基于集成学习技术的恶意软件检测方法

李 芳[1,2], 朱子元[1,2], 闫 超[1], 孟 丹[1,2]

[1] 中国科学院信息工程研究所 北京 中国 100093
[2] 中国科学院大学 网络空间安全学院 北京 中国 100049

摘要 近年来, 低级别微结构特征已被广泛应用于恶意软件检测。但是, 微结构特征数据通常包含大量的冗余信息, 且目前的检测方法并没有对输入微结构数据进行有效地预处理, 这就造成恶意软件检测需要依赖于复杂的深度学习模型才能获得较高的检测性能。然而, 深度学习检测模型参数量较大, 难以在计算机底层得到实际应用。为了解决上述问题, 本文提出了一种新颖的动态分析方法来检测恶意软件。首先, 该方法创建了一个自动微结构特征收集系统, 并从收集的通用寄存器(General-Purpose Registers, GPRs)数据中随机抽取子样本作为分类特征矩阵。相比于其他微结构特征, GPRs 特征具有更丰富的行为特征信息, 但也包含更多的噪声信息。因此, 需要对 GPRs 数据进行特征区间分割, 以降低数据复杂度并抑制噪声。本文随后采用词频-逆文档频率 (Term Frequency - Inverse Document Frequency, TF-IDF)技术从抽取的特征矩阵中选择最具区分性的信息来进行恶意软件检测。TF-IDF 技术可以有效降低特征矩阵的维度, 从而提高检测效率。为了降低模型复杂度, 并保证检测方法的性能, 本文利用集成学习模型来识别恶意软件。实验表明, 该集成学习模型具有 99.3%的检测准确率, 3.7%的误报率, 优于其他现有方法且模型复杂度低。此外, 该方法还可以用于检测真实数据中的恶意行为。

关键词 恶意软件检测; 通用寄存器; 集成学习; 词频-逆文档频率
中图法分类号 TP309.5 DOI 号 10.19363/J.cnki.cn10-1380/tn.2024.01.10

Malware Detection Method Based on Ensemble Learning Technology

LI Fang[1,2], ZHU Ziyuan[1,2], YAN Chao[1], MENG Dan[1,2]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract In recent years, low-level hardware microarchitecture features have been widely used for malware detection. However, most of the microarchitecture features contain a large amount of redundant information, and current detection methods do not effectively preprocess the input micro-architecture data, which results in complex deep learning model to obtain high malware detection performance. However, the deep learning detection model has a large number of parameters, thus it is difficult to be practically applied at low-level hardware of computers. To solve these problems, we propose a novel dynamic analysis method to detect malware. First of all, this method creates an automatic microarchitecture feature collection system, and randomly extracts the sub-samples from the collected General-Purpose Registers (GPRs) data as the classification feature matrix. Compared with other microarchitecture features, GPRs features have much richer behavioral characteristics, but also contain noise information. Therefore, we divide the GPRs data into different intervals to reduce data complexity and inhibit noise. Then, we adopt Term Frequency-Inverse Document Frequency (TF-IDF) technique to select the most discriminative information from these matrices, for malware detection. TF-IDF technology can effectively reduce the dimension of the characteristic matrix and improve detection efficiency. In order to reduce the complexity and ensure the performance of the detection method, this paper uses ensemble learning model to identify malware. Experimental results indicate that the detection accuracy of the ensemble learning model is 99.3% with 3.7% false positive rate, which is better compared with other existing methods, and the complexity of our proposed model is lower. Besides, our method can also achieve higher detection rate in real data.

Key words malware detection; general-purpose registers; ensemble learning; term frequency-inverse document frequency

通讯作者: 朱子元, 正高级工程师, Email:zhuziyuan@iie.ac.cn。
本课题得到中国科学院战略先导专项(No. XDC02010400), 国家重点研发计划(No. 2018YFB2202100)的资助。
收稿日期: 2020-04-10; 修改日期: 2020-08-11; 定稿日期: 2022-12-20

1 引言

1.1 研究背景及意义

世界经济论坛《全球风险报告》[1]指出, 目前全球每天有超过 50%的人使用互联网。尽管数字技术为全球大多数人带来了巨大的经济利益以及社会利益, 但是它也有着潜在的风险。数字化前沿全球风险感知调查的两组受访者(多利益相关方和全球杰出青年)在未来十年内前十大长期风险(科技、生态环境及社会等)列表中确定了与网络攻击相关的问题。其中科技类中的网络攻击则被评为 2020 年第五大风险, 可见保证网络安全的重要性。

众所周知, 恶意软件是网络安全的主要威胁之一, 其可以被定义为一系列程序或者代码的集合。恶意软件在未经许可或用户不知情的情况下入侵计算机系统并故意实施攻击, 例如破坏计算机功能、破坏数据窃取私密信息等一系列的恶意行为。自计算机诞生以来, 恶意攻击的威胁就存在。但是早期的恶意软件设计是以了解计算机的真正潜力为目的, 而不是对计算机系统, 计算机网络以及用户进行伤害、窃取或操纵。因此, 这段时间的恶意软件其实并没有恶意。现代恶意软件越来越趋向于一种组织完善的活动, 它具有自己的供应链, 形成完整的地下经济。它既可以是大型地下组织用来赚钱的工具, 也可以是政府用于间谍和攻击的武器。例如针对普通日常计算机的恶意软件, 它可以窃取银行和信用卡信息(直接窃取金钱), 收集电子邮件地址(出售给垃圾邮件发送者)或远程控制计算机等。然而, 恶意软件的主要威胁不仅仅是针对一般的计算机, 而是可以针对特定的公司或政府。这些恶意软件旨在促进盗窃贸易或国家机密, 窃取关键信息(例如敏感电子邮件)或攻击基础设施。例如, Stuxnet 这种恶意软件, 旨在攻击和破坏伊朗的各种核设施[2]。由于许多恶意软件是开源的, 混淆工具也非常容易获得, 这就造成了恶意软件数量成指数级增长。例如, AV-TEST[3]最新统计报告显示, 每天约有 35 万多个新型恶意软件和潜在有害的应用程序被注册。到 2020 年 2 月 9 日, 恶意软件总量超过 1021.19 百万(图 1 (a)), 其中仅 2019 年新生成的恶意软件总量就接近 144.9 百万(图 1 (b))。传统的商业恶意软件检测工具都是基于恶意软件标签实现恶意软件检测, 虽然这些方法能够快速识别已知标签的恶意软件, 但是对数据库中不存在的恶意软件及新型恶意软件都不能实现有效地防御。因此, 鉴于恶意软件发生频率高、危害严重的特点, 有效的恶意软件识别对计算机安全, 人身财产安全甚至国家安全都有着重要的意义。

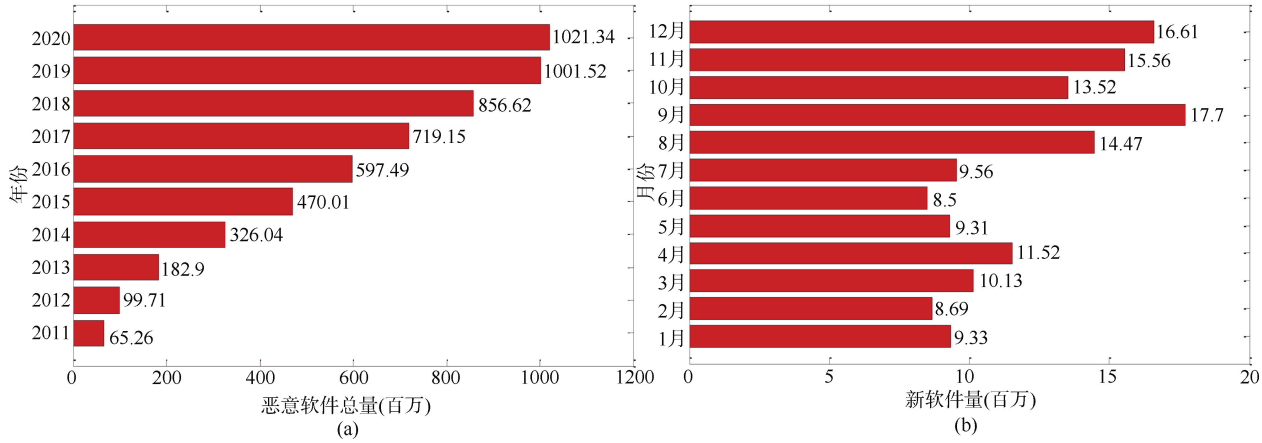

图 1 恶意软件统计: (a)恶意软件总量; (b)新型恶意软件量

Figure 1 Malware statistics: (a) total malware; (b) new malware

1.2 基于微结构特征实现恶意软件检测成为重要手段

按照所分析的特征, 目前的恶意软件检测方法主要分为两大类: 高级别特征与微结构特征。其中, 高级别特征主要从计算机系统结构软件层进行收集。经过验证能够有效实现恶意软件识别的特征主要有应用程序接口(Application Programming Interface, API)顺序[4], 控制流图(Control Flow Graph , CFG)[5]等。最近, 一些研究者开始使用低级别的微结构特征实现恶意行为检测。由于微结构特征不仅可以获得恶意行为与良性行为之间更细微的行为信息差别, 而且可以清晰化恶意行为与良性行为之间的模糊边界。此外, 它还有助于提高恶意行为检测的灵敏度[6-7]。基于微结构特征的检测技术并不需要等待微结构特征被转换成高级别特征就可以直接利用微结构特征本身实现对非法行为的识别。微结构特征的获取主

要是从计算机体系结构的硬件结构层中抓取行为信息。虽然计算机底层数据繁杂, 但是 Khasawneh 等人[8]以及 Ozsoy 等人[9]从计算机底层收集了多种微结构特征, 并使用机器学习技术及神经网络技术证明了操作码(Operation code, Opcode)最大频率差异特征具有优异的检测潜能。同时, Demme 等人[10]首次验证了硬件性能计数器(Hardware Performance Counters, HPC)数据也可以用来进行恶意行为检测。虽然以上方法能够使用微结构特征实现恶意软件检测, 但是依旧存在以下问题:

- **如何选择合适的微结构特征:** 选择具有区分性的微结构特征对检测性能起到决定性的作用。但是, 计算机微结构层存在多种微结构特征。例如, Ozsoy 等人[9]对比了三类微结构特征: 基于一致性指令的特征, 基于内存地址模式的特征及基于架构事件特征。仅基于指令的特征就至少包含指令类别频率、Opcode 最大频率差异、Opcode 类别及 Opcode 序列四种微结构特征。该方法通过对比三类特征, 最后提出 Opcode 最大频率差异在恶意行为检测时具有最好的区分性能。然而, Opcode 最大频率差异特征只是简单的对比了恶意样本及良性样本之间存在的频率差异, 并没有考虑重要的语义信息。而且, 作者实验所使用特征的依赖环境较为简单, 这就造成在复杂环境下, 采用 Opcode 最大频率差异特征不能获得令人满意的检测性能。
- **如何去除冗余信息:** 在选择好所依赖的微结构特征后, 还会面临冗余信息的干扰。在计算机底层, 短时间内会产生大量的微结构特征。仅针对 Opcode 特征, 一秒时间就会从计算机底层收集上千万条行为记录信息。但是, 这些数据不仅包含恶意行为信息, 还会包含合法行为信息。由于恶意软件本身就包含合法代码[10]。于是所收集的微结构特征数据中的冗余信息不仅会干扰恶意行为检测效率还会影响恶意行为的检测性能。目前, 基于微结构特征实现恶意软件检测的方法都没有提出有效降低特征中冗余信息的方法。如图 2(a)[9]所设计的检测方法, 它并没有提出抑制冗余信息的措施, 而是直接将收集到的微结构特征作为检测模型的输入。在微结构特征分析时进行冗余信息的去除是一种有效提取特征的措施, 但是图 2(a)[9]与(b)[11]都是实时、连续、盲目地使用从处理器流水线中抽取的所需微结构特征, 而并没有做有效的预处理。

