计算机科学
COMPUTER SCIENCE

基于API序列特征工程与特征学习的恶意代码检测方法

杨一哲, 芦天亮, 彭舒凡, 李啸林

引用本文

杨一哲, 芦天亮, 彭舒凡, 李啸林. 基于API序列特征工程与特征学习的恶意代码检测方法[J]. 计算机科学, 2025, 52(12): 321-330.

YANG Yizhe, LU Tianliang, PENG Shufan, LI Xiaolin. Malware Detection Based on API Sequence Feature Engineering and Feature Learning [J]. Computer Science, 2025, 52(12): 321-330.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.250300056

基于 API 序列特征工程与特征学习的恶意代码检测方法

杨一哲　芦天亮　彭舒凡　李啸林
中国人民公安大学信息网络安全学院　北京 100038
(694717399@qq.com)

摘　要　基于 API 序列的恶意代码分析方法能够有效捕捉程序运行时的行为特征。然而,现有检测方法通常仅关注 API 名称,而忽略了参数以及返回值,或者难以充分挖掘它们的语义信息以及参数间的关联性,导致检测性能受限。为解决此问题,提出了一种结合系统化特征工程与深度神经网络架构的恶意代码检测方法。该方法针对 API 名称、参数及返回值的数据特性,对 API 序列实施结构化编码,继而通过多个 RefConv 卷积块来提取每个 API 调用的多尺度特征,最终将特征向量输入基于 BiGRU-BiLSTM 的并行循环神经网络,以学习 API 序列之间的长短期依赖关系。实验构建并开放了规模为 2.5 万的 API 序列数据集,在综合性能检测实验中,所提方法达到了 93.55%的准确率;并通过时间概念漂移、空间概念漂移以及消融实验,验证了所提方法可以有效检测恶意代码。

关键词:恶意代码检测;API 序列;特征工程;RefConv;BiGRU;BiLSTM
中图分类号　TP309

Malware Detection Based on API Sequence Feature Engineering and Feature Learning

YANG Yizhe,LU Tianliang,PENG Shufan and LI Xiaolin
College of Information Network Security,People's Public Security University of China,Beijing 100038,China

Abstract　API sequence-based malware analysis methods can effectively capture the behavioral characteristics of programs during runtime. However,existing detection approaches typically focus solely on API names while neglecting parameters and return values,or fail to adequately explore their semantic information and inter-parameter correlations,resulting in limited detection performance. To address this,this paper proposes a malware detection method combining systematic feature engineering with a deep neural network architecture. Specifically,the method implements structured encoding of API sequences based on the data characteristics of API names,parameters,and return values. Multiple RefConv convolutional blocks are then employed to extract multi-scale features for each API call. Finally,the feature vectors are fed into a parallel recurrent neural network based on BiGRU-BiLSTM to learn long-term and short-term dependencies within API sequences. Experiments conduct on a dataset containing 25000 API sequences,this method achieves 93.55% accuracy in comprehensive performance tests. Validation through temporal concept drift,spatial concept drift,and ablation experiments demonstrates that the proposed method can effectively detect malware.

Keywords　Malware detection,API sequence,Feature engineering,RefConv,BiGRU,BiLSTM

1　引言

在互联网高速发展的背景下,恶意代码数量呈现快速增长的趋势,且其传播方式不断更新,已经成为网络安全领域中最严重的攻击方式之一。SonicWall 2024 年年中网络威胁报告显示,基于恶意代码的威胁在 2024 年上半年激增,与 2023 年同期相比增长了 30%[1]。在此背景下,构建高效精准的恶意代码检测机制已成为维护网络空间安全的战略需求。

恶意代码检测分为两种主要检测方法:静态检测方法与动态检测方法。静态检测方法着眼于分析恶意代码本身的静态属性。部分研究依赖模式匹配来标记恶意代码,它们擅长检测已知恶意代码模式,却难以应对零日或多态恶意代码[2]。还有一些文献通过学习二进制程序的操作码[3]、函数依赖图[4]等静态方法进行研究,这样的静态分析方法无需执行代码[5],因而具备较快的执行速度。但是,静态分析方法很容易被恶意代码通过各种方式绕过,如加壳、自修改代码(Self-Modifying Code,SMC)技术和混淆技术[6]等。

动态检测方法关注恶意代码在执行过程中的行为特征。基于行为特征的检测器具备识别多态或者变形的恶意代码的能力,因为它们关注的是语义而不是句法[7]。恶意代码最终会执行一些恶意行为,如与控制器通信,下载其他恶意代码和访问权限文件[8],这些恶意行为会通过恶意代码所调用的

到稿日期:2025-03-11　返修日期:2025-06-04
基金项目:公安部科技计划项目(2023JSM09)
This work was supported by the Science and Technology Program of Ministry of Public Security(2023JSM09).
通信作者:芦天亮(lutianliang@ppsuc.edu.cn)

API(Application Program Interface)函数表现出来,因此一个软件的 API 序列包含了程序在实际执行过程中的行为特征信息,是恶意代码检测的重要特征[2,9-10]。然而,由于以下两项原因,研究高性能的恶意代码检测器仍然面临挑战。

1)研究者经常忽视 API 序列的许多有用特征。API 序列包含许多信息,如 API 名称、API 参数、API 返回值和调用时间等。但是,API 序列中各类型特征的格式繁杂,导致特征提取困难,现有的动态恶意代码检测方法更倾向于仅处理由 API 调用名称组成的序列,而忽略了 API 序列中的参数信息以及返回值信息,或者难以充分挖掘它们的语义信息以及参数间的关联性。

具体表现为:文献[11]仅考虑 API 参数的统计信息,CruParamer[12]仅将 API 序列的运行时参数作为划分 API 调用敏感性的依据,却没有深挖 API 参数间的联系。这种信息缺失,会导致检测模型的特征表达能力受限。例如,在检测恶意文件操作时,API CreateFile 的字符串参数 lpFileName 指明了要创建的文件名,若忽略该参数特征,则无法建立其与后续 ReadFile 等 API 调用参数的关联性。文献[13-14]虽然实现了对 API 参数中字符串的编码操作,但是忽略了返回值的贡献。返回值可能以句柄形式成为后续系统调用的凭证(如通过 CreateFile 返回的句柄执行 WriteFile),捕捉参数与返回值的句柄对应关系能有效跟踪资源操作链。

2)现有的公开 API 序列数据集较少,且这些数据集由于缺乏时间以及病毒种类标注,难以用于概念漂移检测(即恶意代码的特征随时间演化导致的模型性能退化问题)[15]。

为解决上述两方面的问题,本文开展了恶意代码检测模型方面的研究。主要工作如下:

1)开放了一个适用于概念漂移检测的大规模基准 API 调用序列数据集。该基准数据集收录了恶意代码的 API 调用序列及对应的调用参数信息与返回值信息。同时,每个恶意样本都有对应的病毒种类标签与年份标签,可用于评估恶意代码检测器应对概念漂移的鲁棒性。该数据集一共包含 17012 条恶意软件和 8293 条良性软件。

