doi: 10.3969/j.issn.0490-6756.2020.04.010

基于最长频繁序列挖掘的恶意代码检测

黄琨茗, 张 磊, 赵 奎, 刘 亮
(四川大学网络空间安全学院, 成都 610065)

摘 要: 基于动态 API 序列挖掘的恶意代码检测方法未考虑不同类别恶意代码之间的行为差别,导致代表恶意行为的恶意序列挖掘效果不佳,其恶意代码检测效率较低. 本文引入面向目标的关联挖掘技术,提出一种最长频繁序列挖掘算法,挖掘最长频繁序列作为特征用于恶意代码检测. 首先,该方法提取样本文件的动态 API 序列并进行预处理;然后,使用最长频繁序列挖掘算法挖掘多个类别的最长频繁序列集合;最后,使用挖掘的最长频繁序列集合构造词袋模型,根据该词袋模型将样本文件的动态 API 序列转化为向量,使用随机森林算法构造分类器检测恶意代码. 本文采用阿里云提供的数据集进行实验,恶意代码检测的准确率和 AUC(Area Under Curve)值分别达到了 95.6%和 0.99,结果表明,本文所提出的方法能有效地检测恶意代码.

关键词: 恶意代码; 最长频繁序列; 序列挖掘; 词袋模型; 随机森林算法
中图分类号: TP391.1 **文献标识码**: A **文章编号**: 0490-6756(2020)04-0681-08

Malware detection based on longest frequent API sequence

HUANG Kun-Ming, *ZHANG Lei*, *ZHAO Kui*, *LIU Liang*
(College of Cybersecurity, Sichuan University, Chengdu 610065, China)

Abstract: Existing malware detection methods based on dynamic API sequence mining do not consider the behavior differences between different types of malware, resulting in low efficiency of malicious code detection. In this paper, an object-oriented association mining technology is introduced, and a malware detection method is proposed based on the longest frequent sequence mining algorithm of the same category. First, the method extracts the dynamic API sequences of sample files and preprocesses them; then, the longest frequent sequence mining algorithm is used to mine the longest frequent sequence sets of multiple categories; finally, the longest frequent sequence set is used to construct the word bag model, according to the word bag model, the dynamic API sequences of sample files are transformed into vectors, so that the longest frequent sequence mining algorithm can be used to mine the longest frequent sequence sets of multiple categories. Random forest algorithm is used to construct classifier to detect malicious code. In this paper, we use the data set provided by the Aliyun Security Algorithms Challenge. The accuracy rate and AUC of malware detection are 95.6% and 0.99, respectively. The results show that the proposed method can effectively detect the malware.

Keywords: Malware detection; Longest frequent API sequence; Sequence mining; Bag-of-word; Malware detection

收稿日期: 2019-10-17
基金项目: 四川省重大科技专项(2018GZDZX0010); 国家重点研发计划项目(2017YFB0802900)
作者简介: 黄琨茗 (1992—), 男, 四川南充蓬安人, 硕士研究生, 研究方向为恶意代码检测.
通讯作者: 赵奎. E-mail: zhaokui@scu.edu.cn

1 引 言

恶意代码包括病毒、蠕虫、木马、后门、Rootkit、流氓软件以及广告软件等. 目前恶意代码的检测主要分为基于可移植执行文件及其反汇编文件的静态检测和基于恶意代码运行过程的行为特征的动态检测.

静态检测方法[1]的特征主要包括可移植执行文件及其反汇编文件的字节码、汇编指令、导入函数和分节信息等. 该方法对使用了代码混淆技术[2]的已知恶意代码和未知的恶意代码作用有限.

