基于SDR句嵌入的挖矿恶意软件早期检测方法

钟凯, 郭春, 李显超, 申国伟

引用本文

钟凯, 郭春, 李显超, 申国伟. 基于SDR句嵌入的挖矿恶意软件早期检测方法[J]. 计算机科学, 2024, 51(12): 303-309.

ZHONG Kai, GUO Chun, LI Xianchao, SHEN Guowei. Cryptomining Malware Early Detection Method Based on SDR [J]. Computer Science, 2024, 51(12): 303-309.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231200041

基于 SDR 句嵌入的挖矿恶意软件早期检测方法

钟 凯[1] 郭 春[1] 李显超[2] 申国伟[1]
1 贵州大学计算机科学与技术学院公共大数据国家重点实验室 贵阳 550025
2 贵州省云计算与大数据专业硕士研究生工作站 贵阳 550014
(boneink@126.com)

摘 要 挖矿恶意软件以盗用设备的计算资源来挖掘加密货币为目标,在大量消耗计算资源的同时还严重危害网络安全。当前的挖矿恶意软件动态检测方法主要依据样本长时间运行过程中收集的主机行为或网络流量来进行检测,未能兼顾检测的及时性和准确性。通过对挖矿恶意软件运行初期的 DLL 调用和 API 返回值进行分析,提出一种 API 句嵌入方法 SDR,并基于 SDR 进一步提出一种基于 SDR 的挖矿恶意软件早期检测方法 CEDS。CEDS 利用 SDR 将软件运行初期的 API 名称序列、API 返回值序列和 DLL 序列转化为句向量序列,使用 TextCNN 建立模型来进行挖矿恶意软件的早期检测。实验结果表明,CEDS 能够以 0.5106s 的平均时长和 96.75%的准确率判别一个软件样本是挖矿恶意软件还是良性软件。

关键词:挖矿恶意软件;动态分析;早期检测;句向量;深度学习
中图分类号 TP309

Cryptomining Malware Early Detection Method Based on SDR

ZHONG Kai[1],GUO Chun[1],LI Xianchao[2] and SHEN Guowei[1]
1 State Key Laboratory of Public Big Data,College of Computer Science and Technology,Guizhou University,Guiyang 550025,China
2 Guizhou Cloud Computing and Big Data Professional Master's Workstation,Guiyang 550014,China

Abstract Cryptomining malware aims to steal computing resources from devices to mine cryptocurrency,seriously compromising network security while consuming a large amount of computing resources. Current dynamic detection methods for cryptomining malware mainly rely on host behavior or network traffic collected during a long sample run for detection,which does not balance the timeliness and accuracy of detection. By analyzing the DLL(dynamic link library) called and the return value of the API called by the cryptomining malware at the early stage of operation,we propose an API sentence embedding method based on DLL and API return value(SDR),and further propose a cryptomining malware early detection method based on SDR(CEDS). CEDS uses SDR to convert the API name sequences,API returns value sequences,and DLL sequences generated in the early stages of software operation into sentence vector sequences,and uses TextCNN to build a model for early detection of cryptomining malware. Experimental results show that CEDS can determine whether a software sample is cryptomining malware or benign software with an average time of 0.5106 s and an accuracy of 96.75%.

Keywords Cryptomining malware,Dynamic analysis,Early detection,Sentence embedding,Deep learning

1 引言

挖矿恶意软件是一种在未经授权的情况下感染受害者设备并秘密利用设备的计算资源进行加密货币挖掘活动的恶意软件[1]。Malwarebyte 公司 2022 年的一份报告指出,随着加密货币价格的飙升,检测到的挖矿恶意软件数量同比增加了 300%以上[2]。因此,挖矿恶意软件检测是当前网络安全领域的一个重要课题。

近年来,基于机器学习的动态检测方法成为了挖矿恶意软件检测的主流研究方向。然而,常规动态检测方法由于需收集较长时间的软件行为数据,因此难以及时检出挖矿恶意软件。为了更及时地检出挖矿恶意软件,在挖矿恶意软件运行初期即能进行检测的早期检测方法正受到越来越多的关注。现有的挖矿恶意软件早期检测方法主要通过收集样本运行初期调用的 API 或产生的网络流量等行为数据来实现对挖矿恶意软件的早期检测。然而,减少行为数据收集时长虽有助于提升检测及时性,但也使得能用于分析的软件行为数据量减少,这对高检测率地检出挖矿恶意软件是一个挑战。

到稿日期:2023-12-06 返修日期:2024-04-30
基金项目:国家自然科学基金(62162009);贵州省高等学校大数据与网络安全创新团队(黔教技[2023]052);贵州省科技计划项目(黔科合平台人才 GHB[2023]001)
This work was supported by the National Natural Science Foundation of China(62162009),Big Data and Network Security Innovation Team of Universities in Guizhou Province([2023]052) and Science and Technology Program of Guizhou Province(GHB[2023]001).
通信作者:郭春(gc_gzedu@163.com)