2)提出了对 API 序列进行结构化编码的特征工程方法。API 序列中的 API 返回值和 API 参数经常被研究者忽视,本文结合 API 序列各项部分的特点,设计了一套结构化编码方法。

3)构建了基于并行循环学习神经网络模型的特征学习方法。本文所设计的神经网络模型使用双重 RefConv[16] 卷积块来增强局部特征提取能力,部署了基于 BiGRU 与 BiLSTM 的并行循环学习模块,使模型能同时处理 API 序列的长短期时序信息。

4)探讨了所提出的恶意代码检测方法在现实世界的实际表现能力。本文设计了多项实验,分析了模型应对时间概念漂移问题、空间概念漂移问题以及综合检测的效果,并进行了详细的对比实验以及消融实验,探讨了实验结果的解释性。

2 相关工作

2.1 特征工程方法

早期研究中,特征工程方法主要聚焦于 API 序列的浅层特征提取。文献[17]提出的 MIST 框架,通过指令集提取 API 序列特征。此方法的参数特征高度依赖于专家知识,在面对新型攻击模式时存在局限性。文献[18]将 API 序列编码为字节数据,但它仅保留 MIST 的一级特征,忽略了对 API 参数的进一步处理。

近年来,针对 API 序列的特征工程有更多进展。文献[19]通过移除冗余的 API 调用并标记噪声序列来优化特征空间,但该方法仅针对 API 名称进行筛选,可能导致关键行为特征的误删。在参数处理方面,文献[14]利用不同的哈希散列方法处理 API 序列的不同部分,并考虑了字符串的统计特征。但是,仅仅使用哈希散列方法提取到的特征不够充分。在此基础上,DMalNet[13]进一步提出用相似度函数来处理字符串参数,然后将 API 序列转换为 API 调用图,用图神经网络来进行恶意代码检测。然而,DMalNet 对 API 参数的语义特征提取不够充分,同时忽略了返回值的影响。文献[20]仅提取每个 API 调用中至多 3 个参数,利用 BERT 计算语义向量嵌入,并用自编码器降维至 32 维,最终保留函数类别、函数名及参数的嵌入表示。该方法虽保留了 API 参数的核心特征,但存在语义损失风险。API2VEC++[21]创新性地采用图游走算法生成序列路径,结合 BERT 建模 API 间的联系。文献[22]通过频率阈值筛选去除参数噪声,再基于信息增益精选 3000 个高区分度参数,使用 One-Hot 编码表示参数存在性。CTIMD[23]利用 word2vec 计算 API 序列的语义嵌入,结合入侵指标和启发式规则计算参数敏感度,最终通过多层感知机融合生成增强的 API 向量。这些方法虽然实现了对 API 参数的建模,但都存在一定的语义损失问题,并且均未考虑返回值的贡献。

上述研究虽然持续推进了特征工程的演进,但在实际应用中仍存在两个短板:1)参数之间的深层联系尚未被充分挖掘;2)参数与返回值这类关键要素常被割裂对待,这正是本文设计特征工程方法时要重点突破的方向。

2.2 特征学习方法

特征学习方法主要经历了从统计学习到深度学习的演进过程。文献[24]基于 n-gram 统计 API 调用频率,采用机器学习方法进行特征学习。然而,这样的方法其实是孤立地看待不同的 API 调用的。

随着深度学习的发展,研究者提出了多种特征学习方案。文献[25]首次提出用最大池化对 API 向量进行采样,用 RNN 来提取特征的方法。此后,文献[26]提出用两个大小为 3 的 CNN 块来提取特征,用一个维度为 100 的 LSTM 网络对 API 向量进行特征学习的方法。文献[14]进一步构建了基于门控 CNN 与 BiLSTM 的深度学习模型来进行特征学习。文献[27]用 word2vec 生成 API 序列的词嵌入向量,然后用多层 BiGRU 对 API 向量进行学习。文献[22]利用全连接层学习经 One-Hot 编码的高频参数,并通过 BiGRU 与注意力机制处理 API 序列的词嵌入向量,最后将参数特征与序列特征拼接进行分类。文献[23]分别采用 TextCNN、注意力双向 LSTM 和 Transformer 3 种深度学习模型进行 API 向量的特征学习。

现有方法仍存在明显的局限性。多数研究未能系统考虑

参数与返回值的关联性，或需要单独设计参数处理模块，因此特征学习方法仍需进一步改进。

3 恶意代码检测模型

3.1 整体架构

恶意代码检测模型由 PE 文件处理模块、特征工程模块和特征学习模块组成，如图 1 所示。

PE 文件处理模块基于 Cuckoo 沙箱获取可执行文件的分析报告，并从中提取 API 调用序列。Cuckoo 是一款用 Python 编写的自动化恶意代码分析系统，采用 API Hook 技术，在 Windows 虚拟机环境下监控 PE 文件的执行过程，捕获其动态 API 调用序列及相关参数。

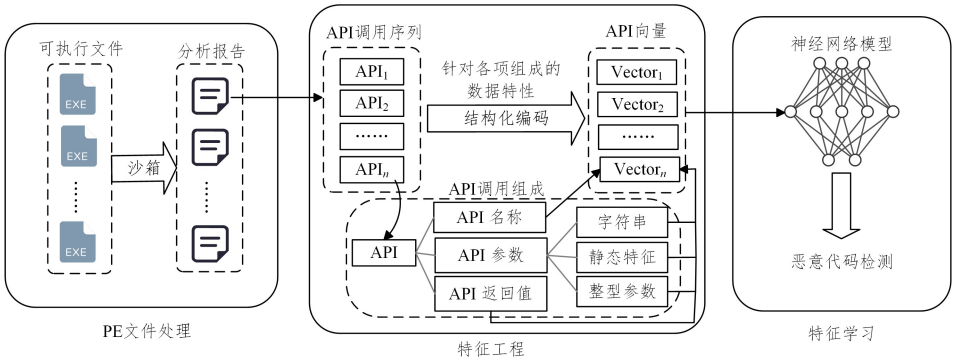

图 1 整体架构

Fig. 1 Architecture of model

特征工程模块针对 API 调用各项组成的数据特性，应用结构化编码方法对原始数据进行处理。该模块依据表 1 定义的编码规则，将长度为 n 的 API 序列转换为 $(n,162)$ 维的特征向量。

表 1 API 特征工程

Table1 API feature engineering

特征	子特征	编码维度	编码方法
API 名称	调用名称	32	去除 W/A 后缀，word2vec 计算词嵌入
API 名称	调用类别	8	One-Hot 编码
API 返回值	返回地址	16	API 关系矩阵，SVD 降维
API 参数	字符串参数	90	相似度编码，静态特征编码
API 参数	整型参数	16	特征哈希

特征学习模块通过神经网络模型对特征向量进行深入分析，从 API 调用序列中识别潜在的恶意行为模式，最终输出样本的恶意性判定结果。神经网络模型的具体结构将在 3.3 节进行进一步介绍。