为克服基于静态检测方法的不足,研究者们提出基于行为的方法,相对于特征码,程序的行为更加固定. 恶意代码往往通过执行关键 API 调用函数实现恶意行为,系统 API 调用序列可作为恶意代码检测方法的最有效特征之一,API 调用序列特征被研究者广泛使用. 2013 年 Ahmadi 等人[3]使用迭代模式的挖掘方法来挖掘频繁序列,他们认为恶意软件编写者调用重复性的 API,特别是循环加密或解密和感染. 2014 年 Cho 等人[4]应用序列队的方法检测变种恶意代码,该方法能够发现恶意软件 API 调用序列中的相同部分并检测出相似行为. 2015 年 Ki 等人[5]使用 DNA 序列抽象出恶意软件的 API 调用序列模式,得出相同的恶意代码功能可以包含在不同的类别中,Pirscoveanu 等人[6]用取前两百个不重复 API 函数,API 函数的频繁度等作为特征,用于恶意代码分类. 2016 年,Kolosnjaji 等人[7]提取动态 API 序列的 kernel 系统调用序列,使用 N-Gram 和 CNN 提取序列特征输入到 LSTM 网络中进行恶意代码检测. 2018 年,荣俸萍等人[8]使用模式挖掘算法得到 API 调用序列并结合随机森林模型来识别恶意代码,该方法使用改进的沙箱模型能有效检测逃避性样本. 以上基于 API 序列的恶意代码检测都没有考虑不同类别恶意代码的行为差异,而把所有恶意代码集中到一起进行特征挖掘,不能充分挖掘能代表恶意行为的序列模式,以至于不能有效的检测恶意代码.

基于恶意代码的行为依赖图[9-10]需通过动态污点分析的方法获取,该方法需要人工操作,速度慢效率低,还可能造成污点爆炸等问题. N-Gram 方法[11-13]被广泛用于 API 序列的特征提取,N-Gram 提取的子序列维度很大. 其中,文献[14]采用冗余子序列去除法处理子序列维度很大的问题,使用信息增益筛选 N-Gram 序列,输入到 LSTM 网络模型中进行分类,其 API 序列特征可解释性差,恶意样本在 API 序列中插入不相关 API 来逃避检测,会导致 N-Gram 失效.

为了改进以上基于恶意代码行为的动态检测技术的不足,本文选择挖掘不同种类恶意代码的最长频繁序列以表征恶意代码的恶意行为,提出了一种基于多种类恶意代码的最长频繁序列集合的恶意代码检测方法. 对于样本文件在沙箱中动态运行产生的系统 API 调用序列,使用改进的序列挖掘算法挖掘出同一类型的恶意代码的最长频繁序列,并将挖掘的多种类的最长频繁 API 序列集合合并,构造词袋模型,基于该词袋模型使用向量表征样本文件的动态 API 序列,构造随机森林分类器用于恶意代码的检测. 本文的主要贡献如下.

1) 采用了挖掘不同种类恶意代码的最长频繁序列的策略:不同种类的公共恶意行为往往有所区别,勒索软件会进行大量的注册表读写和文件读写操作,蠕虫病毒有大量自我复制和网络行为,特洛伊木马监听键盘记录和窃取账号密码等,挖掘不同种类的最长频繁序列集合能代表不同恶意行为模式.

2) 提出了新的最长频繁序列挖掘算法:本文提出的最长频繁序列挖掘算法在引入面向目标的关联挖掘技术的基础上进行了去重、剪枝的过程,保留关键 API 序列用于挖掘,大大减小了挖掘的时间复杂度,在运行过程中基于深度优先策略,通过判断候选项是否属于可扩展项,边挖掘边剪枝,进一步减小了挖掘过程中的时间和空间消耗.

3) 提出了一种基于最长频繁序列集合表示样本文件的方法:样本的动态 API 序列可以看作具有时间序列的一维文本,单个最长频繁序列可以看成能表征文本类别的一个特征. 本文使用挖掘的最长频繁序列集合构造词袋模型,基于此模型把样本文件动态 API 序列转化为特征向量,特征向量能有效的包含样本文件的恶意行为.

2 最长频繁序列挖掘与恶意代码检测

2.1 方法概述

本文提出的恶意代码检测方法主要检测 Windows 系统的恶意可执行文件 PE(Portable Executable)文件,采用本文提出的序列挖掘算法挖掘出各个类别的最长频繁序列,使用最长频繁序列集

合构造词袋模型，把样本文件由向量表示，使用随机森林构造分类器进行恶意代码检测. 如图 1 方法实现流程图分为两个阶段，即特征挖掘和恶意代码检测阶段.