因此，如何实现高准确率的挖矿恶意软件早期检测是当前研究亟待解决的一个难题。

针对该问题，本文分析了挖矿恶意软件在运行初期的DLL(Dynamic Link Library)调用和API(Application Programming Interface)返回值，并在此基础上设计了一种能够将API名称序列、DLL序列和API返回值序列融合为句向量序列的API句嵌入方法SDR(API Sentence embedding based on DLL and API Return value)。随后，本文结合SDR和TextCNN(Text Convolutional Neural Network)，提出了一种基于SDR的挖矿恶意软件早期检测方法CEDS(Cryptomining malware Early Detection method based on SDR)。与现有挖矿恶意软件早期检测方法相比，CEDS在准确性方面表现更佳。本文的主要贡献如下：

1)提出了一种基于API名称、DLL和API返回值的API句嵌入方法SDR。SDR的提出是基于对挖矿恶意软件运行初期的DLL调用和API返回值的分析结果，它能够在丰富API名称序列信息的同时，缩短所生成的特征向量的长度。

2)提出了基于SDR的挖矿恶意软件早期检测方法CEDS。CEDS利用SDR从软件运行初期产生的API名称序列、API返回值序列和DLL序列中提取特征向量，使用TextCNN建立高效的检测模型，能够实施挖矿恶意软件的早期检测。

3)在包含不同类别的挖矿恶意软件和良性软件的数据集上对CEDS进行实验测试。实验结果表明，通过从软件运行初期行为数据中提取的SDR句向量，CEDS能够实现对挖矿恶意软件快速且准确的检测。

2 相关工作

近年来，国内外研究人员提出了一些面向挖矿恶意软件的常规动态检测方法以及早期检测方法，并取得了一些研究成果。

2.1 常规动态检测方法

近年来，使用动态特征进行检测成为了挖矿恶意软件检测研究的主流。常规的动态检测方法收集挖矿恶意软件长时间运行过程中产生的网络流量或主机行为进行检测。Ahmad等[3]从收集的网络流量中提取TCP/IP流量特征，构建了一种基于树突状细胞算法的挖矿恶意软件检测模型。Muñoz等[4]使用挖矿恶意软件产生的NetFlow/IPFIX网络流量实现了对挖矿恶意软件的检测。Caprolu等[5]分析不同挖矿样本和良性样本之间进站、出站流量包大小和到达时间间隔的区别，使用机器学习算法建立检测模型。Tanana等[6]从软件的CPU使用率、网络连接和加密DLL调用中提取多个特征来进行挖矿恶意软件检测。Berecz等[7]提取了特定API和PE文件特征，使用机器学习算法建立检测模型。Darabian等[8]使用挖矿恶意软件的API和PE文件，结合深度学习模型进行挖矿恶意软件检测。Mani等[9]使用深度学习模型和处理器性能计数器来识别挖矿恶意软件的行为模式，从而实现检测。Karn等[10]使用n-gram从API序列中提取挖矿恶意软件的特征后，使用机器学习算法构建检测模型。常规动态检测方法通过收集较长时间的主机行为或流量进行检测，能够获取高的检测精度，但由于其行为数据收集时间较长，因此难以实现对挖矿恶意软件的及时检测。

2.2 早期检测方法

当前挖矿恶意软件早期检测方法的研究重点在于缩短样本的行为数据收集时间，以尽早检出挖矿恶意软件，从而减少挖矿恶意软件的挖矿行为所带来的硬件损耗。Sun等[11]提出了一种基于卷积函数的流量特征提取方法，通过提取样本所产生流量的前几个数据包的卷积特征，使用机器学习算法实现对挖矿恶意软件的检测。但挖矿恶意软件主要在连接矿池后才有网络流量产生，所以该方法需要较长的数据收集时间，导致其检测及时性较常规检测方法优势不明显。Cao等[12]提出了挖矿行为多样期的概念，通过n-gram和TF-IDF算法提取样本该时期内的API序列来进行挖矿恶意软件检测。该方法以4s作为API采集的截止时刻并使用随机森林算法(Random Forest，RF)能够在实验中获得98.33%的准确率，但该准确率并未显著高于常规动态检测方法。Cao等[13]通过挖矿恶意软件运行初期一定长度的API序列、API操作类别序列和DLL序列，结合深度学习模型实现挖矿恶意软件的早期检测。但该方法使用的是未去重的DLL序列，这对它的检测准确率有不利影响。

总体来说，一方面，常规动态检测方法需要收集较长时间的网络流量或主机行为才能获取高的检测率，检出时间较为滞后；另一方面，现有挖矿恶意软件早期检测方法存在着可用于分析的软件行为数据类型少或行为数据采集时间较长等问题，在检测率和及时性方面均有提升空间。

3 挖矿恶意软件早期行为分析

为了早期检测挖矿恶意软件，Cao等[12]提出了一种利用软件从开始运行到初次建立网络连接之间所产生的API序列进行检测的挖矿恶意软件早期检测方法。但该方法仅使用API名称，未利用调用API的DLL、API返回值等软件运行过程中所调用API的相关信息，因而需要以4s作为采集API的截止时刻才能获取高的检测率。为进一步提升挖矿恶意软件检测的及时性和检测率，本文同时使用API名称、调用API的DLL和API返回值来检测挖矿恶意软件。

