实际含义。为了全面阐释特征矩阵构建的完整流程，借助数据流图，采用面向过程的编程建模方法，将生成最终特征矩阵的每一步骤详细呈现出来。

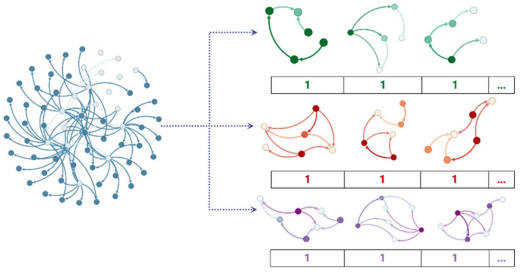


图 11 特征向量构建图

Fig. 11 Diagram of feature vector construction

在特征子图(五点图、六点图、七点图)提取完成后，整个工作流程可以大致分为两个主要部分：1)收集特征子图的结构信息，用于构建特征矩阵的维度；2)将每个 APK 应用的特征表示为向量，完成向量嵌入。

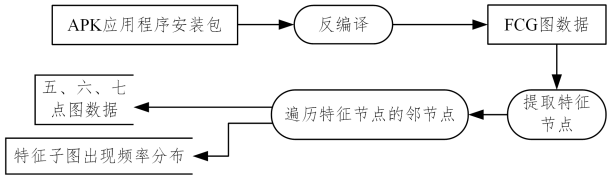


图 12 特征子图生成和频率统计的数据流图

Fig. 12 Data flow graph of feature subgraph generation and frequency statistics

在图 13 中，使用的图对比工具是 NetworkX，基于 VF2 算法实现图的同构性判断。通过对比两个图中节点的邻接属性(出入度)，判断两个图是否同构。

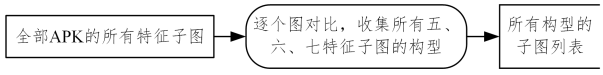


图 13 收集特征子图构型构建特征矩阵维度数据流图

Fig. 13 Data flow graph for collecting feature subgraph patterns to construct feature matrix dimensions

如图 14 所示，完成每个 APK 样本的特征向量嵌入后，就完成了数据的预处理工作，需要把数据输入到机器学习方法中，开展后续的监督分类任务。

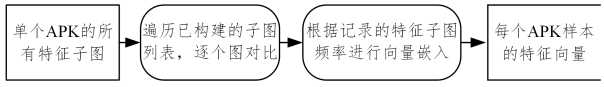


图 14 特征矩阵向量嵌入数据流图

Fig. 14 Data flow graph for feature matrix vector embedding

4 实验过程与结果评估

本章通过对比实验验证了本文方法的有效性。首先介绍实验所用的数据集，随后展示该方法在二分类任务性能上的优势。

4.1 数据集

DREBIN[6]数据集是一个专门针对安卓恶意软件检测的公开数据集，由 Arp 等于 2014 年首次提出。该数据集以静态分析为核心，旨在通过轻量化特征提取来快速、有效地检测恶意应用程序。DREBIN 数据集包含 123453 个安卓应用样本，其中 5560 个为恶意软件。这些恶意软件样本主要来源于 Google Play 商店和非官方市场，涵盖了从简单到复杂的多种恶意行为，包括广告软件、间谍软件、勒索软件和 SMS 欺诈工具等。

VirusShare[7]数据集是一个专门面向恶意软件研究的共享平台，由安全研究员 Murdock 于 2010 年建立。该数据集的创建目的是解决恶意软件样本难以获取的问题，同时为学术研究、恶意软件检测和防御策略设计提供必要的实验支持。作为当前最大规模的恶意软件数据库之一，VirusShare 数据集涵盖了超过 50 TB 的恶意软件样本，并持续更新以反映新出现的威胁。其内容包括多种类型的恶意软件，如病毒、蠕虫、木马、勒索软件、广告软件、间谍软件以及针对特定平台(如安卓和 iOS)的移动恶意软件样本。此外，它还包含针对嵌入式设备和物联网(IoT)的恶意软件，充分体现了当前威胁的多样性和复杂性。

