软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(7):3209−3225 [doi: 10.13328/j.cnki.jos.007230] [CSTR: 32375.14.jos.007230]

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

基于 API 聚类和调用图优化的安卓恶意软件检测*

杨宏宇[1,2], 汪有为[2], 张 良[3], 胡 泽[1], 姜来为[1], 成 翔[4]

[1](中国民航大学 安全科学与工程学院, 天津 300300)

[2](中国民航大学 计算机科学与技术学院, 天津 300300)

[3](School of Information, The University of Arizona, Arizona 85721, USA)

[4](扬州大学 信息工程学院, 江苏 扬州 225127)

通信作者: 杨宏宇, E-mail: hyyang@cauc.edu.cn; 张良, E-mail: liangzh@arizona.edu;
胡泽, E-mail: zhu@cauc.edu.cn

摘 要: 安卓操作系统和恶意软件的持续进化导致现有检测方法的性能随时间大幅下降. 提出一种基于 API 聚类和调用图优化的安卓恶意软件检测方法 DroidSA (droid slow aging). 首先, 在恶意软件检测之前进行 API 聚类, 生成代表 API 功能的聚类中心. 通过设计 API 句子概括 API 的名称、权限等重要特征并使用自然语言处理工具对 API 句子的语义信息进行挖掘, 获得更全面反映 API 语义相似性的嵌入向量, 使聚类结果更为准确. 然后, 为了确保提取到更能准确反映软件行为逻辑的 API 上下文信息, 采用调用图优化方法对从待检测软件中提取的函数调用图进行优化并得到优化后的调用图, 在删除图中难以识别的未知方法的同时保留 API 节点之间的连接性. 为了提高对安卓框架和恶意软件变化的适应性, DroidSA 从优化后的调用图中提取函数调用对, 将调用对中的 API 抽象为 API 聚类时获得的聚类中心. 最后, 使用独热编码生成特征向量, 并从随机森林、支持向量机和 K 近邻算法中选择表现最好的分类器进行恶意软件检测. 实验结果表明, DroidSA 的恶意软件检测平均 $F1$ 值为 96.7%; 在消除时间偏差的实验设置下, 经 2012–2013 年的软件样本集合训练后, DroidSA 对 2014–2018 年的恶意软件样本的检测平均 $F1$ 值达到 82.6%. 与经典检测方法 MaMaDroid 和 MalScan 等相比, DroidSA 始终能将各项检测指标稳定地维持在高水平且受到时间变化的影响较小, 能有效检测进化后的恶意软件.

关键词: 恶意软件检测; API 语义; 模型老化; 函数调用图; 机器学习

中图法分类号: TP311

中文引用格式: 杨宏宇, 汪有为, 张良, 胡泽, 姜来为, 成翔. 基于API聚类和调用图优化的安卓恶意软件检测. 软件学报, 2025, 36(7): 3209–3225. http://www.jos.org.cn/1000-9825/7230.htm

英文引用格式: Yang HY, Wang YW, Zhang L, Hu Z, Jiang LW, Cheng X. Android Malware Detection Based on API Clustering and Call Graph Optimization. Ruan Jian Xue Bao/Journal of Software, 2025, 36(7): 3209–3225 (in Chinese). http://www.jos.org.cn/1000-9825/7230.htm

Android Malware Detection Based on API Clustering and Call Graph Optimization

YANG Hong-Yu[1,2], WANG You-Wei[2], ZHANG Liang[3], HU Ze[1], JIANG Lai-Wei[1], CHENG Xiang[4]

[1](School of Safety Science and Engineering, Civil Aviation University of China, Tianjin 300300, China)

[2](School of Computer Science and Technology, Civil Aviation University of China, Tianjin 300300, China)

[3](School of Information, The University of Arizona, Arizona 85721, USA)

[4](School of Information Engineering, Yangzhou University, Yangzhou 225127, China)

Abstract: As both Android frameworks and malware continue to evolve, the performance of existing malware classifiers degrades significantly over time. This study proposes droid slow aging (DroidSA), a method for Android malware detection based on API clustering

* 基金项目: 国家自然科学基金 (62201576, U1833107); 江苏省基础研究计划 (BK20230558)
收稿时间: 2023-07-15; 修改时间: 2023-11-03; 采用时间: 2024-05-23; jos 在线出版时间: 2024-08-21
CNKI 网络首发时间: 2024-08-22

and call graph optimization. Firstly, API clustering is performed before malware detection to generate cluster centers that reflect API functionality. To make clustering results more accurate, this study obtains embeddings fully reflecting the semantic similarity of APIs by designing API sentences to summarize vital features such as API names and permissions and using NLP tools to mine the semantic information of API sentences. Then, call graphs are extracted from apps and optimized by removing unknown methods while preserving the connectivity among API nodes. Call graph optimization enables detection methods to extract more robust contextual information of APIs which reflects the mode of app behavior. DroidSA extracts pairs of function calls from the optimized call graphs and abstracts the APIs in the pairs into cluster centers obtained in API clustering to better adjust to the changes in Android frameworks and malware. Finally, one-hot encoding is used to generate feature vectors, and the best-performing classifier is selected from random forests, support vector machines, and the k-nearest neighbors algorithm for malware detection. Experimental results demonstrate that DroidSA achieves an average *F*1-Measure of 96.7% for malware detection. Under the experimental setup where temporal bias is eliminated, DroidSA trained with apps from 2012 to 2013 achieves an average *F*1-Measure of 82.6% when detecting malware developed from 2014 to 2018. Compared with the state-of-the-art detection methods MaMaDroid and MalScan, DroidSA stably maintains high detection metrics with minimal impact from temporal changes and effectively detects evolved malware.

Key words: malware detection; API semantics; model aging; function call graph; machine learning

安卓平台广泛普及的同时伴生了恶意软件的快速增长, 虽然机器学习方法在恶意软件检测领域取得了出色成绩[1−4], 但持续进化的恶意软件仍给检测工作造成严重的困扰[5,6]. 恶意软件开发者通过改变软件恶意行为的实现方式以逃避现有的检测方法[7], 具体体现在新开发的恶意软件中存在大量旧恶意软件中未出现的 API, 这导致新开发的恶意软件与旧恶意软件在特征分布上有着显著区别.

根据特征类型不同, 可将现有安卓恶意软件检测方法划分为基于 API 频率信息的检测方法和基于 API 上下文信息的检测方法. 基于 API 频率信息的检测方法[8−10]漏报率高且鲁棒性差, 检测模型易受到恶意攻击. 基于 API 上下文信息的检测方法[11,12]能更准确识别软件的恶意行为, 但由于缺乏对函数调用图中大量未知函数的合适处理方式, 难以有效提取 API 的上下文信息. 此外, 上述两类检测方法均未考虑软件中 API 的频繁变化, 导致使用旧软件训练的分类器难以检测进化后的恶意软件 (观念迁移样本), 性能也随时间不断下降, 这种现象被称为模型老化.

API 聚类将功能相似的 API 归为一类, 在恶意软件检测阶段使用 API 聚类中心替换特征向量中的 API 能使分类器有效识别训练时未出现的 API, 使检测方法适应安卓框架和恶意软件中 API 的变化. 现有 API 聚类方法从包名、参数、返回值等特征中提取 API 语义信息, 忽略 API 的方法名和权限等关键特征, 难以全面提取 API 语义信息, 导致聚类结果对分类器的性能提升有限. Mariconti 等人[13]提出一种构建恶意行为马尔可夫链的安卓恶意软件检测方法 MaMaDroid, 根据包名对 API 进行聚类, 使检测方法能够识别现有包中出现的新 API. 但相同包中的 API 不一定都具有相同的功能, MaMaDroid 也无法识别新添加包中的 API. Pendlebury 等人[5]通过实验发现: 在消除时间偏差的实验设置下, MaMaDroid 的 *F*1 值仅 3 个月后就从 90% 以上下降到 58%.

针对上述问题, 本文提出一种基于 API 聚类和调用图优化的安卓恶意软件检测方法 DroidSA (droid slow aging). 首先, 在恶意软件检测之前进行 API 聚类, 生成概括 API 特征的 API 句子, 使用自然语言处理工具挖掘 API 句子的语义信息, 生成嵌入向量并进行聚类, 获得代表 API 功能的聚类中心. 然后, 从软件中提取函数调用图并对调用图进行优化, 删除图中难以识别的未知函数, 同时保留 API 节点之间的连接性. 其次, 从优化后的调用图中提取函数调用对并将调用对中 API 抽象为 API 聚类中心, 将其他函数抽象为包. 最后, 使用独热编码生成特征向量并使用机器学习分类器进行恶意软件检测.

本文的主要贡献如下.

(1) 提出一种基于语义距离的 API 聚类方法. 通过设计 API 句子概括 API 的特征, API 句子不仅概括 API 的方法名、权限等重要特征, 还能将特征数量不一致的 API 统一映射到固定大小的特征向量. 使用自然语言处理工具对 API 句子中蕴含的丰富语义信息进行充分挖掘, 进而更全面地提取 API 语义信息, 使聚类结果更为准确.