3.1 挖矿恶意软件调用的DLL分析

DLL是一个包含了可由另一个模块(应用程序或DLL)使用的函数和数据的模块。Windows API是作为一组DLL实现的，于是任何使用Windows API且非静态编译的进程都需要调用DLL[14]。由于动态链接技术默认采用延迟绑定机制，若调用API的DLL不在内存中，则需要先将相应的DLL载入内存。挖矿恶意软件运行后，会执行进程注入、信息收集、系统监控、创建副本和连接矿池等行为[12]。上述各项行为实际上是在调用Windows API操作相关的系统资源。由于挖矿恶意软件连接矿池前执行的各行为所需的系统资源不同，因此它调用相关API的DLL的类型也不同，会涵盖内存管理、线程切换、网络连接等多种类型。因此，挖矿恶意软件运行过程中调用的API序列和调用这些API的DLL组成的DLL序列能够细粒度地反映它的行为。

本文做出以下设定：

1)直接从软件调用的API相关数据中提取的API名称序列和调用这些API的DLL组成的序列，称为原生API

序列和原生 DLL 序列。

2）在原生 DLL 序列中，当连续出现的 DLL 名称相同时，仅保留这些 DLL 名称中的一个，称为一次 DLL 调用。

3）参照 DLL 调用的定义处理原生 DLL 序列后获得的序列，称为 DLL 调用序列。

之前的恶意软件动态检测方法[13-15]在使用 DLL 序列时大多未考虑加载 DLL 所采用的延迟绑定机制，因而存在两个问题：1）原生 DLL 序列较长，不利于实现快速分类；2）原生 DLL 序列并不能准确体现 DLL 的实际加载过程。本文以从 XMRig 挖矿恶意软件运行后得到的 API 相关数据中提取的部分原生 API 序列和原生 DLL 序列为例（见表 1），该软件通过 kernelbase.dll 连续调用了多个 API，此时原生 DLL 序列中会连续出现重复的 kernelbase.dll。由于延迟绑定机制的存在，实际情况是系统仅需加载一次 kernelbase.dll[16]。因此，DLL 调用序列的长度较原生 DLL 序列的长度短。

表 1　XMRig 挖矿软件调用的部分原生 API 及 DLL

Table 1　Part of the native APIs and DLLs called by XMRig cryptomining malware

API	DLL
DllMain	ntdll.dll
RtlImageNtHeader	crypt32.dll
RtlInitializeCriticalSectionAndSpinCount	kernelbase.dll
RtlInitializeCriticalSectionAndSpinCount	kernelbase.dll
RtlAcquirePebLock	kernelbase.dll
RtlFindClearBitsAndSet	kernelbase.dll
RtlReleasePebLock	kernelbase.dll
RtlAllocateHeap	kernelbase.dll
RtlAllocateHeap	kernelbase.dll
RtlAllocateHeap	kernelbase.dll

信息熵可用来度量随机变量的不确定性[17]。一个特征的信息熵越高，表示该特征的不确定性越大（即信息量越大），其越适合作为分类特征。为比较相同软件调用的原生 DLL 序列和 DLL 调用序列所包含的信息量，在 VMware 中搭建的 Windows10 中运行了涉及多种加密货币的 100 款挖矿恶意软件和 100 款良性软件，并使用 API Monitor 收集各软件运行后 60s 内的 API 相关数据（即 API 名称、调用 API 的 DLL 和 API 返回值），从中提取原生 DLL 序列和 DLL 调用序列。其中挖矿恶意软件均在其默认 CPU 利用率阈值下运行。

表 2 列出了上述软件的原生 DLL 序列和 DLL 调用序列的信息熵均值。可以看出，良性软件和挖矿恶意软件在 60s 内的 DLL 调用序列的信息熵均值明显高于它们的原生 DLL 序列的相应值，表明将原生 DLL 序列处理为 DLL 调用序列可以减少原生 DLL 序列中的冗余信息。

表 2　良性软件和挖矿恶意软件各自两种 DLL 序列的信息熵均值

Table 2　Average entropy values of two DLL sequences of the benign software and cryptomining malware

	良性软件		恶意挖矿软件	
	原生 DLL 序列	DLL 调用序列	原生 DLL 序列	DLL 调用序列
信息熵均值	2.2890	2.6670	1.5350	2.0431

3.2　挖矿恶意软件运行中的 API 返回值分析

API 返回值指 API 执行完毕后所返回的值[18]。返回值的作用是将函数或方法的执行结果传递给调用者，以便调用者能够根据该值来判断函数或方法的执行是否成功或执行后续操作。因此，API 返回值可以被视为 API 执行情况的一种反馈，本文将其用于增加原生 API 序列的信息量。

由于 API 返回值内容没有固定的格式，因此需要先对 API 返回值进行规范化处理。本文将对从 3.1 节收集的 API 相关数据中提取的 API 返回值进行分析，并将其划分为表 3 所列的 6 种类型。其中，由于 Address，Number 和 String 类型的返回值内容不固定，将 Address，Number 和 String 类型的返回值分别用“address”“number”“string”代替。Boolean 和 Status 类型的返回值反映了 API 的执行结果，含义明确且可取的值有限，因此这两种类型返回值的内容不变。

表 3　API 返回值类型及示例

Table 3　API return value categories and examples

类型	示例
Address	0x
Number	12，−1，1.232344
Boolean	True，False
Status	status_device_does_not_exist，error_partial_copy
String	‘5’‘%’
Void	Null