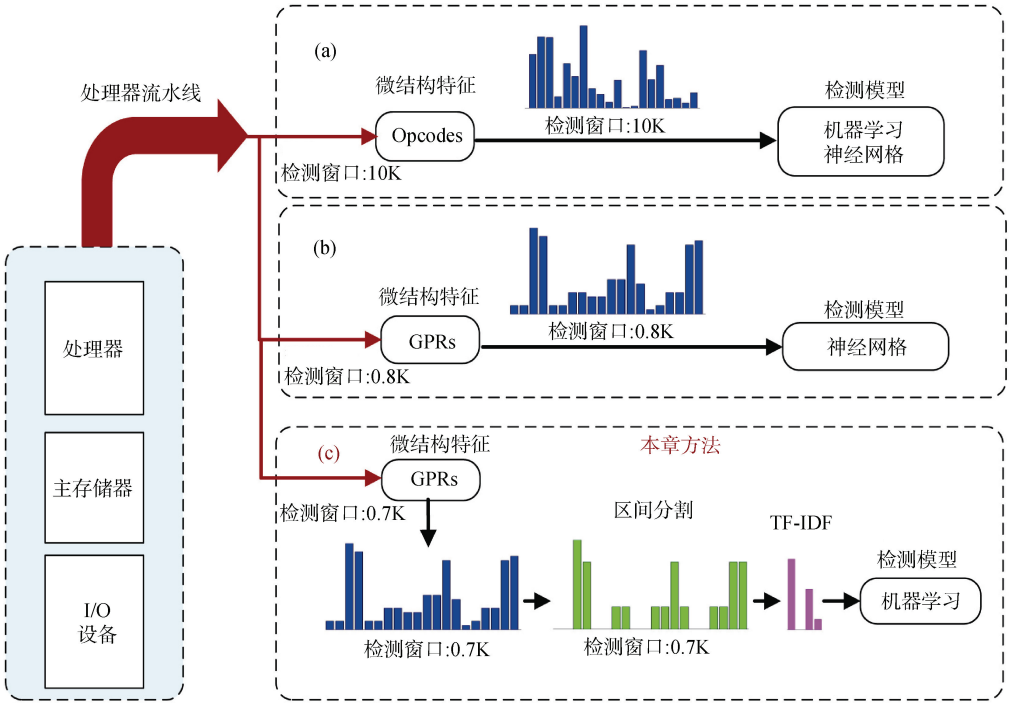

图 2 基于不同微结构特征的恶意行为检测

Figure 2 Malware detection based on different micro-architecture features

- **如何缩短检测窗口:** 检测窗口的大小也会影响恶意软件检测模型的效率及性能。但是, 目前基于微结构特征实现恶意软件检测的方法都依赖于较长的检测窗口。例如, Ghiasi 等人[12]

提出的检测方法需要分析整个样本在执行过程中所产生的全部微结构特征, 这种方式需要恶意软件完全执行完毕后才能实现有效地检测, 而且只能用于事后分析。Ozsoy 等人[9]及 Khasawneh 等人[8]所提出的方法需要分析连续 10K 指令的微结构特征(如图 2(a))。对于 Opcode 最大频率差异特征来说, 10K 指令的长度对检测模型计算量影响不大(经过统计 Opcode 出现频率差异后并排序, 仅选择前 35 个 Opcodes 用于最终的模型输入), 但是对于其他微结构特征, 10K 指令的数据量可能会严重影响模型的检测性能。而且, 即使选择 Opcode 最大频率差异作为检测特征, 也只能等待 10K 指令信息被全部收集完毕才能实行检测, 这就不可避免地影响模型的检测灵敏度。

- **如何降低模型复杂度:** 在图 2(b)中, Li 等人[11]提出了一种利用 GPRs 特征的深度学习技术检测方法, 它在实现较高检测率的同时, 又可以只依赖较短的检测窗口(0.8K)。 但是, 由于该方法具有很高的模型复杂度, 因此很难进行实际应用。事实上, 采用微结构特征实现恶意行为检测是为了更好的硬件实现, 以降低检测过程对处理器的影响。因此, 有必要通过减少微结构特征的数据冗余来达到简化检测模型的目的。

为了解决上述问题, 本文选择 GPRs 微结构特征用于恶意软件检测, 因为该特征与其他微结构特征相比具有更丰富的行为信息。例如, 图 3 展示了当一条 Call 指令从文件中读取 10 个字节(依赖于“*kernel32.ReadFile*”函数)时, GPRs 内容的变化。其中, 标记为红色的数据表示 GPRs 更改后的值。可以发现, GPRs 内容发生了显著变化, 这表明 GPRs 的特征对细粒度的行为敏感。此外, GPRs 微结构特征易于检测未知恶意软件, 因为在逃避检测时, 变异的恶意软件通常不会更改特定点的值(标志和常量)[12]。

E8 7D000000				CALL <JMP.&kernel32.ReadFile>												
Registers(FPU)																
EAX	0012FF78	0012FF78	0012FF78	0012FF78	0012FD4C	0012FD4C	0012FF78	00000034	00000000	00000000	00000000	00000000	00000000	00000000	00000001	00000001
ECX	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	76E2316F	74FAABCD	7598DB02
EDX	00260180	00260180	00260180	00260180	00260180	00260180	00260180	00260180	00260180	00260180	00260180	00260180	00260180	00260180	76E164F4	76E164F4
EBX	7FFD7000	7FFDF000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000	7FFDB000
ESP	0012FD64	0012FD60	0012FD5C	0012FD58	0012FD30	0012FD30	0012FD30	0012FD30	0012FD30	0012FD2C	0012FD28	0012FD24	0012FD20	0012FD1C	0012FD30	0012FD60
EBP	0012FF84	0012FF84	0012FF84	0012FF84	0012FD5C	0012FD5C	0012FD5C	0012FD5C	0012FD5C	0012FD5C	0012FD5C	0012FD5C	0012FD5C	0012FD5C	0012FD5C	0012FF84
ESI	012FD78	012FD78	012FD78	012FD78	012FD78	012FD78	00000034	00000034	00000034	00000034	00000034	00000034	00000034	00000034	00000034	0012FD78
EDI	00000000	00000000	00000000	00000000	00000000	0012FF78	0012FF78	0012FF78	0012FF78	0012FF78	0012FF78	0012FF78	0012FF78	0012FF78	0012FF78	00000000

图 3 单个微结构事件 GPRs 数据变化

Figure 3 GPRs data changes of single micro-architecture event

为了实现有效的恶意软件检测, 本文提出了一种动态恶意软件检测技术。与 Li 等人[11]将收集到的 GPRs 微结构特征直接作为模型输入(见图 2(b))不同, 本文对复杂的 GPRs 特征设计了一种有效的预处理方式。首先, 使用特定的检测窗口从 GPRs 中随机提取特征矩阵。但是即使检测窗口较小, 特征矩阵仍具有大量 GPRs 内容(例如, 32 位系统 GPRs 微结构特征数据范围是从“ 0x00000000”到“ 0xffffffff”)。因此, Li 等人[11]才需要依赖复杂的神经网络技术实现恶意软件检测。与图 2(b)不同, 本文将 GPRs 特征数据平均划分到一定数量的区间进行降维, 然后再进行恶意行为检测。该数据降维过程不仅可以保留数据多样性, 降低数据的复杂度, 还可以抑制噪声的干扰。此外, 为了进一步抽取有价值的特征信息, 本文首次将 TF-IDF 技术应用到 GPRs 特征上, 从 GPRs 微结构特征矩阵中抽取重要的特征信息。最后, 本文还构建了用于恶意软件检测的集成学习模型, 该模型极大地减少了检测模型的参数。此外, 本文还验证了该集成学习模型对真实微结构特征数据检测的有效性。本文主要有三点贡献:

GPRs 特征矩阵包含大量的数据信息, 如果直接将其应用于恶意软件检测, 则需要使用复杂的检测模型才能获得较高的检测性能。因此, 本文提出对 GPRs 进行特征区间分割, 以降低数据复杂性并抑制数据噪声。

将 GPRs 特征值划分到不同的区间后, 本文使用 TF-IDF 技术从特征矩阵中提取关键信息数据, 以降低数据维数。如图 2(c)所示, 从 0.7K 指令检测窗口减小到仅 503 个关键字长度。

基于 TF-IDF 技术构建了一个集成学习模型来检测恶意软件。该模型不仅具有较高的检测性能, 还可以有效地识别未知恶意软件。此外, 本文还验证了该方法可以用于检测真实数据中的恶意行为。

2 相关工作

本小节将详细介绍和阐述目前已有的恶意软件检测方法, 总体来说根据检测方法主要分为三类: 特征匹配技术, 机器学习技术和神经网络技术, 而检测特征可以分为两类: 高级别特征和微结构特征。

2.1 特征匹配技术

特征匹配技术是一种快速识别恶意软件的技术, 使用特征匹配实现恶意软件检测的基本流程如图 4 所示。首先, 检测引擎对待检测的可执行文件进行相关分析, 并获得该文件的特征码。然后, 该特征码与检测引擎检索库中的特征集进行匹配, 如果发生匹配项, 则该检测引擎对该检测文件发出恶意警报。若未发现匹配项, 则对该文件进一步分析。如果发现该文件为恶意软件, 则将该文件的特征码加入检索库以对特征集更新。其中特征码是唯一代表文件的序列, 该序列可以是文件的哈希值, 字节或指令, 而传统的恶意软件检测引擎几乎都是依赖特征匹配技术。因此, 当恶意软件随着时间不改变自己的特征标签时, 商业检测引擎是可以准确的识别该软件。但是, 当恶意软件制造者对恶意软件进行一些简单的混淆变化, 例如增加无效代码, 改变代码结构等, 该文件的特征码就会随之改变。即使简单混淆操作并没有改变恶意软件的功能, 还是能够绕过检测引擎的检测[13]。

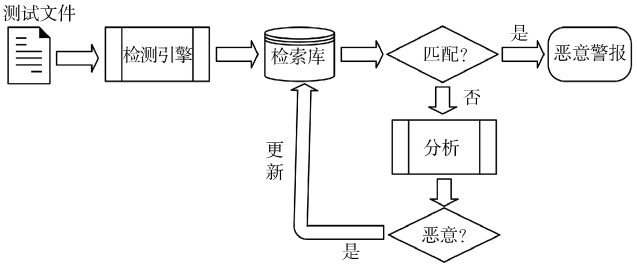

图 4 基于特征匹配技术基本流程

Figure 4 Basic flow based on feature matching technique

Christodorescu 等人[14]认为传统的模式匹配方式只考虑了语法的信息匹配, 而忽略了指令的语义信息。于是, 提出了一种合并指令语义的措施来改进匹配策略。该方法将恶意软件的恶意行为形式化表示, 因此当某些恶意行为以不同的形式出现时, 它就可以被发现。上述方式可以抵御黑客使用的常见混淆技术。后来, Christodorescu 和 Jha 等人[13] 认为想要检测混淆的恶意软件, 检测引擎必须首先消除恶意软件作者所使用的混淆变换。因此设计了一种检测可执行文件中恶意模式的体系结构, 该体系结构在进行匹配前可以抵消常见的混淆转换。后来, 研究者们认为提取有价值的特征来对恶意样本生成特征标签是非常重要的, 于是 Zhang 和 Reeves [15]提出了一种利用恶意软件所执行的系统调用特征来实现恶意软件检测。他们认为系统调用可以表征代码片段的语义或功能, 同时还考虑了为系统调用所需要的参数指令。为了有效判断待检测可执行文件是否恶意, Zhang 和 Reeves[15]提出了一种模式匹配的方法来判断语义的相似性。Ye 等人[16]首次提出采用 API 调用来实现恶意行为检测, 因为 API 特征序列可以反映程序的语义行为信息。该方法首先使用可执行文件解析工具抽取文件中的 API 信息(如果可执行文件被第三方工具进行压缩或者嵌入了自制的打包程序, 则需要先对其进行解压缩, 然后再使用可执行文件解析器)。然后, 将 API 在 API 查询数据库中对应的整数 ID 作为可执行文件的签名存储在标签库中。随后, 使用面向目标的关联挖掘算法生成类别关联规则, 并将该规则记录在规则数据库中。为了最终确定可执行文件是否恶意, Ye 等人[16]将 API 调用与生成规则一起传递给恶意软件检测模块, 以执行分类。

以上方式都是基于静态检测技术, 众所周知静态检测技术需要对被分析文件反汇编, 而这就需要大量的人工分析, 费时费力。因此 Kolbitsch 等人[17]提出了一种动态检测方法, 该方法首先在隔离环境中分析恶意程序以构建行为模型, 该模型表示了恶意行为在执行任务时所必须的系统调用之间的信息流。然后, 作者使用信息流的程序片与待检测软件程序进行匹配。实验证明该方法可以有效的在用户主机上实现恶意行为检测。但是, 分析信息流是一种复杂的步骤, 因此 Park 等人[18]提出了利用系统调用图进行恶意软件分类的方法。该方法首先捕获了可执行文件在执行时的系统调用信息及系统调用参数信息。然后, 计算待检测可执行文件与已知恶意软件的相似性。由于同一个恶意家族共享相同或相似的行为, 因此每个恶意家族都有显示相似特征的趋势[19], 例如系统修改、产生新的进程, 注册表修改及网络通信等。随后, Park 等人[20]提出了一种通用行为图的检测方法, 该方法针对一个恶意家族的所有恶意软件实例只使用一个代表性的图, 而不是对每个实例使用一个行为图。由于大多数新恶意软件都是已知恶