3.2 API 特征工程

API 序列中的每个 API 主要由 3 部分构成，即 API 名称、API 参数和 API 返回值，如图 1 特征工程部分所示。

API 序列的不同组成成分具备不同的特性。API 名称通常具备更强的上下文联系；API 参数的上下文联系较弱，但是其提供的参数名和参数值的键值对可以提供此次调用的详细信息；API 返回值则可以作为一个动态的信息来源，提供与调用相关的即时反馈，如返回的句柄、指针或地址等。针对 API 序列的不同特征的特性，本文利用不同的方法将长度为 n 的 API 序列编码为 $(n,162)$ 形状的向量。表 1 展示了单次 API 调用所映射形成的向量的各组成部分。

3.2.1 API 名称

由 API 名称组成的序列视为一串自然语言文本，利用 word2vec 模型将这些 API 名称转换为词嵌入向量，从而为每个 API 调用生成一个固定维度的向量表示。考虑某恶意文件的 API 调用序列：

“LdrLoadDll，LdrGetProcedureAddress，LdrGetProcedureAddress，NtOpenSection，NtMapViewOfSection，RegOpenKeyExW，RegQueryValueExW，RegCloseKey，RegOpenKeyExW，RegOpenKeyExW，RegQueryValueExW，RegCloseKey，RegOpenKeyExW，RegOpenKeyExW，RegQueryValueExW。”

注意到 Windows 系统中函数名后的“A”和“W”后缀代表了该函数的 ANSI 和 Unicode 版本，不同版本的调用方式和作用几乎完全一样，如 RegOpenKeyExW 函数与 RegOpenKeyExA 函数。而 word2vec 模型（本文中基于 Skip-Gram）是将 API 调用视为独立的词来进行训练的，本文在预处理阶段去除 API 名称中的“A”和“W”后缀，这样，模型只需要关注 API 函数的核心名称，而不是它们的不同版本，从而减少了特征空间的冗余性，提高了模型的效率。通过这样的映射操作，本文将 word2vec 模型原本的 302 维特征空间降至 263 维。

malapi.io 网站提供了常见的恶意 API 的功能分类：枚举，注入，规避，间谍，互联网，反调试，勒索软件，助手。为了丰富 API 名称的语义信息，本文在提取 API 名称的向量嵌入时，也基于 API 名称对其可能的恶意应用类型进行推断，生成一个 8 维的 One-Hot 向量表示 API 类型。例如，函数 LoadLibraryA 经常被用于注入操作与规避操作，则其 API 调用类别的向量被映射为[0,1,1,0,0,0,0,0]。

通过以上 word2vec 词嵌入向量转换工作以及 one-hot 向量映射工作，可以把 API 名称映射为 40 维度的向量嵌入。

3.2.2 API 参数

1）字符串参数

API 参数中涉及的字符串种类繁多，对 API 参数中的所有字符串进行处理是不现实的。根据以往的研究，最重要的字符串参数是文件路径、DLL、注册表键值、URL 和 IP 地址的值[14]。本文通过正则表达式提取这 5 项字符串参数，利用相似度编码技术[29]对这些字段进行相似度编码[13]。相似度编码是基于相似度函数的一种编码方法，这种编码方法通过

计算共享的字符子串(n-gram)的比例,能够捕捉字符串之间的相似性。

相似度编码的原理如下:记总字符串集合为 D,C 为 k 个典型字符串的集合(在应用中,定义最频繁出现的字符串为典型字符串),$G_n(s)$为字符串 s 中所有连续的 n-gram 组成的集合。接下来计算字符串 x 的相似度编码,其中 $x\in D$,$C\subset D$。

定义一个字符串相似性函数 $sim(s_1,s_2)$,其作用是计算两个字符串之间公共 n-gram 的比例。

$$G(s)=\bigcup_{n\in\{3,4,5,6,7\}}G_n(s) \tag{1}$$

$$sim(s_1,s_2)=\frac{G(s_1)\bigcap G(s_2)}{G(s_1)\bigcup G(s_2)} \tag{2}$$

例如,对于字符串“cook”与字符串“cooker”,其对应的 2-gram 子串分别为“co” “oo” “ok”与“co”“oo” “ok” “ke” “er”,当 $n=2$ 时,两字符串共同的子串数量为 3,总子串数量为 5,此时$sim_{n=2}$(“cook”,“cooker”)=3/5。

那么,对于字符串 x,其编码向量定义为:

$$\phi(x)=[sim(x,c_1),sim(x,c_2),\cdots,sim(x,c_k)] \tag{3}$$

下面给出对 API 参数中的字符串进行相似度编码的步骤(编码维度为 16)。

(1)利用正则表达式,提取 5 种类型的字符串,如表 2 所列。

表 2 正则表达式

Table 2 Regular expression

字符串类别	正则表达式
文件路径	[a-zA-Z]:\\(?:[^\\/:*?"<>|\r\n]+\\)*[^\\/:*?"<>|\r\n]*
DLL	.+\.dll$
URL	[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,5}
注册表键值	HKEY_[A-Za-z_\.]+(?:\\[^\\/:*?"<>|\s]+)+
IP 地址	(?:[0-9]{1,3}\.){3}[0-9]{1,3}

(2)典型字符串的构建采用全局统计与局部序列调用特征相结合的方式。首先从全量数据中提取每种字符串类别出现频率最高的前 10 个字符串,同时在本条 API 调用序列内筛选出出现次数最多的前 6 个字符串。将这两个字符串集合取并集作为最终使用的典型字符串集合,以同时捕捉跨序列共性模式与当前序列特异模式。

(3)计算每次 API 调用中 API 参数中的每种字符串类别的相似度编码向量。假设对于某次 API 调用,可以提取出一个或多个 DLL 字符串,基于 DLL 典型字符串计算每个 DLL 字符串的低维编码,对所有的 DLL 字符串的低维编码进行求和,求和结果作为此次 API 调用 DLL 格式字符串的特征嵌入。

相似度编码可以捕捉字符串之间的形态相似性,但是忽略了不同种类字符串中的统计特征与语义特征。例如,IP 地址类型中公网地址与私有地址的区别,文件路径深度,字符串熵值。公网地址可能意味着恶意代码在向服务器传递信息,文件路径深度和字符串熵值有助于神经网络识别随机生成的路径文件地址或者隐藏的文件,URL 中的路径深度意味着潜在的恶意代码通信。因此,在特征提取阶段,本文不仅对目标字符串进行相似性编码,而且结合字符串的静态特征进行进一步处理。根据如表 3 所列的字符串静态特征编码方案,单次 API 调用的特征表示通过以下方式构建:对于熵值特征采用最大值计算方法,其余静态特征(包括路径深度、系统 DLL 数量等)则采用均值计算方法。

表 3 静态特征编码

Table 3 Static feature encoding for strings

类别	静态特征	编码维度
文件路径	熵,路径深度	2
DLL	熵,系统 DLL 数量	2
URL	熵,路径深度	2
注册表键值	键名熵值,敏感子键数量(RUN,Startup 等)	2
IP 地址	公网 IP 数量,特殊 IP 数量(0.0.0.0 等)	2