需要注意，API 返回值反映的是 API 的执行结果，并非独立存在的信息。表 4 列出了 3.1 节中挖矿恶意软件调用频率前 5 的 API，其中调用次数排名第 5 的“RtlNtStatusToDosError”的返回值为 Status 类型，可取的值有限且含义明确。本文以该 API 为例，对比良性软件和挖矿恶意软件调用相同 API 所得到返回值的占比情况，结果如表 5 所列。从表中可以看到，良性软件和挖矿恶意软件调用该 API 所得到的返回值在占比上存在着明显区别。因此，在原生 API 序列中增加返回值信息有助于进一步提高对良性软件和挖矿恶意软件的区分度。

表 4　挖矿恶意软件调用次数前 5 的 API

Table 4　Top 5 APIs called by cryptomining malware

API	调用次数
realloc	2090967
CharNextA	1304656
RtlFreeUnicodeString	103537
lstrcpynA	82907
RtlNtStatusToDosError	82068

表 5　良性软件和恶意挖矿软件调用“RtlNtStatusToDosError”所得返回值的占比

Table 5　Percentage of return values from calls to “RtlNtStatus ToDosError” by benign software and cryptomining malware

（%）

良性软件返回值	占比	恶意挖矿软件软件返回值	占比
error_invalid_handle	0.389	error_success	0.335
error_success	0.316	error_file_not_found	0.042
error_file_not_found	0.192	error_invalid_parameter	0.003
error_no_token	0.017	error_access_denied	0.002
error_gen_failure	0.014	error_sxs_key_not_found	0.002
others	0.072	others	0.616

4 恶意挖矿软件早期检测方法 CEDS

本文提出的恶意挖矿软件早期检测方法 CEDS 的框架如图 1 所示，由数据收集、数据处理、模型训练和检测组成。CEDS 的检测流程为：首先收集待测样本的 API 相关数据，并从中提取原生 API 序列、DLL 调用序列和 API 返回值序列；然后使用 SDR 句向量嵌入方法生成这些序列对应的句向量序列；最后将其输入到构建的深度学习检测模型，得到检测结果。

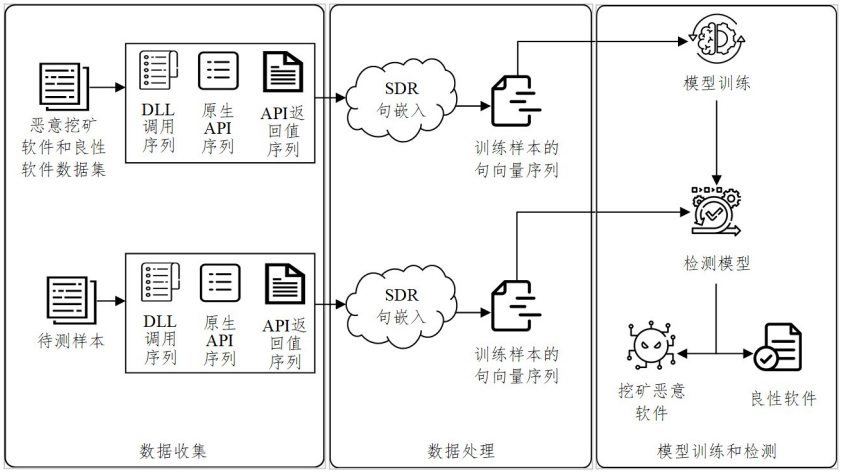

图 1 CEDS 框架

Fig. 1 Framework of CEDS

4.1 数据收集

数据收集阶段，CEDS 收集训练样本和待测样本的 API 相关数据，并从中提取原生 API 序列、DLL 调用序列和 API 返回值序列。由文献[12]可知，挖矿恶意软件在不同 CPU 利用率阈值下运行时所调用的 API 序列存在差异，故本文在收集挖矿恶意软件数据时，将收集每个样本分别在 25%，50%，75%和 100%这 4 个 CPU 利用率阈值下各 1 条 API 相关数据，即每个挖矿恶意软件收集 4 条 API 相关数据。

由于仅以时间阈值作为数据收集截止条件难以限制挖矿恶意软件在数据收集阶段执行的操作数量，本文采用多个截止条件来共同限制数据收集时长。

表 6 以 200 为单位长度区间，统计了样本首次调用 socket/WSASocket API 或 socket 相关 DLL 时已调用的 DLL 调用序列长度。其中，大部分恶意挖矿样本在首次调用 socket 相关 DLL 或 socket/WSASocket 时已调用的 DLL 调用序列长度在 800 以内。因此，为了防止出现单一截止条件造成数据收集时长过长的问题，一旦首次满足以下 3 个条件之一，CEDS 即结束数据收集。

1) DLL 调用序列达到预设长度 L；

2) 样本调用了 socket 相关 DLL 或 socket/WSASocket；

3) 样本运行时长达到预设时长 T。

表 6 样本首次调用 socket 相关 DLL 或 socket/WSASocket 时已调用的 DLL 调用序列长度分布

Table 6 Length of the called DLL call sequence when the samples first call the socket/WSASocket API or a socket-related DLL

DLL 调用序列长度区间	良性软件个数	恶意软件个数
0～200	4	12
200～400	3	34
400～600	3	18
600～800	2	20
800～1 000	1	6
1 000 以上	41	4