MalRadar[18]是一个专注于安卓恶意软件的高质量数据集，旨在通过可靠的方法为安全研究提供最新、准确的恶意软件样本。MalRadar 数据集包含 4534 个独特的安卓恶意软件样本，包括 APK 文件及其元数据，这些样本涵盖了从 2014—2021 年 4 月期间发布的恶意软件。每个样本都经过安全专家的手动验证，并附有详细的行为分析报告，确保数据的准确性和可靠性。MalRadar 的构建过程不仅保证了数据的权威性，还通过行为分析与 IoC 信息为恶意软件研究提供了全面的支持。作为一个不断增长和更新的数据集，MalRadar 为恶

意软件检测、家族分类、动态行为分析和威胁情报研究提供了宝贵的资源。

AndroZoo[21]是一个规模庞大且不断扩展的安卓应用程序数据集，被广泛应用于移动应用安全和恶意软件分析研究领域。该数据集由多个来源收集，包括官方的 Google Play 应用市场，目的是为学术研究和工业实践提供可靠的数据支持。目前，AndroZoo 已包含超过 24 929 691 个独特的 APK 文件，涵盖了不同时间段和应用类别的广泛样本。每个 APK 文件都经过多个杀毒引擎的扫描和分析，以识别其是否被检测为恶意软件，从而为研究安卓恶意软件检测提供了宝贵的信息。AndroZoo 的主要特点在于其样本的多样性和数据的及时更新。研究人员可以通过该数据集获取到真实世界中的应用样本和恶意软件，这对构建和评估恶意软件检测模型、动态行为分析、应用分类、隐私泄露检测等任务具有重要意义。此外，AndroZoo 数据集还附带详细的元数据，例如应用的哈希值、DEX 编译时间、DEX 尺寸、APK 包名、APK 的市场来源。

MalwareBazaar[19]是一个专门为 IT 安全研究人员设计的恶意软件样本共享平台。它旨在解决研究人员在获取恶意软件样本时面临的诸多问题，如需要在多个平台上注册、下载限制以及付费订阅等。通过 MalwareBazaar，研究人员可以轻松地共享和获取恶意软件样本，从而更高效地进行威胁情报的分析和研究工作。该平台的创建，不仅简化了恶意软件样本的获取流程，还降低了研究成本，为 IT 安全研究人员提供了一个便捷、高效的资源共享社区。

本文使用的恶意软件数据集包括 DREBIN，VirusShare 和 MalRadar，而良性软件样本则来自 AndroZoo。为了确保数据集的平衡性，在实验中从 DREBIN，MalRadar 和 VirusShare中随机抽取了 4000 个 APK 样本，并从 AndroZoo 历年数据中随机抽取了 4000 个 APK 样本。这种随机抽样方法确保了恶意软件和良性软件样本数量的均衡，为后续的实验分析提供了可靠的数据基础。

4.2 对比实验

本研究的实验工作运行在 Ubuntu 操作系统的服务器上，整个实验过程包括安卓应用的反编译、函数调用图(FCG)的生成与过滤、子图提取和使用数据训练，均在 CPU 环境下完成。实验所用的工具主要包括 Python 生态中的 Androguard 和 Scikit-learn，它们分别负责应用的静态分析与模型训练阶段。

4.2.1 实验指标介绍

在实验中，对 3 种机器学习模型的分类性能进行了详细的对比分析，分别为随机森林(Random Forest)、MLP 神经网络以及 Stacking 模型。在对比过程中，主要关注以下 3 个关键性能指标：准确率(Accuracy)、F1 分数(F1 Score)以及 AUC。通过综合对比，可以全面评估每种模型在处理恶意软件检测任务时的表现。

$$Accuracy=\frac{TP+TN}{TP+TN+FP+FN} \tag{5}$$

$$Precision=\frac{TP}{TP+FP} \tag{6}$$

$$Recall(TPR)=\frac{TP}{TP+FN} \tag{7}$$

$$F1\text{-}Score=\frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} \tag{8}$$