通过相似度函数编码与静态特征编码,将每种类型的字符串编码为 16+2=18 维度,因此 API 参数中的字符串部分最终被编码为 18×5=90 维度的特征向量。

2)整型参数

API 参数中位于用户内核地址的数值部分在 API 返回值的特征处理部分已经过处理,接下来考虑数值部分较小的参数值。同样的整数值、不同的参数名可能表示完全不同的语义。例如,参数名为 “pid”的数字 22 与参数名为 “port”的数字 22 就不同。为了解决这一问题,本文采纳文献[14]提出的思路,利用特征哈希的方法对此类参数进行编码,将参数键值对编码为维度为 16 的特征向量。

特征哈希的原理如下:假设 X 表示一次 API 调用的整数参数列表,每一个参数X^j包含两部分,即参数名X^j_{name}与参数值X^j_{value},设 M 为哈希桶的数量(也是特征向量的维度)。

下面定义两个哈希函数。

$h(X^j_{\text{name}})$:将参数名称映射到哈希桶的索引,即 0,1,…,$M-1$。

$\xi(X^j_{\text{name}})$:将参数名称映射到值±1。

其中,对于第 $i(0\leqslant i\leqslant M-1)$个哈希桶,其值$\phi_i(X)$的计算式为:

$$\varphi_i(X)=\sum_{j:h(X^j_{\text{name}})=i}\xi(X^j_{\text{name}})\log(|X^j_{\text{value}}|+1) \tag{4}$$

式(4)中,对于每个参数X^j,如果$h(X^j_{\text{name}})=i$,则将$\xi(X^j_{\text{name}})\log(|X^j_{\text{value}}|+1)$加入第 i 个哈希桶的值中。对$X^j_{\text{value}}+1$的值取对数,可以进一步压缩数值范围,处理稀疏分布的整数值。

最终经过特征哈希后,整数参数列表 X 被转换为一个特征向量:

$$[\varphi_0(X),\varphi_1(X),\cdots,\varphi_{M-1}(X)] \tag{5}$$

即每个哈希桶的计算值构成了这个特征向量。

3.2.3 API 返回值

API 序列的返回值与 API 参数中的 16 进制地址同样蕴含着丰富的信息,例如 Windows 句柄地址、字符串地址和结构地址等。然而,学习这些地址的数值信息意义不大。文献[28]认为,返回值与参数相似的 API 倾向于执行相同的任务,本文也更推荐学习相同地址之间的关系。为了给嵌入向量建立起地址间的联系,进一步为它们构建了 16 进制地址的关系矩阵,然后用 SVD 算法降维,生成每次 API 调用的返回值地址的嵌入表示。构建方法如下:

1)首先通过正则表达式筛选出范围在 0x00010000 到

0xBFFFFFFF(该范围是用户内核地址空间范围)之间的 16 进制地址。限制地址位于用户内核地址空间，能够有效排除非地址元素的影响，如状态码和端口号等。

2)为长度为 n 的 API 序列建立一个形状为(n,n)的零矩阵。第 i 行第 j 列的值表示两个 API 调用间具有相同 16 进制地址的数量，显然这是一个实对称矩阵。

3)用 SVD 奇异值分解对此矩阵进行降维处理，最终得到形状为(n,16)的 API 返回值向量。此返回值向量不仅可以捕捉返回值之间地址的联系，还能捕捉 API 参数中 API 调用间地址的联系。

3.3 API 特征学习

针对恶意代码 API 调用序列的时空特性，本文提出基于多模态特征融合的并行学习架构，如图 2 所示。

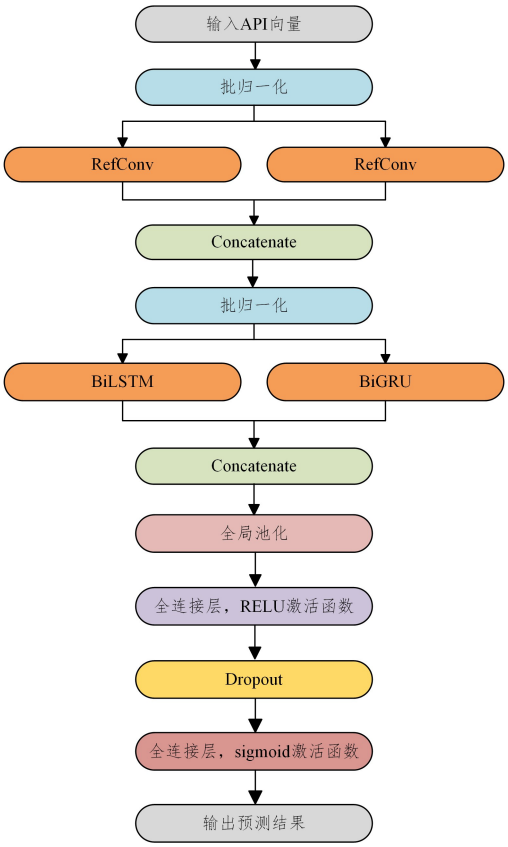

图 2 API 特征学习

Fig. 2 API feature learning

恶意代码往往通过长周期 API 组合实现隐蔽攻击(如内存驻留)，同时借助高频短序列完成敏感操作(如文件遍历)。传统单一循环网络难以兼顾长短期依赖特性，故设计 BiLSTM与 BiGRU 的并行循环学习架构。下面给出选择依据。

通过 word2vec 生成的 32 维词嵌入具有强上下文关联，BiLSTM 的长短期记忆机制可捕捉数百步的跨 API 语义组合模式(如 CreateFileW-ReadFile-WriteFile 的攻击链条)。其遗忘门可选择性保留历史状态，输出门调控信息传递强度，适合建模具有潜伏期特征的恶意行为。

参数哈希特征(16 维)与返回值矩阵(16 维)具有较高的时序敏感性。BiGRU 通过双门控机制(更新门调节历史状态保留程度，重置门控制当前输入融合强度)，能够有效捕获相邻 API 调用间的参数特征差异。例如在注册表键值修改场景中，连续调用 RegSetValueExW 时注册表参数值的动态变化，可通过重置门快速识别。

总体来看，所提出的特征学习方法首先利用双重 RefConv 卷积块解析 162 维度的结构化特征向量，其中不同的卷积核大小(1 和 3)有利于多尺度特征解析，使语义连续性特征(名称词嵌入)与参数离散性特征(哈希值、返回值)在早期卷积阶段即实现物理意义明确的分离。经批标准化处理后，两路循环网络分别处理不同性质的特征：BiLSTM 通过 150 维隐藏状态建模跨 API 的持久性行为模式(如进程注入攻击中 VirtualAllocEx 与 CreateRemoteThread 间长达数百步的间隔依赖)，BiGRU 的 150 维状态则聚焦于相邻 API 的瞬时特征组合(如勒索软件在 0.5s 内连续调用 CreateFile/WriteFile 时的参数变化频率)。二者的拼接表征完整覆盖了恶意代码攻击链的时序特性。

