

基于API语义聚类的安卓恶意软件检测方法研究

王练, 青山科

(重庆邮电大学, 重庆 400065)

摘要: 针对传统安卓恶意软件检测方法在面对变种样本时存在检测准确性下降和模型老化问题, 提出一种基于应用程序编程接口(application programming interface, API)语义聚类的恶意软件检测方法。首先, 通过函数调用图优化算法去除冗余节点, 并结合Skip-Gram模型对API调用序列进行语义向量嵌入, 以捕捉API间的功能关联与上下文相似性; 然后, 利用K-means聚类将语义相近的API归为同类, 并通过动态扩展算法适应新出现的API。为验证所提方法的有效性, 实验部分首先基于2017—2018年的大量数据集进行训练, 然后针对2019—2023年进化的恶意样本进行检测。将所提方法与经典检测方法MaMaDroid和MalScan等开展对比实验, 结果显示所提方法的平均 $F1$ 分数达到96.6%。在模型老化实验中, 所提方法的平均 $F1$ 值提升了8.3%~23.1%。此外, 通过API语义特征对比实验和不同类簇数量实验, 进一步验证了语义提取精度与类簇设置对检测性能具有提升效果。综合上述实验结果可知, 所提方法能够实现更高的检测准确性和更低的老化速度。

关键词: 安卓恶意软件; 语义聚类; API调用图; 模型老化; 函数调用图

中图分类号: TP393

文献标志码: A

DOI:10.11959/j.issn.2096-109x.2025075

Research on Android malware detection method based on API semantic clustering

WANG Lian, QING Shanke

Chongqing University of Posts and Telecommunications, Chongqing 400065, China

Abstract: To address the challenges of declining detection accuracy and model aging in traditional Android malware detection methods when encountering variant samples, a malware detection method based on API semantic clustering was proposed. Redundant nodes were eliminated through a function call graph optimization algorithm, and API call sequences were embedded into semantic vectors using the Skip-Gram model to capture functional and contextual similarities between APIs. Semantically similar APIs were then grouped via K-means clustering, and a dynamic extension algorithm was introduced to adapt to newly emerged APIs. To verify the effectiveness of the proposed method, experiments were conducted by training on a large-scale dataset from 2017—2018, followed by detection on evolved malicious samples from 2019—2023. The proposed method was compared with the classic detection methods such as MaMaDroid and Malscan through experiments. The results showed that the average $F1$ score of the proposed method reached 96.6%. In model aging experiments, the proposed method improved the average $F1$ -score by 8.3%~23.1%. Furthermore, comparative experiments on API semantic features and cluster quantity analysis confirm that the precision of semantic extraction and cluster configuration significantly improves detection performance. Experimen-

收稿日期: 2025-03-21; 修回日期: 2025-07-06

基金项目: 重庆市教委科学技术研究项目(KJZD-K202500604)

Foundation Item: The Science and Technology Research Program of Chongqing Municipal Education Commission (KJZD-K202500604)

tal results indicate that the proposed method achieves higher detection accuracy and exhibits a slower aging rate.

Keywords: Android malware, semantic clustering, API call graph, model aging, function call graph

0 引言

21 世纪以来, 移动通信技术蓬勃发展, 移动网络基础设施实现全域覆盖, 智能终端设备全面普及, 移动互联网服务深刻改变了人们的生产、生活方式。安卓 (Android) 操作系统凭借其开源特性和出色的跨平台兼容性, 在智能终端领域得到广泛应用, 并占据市场主导地位^[1]。然而, 这种开放生态也为恶意软件提供了可乘之机。卡巴斯基 2023 年第一季度安全报告显示, 其第一季度共检测到 307 529 个移动设备恶意软件安装包^[2]。这些恶意软件通过隐蔽渠道植入用户设备, 实施数据窃取、远程控制等恶意行为, 严重威胁用户的数字资产和隐私安全。为应对这一挑战, 各类恶意软件检测工具被广泛部署于应用市场和终端设备^[3]。在此背景下, 如何提升恶意软件检测的准确率和高效性, 已成为学术界和产业界共同关注的核心议题。

1 相关工作

当前, 相关研究主要聚焦于恶意软件检测的理论与方法。现有安卓恶意软件检测方法可分为静态检测和动态检测两大类^[4], 二者各有优劣。动态检测能在应用程序运行时实时监控其行为, 及时发现恶意活动, 并在抗混淆方面表现优秀, 但其检测过程耗时较长^[5]。静态检测通过分析应用程序的代码与结构实现检测, 通常比动态检测速度更快、效率更高, 但其在应对代码混淆时能力稍显不足^[6-7]。随着机器学习技术的发展, 基于数据驱动的检测方法展现出更优越的性能^[8]。近年来, 深度学习在安卓恶意软件检测领域取得显著进展。Dong 等^[9]提出了一种融合权限与 API 调用图的 CNN-DNN 混合架构, 通过结合语义抽象与空间建模能力, 显著提升了模型对复杂恶意行为的识别精度。Shu 等^[10]系统综述了 CNN 在安卓检测中的典型应用, 指出现有方法在特征表达与鲁棒性方面仍存在不足; Naeem 等^[11-12]提出的基于图像纹理特征的堆叠集成模型, 不仅增强了对恶意软件变种的检测能力,

还通过颜色图像与神经网络集成优化了跨平台检测性能。

然而, 恶意软件开发者不断通过修改恶意行为来规避检测, 导致新旧恶意软件在特征分布上存在显著差异, 从而引发概念漂移问题^[13-16]。概念漂移是指数据流的统计特性随时间发生变化的现象, 这会导致基于历史数据训练的机器学习模型检测新样本时性能显著下降, 即出现模型老化问题^[17]。卡巴斯基 2019 年发布的白皮书^[18]显示, 某商业机器学习分类器的检测准确率在短短 3 个月内从接近 100% 急剧下降至 60%。

针对模型老化问题, 学术界已开展一系列研究。早期研究中, D'Angelo 等^[19]发现恶意软件常通过删除、替换或添加无用的应用程序编程接口 (application programming interface, API) 调用来逃避检测, 为此提出基于关联规则的检测方法, 通过提取非相邻且具有代表性的 API 子序列来应对这类对抗性修改。Ariconti 等^[20]提出的 MaMaDroid 方法能够适应安卓规范的变化, 该方法首先将 API 调用抽象为来自各安卓应用程序的 API 执行路径中的相应包, 然后进一步抽象为马尔可夫模型, 最终生成应用程序的特征表示。Yumlembam 等^[21]将信息检索领域的最佳匹配 (best matching, BM) 25 评分函数引入恶意软件检测, 利用 BM25 评分函数计算每个 API 的得分, 以捕捉良性应用与恶意应用的差异, 并基于这些 API 的 BM25 得分进行特征选择。但该方法存在漏报率高和鲁棒性不足的缺陷。Wu 等^[22]提出的 MalScan 框架通过 API 调用关系和社交网络分析的高可扩展性实现高效检测, 其将应用的功能调用图视为复杂社交网络, 通过社交网络中的中心性分析发现调用图的中心节点。但上述方法均缺乏对函数调用图的精简与优化, 难以提取有效的 API 上下文信息。