1）特征挖掘阶段：首先提取样本文件的动态 API 调用序列，对动态 API 进行预剪枝；其次，基于同一类别的恶意代码挖掘出对恶意代码和正常样本有区分作用的最长频繁 API 序列；然后，合并挖掘出的不同类别恶意代码最长 API 序列集合，构造词袋模型；最后，根据词袋模型将样本文件的动态 API 转化为向量.

2）恶意代码检测阶段：首先使用训练样本构造分类器；然后提取测试样本的动态 API 调用序列，进行简单的去重处理；最后把去重的样本文件的 API 序列转化为向量，用训练的分类器模型进行恶意代码检测.

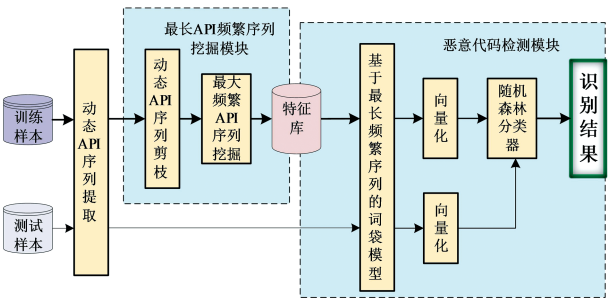

图 1　方法实现流程图

Fig. 1　Flow chart of the method

2.2　动态 API 提取及预剪枝

本文使用系统 API 调用序列表征恶意代码的行为，基于 Cuckoo Sandbox 搭建恶意代码动态分析系统，为了更好地模拟正常用户的环境，在客户机上安装一些常用的应用软件，并留下使用记录. 该分析系统通过 Hook 技术提取恶意代码运行过程的 API 调用，形成样本报告，实现了 API 序列的批量自动提取.

2.2.1　*动态 API 预剪枝*　由于恶意软件的动态 API 序列往往比较长，而恶意代码要完成恶意行为必须执行关键的 API 函数，过多的非关键 API 函数在模式挖掘过程中会产生大量冗余且无益的序列，这些会大大增加特征挖掘过程的时间消耗. 所以，需要删除动态 API 序列中的非关键的 API 函数，对不同类别的动态 API 集合序列进行预剪枝，预剪枝分为动态 API 去重和关键 API 函数筛选.

2.2.2　*动态 API 去重*　序列模式中存在同一种 API 序列模式被频繁调用，存在信息冗余. 为了减少挖掘过程中的复杂度，并且要保证保留的信息不减少，本文采用了连续 API 序列模式去除方法[13]，即只保留连续重复出现的 API 子序列中的单个子序列，如表 1 展示了 API 序列去重前后的 API 序列.

表 1　API 序列去重

Tab. 1　Eliminating the duplicated API sequence

去重前的 API 序列	去重后的 API 序列
$[API_1, API_1, API_1, API_2, API_4, API_2]$	$[API_1, API_2, API_4, API_2]$
$[API_3, API_3, API_4, API_4, API_3, API_4, API_5]$	$[API_3, API_4, API_5]$

2.2.3　*关键 API 函数筛选*

定义 1　设 $f=\{s_1, s_2, \cdots, s_n\}$ 是同一个类别去重后的恶意代码样本动态 API 序列集，$h=\{h_1, h_2, \cdots, h_m\}$ 是正常样本去重后的动态 API 序列集. 令 a_k 表示该类别动态 API 序列中的 API 调用序列第 k 个 API 调用；$freq(a_k, f)$ 表示 a_k 在恶意代码类别 f 中的加权频率；$freq(a_k, h)$ 表示 a_k 在 h 中的加权频率. 它们的相关度的计算公式如下.