式(5)是准确率的计算式，用于衡量模型能正确分类的能力。*Precision* 是被正确预测的正类样本占所有预测为正类样本的比例。*Recall* 是被正确预测的正类样本占所有实际正类样本的比例。*F*1 *Score* 是 *Precision* 和 *Recall* 的调和平均数，两个指标相差较大时，*F*1 会显著降低。

$$FPR=\frac{FP}{FP+TN} \tag{9}$$

$$AUC=\int_0^1 TPR(FPR)\mathrm{d}(FPR) \tag{10}$$

AUC(Area Under Curve)是 ROC 曲线(Receiver Operating Characteristic Curve)下的面积，其横坐标为假正率(False Positive Rate，FPR)，纵坐标为真正率(True Positive Rate，TPR)。AUC 通过计算 ROC 曲线与横轴之间的面积，全面反映了模型在不同分类阈值下 FPR 和 TPR 的动态变化关系，从而描述了模型对正负样本的区分能力。

$$0\leqslant TPR\leqslant 1 \tag{11}$$

$$\int_0^1 0\mathrm{d}(FPR)\leqslant\int_0^1 TPR(FPR)\mathrm{d}(FPR)\leqslant\int_0^1 1\mathrm{d}(FPR) \tag{12}$$

根据式(15)和式(16)的推导，$0\leqslant AUC\leqslant 1$，当真正率(True Positive Rate，TPR) 达到最大值 1 时，FN 的数量为 0，这表明模型能够完全正确地识别所有正类样本，此时模型的分类效果达到最优。因此，当 AUC 的值越接近 1 时，说明模型在不同分类阈值下对正负样本的区分能力越强，能够更稳定地保持较高的 TPR 和较低的假正率(False Positive Rate，FPR)。

4.2.2 特征维度的构建

根据 3.2 节中的步骤，首先对 DREBIN 和 AndroZoo 数据集中的部分应用程序进行反编译处理，成功提取了函数调用图(FCG)数据。在构建特征矩阵和嵌入特征向量的过程中，不同规则集收集的特征矩阵维度如表 3 所列。

表 3 不同规则集收集的特征矩阵维度

Table 3 Dimensions of feature matrices collected by different rule sets

过滤规则集	提取的特征维度
MOBSF_rule(Ours)	**56 288**
API_Level_22	10 969
API_Level_21	13 550
API_Level_19	15 014
API_Level_18	11 536

基于 MOBSF_rule 规则集收集的特征维度达到 56 288，反映了 MOBSF_rule 在安卓恶意软件检测任务中强大的信息提取能力。

4.2.3 实验对比

本小节首先对实验的具体步骤进行了整体介绍，详细阐述了实验设计和数据集的使用情况。接着，通过实验验证了 MOBSF_rule 规则集在分类任务中的可靠性，在此基础上进一步优化了随机森林方法，提出了基于异源集成的 Stacking 方法。实验结果表明，Stacking 方法在性能指标上优于传统的随机森林方法，能够进一步提升分类器的准确性和鲁棒性。

为确保本文方法具备良好的泛化能力，本文采用了 K 折交叉验证（K-Fold Cross-Validation）。在这种方法中，数据集被分成 K 个大小相近的子集（称为“折”）。每次选择一个子集作为验证集（如图 5 中灰色网格），剩余的 $K-1$ 个子集作为训练集，重复 K 次，使得每个子集都能被用作一次验证集。最终，将所有 K 次的验证结果取平均值，得到模型的整体性能评估。

在随机森林、MLP 神经网络以及 Stacking 集成学习的训练过程中，统一采用了 K 折交叉验证方法，以确保评估结果的可靠性和稳定性。这种评估方法不仅能够充分利用有限的数据集，还能有效降低模型评估结果的方差，为不同模型的性能比较提供了可靠的依据。

在模型性能评估对比实验中，首先选择了随机森林这一机器学习方法。对 MOBSF_rule，API_Level_22，API_Level_21，API_Level_19 这 4 种规则集进行了详细的对比分析。这一选择是基于 S3Feature[6] 的研究成果，该研究同样采用了随机森林模型，并验证了其在安卓恶意软件检测任务中的有效性和可行性。

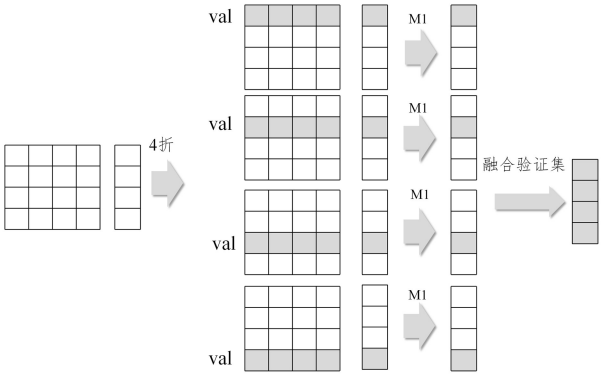


图 15　交叉验证过程图

Fig. 15　Cross-validation process diagram

考虑到近年来安卓应用体积日益臃肿的趋势，为避免超大函数调用图（FCG）带来的高额时间成本，在实验中设定了关于软件尺寸的样本筛选标准：仅对尺寸大小在 15MB 以内的函数调用图进行特征提取。这一限制条件的确立主要基于以下两点考虑：首先，超大函数调用图（FCG）会显著增加特征提取和模型训练的计算开销；其次，通过限制样本规模，控制变量，保持数据的代表性和实验结果的可靠性。