针对恶意软件演化过程中新旧变种在 API 功能上保持相似性的特点, 现有研究提出通过 API 聚类来提升检测模型的适应性。这类方法的核心思想是将功能相似的 API 归为同一类别,

并在特征表示中使用聚类中心替代原始API,从而增强检测方法对API变化的鲁棒性。Zhang等^[23]设计了APIGraph,通过API聚类对现有安卓恶意软件分类器进行了增强。其认为具有相似参数、权限和返回值的API必然具有相似的功能。该方法能从特征空间捕获恶意软件变体的内在语义,从而延缓分类器老化。然而,该方法未能充分利用API方法名蕴含的丰富语义信息,而这些信息往往能直接反映API的功能特性。Cui等^[24]提出一种基于图的API嵌入方法用于恶意软件检测,该方法通过构建时间过程图(time process graph, TPG)和时间API图(time API graph, TAG),分别对进程间和进程内的行为进行建模,并设计启发式随机游走算法捕获恶意行为。然而,该方法忽略了权限和异常等关键特征。

API调用作为软件运行的核心行为,其调用序列的组合模式能够体现应用程序的特定行为语义。然而,安卓应用程序的API本身并不具备明确的语义特征,无法直接区分正常或恶意行为。此外,现有API聚类方法在语义提取过程中往往忽略方法名和权限等关键特征,导致聚类结果对分类器性能的提升有限。针对这些问题,本文提出一种基于API语义聚类的安卓恶意软件检测方法。本文主要贡献如下。

1) 设计了一种基于API语义聚类的方法。在API语义特征提取阶段,运用递归算法提取完整的API上下文调用序列。在向量嵌入阶段,利用Skip-Gram模型进行训练,构建由调用路径中所

有唯一的API组成的词汇表。然后,该框架将每个目标API与其相邻API配对,并使用全部的API配对来训练Skip-Gram模型。与传统基于调用关系的方法不同,该方法更注重API间的语义相似性,能有效识别API的功能类别和行为特征。

2) 引入了一种调用图优化算法。通过去除调用图中的未知节点,同时保持其前驱节点和后继节点的连接关系。优化后的调用图能够更准确地反映应用程序的真实行为特征,避免无关节点对API语义的干扰。

2 基于API语义聚类的安卓恶意软件检测方法

基于API语义聚类的安卓恶意软件检测方法主要包括静态分析、特征提取、API聚类和扩展、模型训练和检测结果分类4个阶段,整体架构如图1所示。

2.1 静态分析阶段

本阶段的主要目标是从安卓应用(APK文件)中提取有效信息。具体而言,首先使用apk-tool^[25]工具对APK文件进行反编译,从而获取应用的清单文件和Dex文件。其中,Dex文件包含提取API特征所需的函数调用图和Dalvik字节码特征等信息。

为提取具有语义表征能力的API调用序列,需要先构建精确的函数调用图。具体实现过程如下:①使用baksmali工具将.dex文件反编译为.smali中间代码,完整保留字节码的语义信息;②分析.samli文件中的函数调用指令,构建函数

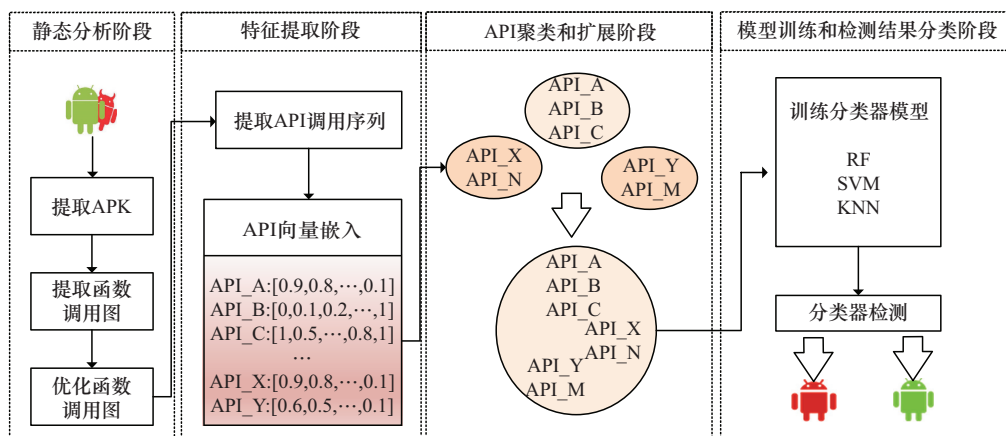


图1 所提方法的整体架构

调用图^[26]。由于函数调用图能够准确表征软件中 API 的具体行为特征, 因此基于该图提取的 API 调用序列具有充分的语义信息。图 2 展示了函数调用图的基本结构, 由函数(节点)及函数之间的调用关系组成, 通过调用序列的组合最终形成软件的具体行为模式。不过, 现有函数调用图中普遍存在大量未知函数节点, 而现有检测方法对这些未知函数节点通常采取全部保留策略^[27]。

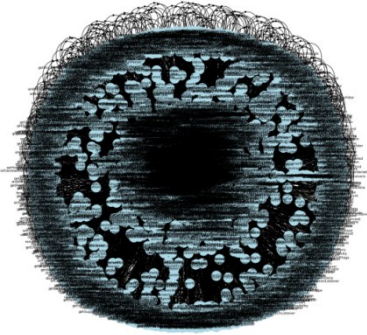


图 2 API 调用图

针对大规模图结构中节点数量过多导致图学习时间成本增加的问题, 本研究提出一种基于图结构优化的节点精简方法。具体而言, 本方法在函数调用图构建阶段即实施优化策略, 通过算法 1 所示的优化流程对原始图结构进行精简。

算法 1 调用图优化算法

输入 初始函数调用图 FCG, 安卓官方文档中的 API 集合 AODs

输出 优化后的函数调用图 FCG_{final}

```

1. for node in FCG do
2.   if node is entry function then
3.     continue
4.   end if
5. if node not in AODs then
6.   continue
7. end if
# 过滤调用图中的入口函数、官方 API 集合
8. nodecallers ← FCG.collect(predecessor(node))
# 收集当前节点的前驱节点
9. nodecallers ← FCG.collect(successor(node))
# 收集当前节点的后继节点

```

```

10. for nodestart in nodecallers do
11. for nodeend in nodecallers do
12. FCG.add_edge(nodestart, nodeend)
# 连接调用节点和被调用节点
13. end for
14. end for
15. remove node from FCG
# 从调用图中移除当前节点
16. end for
17. return FCGfinal

