基于API时序关系图的恶意软件检测方法

李勇男[1,2]，赵莹[3]
（1.中国人民公安大学国家安全学院，北京 100038；2.智慧警务与国家安全风险治理重点实验室，四川 泸州 646000；
3.桂林电子科技大学商学院，广西 桂林 541004）

摘 要：为解决恶意软件检测误报率和漏报率高等问题，提出了一种基于API时序关系图（ATRG）的恶意软件检测方法。首先，提出了ATRG模型，用于融合恶意软件执行过程中的API调用序列，对API调用序列进行建模，有效地表征软件的执行行为；然后，设计了一种时序随机游走算法，从ATRG中提取路径，以刻画细粒度的软件行为特征；最后，基于提取的大量路径，借助深度学习模型对路径进行编码，并构建二分类器来识别恶意软件。实验结果表明，该方法在恶意软件数据集上的检测精确率、召回率和F1分数分别达到97.83%、98.03%和97.93%，相比于现有的最优恶意软件检测方法API-BiLSTM，分别提升了2.71%、2.76%和2.74%。

关键词：恶意软件；恶意软件检测；深度学习；API调用序列；API时序关系图
中图分类号：TP389.1
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2025227

Malware detection method based on API temporal relationship graph

LI Yongnan[1,2], ZHAO Ying[3]
1. School of National Security, People's Public Security University of China, Beijing 100038, China
2. Intelligent Policing and National Security Risk Management Laboratory, Luzhou 646000, China
3. School of Business, Guilin University of Electronic Technology, Guilin 541004, China

Abstract: To solve the problems of high false positives and high false negatives during malware detection, a malware detection method based on an ATRG (API temporal relationship graph) was proposed. First, an ATRG model was introduced that fuses the API call sequences during the execution of malware, effectively representing software execution behavior. Then, a temporal random walk algorithm was designed to extract paths from the ATRG, capturing fine-grained software behaviors. Finally, based on a large number of extracted paths, a deep learning model was utilized for path encoding and a binary classifier was constructed to identify malware. Experimental results show that the proposed method achieves a detection accuracy, recall, and F1-score of 97.83%, 98.03%, and 97.93% respectively on the malware dataset, representing an improvement of 2.71%, 2.76%, and 2.74% over the current best malware detection method, API-BiLSTM.

Keywords: malware, malware detection, deep learning, API call sequence, API temporal relationship graph

收稿日期：2025–07–01；**修回日期：**2025–12–12
通信作者：李勇男，liyongnan@ppsuc.edu.cn
基金项目：某省部级重点基金资助项目（No.2023JSZ04）；中国人民公安大学2024年度基科费资助项目（No.2024JKF01）；智慧警务与国家安全风险治理重点实验室开放课题资助项目（No.ZHKFYB2502）
Foundation Items: Ministerial-level Key Scientific Research Project (No.2023JSZ04), The 2024 Basic Scientific Research Fund Project of People's Public Security University of China (No.2024JKF01), The General Project of the Open Fund of the Key Laboratory of Intelligent Policing and National Security Risk Governance (No.ZHKFYB2502)

0 引言

随着互联网的蓬勃发展，各种类型的软件已经应用于生活的方方面面。然而，恶意软件的数目呈持续上升的趋势。据Statista统计分析，2023年全球观测到的恶意软件威胁达到了66亿起[1]。这些恶意软件日益复杂，传播方式多样，对计算机系统和网络安全构成了严重威胁。如何进行高效、准确的恶意软件分析和检测，已成为一个亟待解决的研究课题。应用程序接口（API, application programming interface）调用序列能够有效地反映恶意软件的执行行为，因此，基于API调用序列的恶意软件检测技术已成为研究的热点[2-5]。

近年来，研究人员提出了多种恶意软件检测方法[6-19]。目前的恶意软件检测技术主要分为以下3类。基于签名的检测技术[6-10]依赖已知恶意软件的签名特征库，通过比对特征库中的已知签名来判断目标样本是否为恶意软件，甚至可以准确地确定其所属的恶意软件家族。基于行为的检测技术[11-14]通过监控目标程序的执行行为来判断其是否具有恶意性。该方法可以监测并记录恶意软件的行为信息，如API调用序列、系统调用序列等，然后利用机器学习或深度学习方法对这些行为信息进行学习，构建分类模型，以检测已知甚至未知的恶意软件，具有较高的灵活性和适应性。基于深度学习的检测技术[15-19]利用深度学习模型，如卷积神经网络（CNN, convolutional neural network）、图神经网络（GNN, graph neural network）和循环神经网络（RNN, recurrent neural network），对静态分析或动态分析所得数据进一步提取特征，从而实现高效且准确的恶意软件检测。由于其出色的特征提取能力，该类技术通常具备检测未知恶意软件或恶意软件变体的能力。然而，现有检测方法仍存在一定局限。具体而言，基于签名的检测技术难以识别新型或未知恶意软件（如利用0-day或1-day漏洞的样本），因此容易出现较高的漏报率；基于行为的检测技术则可能将正常软件行为误判为恶意行为，尤其在面对对抗性恶意软件时检测能力不足；而基于深度学习的检测技术在应对复杂恶意软件时，难以挖掘出潜在的恶意行为，面临漏报与误报的挑战。

本文围绕上述挑战，对恶意软件检测技术进行了深入探索，旨在提出一种新的恶意软件检测模型，利用动态API调用序列中蕴含的丰富行为信息，设计解决方案以提升恶意软件检测的效果。本文提出了一种基于API时序关系图（ATRG， API temporal relationship graph）的恶意软件检测方法，该方法通过对恶意软件动态执行的API调用序列进行建模，并结合深度学习模型实现恶意软件的检测。本文首先提出了ATRG模型，该模型将API作为节点，以API之间的时序关系为边。由于相邻API往往涉及细粒度的行为，如木马窃密行为，包括InternetOpenUrlA（打开链接，连接目标服务器）、GetWindowTextA（捕获用户输入的信息）、Send（将信息发送给远程服务器），ATRG能够有效刻画软件的执行行为。基于ATRG，本文设计了时序随机游走算法，该算法在图中进行游走，遵循API的时序关系，生成符合软件行为模式的API路径，从而表征更细粒度的软件执行行为，特别是恶意行为。最后，针对生成的API路径，本文利用预训练模型进行嵌入表征，并使用TextCNN模型构建二分类器，实现恶意软件的检测。

本文提出的检测方法在Keras/Tensorflow平台上实现，并在恶意软件数据集上进行了方法与性能评估。实验结果表明，本文方法在恶意软件数据集上的检测精确率、召回率和F1分数分别达到了97.83%、98.03%和97.93%，相比现有的方法（如API-BiLSTM），分别提升了2.71%、2.76%和2.74%。