意家族的变体, 因此该方法是可行的[21]。

以上方法都是在计算机系统层或应用层使用高级别特征对恶意软件进行识别。对于微结构特征, Leder 等人[22]提出使用值集作为特征实现恶意软件检测。他们认为即使混淆后的恶意软件改变了代码结构, 但是某些恶意行为固有的常数值不会改变。作者首先从寄存器、内存、堆栈等位置抽取相关内容; 然后将抽取的内容利用值集的方式进行表达。最后, 使用相似性匹配来实现恶意软件检测。在 Leder 等人[22]的基础上, Ghiasi 等人[12] 仅使用寄存器的内容就实现了恶意软件的检测。该方法使用相关工具抽取程序在 API 发生时前后寄存器的内容, 根据内容计算文件之间的相似性来实现恶意行为检测。但是, 该检测方法需要分析整个样本的特征信息, 因此只能用于事后分析。

2.2 机器学习技术

编写软件是一个很难的问题, 无论恶意软件还是良性软件都是如此。它不仅仅是对软件算法的设计与实现, 还需要考虑软件在实现与执行过程中各种因素的影响。例如, 如何根据运行的网络环境或空闲时间做出何时传播的决策。因此, 恶意软件作者通常在创建新的恶意软件时重复利用源恶意软件的代码以及代码的模式。这就代表着相关恶意软件之间存在固有的模式和相似性, 因此恶意软件检测者利用这一弱点将恶意软件检测看成一个分类问题, 它可以学习二进制代码文件中的模式, 以对未知恶意软件进行分类。使用机器学习方法实现恶意软件检测的整个过程可以分为两个阶段: 训练阶段与测试阶段, 如图 5 所示。在训练阶段(图 5 (a)), 首先将数据集中的样本随机抽取一部分样本作为训练集(剩下的样本为测试集), 训练集包括恶意样本及良性样本。然后, 对数据集中的每个样本进行解析, 抽取每个样本的特征构成向量集。最后将特征向量作为机器学习分类算法的输入实现分类器的训练。其中分类算法包括决策树(Decision Tree , DT)及支持向量机(Support Vector Machine , SVM)等。接下来, 在测试阶段(图 5 (b)), 使用在训练阶段学习好的分类模型对测试集的恶意样本及良性样本集进行分类。同训练阶段一样, 对测试集中的每个样本进行解析并获得特征向量集。利用此特征向量集, 分类器将测试集分类为恶意或者良性。在测试阶段, 模型将测试集中文件的真实类别与分类器分类的类别进行比较, 通过相应的评价标准来度量评估生成分类器的性能。

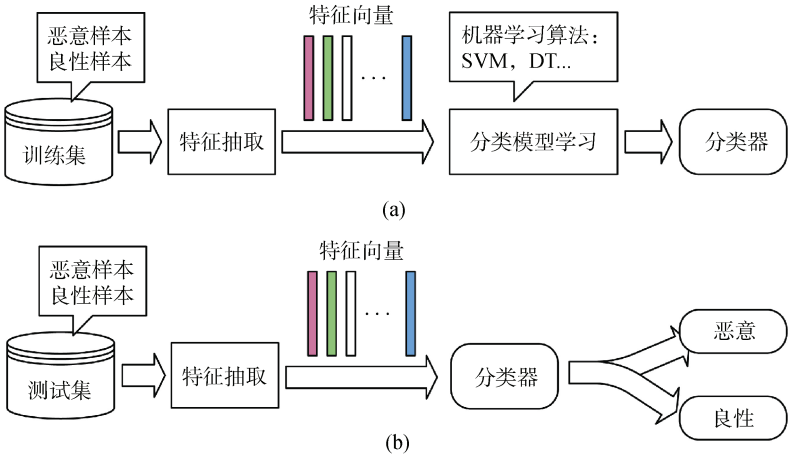

图 5 机器学习检测的训练及测试

Figure 5 Training and testing phases of Machine Learning classifiers

Schultz 等人[23]首次提出了利用机器学习技术来实现恶意软件检测。使用机器学习技术进行恶意行为检测的第一步是确定可执行文件的表示形式, 该方法使用了三种不同的特征: 字节序列 *n*-gram, 可移植的可执行文件(Portable Executable , PE)特征与字符串特征。其中字节 *n*-gram 特征是从可执行文件中提取的 *n* 字节序列, 而 *n*-gram 已被广泛应用于语言建模与语言识别领域[24-26]。可执行文件中的 *n* 字节序列与文本分类问题中的单词语法一致, 因此 *n*-gram 有信息的重叠, 它不仅仅可以捕获有关长度字符串的统计信息, 也可以获取长字符串的频率信息。PE 特征是从 WIN32 可执行文件(EXE 或 DLL)中抽取的某些重要结构, 该结构可以表明此文件已被创建或被感染的恶意行为信息。例如 PE 头信息, 它描述了可执行文件的物理结构(创建时间、修改的时间、文件大小等), 而字符串特征指程序文件中编

码的纯文本字符串。Schultz 等人[23]设计了四个分类器进行对比实验, 分别为基于签名分类, 基于规则 RIPPER 算法的分类, 基于朴素贝叶斯(Naive Bayes, NB) 分类及基于多重朴素贝叶斯(Multi-Naive Bayes)分类。实验结果表明使用机器学习技术可以更有效地实现恶意软件检测。

Bilar[27]首次提出了使用 Opcode 特征来实现恶意软件检测, 该方法研究了恶意代码与良性代码之间的 Opcode 频率分布, 并发现恶意代码与良性代码中相同的 Opcode 出现频率具有明显的差异。后来, Dolev 和 Tzachar[28]首次提出了利用 Opcode 序列来实现恶意软件检测。由于完整的机器语言包含一个 Opcode 及一个或多个操作数, 而 Opcode 是机器语言的一部分, 因此可以用 Opcode 序列表示可执行文件。基于 Dolev 和 Tzachar[28]的方法, Moskovitch 等人[29]从反汇编的可执行文件中抽取 Opcode 序列, 并使用多个机器学习算法设计对比实验。实验结果显示 2-gram 的 Opcode 序列能够获得更好的性能。并且, 与 NB, 增强贝叶斯(Boost NB) 等其他传统机器学习算法相比, 增强决策树算法(Boost DT)能够获得更好的检测性能。为了避免将有意义的 Opcode 序列分成多个子序列, Zhao 等人[30]仅考虑部分控制流信息(基本块)中的 Opcode 序列, 但是不考虑基本块之间的执行顺序。在 Zhao 等人[30]所提出的方法中, 作者对比了决策树(J48), 自主聚合(Bagging)及随机数(Random Forest , RF)三种算法, 最终验证 RF 算法能够获得更高的检测性能。但是, 该方法没有充分考虑控制流的信息, 因此会导致抽取的 Opcode 序列无法准确地描述原始可执行文件的行为。于是, Ding 等人[31]提出了一种利用 Opcode 序列就可以完全表示可执行文件行为的方法。该方法首先将 Opcode 分为了三类: 顺序 Opcode, 条件分支 Opcode 及无条件分支 Opcode。接着, 根据 Opcode 的类型将程序分为基本块, 并按照顺序对每个基本块进行编号。最后, 将程序的基本块按照一定的数据结构进行存储并构造程序的 CFG。为了更准确全面地使用 Opcode 序列所表达的程序行为, 该方法遍历了 CFG 以获取所有的可执行路径。其中 Opcode *n*-gram 使用固定长度的滑动窗口进行提取。文中实验对比了三种分类器, 分别为 K 最近邻(KNN, k-Nearest Neighbor), 决策树(C4.5)以及 SVM。最终实验证明 C4.5 分类器能够获得最高的检测性能。与 Opcode 类似, Chan 和 Song[32]结合 API 调用特征与安卓恶意软件权限特征利用机器学习技术进行恶意软件检测, 他们的实验对比了 NB, SVM, DT 及 RF 等分类器, 最终结果表明 RF 能够获得更高的检测性能, 而且增加 API 调用信息有助于识别安卓恶意软件。

对于使用机器学习的动态检测技术, 分析从隔离环境中收集的所有特征数据是没有必要的。实际上可执行文件在执行过程中产生的行为特征同时包含恶意行为信息以及合法行为信息。因此分析所有的特征不仅会降低检测方法的检测性能, 还会影响检测效率。于是, Uppal 等[33]利用 API 特征实现恶意软件检测。与 Sathyanarayan 等人[34]分析所有收集到的 API 特征不同, Uppal 等人[33]从 API 数据集中抽取有区分性的 API 序列, 利用该序列进行恶意软件检测。首先, 该方法捕捉可执行软件在执行过程中所有的 API 调用数据。然后, 从收集的 API 调用信息中抽取 API 序列特征。最后, 使用 gram 及优势比技术提取具有有效区分恶意行为及合法行为的 API gram 作为分类的输入。作者对比了 NB, RF, DT, SVM 4 种传统机器学习技术, 最后证明 SVM 能够获得最优的检测性能。后来, Kawaguchi 和 Omote[35]提出了一种新的基于 API 特征动态检测方法。该方法认为在动态收集恶意行为信息过程中无法准确估计样本执行所需要的时间, 于是其设计的分类模型仅需要分析样本执行前 90s 产生的 API 信息。文中实验对比了 SVM, C4.5, RF, NB 以及 KNN 五种经典机器学习算法, 最后证明利用前 90s 的特征信息可以有效实现恶意软件分类, 其中 RF 检测结果最优。

在微结构特征级别, Malone 等人[36]与 Xia 等人[37]将 HPC 数据特征应用到异常检测的问题上, 并取得了有效的检测结果。于是, Demme 等人[10] 提出了利用 HPC 特征数据来实现恶意软件检测。该方法收集样本在执行过程中产生的多维 HPC 数据, 并利用机器学习技术实现恶意软件检测。但是该方法的最高检测准确率只有 90%, 检测准确率低的主要原因可能是检测器直接对所收集到的数据进行分析而未进行有效地数据预处理, 因此数据包含了大量噪声。为了进一步降低 HPC 特征的维度, Sayadi 等人[38]采用主成分分析(Principal Component Analysis , PCA)技术从收集的特征数据中选择最佳的 HPC 特征数据。为了提高检测性能, Yerima 等人[39]利用集成学习技术构建最终的恶意软件检测模型。同 Sayadi 等人[38]提出的方法类似, Khasawneh 等人[8]也利用集成学习技术设计恶意软件检测模型, 但是该方法主要验证了 Opcode 特征在计算机微结构级别的检测性能。虽然 Sayadi 等人[38]与 Khasawneh 等人[8] 能够利用微结构特征并使用机器学习技术实现有效的恶意软件检测, 但是这两种方法都需要依赖过长的检测窗口。

2.3 神经网络技术

近年来, 神经网络技术在计算机视觉及自然语言处理领域得到了广泛应用并大放异彩[40-41]。深度学习是一种特殊的人工神经网络, 用于学习输入与输出之间的隐藏关系, 多层神经元以不同的权重和激活函数相互连接。简单来说, 将输入数据送入网络第一层, 在经过第一层卷积后再经过激活函数作为第二层的输入, 以此类推构成简单的神经网络。不同层之间的链接权重是根据后向传播进行调整的, 其由预测标签与真实标签之间的距离决定。