具体按照以下步骤展开：

1）数据预处理与样本筛选。对 DREBIN，VirusShare，MalRadar，AndroZoo 这 4 个公开数据集进行预处理，过滤掉 FCG 大小超过 5MB 的样本。这一步骤不仅有效控制了数据规模，还确保了后续特征提取和模型训练的高效性。

2）基于规则集提取特征节点。分别采用 MOBSF_rule，API_Level_22，API_Level_21 和 API_Level_19 这 4 种规则集，从每个应用的函数调用图（FCG）中提取特征节点集。

3）特征向量嵌入。针对每个特征节点，对其邻域进行遍历，并基于表 3 中构建的特征矩阵，统计不同构型的子图在特征矩阵中的出现频率分布。

根据表 4 所展示的关键指标，严格在同一数据集下对这 4 个规则集的性能进行了对比分析。然而，这 3 个指标仅能说明规则集提取的特征能够可靠地进行恶意软件检测的二分类任务，在当前安卓系统和安卓应用生态中存在显著局限性。在以往的研究中，未能提取到特征节点的应用软件通常被忽视，甚至被简单地视为无效或“垃圾样本”，从而影响了对模型性能的全面评估。实际上，在数据挖掘领域，这种现象属于数据的非随机丢失（Missing Not at Random，MNAR），即数据的丢失并非随机发生，而是与数据本身的特征或生成机制有关。使用特征提取有效率来衡量数据的非随机丢失程度：特征提取有效率越高，得到样本向量嵌入的大小越完整，数据的非随机丢失程度越低；特征提取有效率越低，存在很多样本向量丢失的情况，导致数据的非随机丢失程度越高。

在基于 MOBSF_rule 规则集提取的特征向量基础上，本文采用了随机森林（Random Forest）、多层感知机（Multilayer Perceptron，MLP）神经网络 和 Stacking 三组机器学习方法进行性能对比，旨在通过不同模型的优势互补，进一步提升分类任务的性能指标。

从集成学习的分类角度来看，集成学习方法主要分为同源集成（Homogeneous Ensembles）和异源集成（Heterogeneous Ensembles）。随机森林是一种典型的同源集成方法，其基学习器均为决策树，通过 Bagging 方法（有放回地随机抽样）生成多棵决策树并集成其预测结果，从而提升模型的稳定性和泛化能力。而 Stacking 则是一种异源集成方法，其基学习器可以是不同类型的模型（如决策树、支持向量机、神经网络等），通过元学习器（Meta-Learner）对基学习器的预测结果进行二次学习，从而充分利用不同模型的优势，进一步提升分类性能。通过对比这 3 组模型在准确率（Accuracy）、F1 分数（F1 Score）和 AUC 等关键指标上的表现，能够全面评估不同模型在基于 MOBSF_rule 规则集提取的特征向量上的分类性能，从而为安卓恶意软件检测任务选择最优的模型架构提供实验依据。

Stacking（堆叠）[24] 是一种经典的异源集成学习技术，其主要目标是通过结合多个基础模型（一级模型）的预测结果，训练一个元模型（二级模型），从而提升整体的预测性能。Stacking 的核心思想是通过不同模型的组合来充分利用它们的优势，弥补单一模型的不足，从而提高泛化能力。与其他集成学习方法（如 Bagging 和 Boosting）相比，Stacking 更加灵活，因为它允许基础模型的类型和数量不受限制，并通过元模型来学习如何最佳地组合这些基础模型的输出。

在 Stacking 中，基础模型通常用于对数据的不同特性进行独立建模，它们的输出被称为“一级输出”。这些一级输出作为新的特征被输入到元模型中，而元模型的任务是学习基础模型之间的关系，并进行最终的预测。基础模型可以是任意类型的机器学习算法，例如决策树、支持向量机、神经网络等，元模型也可以是任意类型的学习算法，通常选择逻辑回归、线性回归或简单的神经网络。