综上所述，本文的主要贡献包括以下3个方面。

1) 提出了一种针对软件执行API序列的时序关系图模型，用于表征软件的执行行为。

2) 提出了一种时序随机游走算法，通过在ATRG模型中挖掘路径，刻画细粒度的执行行为。

3) 借助深度学习构建了分类器，实现了恶意软件的检测，并在公开数据集上验证了该方法的有效性。

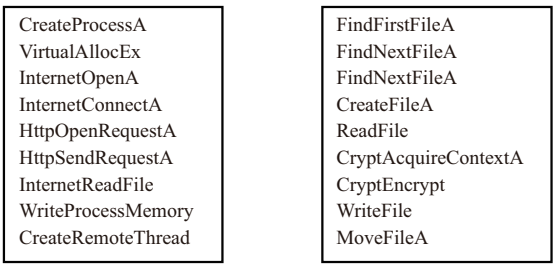
1 背景与相关研究

1.1 软件执行与API调用序列

恶意软件是指在未经用户同意的情况下安装，并带有恶意意图的应用程序，其主要目的是破坏计算机系统，进而对用户的隐私、财产等造成不可估量的损失。例如，近年来勒索软件迅速蔓延，对全球各地的医院信息系统、企业网络以及个人电脑构成了严重威胁。勒索软件通过加密用户的重要文件，要求支付赎金以解密，导致许多机构和个人蒙

受巨大的经济损失，甚至导致关键服务的中断。

恶意软件的动态分析是常用的检测方法之一[2-5]，通过在隔离的沙箱环境中执行可疑软件，捕获其执行过程中产生的API调用序列，并对这些序列进行深入分析，以识别潜在的恶意行为，从而实现对恶意软件的检测。与静态分析不同，动态分析能够捕获实际触发的恶意行为，从而有效避免恶意软件利用混淆、加密等手段逃避检测。API序列样例如图1所示，图1(a)展示了一个通过远程下载恶意软件并在本地执行恶意代码的API序列示例，图1(b)展示了一个遍历本地文件并进行加密的勒索行为示例。



(a) 远程下载执行恶意代码　　(b) 文件遍历与加密勒索行为

图1　API序列样例

在沙箱环境中，恶意软件运行时会执行各种API调用，例如文件读写、网络通信、进程创建等。这些API调用序列是分析恶意行为的重要依据。通过记录和分析这些API调用，可以识别出恶意软件的行为模式。例如，一些恶意软件可能试图修改系统文件、窃取用户信息或建立远程连接，这些行为特征都可以在API调用序列中清晰地反映出来。

1.2　恶意软件检测技术概述

恶意软件检测技术主要分为3类：基于签名的检测技术、基于行为的检测技术和基于深度学习的检测技术。

在基于签名的恶意软件检测技术方面，Ye等[8]通过分析文件之间的依赖关系、调用关系和引用关系，生成短的字节序列作为签名，从而实现了常见恶意软件行为的检测。Wael等[9]则针对VBScript语言，收集了258个签名特征，包括函数特征和指令特征，并利用多种机器学习算法进行恶意软件检测。为了解决人工提取签名带来的负担，Zhang等[10]提出了一种基于N元语法属性相似度的恶意软件检测和分类方法，实现了自动化特征提取和检测。基于签名的检测方法具有速度快、精确率高的优势，但其局限于检测已知的恶意软件，难以应对混淆、加密等对抗手段；此外，这种方法还依赖于大量的人工特征提取，增加了人力成本。

在基于行为的恶意软件检测技术方面，Ki等[11]深入分析了软件行为特征，从目标样本中提取API调用序列，并将其建模为DNA序列，然后使用DNA序列比对算法进行恶意软件检测。Hansen等[12]将动态行为信息转换为鲁棒的特征向量，这些向量主要包括API、每个API的调用频率以及相应的参数信息，并利用随机森林模型对恶意软件进行分类。Kim等[13]采用N元语法提取API序列中的特征，并进行分组，随后使用词频–逆文档频率（TF-IDF）算法生成系统调用特征，再结合支持向量机和随机梯度下降算法进行恶意软件检测。基于行为的检测技术证明了动态分析获取的行为特征，尤其是API序列，在表达恶意软件行为信息方面的有效性。然而，如何精确提取行为特征仍然是该领域需要解决的关键问题。

在基于深度学习的恶意软件检测技术方面，Zhang等[15]则将API调用及其相关参数编码在一起，然后使用卷积神经网络和长短期记忆（LSTM）网络来学习API调用的表示。Kolosnjaji等[16]将API序列输入循环神经网络中，训练用于恶意软件检测的二分类器。Chen等[17]评估了API调用参数的敏感性，通过编码表示单个API调用及其不同参数的威胁程度，最终使用文本卷积神经网络（TextCNN）和双向长短期记忆（BiLSTM）网络训练恶意软件分类器进行检测。Rosenberg等[18]将原始API序列划分为多个子序列，并分别使用循环神经网络检测每个子序列的恶意性。Maniriho等[19]结合卷积神经网络和双向门循环单元，从原始API序列中提取特性。Zhang等[20]将API调用序列转化为灰度图像，并通过多种深度学习算法提取语义细节、时序特征、API调用频率等多维度数据。此外，一些研究倾向于将API调用表示为图结构，并应用基于图的深度学习算法进行分析。例如，Boujnouni等[21]从原始API调用序列中提取数据依赖关系，构建API调用图，然后使用最长公共子序列算法进行图匹配，以应用于恶意软件检测任务。Zhang等[22]通过解析Android API文档，提取API节点、权限节点及其之间的关系构建图模型，并设计

了知识图嵌入算法来学习 API 的嵌入表示。LI 等[23]则对 API 序列及其参数信息进行编码，构建 API 调用图，并使用图卷积网络（GCN, graph convolutional network）进行恶意软件检测。Feng 等[24-25]首先构建静态的网络行为函数调用图，然后利用软件的动态网络流量特征构建节点交互图与边/节点关系图，最终采用图神经网络进行恶意软件检测。Cui 等[26]从原始 API 序列中进行分组，使用图模型表征分组后的 API 序列，并结合 Doc2Vec 和 KNN 构建检测模型。Zhen 等[27]提取进程事件序列及进程调用关系，结合进程结构信息与事件语义构建样本的事件关系图，利用注意力门控图神经网络实现恶意软件检测。Liu 等[28]基于函数调用图进行剪枝生成敏感函数调用图，保留安全相关 API 上下文，并利用图卷积神经网络进行图嵌入表示。上述方法都依赖于原始的 API 调用序列来进行特征挖掘与向量表达，但由于 API 调用序列的粒度较粗，这在一定程度上限制了模型表达能力。