```

算法 1 保留了入口函数节点和官方 API 集合(AODs)。入口函数节点包括应用程序执行流程的起始节点, 如 Activity.onCreate()等。官方 API 集合(AODs)通过解析 Android SDK 文档(API Level 24-30)构建(生成的 API 白名单), 包含 android.、java.、javax.*等标准包路径下的核心 API。此外, 需要移除所有未被白名单收录的未知调用节点, 以降低函数调用图的冗余度, 并重建受影响节点的前驱和后继连接关系, 以保留核心 API 调用语义。经上述处理得到的精简调用图更聚焦核心 API 行为, 可以提升后续语义分析的效率和准确性。

2.2 特征提取阶段

本阶段的核心任务是从函数调用图中提取具有丰富语义信息的 API 调用序列。由于函数调用图是有向图结构, 仅依赖直接邻接节点难以完整刻画 API 的调用上下文。因此, 本文采用递归遍历算法提取对应函数的完整 API 调用链, 确保获取足够的语义信息。在将函数调用图转换为邻接矩阵时, 自定义函数节点会被映射为对应的 API 调用序列, 从而增强语义表达能力。实现流程如算法 2 所示。

算法 2 API 序列提取

输入 函数调用图的节点 FCG_{nodes}

输出 API 调用路径 path_{API}

```

1. pathAPI ← []
# 存储所有 API 调用路径
2. for node in FCGnodes do
3.   pathcurrent ← []
# 存储当前节点及其邻居节点路径
4.   nodevisited ← []

```

```

# 存储访问过的节点
5. if node has neighbour then
# 判断当前节点是否有邻居
6. for each nodeneighbour do
# 遍历当前节点的每个邻居
7. if nodeneighbour not in nodevisited then
# 若邻居节点还未被访问
8. nodevisited ← nodeneighbour
# 将邻居节点标记为已访问
9. pathcurrent ← nodeneighbour
# 将邻居节点加入当前路径中
10. end if
11. end for
12. end if
13. pathAPI ← pathcurrent
14. end for
15. return pathAPI
# 返回所有API的路径

```

在完成API调用序列的特征提取后，需要将这些特征转换为具有语义表征能力的向量形式。这一转换过程面临3个主要挑战。①特征维度不一致性：不同API的调用上下文长度存在显著差异。②权限依赖性差异：部分API（如涉及敏感数据的操作）需要特定权限才能执行，而基础功能API通常不需要权限就能执行。③调用频率不均衡：系统核心API的调用频率远高于特定功能API。为解决上述问题，本研究引入了基于Skip-Gram模型的分布式表示方法。模型参数（如上下文窗口大小、向量维度等参数）设置借鉴了文献[28]中经过验证的配置方案，具体见表1。这些参数在API语义建模与嵌入表示任务中已被广泛验证，能够有效捕捉API的语义特征。

表1 模型参数设置

参数	数值
上下文窗口大小	8
向量维度	128
负采样数量	5
训练轮次	10
初始学习率	0.025
最小词频	5
多线程数	4

本文采用Skip-Gram模型实现API语义向量化，主要基于两方面考量。①模型适配性：Skip-Gram模型特别适合处理序列化数据，能有效捕捉API调用序列中的上下文关联。②语义表征能力：该模型可使具有相似调用上下文的API在向量空间中距离相近。具体而言，首先根据API在一组应用程序中的上下文，将每个API映射到多维向量空间，向量取值限定在[0,1]区间。其中，API的上下文被定义为固定大小窗口内的相邻API集合，该窗口包含所有提取到的API调用序列。在向量空间中，具有相似上下文的不同API会呈现较近的映射关系。

Skip-Gram模型的目标是在给定中心词 w_t 的情况下，预测上下文词 w_{t+s} 出现的概率，其数学表达式如式(1)所示。

$$L(\theta) = \frac{1}{T} \prod_{t=1}^T \prod_{-m \leq s \leq m, s \neq 0} \log P(w_{t+s}|w_t) \quad (1)$$

其中， T 表示语料库中单词的总数， m 表示窗口大小， $\log P(w_{t+s}|w_t)$ 表示给定中心词 w_t 时预测上下文词 w_{t+s} 的条件概率。

Skip-Gram模型采用Softmax函数计算上下文词的条件概率，其数学表达式如式(2)所示。

$$P(w_{t+s}|w_t) = \frac{\exp(\mathbf{v}_{w_t}^T \mathbf{v}_{w_{t+s}})}{\sum_{k=1}^V \exp(\mathbf{v}_{w_t}^T \mathbf{v}_k)} \quad (2)$$

其中， $\mathbf{v}_{w_t}$ 表示中心词 w_t 的词向量。Softmax函数将词向量内积得分 $\mathbf{v}_{w_t}^T \mathbf{v}_{w_{t+s}}$ 转换为标准化概率分布，确保所有可能上下文词的概率之和等于1。内积得分 $\mathbf{v}_{w_t}^T \mathbf{v}_{w_{t+s}}$ 直接反映两个词的语义关联强度。当得分显著大于0时，表明中心词与上下文词语义相关性较强，在嵌入空间中距离较近；当得分接近0时，表明中心词与上下文词语义相关性较弱，在嵌入空间中距离较远。

API向量嵌入的实现流程如图3所示。基于从API路径中提取的API调用序列，本方案在构建由路径中所有唯一的API组成的API词汇表时，对于其中的每个目标API及每条提取的API路径，系统会收集与该目标API相距不超过 S 个API的所有相邻API。随后，该框架将每个目标API与其相邻API进行一一配对，并利用这些API配对来训练Skip-Gram模型。

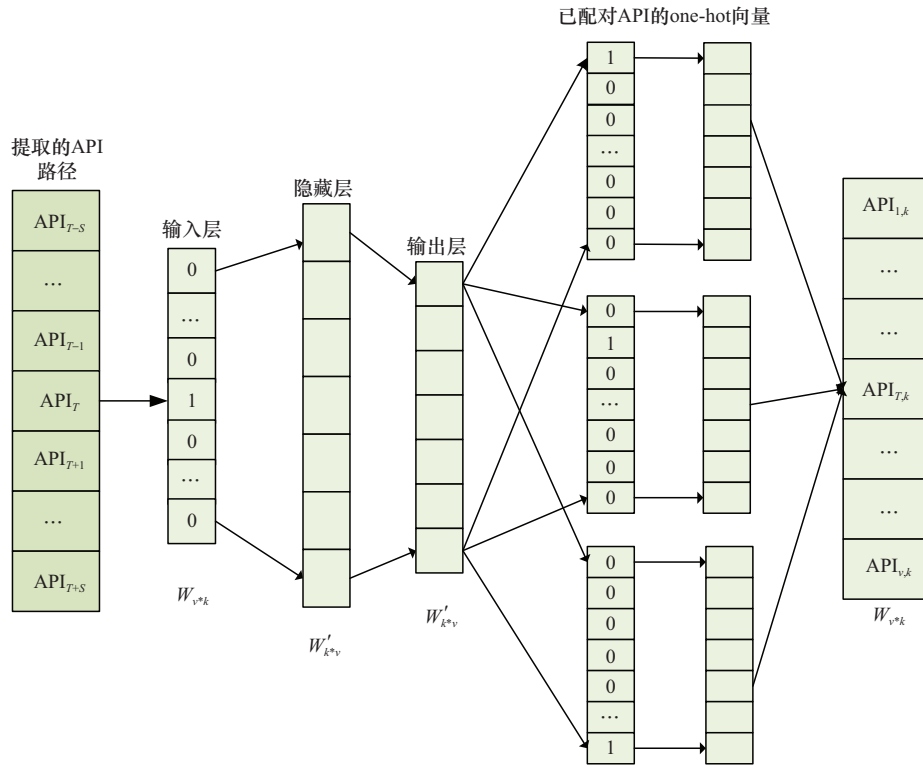


图3 API向量嵌入的实现流程

2.3 API聚类 and 扩展阶段

在通过 Skip-Gram 模型完成不同 API 的向量嵌入后, 语义相近的 API 在特征向量空间中的欧氏距离较近。为进一步分析这些向量, 本文采用聚类算法进行处理。