(2) 提出一种调用图优化方法. 调用图优化方法删除函数调用图中难以识别的未知函数节点并保留 API 节点之间的连接性. 调用图中任意两个 API 节点 (API_X 和 API_Y) 之间若存在一条全部由未知函数节点组成的路径, 则在优化后的调用图中 API_X 直接调用 API_Y. 调用图优化使检测方法能提取到更多更准确反映软件行为逻辑的 API 上下文信息.

(3) 提出一种安卓恶意软件检测方法, 通过将特征向量中的 API 抽象为 API 聚类中心, 使检测方法能适应安卓框架和恶意软件中 API 的频繁变化. 与现有方法相比, 在不需要任何重训练的情况下, 检测方法的模型老化速度更慢, 能有效检测进化后的恶意软件.

1 相关工作

本文的主要研究内容是进化后恶意软件的检测方法, 当前恶意软件检测领域应对模型老化的方法主要分为 4 类: 基于 API 聚类的检测方法、基于重训练的检测方法、基于软件关系图的检测方法和基于异常样本识别的检测方法.

1.1 基于 API 聚类的检测方法

API 聚类将功能相似的 API 归为一类, 能使检测方法适应软件中 API 的频繁变化. 本文认为 API 聚类是解决模型老化问题的众多方法中最有效和最直接的.

Mariconti 等人[13]提出一种构建恶意行为马尔可夫链的安卓恶意软件检测方法 MaMaDroid, 使用包名对 API 进行聚类, 通过将调用图中的 API 调用对抽象为包调用对, MaMaDroid 能够识别现有包中出现的新 API, 但相同包中的 API 不一定都具有相同的功能, MaMaDroid 也无法识别新添加包中的 API, 这导致 MaMaDroid 难以有效检测进化后的恶意软件.

Lei 等人[14]提出一种事件敏感的安卓恶意软件检测方法 EveDroid, 将 API 拆分成单词, 包括包名, 类名, API 方法名, 返回值和参数; 然后使用 doc2vec[15]进行编码, 生成 API 的嵌入向量; 最后使用 k-means 算法进行聚类. 该方法使用自然语言处理工具对 API 的特征进行挖掘, 但在提取 API 语义信息时未考虑 API 的权限信息. 权限在安卓生态系统中具有极其重要的作用[16], 因此 EveDroid 难以全面提取 API 的语义信息.

Zhang 等人[7]提出一种使用 API 聚类对现有安卓恶意软件分类器的增强方法 APIGraph, 首先从安卓官方文档中提取 API 特征; 其次构建实体关系图; 然后通过 TransE 算法[17]生成 API 的嵌入向量; 最后使用 k-means 算法进行 API 聚类. 在强化现有分类器时, APIGraph 将现有检测方法特征向量中的 API 替换为 API 聚类中心. 该论文认为使用相似参数、相似权限、拥有相似返回值的 API 具有相似的功能, 但在 API 语义嵌入时未考虑 API 方法名中蕴含的丰富语义信息, 导致难以全面提取 API 的语义信息.

Xu 等人[18]提出一种基于 API 聚类的安卓恶意软件检测方法 SDAC, 通过双层神经网络捕获 API 调用和其上下文之间的映射关系, 将隐藏层作为反应 API 语义的嵌入向量用于聚类. 但该方法使用训练和测试软件样本同时训练双层神经网络, 存在数据窥探[19]行为. 该方法在实际应用场景中无法识别软件中出现的新 API.

1.2 基于重训练的检测方法

重训练是使用观念迁移样本重新对分类器进行训练. 根据标签来源的不同, 可将重训练分为真实标签重训练和伪标签重训练. 真实标签来源于恶意软件检测领域的专家进行的人工标注; 伪标签来源于分类器的检测结果, 这些检测结果往往具有很高的置信度.

Narayanan 等人[20]提出一种自适应且可扩展的安卓恶意软件检测方法 DroidOL, 从函数调用图中提取软件的敏感行为特征, 在检测过程中使用误分类样本和其真实标签更新分类器, 使分类器能够适应安卓软件的变化.

Xu 等人[21]提出一种基于在线学习的安卓恶意软件检测方法 DroidEvolver, 该论文认为使用不同优化器训练的分类器具有不同的老化速度, 因此在训练阶段构建一个模型池, 模型池中的分类器使用不同的优化器进行训练. 在检测阶段通过置信度判断老化模型, 使用伪标签进行轻量级的在线学习, 更新老化模型.

基于重训练的检测方法总体存在以下不足: 真实标签重训练需要大量且优质的观念迁移样本, 这些样本往往难以及时获得, 同时还需要花费大量的人力资源对观念迁移样本进行人工标注; 伪标签重训练不需要大量的人力资源对样本进行人工标注, 但存在两个问题: 一是伪标签一旦出现错误, 会导致分类器的性能出现断崖式下滑; 二是恶意攻击者会针对检测模型的伪标签重训练更新机制设计特定样本实施恶意攻击.

1.3 基于软件关系图的检测方法

软件关系图是指构建一个以软件为主要节点的实体关系图反映软件之间的相似性, 在检测过程中使用待检测

样本和软件关系图中节点的相似性辅助分类器做出判断.

Gu 等人[22]提出一种在安卓生态环境下缓解模型老化的方法 GraphEvolveDroid, 通过构建软件关系图反映软件之间的进化关系, 辅助分类器进行检测.

Hei 等人[23]提出一种基于异构图注意力网络的安卓恶意软件检方法 Hawk, 在训练阶段首先构建软件关系图模拟软件之间的相似性, 然后使用图神经网络学习软件之间的相似性, 生成特征向量并训练分类器. 在检测阶段设计一种增量聚合方法 MsGAT++, 能够快速生成异构图外新样本的特征向量并进行恶意软件检测, 无需更新软件关系图并重新训练图神经网络.

基于软件关系图的检测方法总体存在以下不足: 首先, 代码重用是软件开发过程中的重要组成部分, 良性软件开发者也会使用恶意软件中的部分代码, 这给基于软件关系图的检测方法带来严重困难; 其次, 软件之间的相似性不代表软件的行为逻辑, 这导致基于软件关系图的检测方法难以学习恶意软件的行为特征.

1.4 基于异常样本识别的检测方法

异常样本识别是在分类器决策前识别观念迁移样本. 分类器难以对观念样本做出准确判断, 因此将这些样本从分类器的待检测软件中排除, 交给恶意软件领域的专家进行检测.

Yuan 等人[24]提出一种基于双头神经网络的观念迁移样本识别方法, 使用神经网络检测软件类别, 在神经网络的输出端设置两个并行的输出层, 分别输出预测软件不同类别的概率. 通过输出层预测概率之间的差值识别观念迁移样本.

Karbab 等人[25]提出一种自适应的安卓恶意软件检测方法 PetaDroid, 分类器对观念迁移样本做出决策时通常具有低置信度, 体现在预测软件属于良性的概率和属于恶意的概率十分接近, 通过在检测恶意软件时过滤低置信度样本, 大幅提高检测方法的有效性.

异常样本识别能大幅提高检测方法的有效性, 但不具备检测进化后恶意软件的能力.

2 DroidSA 的总体设计

为有效检测进化后的恶意软件, 本文提出一种基于 API 聚类和调用图优化的安卓恶意软件检测方法 DroidSA. DroidSA 的框架如图 1 所示, DroidSA 由基于语义距离的 API 聚类和恶意软件检测两部分组成.

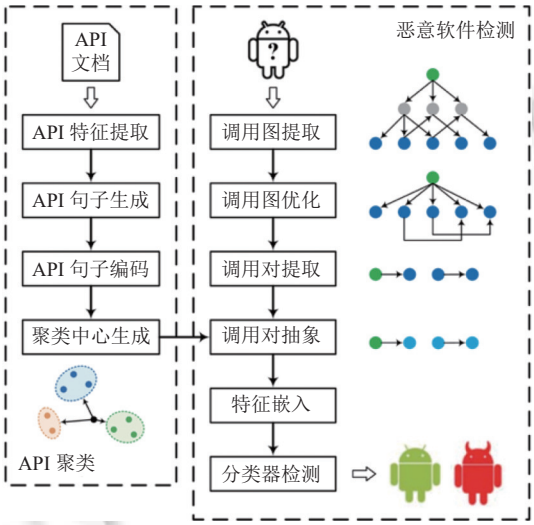


图 1 DroidSA 框架图

基于语义距离的 API 聚类的核心功能是将功能相似的 API 分配到相同聚类中心. 经过 API 聚类处理, 生成代表 API 功能的聚类中心. 该部分包括 4 个过程.

(1) API 特征提取: 从 API 文档中提取 API 特征, 包括类名、方法名、异常、权限等特征.

(2) API 句子生成: 对 API 的特征信息进行概括, 根据设计的规则生成每个 API 对应的 API 句子.

(3) API 句子编码: 使用 BERT[26]对 API 句子进行编码, 获得反映 API 语义相似性的嵌入向量.

(4) 聚类中心生成: 使用 k-means[27]算法对 API 嵌入向量进行聚类, 生成代表 API 功能的聚类中心.

在恶意软件检测时, API 聚类中心用于替换特征向量中的 API.

恶意软件检测部分包括 6 个过程.