同机器学习方法检测恶意软件过程一样, 利用神经网络技术来设计恶意软件检测模型也分为两个阶段: 训练阶段与测试阶段(如图 6)。神经网络技术能否应用到恶意软件检测领域, 主要取决于如何将收集到的特征转换成检测模型的输入。Saxe 和 Berlin[42]从可执行文件中抽取字节/熵直方图特征、可执行文件导入特征、二维字符串直方图特征以及可执行文件元数据四种类型的特征作为神经网络检测模型的输入。为了将以上特征转换成模型可识别的数据类型, 作者分别进行了特定的处理。例如对于可执行文件导入特征, 作者初始化了一个 256 长度的数组, 并将二进制文件的 DLL 名称及导入函数的每个元组散列到[0,255] 范围内。最后, 作者将以上特征数据向量进行集成作为分类模型的最终输入。检测模型为四层的前馈神经网络(Feedforward Neural Networks, FNNs)。实验证明该方法可以有效地实现恶意软件检测。Kolosnjaji 等人[43]将可执行文件元数据、可执行文件导入数据及 Opcode 数据作为分类特征。为了提升检测性能, Kolosnjaji 等人[43]采用一种混合方法来设计神经网络结构, 该结构分为卷积网络(Convolutional Neural Networks, CNNs)与 FNNs 两部分。其中可执行文件元数据及可执行文件导入数据由 FNNs 进行分析, 而 Opcode 特征由 CNNs 进行分析, 最后将两个网络的输出进行集成实现恶意软件检测。实验结果证明该方法优于 FNNs 模型和 SVM 模型。基于灰度图的检测方式则不需要从可执行文件中抽取相关特征信息, 而是直接将可执行文件转成灰度图并输入到神经网络中。例如 Gibert 等人[44]认为将恶意软件可视化为灰度图像, 这有利于保留文件全局结构, 并可以捕获不同恶意家族之间的细微差异。于是, 在 Nataraj 等人[45]的基础上, 他们利用 CNNs 来实现恶意软件检测, 并且实验证明了该方法与其它基于灰度图的技术相比具有更高的分类准确率。与 Nataraj 等人[45]的方法不同, Yen 和 Sun [46]认为将可执行文件直接转成图像作为神经网络模型的输入会造成错误的判断。于是, Yen 和 Sun[46]首先从反汇编后的样本中抽取重要性的单词, 接着将重要的单词打包成多个, 最后, 利用 djb2 算法将生成的组数据转换成彩色图像后送入 CNNs 模型进行检测。实验结果证实该方法优于传统的机器学习方法[47]。Chen 等人[48]则选择 word2vec 技术从收集到的多种特征中抽取更有价值的信息, 实验证明了使用 word2vec 能够更好地表征恶意软件行为信息, 其检测精度提高了 1 至 2 个百分点。此外, 该方法还与多个流行的恶意软件检测引擎(ESET, Kaspersky, Kingsoft, McAfee, 360, Symantec, Tencent 等)进行了对比, 进一步验证了它的有效性。Yan 等人[49]则提出了一个利用长短期记忆网络(Long Short-Term Memory, LSTM)的分层降噪网络来实现恶意软件检测。由于 LSTM 擅长处理具有时序特征的数据, 而恶意软件的 Opcode 序列同样具有时序行为信息, 因此使用 LSTM 对 Opcode 进行恶意软件检测是可行的。但是, 由于梯度消失问题, 大多数的 Opcode 序列对于 LSTM 来说长度过长。因此 Yan 等人[49]提出了分层结构。第一层 LSTM 对 Opcode 子序列进行计算来学习密集表示。第二层 LSTM 则可以利用该表示进行恶意软件检测。考虑到恶意行为仅出现在部分 Opcode 序列中出现, 该方法还设计了降噪策略以提升检测性能。最后实验结果表明该方法能够捕获较长的 Opcode 序列特征, 同时也可以获得优异的检测性能。

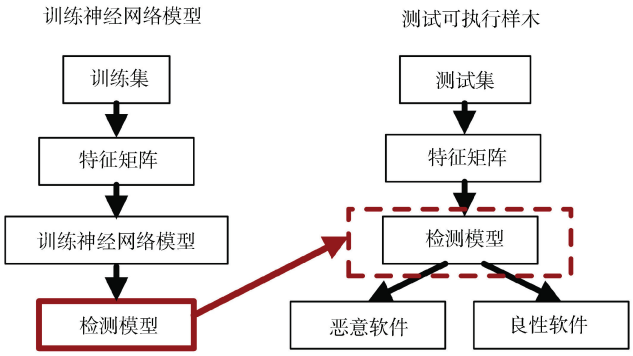

图 6 神经网络检测的训练及测试

Figure 6 Training and testing phases of neural network classifiers

以上检测方法都是基于静态检测, 而学术界和工业界越来越关注采用动态分析来自动执行对恶意软件的检测。基于 API 特征, Maniath 等人[50]提出使用 LSTM 网络来实现恶意软件检测。该方法认为在进程执行时应用程序接口调用列表可以视为单词序列, 而该单词序列也具有丰富的时序信息, 因此提

出使用 LSTM 网络构造检测模型。Maniath 等人[50]首先从改良版的 cuckoo 隔离环境中抽取样本执行时的 API 信息, 同时将收集的时间延长到 20min 以防止某些样本包含延迟执行策略。然后, 从收集到的 API 特征集中滤除不重要的 API, 并把保留的 API 序列用整数进行标记。最后, 将设置好的 API 序列作为LSTM网络的输入, 以实现恶意软件检测。但是, 该方法仅限制于对勒索软件的有效检测。与 Maniath 等人的方法[50]不同, Liu 和 Wang [51]去除从 cuckoo 沙箱中收集的冗余 API 信息, 然后利用自然语言处理技术 word2vec 对 API 特征进行向量化, 最终使数据集中的每一个 API 都被分配唯一一个二进制向量, 该向量长度固定为 50。接着, 他们选择使用双向 LSTM 而不是传统的 LSTM 构造检测模型。由于双向 LSTM 同时考虑了过去的特征(由正向过程提取)和将来的特征(由反向过程提取), 实验证明双向 LSTM 检测模型优于传统的 LSTM 模型。对于抽取特征的向量化, Abderrahmane 等人[52] 利用独热编码技术将收集到的系统调用数据生成独立的二进制矢量。为了学习相邻系统调用之间的相关性, 该方法将连续多个向量化的系统调用数据以矩阵方式送入检测模型, 而检测模型是基于 CNNs 设计的。实验对比了其他动态检测技术, 并证明该方法可以有效实现恶意软件检测。Kolosnjaji 等人[53]同样将收集到的系统调用特征向量化, 但是该方法构建的检测模型为 CNNs 网络与 LSTM 网络两种网络结构的结合。与其他传统的机器学习技术相比, 该方法可以有效提升恶意软件检测性能。Xiao 等人[54]利用 LSTM 网络设计了一个新颖的检测模型。该模型包含两个并行 LSTM 网络结构, 其中一个 LSTM 输入数据为恶意样本行为信息数据, 另外一个 LSTM 输入数据为良性样本行为信息数据。这个模型通过比较得分来判断待检测样本是否为恶意的。此外, 该检测方法将收集到的系统调用顺序看做自然语言中的句子来进行特征向量化。实验对比了 Xiao 等人[54]与 Xiao 等人[55]两种检测方法, 并证明了该检测模型能够获得更好的检测性能。

在微结构特征级别, Ozsoy 等人[9,56]从计算机底层直接抽取三类微结构特征: 基于一致性指令的特征, 基于内存地址模式的特征及基于架构事件的特征。在将上述三类特征进行向量化后, 分别利用机器学习技术和神经网络技术实现恶意软件检测模型的构建。通过利用机器学习技术与神经网络技术对三类微结构特征的比较, 证明了 Opcode 最大频率差异特征对恶意软件检测更有效, 并且实验也说明了神经网络技术的检测性能优于机器学习技术。接着, 为了进一步提升检测性能, Khasawneh 等人[8]提出了集成学习策略来实现恶意软件检测。该方法分别设计了通用分类器、通用集成分类器、专用集成分类器及混合分类器(通用分类器集成专用分类器)四类分类模型进行对比。实验依旧表明与传统机器学习技术相比, 神经网络技术能够获得更高的检测性能。此外, 采用集成学习技术来实现恶意行为检测可以提升检测准确率并降低误报率。

以上利用微结构特征实现恶意软件检测的方法大都忽略了微结构特征的冗余信息, 并需要依赖较大的检测窗口, 例如Ozsoy 等人[9,56]及Khasawneh 等人[8]都需要直接分析 10K 指令的微结构特征。虽然 Li 等人[11]将检测窗口降低到0.8K, 但是需要依赖复杂的神经网络模型才能保证优异的检测性能。与上述方法不同, 本文使用 TF-IDF 技术降低 GPRs 特征的冗余信息, 并利用集成学习技术设计恶意软件检测模型, 该模型不仅可以依赖较低的检测窗口(0.7K)获得较高的检测性能, 还可以有效识别未知恶意软件。

3 基于集成学习技术的恶意软件检测

为了缩小检测窗口、降低特征冗余、提高检测效率和性能, 本文提出了基于集成学习技术的恶意软件检测方法。如图 7 所示, 该检测模型主要分为三个部分: 微结构特征收集, 微结构特征预处理以及集成检测模型设计。

3.1 微结构特征收集

本文方法利用GPRs微结构特征进行恶意软件检测, 而如何从CPU中收集恶意样本与良性样本在执行过程中所产生的行为信息至关重要。基于动态分析的恶意软件检测技术都是将待分析的可执行文件在隔离环境中进行执行, 因为直接将恶意软件在物理机中执行而不隔离会对计算机系统及网络造成恶劣的影响。通常动态检测的方法需要指定可执行文件在隔离环境中的运行时间, 并收集在此执行时间段内待分析样本产生的所有行为信息以便后续处理。手动地依次执行每个样本, 并且等待所有可执行样本执行完毕需要浪费大量人力及时间。虽然静态分析技术只需要几秒就可以完整的反汇编一个可执行样本获得所需数据, 但是面对加壳与加密的可执行样本需要事先进行加密与解密。而且, 一但面对无法解包与解密的可执行文件, 就只能通过动态分析技术才能实现对样本的有效分析。近年来, 新生成的恶意软件变得越来越高级且复杂, 已经无法对其进行有效地解包及解密。此外, 对复杂样本的解包与解密也需要花费大量的人力与时间。

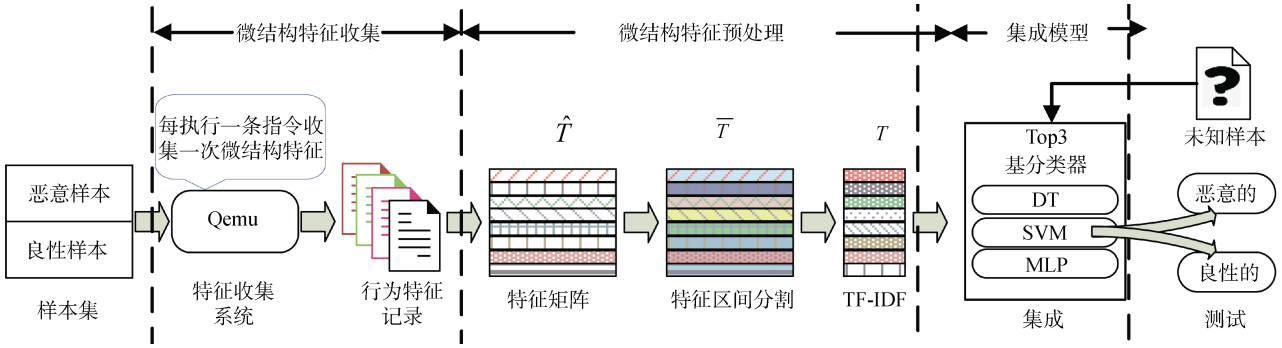

图 7　基于集成学习技术的恶意软件检测

Figure 7　Malware detection based on ensemble learning technique

因此, 本文创建了一个自动微结构特征收集系统, 如图8所示。该收集系统的输入为可执行恶意样本与良性样本集, 输出为待分析的GPRs, Opcode等微结构特征。近年来, 虽然有许多动态行为信息收集系统被提出[57-58], 但是它们大都被用来收集高级别特征。此外, 在收集微结构特征上, 它们存在一些缺陷, 例如Okane等人[59]设计的隔离环境需要手动设置与操作。而且, 目前存在的隔离环境大多数都是用来提取Opcode特征, 无法收集GPRs特征。因此, 本文设计了一个可以自动收集GPRs特征的隔离环境。该环境主要依赖改进的Qemu[60]模拟器来实现微结构特征的收集。在这里, Qemu是基于动态二进制翻译的开源仿真器, 可以直接通过修改相应的Qemu源代码来获取所需的信息。为了高效准确地从Qemu 模拟器中收集到微结构特征, 本文在不修改处理器微码的情况下对其增加微结构特征保存模块。该模块可以直接从CPU中无间断的收集各种运行过程中的特征信息。

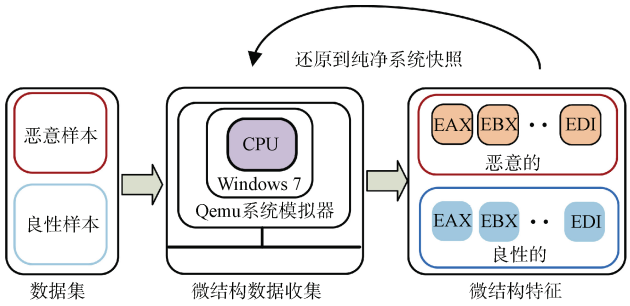

图 8　微结构特征自动收集系统

Figure 8　Automatic micro-architecture feature collection system

文中提出的自动微结构收集系统创建在虚拟机64 位的Ubuntu 16.04 操作系统上。随后, 使用Qemu模拟器在该操作系统上模拟标准计算机环境(32 位Windows 7)。为了支持恶意软件操作, 本文在此Qemu上禁用了防火墙与Windows 安全服务, 并将其连接到网络。自动微结构数据收集系统的使用基本步骤如下:

第一, 将数据集的恶意样本与良性样本文件复制到虚拟机并依次排列以供顺序执行。

第二, 为了避免先前恶意样本在执行完成后带来潜在影响, 本文为隔离环境创建了纯净的系统基准快照。

第三, 修改后的Qemu可以自动从数据集中加载一个可执行样本并执行它。在这里, 每执行一条指令就收集一次微结构特征信息。

第四, 当每一个可执行样本执行两分钟时, Qemu 将恢复快照并自动加载下一个可执行样本。当数据集中的样本全部执行完毕后, 自动微结构数据收集系统将自动停止。在该隔离环境中, 监控的时长和数据量是需要权衡的问题。由于处理器处理的底层数据庞大, 假设处理器运行在1Ghz, CPI=2, 则每秒执行的指令大约有5000万条指令, 那么所收集的微结构特征数据量也非常巨大。参考论文[61], 大多数恶意程序在前两分钟就已经执行完了恶意行为, 所以本文以两分钟作为最终的收集时间。这里注意, 收集两分钟的微结构特征数据只是一个用于训练分类器模型的必要过程, 在执行在线恶意软件检测时, 只需要分析较小检测窗口的子样本就可以实现恶意软件检测。因此, 收集测试数据所需要的时间将远小于两分钟。

3.2　微结构特征预处理

经过微结构特征收集后, 就可以获得数据集样本执行过程中产生的 GPRs 行为信息。因为在初步实验中, 本文发现仅使用单个 GPRs 特征很难有效地实现恶意软件检测, 于是这里同时分析八个 GPRs 以用于恶意行为分析。由于底层微结构特征数据量大, 为了设计简单的分类模型就需要降低微结构特征的复杂度。因此, 本文提出了一种有效地针对 GPRs 微

结构特征的预处理方式, 主要分为以下三个步骤, 生成特征矩阵, 特征区间分割以及利用 TF-IDF 技术进行特征抽取。

3.2.1 特征矩阵生成

如图 9 所示, 首先, 将从微结构特征收集系统中获得的八个 GPRs 的向量按照 EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI 顺序组成一个特征矩阵 $\boldsymbol{S}$, 可以写成:

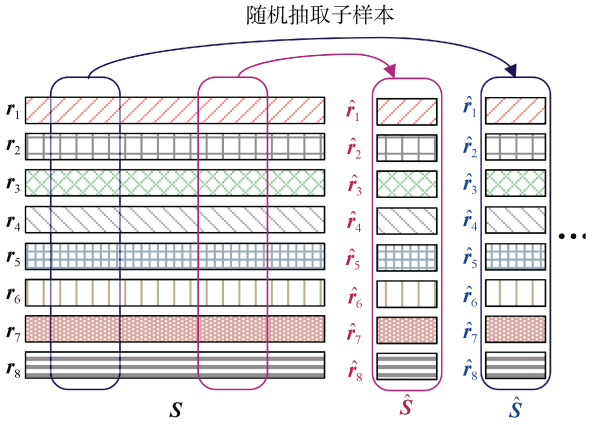

图 9 子样本生成示意图

Figure 9 Example to illustrate the generation of sub-samples

$$\boldsymbol{S}=[\boldsymbol{r}_1,\boldsymbol{r}_2,\cdots,\boldsymbol{r}_3,\boldsymbol{r}_8],$$

这里$\boldsymbol{S}$是一个高度为8的二维矩阵, 表示一个可执行样本文件中八个GPRs信息的组合。$\boldsymbol{r}_i(i\in\{1,\cdots,8\})$是一维向量, 代表八个GPRs 的其中之一。$\boldsymbol{S}$包含了可执行样本执行两分钟的微结构特征信息, 直接使用整个$\boldsymbol{S}$数据信息进行检测模型的训练是不可行的。因此, 本文选择从$\boldsymbol{S}$中抽取部分信息作为子样本以训练恶意软件检测模型。具体来说, 子样本的生成通过随机抽取固定长度的 $\hat{\boldsymbol{S}}$ 来获得, 而本文将此固定长度称为**样本长度**(检测窗口):

$$\hat{\boldsymbol{S}}=[\hat{\boldsymbol{r}}_1,\hat{\boldsymbol{r}}_2,\cdots,\hat{\boldsymbol{r}}_3,\hat{\boldsymbol{r}}_8],$$

这里 $\hat{\boldsymbol{S}}$ 为从$\boldsymbol{S}$中抽取的子样本, 而 $\hat{\boldsymbol{r}}_i$ 代表一个具有固定子样本长度的寄存器信息。因此,

$$\hat{\boldsymbol{T}}=\left[\hat{\boldsymbol{S}}_1,\hat{\boldsymbol{S}}_2,\cdots,\hat{\boldsymbol{S}}_N\right],N\in\mathbb{R}$$

代表训练集成检测模型的所有样本, 其中N代表子样本的数量。通过实验, 本文验证了从$\boldsymbol{S}$中随机抽取分割点不会影响恶意软件的检测性能, 这也证明了该方法的有效性。

3.2.2 特征区间分割

虽然经过随机截取生成的特征矩阵与源数据相比有着较小的尺寸, 但是其包含的 GPRs 微结构数据仍然是复杂的。GPRs 特征值范围是从‘0x00000000’至‘0xffffffff’, 这就意味着如果将随机特征矩阵 $\hat{S}$ 直接作为检测模型的输入, 那么就会需要复杂的模型才能获得较好的检测性能。因此, 本文提出了将 GPRs 特征值划分到不用的区间(区间量为 p)进行数据降维, 划分方式为:

$$\overline{\boldsymbol{S}}=\left\lfloor\frac{\hat{S}}{p}\right\rfloor.$$

其中 $\overline{\boldsymbol{S}}$ 为区间分割后的特征向量矩阵。

如图10所示, 将GPRs微结构特征进行区间分割后, (b)不仅保留了原始数据(a)的特征多样性, 还大大降低了GPRs 特征值范围。比如, 原先GPRs 特征值范围为[0.4294967295], 经过区间分割后, 特征值范围变成 [0, 512], 显著降低了数据维数。同时, 这种处理方式还可以抑制噪声的干扰。经过特征区间分割后, 随机特征矩阵 $\hat{\boldsymbol{T}}$ 可以写成 $\overline{\boldsymbol{T}}$

$$\overline{\boldsymbol{T}}=\overline{S}_1,\overline{S}_2,\cdots,\overline{S}_N,N\in\mathbb{R}$$

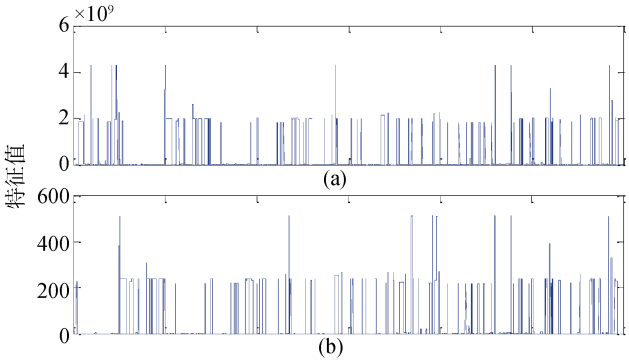

图 10 GPRs 特征区间分割图示

Figure 10 The illustration of GPRs'data after partition

3.2.3 特征 TF-IDF 处理

通常, TF-IDF 算法被用于从词库中抽取文档的关键词。在本文中, 可以将 GPRs 特征值看作文本中的词, 然后就可以利用 TF-IDF 算法从特征矩阵 $\overline{\boldsymbol{T}}$ 中抽取有价值的关键值。其中, TF 表示在某个子样本 $\overline{S}_N$ 中特征值 d 出现的频率, 可以写成:

$$TF_{d,\overline{S}_N}=\frac{count(d)}{\left|\overline{S}_N\right|},$$

这里, $count(d)$ 表示特征值 d 在 $\overline{S}_N$ 中出现的次数, $\left|\overline{S}_N\right|$ 为该子样本中的总特征值个数。IDF 用来表示特征值在整个词库中的受欢迎程度, 可以写为:

$$IDF_d=\log\frac{N}{1+\sum_{t=1}^{N}K\left(d,\overline{S}_N\right)},$$

式中, $K\left(d,\overline{S}_N\right)$表示子样本$\overline{S}_N$中是否有关键值 d。而 *TF-IDF* 为:

$$TF\text{-}IDF_{d,\overline{S}_N} = TF_{d,\overline{S}_N} * IDF_d.$$

TF 与 *IDF* 属于相互调节的关系。例如, 当特征值 d 在$\overline{S}_N$中出现频率较高, 但是在大量的样本中都有d的出现, 这说明特征值d并不关键, 而此时 IDF_d 值较小, 当将 *TF* 与 *IDF* 相乘后, 特征值 d 的权重就会降低。相反, 如果特征值 d 在子样本$\overline{S}_N$中出现频率较低, 但是其具有非常有价值的区分性信息, 而且其他子样本中几乎未出现, 此时 IDF_d 趋近于 1, 使得该特征值 d 的权值并没有很小。可以看出, 当一个特征值在子样本$\overline{S}_N$中频率越高, 并且在数据集中普遍度低, 该特征值的权值越高。但是如果该特征值并不具有较高的区分性价值, 这种情况为收集的数据集存在偏斜, 这会干扰最终模型的学习。只需要通过大量的样本进行训练就可以解决此类问题并得到泛化性较好的分类器。

总之, TF-IDF 用于确定特征值在样本集中的重要程度, 即对有价值的特征值赋予较高的权值, 对冗余信息赋予较低的权值。例如在子样本$\overline{S}_2$中, 具有重要区分价值的特征值“151”的权值为“0.9703176434770276”, 而不重要的特征值“66”的权值为“0.006495070454976745”。

本文只考虑 1-gram 的特征值进行检测模型的训练, 因为经过实验验证当 n-gram 中 $n\geqslant 2$ 时, T 的特征维度将会大幅增加, 这会严重影响模型的检测效率, 而且检测性能并不会大幅提高。如表 1 所示, 当 GPRs 微结构特征采用 2-gram 进行模型训练时, 关键词数量为 35589, 这意味着 T 输入维度为 8×35589×1。然而, 区间分割后$\overline{T}$的维度为 8 × 512 × 1, 可见数据维度增加了接近 70 倍。

表 1 使用不同 *n*-gram 的特征关键词数

Table 1 The number of key words using different *n*-grams

n-gram 关键词	#GPRs 微结构特征	#Opcodes 微结构特征
1-gram	503	201
2-gram	35589	11396
3-gram	73043	93282

3.3 集成学习检测模型设计

在对微结构特征预处理后, 本文利用集成学习技术构建恶意软件检测模型, 因为它可以用来提高传统机器学习分类器的准确性[62-63]。如图11所示, 首先, 选择多个传统机器学习分类器构建基分类器集, 如NB, RF, LR, DT, SVM 等。然后, 在基分类器中选择检测性能排名前三(TOP 3)的分类器来构建恶意软件检测模型, 并通过决策函数输出最终的检测结果。事实上, 集成学习模型已经被证明可以有效地区分恶意软件与良性软件[64]。为了验证该检测模型的有效性, 本文所有实验均使用10折交叉验证进行测试。

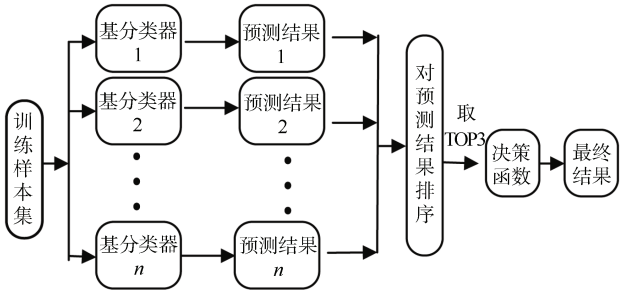

图 11 集成检测器

Figure 11 Ensemble detector

4 实验设置

4.1 实验环境

本文实验使用Python机器学习库Sklearn实现模型算法的学习, 该 sklearn 库提供 sklearn.ensemble, 其中包含多种机器学习方法的接口以及集成算法的接口, 便于集成模型的设计与实现。在本文, 所有实验均在 64 位 Ubuntu 18.04 操作系统环境中执行。