在训练阶段, 使用 K-means 聚类算法将 API 向量划分为 k 个簇, 每个 API 向量归属于平均值最近的簇, 各簇规模差异不超过 1。基于聚类结果构建初始特征集, 建立簇与特征的一一映射关系。同一簇中的 API 被赋予相同特征, 反映其在上下文中的语义相似性。

当新时间段的数据集到来后, 采用相同的特征处理方法生成新的 API 向量空间。对于新向量空间中出现的新 API, 通过类簇扩展算法进行处理。先计算新 API 向量与现有各簇中心的距离, 再将新的 API 分配到与其最接近的现有类簇中。这样不仅可以完成类簇特征集的扩展, 还能保持类簇数量不变, 基于语义相似性实现新 API 的有效归类。具体实现细节如算法 3 所示。

算法3 API类簇扩展算法

输入 旧类簇 $\text{Cluster}_{\text{old}}$, 新增 API 集合

API_{new} , API 字典 API_{dict}

输出 扩展后的类簇 $\text{Cluster}_{\text{new}}$

```

1. for api_new in API_new do
# 遍历新增的每个 API
2. minDis ← ∞
# 初始化最小距离
3. cluster_idx ← -1
# 初始化类簇索引
4. for cluster in Cluster_old do
# 遍历旧类簇
5. distances ← []
# 存储距离
6. for api in cluster do
# 遍历当前簇中的每个 API
7. if api in API_dict then
# 如果 API 在字典中
8. distances ← 计算 api 和 api_new 的距离
# 计算并存储距离
9. end if
10. end for
11. avgDis ← Average(distances)

```

```

# 计算平均距离
12. if avgDis < minDis then
# 若当前类簇的平均距离更小
13. minDis ← avgDis
# 更新最小距离
14. cluster_idx ← 当前簇的索引
15. end if
16. end if
17. if minDis < threshold then
# 若最小距离小于阈值
18. 将 apinew 扩展到 Clusternew 的第 cluster_idx
   个类簇
19. else
20. 将 apinew 作为新类簇添加到 Clusternew
21. end if
22. end for
23. return Clusternew

```

如图4所示, 向量空间中形成了两个由训练集生成的聚类, 分别是包含特征向量 $\mathbf{API}_A$ 、 $\mathbf{API}_B$ 的特征空间 M , 以及包含 $\mathbf{API}_C$ 的特征空间 Q 。针对测试集生成的新向量 $\mathbf{API}_D$, 系统执行以下处理流程: 首先计算 $\mathbf{API}_D$ 与各特征空间的平均距离, 包括与特征空间 M ($\mathbf{API}_A$ 、 $\mathbf{API}_B$) 和特征空间 Q ($\mathbf{API}_C$) 的距离度量。计算结果表明, $\mathbf{API}_D$ 与特征空间 M 的平均距离最小, 因此系统将其归入特征空间 M 。在此过程中, 特征空间 Q 保持原有组成不变。

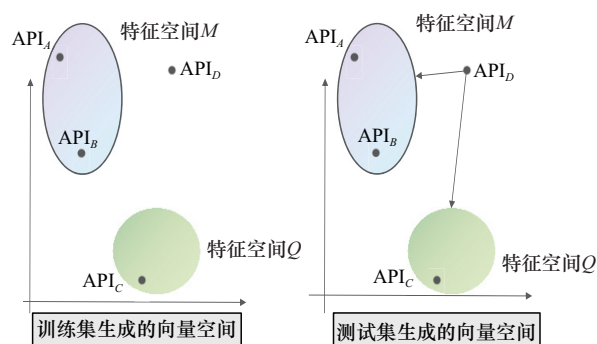


图4 API 聚类过程

2.4 模型训练和检测结果分类阶段

在训练阶段, 给定一个特征集合 (该集合既可以是训练阶段的初始特征集合, 也可以是测试阶段的扩展特征集合), 通过特征集合与应用特

征向量之间的一对一映射, 为每个应用生成一个二进制特征向量。本文为训练集中的每个应用生成一个特征向量, 并将该特征向量与应用的真实标签相关联, 再利用所有特征向量及相关标签训练分类模型。

在测试阶段, 本文先为测试集中的每个应用生成一个特征向量, 再将训练好的分类模型作为工具, 以每个应用的特征向量为输入, 输出对应的预测标签。

3 实验与评估

3.1 评价指标

本文采用准确率 (Acc)、精确率 (Precision)、召回率 (Recall) 和 $F1$ 分数 ($F1_Score$) 等指标评估本文方法的性能。这些指标基于真阳性 (true positive, TP)、真阴性 (true negative, TN)、假阳性 (false positive, FP) 和假阴性 (false negative, FN)。TP 表示恶意样本被正确检测为恶意样本的数量, TN 表示正常样本被正确检测为正常样本的数量, FP 表示正常样本被错误检测为恶意样本的数量, FN 表示恶意样本被错误检测为正常样本的数量。

准确率是指模型预测正确的样本占总样本的比例, 其表达式如式(3)所示。

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \quad (3)$$

精确率是指模型预测为正的样本中实际为正的样本的比例, 其表达式如式(4)所示。

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

召回率是指实际为正的样本中被模型正确预测为正的样本的比例, 其表达式如式(5)所示。

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

$F1$ 分数是精确率和召回率的调和平均数, 用于评估模型的整体性能, 其表达式如式(6)所示。

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall} \quad (6)$$

3.2 实验设置

本文实验的软硬件配置见表2。实验所用的良性软件与恶意软件样本集信息见表3。其中, 良性软件主要从 Google Play 商店、应用宝等第三方平台获取; 恶意软件样本集涵盖木马、蠕虫、广告软件、间谍软件、勒索软件等类型, 来

源于 VirusTotal、AndroidZoo 及 AMD 等权威恶意软件数据库。该样本集共包含 6 477 个良性软件和 6 544 个恶意软件，覆盖的时间跨度为 2017—2023 年。