(1) 调用图提取: 使用静态分析工具从待检测软件中提取函数调用图.

(2) 调用图优化: 对函数调用图进行优化, 删除图中无法识别的未知函数节点并保留 API 节点之间的连接性, 生成优化后的函数调用图.

(3) 调用对提取: 从优化后的调用图中提取函数调用对.

(4) 调用对抽象: 将调用对中的 API 抽象为 API 聚类部分获得的 API 聚类中心, 将其他函数抽象为包, 生成聚类中心调用对.

(5) 特征嵌入: 使用独热编码对聚类中心调用对进行特征嵌入; 使用主成分分析将高维特征向量映射到低维空间, 生成待检测软件的特征向量.

(6) 分类器检测: 使用机器学习分类器进行恶意软件检测, 检测样本属于良性软件还是恶意软件.

3 基于语义距离的 API 聚类

基于 API 聚类的检测方法能适应软件中 API 的频繁变化, 但现有聚类方法在提取 API 语义信息时忽略方法名和权限等关键特征, 导致难以全面提取 API 语义信息. API 的方法名蕴含丰富语义信息, 概括 API 的作用. 例如方法名为 getDeviceID 或 setWifiEnabled 的 API, 其方法名充分反应 API 的功能. API 的权限在安卓操作系统中极为重要[16]. 使用自然语言处理工具对方法名和权限等关键特征中蕴含的语义信息进行充分挖掘, 能更全面的提取 API 语义信息, 使聚类结果更为准确.

本文提出的基于语义距离的 API 聚类方法包括 4 个过程: 特征提取、API 句子生成、API 句子编码和聚中心生成.

3.1 API 特征提取

特征提取是从安卓官方文档中提取 API 的特征, 包括包、类名、方法名、参数、权限、异常和返回值 7 种特征. 在本文研究中, 使用 APIGraph[7,28]提供的安卓官方文档和特征提取方法.

表 1 展示 API android.telephony.TelephonyManager.getDeviceId 的特征提取结果. 为减少一些频繁出现的特征对 API 语义信息嵌入造成影响, 在特征提取过程中忽略 int、boolean、String、float 等特征, 表 1 展示的 API 使用 int 类型变量作为参数, 返回值为 String 类型, 这两类特征在提取时被忽略.

表 1 API 特征提取示例

特征	提取结果
权限	android.permission.READ_PHONE_STATE
参数	无
异常	无
返回值	无
包	android.telephony
类名	Telephony $ Manager
方法名	get $ Device $ ID

本文在 APIGraph 特征提取方法的基础上, 对类名和方法名两类特征进行修饰: 利用方法名和类名的驼峰命名格式 (除首个单词外, 其余单词的首字母均为大写, 例如 setWifiEnabled、sendTextMessage 等) 将其拆分为单词的

组合. 表 1 中 API 的方法名为 getDeviceID, 被拆分为 get、Device 和 ID 这 3 个单词, 使用‘$’符号进行分割. 此外, 在类名和方法名中有一些单词全部由大写字母组成, 这些单词是缩略符, 例如 SQL、URL、ID 等, 需要对缩略符进行额外识别.

3.2 API 句子生成

完成特征提取后, 需要将 API 的特征转换为反应其语义相似性的编码. 然而 API 的特征数量不一致问题给语义信息提取造成困难. 例如一些 API 的方法名可以拆成 5 个单词, 而另一些 API 的方法名只能拆成 3 个单词; 一些 API 执行时需要申请权限, 而另一些 API 在执行时不需要权限. 因此需要设计一种编码方式, 即能全面概括 API 特征的语义信息, 又能将特征数量不一致的 API 全部映射到固定大小的特征向量. 为解决上述问题, 本文设计一种 API 句子的生成规则, 根据该规则生成 API 句子对 API 的特征进行概括.

根据特征数量不同, 将 API 的特征划分为共有特征和差异特征. 共有特征包括方法名、类名和包, 每个 API 都有且仅有一个方法名, 类名和包.

共有特征对应的 API 句子生成规则为:

method *method_name* from class *class_name* from package *package*

差异特征包括权限、异常、参数和返回值. 每类特征分别对应两种情况: 特征数量为 0 和特征数量至少为 1. 以权限特征为例, 分别对应 API 执行时不需要权限和至少需要一种权限. 差异特征对应的 API 句子生成规则如表 2 所示.

表 2 差异部分 API 句子生成规则

特征类别	特征数量为0	特征数量至少为1
Permission	use none permission	use *peremission*_1 *permission*_2, …
Parameter	use none parameter	use parameter *parameter*_1 *parameter*_2, …
Exception	throw none exception	throw *excepthion*_1 *exception*_2, …
Return	return none	return *return_value*

API 句子生成的伪代码如算法 1 所示. 输入为 API_X 的特征, 输出为 API_X 对应的 API 句子 *S*. 第 1 行生成共有特征对应的部分 API 句子; 第 2–5 行调用算法 2, 生成差异特征对应的部分 API 句子; 第 6 行将 API 句子中所有的大写字母替换为小写; 第 7 行将 API 句子中所有的‘$’、‘_’和‘.’替换为空格.

算法 1. API 句子生成算法.

输入: API_X 的特征, 包括: *Method_name*: 方法名, *Class_name*: 类名, *Package*: 包, *Permission*: 权限, *Exception*: 异常, *Parameter*: 参数, *Return*: 返回值;
输出: *S*: API_X 对应的 API 句子.

1. *S* = “method” + *Method_name* + “from class” + *Class_name* + “from package” + *Package*
2. *S* = *S* + *Algorithm*2(*Permission*, “use none permission”, “use permission”)
3. *S* = *S* + *Algorithm*2(*Exception*, “throw none exception”, “throw exception”)
4. *S* = *S* + *Algorithm*2(*Parameter*, “use none parameter”, “use parameter”)
5. *S* = *S* + *Algorithm*2(*Return*, “return none”, “return”)
6. replace all uppercase in *S* by lowercase
7. replace all ‘$’, ‘_’, and ‘.’ in *S* By ‘ ’
8. **return** *S*

单个差异特征对应的部分 API 句子生成算法如算法 2 所示. 算法 2 在算法 1 的第 2–5 行被调用, 输入是某一类差异特征 *Features* 和对应的 API 句子生成规则 S_0 和 S_1, 输出是单个差异特征生成的部分 API 句子 S_{part}. 算法 2

分别考虑差异特征数量为 0 和差异特征数量至少为 1 这两种情况.

算法 2. 单个差异特征对应的部分 API 句子生成算法.

输入: *Features*: 某类差异特征, S_0: 特征数量为 0 时句首部分, S_1: 特征数量至少为 1 时句首部分;
输出: S_{part}: 单个差异特征生成的部分 API 句子.

1. **if** *Features* is none **then**:
2. $S_{part} = S_0$
3. **else**
4. $S_{part} = S_1$
5. **for** each feature *f* in *Features* **do**:
6. $S_{part} = S_{part} + ' ' + f$
7. **end for**
8. **end if**
9. **return** S_{part}

将表 1 中展示的 API 和其特征输入算法 1, 得到最终生成的 API 句子如下:

method get device id from class telephony manager from package android telephony use android permission read phone state throw none exception use none parameter return none

3.3 API 句子编码

生成 API 句子后, 使用在 BooksCorpus 和 Wikipedia 等语料库上预训练好的 BERT 模型对不同长度的 API 句子进行编码, 取编码结果中第 0 个词向量 CLS 对应的 768 维特征向量作为 API 的语义嵌入. BERT 由双向 Transformer 子结构堆叠而成, 是一个经典的预处理模型[26]. CLS 是 classification 的缩写, 对应的特征向量概括整句句子的语义信息, 常被用于文本情感分类等下游任务.

通过生成 API 句子并使用 BERT 对 API 句子进行编码, 特征数量不一致的 API 被全部映射到固定大小的嵌入向量. 本文在提取 API 语义信息时既考虑 API 名称和权限中蕴含的丰富语义信息, 也考虑参数、异常、返回值等不可或缺的重要特征.

在设计 API 句子的生成规则时, 意思相同但表达方式不同的规则会对分类器的衰减速度造成影响. 例如将规则 use none permission 改成 do not use permission 或 use no permission 会削弱分类器检测进化后恶意软件的能力. 与 not 或 no 相比, BERT 能从包含单词 none 的 API 句子中提取更多信息.

3.4 聚类中心生成

BERT 对 API 句子进行编码后, 生成 768 维的特征向量, 该特征向量维度过大, 对聚类过程中的距离计算造成较大压力. 因此, 首先使用主成分分析[29]进行降维, 取方差比为 95%, 得到降为后 66 维度的特征向量; 然后使用 Elbow 方法[27]确定聚类中心个数, 根据 Elbow 方法, 取 2 000 为聚类中心个数; 最后使用 k-means 算法进行聚类. 完成聚类后, 功能相似的 API 被分配到相同聚类中心.

当安卓官方对 API 文档进行更新时, 例如添加新 API 或修改已有 API, 需要同步更新 API 的聚类结果. API 聚类中心的更新过程如图 2 所示.