1.3　总结

尽管现有检测方法在恶意软件防御中发挥了一定作用，但仍存在诸多局限性。具体而言，基于签名的检测技术依赖已知特征进行匹配，无法识别新型或未知的恶意软件（如包含 0-day 和 1-day 漏洞的样本），因此容易出现较高的漏报率。基于行为的检测技术虽然能够捕获部分未知威胁，但在复杂系统环境中常常会将正常软件行为误判为恶意行为，从而产生较多误报；同时，对于具有较强隐蔽性的恶意软件，尤其是在执行过程中注入大量良性行为来混淆检测的样本，该方法也表现出明显的局限。基于深度学习的检测技术在面对多样化和复杂化的恶意软件时，往往难以充分挖掘其潜在的恶意特征，受限于训练数据覆盖度和模型泛化能力，仍然难以在漏报与误报之间取得理想的平衡。

2　方法设计与实现

2.1　方法概述

本文提出了一种基于 ATRG 模型的恶意软件检测方法，对恶意软件动态执行的 API 序列进行建模，并借助深度学习模型实现恶意软件的检测。该方法包括 3 个模块：ATRG 模型、时序随机游走算法和深度学习检测模型，整体设计如图 2 所示。

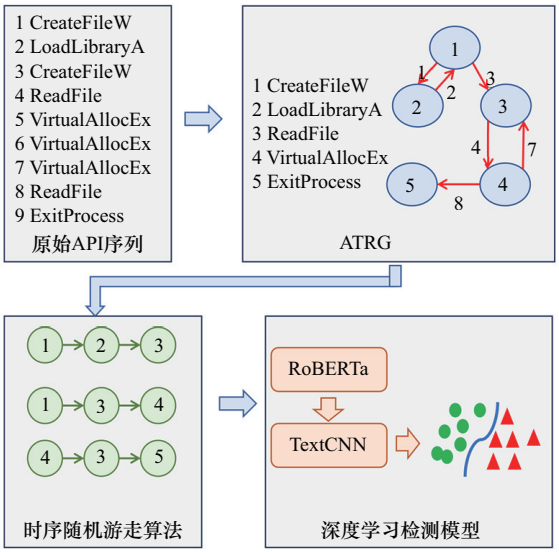


图 2　整体设计

1) ATRG 模型。本文通过分析原始 API 序列，设计了 ATRG 来对软件执行行为进行建模。在 ATRG 中，节点表示单个 API 调用，边表示 API 调用之间的时序关系。根据原始 API 序列中的逻辑顺序，时序信息被建模并作为边的属性进行绑定，以更准确地反映 API 调用的时序关系。

2) 时序随机游走算法。基于 ATRG 模型，该算法对图进行搜索，提取能够表征细粒度软件行为，尤其是恶意行为的路径，即 API 子序列。该算法考虑 API 调用的时序关系，通过结合时序信息的随机游走，提取有效的恶意软件行为表示。最终，该算法生成若干条 API 路径，构成 API 路径库，用于进一步分析。

3) 深度学习检测模型。在 API 路径库的基础上，本文首先利用 RoBERTa 模型[22]进行预训练，深度学习 API 路径之间的关系，并对 API 进行编码。随后，利用编码后的 API 序列，使用 TextCNN 模型构建二分类器，以实现恶意软件的检测。

2.2　ATRG 模型

2.2.1　逻辑时序属性

原始 API 序列中存在调用的先后顺序，这种顺序关系蕴含着 API 之间的逻辑关联和语义完整性。例如，在进行网络通信时，通常先调用 InternetOpenUrl 访问域名，然后使用 InternetReadFile 读取相关文件。同样的原理也适用于文件操作和系统操作等行为。

为增强关系图模型的表达能力，本文为其添加

了逻辑时序属性，使其能够有效建模API调用的时序信息，从而确保行为表达的逻辑正确性和语义完整性。具体来说，原始API调用序列能够显示程序执行过程中的先后顺序。这意味着，如果在原始API序列中某一API（如s）出现在另一API（如t）之前，这表明s先于t被调用。需指出的是，这并不意味着s和t之间存在功能上的直接关联，也不表明t的调用是由s直接或间接触发的。因为仅从API调用序列中，无法获得足够的信息来揭示调用者与被调用者之间的直接或间接关系。因此，原始API调用序列中的逻辑时序信息可以用于识别并捕捉进程间的细粒度执行行为信息。

具体来说，本文使用符号T来表示原始API调用序列中的逻辑时间顺序，从0开始赋值，以固定的步长1单调递增，直至原始API序列中的最后一个API调用。这样，每个API调用都被赋予了一个逻辑时间索引，用于表示其在原始API序列中的时序信息。

2.2.2　ATRG构建

在为每个API分配逻辑时间后，就可以定义和建模ATRG。本文用符号S表示目标进程的原始API调用序列，而与S对应的ATRG可以用三元组$<V,E,A>$来表示。其中，V代表节点集合，即目标进程中调用的API集合，每个节点$v \in V$表示一个单独的API调用；E表示边的集合，每个有向边$e \in E$表示2个API调用之间的顺序关系，基于逻辑时间，例如，如果API节点s在节点t之后立即被调用（逻辑时间差为1），那么在ATRG中就会存在一条从s指向t的有向边e_{st}；A则为节点或边的属性，即逻辑时间。

由于API在序列中可能会重复出现，其出现次数通常少于序列的总长度，因此ATRG有效地减小了序列的规模。同时，2个API之间可能存在多条具有不同时序属性的边，表示某一行为被多次执行，因此ATRG具有多重图的特性。

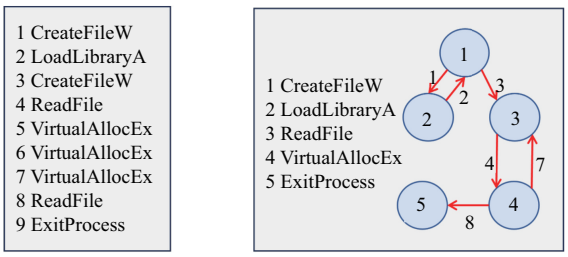
基于上述定义，ATRG的建模过程可以分为3个步骤：逻辑时间赋值、节点构建和连边创建。具体而言，对于给定的原始API调用序列，首先为序列中的每个API调用赋予相应的逻辑时间属性，从0开始，单调递增1，直到最后一个API调用，如式(1)所示。接着，针对赋予逻辑时间的API序列，将第一个API作为初始节点直接添加到ATRG中。随后，检查下一个API调用节点是否已存在于ATRG中。如果不存在，则将该API节点作为一个新节点添加到ATRG中，如式(2)所示，并从上一个API节点引出一条有向边指向当前节点，同时将相应的逻辑时间索引值作为该边的属性值，如式(3)所示。如果存在，则直接引出一条有向边指向当前节点，同时将相应的逻辑时间索引值作为该边的属性值。这个过程将迭代执行，直到最后一个API节点。

$$\forall a_i \in S, \text{time}(a_i) = i \tag{1}$$

$$\forall a_i \in S, \begin{cases} a_i \notin V \Rightarrow V \leftarrow V \cup \{a_i\} \\ a_i \in V \Rightarrow V \leftarrow V \end{cases} \tag{2}$$

$$\forall i \in [0, n-2], E \leftarrow E \cup (a_i, a_{i+1}, i) \tag{3}$$

ATRG实例如图3所示，对于图3(a)所示的原始API序列（左侧的编号表示时序），生成的ATRG如图3(b)所示。在ATRG中，每个节点的编号对应于API调用，如1表示CreateFileW，5表示ExitProcess。图3(b)中的有向边表示节点之间的顺序关系，边上的属性为对应的时序值。例如，节点1与节点2之间存在2条有向边，时序属性分别为1和2。此外，对于连续出现的VirtualAllocEx调用（时序5～时序7），经过去重处理后，仅保留了时序为7的边。由此可见，与原始API序列相比，ATRG在保留时序关系的同时，有效地减少了数据规模。