该设计突破了传统单循环网络的视角局限性。消融实验表明(见 4.4.2 节)，并行架构相比单一 BiLSTM 网络，F1-score 提升了 0.029，验证了多模态特征需要并行时序建模的假设。以下介绍各模块的具体设计情况。

3.3.1 输入模块

经过特征工程处理后，本文将长度为 n 的 API 序列编码为(n,d)形状的向量，这里 d(162 位) 为 API 特征的维度。接着，将向量输入批归一化模块，对输入数据进行批标准化，调整数据分布。通过对每一层的输入进行标准化处理(使其均值为 0，方差为 1)，减小了激活值的分布变化，避免了梯度爆炸或梯度消失问题。

3.3.2 双重卷积模块

这两个卷积模块使用不同的卷积核大小(1 和 3)来提取输入特征的局部模式。对于 API 名称序列(如 CreateFileW，ReadFile，WriteFile 的连续语义操作)，采用窗口大小为 3 的 RefConv 块，通过分析相邻 3 个 API 名称的关系，捕捉特定行为模式。对于参数特征，采用窗口大小为 1 的 RefConv 块，直接分析单个 API 调用的参数特征，避免多步卷积造成的特征混淆。总体来说，本文模型通过双重 RefConv 块进行多通道卷积操作，得到不同尺度的特征，从而捕捉相邻 API 序列间各组成成分间向量嵌入的关联性。以下介绍 RefConv 块的工作原理。

RefConv 块是一种经过强化的卷积核，具备更强的非线性特征学习能力[16]。RefConv 块通过重参数化机制增强卷积核的表征能力。其基于预训练模型的深度卷积基权重W_b，引入可学习的 3×3 密集卷积核W_r进行重聚焦变换，生成强化后的卷积核W_t。

$$W_t = W_b * W_r + W_b \tag{6}$$

其中，* 表示卷积操作。该设计突破了传统深度卷积的通道隔离限制，使每个输出通道的权重融合所有输入通道的基权重信息，从而建立跨通道关联，同时通过残差连接保留基权重的先验知识，确保训练的稳定性。通过消融实验，本文进一步证明了 RefConv 块对 API 序列向量的特征学习方面的促进作用。

3.3.3 并行循环学习模块

卷积模块提取的多尺度特征，经批归一化处理后，再通过 BiLSTM 与 BiGRU 进行并行时序建模。

其中，BiLSTM 分支使用 150 个隐藏单元，LSTM 网络中的门控机制以及 3 个控制门（输入门、遗忘门、输出门）设计有利于神经网络捕捉 API 序列中跨越数百步的长程依赖关系。其遗忘门可长期记忆关键 API 调用，例如在文件篡改攻击中，当检测到初始的“CreateFileW”调用后，即使间隔数百步出现“WriteFile”操作，网络仍能识别两者的关联性。输出门则通过权重调节，突出高危 API（如进程注入类调用）的语义重要性。

BiGRU 分支采用相同规模的隐藏单元，有着更简洁的更新门和重置门设计，更擅长捕捉 API 调用间的局部短期依赖关系。以勒索软件加密过程为例，当文件扩展名产生变化时，重置门会弱化之前参数的影响，快速捕获当前变化的特征。在注册表路径修改场景中，该机制可在数个 API 调用内检测到异常权限变更。

两分支输出在特征维度进行拼接，形成 300 维的复合时序表征。这种并行架构使模型既能通过 LSTM 捕获 API 序列的长期模式，又能通过 GRU 快速学习短期动态特征。

3.3.4 特征聚合模块

时序特征经最大全局池化进行空间维度压缩，保留各通道的最大激活值。该操作可以降低特征维度，减少后续全连接层的参数规模，增强模型对特征位置的鲁棒性。池化输出通过包含 96 个神经元的多层感知机进行非线性变换，使用 ReLU 激活函数增强模型表达能力，配合 50% 概率的 Dropout 层防止过拟合。

3.3.5 输出模块

最终，通过 Sigmoid 函数将特征映射到[0，1]区间，输出样本属于目标类别的概率。整个网络通过端到端的方式联合优化，损失函数采用交叉熵损失函数。

4 实验与分析

4.1 数据集构建

为了帮助研究人员研究基于 API 序列及其参数特征的恶意代码检测与分类模型，本文构建了一个大规模 API 序列数据集，并将其开源[1)]。数据集构建流程如图 3 所示。其中，开源平台指 GitHub 网站，开源数据集指 PE-ML-Dataset 数据集以及 VirusShare.com 网站提供的相关数据集。

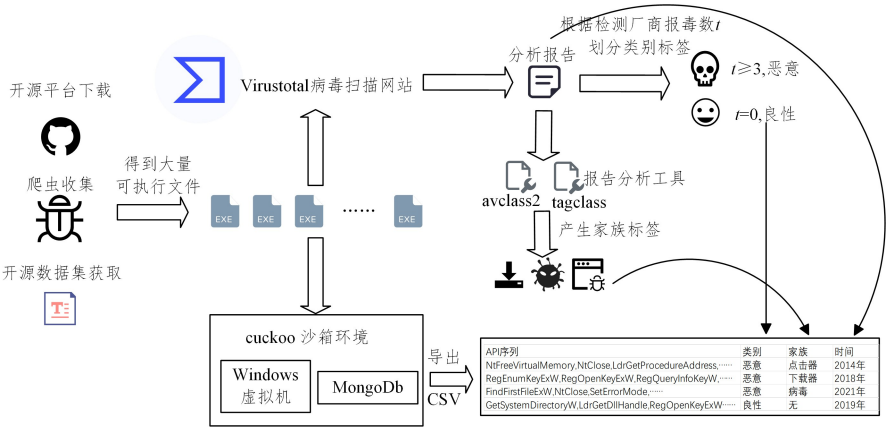

图 3 数据集构建流程

Fig. 3 Flow of dataset construction

4.1.1 可执行文件收集

所有可执行文件经 MD5 去重处理并排除沙箱运行后 API 序列长度不足 10 的文件，按来源分类处理如下。

恶意可执行文件全部来源于 VirusShare.com，收集涵盖从 2013 年到 2024 年的病毒压缩包，包括 VirusShare_00177.zip，VirusShare_x86-64_WinEXE_20130711.zip，VirusShare_00114.zip，VirusShare_00486.zip 等。

良性文件来自下列渠道：从 GitHub[2)] 获取 1458 个文件；通过爬虫技术从 portablefreeware.com 采集数据后获得 1443 个文件；从 PE-ML-Dataset[3)] 提取 3282 个文件。在研究过程中，注意到一些可执行文件虽然是从 VirusShare.com 收集的，但是它们经过 VirusTotal 扫描后其得分为零，即所有的病毒扫描引擎都不认为这些可执行文件为恶意文件。为了进一步增加良性可执行文件的多样性，本文也把此类文件添加到良性样本中。具体来源情况如表 4 所列。