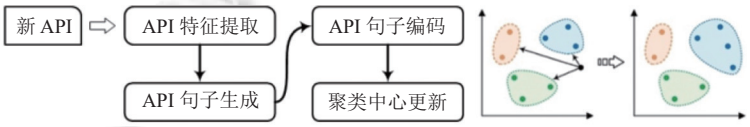


图 2 API 聚类中心更新

首先提取新 API 的特征并生成 API 句子; 然后使用 BERT 对 API 句子进行编码并使用主成分分析进行降维, 获得新 API 的嵌入向量; 在聚类过程中只需要将新 API 分配到距离最近的聚类中心并更新该聚类中心的坐标. 而 APIGraph 需要更新实体关系图并使用 TransE 算法重新训练整个实体关系图, 生成所有 API 的特征向量, 再重新进行聚类. 以 API level 28 更新至 API level 29 为例, API level 28 中有 58 291 个 API, API level 29 新增了 834 个 API. 在更新 API 的聚类结果时, 本文提出的 API 聚类方法只需要处理新增的 834 个 API. 而 APIGraph 不仅需要处理 834 个新 API, 还需要重新处理 API level 28 中已有的 58 219 个 API. 因此, 本文提出的 API 聚类方法的更新代价远小于 APIGraph.

4 调用图优化

4.1 问题提出

API 全称为应用编程接口, 是实现系统功能 (例如: 网络通信、蓝牙连接等) 的一类特殊函数. 除 API 外, 软件中还存在着大量未知函数和少量其他函数, 根据代码保护机制[30]可将未知函数分为自定义函数 (例如: com.xiaomi.mipush.sdk.PushMessageHandler) 和模糊函数 (例如: com.a.b.c.e123); 其他函数主要是类的初始化函数 (例如: java.io.FileOutputStream.init), 其所属的类出现在安卓官方文档中.

函数调用图也称控制流图, 用于表示软件的函数调用关系. 未知函数、其他函数和 API 在调用图中都是一个节点, 未知函数节点在调用图节点中占大部分.

现有检测方法对函数调用图中未知函数节点的处理方式分为统一抽象和全部保留.

统一抽象是将未知函数节点统一抽象为一个未知节点, 这种方式严重降低函数调用图的结构信息. 由于安卓特殊的事件触发机制, 通过静态分析提取的函数调用图不包含函数的执行顺序信息. 对所有未知节点进行统一抽象后, 大部分 API 的前驱和后继都是未知节点, 这导致函数调用图中 API 的上下文信息被严重降低.

全部保留是将函数调用图中的未知函数节点全部保留, 这种方式产生大量冗余信息. API 数量有限且功能明确, 因此常被用于描述软件的行为特征. 当前恶意软件检测方法主要基于 API 的上下文信息, 而未知函数的名称和功能由软件开发者定义和修改, 数量庞大且种类繁多, 难以有效提取软件信息. 将调用图中的未知函数全部保留会严重增加后续检测工作的计算开销, 影响检测方法提取 API 的上下文信息.

API 上下文信息能更好地反映软件的行为逻辑, 即恶意软件检测应基于函数调用图中以 API 节点为中心的局部信息. 然而, 在当前安卓生态环境下, 未知函数节点在调用图节点中的占比较大, 且现有检测方法对调用图中未知函数节点的处理方式造成难以提取 API 上下文信息.

4.2 调用图优化算法

为解决调用图中大量未知函数造成的 API 上下文信息提取困难的问题, 本文提出一种调用图优化方法. 未知函数本身不包含任何信息, 或者说未知函数中包含的少量信息极难提取, 但未知函数前驱和后继 API 节点之间的调用关系反映软件的行为逻辑, 这有助于分类器识别恶意软件.

调用图优化方法在删除图中所有未知函数节点的同时, 保留 API 节点之间的连接性. 函数调用图中任意两个 API 节点 (API_X 和 API_Y) 之间若存在一条全部由未知函数节点组成的路径, 即 API_X 通过调用若干未知函数调用 API_Y, 则在优化后的调用图中 API_X 直接调用 API_Y.

调用图优化算法的伪代码如算法 3 所示. 输入为函数调用图和安卓官方文档, 输出为优化过后的调用图.

- 算法 3 的第 2–7 行: 过滤函数调用图中的关键函数节点, 包括: 入口函数、API 和其他函数, 这些节点之间的调用关系反映软件的行为逻辑, 应予以保留.
- 算法 3 的第 8/9 行: 获得待删除节点的前驱节点集合和后继节点集合.
- 算法 3 的第 10–14 行: 保持待删除节点的前驱和后继节点之间的连接性, 对待删除节点的前驱和后继节点进行遍历, 在调用图中添加每个前驱节点和每个后继节点组成的有向边.
- 算法 3 的第 15 行: 从函数调用图中删除未知函数节点.

算法 3. 调用图优化算法.

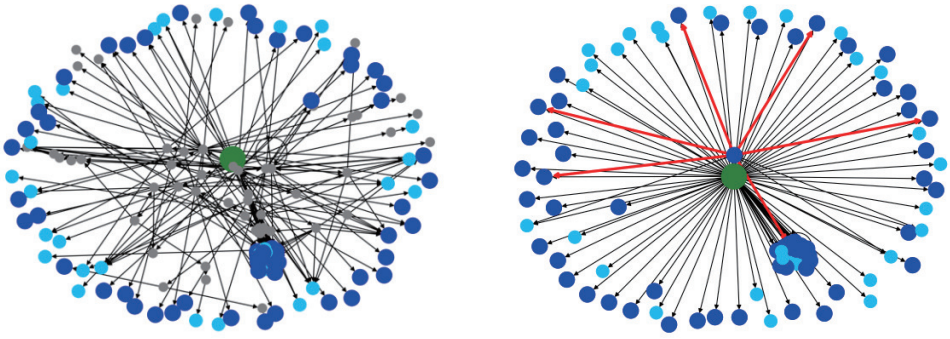
输入: $CG(V, E)$: 函数调用图, *AODs*: 安卓官方文档;
输出: $CG_{opt}(V_{opt}, E_{opt})$: 优化后的调用图.

1. **for** each node *n* in *V* **do**:
2. **if** *n* is *entry function* **then**:
3. **continue**
4. **end if**
5. **if** *n* in *AODs* or *n.class* in *AODs* **then**:
6. **continue**
7. **end if**
8. collect predecessor nodes of *n* as *callers*
9. collect successor nodes of *n* as *callees*
10. **for** each node *start* in *callers* **do**:
11. **for** each node *end* in *callees* **do**:
12. add edge (*start*, *end*) to *CG*
13. **end for**
14. **end for**
15. remove node *n* from *CG*
16. **end for**
17. **return** *CG*

通过调用图优化, 在保留图中关键节点整体连接性的同时, 删除图中无法识别的未知函数节点. 优化后的调用图只存在 3 类函数: 入口函数、API 和其他函数, 这 3 类函数之间的调用关系能更准确地反映软件的行为逻辑, 可用于后续 API 调用的上下文信息提取.

4.3 调用图优化示例

良性软件 app.eazi.ease126(md5:2149914d3284d4d0561f9d80dc1a d186) 的函数调用图和优化后的调用图如图 3 所示. 图 3 中, 绿色节点为入口函数, 灰色节点为未知函数, 深蓝色节点为 API, 淡蓝色节点为其他函数. 函数调用图中, 所有的边用黑色表示, 优化后的调用图中新增的 API 调用对用红色边表示. 这些调用对在函数调用图中至少存在一条全部由未知函数节点组成的路径.

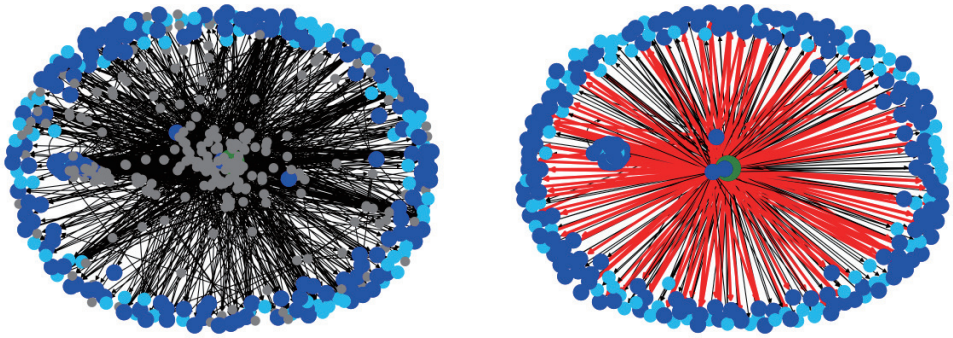


(a) 函数调用图 (b) 优化后的调用图

图 3 良性软件 app.eazi.ease126 的调用图优化

如图 3(a) 所示, 未知函数节点在函数调用图中占大部分, 严重影响了 API 上下文信息的提取. 调用图优化删除了图中所有难以识别的未知函数节点, 同时保留 API 节点之间的连接性, 使检测方法能提取到更多更准确反映软件行为逻辑的 API 上下文信息.

恶意软件 de.android_telefonie.super.appmanager(md5:79a58748d8e346af229c60a6ed529ab8) 的函数调用图和优化后的调用图如图 4 所示.