(a) 原始API序列　　(b) ATRG

图3　ATRG实例

2.2.3　小结

现有的API图模型大多基于共现或调用关系构建，侧重于表征API之间的功能关联与统计依赖性。相比之下，本文提出的ATRG模型存在显著区别，其核心优势主要体现在3个方面。

1) 从静态关联到动态时序。现有模型中的边主要反映API在代码结构或统计意义上的功能关

联，而ATRG中的边则严格表示实际执行流中的逻辑先后顺序，从而有效确保行为语义的逻辑正确性。

2) 从统计摘要到轨迹保真。现有模型通常是对大规模数据集的统计抽象，不可避免地损失具体执行过程中的细粒度信息。ATRG模型则通过引入逻辑时间索引，对原始API调用序列实现高保真编码，能够准确区分细微的执行差异。

3) 从关联学习到逻辑保障。现有模型擅长识别功能组合关系，但难以约束调用的实际顺序。ATRG模型的核心优势在于其对行为逻辑的严格保障，能够精确刻画诸如“先打开后读取”等关键时序依赖，为恶意软件分析等下游任务提供更为可靠的行为建模基础。

2.3 时序随机游走算法

ATRG刻画了软件的执行行为。为了进一步提取细粒度的执行行为，需在ATRG上进行游走，生成API路径，从而表征更为细粒度的行为。当前常用的图游走算法（如Node2Vec[29]和DeepWalk[30]），通常通过随机方式遍历图的节点和边，其在同质图结构上取得了显著成效。然而，这些方法在应用于ATRG时存在根本性的理论局限。① 忽略节点间的时序依赖。传统随机游走算法将图视为静态拓扑，游走中仅考虑节点的结构邻近性，忽略了API节点间的严格时序关系。而恶意软件行为往往依赖于特定的调用序列（如“先下载再执行”），无序的游走会破坏此类关键语义，导致生成路径无法真实反映执行逻辑。② 无法表达因果与偏序关系。ATRG中的边代表执行中的先后顺序，具有明确的时序语义。Node2Vec等方法虽可调节参数以控制广度和深度遍历时的倾向，但仍无法保证游走路径符合实际API调用发生的先后顺序，因而难以捕捉到行为中的因果与条件依赖。③ 路径语义失真问题。在恶意软件分析中，API调用的顺序直接影响行为的恶意性（例如，先写入文件再设置自启动，顺序相反的行为具有完全不同的语义）。传统游走可能生成反向或无效路径，从而引入噪声，降低下游分类与检测模型的性能。

为了解决上述问题，本文设计了一种时序随机游走算法，以确保API的顺序关系得到保持，同时引入若干重要规则来选择游走的下一个节点，以保证游走的多样性。具体而言，给定当前游走的节点v_s，下一步游走目标节点v_d的选择遵循以下规则。

1) 随机采样。对于图中与当前节点v_s相邻的所有节点$N(v_s)$，将其视为潜在的候选节点。算法通过随机选择这些候选节点来生成多样化的路径，以捕捉不同类型的恶意行为模式，如式(4)所示。

$$p_r(v|v_s)=\frac{1}{N(v_s)},v\in N(v_s) \tag{4}$$

2) 全局时序递增。由于API执行具有先后顺序，算法需严格依据逻辑时间索引T递增的方向选择后续节点，即仅允许从当前节点游走到时间戳更大的邻居节点。这一机制确保了生成的路径与原始执行序列在逻辑顺序上保持一致，从而维护了恶意行为片段的真实性与完整性。具体来说，算法仅选择那些时序值大于当前节点v_s时序的目标节点v_d，以确保路径中的时间保持递增，如式(5)所示。

$$p_s(v|v_s)=\begin{cases}1, & t(v)>t(v_s)\\ 0, & t(v)\leqslant t(v_s)\end{cases} \tag{5}$$

3) 跳跃节点。恶意行为通常由多个关键API按特定顺序调用构成（即行为签名），其时序模式往往表现出短程严格顺序与长程条件触发相结合的特征。为捕捉此类模式，本文在游走过程中引入一定概率的跳跃机制（默认概率值p=0.3），在保持时序约束的前提下，允许从当前节点跳转到非直接相邻的节点，即从全局候选集合V中选择非直接邻居节点（即$V-N(v_s)$），如式(6)所示。这种机制能模拟恶意软件执行中存在的短路径依赖和长程依赖（如系统回调之间的间隔调用），从而捕捉复杂调用关系，同时增强路径的多样性，提升行为特征的表达能力。

$$p_j(v|v_s)=\begin{cases}p\dfrac{1}{|V|-|N(v_s)|}, & v\notin N(v_s)\\ (1-p)\dfrac{1}{|N(v_s)|}, & v\in N(v_s)\end{cases} \tag{6}$$

4) 循环路径回溯。在路径生成过程中，游走可能会陷入局部的循环或偏离全局路径，例如在循环的API序列片段中。算法在检测到此类问题时，将返回上一节点并继续游走，以避免路径陷入死循环，如式(7)所示。

$$p_l(v|v_s)=\begin{cases}1, & v=v_{s-1}\\ 0, & \text{其他}\end{cases} \tag{7}$$

5) 路径去重。为了避免生成过于相似的路径，算法引入路径去重机制，在已生成的路径集合 $\prod$ 中进行匹配，如果与已有路径重合则不选择该节点，减少路径的冗余性，如式(8)所示。

$$p_u(v|v_s)=\begin{cases}0, & (v_i,\cdots,v_j)\in\Pi\\ 1, & \text{其他}\end{cases} \tag{8}$$

最终，游走选择概率需在候选节点集合 $C(v_s)$ 上归一化处理，如式(9)所示。

$$P(v_{s+1}=v|v_s)=\frac{p_r p_s p_j p_l p_u}{\sum\limits_{v\in G(v_s)} p_r p_s p_j p_l p_u} \tag{9}$$