表 4 可执行文件来源

Table 4 Source of executable file

类别	来源	总数量
恶意	VirusShare.com	17012
良性	github.com	1458
良性	VirusShare.com	2110
良性	portablefreeware.com	1443
良性	PE-ML-Dataset	3282

4.1.2 标签分类与评估

本文将恶意代码上传到 VirusTotal 病毒扫描网站，利用 t 阈值法，定义 t 等于 3，筛选出检测出恶意病毒的厂商数量大于或等于 3 的恶意样本，这也是学术界通用的措施[30]。将符合条件的恶意代码放在 cuckoo 沙箱软件中，设置运行时长为

1) github.com/123yonghu/api_sequence_dataset

2) github.com/tgrzinic/phd-dataset

3) practicalsecurityanalytics.com/pe-malware-machine-learning-dataset

2 min，相关研究表明，2 min 的执行时间足以让恶意代码暴露其恶意行为[31]。考虑到恶意代码的收集时间必然晚于其实际出现时间，为严谨起见，将恶意代码最早上传至 VirusTotal 网站的时间作为其时间标签。为了便于研究人员在他们的模型中评估恶意代码类型的分类效果，优先采用 avclass2[32] 辅助标记各个样本的恶意类型，同时用 tagclass[33] 标记 avclass2 不能处理的恶意代码类型，实现了每一个恶意代码与其类型标签的一一对应。具体类型分布情况如表 5 所列。

表 5 恶意代码类型分布

Table 5 Distribution of malware according to their types

种类	数量	百分比/%
Grayware	3958	23.27
Downloader	3331	19.58
Ransomware	4601	27.05
Others	1298	7.63
Exploit	1112	6.54
Clicker	834	4.90
Backdoor	675	3.97
Virus	670	3.94
Worm	533	3.13

4.1.3 平衡数据集构建

基于以上收集到的数据集特征，将整体的数据集分割为恶意和良性比例较为平衡的两部分，用于执行不同类别的任务。具体分割方式如表 6 所列。

表 6 数据集划分

Table 6 Dataset segmentation

数据集	组成	适用任务
Dataset1	10300 恶意+8293 良性	通用检测任务
Dataset2	6712 恶意+8293 良性	分类任务

Dataset1 与 Dataset2 共享相同数量的良性样本。Dataset2 包含 8 类恶意样本，具体分布如下：后门程序(Backdoor) 675 例，点击器(Clicker) 834 例，下载器(Downloader) 1 000 例，漏洞利用(Exploit) 1 000 例，灰色软件(Grayware) 1 000 例，勒索软件(Ransomware) 1 000 例，病毒(Virus) 667 例，蠕虫(Worm) 533 例。

4.2 综合性能比较

为了全面评估所提出方法的检测能力，在 Dataset1 数据集上进行实验，并采用五折交叉验证的方式进行性能评估。关于评价指标方面，本文后续的实验均采取准确率(Accuracy)和 F1-score 这两个评价指标对模型的检测能力进行评估。本文选取了以下具有代表性的恶意代码检测模型进行对比。

DMDS[14]：该方法采用特征散列技术对与 API 名称相关的 API 调用参数进行编码，并利用基于 CNN-LSTM 结构的神经网络进行特征学习。

FastText[19]：该方法首先对 API 名称进行净化处理，去除 API 调用的参数信息，然后将净化后的 API 序列输入 fastText 模型进行分类。

DMalNet[13]：该方法提出了一种混合特征编码器，从 API 名称和参数中提取语义特征。同时，该方法构建 API 调用图，将 API 调用之间的关系转换为图的结构信息，并使用图神经网络进行恶意代码检测。

在 Dataset1 数据集上的实验结果如表 7 所列。可以看出，所提出方法在准确率和 F1 分数方面均优于其他方法，表明了其在恶意代码检测任务中的有效性。

表 7 综合性能比较

Table7 Comparison of comprehensive performance

模型	准确率/%	F1 分数
DMDS	92.87	0.9344
FastText	90.97	0.9084
DMalNet	92.65	0.9327
MyModel	93.55	0.9418

结果表明，相较于现有方法，MyModel 在准确率和 F1 分数方面均实现了提升，这可能得益于，所提出的特征提取策略与神经网络模型能够更全面地捕捉 API 调用序列中的行为特征。下一步，将讨论所提出模型应对概念漂移问题的表现能力。

4.3 概念漂移问题研究

在实际应用中，恶意代码的特征分布往往会随着类型或时间的变化而发生漂移，这种概念漂移会使得模型在训练集和实际部署数据之间产生分布不一致，从而影响检测性能。本节将分别从“空间概念漂移”和“时间概念漂移”两个方面探讨该问题对各模型检测性能的影响。

4.3.1 空间概念漂移

本节采用 Dataset2 作为实验数据集来研究恶意代码类型之间的空间概念漂移问题。

实验设计采用留一类型的方式，即在每次实验中，将某一恶意代码类型作为“未知类型”从训练集中剔除后作为测试集使用，其余类型作为训练集。同时，在构造测试集时保证恶意样本与良性样本数量相同，并确保不同模型使用完全一致的测试集划分，以便对比各模型在面对未知恶意类型时的泛化能力。不同类型样本的准确率结果如表 8 所列。

表 8 不同类型样本的准确率

Table 8 Accuracy on samples from varying types

(%)

模型	Downloader	Exploit	Grayware	Ransomware	Virus	Backdoor	Worm	Clicker
DMDS	92.35	92.45	72.8	95.7	83.81	76.37	81.61	89.93
FastText	79.00	91.85	60.4	94.8	82.91	71.48	86.1	88.6
DMalNet	92.05	86.15	91.5	96.45	86.06	80.59	92.03	91.91
MyModel	93.05	93.89	91.14	96.7	86.88	84.37	93.94	93.88

通过对各模型跨类型检测能力的横向对比发现，MyModel 展现出一定的空间鲁棒性优势。具体而言：1)在 8 类未知恶意软件检测中，MyModel 在 Downloader(93.05%)和 Exploit(93.89%)等 6 种类型上取得最优结果；2)其平均检测准确率(各类权重相同)达 91.73%，较次优模型 DMalNet (89.59%)提升 2.14 个百分点。MyModel 的跨类型检测

优势，可归因于所提出的结构化编码与并行时序建模架构。对于 API 参数，返回值实施的结构化编码技术有利于模型捕捉复杂行为模式。双重 RefConv 模块及并行堆叠的 BiGRU 与 BiLSTM 模型，有助于模型高效学习 API 序列向量中的长短期依赖。

然而，在 Grayware 类型上，DMalNet 的表现略好于 MyModel，推测可能是因为 Grayware 样本的 API 特征与良性软件的 API 特征相似，DMalNet 采用的 GNN 模型更容易捕获 Grayware 样本与良性软件的不同之处。

4.3.2 时间概念漂移

除类型外，恶意代码的分布还可能随时间发生变化，形成时间概念漂移。为了评估模型对时间漂移的适应性，选取 Dataset1 进行实验，并将样本按时间划分为不同测试集。