(a) 函数调用图 (b) 优化后的调用图

图 4 恶意软件 de.android_telefonie.super.appmanager 的调用图优化

函数调用图中大量 API 节点的前驱节点是未知函数节点, 即软件的大部分 API 由未知函数调用. 这说明软件中实现系统功能的敏感 API, 其局部上下文信息大多都是检测过程中难以识别的未知方法, 这给提取软件的行为特征造成严重困难. 本文提出的调用图优化方法在保留图中 API 节点连接性的同时, 删除图中所有的未知函数节点. 优化后的调用图增加了许多反映软件行为逻辑的 API 调用对, 这些新增的 API 调用对大多与线程活动相关, 而恶意软件常常伪装成良性软件, 通过多线程活动实施恶意行为. 这说明调用图优化能使检测方法提取到更多用于区分软件类别的关键特征.

5 恶意软件检测

本文提出一种能有效检测观念迁移样本的安卓恶意软件检测方法. 该方法属于静态检测方法, 使用软件的 API 上下文信息作为特征. 恶意软件检测过程如图 5 所示.

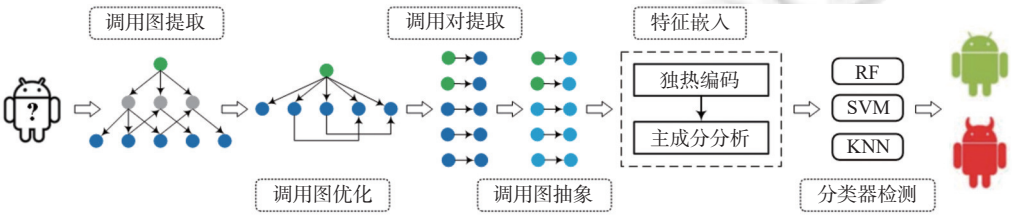


图 5 恶意软件检测过程

恶意软件检测的具体过程如下.

(1) 调用图提取: 使用静态分析工具 FlowDroid[31]从软件样本集的 APK 文件中提取函数调用图.

(2) 调用图优化: 对函数调用图进行优化, 删除图中无法识别的未知函数节点并保留 API 节点之间的连接性, 生成优化后的函数调用图.

(3) 调用对提取: 从优化后的调用图中提取函数调用对, 调用对包括调用函数和被调用函数.

(4) 调用对抽象: 对函数调用对中的节点进行抽象, 将调用对中的 API 抽象为 API 聚类部分获得的 API 聚类

中心, 将其他函数抽象为包, 生成聚类中心调用对. 调用对抽象将频繁变化的 API 抽象为 API 聚类中心, 使检测方法适应安卓框架和恶意软件的变化.

(5) 特征嵌入: 首先使用独热编码对软件样本集的聚类中心调用对进行特征嵌入; 然后使用主成分分析[29]将高维特征向量线性映射到低维空间, 生成软件样本集的特征向量集合.

(6) 分类器检测: 分类器检测包括样本集划分、训练和检测这 3 部分. 首先使用十折交叉验证法将特征向量集合划分为训练集和测试集; 然后使用训练集训练 K 近邻 (k nearest neighbor, KNN)、支持向量机 (support vector machine, SVM)[32]和随机森林 (random forest, RF)[33]这 3 种分类算法; 最后将测试集输入训练好的分类器进行恶意软件检测, 选择检测平均 *F*1 值最高的分类算法作为最终检测结果.

6 实验及结果分析

在实验中, 从恶意软件检测的有效性方面对本文提出的 DroidSA 进行验证评测. 在有效性测试实验中, 首先验证 DroidSA 能否实现恶意软件检测; 然后验证 DroidSA 能否有效检测观念迁移样本; 最后分别评估 API 聚类和调用图优化对 DroidSA 检测性能的影响.

6.1 实验设置

实验中使用的软件样本集如表 3 所示, 该软件样本集包括 42 450 个恶意软件样本和 42 154 个良性软件样本, 时间跨度为 7 年. 恶意软件来源于 AndroZoo[34]、VirusTotal[35]和 AMD 软件样本集[36], 在 VirusTotal 中至少被 15 个反病毒引擎标记为恶意. 良性软件来源于 Google Play 商店[37], 通过 AndroZoo 下载, 被 VirusTotal 中所有反病毒引擎标记为良性.

表 3 实验软件样本集

年份	良性软件 (B)	恶意软件 (M)	M + B	M / (M + B) (%)	平均大小 (MB)
2012	4418	4624	9042	51.1	2.23
2013	6488	6519	13007	50.1	2.65
2014	6364	6409	12773	50.2	4.58
2015	6506	6494	13000	50.0	5.08
2016	6474	6449	12923	49.9	5.09
2017	5348	5388	10736	50.2	7.79
2018	6556	6567	13123	50.0	6.95
总计	42154	42450	84604	50.2	4.96

在训练和测试过程中, 采用十折交叉验证法对 DroidSA 进行评估. 具体方法为: 将软件样本集分为 10 份, 其中 9 份用于训练模型, 剩下一份用作测试进行模型评估, 重复 10 次直至每一份软件样本都经过测试评估. 最后, 对 10 次的测试结果取平均值, 得到评估结果. 采用精确度 (precision, Pre)、召回率 (recall, Rec)、*F*1 值 (*F*1-Measure) 和准确率 (accuracy, Acc) 指标衡量 DroidSA 的检测效果.

6.2 恶意软件检测

为验证本文 DroidSA 对恶意软件的检测能力, 采用 DroidSA、MalSan[12]和 MaMaDroid[13]这 3 种方法进行检测对比实验. 对表 3 中 2012–2018 这 7 年软件样本分别进行十折交叉验证测试, 得到上述 3 种方法对恶意软件的 4 个检测性能指标, 实验结果如表 4 所示.

由表 4 可见, 在支持向量机、随机森林和 K 近邻 3 种分类算法中, DroidSA 使用随机森林时检测效果最好, 取得 96.7% 的平均 *F*1 值, 与 MaMaDroid 和 MalScan 相比分别提升 4.3% 和 6.4%. DroidSA 的平均准确率为 96.7%, 与 MaMaDroid 和 MalScan 相比分别提升 4.3% 和 6.1%; 精确率为 98.5%, 与 MaMaDroid 和 MalScan 相比分别提升 5.5% 和 5.6%; 召回率为 94.8%, 与 MaMaDroid 和 MalScan 相比分别提升 3.2% 和 6.2%.

由表 4 可见, MaMaDroid 的各项指标随时间推移缓慢提升, MalScan 在 2012–2014 年间精确率较高, 召回率较

低, 而在 2015–2018 年间精确率较低, 召回率较高, 此类现象可能与安卓生态环境的变化[38]相关. 相比之下, DroidSA 始终能将各项指标稳定地维持在高水平, 受到时间变化的影响较小.

表 4 对同一时间段内开发的恶意软件的检测能力 (%)

年份	MaMaDroid				MalScan				DroidSA-RF			
	Acc	Pre	Rec	*F*1	Acc	Pre	Rec	*F*1	Acc	Pre	Rec	*F*1
2012	89.6	91.8	87.5	89.6	92.1	99.4	85.0	91.6	96.8	99.5	94.2	96.8
2013	89.2	90.2	88.1	89.1	88.6	98.0	78.8	87.4	95.8	97.0	94.6	95.8
2014	91.6	91.5	91.7	91.6	90.3	97.6	82.7	89.5	96.5	98.4	94.6	96.5
2015	93.2	93.9	92.4	93.2	91.6	89.1	94.8	91.9	96.5	98.7	94.1	96.4
2016	94.5	94.8	94.1	94.4	90.9	87.9	95.0	91.3	96.4	98.7	93.9	96.3
2017	94.7	95.6	93.8	94.7	91.4	90.2	92.9	91.5	97.0	98.7	95.3	97.0
2018	96.0	96.3	95.6	96.0	93.0	90.6	96.1	93.2	97.8	98.7	97.0	97.8
平均	92.7	93.4	91.9	92.7	91.1	93.3	89.3	90.9	**96.7**	**98.5**	94.8	**96.7**

年份	DroidSA-SVM				DroidSA-1NN				DroidSA-3NN			
	Acc	Pre	Rec	*F*1	Acc	Pre	Rec	*F*1	Acc	Pre	Rec	*F*1
2012	95.5	96.1	95.2	95.6	95.8	95.6	96.3	95.9	95.7	96.7	94.8	95.7
2013	95.2	94.8	95.7	95.2	95.4	95.5	95.3	95.4	95.6	97.4	93.7	95.5
2014	95.6	96.8	94.3	95.6	95.4	95.3	95.6	95.4	95.5	96.3	94.6	95.5
2015	95.1	96.0	94.1	95.0	96.1	96.1	96.2	96.1	95.8	96.1	95.4	95.7
2016	95.5	95.6	95.3	95.4	94.7	92.7	97.0	94.8	95.8	96.1	95.5	95.8
2017	96.7	97.1	96.4	96.7	96.6	96.5	96.8	96.7	96.5	96.7	96.4	96.5
2018	97.3	97.4	97.1	97.3	97.5	97.0	98.0	97.5	97.5	97.2	97.7	97.5
平均	95.8	96.3	95.4	95.8	95.9	95.5	**96.5**	96.0	96.1	96.6	95.4	96.0

6.3 观念迁移样本检测

通过设计 3 个实验场景评估 DroidSA 对进化后恶意软件的检测能力. 选择 MaMaDroid、MalScan、AE-MaMaDroid[7,13]和 AE-MalScan[7,12]作为对比方法. AE-MaMaDroid 和 AE-MalScan 分别代表使用 APIGraph[7]对 MaMaDroid 和 MalScan 的增强. APIGraph 是一种对恶意软件分类器的增强方法, 使用聚类时得到的 API 聚类中心代替特征向量中的 API, 能延缓分类器的老化速度.