4.2 数据处理：API 句嵌入方法 SDR

数据处理阶段，CEDS 通过 API 句嵌入方法 SDR，将原生 API 序列、DLL 调用序列和 API 返回值序列处理为 SDR 句向量序列。

将 CEDS 在本阶段使用的 DLL 调用序列表示为 $D'=\{d_1,d_2,\cdots,d_n\}$，$n$ 为其长度。将原生 API 序列中的每个 API 名称视为一个单词，并将一次 DLL 调用中涉及的 API 视为一个 API 句子，可得 API 句子序列 $S=\{s_1,s_2,\cdots,s_n\}$，s_j 为 d_j 对应的 API 句子，有 $s_j=\{a_{m_{j-1}+1},a_{m_{j-1}+2},\cdots,a_{m_j}\}$，$j\in 1,2,\cdots,n$，$(m_j-m_{j-1})$ 为 s_j 中 API 的数量，$m_0=0$。将 s_j 中 API 对应的返回值按照 3.2 节中的方法进行处理，得到规范化后的返回值序列 $r_j=\{r_{m_{j-1}+1},r_{m_{j-1}+2},\cdots,r_{m_j}\}$，$j\in 1,2,\cdots,n$。

接下来，CEDS 使用 SDR 获得 s_j 对应的句向量。如图 2 所示，SDR 步骤为：

1) 添加返回值信息：将 API 句子 s_j 中每个 API 对应的返回值插入到 API 名称之后，获得带有返回值的 API 句子 $s_j'=\{a_{m_{j-1}+1},r_{m_{j-1}+1},a_{m_{j-1}+2},r_{m_{j-1}+2},\cdots,a_{m_j},r_{m_j}\}$，$j\in 1,2,\cdots,n$。

2) 获取词向量：CEDS 使用 Word2vec 算法获得 s_j' 中每个单词的词向量 $\boldsymbol{u}_i$（$i\in 2m_{j-1}+1,2m_{j-1}+2,\cdots,2m_j$，其中 $(2m_j-2m_{j-1})$ 为 s_j' 中单词的数量）以及和对应 DLL 调用 d_j 相同维度的词向量 $\boldsymbol{q}_j$。Word2vec 算法能够根据序列的上下文关系来生成词向量，并且具有计算速度快、所生成词向量维度低的优点[19]。

3) 获取句向量：通过式(1)计算 s_j' 对应的句向量 $\bar{\boldsymbol{u}}_j$。之后，将 $\boldsymbol{q}_j$ 和 $\bar{\boldsymbol{u}}_j$ 横向拼接起来，获得 s_j 对应的带有 API 返回值和 DLL 信息的句向量 $\boldsymbol{v}_j=(\boldsymbol{q}_j,\bar{\boldsymbol{u}}_j)$。

$$\bar{\boldsymbol{u}}_j=\frac{\sum_{i=2m_{j-1}+1}^{2m_j}\boldsymbol{u}_i}{m_j} \tag{1}$$

其中，$(2m_j-2m_{j-1})$ 为 s_j' 中单词的数量。

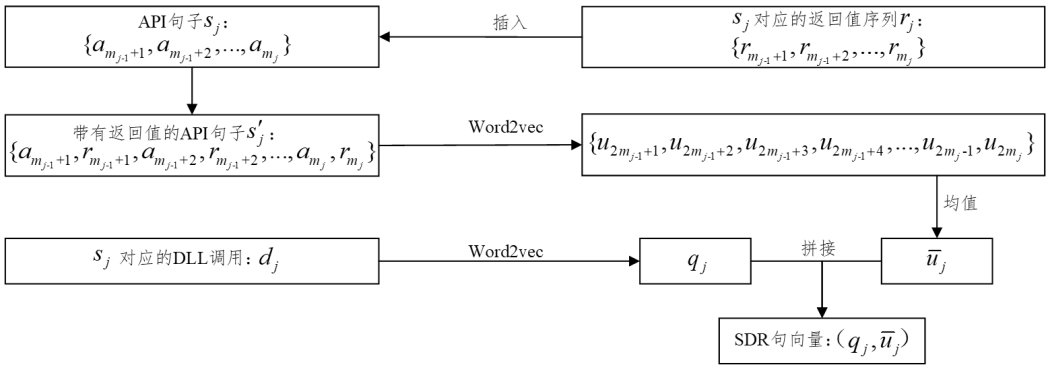

图 2　SDR 生成 API 句子 s_j 对应的句向量的过程示意

Fig. 2　Schematic of the process of generating sentence vector of API sentence s_j by SDR

通过上述步骤，CEDS 获得了 API 句子 s_j 对应的 SDR 句嵌入向量 $\boldsymbol{v}_j$ 以及 API 相关数据对应的 SDR 句向量序列 $\boldsymbol{V}=\{\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n\}$。SDR 句向量包含了样本运行初期的 API 名称、DLL 调用和 API 返回值等信息，所含软件运行信息多于原生 API 序列所含信息。