恶意样本：测试集由 2020 年的 139 个恶意文件和 2021 年从原始的 437 个文件中随机抽样的 139 个样本构成，其余 9721 个恶意文件作为训练集。

良性样本：为保证测试集公平，从每个测试集中选取数量相同的良性文件，其余 7850 个良性文件作为训练集的一部分。

实验中，分别以 2020 年和 2021 年的恶意文件作为测试集，并保证各模型的测试集构成完全一致。表 9 列出了以不同年份恶意文件作为测试集时，各模型的准确率和 F1 分数。

表 9 不同年份样本的检测性能

Table 9 Performance on samples from varying years

模型	2020 年		2021 年	
	准确率/%	F1 分数	准确率/%	F1 分数
DMDS	91.67	0.9199	93.21	0.9243
FastText	89.13	0.8913	80.49	0.7967
DMalNet	91.30	0.9055	88.76	0.8596
MyModel	94.20	0.9394	94.02	0.9351

从表 9 可以看出，当面对时间概念漂移问题时，各模型的检测性能均有所波动。尤其是 FastText 模型，其在 2021 年测试集上的准确率和 F1 分数显著下降；而本文模型（MyModel）则能在两种测试条件下均保持较高的性能，这表明所提方法在抵抗时间概念漂移方面具备较强的鲁棒性。

4.4 消融实验

本节通过消融实验验证各模块对模型整体性能的贡献。实验分为两部分：一是对特征工程方法的消融研究，二是对神经网络结构的消融研究。所有实验均基于 Dataset1 进行，其中在消融特征工程部分，保持神经网络结构不变；在消融神经网络部分，保持特征工程方法不变。

4.4.1 特征工程消融研究

为验证各个特征模块对整体性能的影响，依次构造了以下 3 组特征输入。

仅使用 API 名称：去除冗余后，利用 word2vec 计算 32 维词嵌入，并结合推断得到的 8 维 one-hot 向量表示 API 的类型，最终形成 40 维 API 名称向量。

API 名称＋API 参数：在前一基础上增加对 API 参数的处理，包括对字符串的相似度编码、字符串的静态语义特征的提取，以及用特征哈希的方法对整型参数的编码处理。

API 名称＋API 参数＋API 返回值：在上述特征的基础上，进一步引入 API 返回值模块，通过关系矩阵进行建模，最终用奇异值分解进行降维。

实验结果如表 10 所列。

表 10 在特征工程模块上的消融研究

Table 10 Ablation study on feature engineering module

模块	准确率/%	F1 分数
API 名称	89.70	0.9344
API 名称＋API 参数	92.83	0.9084
API 名称＋API 参数＋API 返回值	93.52	0.9418

可以看出，随着特征模块的逐步加入，模型的准确率和 F1 分数均明显提升，表明各特征模块对恶意代码的行为模式均具有补充作用。仅 API 名称（纯语义特征）可以通过 word2vec 生成的 32 维词嵌入建立 API 间的语义关联，但无法识别参数的动态特征。此时模型对具有参数伪装行为的恶意样本（如修改注册表键值，但保持 API 名称不变）的检测能力有限，准确率仅为 89.70%。引入 API 参数特征后，模型能够捕捉攻击行为中参数层面的关联，从而将模型准确率提升至 92.83%。进一步增加 API 返回值特征后，通过关系矩阵建模返回值间的隐性关联（如内存申请类 API 返回的句柄被后续“WriteProcessMemory”调用所引用），使模型可追踪跨 API 的资源传递链。该模块使检测准确率最终提升至 93.52%，这验证了所提出检测方法可以学习到动态返回值信息，从而应对长周期攻击行为。

4.4.2 神经网络结构消融研究

本小节主要考查了两个关键模块的设计对模型性能的影响，即双重卷积模块的设置以及并行循环学习模块的构成。

1）双重卷积模块的消融研究

实验中对比了以下 3 种结构。

（1）仅含有一个 CNN 块：256 维的标准 CNN 块。

（2）仅含有一个 RefConv 块：使用单个大小为 1 的 RefConv 块进行多尺度卷积特征提取。

（3）将两个 RefConv 块连接起来：利用不同卷积核（大小分别为 1 和 3）提取多尺度局部模式，并进行融合。

实验结果如表 11 所列。

表 11 在卷积模块上的消融研究

Table 11 Ablation study on convolutional module

模块	准确率/%	F1 分数
仅含有一个 CNN 块	90.20	0.9002
仅含有一个 RefConv 块	92.30	0.9200
将两个 RefConv 块连接起来	93.52	0.9418

结果显示，仅使用 CNN 卷积块时，准确率较低。而当仅使用单一 RefConv 卷积块时，窗口大小为 3 的卷积核可识别 API 名称中的局部语义组合，但对参数变化的敏感性不足。窗口大小为 1 的卷积核可直接作用于参数哈希值，从而可以检测相邻 API 调用中参数值的快速变化，但无法感知跨步语义关联。随着 RefConv 块的引入与连接，模型性能逐步提升。这验证了 RefConv 卷积块在多尺度特征提取方面的重要作用，同时引入更大尺寸的卷积核可以捕捉到更多的信息，增强了神经网络的总体学习能力。

2）并行循环学习模块的消融研究

针对并行循环学习模块部分，分别测试了以下 3 种方案。

(1)仅含有 BiLSTM 网络：采用单一的双向长短期记忆网络进行时序建模。

(2)仅含有 BiGRU 网络：采用单一的双向门控循环单元进行时序建模。

(3)BiLSTM 与 BiGRU 网络并行学习：同时采用 BiLSTM 与 BiGRU 对多尺度卷积后的特征进行并行时序建模，并将两者输出在特征维度上进行拼接。

实验结果如表 12 所列。

表 12 在循环神经网络上的消融研究

Table 12 Ablation study on recurrent neural networks

模块	准确率/%	*F*1 分数
仅含有 BiLSTM 网络	92.21	0.9189
仅含有 BiGRU 网络	92.36	0.9351
BiLSTM 与 BiGRU 并行学习	93.52	0.9418

结果显示，单一 BiLSTM 网络对参数特征的建模能力较弱，因其长程记忆机制易受高频参数噪声干扰，单一 BiGRU 网络虽能捕捉相邻 API 间的参数差异(如注册表键值在短时间内的多次修改)，但难以识别跨数百步的语义依赖。使用并行循环架构可以结合两者优势，充分发挥两种网络在捕捉长短时序特征上的互补优势，从而显著提升模型的检测性能。

综上所述，通过对特征工程和神经网络结构的消融实验，本文验证了各模块对整体检测性能的正向贡献。特征工程方面，逐步增加 API 参数和 API 返回值模块能够不断增强模型的表达能力；在神经网络设计上，RefConv 块和 BiLSTM 与 BiGRU 的并行时序建模均对性能提升起到了关键作用。这些消融实验结果充分证明了，本文模型设计的合理性和有效性，为后续模型优化和改进提供了理论依据。

结束语 本文提出了一种结合系统化特征工程与深度神经网络架构的恶意代码检测方法。该方法针对 API 名称、参数及返回值的数据特性，实现了对 API 序列的结构化编码，并结合 RefConv-BiGRU-BiLSTM 并行循环神经网络，进一步学习 API 调用间的时序依赖关系。同时，为了便于研究者评估恶意代码检测器的鲁棒性，本文构建并开放了一个适用于概念漂移检测的大规模基准 API 调用序列数据集。通过时间和空间概念漂移检测，以及综合性能评估和消融分析，验证了该方法的有效性。

未来的工作将聚焦于以下 3 个方面：1)探索进程间 API 调用的关联建模方法；2)扩展基准数据集对传统机器学习方法的兼容性测试；3)研究基于增量学习的动态部署方案，提升模型对新型恶意代码的持续适应能力。

参 考 文 献

[1] SonicWall. 2024 Mid-Year Cyber Threat Report [EB/OL]. (2024-08-30) [2024-12-01]. https://www.sonicwall.com/resources/white-papers/mid-year-2024-sonicwall-cyber-threat-report.