表 2 实验软硬件配置

配置	参数
操作系统	Windows10 64 位
处理器	Intel i5 3.2 GHz
内存	ARM 16 GB
显卡	3080 12 GB

表 3 实验软件样本集

年份	良性软件 样本/个	恶意软件 样本/个	样本 总数/个	平均大小/ MB
2017	981	983	1 964	2.18
2018	959	914	1 873	2.59
2019	953	933	1 886	4.41
2020	897	956	1 853	5.18
2021	917	896	1 813	5.21
2022	863	903	1 766	7.65
2023	907	959	1 866	7.45

本文通过 3 组实验对所提方法进行综合性能评估。首先，验证模型对恶意软件的识别准确率；然后，基于历史数据集训练模型，并使用新型样本集测试模型的泛化性能，分析模型的抗老化能力；最后，评估本文方法在不同机器学习分类模型下的检测效果。

本文采用十折交叉验证所提方法的性能。首先，将整个软件样本集均匀划分为 10 个互斥的子集，各子集样本量基本一致。在每一轮验证中，选取 9 个子集作为训练集用于模型训练，剩

余 1 个子集作为测试集用于模型性能评估。该过程重复 10 次，确保每个子集均被用作测试集 1 次，避免数据偏差。完成全部 10 轮测试后，计算各次评估指标的平均值，作为模型的最终性能评估结果。本文采用 $F1$ 分数、准确率、精确度和召回率 4 个关键指标综合评估所提方法的性能。

3.3 实验结果与分析

为深入评估所提方法的有效性，本文选取了 4 种与本文领域相近的方法开展对比实验。MaMaDroid^[20]通过从调用图中提取抽象函数调用序列来构建行为模型，并使用其提取特征进行分类。MalScan^[22]是一种轻量级的基于图的方法，其将应用程序的函数调用图视作社交网络，运用社交网络中心性分析来表示图的语义特征。API2Vec^[24]采用图模型表示原始序，借助设计的启发式随机游走算法捕捉细粒度恶意软件行为。SeGDroid^[29]通过融合 Word2Vec 与社交网络中心性节点的方法，提取函数调用图中的 API 语义及图结构信息。

为全面评估所提方法在恶意软件检测中的性能表现，本文使用上述 5 种方法分别对 2017—2023 年开发的恶意软件进行检测，并对这 7 年的检测结果求平均值。所有实验结果均通过随机森林机器学习算法训练与测试获得。实验结果见表 4。

实验数据显示，MaMaDroid 的 $F1$ 分数和 Acc 随着时间推移呈缓慢提升趋势，MalScan 在 2018 年的 Acc 降至 88.3%，反映该方法对 API 调用关系具有较高的敏感性；API2Vec 的 Acc 波动较大，原因在于其依赖的图模型对新 API 的嵌入效果不稳定。相比之下，本文方法的 $F1$ 分数和 Acc 均优于其他对比方法，进一步验证了 API 语义聚类 and 调用图优化的有效性。

表 4 同一年数据集上的对比实验结果

检测方法	2017 年		2018 年		2019 年		2020 年		2021 年		2022 年		2023 年	
	$F1_Score$	Acc	$F1_Score$	Acc	$F1_Score$	Acc	$F1_Score$	Acc	$F1_Score$	Acc	$F1_Score$	Acc	$F1_Score$	Acc
MaMaDroid	89.6%	89.9%	89.4%	89.5%	91.7%	91.9%	93.3%	93.5%	94.8%	92.6%	95.2%	94.8%	96.3%	96.3%
MalScan	91.6%	91.8%	87.4%	88.3%	90.5%	90.6%	91.5%	91.9%	92.3%	91.7%	92.1%	93.0%	94.2%	95.3%
API2Vec	89.3%	90.2%	92.6%	89.6%	93.5%	90.4%	94.9%	91.3%	93.6%	90.5%	94.5%	92.1%	96.1%	93.0%
SeGDroid	94.4%	94.2%	95.6%	95.3%	94.3%	93.9%	95.9%	95.6%	96.0%	95.4%	97.3%	96.0%	96.1%	96.2%
本文方法	95.8%	96.5%	96.1%	95.8%	95.2%	96.5%	96.9%	97.1%	97.1%	96.4%	96.3%	98.2%	97.9%	98.5%

为系统评估API聚类处理对模型检测性能及运行效率的影响,本文以2017—2018年恶意软件样本为训练集、2019—2023年恶意软件样本为测试集,采用逐年评估的方式进行分析,实验结果见表5。实验数据显示,未聚类模型的性能随时间推移逐年下降(如 $F1$ 分数从2019年的72.7%降至2023年的33.1%),表现出显著的模型老化现象。经过聚类处理后,各项指标均有明显提升(2023年的 $F1$ 分数提高至71.6%)。此外,聚类处理将特征空间从高维稀疏的原始API序列压缩为具有语义一致性的类簇向量,大幅减少了检测时间开销,平均训练时间从1 185.27 s降至平均14.95 s。

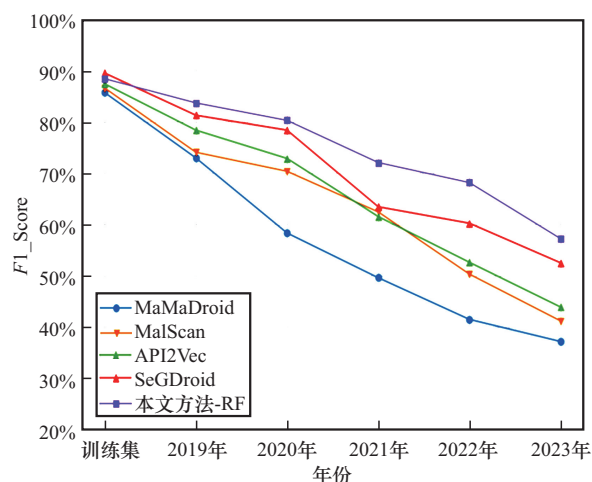
为验证所提方法的抗老化性能,本文设置了以下两种实验场景。

1) 场景1: 使用2017—2018年恶意数据样本作为训练集,构建分类器模型,使用2019—2023年新出现的恶意样本作为测试集,评估该分类器的检测性能。

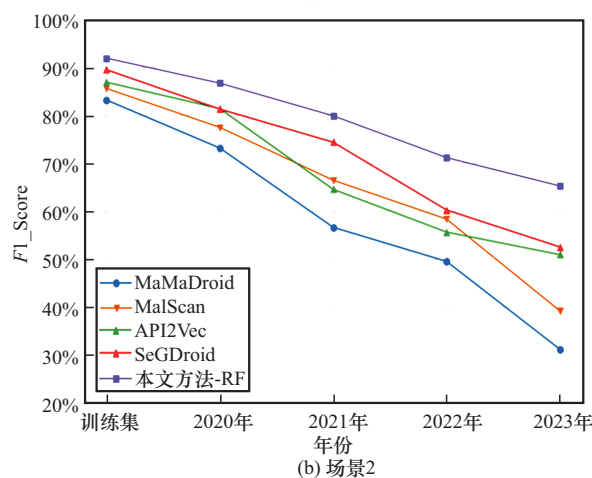
2) 场景2: 使用2018—2019年恶意数据样本作为训练集,构建分类器模型,使用2020—2023年新出现的恶意样本作为测试集,评估该分类器的检测性能。