时序随机游走算法步骤如算法 1 所示。

算法 1　时序游走算法

输入　关系图 $G=<V,E,A>$，最大路径长度 L，跳跃概率 p，当前节点 v_s

输出　路径 path

1) path = { v_s }
2) t_s 为当前节点 v_s 的时间
3) for i in (1, L)
4)　$N(v_s)$=v_s 在图 G 中的邻接节点集合
5)　　if $N(v_s)$ is NULL
6)　　break
7)　W= []// 初始化时间权重列表
8)　for each v_j in $N(v_s)$
9)　　$t_j=v_j$ 的时序
10)　　$w_j=t_j-t_s$ // 当前节点的时序权重
11)　　$W=W\cup w_j$ // 放于列表中
12)　$W^*=\text{norm}(W)$ //归一化
13)　$P=\text{cal}P(V,N(v_s),W^*,\prod,p)$ // 计算 P
14)　v_d=根据 P 中的概率从 $N(v_s)$ 中选节点
15)　path = path $\cup$ v_d // 节点加入路径中
16)　$v_s=v_d$ //更新节点，从 v_d 继续游走
17)　$t_s=t_d$ //更新时间

2.4　深度学习检测模型

2.4.1　API路径的预训练及编码

在生成 API 路径后，本文采用 RoBERTa 模型[31]对这些路径进行预训练。RoBERTa 是一种改进的 BERT 模型，能够在大规模无监督语料上进行自适应预训练，从而学习上下文相关的词向量表示。本文将 API 视为“词汇”，将 API 路径视为“句子”，通过 RoBERTa 模型的预训练，学习每个 API 的嵌入向量表示。令 N 为 API 路径数量，M 为 API 数量，该模型的损失函数为 L，如式(10)所示。

$$L=-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}1_{\text{masked}}(j)\log P(w_j=w_j^*|x_i;\theta) \tag{10}$$

其中，$1_{\text{masked}}(j)$ 为指示函数，当位置 j 的 API 被掩盖时值为 1；w_j^* 表示位置 j 被遮盖 API 的真实标签；x_m 表示经随机遮盖处理后的序列；θ 表示模型中的训练参数；$P(w_j=w_j^*|x_m;\theta)$ 表示模型预测位置 j 的 API 为真实标签 w_j^* 的概率。

这种预训练方式使模型能够在大量 API 路径数据上捕捉 API 之间的复杂依赖关系和行为模式。通过这种方法，本文为每个 API 生成了高维的、上下文相关的向量表示（即 API 编码），这些编码不仅包含了 API 的语义信息，还保留了它们的时序关系。

2.4.2　基于 TextCNN 的恶意软件检测

在获得 API 编码后，本文使用 TextCNN 模型[32]对编码后的 API 序列进行训练和分类。TextCNN 是一种经典的文本分类模型，通过在不同尺寸的卷积核上进行卷积操作，能够有效捕捉序列中的局部特征。本文将编码后的 API 序列输入 TextCNN 模型中，并以恶意/非恶意标签进行监督训练。具体而言，TextCNN 通过一维卷积层提取 API 序列中的局部模式，通过池化层减少特征维度，并最终通过全连接层进行分类。在最后一层中，模型使用 Softmax 函数作为决策函数，将模型原始输出转化为概率分布，如式(11)所示。该模型能够学习 API 序列中的关键特征，从而准确识别出软件的恶意或非恶意行为。

$$P(y=j|z)=\frac{\mathrm{e}^{z_j}}{\sum\limits_{k=1}^{K}\mathrm{e}^{z_k}} \tag{11}$$

其中，$z=[z_1,z_2,\cdots,z_K]$ 为全连接层的输出向量。本文中，模型用于检测软件为恶意或非恶意，因此 K=2。

3　实验与结果分析

3.1　实验设置

3.1.1　模型设置

本文采用 RoBERTa-base 模型对 API 路径进行

预训练。该模型包含约1.25亿个参数，12层（即12个Transformer编码器块），每一层的隐藏状态维度为768，并配备了12个自注意力头。在训练中，设置batch size为32，epoch为10，learning rate为0.00 01，dropout为0.1。

在TextCNN模型中，API序列的嵌入维度同样为768维，卷积核数量设置为128，卷积核大小为5，并采用最大池化（max pooling）来缩减特征维度。优化器选择用Adam，batch size为64，epoch为20，learning rate为0.001。本文中的所有模型均使用Keras进行实现，并在Tesla A100 40G GPU上进行了训练。

3.1.2 数据集

本文使用两组数据集对模型进行评测。①课题组前期准备的数据集，命名为vVirusSign。该数据集包含12 316个恶意样本和11 276个良性样本，其中恶意样本来自恶意软件库VirusSign[33]，覆盖150个恶意软件家族，其占比最高的10个恶意软件家族如表1所示；良性样本来自NSRL[34]，皆为Windows操作系统上的合法应用程序。②CICMalDroid 2020数据集[35-36]，该数据集共包含1 795个良性样本与9 803个恶意样本，其中恶意样本涵盖1 253个广告软件（Adware）、2 100个银行木马（Banking）、3 904个短信恶意软件（SMS）以及2 546个风险软件（Riskware）。鉴于本文研究侧重于恶意软件检测，为缓解类别不平衡可能带来的模型偏差，本文从上述4个恶意软件家族中等比例抽样出1 795个恶意样本，使其与良性样本数量保持平衡。本文采用5折交叉验证的方法进行训练和测试，并计算各项指标的平均值。

表1 占比最高的10个恶意软件家族

家族	占比	家族	占比
Dinwod	21.34%	sytro	4.01%
berbew	12.10%	drolnux	3.99%
zusy	6.77%	gamania	3.70%
vobfus	5.08%	sfone	3.30%
virlock	4.91%	visel	3.29%

3.1.3 对比方法

本文将与近年来在恶意软件检测领域的代表性工作进行对比。①API-Bert模型[37]，通过从API序列中排除冗余的API调用，然后分别使用Bert和FastText模型进行恶意软件检测。② API-BiLSTM[15]，利用API序列的内在特征，使用语义链对API调用序列进行建模，并采用双向长短期记忆网络（BiLSTM）进行检测。③ DMalNet[21]，将API调用及其参数特征结合起来，按API调用的顺序构建调用图，并使用图神经网络进行检测。4) CruParamer[17]，将API调用和运行时参数一起编码，然后应用深度神经网络进行恶意软件检测。

本文采用常用的性能指标，即精确率（Precision）、召回率（Recall），以及F1分数（F1-score）来评价检测效果。

3.2 检测性能对比

表2和表3分别展示了不同模型在2个数据集上的检测性能。从表2和表3可以看出，本文方法在精确率、召回率和F1分数3个指标上均达到了最佳效果。具体而言，在vVirusSign数据集上，本文方法的精确率、召回率和F1分数分别为97.83%、98.03%和97.93%，相比于现有方法中表现最优的API-BiLSTM，分别提升了2.71%、2.76%和2.74%。此外，在CICMalDroid数据集上，本文方法在3个指标的值分别为98.26%、98.81%和98.53%，相比API-BiLSTM分别提升了0.99%、1.4%和1.19%。