[2] AMER E, ZELINKA I. A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence[J]. Computers & Security, 2020, 92: 101760.

[3] KAKISIM A G, GULMEZ S, SOGUKPINAR I. Sequential opcode embedding-based malware detection method[J]. Computers & Electrical Engineering, 2022, 98: 107703.

[4] YAN J, YAN G, JIN D. Classifying malware represented as control flow graphs using deep graph convolutional neural network [C]// 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks(DSN). IEEE, 2019: 52-63.

[5] GOPINATH M, SETHURAMAN S C. A comprehensive survey on deeplearning based malware detection techniques[J]. Computer Science Review, 2023, 47: 100529.

[6] DAMODARAN A, TROIA F D, VISAGGIO C A, et al. A comparison of static, dynamic, and hybrid analysis for malware detection[J]. Journal of Computer Virology and Hacking Techniques, 2017, 13: 1-12.

[7] GAO Q Q, SHI Z B, QIN Y M, et al. Interpretable malicious code detection method based on API sequence[J]. Computer Engineering and Design, 2023, 44(6): 1642-1648.

[8] CUI L, CUI J, JI Y, 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. 2023: 261-273.

[9] WANG P, TANG Z, WANG J. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling [J]. Computers & Security, 2021, 106: 102273.

[10] 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.

[11] AHMED F, HAMEED H, SHAFIQ M Z, et al. Using spatio-temporal information in API calls with machine learning algorithms for malware detection[C]// Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence. 2009: 55-62.

[12] CHEN X, HAO Z, 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.

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

[14] ZHANG Z, QI P, WANG W. Dynamic malware analysis with feature engineering and feature learning[C]// Proceedings of the AAAI Cconference on Artificial Intelligence. 2020: 1210-1217.

[15] GUERRA-MANZANARES A, LUCKNER M, BAHSI H. Concept drift and cross-device behavior: Challenges and implications for effective android malware detection[J]. Computers & Security, 2022, 120: 102757.

[16] CAI Z, DING X, SHEN Q, et al. Refconv: Re-parameterized refocusing convolution for powerful convnets[J]. arXiv: 2310.10563, 2023.

[17] TRINIUS P, WILLEMS C, HOLZ T, et al. A malware instruction set for behavior-based analysis[C]// Sicherheit 2010. Sicherheit, Schutz und Zuverlässigkeit. Gesellschaft für Informatik eV, 2010: 205-215.

[18] QIAO Y, YANG Y, HE J, et al. CBM: free, automatic malware

analysis framework using APIcall sequences[C]// Knowledge Engineering and Management: Proceedings of the Seventh International Conference on Intelligent Systems and Knowledge Engineering(ISKE 2012). Springer, 2014: 225-236.

[19] YESIR S, SOĞUKPINAR İ. Malware detection and classification using fasttext and bert[C]// 2021 9th International Symposium on Digital Forensics and Security (ISDFS). IEEE, 2021: 1-6.

[20] WONG G W, HUANG Y T, GUO Y R, et al. Attention-based API locating for malware techniques[J]. IEEE Transactions on Information Forensics and Security, 2023, 19: 1199-1212.

[21] CUI L, YIN J, CUI J, et al. API2Vec++: Boosting API Sequence Representation for Malware Detection and Classification [J]. IEEE Transactions on Software Engineering, 2024, 50(8): 2142-2162.

[22] ZHOU B, HUANG H, XIA J, et al. A novel malware detection method based on API embedding and API parameters[J]. The Journal of Supercomputing, 2024, 80(2): 2748-2766.

[23] CHEN T, ZENG H, LYU M, et al. CTIMD: Cyber threat intelligence enhanced malware detection using API call sequences with parameters[J]. Computers & Security, 2024, 136: 103518.

[24] UPPAL D, SINHA R, MEHRA V, et al. Malware detection and classification based on extraction of API sequences[C]// 2014 International Conference on Advances in Computing, Communications and Informatics(ICACCI). IEEE, 2014: 2337-2342.

[25] PASCANU R, STOKES J W, SANOSSIAN H, et al. Malware classification with recurrent networks[C]// 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015: 1916-1920.

[26] KOLOSNJAJI B, ZARRAS A, WEBSTER G, et al. Deep learning for classification of malware systemcall sequences[C]// AI 2016: Advances in Artificial Intelligence: 29th Australasian Joint Conference. Springer, 2016: 137-149.

[27] DAI Y, LI H, QIAN Y, et al. SMASH: A malware detection method based on multi-feature ensemble learning[J]. IEEE Access, 2019, 7: 112588-112597.

[28] SALEHI Z, SAMI A, GHIASI M. MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values[J]. Engineering Applications of Artificial Intelligence, 2017, 59: 93-102.

[29] CERDA P, VAROQUAUX G, KÉGL B. Similarity encoding for learning with dirtycategorical variables[J]. Machine Learning, 2018, 107(8): 1477-1494.

[30] ZHU S, SHI J, YANG L, et al. Measuring and modeling the label dynamics of online {Anti-Malware} engines[C]// 29th USENIX Security Symposium (USENIX Security 20). 2020: 2361-2378.

[31] KÜCHLER A, MANTOVANI A, HAN Y, et al. Does every second count? time-based evolution of malware behavior in sandboxes[C]// NDSS 2021, Network and Distributed Systems Security Symposium. Internet Society, 2021.

[32] SEBASTIÁN S, CABALLERO J. Avclass2: Massive malware tag extraction from av labels[C]// Proceedings of the 36th Annual Computer Security Applications Conference. 2020: 42-53.

[33] JIANG Y, LI G, LI S. TagClass: A tool for extracting class-determined tags from massive malware labels via incremental parsing[C]// 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2023: 193-200.

YANG Yizhe, born in 2001, postgraduate, is a member of CCF (No. Z0786G). His main research interests include malware detection and so on.

LU Tianliang, born in 1985, Ph.D, professor, Ph.D supervisor. His main research interests include cyber security and artificial intelligence.

(责任编辑:李亚辉)