4.3　模型训练和检测

模型训练阶段，CEDS 利用 SDR 句向量序列训练一个 TextCNN 模型用于挖矿恶意软件检测。

TextCNN 是一种被广泛用于文本分类的 CNN(Convolutional Neural Networks)，它利用一个或多个一维滑动窗口从序列数据中提取特征[20]。一方面，CEDS 通过 Text-CNN可以提取 SDR 句向量序列的上下文关系；另一方面，TextCNN 模型结构简单、参数少、训练速度快，能省去传统机器学习中耗时的特征工程，使 CEDS 具备较高的检测效率。

CEDS 采用 SDR 句嵌入方法将 API 相关数据转换为向量 $\boldsymbol{V}$，作为 TextCNN 的输入层。由于 CEDS 的目的是实现挖矿恶意软件和良性软件的二分类，本文将良性软件的 API 相关数据标记为 0，挖矿恶意软件的 API 相关数据标记为 1，从而得到数据集 $D=\{(\boldsymbol{V}_1, y_1), (\boldsymbol{V}_2, y_2), \cdots, (\boldsymbol{V}_t, y_t)\}$，$y_i \in \{0, 1\}$，$\boldsymbol{V}_i$ 为第 i 个样本对应的 SDR 句向量序列，t 为样本数量。

TextCNN 中卷积层通过在 SDR 句向量序列上移动卷积核来提取局部特征，并根据输入序列参数计算输出特征图。TextCNN 中池化层用于对特征图进行降维。TextCNN 的全连接层将前一层所有神经元与当前层全部相连。最后，Text-CNN的输出层使用 softmax 函数归一化全连接层的输出，以得到属于各类别的概率。

在检测阶段，CEDS 将待测样本的原生 API 序列、DLL 调用序列和 API 返回值序列输入训练好的模型进行判别，从而实现挖矿恶意软件和良性软件的二分类。

5　实验和结果

5.1　实验环境与样本

使用配置为 Intel 10700F CPU，16 GB RAM 的主机采集样本运行数据，这些样本运行在 VMware 中部署的 Windows 10 上。使用 API Monitor 收集各样本运行后 60 s 内的 API 相关数据。数据处理、模型训练和检测步骤所使用的主机配置为 AMD EPYC 7742 CPU，0.98 TB RAM。开发环境为 Python 3.8，scikit-learn 0.24.1 和 glove 1.0.1。实验中 TextCNN 的超参数设置为：学习率＝0.0001，$Epoch=15$，$Batch\ size=20$。

为检验 CEDS 对已知和未知挖矿恶意软件的检测能力，我们搜集了 722 款良性软件和 237 款挖矿恶意软件。良性软件样本来自 360 软件管家、腾讯软件中心和联想应用商店等平台，涵盖了办公软件、聊天通讯和安全杀毒等类别；挖矿恶意软件搜集自微步在线云沙箱、any.run 和 GitHub 等平台，这些软件活跃于 2019－2023 年，涵盖了 BTC，XMR，BCD 和 VTC 等主流数字货币。根据 4.1 节的数据收集方法，我们收集 237 款挖矿恶意软件在 4 个不同 CPU 利用率阈值下的不同运行数据(共 948 条 API 相关数据)，以及 722 款良性软件的各 1 条运行数据(共 722 条 API 相关数据)，所收集 API 相关数据的训练集和测试集划分情况如表 7 所列。训练集包含 189 款挖矿恶意软件的各 2 条 API 相关数据和 288 条良性软件的 API 相关数据，已知测试集包含上述 189 款挖矿恶意软件剩余的 378 条 API 相关数据和 288 条良性软件的 API 相关数据，未知测试集包含 146 条良性软件的 API 相关数据和与训练样本不同的 48 款挖矿恶意软件的 192 条 API 相关数据。

表 7　实验样本的 API 相关数据数量及数据集划分

Table 7　Number of API-related data of experimental sample and dataset partitioning

类别	训练集	已知测试集	未知测试集	总量
办公软件	46	46	23	115
聊天通讯	33	33	16	82
安全杀毒	16	16	8	40
影音软件	48	48	24	120
游戏	32	32	16	80
压缩软件	18	18	11	47
压力测试	5	5	4	14
加密软件	20	20	10	50
其他应用	70	70	34	174
良性软件总数	288	288	146	722
挖矿恶意软件	378	378	192	948
所有样本总数	666	666	338	1670

5.2 评估指标

本文使用准确率(Accuracy)、精确率(Precision)、召回率(Recall)和F1-score(F1)对方法的检测效果进行评估。

5.3 实验结果

为了分析不同DLL调用序列长度 L 对CEDS检测结果的影响,本文对CEDS在不同 L 值下的检测结果进行测试。结合4.1节的分析,本文将数据收集预设时长 T 设定为1s,观察CEDS在不同 L 值下的检测结果。

CEDS在已知测试集上的检测结果如表8所列,在{100,200,300,400,500,600}的取值范围内,CEDS取 $L=400$ 时检测效果最佳。通过分析发现,这是因为当 L 在{100,200,300,400}内取值时,大部分挖矿恶意软件仍处于网络连接前的准备阶段,它们在该阶段大量调用与字符串和注册表相关的API,这与良性软件运行初期所调用的API区别明显。而随着挖矿恶意软件的持续运行,部分挖矿恶意软件在 L 为{500,600}时建立了网络连接并开始网络交互,这部分挖矿恶意软件在此阶段所调用的API序列与良性软件调用的API序列的区别较前面阶段小。