4.2 数据集

本实验所用的恶意样本数据集来自两个通用的数据库, VxHeaven[65]及VirusShare[66]。虽然VxHeaven包含585个恶意家族类型, 总共超过20K的恶意样本, 但是该数据集在2010年以后没有再更新。如果仅仅的依赖VxHeaven 数据集, 会降低检测模型对未知样本的检测鲁棒性。相反, VirusShare数据库是每天都在更新的。因此在本文, 为了保证恶意样本多样化, 从VxHeaven(60%)和VirusShare(40%)中随机下载了2715个可执行文件。尽管该数据集样本量比基于高级别特征恶意软件检测方法要少, 但是本文认为该数据集足以证明该方法的有效性。因为该模型在测试未知样本集时, 依然具有很高的检测准确性。此外, Ozsoy 等[9,56]仅使用包含1087 个恶意软件样本的数据集进行实验。这里注意, VirusShare 数据库中本文选择下载的恶意样本文件是Windows 32系统的可执行文件。良性样本数据集(共1526个)包含各种合法程序, 例如操作系统中的本机使用程序和应用程序可

执行文件。微结构特征自动数据收集系统会调用所有恶意和良性样本以提取微结构特征。由于某些恶意软件具有反虚拟机技术, 可以感知它们是否在虚拟环境中工作。如果这些恶意软件检测到自己在虚拟机中, 它们将隐藏执行自己的恶意行为以避免被检测。还有一些特殊的恶意软件, 例如定时炸弹恶意软件, 该恶意软件旨在在特定时间被激活。经过使用peframe工具和人工排除, 表2中未发现以上特殊类型的恶意可执行文件。为了设计检测模型并实现合理的验证, 并按照Wang 等[67]的方式将恶意样本集分为两个子集: 已知样本集与未知样本集。其中已知恶意样本集用来训练恶意行为检测模型, 而未知恶意样本集用来验证检测模型的检测有效性。同样, 将收集到的1526个良性样本集分成已知样本集(1220个良性样本)与未知样本集(306个未知样本)。这里, 所有样本集都由微结构特征收集系统(第3.1小节)依次收集微结构特征。

表 2 恶意样本数据集

Table 2 Malware Dataset

恶意家族	#总量	#已知	#未知
Backdoor	739	591	148
Worm	370	296	74
Virus	386	308	78
Rootkit	348	278	70
HackTool	60	48	12
Trojan	812	649	163

4.3 评价标准

本文将恶意样本和良性样本的标签分别设置为正样本和负样本(如表 3)。如果样本真实标签和预测的标签都是恶意的(阳性), 则它属于正确的分类(*True Positive*, *TP*)。类似地, 当输入样本是良性的并且也被识别为良性时, 分类将被视为真阴性(*True Negative*, *TN*)。相反, 将任何恶意样本归类为良性或将任何良性样本归类为恶意, 分别定义为假阴性(*False Negative*, *FN*)和假阳性(*False Positive*, *FP*)。表 3 展示了分类的所有可能结果[68]。为了全面衡量检测性能, 本文还使用准确率(*Accuracy*)与 *F1-Measure* 作为模型的评价标准。在计算机底层, 现实生活中所检测到的微结构信息大多数都是合法行为, 因此检测模型还应该具有较低的误报率(*FPr*)。*FPr* 表示所有负样本中所占比例 *FP* 的数量。以下是三个综合指标:

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

$$F1\text{-}Measure = \frac{2 * TP}{2 * TP + FP + FN},$$

$$FPr = \frac{FP}{FP + TN}.$$

表 3 混淆矩阵

Table 3 Confusion matrix

预测类别	真实类别	
	恶意软件	良性软件
恶意软件	True Positive (TP)	False Positive (FP)
良性软件	False Negative (FN)	True Negative (TN)

5 实验结果与分析

在本节中, 首先分析检测窗口以及微结构特征值区间分割数对模型检测性能的影响。然后, 将集成分类模型与其他恶意行为检测方法(使用不同的微结构特征)进行比较。此外, 本节还评估了本文集成分类模型对未知恶意软件的检测效果。最后, 文章还介绍了该方法在真实数据上的检测有效性。

5.1 检测窗口评估

如图12所示, 本小节分析检测窗口对恶意行为检测的影响。这里, 检测窗口的大小是指检测模型所需分析的指令条数。为了选择合适的检测窗口, 本文分别在已知数据集上学习了10种集成检测模型, 这些模型具有不同的检测窗口大小: 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000。检测结果如图12所示, 从图中可以发现较小的检测窗口具有较低的检测

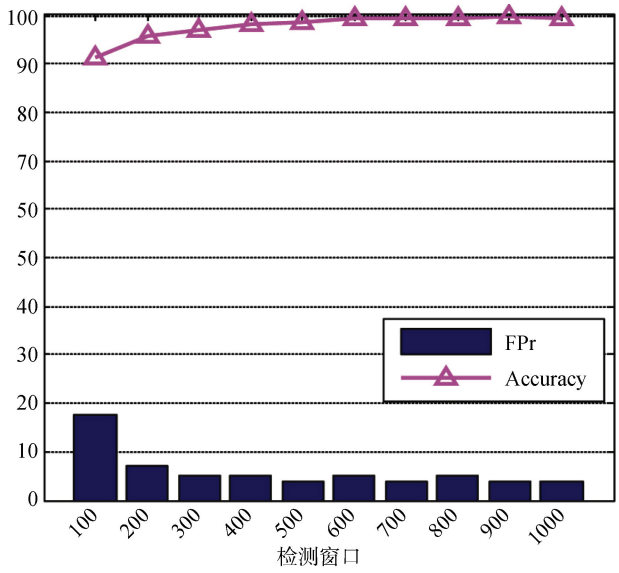

图 12 检测窗口对检测性能影响

Figure 12 Effect of window size on detection performance

*Accuracy*和较高的*FPr*。例如, 当窗口大小为100 时, *Accuracy*为0.913, *FPr*为0.178。但是, 当检测窗口大小增加时, *Accuracy*将显著增加, *FPr*也将减少。例如, 当窗口大小从100增加到700 时, *Accuracy*从0.913增加到0.993, *FPr*从0.178减少到0.037。随着窗口大小的不断增加, 检测性能将趋于平稳。这是因为较大的窗口将引入更多的冗余信息, 这将会对影响检测模型的检测[9,56]。因此, 为了平衡检测效率和*FPr*, 本文最终将窗口大小选择为0.7K(即700条指令)以进行恶意行为检测。

5.2 区间值评估

通过上节的分析, 本文选择0.7K作为检测窗口的大小。在这里, 特征分割数也是影响恶意软件检测性能的重要因素。为了评估其影响, 文章通过使用不同的分割数量来评估所设计的集成检测模型。评估结果如图13, 实验展示了使用六种分割区间值的模型检测结果。从图中可以观察到, 当分区数量变大时, 恶意软件检测性能变得更好。主要原因是较大的分区数可以保留更多GPRs微结构数据信息。但是, 太大的分割数将会直接增加模型输入的维数, 因此计算成本也会急剧增加。从图13可以看出, 当GPRs微结构特征值分割为512个区间时, 检测模型表现最佳。虽然512个区间与1024个区间检测性能非常相近, 但是本文最终将GPRs特征值分割区间数设置为512, 可以平衡计算复杂度和检测性能。根据以上分析, 本文将使用0.7K大小的检测窗口和512 GPRs特征分割数评估以下所有实验。

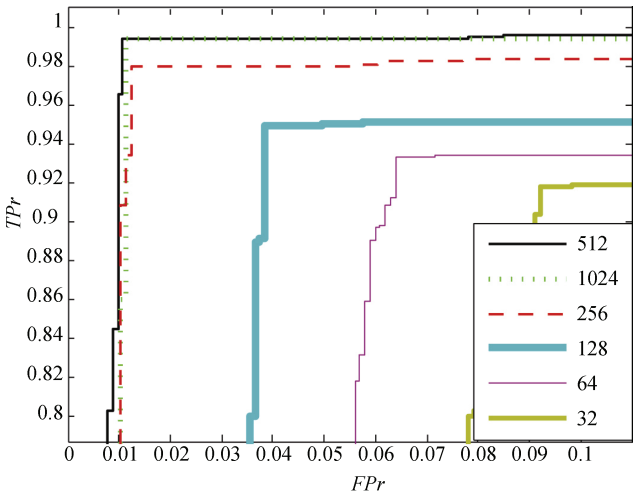

图 13 使用不同分割区间数量的 ROC 曲线

Figure 13 ROC curves using different number of partitions

5.3 微结构特征评估

Ozsoy 等人[9]提出 Opcode 特征能够有效实现恶意软件检测, 而 Li 等人[11]验证了 GPRs 微结构特征也可以用于恶意软件检测。因此在本节, 我们评价不同微结构特征在集成检测模型上的检测性能, 特征分别为 Opcode TF-IDF, GPRs 以及 GPRs TF-IDF。这里, Opcode 与 GPRs 特征是同时从微结构特征收集系统中获得的。对于 Opcode TF-IDF 特征, 列出了当使用 1-gram, 2-gram 以及 3-gram 作为输入时, 集成模型的检测结果。

如图 14 以及表 4 所示, 当 Opcode TF-IDF 选择 1-gram 时, 检测性能最低, 2-gram 以及 3-gram 时, 两者的检测性能是接近的(3-gram 检测 *Accuracy* 高于 2-gram 0.7%, 检测 *FPr* 高于 2-gram 0.6%)。但是当 $n=3$ 时, 所分析的关键词序列过长(表 1, 3-gram 序列长度为 93282), 这会严重影响检测效率, 因此在实际应用中通常会选择 2-gram[28]。图 14 以及表 4 还展示了 GPRs 特征未经过特征区间分割以及 TF-IDF 预处理而直接输入时, 集成检测模型的结果。可以发现, 使用未经预处理的 GPRs 微结构特征很难获得优异的检测性能(*Accuracy* 为 0.919, *FPr* 为 0.083)。当 GPRs 微结构数据经过特征区间分割以及 TF-IDF 抽取特征后, 集成模型可以获得较高的检测性能, 这说明对 GPRs 微结构特征进行合理预处理的重要性。另外, 本节还验证了 GPRs TF-IDF 特征优于 Opcode TF-IDF。如表 4 所示, 当使用相同的实验环境以及相同的检测模型时, Opcode TF-IDF 最佳检测性能结果低于 GPRs TF-IDF。其中, GPRs TF-IDF *Accuracy* 为 0.993, *FPr* 为 0.037, *Accuracy* 比 Opcode TF-IDF 高 3.5%, *FPr* 比 Opcode TF-IDF 低 3.4%。

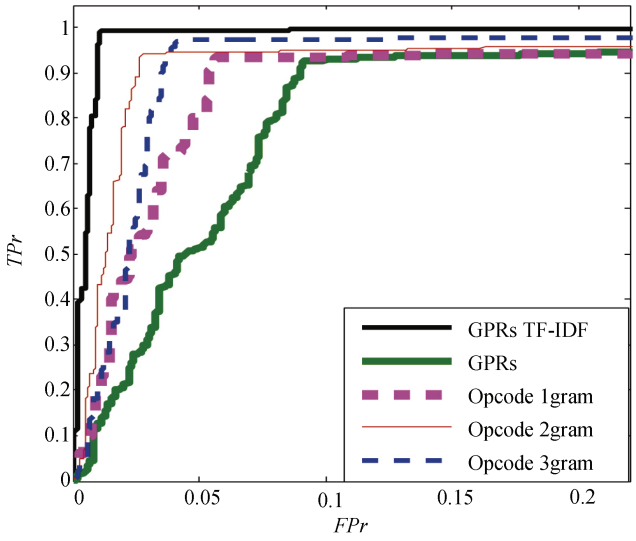

图 14 不同微结构特征检测性能 ROC 曲线

Figure 14 ROC curves using different low-level features

表 4 不同微结构特征检测性能比较

Table 4 Comparisons of using different types of low-level features

微结构特征	Opcode TF-IDF			GPRs 源数据	**GPRs TF-IDF**
	Opcode 1-gram	Opcode 2-gram	Opcode 3-gram		
Accuracy	0.939	0.958	0.965	0.919	**0.993**
F1-Measure	0.940	0.958	0.963	0.920	**0.993**
FPr	0.114	0.071	0.077	0.083	**0.037**

表 5 不同模型检测性能

Table 5 Detection performance of different models

模型	SVM	MLP	KNN	Fusion-ST[11]	CNNs	LSTMs	本文方法
Accuracy	0.525	0.564	0.521	0.953	0.824	0.732	**0.993**
F1-Measure	0.507	0.558	0.239	0.951	0.831	0.748	**0.993**
FPr	0.505	0.449	0.235	0.072	0.214	0.327	**0.037**