● 场景 1 (Scenario A) 使用 2012–2013 年的软件进行十折交叉验证, 获得 10 个训练好的分类器, 分别对 2014–2018 年开发的恶意软件进行检测, 取 10 次检测结果的平均值. 结果如图 6(a) 所示.

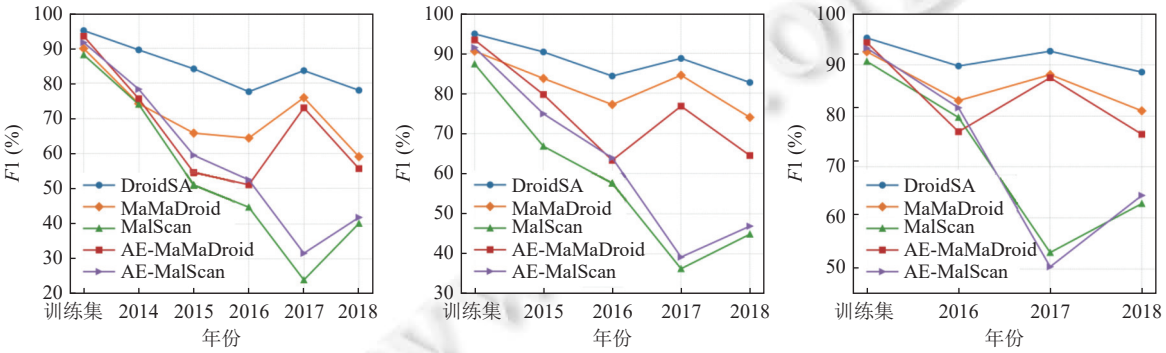


图 6 不同方法对观念迁移样本的检测能力

● 场景 2 (Scenario B) 使用 2013–2014 年的软件进行十折交叉验证, 获得 10 个训练好的分类器, 分别对 2015–2018 年开发的恶意软件进行检测, 取 10 次检测结果的平均值. 结果如图 6(b) 所示.

● 场景 3 (Scenario C) 使用 2014–2015 年的软件进行十折交叉验证, 获得 10 个训练好的分类器, 分别对 2016–2018 年开发的恶意软件进行检测, 取 10 次检测结果的平均值. 结果如图 6(c) 所示.

由图 6 可见, DroidSA 的衰减速度显著慢于其他方法. 场景 1 中, DroidSA 对 2014–2018 年的恶意软件的检测平均 $F1$ 值为 82.6%, 比 MaMaDroid 的 67.9% 提升 22%; 场景 2 中, DroidSA 对 2015–2018 年的恶意软件的检测 $F1$ 值都高于 80%; 场景 3 中, DroidSA 对 2016–2018 年的恶意软件的检测平均 $F1$ 值达到 90.4%. 上述结果是在没有任何重训练的情况下实现的.

与 MaMaDroid 将 API 简单抽象为包的聚类方法相比, DroidSA 充分考虑 API 方法名和权限等特征中蕴含的丰富语义信息, 聚类方法更为合理. 在恶意软件检测阶段, DroidSA 通过调用图优化使其能够提取到更多反映软件行为逻辑的关键特征信息, 因此由旧软件样本训练后的 DroidSA 能有效检测进化后的恶意软件.

MalScan 基于敏感 API 的中心度信息进行恶意软件检测, 但大部分敏感 API 在 2016 年后便很少被使用, 因此在对 2017 年软件样本检测时 MalScan 的 $F1$ 值下降幅度明显. AE-MalScan 使用 APIGraph 生成的聚类中心在函数调用图中的中心度信息替换敏感 API, 能有效延缓 MalScan 的衰减速度. MaMaDroid 在使用 APIGraph 进行增强后, 衰减速度反而加快, 具体原因将在第 6.5 节中进行分析.

6.4 不同聚类方法的有效性验证

API 方法名概括了 API 的功能, 是 API 语义信息提取的关键特征. 为评估 API 方法名对聚类结果的影响, 通过实验对比 3 种基于 API 聚类的检测方法对观念迁移样本的检测能力. 第 1 种检测方法 (DroidSA-onehot, DOH) 使用本文提出的 API 聚类方法; 第 2 种检测方法 (without method name, WMN) 的 API 聚类方法在生成 API 句子时忽略 API 方法名, 其他步骤与本文提出的基于语义距离的 API 聚类方法相同; 第 3 种检测方法 (APIGraph-onehot, AOH) 使用 APIGraph 聚类方法.

上述 3 种检测方法均使用代表聚类中心是否出现的独热编码生成特征向量, 在支持向量基、随机森林和 K 近邻 3 种分类算法中选择对应表现最好的分类算法, 进行恶意软件检测. 选择 Scenario A (图 7(a)) 和 Scenario B (图 7(b)) 两个实验场景进行十折交叉验证. DroidSA 和 3 种基于 API 聚类的检测方法对观念迁移样本的检测结果如图 7 所示.

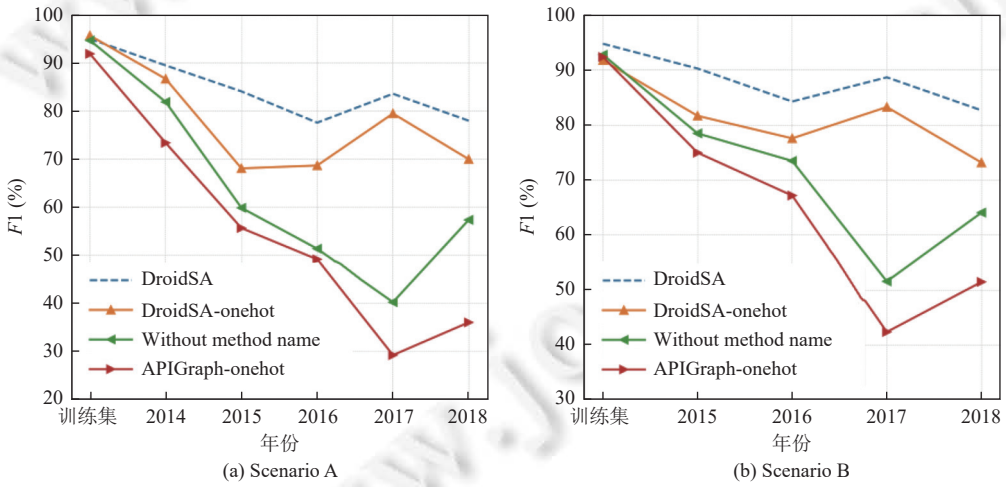


图 7 3 种基于 API 聚类的检测方法对观念迁移样本的检测能力

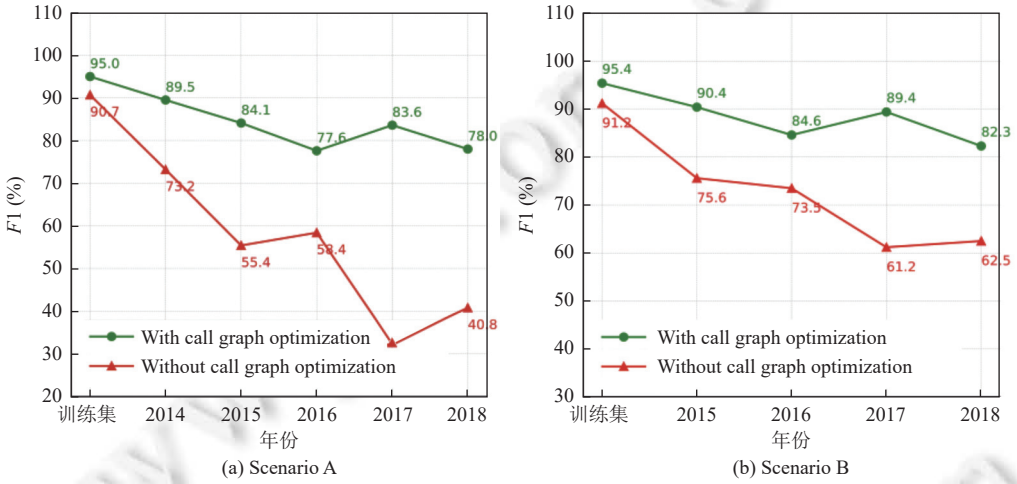
在图 7(a) 所示的 Scenario A 中, WMN 对 2014–2018 年的恶意软件的检测平均 $F1$ 值为 58.2%, 与 AOH 相比提高 19.5%; DOH 对 2014–2018 年的恶意软件的检测平均 $F1$ 值为 74.6%, 与 WMN 相比提高 28.2%; DroidSA 对 2014–2018 年的恶意软件的检测平均 $F1$ 值为 82.6%, 与 AOH 相比提高 10.7%. 在图 7(b) 所示的 Scenario B 中, WMN 对 2015–2018 年的恶意软件的检测平均 $F1$ 值为 66.9%, 与 AOH 相比提高 13.4%; DOH 对 2015–2018 年的

恶意软件的检测平均 $F1$ 值为 79.0%, 与 WMN 相比提高 18.1%; DroidSA 对 2015–2018 年的恶意软件的检测平均 $F1$ 值为 86.5%, 与 DOH 相比提高 9.5%.