表2 不同模型在vVirusSign数据集的检测性能

检测方法	精确率	召回率	F1-score
API-BiLSTM	95.12%	95.27%	95.19%
API-Bert	90.02%	92.03%	91.01%
DMalNet	94.66%	95.08%	94.87%
CruParamer	95.33%	92.23%	93.75%
本文方法	97.83%	98.03%	97.93%

表3 不同模型在CICMalDroid数据集的检测性能

检测方法	精确率	召回率	F1-score
API-BiLSTM	97.27%	97.41%	97.34%
API-Bert	93.22%	92.59%	92.9%
DMalNet	95.41%	95.17%	95.29%
CruParamer	96.72%	94.05%	95.37%
本文方法	98.26%	98.81%	98.53%

API-Bert在去重后的API序列上进行训练，但由于Bert和FastText等方法在长序列的语义学习能力方面有所不足，因此在3个指标上表现不佳。DMalNet通过针对API及其参数构建调用图，能够较好地反映程序的执行行为；然而，其直接使用图神经网络对图进行训练，未能有效捕获大规模图中的细粒度行为，因此检测性能低于本文方法。CruParamer利用API运行时参数进行安全语义归类，能够细粒度地分析恶意行为，但其在原始的长API序列上训练，难以有效捕获长序列中的语义行为。

本文方法一方面利用图模型捕捉程序的执行行为，另一方面通过时序随机游走算法提取细粒度的执行路径，并结合深度学习的预训练对这些行为进行编码，从而更好地捕获执行过程中的语义信息，最终实现了卓越的检测能力。

3.3　消融实验

本文方法中涉及3个核心优化模块，即ATRG模型、时序随机游走算法和RoBERTa预训练。为了评估每个模块的效果，本文设计了以下几种配置进行测试。①基准模型，输入为原始API序列，并使用TextCNN进行训练。②ATRG模型，使用ATRG模型对API序列进行表征，并通过图神经网络进行训练，以评估ATRG模型的效果。③ATRG+游走模型，使用ATRG模型对API序列进行表征，并利用时序随机游走算法生成路径，再通过word2vec对路径进行预训练。④本文方法，使用RoBERTa模型进行预训练，并使用TextCNN进行训练。

表4展现了不同配置下模型在vVirusSign数据集上的性能。从表4中可以看出，各种配置均带来了性能提升。例如，在F1分数方面，这3种优化分别带来了0.82%、1.9%和1.93%的提升。具体来说，相较于基准模型，使用ATRG模型使精确率、召回率和F1分数分别提升了0.55%、1.08%和0.82%，表明ATRG模型在刻画程序执行行为方面具有优势。在此基础上，时序随机游走算法进一步将3个指标提升了1.03%、2.76%和1.90%，主要归功于该算法能够捕获细粒度的恶意行为模式，同时引入的跳跃机制有助于捕捉长距离的API依赖关系。相比于word2vec，本文采用RoBERTa模型对提取的API路径进行预训练，进一步将性能提升了2.01%、1.84%和1.93%。这些性能提升充分证明了本文所提出的3个模块的有效性。

表4　不同配置的检测性能

检测方法	精确率	召回率	F1-score
基准模型	94.24%	92.35%	93.28%
ATRG模型	94.79%	93.43%	94.10%
ATRG+游走	95.82%	96.19%	96.00%
完整模型	97.83%	98.03%	97.93%

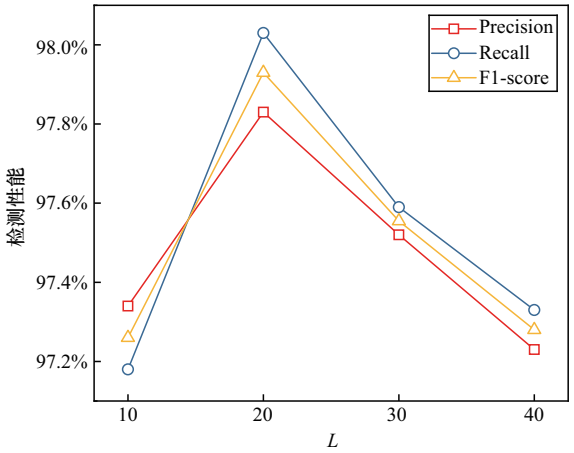
3.4　模型参数配置影响

在本文提出的模型中，有2个模块使用了超参数配置。首先，在时序随机游走算法中，本文设置了最大路径长度L（默认值为20）和跳跃频率p（默认值为0.3）。其次，在神经网络模型中，本文使用了默认的RoBERTa模型进行预训练，并使用TextCNN进行检测模型的训练，其中卷积核数量为128，卷积核大小设置为5。本文将对这些超参数分别进行评估，以分析它们对模型性能的影响。

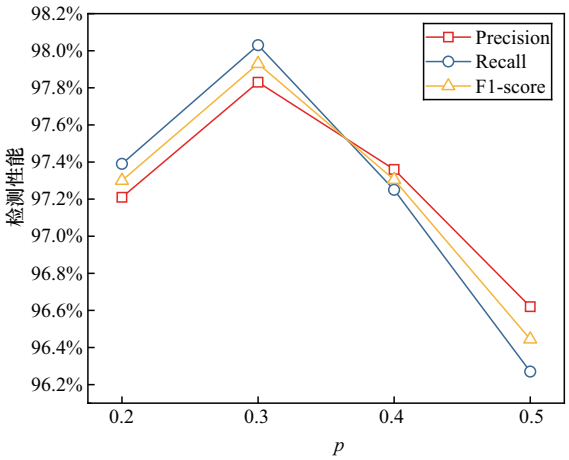
3.4.1　时序随机游走算法超参数

时序随机游走算法超参数的检测性能如图4所示。分别将L设置为10、20、30、40，检测性能如图4(a)所示，随着L的增加，精确率、召回率以及F1分数先增大后减小。例如，当L=10时，F1分数为97.26%；随着L增大到20，F1分数达到最高点97.93%。这是因为较大的L值使API路径包含更多的API调用，从而更丰富地表达了行为的语义。然而，当L进一步增大至30时，F1分数开始下降。这是由于过大的L值包含了过多API调用，可能引入与细粒度行为无关的API，从而混淆了行为模式。

本节分别将p设置为0.2、0.3、0.4、0.5进行评估。检测性能如图4(b)所示，随着跳跃频率的增加，精确率、召回率和F1分数同样先增大后减小。具体来说，当p从0.2增大到0.3时，F1分数从97.3%增大至97.93%。这是因为较高的跳跃频率使游走过程中可以探索到不直接相邻但在一定时间窗口内的API，从而增加了API路径的多样性，有助于捕捉时间跨度较大的恶意行为，提升检测性能。然而，随着跳跃频率进一步增加，游走会更频繁地探索较大时间窗口内的API，导致路径中包含了可能无关的API，从而削弱了对执行行为的准确反映，最终影响检测性能。