5.4 模型评估

此外, 本文还将所设计的集成模型与其他流行的分类模型进行了对比。如图 15 以及表 5 所示, 分类方法同时包含传统机器学习技术和复杂的神经网络技术。从检测结果可以发现, 单一机器学习方法很难实现较高的检测性能。例如 KNN 分类器 *Accuracy* 为 0.521, *FPr* 为 0.235。主要原因为 GPRs 微结构特征包含过多的冗余信息, 因此难于分析。此外, 本文与 Li 等人[11]的深度学习模型 Fusion-ST 以及经典的 CNNs, LSTMs 分类模型进行了比较。实验显示, 深度学习模型优于传统的机器学习模型, 其中 CNNs 检测性能优于 LSTMs(CNNs 检测 *Accuracy* 为 0.824, *FPr* 为 0.214。LSTMs 检测 *Accuracy* 为 0.732, *FPr* 为 0.327)的原因为 CNNs 结构可以充当滤波器抑制 GPRs 中的噪声, 而直接利用 LSTMs 的时序属性分析 GPRs 特征容易受到噪声干扰。于是 Fusion-ST 方法融合了 GPRs 的时空特性(CNNs 与 LSTMs 结合), 并获得了良好的检测性能。但是, 它的检测 *Accuracy* 为 0.953, *FPr* 为 0.072, 低于本文集成检测模型。这是因为 Fusion-ST 检测方法直接分析了 GPRs 的内容, 保留了过多的冗余信息。虽然 Fusion-ST 模型可以用于恶意行为检测, 但是复杂的输入数据仍然会影响其检测性能。因此, 本文认为有必要对 GPRs 数据进行适当的预处理, 并提取出更多有价值的信息以进行恶意软件检测。本文的检测结果也证实了这一想法, 如本文模型检测 *Accuracy* 为 0.993, *F1-Measure* 为 0.993, 检测 *FPr* 为 0.039, 均优于其他方法模型。

5.5 未知样本评估

为了验证集成检测模型对未知恶意软件检测的有效性, 本文根据 Wang 等人[67] 的方法将数据集分为两部分(如表 2 所示)。本节将在未知恶意样本集和剩余的良性数据集上进行集成模型的测试。实验对比了传统的机器学习算法与较复杂的神经网络算法。检测结果如表 6, 从中可以发现, 与其他检测模型相比, 本文集成学习检测方法效果更好。其中, 基于传统机器学习的方法检测性能较差(例如, DT 模型检测 *Accuracy* 为 0.829, 但是 *FPr* 为 0.406), 这表明简单的机器学习方法无法从 GPRs 中学习有价值的特征信息。虽然 Fusion-ST[11] 方法可以识别未知恶意软件, 与本文的集成策略检测性能相近(集成检测模型的检测 *Accuracy* 比 Fusion-ST 模型至少高 1.8%, 而检测 *FPr* 比 Fusion-ST 模型低 1.2%)。但是, Fusion-ST 模型利用深度学习技术构建模型, 比集成方法更复杂。例如 Fusion-ST 检测模型实现需要约 250M 次浮点运算, 约有 99.6K 个参数。相反, 本文

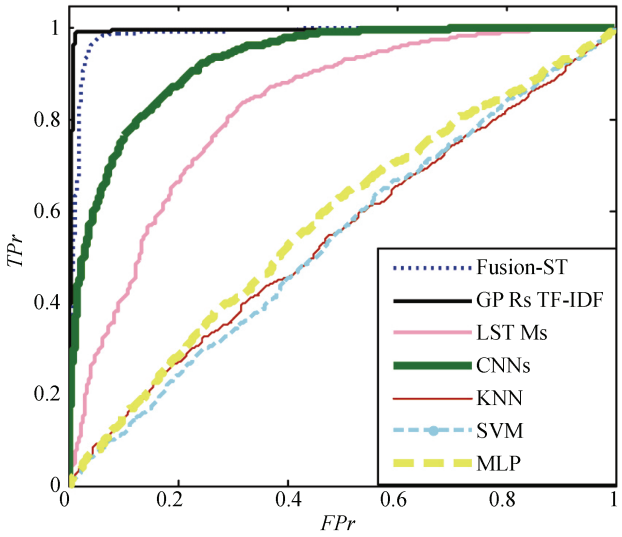

图 15 不同模型检测性能 ROC 曲线

Figure 15 ROC curves of different models

表 6 未知样本评估

Table 6 Evaluation of detecting unknown malware

模型	SVM	MLP	KNN	DT	Fusion-ST[11]	本文方法
Accuracy	0.518	0.562	0.532	0.829	0.921	**0.939**
F1-Measure	0.495	0.584	0.263	0.838	0.926	**0.941**
FPr	0.502	0.477	0.231	0.406	0.132	**0.120**

集成检测技术仅需要依赖约 4.1K 个参数就可以有效实现恶意软件检测。

5.6 真实数据评估

为了进一步验证本文集成检测方法的有效性, 本节从真实的物理机器中收集了微体系结构行为特征, 并对文中模型进行了评估。首先, 本文从 VirusShare 下载了 205 个 Linux 可执行恶意软件, 并收集了 25 个合法进程的微结构行为信息。然后, 在物理机器上执行所有样本集, 并使用内部工具在执行期间收集 GPRs 行为信息。集成模型的最终检测 *Accuracy* 接近 0.99, *FPr* 为 0.167。其中 *FPr* 较高的原因是负样本集太少, 这会影响最终的 FPr。此外, 在本实验中所收集的低级别数据都是基于 64 位 Ubuntu 系统, 这意味着 GPRs 的内容范围是从“0x0000000000000000”到“0xffffffffffffffff”, 与本文的训练数据不同。但是, 本文所设计的集成检测模型依旧能够实现较好的检测结果。

5.7 高级别特征与微结构特征比较

本文提出的恶意软件检测方法与近几年使用高级别特征与微结构特征实现恶意软件检测方法的检测结果对比如表 7 所示。

高级别特征: Tian 等人[69]使用机器学习技术, 分析了 1368 个恶意样本与 456 个良性样本执行过程中产生的 API 序列, 最终获得 0.970 以上的检测准确率。Amer 等人[70]则提出了一种利用 API 调用序列存在上下文关系的恶意软件检测方法, 该方法使用马尔科夫链表示方法对 API 产生的行为序列进行建模, 最终检测 *Accuracy* 为 0.990, *FPr* 为 0.010。以上这两种方法都基于动态检测分析技术, Han 等人[71]同时考虑了静态 API 与动态 API 之间的差异性与相关性, 通过对静态与动态 API 序列的关联与融合构建了一个可解释的恶意软件检测框架, 检测准确率为 0.9789。

表 7 高级别特征与微结构特征检测性能对比

Table 7 Comparison of high-level features and microarchitecture features

	方法	特征	恶意样本/良性样本	Accuracy	FPr
高级别特征	Tian 等人[69], 2010	API	1368/456	0.973	—
	Amer 等人[70], 2019	API	30658/21422	0.990	0.010
	Han 等人[71], 2019	API	5567/1174	0.9789	—
微结构特征	Demme 等人[10], 2013	HPC	503/210	<0.900	<0.100
	0zsoy 等人[9], 2016	Opcode 频率	1087/467	—	0.090 ~0.160
	Khasawneh 等人[8], 2018	Opcode 频率	3653/554	0.938	0.056
	本文方法	**GPRs TF-IDF**	**2715/1526**	**0.993**	**0.037**

微结构特征: Demme 等人[10]认为在系统受到攻击与未收到攻击时, 所统计的 HPC 特征具有明显的区分性。于是该方法对 503 个恶意样本与 210 个良性样本使用机器学习技术训练分类模型, 并首次验证了 HPC 特征可以用于实现恶意软件检测。但是该方法的检测性能较低, 最高检测 *Accuracy* 为 0.900, *FPr* 接近 0.100。Ozsoy 等人[9] 与 Khasawneh 等人[8]都使用 Opcode 最大差异频率特征实现恶意软件检测, 并基于机器学习技术与神经网络技术构造了分类模型。这两种检测方法虽然都能够有效识别恶意软件, 但是都需要连续分析 10K 指令的信息, 即对连续 10K 指令的信息进行分析后才能做出最终决策, 这将不可避免的影响模型的检测灵敏度以及检测效率。而本文方法, 只需要分析 0.7K 指令的信息, 就可以做出有效决策, 并且与目前存在的微结构检测技术相比具有较高的检测性能(*Accuracy* 为 0.993, *FPr* 为 0.037)。

总之, 利用高级别特征以及微结构特征都可以实现对恶意软件的检测。但是使用微结构特征实现恶意软件检测能够有效避免恶意软件在应用层或操作系统层逃避检测的情况发生[7], 并且不需要等待低级别特征转换成高级别特征, 就可以直接依赖底

层低级别数据直接实现决策, 提高检测灵敏度和检测效率。通过以上对比可以发现, 目前微结构特征实现恶意软件检测所依赖的数据集较小, 主要原因为在计算机底层, 短时间内会收集到大量的微结构特征, 在模型的设计过程中需要权衡所需要分析的数据量。虽然目前存在的微结构特征检测技术 *FPr* 相比较与某些高级别特征检测技术较高, 例如 Khasawneh 等人[8] *FPr* 为 0.056, 但是在计算机底层获得的微结构特征数据复杂度较高, 并且容易受到噪声的干扰。而本文与目前存在的微结构检测技术相比, 能够依赖较小的检测窗口(分析 0.7K 指令信息)就可以获得较高的检测性能, 其中检测 *Accuracy* 为 0.993, *FPr* 为 0.037。

6 结论

利用集成学习策略, 本文提出了一种基于 GPRs 特征值频率分析的恶意软件检测方法。虽然低级别微结构特征能够有效实现恶意软件检测, 但是它们包含过多的冗余信息和噪声信息, 这不可避免的会影响检测模型的检测效率和检测性能。因此, 本文使用 TF-IDF 技术来降低特征数据的复杂性, 并抽取有区分性的信息以进行恶意软件检测。该预处理方式有利于检测模型的学习。另外, 本文构造了集成分类模型, 以进一步提高检测性能。在未知数据集和真实数据上的综合实验证明, 本文检测方法比其他代表性的模型具有更好的检测性能。目前, 本文工作主要集中在二分类问题上, 下一步将把它扩展到多分类问题。同时, 文章也致力于将该方法扩展到其他体系结构, 例如 ARM, RISC 和 MIPS。

参考文献

[1] The Global Risks Report 2020. World Economic Forum. https://www.weforum.org/reports/the-global-risks-report-2020. Mar. 2020.

[2] LeDoux C, Lakhotia A. Malware and Machine Learning[M]. Intelligent Methods for Cyber Warfare. Cham: Springer International Publishing, 2014: 1-42.

[3] Malware Statistics. AV-TEST. https://www.av-test.org/en/statistics/malware/. Feb. 2020.

[4] Peiravian N, Zhu X Q. Machine Learning for Android Malware Detection Using Permission and API Calls[C]. *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, 2014: 300-305.

[5] Ma Z, Ge H R, Liu Y, et al. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms[J]. *IEEE Access*, 2019, 7: 21235-21245.

[6] Banin S, Dyrkolbotn G O. Multinomial Malware Classification via Low-Level Features[J]. *Digital Investigation*, 2018, 26: S107-S117.

[7] Tang A, Sethumadhavan S, Stolfo S J. Unsupervised Anomaly-Based Malware Detection Using Hardware Features[M]. Research in Attacks, Intrusions and Defenses. Cham: Springer International Publishing, 2014: 109-129.

[8] Khasawneh K N, Ozsoy M, Donovick C, et al. EnsembleHMD: Accurate Hardware Malware Detectors with Specialized Ensemble Classifiers[J]. *IEEE Transactions on Dependable and Secure Computing*, 2020, 17(3): 620-633.

[9] Ozsoy M, Khasawneh K N, Donovick C, et al. Hardware-Based Malware Detection Using Low-Level Architectural Features[J]. *IEEE Transactions on Computers*, 2016, 65(11): 3332-3344.

[10] Demme J, Maycock M, Schmitz J, et al. On the Feasibility of Online Malware Detection with Performance Counters[J]. *ACM SIGARCH Computer Architecture News*, 2013, 41(3): 559-570.

[11] Li F, Yan C, Zhu Z Y, et al. A Deep Malware Detection Method Based on General-Purpose Register Features[C]. *Computational Science - ICCS 2019*: *19th International Conference*, *Faro*, *Portugal*, *June 12-14*, *2019*, *Proceedings*, *Part III*, 2019: 221-235.

[12] Ghiasi M, Sami A, Salehi Z. Dynamic Malware Detection Using Registers Values Set Analysis[C]. *2012 9th International ISC Conference on Information Security and Cryptology*, 2013: 54-59.