AOH 和 WMN 在聚类时都没有考虑 API 方法名, 因此这两种方法衰减速度较快, 难以有效检测进化后的恶意软件. 但 WMN 使用 BERT 对类名和权限等特征中的语义信息进行挖掘, 与 AOH 相比衰减速度较慢. 本文提出的聚类方法 DOH 充分利用方法名称中蕴含的语义信息, 能有效检测进化后的恶意软件, 与 AOH 和 WMN 相比有大幅的提升. 在使用反映 API 上下文信息的聚类中心调用对作为特征向量 (DroidSA) 时, 对进化后恶意软件的检测能力进一步增强. 这说明方法名对 API 聚类十分重要, 能有效提高分类器对观念迁移样本的检测能力.

6.5 调用图优化对检测性能的影响

为评估调用图优化对 DroidSA 衰减速度的影响, 设计了 Scenario A (图 8(a)) 和 Scenario B (图 8(b)) 两个实验场景并进行十折交叉验证. DroidSA 在缺少调用图优化时的检测效果如图 8 所示.



(a) Scenario A (b) Scenario B

图 8 缺少调用图优化时的检测性能

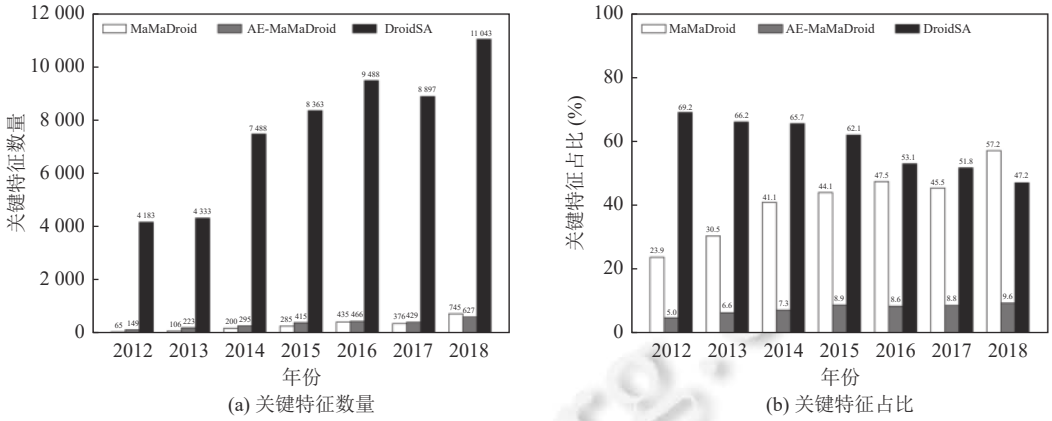
由图 8 可知, 调用图优化对 DroidSA 检测恶意软件的能力提升有限, 但能显著提高 DroidSA 对观念迁移样本的检测能力. 在 Scenario A 中, 未使用调用图优化的 DroidSA 对 2014–2018 年的恶意软件的检测平均 $F1$ 值为 52%, 而在使用调用图优化后, 检测平均 $F1$ 值为 82.6%, 提升幅度为 59%; 在 Scenario B 中, 未使用调用图优化的 DroidSA 对 2015–2018 年的恶意软件的检测平均 $F1$ 值为 68.2%, 在使用调用图优化后, 检测平均 $F1$ 值为 86.7%, 提升幅度为 27%. 表明调用图优化能够显著提升分类器对进化后恶意软件的检测能力.

第 6.3 节的观念迁移样本检测实验结果表明, APIGraph 削弱了 MaMaDroid 对观念迁移样本的检测能力. 其原因是缺失调用图优化过程, 函数调用图中大量未知函数导致 AE-MaMaDroid 难以提取软件的关键特征. 与未知函数相比, 包调用对和聚类中心调用对能更好反映软件的行为逻辑, 能使检测方法有效适应安卓框架的不断变化. 因此, 包调用对和聚类中心调用对是恶意软件检测的关键特征. MaMaDroid、AE-MaMaDroid 和 DroidSA 的特征向量中关键特征的数量和占比如图 9 所示.

图 9 中, MaMaDroid 平均每年只能提取到 317 种包调用对, 即关键特征占特征总数的 41.4%. 由于缺失调用图优化过程, MaMaDroid 在处理未知函数时根据代码保护机制[30]将其抽象为自定义函数或模糊函数, 导致特征向量中出现大量未知函数与包组成的调用对, 这些非关键特征约占特征总数的 60%.

AE-MaMaDroid 将函数调用对中的 API 抽象为聚类中心, 其余步骤与 MaMaDroid 相同. 如图 9 所示, 经过 APIGraph “增强”后 AE-MaMaDroid 平均每年只能提取到 372 种聚类中心调用对, 占特征总数的 7.8%. 剩余 92.2% 的特征是聚类中心与未知函数的调用. 特征空间中表征软件行为逻辑的关键特征数量大幅减少, 导致 APIGraph 反而使 MaMaDroid 的衰减速度加快.

而 DroidSA 在调用图优化后, 平均每年能够提取到 7685 种聚类中心调用对, 占特征总数的 57.7%. 这说明调用图优化能使检测方法提取到更多表征软件行为逻辑的关键特征, 对观念迁移样本检测十分重要.



(a) 关键特征数量 (b) 关键特征占比

图 9 特征向量中关键特征数量和占比

7 总 结

现有安卓恶意软件检测方法未考虑恶意软件中 API 的频繁变化, 导致难以有效检测进化后的恶意软件. 为解决上述问题, 本文提出一种基于 API 聚类和调用图优化的安卓恶意软件检测方法 DroidSA. 通过设计 API 句子概括 API 的名称、权限等重要特征并使用自然语言处理工具对 API 句子的语义信息进行挖掘, 获得更全面反映 API 语义相似性的嵌入向量, 使聚类结果更为准确. 为了确保提取到更能准确反映软件行为逻辑的 API 上下文信息, 采用调用图优化方法对从待检测软件中提取的函数调用图进行优化, 得到优化后的调用图, 在删除图中难以识别的未知方法同时保留 API 节点之间的连接性.

DroidSA 的恶意软件检测平均 $F1$ 值为 96.7%, 与 MaMaDroid 和 MalScan 相比分别提升 4.3% 和 6.4%; 在消除时间偏差的实验设置下, 经 2012 年和 2013 年软件样本集训练后的 DroidSA, 对 2014–2018 年期间的恶意软件样本的检测平均 $F1$ 值达到 82.6%, 与 MaMaDroid、MalScan、AE-MaMaDroid 和 AE-MalScan 相比分别提升 22%、76%、33% 和 57%. 实验结果表明本文提出的恶意软件检测方法始终能将各项检测指标稳定地维持在高水平且受到时间变化的影响较小, 并能有效检测进化后的恶意软件. 未来将尝试使用图神经网络[39]等深度学习模型进行恶意软件检.

References:

[1] Liu J, Su PR, Yang M, He L, Zhang Y, Zhu XY, Lin HM. Software and cyber security—A survey. Ruan Jian Xue Bao/Journal of Software, 2018, 29(1): 42–68 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5320.htm [doi: 10.13328/j.cnki.jos.005320]

[2] Lu H, Jin CJ, Helu XH, Zhu CS, Guizani N, Tian ZH. Autod: Intelligent blockchain application unpacking based on JNI layer deception call. IEEE Network, 2021, 35(2): 215–221. [doi: 10.1109/MNET.011.2000467]

[3] Lu H, Jin CJ, Helu XH, Du XJ, Guizani N, Tian ZH. Deepautod: Research on distributed machine learning oriented scalable mobile communication security unpacking system. IEEE Trans. on Network Science and Engineering, 2022, 9(4): 2052–2065. [doi: 10.1109/TNSE.2021.3100750]

[4] Lu H, Jin CJ, Helu XH, Zhang M, Sun YB, Han Y, Tian ZH. Research on intelligent detection of command level stack pollution for binary program analysis. Mobile Networks and Applications, 2021, 26(4): 1723–1732. [doi: 10.1007/s11036-019-01507-0]

[5] Pendlebury F, Pierazzi F, Jordaney R, Kinder J, Cavallaro L. Tesseract: Eliminating experimental bias in malware classification across space and time. In: Proc. of the 28th USENIX Security Symp. Santa Clara: USENIX Association, 2019. 792–746.

[6] Jordaney R, Sharad K, Dash SK, Wang Z, Papini D, Nouretdinov I, Cavallaro L. Transcend: Detecting concept drift in malware

classification models. In: Proc. of the 26th USENIX Security Symp. Vancouver: USENIX Association, 2017. 625–642.