(a) 不同L下的检测性能



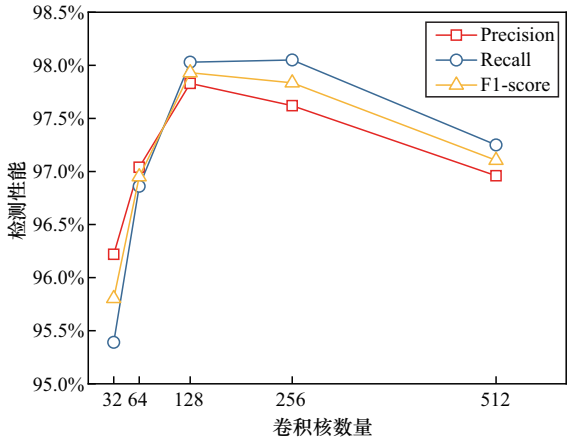
(b) 不同p下的检测性能

图 4　时序随机游走算法超参数的检测性能

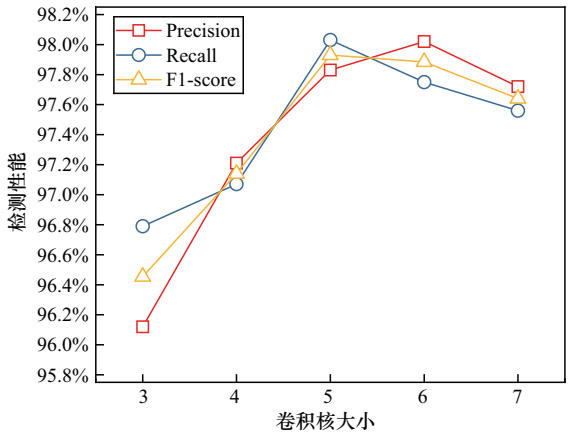
3.4.2　TextCNN 模型超参数

针对 TextCNN 模型中的卷积核数量，分别将其设置为 32、64、128、256 和 512，检测性能如图 5(a)所示，随着卷积核数量的增加，检测性能逐步提升。例如，当卷积核数量从 32 增加到 128 时，F1 分数从 95.8% 增大到 97.93%。这是因为更多的卷积核使模型能够捕捉到更丰富的特征。然而，当卷积核数量进一步增加时，性能提升则趋于停滞。例如，在卷积核数为 256 时，F1 分数为 97.83%，与 128 时几乎相同。然而，当卷积核数量增至 512 时，F1 分数反而减小至 97.1%，这是由于模型出现了过拟合，在测试集上的效果下降。基于上述实验结果，本文最终将卷积核数量设置为 128。

针对 TextCNN 模型中的卷积核大小，分别将其设置为 3、4、5、6、7，检测性能如图 5(b)所示，随着卷积核的增大，检测性能有所提升。例如，当卷积从 3 增大到 5 时，F1 分数从 96.5% 增大至 97.9%。这是因为较大的卷积核可以捕捉到更多的上下文信息和特征。然而，随着卷积核的进一步增大，尽管模型能够捕获更多的特征模式，但同时也可能忽略一些细节信息，导致模型对局部特征的敏感度降低，从而使检测性能停止提升，甚至出现下降趋势。基于该实验结果，本文最终将卷积核大小设置为 5。



(a) 不同卷积核数量下的检测性能



(b) 不同卷积核大小下的检测性能

图 5　TextCNN 模型超参数的检测性能

3.5　检测效率对比

本节实验对模型在实际应用中的检测效率进行了评估。本文统计了测试集中样本的检测时间开销，并计算了处理单个样本的平均时间。图 6 展示了不同模型的平均检测时间对比。本文方法的平均检测时间为 198 ms，其中 ATRG 构建时间为 66.3 ms，在 ATRG 上的时序游走时间为 87.8 ms，路径编码时间为 43.7 ms，检测时间为 0.2 ms。尽管本文方法的检测时间高于 API-BiLSTM 的 65 ms 和 API-Bert 的 19 ms，但考虑到实际恶意软件行为采集时

间通常为分钟级别（现有采集工具多默认设置为2 min），该时间开销相比于采集时间较小，处于可接受范围。同时，鉴于本文方法在检测性能上显著优于对比模型（例如，API-Bert的F1分数仅为91.01%，而本文方法达到97.93%），因此本文方法在实际恶意软件检测场景中具备重要的应用价值。

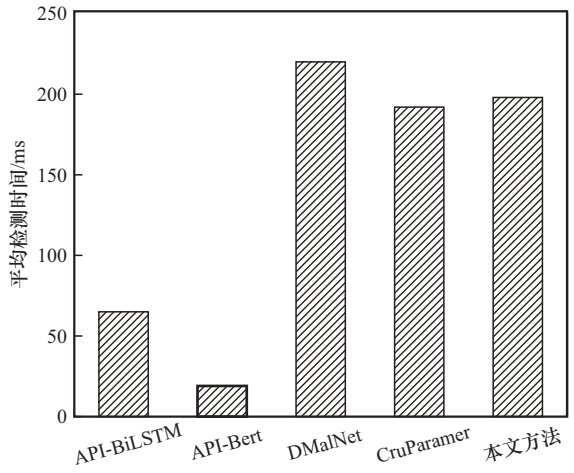


图6　不同模型的平均检测时间对比

4　结束语

本文提出了一种基于ATRG的恶意软件检测方法。该方法的核心思想是通过构建ATRG来表征原始API序列，以刻画软件的执行行为。随后，利用时序随机游走算法，提取能够描述细粒度恶意行为的API路径，并通过预训练模型对大量API路径进行处理，生成API编码。最后，采用深度学习模型进行训练，从而实现高精度的恶意软件识别。实验结果显示，本文方法在检测准确度、召回率和F1分数上分别达到了97.83%、98.03%和97.93%，优于当前最先进的方法。

参考文献：

[1] FERDOUS J, ISLAM R, MAHBOUBI A, et al. A review of state-of-the-art malware attack trends and defense mechanisms[J]. IEEE Access, 2023, 11: 121118-121141.

[2] ASLAN Ö A, SAMET R. A comprehensive review on malware detection approaches[J]. IEEE Access, 2020, 8: 6249-6271.

[3] PEKTAŞ A, ACARMAN T. Deep learning for effective Android malware detection using API call graph embeddings[J]. Soft Computing, 2020, 24(2): 1027-1043.

[4] 陈长青, 郭春, 崔允贺, 等. 基于API短序列的勒索软件早期检测方法[J]. 电子学报, 2021, 49(3): 586-595.
CHEN C Q, GUO C, CUI Y H, et al. Ransomware early detection method based on short API sequence[J]. Acta Electronica Sinica, 2021, 49(3): 586-595.