[13] Christodorescu M, Jha S. Static Analysis of Executables to Detect Malicious Patterns[C]. *The 12th conference on USENIX Security Symposium - Volume 12*, 2003: 12.

[14] Christodorescu M, Jha S, Seshia S A, et al. Semantics-Aware Malware Detection[C]. *2005 IEEE Symposium on Security and Privacy*, 2005: 32-46.

[15] Zhang Q H, Reeves D S. MetaAware: Identifying Metamorphic Malware[C]. *Twenty-Third Annual Computer Security Applications Conference*, 2008: 411-420.

[16] Ye Y F, Wang D D, Li T, et al. An Intelligent PE-Malware Detection System Based on Association Mining[J]. *Journal in Computer Virology*, 2008, 4(4): 323-334.

[17] Kolbitsch C, Comparetti P M, Kruegel C, et al. Effective and Efficient Malware Detection at the End Host[C]. *The 18th conference on USENIX security symposium*, 2009: 351-366.

[18] Park Y, Reeves D, Mulukutla V, et al. Fast Malware Classification by Automated Behavioral Graph Matching[C]. *The Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, 2010: 1-4.

[19] Stinson E, Mitchell J C. Characterizing Bots' Remote Control Behavior[C]. *The 4th international conference on Detection of Intrusions and Malware*, *and Vulnerability Assessment*, 2007: 89-108.

[20] Park Y, Reeves D S, Stamp M. Deriving Common Malware Behavior through Graph Clustering[J]. *Computers & Security*, 2013, 39: 419-430.

[21] Vlachos V, Ilioudis C, Papanikolaou A. On the Evolution of Malware Species[M]. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012: 54-61.

[22] Leder F, Steinbock B, Martini P. Classification and Detection of Metamorphic Malware Using Value Set Analysis[C]. *2009 4th In-*

ternational Conference on Malicious and Unwanted Software, 2010: 39-46.

[23] Schultz M G, Eskin E, Zadok F, et al. Data Mining Methods for Detection of New Malicious Executables[C]. *Proceedings 2001 IEEE Symposium on Security and Privacy*. *S&P*, 2002: 38-49.

[24] Jurafsky D. Speech and language processing[M]. Pearson Education India, 2000.

[25] Kešelj V, Peng F, Cercone N, et al. N-gram-based author profiles for authorship attribution[C]. *16th International Conference of the Pacific Association for Computational Linguistics*, 2003: 255-264.

[26] Pervaiz R, Aloufi K, Zaidi S S R, et al. A Methodology to Identify Topic of Video via N-Gram Approach[J]. *International Journal of Computer Science and Network Security*, 2020, 20(1): 79-94.

[27] Bilar D. Opcodes as Predictor for Malware[J]. *International Journal of Electronic Security and Digital Forensics*, 2007, 1(2): 156-168.

[28] Dolev S, Tzachar N. Malware signature builder and detection for executable code. US Patent EP2189920, 2008.

[29] Moskovitch R, Feher C, Tzachar N, et al. Unknown Malcode Detection Using OPCODE Representation[C]. *The 1st European Conference on Intelligence and Security Informatics*, 2008: 204-215.

[30] Zhao Z Q, Wang J F, Bai J R. Malware Detection Method Based on the Control-Flow Construct Feature of Software[J]. *IET Information Security*, 2014, 8(1): 18-24.

[31] Ding Y X, Dai W, Yan S L, et al. Control Flow-Based Opcode Behavior Analysis for Malware Detection[J]. *Computers & Security*, 2014, 44: 65-74.

[32] Chan P P K, Song W K. Static Detection of Android Malware by Using Permissions and API Calls[C]. *2014 International Conference on Machine Learning and Cybernetics*, 2015: 82-87.

[33] 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*, 2014: 2337-2342.

[34] Kohli P, Bruhadeshwar B. Signature Generation and Detection of Malware Families[C]. *The 13th Australasian conference on Information Security and Privacy*, 2008: 336-349.

[35] Kawaguchi N, Omote K. Malware Function Classification Using APIs in Initial Behavior[C]. *2015 10th Asia Joint Conference on Information Security*, 2015: 138-144.

[36] Malone C, Zahran M, Karri R. Are Hardware Performance Counters a Cost Effective Way for Integrity Checking of Programs[C]. *The sixth ACM workshop on Scalable trusted computing*, 2011: 71-76.

[37] Xia Y B, Liu Y T, Chen H B, et al. CFIMon: Detecting Violation of Control Flow Integrity Using Performance Counters[C]. *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2012: 1-12.

[38] Sayadi H, Mohammadi Makrani H, Randive O, et al. Customized Machine Learning-Based Hardware-Assisted Malware Detection in Embedded Devices[C]. *2018 17th IEEE International Conference on Trust*, *Security and Privacy In Computing and Communications/ 12th IEEE International Conference on Big Data Science and Engineering*, 2018: 1685-1688.

[39] Yerima S Y, Sezer S, Muttik I. High Accuracy Android Malware Detection Using Ensemble Learning[J]. *IET Information Security*, 2015, 9(6): 313-320.

[40] Dong L, Yang N, Wang W, et al. Unified language model pre-training for natural language understanding and generation[C]. *32th Advances in Neural Information Processing Systems*, 2019: 13063-13075.

[41] Ren W H, Tian J D, Wang Q, et al. Dually Connected Deraining Net Using Pixel-Wise Attention[J]. *IEEE Signal Processing Letters*, 2020, 27: 316-320.

[42] Saxe J, Berlin K. Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features[C]. *2015 10th International Conference on Malicious and Unwanted Software*, 2016: 11-20.

[43] Kolosnjaji B, Eraisha G, Webster G, et al. Empowering Convolutional Networks for Malware Classification and Analysis[C]. *2017 International Joint Conference on Neural Networks*, 2017: 3838-3845.

[44] Gibert D, Mateu C, Planes J, et al. Using Convolutional Neural Networks for Classification of Malware Represented as Images[J]. *Journal of Computer Virology and Hacking Techniques*, 2019, 15(1): 15-28.

[45] Nataraj L, Karthikeyan S, Jacob G, et al. Malware Images: Visualization and Automatic Classification[C]. *The 8th International Symposium on Visualization for Cyber Security*, 2011: 1-7.

[46] Yen Y S, Sun H M. An Android Mutation Malware Detection Based on Deep Learning Using Visualization of Importance from Codes[J]. *Microelectronics Reliability*, 2019, 93: 109-114.

[47] Kumar A, Sagar K P, Kuppusamy K S, et al. Machine Learning Based Malware Classification for Android Applications Using Multimodal Image Representations[C]. *2016 10th International Conference on Intelligent Systems and Control*, 2016: 1-6.

[48] Chen T, Mao Q, Lv M, et al. DroidVecDeep: Android Malware Detection Based on Word2Vec and Deep Belief Network[J]. *KSII Transactions on Internet and Information Systems*, 2019, 13(4): 2180-2197.

[49] Yan J P, Qi Y, Rao Q F. LSTM-Based Hierarchical Denoising Network for Android Malware Detection[J]. *Security and Communication Networks*, 2018, 2018: 1-18.

[50] Maniath S, Ashok A, Poornachandran P, et al. Deep Learning LSTM Based Ransomware Detection[C]. *2017 Recent Developments in Control*, *Automation & Power Engineering*, 2018: 442-446.

[51] Liu Y Y, Wang Y W. A Robust Malware Detection System Using Deep Learning on API Calls[C]. *2019 IEEE 3rd Information Technology*, *Networking*, *Electronic and Automation Control Conference*, 2019: 1456-1460.

[52] Abderrahmane A, Adnane G, Yacine C, et al. Android Malware Detection Based on System Calls Analysis and CNN Classification[C]. *2019 IEEE Wireless Communications and Networking Conference Workshop*, 2019: 1-6.

[53] Kolosnjaji B, Zarras A, Webster G, et al. Deep Learning for Classification of Malware System Call Sequences[C]. *AI 2016*:

Advances in Artificial Intelligence: *29th Australasian Joint Conference*, *Hobart*, *TAS*, *Australia*, *December 5-8*, *2016*, *Proceedings*, 2016: 137-149.

[54] Xiao X, Zhang S F, Mercaldo F, et al. Android Malware Detection Based on System Call Sequences and LSTM[J]. *Multimedia Tools and Applications*, 2019, 78(4): 3979-3999.

[55] Xiao X, Wang Z L, Li Q, et al. Back-Propagation Neural Network on Markov Chains from System Call Sequences: A New Approach for Detecting Android Malware with System Call Sequences[J]. *IET Information Security*, 2017, 11(1): 8-15.

[56] Ozsoy M, Donovick C, Gorelik I, et al. Malware-Aware Processors: A Framework for Efficient Online Malware Detection[C]. *2015 IEEE 21st International Symposium on High Performance Computer Architecture*, 2015: 651-661.

[57] Dinaburg A, Royal P, Sharif M, et al. Ether: Malware Analysis via Hardware Virtualization Extensions[C]. *The 15th ACM conference on Computer and communications security*, 2008: 51-62.

[58] Carlin D, O'Kane P, Sezer S. Dynamic Analysis of Malware Using Run-Time Opcodes[M]. Data Analytics and Decision Support for Cybersecurity. Cham: Springer International Publishing, 2017: 99-125.

[59] Okane P, Sezer S, McLaughlin K, et al. Malware Detection: Program Run Length Against Detection Rate[J]. *IET Software*, 2014, 8(1): 42-51.

[60] Bellard F. QEMU, a Fast and Portable Dynamic Translator[C]. *The annual conference on USENIX Annual Technical Conference*, 2005: 41.

[61] Fredrikson M, Jha S, Christodorescu M, et al. Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors[C]. *2010 IEEE Symposium on Security and Privacy*, 2010: 45-60.

[62] Sayadi H, Patel N, P D S M, et al. Ensemble Learning for Effective Run-Time Hardware-Based Malware Detection: A Comprehensive Analysis and Classification[C]. *2018 55th ACM/ESDA/IEEE Design Automation Conference*, 2018: 1-6.

[63] Aghaei E, Serpen G. Ensemble Classifier for Misuse Detection Using N-Gram Feature Vectors through Operating System Call Traces[J]. *International Journal of Hybrid Intelligent Systems*, 2017, 14(3): 141-154.

[64] Wolpert D H. Stacked Generalization[J]. *Neural Networks*, 1992, 5(2): 241-259.

[65] VXHeaven Virus Collection 2010-05-18. VXHeaven. http:// academictorrents.com/details/. Apr. 2020.

[66] VirusShare. VirusShare. https://virusshare.com/. Jul. 2020.

[67] Wang S, Chen Z Z, Yu X, et al. Heterogeneous Graph Matching Networks for Unknown Malware Detection[C]. *The 28th International Joint Conference on Artificial Intelligence*, 2019: 3762-3770.

[68] M A A K, C D J. Automated Multi-Level Malware Detection System Based on Reconstructed Semantic View of Executables Using Machine Learning Techniques at VMM[J]. *Future Generation Computer Systems*, 2018, 79: 431-446.

[69] Tian R H, Islam R, Batten L, et al. Differentiating Malware from Cleanware Using Behavioural Analysis[C]. *2010 5th International Conference on Malicious and Unwanted Software*, 2010: 23-30.

[70] 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.

[71] Han W J, Xue J F, Wang Y, et al. MalDAE: Detecting and Explaining Malware Based on Correlation and Fusion of Static and Dynamic Characteristics[J]. *Computers & Security*, 2019, 83: 208-233.

李芳 于 2016 年在沈阳理工大学计算机技术专业获得硕士学位。现在中国科学院大学网络空间安全学院网络空间安全专业攻读博士学位。研究领域为计算机系统结构。研究兴趣包括: 恶意软件检测、深度学习等。Email: lifang@iie.ac.cn

闫超 于 2016 年在山东农业大学获得学士学位。于 2019 年在中国科学院大学计算机科学专业获硕士学位。目前在中国科学院信息工程研究所工作。研究兴趣包括: 计算机体系结构与恶意软件检测等。Email: yanchao@iie.ac.cn

朱子元 于 2010 年在同济大学控制理论与控制工程专业获得博士学位。现任中国科学院信息工程研究所研究员。研究领域为网络空间安全、计算机系统结构。研究兴趣包括: 安全芯片技术、处理器安全技术、系统安全理论与技术。Email: zhuziyuan@iie.ac.cn

孟丹 于 1995 年在哈尔滨工业大学计算机体系结构专业获得博士学位。现任中国科学院信息工程研究所所长。研究领域包括计算机系统安全, 大数据与云计算等。研究兴趣包括: 计算机系统安全, 云计算安全等。Email: mengdan@iie.ac.cn