[7] Zhang XH, Zhang Y, Zhong M, Ding DZ, Cao YZ, Zhang YK, Zhang M, Yang M. Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware. In: Proc. of the 2020 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2020. 757–770. [doi: 10.1145/3372297.3417291]

[8] Arp D, Spreitzenbarth M, Hübner M, Gascon H, Rieck K. DREBIN: Effective and explainable detection of Android malware in your pocket. In: Proc. of the 21st Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2014. 1–15.

[9] Aafer Y, Du WL, Yin H. DroidAPIMiner: Mining API-level features for robust malware detection in Android. In: Proc. of the 9th Int'l ICST Conf. on Security and Privacy in Communication Networks. Sydney: Springer, 2013. 86–103. [doi: 10.1007/978-3-319-04283-1_6]

[10] Feng RT, Chen S, Xie XF, Meng GZ, Lin SW, Liu Y. A performance-sensitive malware detection system using deep learning on mobile devices. IEEE Trans. on Information Forensics and Security, 2021, 16: 1563–1578. [doi: 10.1109/TIFS.2020.3025436]

[11] Allen J, Landen M, Chaba S, Ji Y, Chung SPH, Lee W. Improving accuracy of Android malware detection with lightweight contextual awareness. In: Proc. of the 34th Annual Computer Security Applications Conf. San Juan: ACM, 2018. 210–221. [doi: 10.1145/3274694.3274744]

[12] Wu YM, Li XD, Zou DQ, Yang W, Zhang X, Jin H. MalScan: Fast market-wide mobile malware scanning by social-network centrality analysis. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 139–150. [doi: 10.1109/ASE.2019.00023]

[13] Mariconti E, Onwuzurike L, Andriotis P, de Cristofaro E, Ross G, Stringhini G. Mamadroid: Detecting android malware by building Markov chains of behavioral models. In: Proc. of the 24th Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2017. 1–16.

[14] Lei T, Qin Z, Wang ZB, Li Q, Ye DP. EveDroid: Event-aware android malware detection against model degrading for IoT devices. IEEE Internet of Things Journal, 2019, 6(4): 6668–6680. [doi: 10.1109/JIOT.2019.2909745]

[15] Lau JH, Baldwin T. An empirical evaluation of doc2vec with practical insights into document embedding generation. In: Proc. of the 1st Workshop on Representation Learning for NLP. Berlin: Association for Computational Linguistics, 2016. 78–86. [doi: 10.18653/v1/W16-1609]

[16] Au KWY, Zhou YF, Huang Z, Lie D. PScout: Analyzing the Android permission specification. In: Proc. of the 2012 ACM Conf. on Computer and Communications Security. Raleigh North: ACM, 2012. 217–228. [doi: 10.1145/2382196.2382222]

[17] Bordes A, Usunier N, Garcia-Duran A, Weston J, Yakhnenko O. Translating embeddings for modeling multi-relational data. In: Proc. of the 26th Int'l Conf. on Neural Information Processing Systems. Lake Tahoe: Curran Associates Inc., 2013. 2787–2795.

[18] Xu JY, Li YJ, Deng RH, Xu K. SDAC: A slow-aging solution for Android malware detection using semantic distance based API clustering. IEEE Trans. on Dependable and Secure Computing, 2022, 19(2): 1149–1163. [doi: 10.1109/TDSC.2020.3005088]

[19] Arp D, Quiring E, Pendlebury F, Pendlebury F, Warnecke A, Pierazzi F, Wressnegger C, Cavallaro L, Rieck K. Dos and don'ts of machine learning in computer security. In: Proc. of the 31st USENIX Security Symp. Boston: USENIX Association, 2022. 3971–3988.

[20] Narayanan A, Liu Y, Chen LH, Liu JL. Adaptive and scalable Android malware detection through online learning. In: Proc. of the 2016 Int'l Joint Conf. on Neural Networks. Vancouver: IEEE, 2016. 2484–2491. [doi: 10.1109/IJCNN.2016.7727508]

[21] Xu K, Li YJ, Deng R, Chen K, Xu JY. DroidEvolver: Self-evolving Android malware detection system. In: Proc. of the 2019 IEEE European Symp. on Security and Privacy. Stockholm: IEEE, 2019. 47–62. [doi: 10.1109/EuroSP.2019.00014]

[22] Gu YH, Li LX. GraphEvolveDroid: Mitigate model degradation in the scenario of Android ecosystem evolution. In: Proc. of the 30th ACM Int'l Conf. on Information & Knowledge Management. ACM, 2021. 3588–3591.

[23] Hei YM, Yang RY, Peng H, Wang LH, Xu XL, Liu JW, Liu H, Xu J, Sun LC. Hawk: Rapid Android malware detection through heterogeneous graph attention networks. IEEE Trans. on Neural Networks and Learning Systems, 2024, 35(4): 4703–4717. [doi: 10.1109/TNNLS.2021.3105617]

[24] Yuan C, Cai JX, Tian DH, Ma R, Jia XQ, Liu WM. Towards time evolved malware identification using two-head neural network. Journal of Information Security and Applications, 2022, 65: 103098. [doi: 10.1016/j.jisa.2021.103098]

[25] Karbab EB, Debbabi M. PetaDroid: Adaptive android malware detection using deep learning. In: Proc. of the 18th Int'l Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 2021. 319–340. [DOI: 10.1007/978-3-030-80825-9_16]

[26] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional Tansformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: Association for Computational Linguistics, 2019. 4171–4186. [doi: 10.18653/v1/n19-1423]

[27] Syakur MA, Khotimah BK, Rochman EMS, Satoto BD. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. IOP Conf. Series: Materials Science and Engineering, 2018, 336: 012017. [doi 10.1088/1757-899X/336/1/

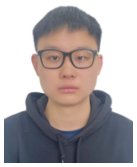
012017]
[28] Source code of APIGraph. https://github.com/seclab-fudan/APIGraph
[29] Abdi H, Williams LJ. Principal component analysis. WIREs Computational Statistics, 2010, 2(4): 433–459. [doi: 10.1002/wics.101]
[30] Schulz P. Code protection in Android. 2012. https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf
[31] Arzt S, Rasthofer S, Fritz C, Bodden E, Bartel A, Klein J, Le Traon Y, Octeau D, McDaniel P. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android Apps. In: Proc. of the 35th ACM SIGPLAN Conf. on Programming Language Design and Implementation. Edinburgh: ACM, 2014. 259–269. [doi: 10.1145/2594291.2594299]
[32] Hearst MA, Dumais ST, Osuna E, Platt J, Scholkopf B. Support vector machines. IEEE Intelligent Systems and their Applications, 1998, 13(4): 18–28. [doi: 10.1109/5254.708428]
[33] Breiman L. Random forests. Machine Learning, 2001, 45(1): 5–32. [doi: 10.1023/A:1010933404324]
[34] Allix K, Bissyandé TF, Klein J, Le Traon Y. AndroZoo: Collecting millions of Android Apps for the research community. In: Proc. of the 13th Int'l Conf. on Mining Software Repositories. Austin: ACM, 2016. 468–471. [doi: 10.1145/2901739.2903508]
[35] VirusTotal. https://www.virustotal.com/gui/home/upload
[36] Wei FG, Li YP, Roy S, Ou XM, Zhou W. Deep ground truth analysis of current Android malware. In: Proc. of the 14th Int'l Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment. Bonn: Springer, 2017. 252–276. [doi: 10.1007/978-3-319-60876-1_12]
[37] Google Play Store. https://play.google.com/store
[38] Suarez-Tangil G, Stringhini G. Eight years of rider measurement in the Android malware ecosystem. IEEE Trans. on Dependable and Secure Computing, 2022, 19(1): 107–118. [doi: 10.1109/TDSC.2020.2982635]
[39] Wu ZH, Pan SR, Chen FW, Long GD, Zhang CQ, Yu PS. A comprehensive survey on graph neural networks. IEEE Trans. on Neural Networks and Learning Systems, 2021, 32(1): 4–24. [doi: 10.1109/TNNLS.2020.2978386]

附中文参考文献:

[1] 刘剑, 苏璞睿, 杨珉, 和亮, 张源, 朱雪阳, 林惠民. 软件与网络安全研究综述. 软件学报, 2018, 29(1): 42–68. http://www.jos.org.cn/1000-9825/5320.htm [doi: 10.13328/j.cnki.jos.005320]

杨宏宇(1969－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为网络与系统安全, 软件安全检测, 网络安全态势感知.

胡泽(1989－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为人工智能, 自然语言处理, 网络信息安全.



汪有为(1999－), 男, 硕士生, 主要研究领域为网络信息安全, 安卓恶意软件检测.

姜来为(1986－), 女, 博士, 讲师, CCF 专业会员, 主要研究领域为网络信息安全.

张良(1987－), 男, 博士, 研究员, 主要研究领域为强化学习, 基于深度学习的信号处理, 网络与系统安全.

成翔(1988－), 男, 博士, 实验师, CCF 专业会员, 主要研究领域为网络与系统安全, 网络安全态势感知, APT 攻击检测.