实验结果如图5所示。本文方法显著优于MaMaDroid。以图5中场景2为例,在近4年的测试集上,本文方法的平均 $F1$ 分数分别为86.6%、79.8%、71.1%、65.2%,MaMaDroid的平均 $F1$ 分数分别为73.1%、56.6%、49.5%、31.2%,MalScan的平均 $F1$ 分数分别为77.4%、66.4%、58.3%、39.2%,API2Vec的平均 $F1$ 分数分别为81.3%、64.5%、55.6%、50.9%,SeGDroid的平均 $F1$ 分数分别为81.2%、74.3%、60.2%、52.5%。在所有的实验中,当使用相同

的训练集和测试集进行评估,本文方法的平均 $F1$ 分数比MaMaDroid、MalScan、API2Vec、SeGDroid分别高了23.1%、15.4%、12.6%、8.6%。就老化速度而言,本文方法的 $F1$ 分数在4年间平均每年下降7.1%。而在相同实验数据集下,MaMaDroid、MalScan、API2Vec和SeGDroid的 $F1$ 分数在4年间平均每年分别下降13.9%、12.7%、10.1%、9.6%。



(a) 场景1



(b) 场景2

图5 不同方法预测跨年份数据样本的 $F1$ 分数对比

表5 模型在未聚类 and 聚类处理情况下的检测性能对比

数据集	Acc		Precision		Recall		$F1_Score$		时间开销/s	
	聚类前	聚类后	聚类前	聚类后	聚类前	聚类后	聚类前	聚类后	聚类前	聚类后
2019年	76.4%	91.2%	73.9%	89.7%	71.6%	92.5%	72.7%	91.0%	1 185.21	14.78
2020年	63.0%	88.3%	60.4%	87.2%	58.1%	89.5%	59.2%	88.3%	1 187.66	14.55
2021年	51.2%	84.1%	49.6%	83.4%	46.9%	85.3%	48.2%	84.2%	1 186.07	14.71
2022年	42.5%	76.2%	40.1%	74%	38.8%	77.9%	39.4%	76.2%	1 180.44	15.13
2023年	36.3%	71.4%	35.7%	69.5%	32.5%	73.8%	33.1%	71.6%	1 186.96	15.60

如图 6 所示, 本文对比了不同分类模型在特征提取结果上的检测效能差异。实验采用 4 类评估指标: TPR 表示恶意样本识别率, TNR 表示良性样本判别精度, $F1$ 分数和准确率则综合衡量整体分类效果。实验结果显示, 1NN 分类器在所有指标上均表现最优, 表明其对 API 调用序列的局部特征模式的捕捉能力更强。相比之下, SVM 因难以处理 API 语义向量的非线性分布, 各项指标均较低。

为提升恶意软件检测系统对变种 API 的鲁棒性, 本文在 API 语义嵌入基础上, 引入 K-means 算法对向量空间中的 API 进行聚类, 以构建稳定的特征表示。为评估聚类数量对 API 语义聚类检测方法性能的影响, 实验采用 2017—2023 年的安卓应用数据集, 统一使用 Skip-Gram 模型对 API 序列进行语义向量嵌入, 并在不同 k 值设置下对向量空间进行聚类。随后, 采用随机森林分

类器对由聚类构建的特征向量进行训练与预测, 并使用 Acc、Precision、Recall、 $F1_Score$ 这 4 项指标进行性能评估。实验结果如图 7 所示, 模型在聚类数量 k 为 150 左右时达到最佳性能。随着 k 值从 10 个递增至 150 个, API 聚类粒度逐渐细化, 使得语义上相似但结构上不同的 API 能够被有效归为同一类, 从而增强特征表示的泛化能力和上下文一致性。随着 k 值进一步增大, 分类性能反而略有下降, 其原因可能是过度聚类导致特征空间变得稀疏, 从而降低了模型对行为相似样本的识别能力。因此, 综合考虑效率和检测性能, 本文最终选取的类簇数目为 150 个。

为评估方法名与权限特征在模型中的具体贡献, 本文设计了 4 组对照实验。所有实验均保持 API 序列嵌入方式、聚类方法和分类器结构一致, 仅在特征构建阶段引入不同的语义信息。具体方案如下: 方案 A 仅以 API 路径作为输入的基