[5] 郑锐, 汪秋云, 傅建明, 等. 一种基于深度学习的恶意软件家族分类模型[J]. 信息安全学报, 2020, 5(1): 1-9.
ZHENG Y, WANG Q Y, FU J M, et al. A novel malware classification model based on deep learning[J]. Journal of cyber security, 2020, 5(1): 1-9.

[6] ABBAS M F B, SRIKANTHAN T. Low-complexity signature-based malware detection for IoT devices[C]//Proceedings of Applications and Techniques in Information Security. Berlin: Springer, 2017: 181-189.

[7] VENUGOPAL D, HU G N. Efficient signature based malware detection on mobile devices[J]. Mobile Information Systems, 2008, 4(1): 712353.

[8] YE Y F, LI T, ZHU S H, et al. Combining file content and file relations for cloud based malware detection[C]//Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2011: 222-230.

[9] WAEL D, SHOSHA A, SAYED S G. Malicious VBScript detection algorithm based on data-mining techniques[C]//Proceedings of the 2017 International Conference on Advanced Control Circuits Systems (ACCS) Systems & 2017 Intl Conf on New Paradigms in Electronics & Information Technology (PEIT). Piscataway: IEEE Press, 2017: 112-116.

[10] ZHANG F Y, ZHAO T Z. Malware detection and classification based on N-grams attribute similarity[C]//Proceedings of the 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC). Piscataway: IEEE Press, 2017: 793-796.

[11] KI Y, KIM E, KIM H K. A novel approach to detect malware based on API call sequence analysis[J]. International Journal of Distributed Sensor Networks, 2015, 11(6): 659101.

[12] HANSEN S S, LARSEN T M T, STEVANOVIC M, et al. An approach for detection and family classification of malware based on behavioral analysis[C]//Proceedings of the 2016 International Conference on Computing, Networking and Communications (ICNC). Piscataway: IEEE Press, 2016: 1-5.

[13] KIM C W. Ntmaldetect: a machine learning approach to malware detection using native API system calls [J]. arXiv Preprint, arXiv: 1802.05412, 2018.

[14] SARACINO A, SGANDURRA D, DINI G, et al. MADAM: effective and efficient behavior-based Android malware detection and prevention[J]. IEEE Transactions on Dependable and Secure Computing, 2018, 15(1): 83-97.

[15] ZHANG Z Q, QI P P, WANG W. Dynamic malware analysis with feature engineering and feature learning[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2020, 34(1): 1210-1217.

[16] KOLOSNJAJI B, ZARRAS A, WEBSTER G, et al. Deep learning for classification of malware system call sequences[C]//Proceedings of Advances in Artificial Intelligence. Berlin: Springer, 2016: 137-149.

[17] CHEN X H, HAO Z Y, LI L, et al. CruParamer: learning on parameter-augmented API sequences for malware detection[J]. IEEE Transactions on Information Forensics and Security, 2022, 17: 788-803.

[18] ROSENBERG I, SHABTAI A, ROKACH L, et al. Generic black-box end-to-end attack against state of the art API call based malware classifiers[C]// Proceedings of Research in Attacks, Intrusions, and Defenses. Berlin: Springer, 2018: 490-510.

[19] MANIRIHO P, MAHMOOD A N, CHOWDHURY M J M. API-MalDetect: automated malware detection framework for windows based on API calls and deep learning techniques[J]. Journal of Network and Computer Applications, 2023, 218: 103704.

[20] ZHANG S H, GAO M Y, WANG L H, et al. A malware-detection method using deep learning to fully extract API sequence features[J]. Electronics, 2025, 14(1): 167.

[21] BOUJNOUNI M E, JEDRA M, ZAHID N. New malware detection framework based on N-grams and Support Vector Domain Description[C]// Proceedings of the 2015 11th International Conference on Information Assurance and Security (IAS). Piscataway: IEEE Press, 2015: 123-128.

[22] 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 Press, 2020: 757-770.

[23] LI C, CHENG Z J, ZHU H, et al. DMalNet: dynamic malware analysis based on API feature engineering and graph learning[J]. Computers & Security, 2022, 122: 102872.

[24] FENG J Y, SHEN L M, CHEN Z, et al. HGDetector: a hybrid Android malware detection method using network traffic and Function call graph[J]. Alexandria Engineering Journal, 2025, 114: 30-45.

[25] FENG P B, GAI L, YANG L, et al. DawnGNN: documentation augmented windows malware detection using graph neural network[J]. Computers & Security, 2024, 140: 103788.

[26] CUI L, YIN J N, CUI J C, et al. API2Vec: boosting API sequence representation for malware detection and classification[J]. IEEE Transactions on Software Engineering, 2024, 50(8): 2142-2162.

[27] ZHEN Y W, TIAN D H, FU X H, et al. A novel malware detection method based on audit logs and graph neural network[J]. Engineering Applications of Artificial Intelligence, 2025, 152: 110524.

[28] 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.

[29] GROVER A, LESKOVEC J. Node2vec: scalable feature learning for networks[C]//Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2016: 855-864.

[30] PEROZZI B, AL-RFOU R, SKIENA S. DeepWalk: online learning of social representations[C]//Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2014: 701-710.

[31] LIU Y, OTT M, GOYAL N, et al. Roberta: a robustly optimized bert pretraining approach[J]. arXiv Preprint, arXiv: 1907.11692, 2019.

[32] KIM Y. Convolutional neural networks for sentence classification[J]. arXiv Preprint, arXiv: 1408.5882, 2014.

[33] MCCARTHY C, HARNETT K. National institute of standards and technology (NIST) cybersecurity risk management framework applied to modern vehicles[R]. 2014.

[34] CUI L, ZHU Y R, YIN J N, et al. APIBeh: learning behavior inclination of APIs for malware classification[C]//Proceedings of the 2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE). Piscataway: IEEE Press, 2024: 1-12.

[35] MAHDAVIFAR S, ABDUL K A F, FATEMI R, et al. Dynamic Android malware category classification using semi-supervised deep learning[C]//Proceedings of the 2020 IEEE International on Dependable, Autonomic and Secure Computing, Conference on Pervasive Intelligence and Computing, International Conference on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). Piscataway: IEEE Press, 2020: 515-522.

[36] MAHDAVIFAR S, ALHADIDI D, GHORBANI A A. Effective and efficient hybrid Android malware classification using pseudo-label stacked auto-encoder[J]. Journal of Network and Systems Management, 2021, 30(1): 22.

[37] YESIR S, SOĞUKPINAR İ. Malware detection and classification using fastText and BERT[C]//Proceedings of the 9th International Symposium on Digital Forensics and Security (ISDFS). Piscataway: IEEE Press, 2021: 1-6.

[作者简介]

李勇男（1984– ），男，吉林长春人，博士，中国人民公安大学副教授、博士生导师，主要研究方向为犯罪数据画像、智能情报感知、网络空间安全治理等。

赵莹（1979– ），女，山西临汾人，桂林电子科技大学副教授、硕士生导师，主要研究方向为信息科学。