本文使用的基学习器 M_1 为随机森林，M_2 为 MLP 神经网络，元学习器为逻辑回归。大致的过程如式（13）－式（16）所示：

$$P_m = h_m(X), m = 1, 2 \tag{13}$$

$$P = [P_1, P_2] \tag{14}$$

$$M(P) = LogisticRegression(P; \theta) \tag{15}$$

$$\hat{y} = M(P) \tag{16}$$

式(13)中，$h_m(X)$是基学习器，P_1和P_1分别是两个基学习器的预测结果。式(14)中，合并预测结果P_1和P_2为元学习器的输入矩阵。式(15)中，θ为逻辑回归函数的参数。式(16)中，$\hat{y}$为模型的最终预测结果。

表5中，在同样的特征输入情况下，本文方法为性能表现优于多层感知机(MLP)方法和随机森林(RF)方法。准确率较其他模型的平均值提升了4.32%，F1 Score提升了3.7%，AUC提升了1.71%。

表4 不同规则集在随机森林模型上的性能对比

Table 4 Performance comparison of random forest models with different rule sets

规则集	*Accuracy*	*F1 Score*	*AUC*
MOBSF_rule(Ours)	**0.8891**	**0.8953**	**0.9691**
API_Level_22	0.8833	0.8880	0.9508
API_Level_21	0.8901	0.8932	0.9556
API_Level_19	0.8661	0.8731	0.9484

表5 基于MOBSF_rule规则集的机器学习方法的性能对比

Table 5 Performance comparison of machine learning methods based on the MOBSF_rule rule set

对比方法	*Accuracy*	*F1 Score*	*AUC*
Ours	**0.9245**	**0.9246**	**0.9764**
MLP	0.8833	0.8880	0.9508
随机森林	0.8891	0.8953	0.9691

结束语 本文的主要贡献如下：

1)由于传统的敏感API集在最新的恶意软件数据集中特征提取的效率太低，因此采用MOBSF_rule规则集开展基于敏感API的安卓恶意软件检测工作，能够有效提取关键API特征节点，包括反射机制、组件交互、签名验证、网络通信以及客户端/服务器(C/S)架构通信等。这些特征的提取为后续的分类任务提供了丰富的信息，显著提高了特征提取的效率，使得该方法提取特征的能力更有效、更稳定。

2)使用Stacking集成学习方法，通过对基学习器的预测结果的二次训练，对比随机森林集成学习方法、多层感知机(MLP)方法，本文方法的预测准确率提升了4.32%，F1 Score提升了3.7%。

由于该方法在处理超大函数调用图(FCG)时具有较大的时间复杂度，特别是在生成子图和获取子图出现频率的特征工程过程中需要耗费大量时间。未来，计划优化特征的数据结构，并引入图神经网络(GNN)技术，以显著提升安卓恶意软件检测在二分类或多分类任务中的效率与效果。这一改进将特别针对当前臃肿的安卓应用生态及特定应用场景，旨在实现更高效、更精准的恶意软件检测能力。

参 考 文 献

[1] AU K W Y, ZHOU Y F, HUANG Z, et al. PScout: analyzing the Android permission specification [J]. Proceedings of the 2012 ACM Conference on Computer and communications security, 2012: 217-228.

[2] ARZT S, RASTHOFER S, BODDEN E. SuSi: A tool for the fully automated classification and categorization of android sources and sinks [R]. Darmstadt: University of Darmstadt, 2013.

[3] FAN M, LIU J, LUO X, et al. Android malware familial classification and representative sample selection via frequent subgraph analysis [J]. IEEE Transactions on Information Forensics and Security, 2018, 13(8): 1890-1905.

[4] OU F, XU J. S3Feature: A static sensitive subgraph-based feature for android malware detection [J]. Computers & Security, 2022, 112: 102513.

[5] LIU Z, WANG R, JAPKOWICZ N, et al. SeGDroid: An Android malware detection method based on sensitive function call graph learning [J]. Expert Systems with Applications, 2024, 235: 121125.

[6] ARP D, SPREITZENBARTH M, HUBNER M, et al. Drebin: Effective and explainable detection of android malware in your pocket [C]// Proceedings of the Ndss 2014. 2014: 23-26.