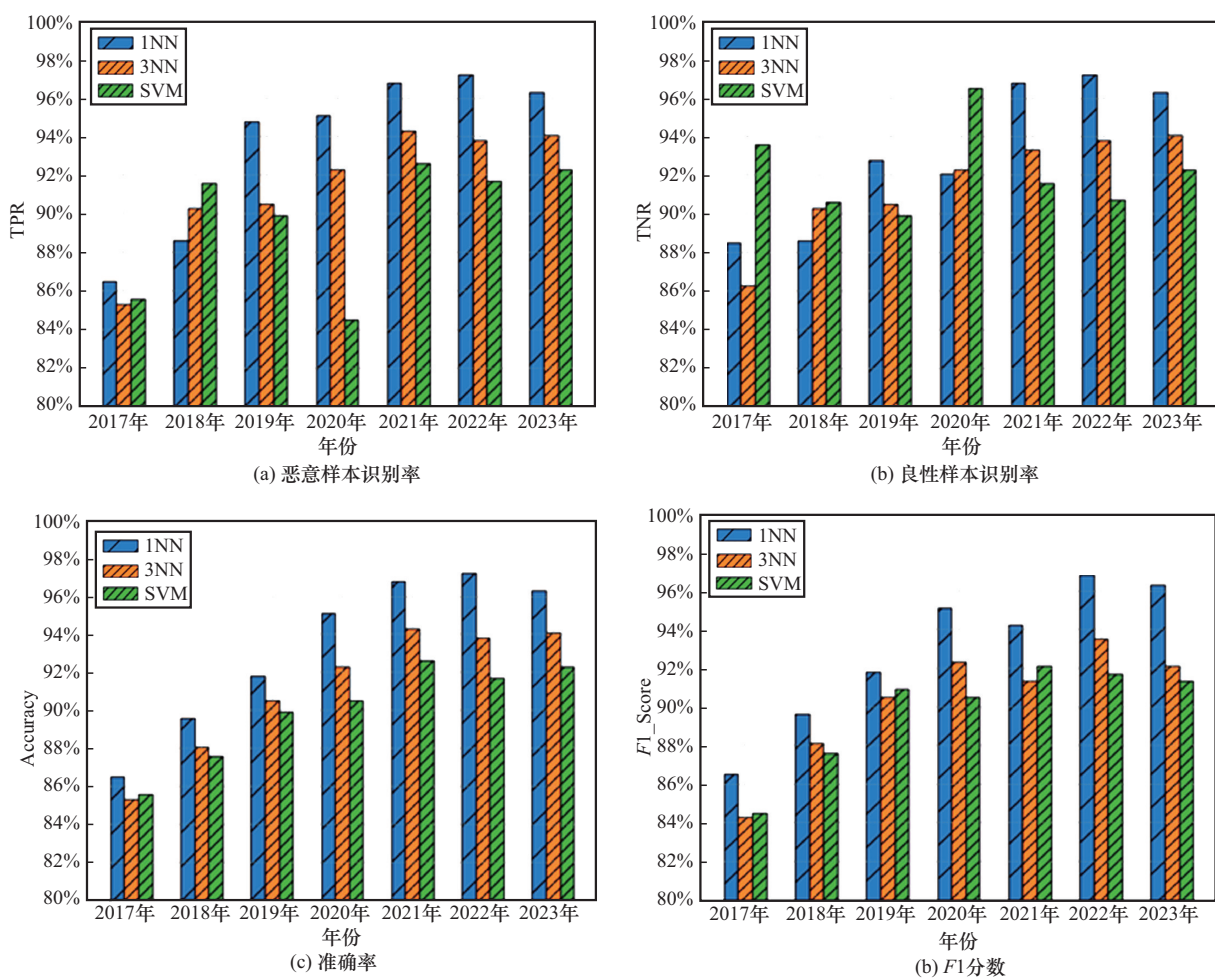


图 6 不同机器学习模型下的检测效果

准组；方案B在API基础上增加方法名信息，用于评估方法名对模型性能的提升效果；方案C在API基础上加入对应权限特征，用于评估权限语义的独立贡献；方案D同时引入方法名与权限信息。最终选取方案D作为本文的完整方案。通过对比不同特征组合下模型的准确率与F1分数变化，进一步验证各语义特征的有效性与互补性。实验结果见表6。

表6 不同特征组合对检测性能的影响

对比方案	Acc	F1_Score	Precision	Recall
方案A	91.0%	89.8%	90.2%	89.4%
方案B	93.6%	92.1%	92.4%	91.8%
方案C	94.1%	92.9%	93.2%	92.6%
方案D	96.8%	95.4%	95.6%	95.2%

方案A仅以API路径作为输入特征，虽然能够反映部分调用行为，但其语义表达能力存在明显局限性。这种局限性主要体现在：不同功能的API可能因路径命名风格相似而出现聚类混淆。以安卓API的全限定路径（如“Landroid/telephony/SmsManager->sendTextMessage”）为例，路径信息主要用于标识类与接口，而实际描述行为语义的关键信息往往在方法名部分（如sendTextMessage、getDeviceId和delete等方法名）。方案B在引入方法名后，显著提升了模型对API调用意图的理解精度，进而提升性能。方案C在引入权限特征后，模型得以从系统层面获取行为约束信息。权限作为独立的安全特征，不仅增强了模型的鲁棒性，更重要的是在向量空间中强化了恶意样本与正常样本的决策边界。例如，具有SEND_SMS和READ_CONTACTS等敏感权限的API更易被准确归类到高风险簇，从而显著提升了聚类结果的区分度。方法名与权限特征具有显著的互补效应，使得方案D取得了最优性能，最终将F1分数提升至95.4%。

本文在同一数据集上对MaMaDroid、MalScan、API2Vec、SeGDroid和本文方法进行了对比实验，以观察其运行时间开销，实验结果见表7。MaMaDroid、MalScan、API2Vec及SeGDroid均为利用调用图中API上下文信息的恶意软件检测方法，与本文方法具有相似性。因此，这些方法

的运行时间可划分为调用图提取、特征提取和分类3个阶段。

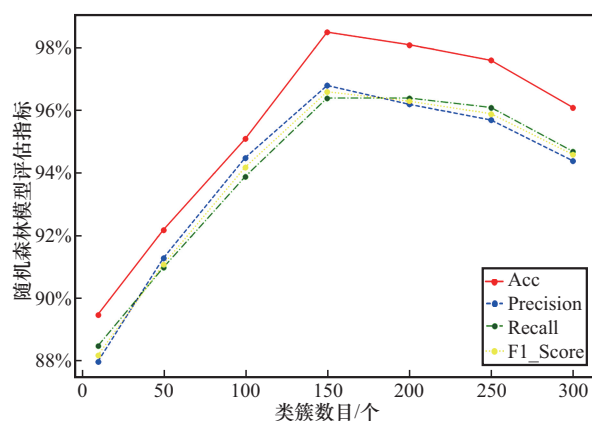


图7 不同类簇数目对分类性能的影响

表7 不同方法的运行时间开销

检测方法	调用图提取/s	特征提取/s	分类/s	总计时间开销/s
MaMaDroid	14.34	0.52	0.02	14.98
MalScan	3.91	3.29	0.02	7.23
API2Vec	4.15	4.35	0.02	8.52
SeGDroid	3.78	4.55	0.02	8.35
本文方法	13.29	0.97	0.02	14.28

在调用图提取阶段，须从APK文件中提取调用图。由于MaMaDroid与本文方法均使用FlowDroid生成调用图，因此这两种方法的时间成本几乎相同。而MalScan、API2Vec和SeGDroid使用AndroidGuard生成调用图。与FlowDroid生成调用图相比，AndroidGuard的时间开销大幅减少。

在特征提取阶段，核心任务是将调用图转换为特征向量，不同方法的时间开销存在差异。由于MalScan需要计算调用图中敏感API的中心性信息，因此该阶段时间开销最大。MaMaDroid将调用图中的函数调用对抽象为包调用对，并通过构造马尔可夫链来建模转移概率，耗时相对较少。API2Vec需要构建API关系文档，将API实体与API关系链接形成异构有向图，耗时相对较多。SeGDroid通过保留敏感API的上下文来减少图中节点的数量，分别构建Word2Vec和Opcode2Vec以嵌入外部和内部节点，从而减少了特征提取的时间开销。本文方法在提取API序列时

借助调用图进行优化,有效控制了时间成本,因此该阶段时间开销较小。

在分类阶段,各方法的处理方式如下:MaMaDroid先通过主成分分析进行降维,再使用随机森林分类器完成检测;MalScan、API2Vec和SeGDroid在此阶段均采用随机森林分类器进行检测,与本文方法一致。