表8 L 取不同值时在已知测试集上的检测结果

Table 8 Detection results obtained by CEDS with different L values on the known test set

L	*Accuracy*	*Recall*	*Precision*	*F1*-score
100	0.9399	0.9656	0.9311	0.9481
200	0.9610	0.9762	0.9560	0.9660
300	0.9655	0.9762	0.9634	0.9698
400	**0.9805**	**0.9868**	**0.9790**	**0.9829**
500	0.9715	0.9868	0.9638	0.9752
600	0.9640	0.9841	0.9538	0.9688

为了验证SDR句嵌入方法的有效性,本文取 $L=400$,在已知测试集上测试使用原生API句向量(仅使用API名称获得的句向量)、带有返回值的API句向量(使用API名称和API返回值获得的句向量)和SDR句向量的检测效果,结果如表9所列。可以看到,使用SDR句向量所得到的 *Accuracy*,*Precision*,*Recall* 和 $F1$ 值均优于使用其他两种句向量所得到的相应指标值。

表9 $L=400$ 时在已知测试集上使用不同句向量的检测结果

Table 9 Detection results using different sentence vectors on the known test set when $L=400$

词向量	*Accuracy*	*Recall*	*Precision*	*F1*-score
原生API句向量	0.9084	0.8783	0.9568	0.9159
带有返回值的API句向量	0.9354	0.9101	0.9745	0.9412
SDR句向量	**0.9805**	**0.9868**	**0.9790**	**0.9829**

本文还对比了不同挖矿恶意软件检测方法在已知测试集上的检测效果。文献[7-8,10,12]在对比实验中均使用表7中样本产生的原生API序列。由于文献[7]和文献[8]均未明确指出其方法的数据收集时长,因此本文在对比实验中将它们的收集时长设定为与CEDS的数据收集时长阈值相同(1s),文献[10]采用其方法设定的数据收集时长60s,文献[12]采用其获得最佳 *Accuracy* 的数据收集时长阈值(4s)。对比结果如表10所列,表中平均总耗时为单个测试样本的平均数据收集时间、平均特征计算时间与平均分类时间之和。从表10可知,在已知测试集上,CEDS的 *Accuracy* 和 *F1-score* 均高于其他检测方法,并且平均总耗时最短。

表10 不同挖矿恶意软件检测方法在已知测试集上的检测结果

Table 10 Detection results of different crytomining malware detection methods on the known test set

方法	*Accuracy*	*F1*-score	平均数据收集时间/s	平均特征计算时间/s	平均分类时间/s	平均总耗时/s
Karn(DT)[10]	0.9144	0.9283	60.0000	0.3309	1.0864	61.4173
Berecz(RF)[7]	0.9294	0.9403	**1.0000**	1.4367	**0.0014**	2.4381
Darabian(ATT-LSTM)[8]	0.9554	0.9585	**1.0000**	**0.0036**	0.1457	1.1493
CEDMB(RF)[12]	0.9670	0.9707	3.0695	0.0636	**0.0014**	3.1345
CEDS	**0.9805**	**0.9829**	**0.4635**	0.0169	0.0330	**0.5134**

CEDS在未知测试集上的检测结果如表11所列。

表11 L 取不同值时CEDS在未知测试集上的检测结果

Table 11 Detection results obtained by CEDS with different L values on the unknown test set

L	*Accuracy*	*Recall*	*Precision*	*F1-score*
100	0.9349	0.9583	0.9293	0.9436
200	0.9556	0.9740	0.9492	0.9614
300	0.9586	0.9740	0.9541	0.9639
400	**0.9675**	**0.9740**	**0.9689**	**0.9714**
500	0.9645	0.9740	0.9639	0.9689
600	0.9586	0.9635	0.9635	0.9635

取值中,$L=400$ 时CEDS所得到的 *Accuracy*,*Precision*,*Recall* 和 $F1$ 值最佳。此外,本文取 $L=400$,在未知测试集上分别使用原生API句向量和带有返回值的API句向量和SDR句向量进行了实验。如表12所列,使用SDR句向量可以获得优于原生API句向量和带有返回值的API句向量的检测精度。

表12 $L=400$ 时在未知测试集上使用不同句向量的检测结果

Table 12 Detection results using different sentence vectors on the unknown test set when $L=400$

词向量	*Accuracy*	*Recall*	*Precision*	*F1-score*
原生API句向量	0.8964	0.8542	0.9591	0.9036
带有返回值的API句向量	0.9320	0.9115	0.9669	0.9383
SDR句向量	**0.9675**	**0.9740**	**0.9689**	**0.9714**

此外,本文还对比了不同挖矿恶意软件检测方法在未知测试集上的检测效果,结果如表13所列。与其他几种检测方法相比,得益于DLL和API返回值的加入,以及SDR句嵌入方法缩短了特征向量的长度,CEDS能够在检测平均总耗时最短的同时,获得最高的 *Accuracy* 和 *F1-score* 值。

表 13　不同挖矿恶意软件检测方法在未知测试集上的检测结果

Table 13　Detection results of different crytomining malware detection methods on the unknown test set