[7] BRUZZESE R. Building visual malware dataset using VirusShare data and comparing machine learning baseline model to CoAtNet for malware classification [C]// Proceedings of the 2024 16th International Conference on Machine Learning and Computing. 2024: 185-193.

[8] QURESHI M A, GILL A M, SADAF M. APK Insight: Revolutionizing Forensic Analysis with a User-Friendly Approach [C]// 2024 International Conference on Engineering & Computing Technologies (ICECT). IEEE, 2024: 1-6.

[9] BHOOSHAN P, SONKAR N. Comprehensive Android Malware Detection: Leveraging Machine Learning and Sandboxing Techniques Through Static and Dynamic Analysis [C]// 2024 IEEE 21st International Conference on Mobile Ad-Hoc and Smart Systems (MASS). IEEE, 2024: 580-585.

[10] HALL-ANDERSEN M, SIMKIN M, WAGNER B. FRIDA: Data availabilitysampling from FRI [C]// Annual International Cryptology Conference. Cham: Springer Nature Switzerland, 2024: 289-324.

[11] KHAN S A, ADNAN M, ALI A, et al. An Android applications vulnerability analysis usingMobSF [C]// International Conference on Engineering & Computing Technologies (ICECT 2024). IEEE, 2024: 1-7.

[12] YANG C, XU Z, GU G, et al. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications [C]// 19th European Symposium on Research in Computer Security (ESORICS 2014). Wroclaw, Poland, Part I. Wroclaw: Springer International Publishing, 2014: 163-182.

[13] YANG W, XIAO X, ANDOW B, et al. Appcontext: Differentiating malicious and benign mobile app behaviors using context [C]// Proceedings of the 37th IEEE/ACM International Conference on Software Engineering. IEEE, 2015: 303-313.

[14] PENDLEBURY F, PIERAZZI F, JORDANEY R, et al. TESSERACT: Eliminating experimental bias in malware classification across space and time [C]// Proceedings of the 28th USENIX Conference on Security Symposium (SEC '19). USA: USENIX Association, 2019: 729-746.

[15] ZHANG X, ZHANG Y, ZHONG M, et al. Enhancing state-of-the-art classifiers with API semantics to detect evolvedAndroid malware [C]// Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. New York:

ACM,2020:757-770.

[16] CAI M,JIANG Y,GAO C,et al. Learning features from enhanced function call graphs for Android malware detection [J]. Neurocomputing,2021,423:301-307.

[17] LO WW,LAYEGHY S,SARHAN M,et al. Graph neural network-based Android malware classification with jumping knowledge [C]// IEEE Conference on Dependable and Secure Computing(DSC 2022). IEEE,2022:1-9.

[18] WANG L,WANG H,HE R,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.

[19] AZEEM M,KHAN D,IFTIKHAR S,et al. Analyzing and comparing the effectiveness of malware detection:A study of machine learning approaches [J]. Heliyon,2024,10(1).

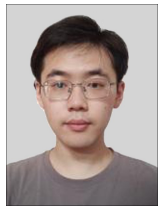
[20] BAI KV,THIRUMARAN M. Hybrid Deep Learning and Behavioral Analysis for Enhanced Malware Detection in Banking [C]// 8th International Conference on Electronics,Communication and Aerospace Technology(ICECA 2024). IEEE,2024:1168-1173.

[21] ALLIX K,BISSYANDÉ T F,KLEIN J,et al. Androzoo:Collecting millions of Android apps for the research community [C]// Proceedings of the 13th International Conference on Mining Software Repositories. 2016:468-471.

[22] SUN Z,WANG G,LI P,et al. An improved random forest based on the classification accuracy and correlation measurement of decision trees [J]. Expert Systems with Applications,2024,237:121549.

[23] DESAI M,SHAH M. An anatomizationon breast cancer detection and diagnosis employing multi-layer perceptron neural network(MLP) and Convolutional neural network(CNN) [J]. Clinical eHealth,2021,4:1-1.

[24] KOOPIALIPOOR M,ASTERIS P G,MOHAMMED A S,et al. Introducing stacking machine learning approaches for the prediction of rock deformation [J]. Transportation Geotechnics,2022,34:100756.



CHEN Weiguo, born in 2001, postgraduate. His main research interests are in software engineering and software analysis.

ZHANG Gaofeng, born in 1983, Ph.D. His main research interests include cloud/edge computing, software security, public safety, and software engineering.