在上述5种方法中,尽管MaMaDroid与本文方法的运行时间开销相对较大,但本文方法在检测效果和模型抗老化能力上均优于其他方案。

4 结束语

针对安卓恶意软件快速变种演化、传统检测方法因API语义信息不足导致模型老化的问题,本文提出一种基于API语义聚类的恶意软件检测方法。该方法通过增强语义特征分析,提升模型对新型恶意软件变种的适应能力,其核心流程包含3个阶段。①函数调用图优化阶段:设计基于官方API白名单的冗余节点过滤算法,去除无关调用节点,仅保留与恶意行为高度相关的关键节点,降低计算复杂度并提升特征质量。②语义表征学习阶段:提出上下文感知的API嵌入模型,采用改进的Skip-Gram架构,将API调用序列映射至低维语义向量空间,有效捕捉API间的功能关联与上下文相似性,增强语义表达能力。③构建动态聚类扩展体系阶段:采用K-means聚类对语义相近的API进行归类,并结合动态扩展算法,使模型能够自适应地融合新出现的API,增强模型对变种样本的适应性。实验结果表明,本文方法在恶意软件检测任务中表现优异(2017—2023年平均 $F1$ 分数达到96.5%),平均 $F1$ 分数提升了8.6%~14.9%。在基于2017—2018年数据集进行训练并对2019—2023年进化的恶意样本进行检测实验中,与MaMaDroid和MalScan等方法相比,平均 $F1$ 分数保持8.6%~23.1%的优势,证明本文方法具有更高的准确性和更慢的老化速度。

参考文献:

- [1] 范铭,刘桢,刘均,等.安卓恶意软件检测方法综述[J].中国科学:信息科学,2020,50(8):1148-1177.
FAN M, LIU T, LIU J, et al. Android malware detection: a survey[J]. Scientia Sinica (Informationis), 2020, 50(8): 1148-1177.
- [2] KIVVA A. Smartphone malware statistics, Q1[R]. 2023.
- [3] DHALARIA M, GANDOTRA E. Android malware detection techniques: a literature review[J]. Recent Patents on Engineering, 2021, 15(2): 225-245.
- [4] 王思远,张仰森,曾健荣,等. Android恶意软件检测方法综述[J]. 计算机应用与软件, 2021, 38(9): 1-9.
WANG S Y, ZHANG Y S, ZENG J R, et al. Overview of Android malware detection methods[J]. Computer Applications and Software, 2021, 38(9): 1-9.
- [5] 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.
- [6] LIU J, HUANG J Z, ZHOU Y, et al. From distributed machine learning to federated learning: a survey[J]. Knowledge and Information Systems, 2022, 64(4): 885-917.
- [7] DEHKORDY D T, RASOOLZADEGAN A. A new machine learning-based method for Android malware detection on imbalanced dataset[J]. Multimedia Tools and Applications, 2021, 80(16): 24533-24554.
- [8] KEYVANPOUR M R, BARANI-SHIRZAD M, HEYDARIAN F. Android malware detection applying feature selection techniques and machine learning[J]. Multimedia Tools and Applications, 2023, 82(6): 9517-9531.
- [9] DONG S, SHU L H, NIE S. Android malware detection method based on CNN and DNN hybrid mechanism[J]. IEEE Transactions on Industrial Informatics, 2024, 20(5): 7744-7753.
- [10] SHU L H, DONG S, SU H D, et al. Android malware detection methods based on convolutional neural network: a survey[J]. IEEE Transactions on Emerging Topics in Computational Intelligence, 2023, 7(5): 1330-1350.
- [11] NAEEM H, CHENG X C, ULLAH F, et al. A deep convolutional neural network stacked ensemble for malware threat classification in Internet of Things[J]. Journal of Circuits, Systems and Computers, 2022, 31(17): 2250302.
- [12] NAEEM H, DONG S, FALANA O J, et al. Development of a deep stacked ensemble with process based volatile memory forensics for platform independent malware detection and classification[J]. Expert Systems with Applications, 2023, 223: 119952.
- [13] ZHU H J, WANG L M, ZHONG S, et al. A hybrid deep network framework for Android malware detection[J]. IEEE Transactions on Knowledge and Data Engineering, 2022, 34(12): 5558-5570.
- [14] DAREM A A, GHALEB F A, AL-HASHMI A A, et al. An adaptive behavioral-based incremental batch learning malware variants detection model using concept drift detection and sequential deep learning[J]. IEEE Access, 2021, 9: 97180-97196.
- [15] FICCO M. Malware analysis by combining multiple detectors and observation windows[J]. IEEE Transactions on Computers, 2021, 71(6): 1276-1290.
- [16] LU H, JIN C J, HELU X H, et al. Research on intelligent detection of command level stack pollution for binary program analysis[J].

- Mobile Networks and Applications, 2021, 26(4): 1723-1732.
- [17] XU K, LI Y J, DENG R, et al. DroidEvolver: self-evolving Android malware detection system[C]//Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P). Piscataway: IEEE Press, 2019: 47-62.
- [18] KASPERSKY. Machine learning methods for malware detection[R]. 2019.
- [19] D'ANGELO G, FICCO M, PALMIERI F. Association rule-based malware classification using common subsequences of API calls[J]. Applied Soft Computing, 2021, 105: 107234.
- [20] ARICONI E, ONWUZURIKE L, ANDRIOTIS P, et al. MaMa-Droid: detecting Android malware by building Markov chains of behavioral models (extended version)[J]. ACM Transactions on Privacy and Security (TOPS), 2019, 22(2): 1-34.
- [21] YUMLEMBAM R, ISSAC B, YANG L Z, et al. Android malware classification and optimisation based on BM25 score of Android API[C]//Proceedings of the IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). Piscataway: IEEE Press, 2023: 1-6.
- [22] WU Y M, LI X D, ZOU D Q, et al. MalScan: fast market-wide mobile malware scanning by social-network centrality analysis[C]//Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE Press, 2019: 139-150.
- [23] ZHANG X H, ZHANG Y, ZHONG M, et al. Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware[C]//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2020: 757-770.
- [24] CUI L, CUI J C, JI Y D, et al. API2Vec: learning representations of API sequences for malware detection[C]//Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2023: 261-273.
- [25] 蒋煦, 张长胜, 戴大蒙, 等. Android平台恶意应用程序静态检测方法[J]. 计算机系统应用, 2016, 25(4): 1-7.
- JIANG X, ZHANG C S, DAI D M, et al. Static detection of Android malware[J]. Computer Systems & Applications, 2016, 25(4): 1-7.
- [26] 朱晋恺, 方兰婷, 季小文, 等. 基于行为特征和语义特征的多模态Android恶意软件检测方法[J]. 电子科技, 2024, 37(5): 71-78.
- ZHU J K, FANG L T, JI X W, et al. Multimodal Android malware detection method based on behavioral and semantic characteristics [J]. Electronic Science and Technology, 2024, 37(5): 71-78.
- [27] 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.
- [28] ZHANG Y X, YANG S M, XU L J, et al. A malware detection framework based on semantic information of behavioral features[J]. Applied Sciences, 2023, 13(22): 12528.
- [29] LIU Z, WANG R Y, 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.

[作者简介]



王练 (1976-), 女, 贵州遵义人, 博士, 大学重庆邮电教授, 主要研究方向为无线网络编码、无线网络安全。



青山科 (1999-), 男, 四川成都人, 重庆邮电大学硕士生, 主要研究方向为恶意软件检测、API语义。