方法	*Accuracy*	*F1-score*	平均数据收集时间/s	平均特征计算时间/s	平均分类时间/s	平均总耗时/s
Karn(DT)[10]	0.9083	0.9169	60.0000	0.3454	0.6156	60.9610
Berecz(RF)[7]	0.8935	0.9053	1.0000	1.2904	**0.0012**	2.2916
Darabian(ATT-LSTM)[8]	0.9290	0.9381	1.0000	**0.0041**	0.1478	1.1519
CEDMB(RF)[12]	0.9379	0.9431	2.9658	0.0657	**0.0012**	3.0327
CEDS	**0.9675**	**0.9714**	**0.4541**	0.0235	0.0330	**0.5106**

结束语　挖矿恶意软件早期检测，对于减少受害主机非必要的硬件损耗和维护网络安全有着重要意义。本文提出了一种综合使用原生 API 序列、DLL 调用序列和 API 返回值序列的 API 句嵌入方法 SDR，并进一步提出了挖矿恶意软件早期检测方法 CEDS。在 722 款良性软件和 237 款挖矿恶意软件产生的 API 相关数据组成的数据集上的实验结果显示，CEDS 在检测准确率和及时性方面均有较好的表现。

然而，随着挖矿恶意软件检测与反检测对抗的持续，后续的挖矿恶意软件必然会集成越来越多的逃避检测技术来绕过包含本方法在内的检测技术。因此，未来我们将研究能够应对后期可能出现的集成了多种逃避技术的挖矿恶意软件的检测方法。

参 考 文 献

[1] TEKINER E, ACAR A, ULUAGAC A S, et al. SoK: cryptojacking malware[C]// IEEE European Symposium on Security and Privacy. 2021: 120-139.

[2] Malwarebytes. 2022 THREAT REVIEW [EB/OL]. [2023-08-18]. https://www.malwarebytes.com/resources/malwarebytes-threat-review-2022/index.html.

[3] AHMAD A, SHAFIUDDIN W, KAMA M N, et al. A New Cryptojacking Malware Classifier Model Based on Dendritic Cell Algorithm[C]// International Conference on Vision, Image and Signal Processing. 2019: 84: 1-84.

[4] MUÑOZ J Z I, SUÁREZ-VARELA J, BARLET-ROS P. Detecting cryptocurrency miners with NetFlow/IPFIX network measurements[C]// 2019 IEEE International Symposium on Measurements & Networking (M&N). IEEE, 2019: 1-6.

[5] CAPROLU M, RAPONI S, OLIGERI G, et al. Cryptomining makes noise: Detecting cryptojacking via Machine Learning[J]. Computer Communications, 2021, 171: 126-139.

[6] TANANA D, TANANA G. Advanced behavior-based technique for cryptojacking malware detection[C]// International Conference on Signal Processing and Communication Systems. 2019: 84: 1-84.

[7] BERECZ G J, CZIBULA I G. Hunting traits for cryptojackers [C]// Proceedings of the 16th International Joint Conference on e-Business and Telecommunications. 2019: 386-393.

[8] DARABIAN H, HOMAYOUNOOT S, DEHGHANTANHA A, et al. Detecting cryptomining malware: a deep learning approach for static and dynamic analysis[J]. Journal of Grid Computing, 2020, 18(2): 293-303.

[9] MANI G, PASUMARTI V, BHARGAVA B, et al. Decrypto pro: deep learning based cryptomining malware detection using performance counters[C]// 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). IEEE, 2020: 109-118.

[10] KARN R R, KUDVA P, HUANG H, et al. Cryptomining detection in container clouds using system calls and explainable machine learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(3): 674-691.

[11] SUN P F, LYU M D, LI H, et al. An early stage convolutional feature extracting method using for mining traffic detection[J]. Computer Communications, 2022, 193: 346-354.

[12] CAO C B, GUO C, SHEN G W, et al. Cryptomining Malware Early Detection Method in Behavioral Diversity Period[J]. Acta Electronica Sinica, 2023, 51(7): 1850-1858.

[13] CAO C B, GUO C, LI X C, et al. Cryptomining Malware Early Detection Method Based on AECD Embedding[J]. Journal of Frontiers of Computer Science and Technology, 2024, 18(4): 1083-1093.

[14] Microsoft. Dynamic-Link Libraries (Dynamic-Link Libraries) [EB/OL]. [2023-08-18]. https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-libraries.

[15] IJAZ M, DURAD M H, ISMAIL M. Static and Dynamic Malware Analysis Using Machine Learning[C]// 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST- 2019). 2019.

[16] Microsoft. How to: Call Windows APIs (Visual Basic) [EB/OL]. [2023-03-30]. https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/com-interop/how-to-call-windows-apis.

[17] SHANNON C E. A Mathematical Theory of Communication [J]. The Bell System Technical Journal, 1948, 27(3): 379-423.

[18] Microsoft. Methods (C# Programming Guide) [EB/OL]. [2023-03-30]. https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods.

[19] MIKOLOVT, SUTSKEVER I, CHEN K, et al. Distributed representations of words and phrases and their compositionality [C]// Proceedings of the 26th International Conference on Neural Information Processing Systems. Curran Associates Inc., 2013: 3111-3119.

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

ZHONG Kai, born in 1997, postgraduate. His main research interests include computer network and information security.

GUO Chun, born in 1986, Ph.D, professor. His main research interests include malicious code detection and intrusion detection.

（责任编辑：柯颖）