$$freq(a_k, f)=\frac{SN(a_k, f)}{SN(f)}\times\frac{AN(a_k, f)}{AN(f)} \tag{1}$$

$$freq(a_k, h)=\frac{SN(a_k, h)}{SN(h)}\times\frac{AN(a_k, h)}{AN(h)} \tag{2}$$

式(1)中，$SN(a_k, f)$ 表示包含 a_k 且属于类别 f 的样本数量；$SN(f)$ 表示类别 f 中的样本总数量；$AN(a_k, f)$ 表示 a_k 在恶意类别 f 的样本集中的动态 API 序列中出现的次数；$AN(f)$ 表示恶意类别 f 的样本文件 API 调用序列中 API 调用的总次数. 式(2)中，$SN(a_k, h)$ 表示样本集 h 中包含 a_k 的样本数量；$SN(h)$ 表示 h 中的样本总数量；$AN(a_k, h)$ 表示 a_k 在 h 的样本集中的动态 API 序列中出现的次数；$AN(h)$ 表示 h 的样本文件 API 调用序列中 API 调用的总次数. 本文当 $freq(a_k, h)>freq(a_k, f)$ 时，可对恶意代码类别集合中所有样本进行合理剪枝.

2.3　最长频繁序列集合挖掘

恶意序列模式挖掘的主要目的是挖掘出对恶意样本和正常样本文件有区分能力的最长恶意 API 调用序列模式. 本文基于 Freespan 算法思想[14]，采取树深度优先策略，提出了基于有限 API 序列集挖掘出最长频繁 API 子序列的算法.

2.3.1　*最长频繁 API 序列模式挖掘相关概念的定义*　我们先给出最长频繁子序列挖掘算法的相关概念定义如下.

定义 2 支持度. 一个序列 S 在序列集 T 中的支持度等于该序列集中包含序列 S 的个数,即该序列的频繁度,记 $Sup_T(S_a)$是数据集 T 中所包含的 S_a序列的个数,其中 $S\in T$,$S_a=\mathrm{sub}(S)$.

定义 3 最大不支持度 Sup_{non}:一个序列 S 在序列集 T 中的支持度等于该序列集中不包含序列 S 的最小个数.

定义 4 频繁序列. 对于子序列 S_a,如果其在序列集 T 中支持度大于或等于设定的最小支持度阈值 $Sup_{\min}$,即 $Sup_T(S_a)\geqq Sup_{\min}$,那么称 S_a 为序列集 T 的频繁序列.

定义 5 最长频繁序列. 设 Sa 是频繁序列,如果不存在频繁序列 S_b是 S_a的超集,则 S_a被称为最长频繁序列.

定义 6 候选扩展项. 如果一个频繁序列的最后一项为 X_2,如果频繁一项集序列 $S_1=[X_1,X_2,X_7,\cdots,X_{59}]$,则候选可扩展项序列为 S_1,其中序列的每一项均是候选可扩展项.

定义 7 可扩展项. 设序列 s 为频繁序列,如果 s 最后面扩展一项后支持度依然大于或等于 $Sup_{\min}$,即扩展后得到的序列还是频繁序列,该项被称为 s 的可扩展项.

定义 7 给出了扩展项的概念,可知频繁 API 序列顺序添加单个候选扩展 API 后得到的新 API 序列还是频繁 API 序列,则该 API 项是原 API 序列的可扩展 API 项. 根据最长频繁子序列的定义可知:如果频繁 API 序列没有扩展项且其不是其他最长频繁 API 序列的子集,则序列 s 为最长频繁 API 子序列. 表 2 频繁 1 项 API 序列的可扩展项所示,展示 API 序列集中频繁 1-项 API 序列集的候选可扩展项及其可扩展项,例如 API_4 可扩展项集合有{API_2,API_5},表中没有可扩展项的记为"∅".

表 2 频繁 1 项 API 序列的可扩展项

Tab. 2 Extensible sequences of one-frequent sequence

频繁 API 序列 1 项集	候选可扩展项	可扩展 API 项
API_2	API_4,API_5,API_7	API_4,API_5, API_7
API_4	API_2,API_5	API_2
API_5	API_2,API_4,API_7	API_2,API_4
API_7	API_2,API_4	∅

2.3.2 最长频繁 API 序列挖掘算法 本文设计了一种最长 API 频繁子序列挖掘算法,利用 Redis 队列结构边挖掘边过滤,算法分两部分.

第一部分为最长频繁序列挖掘部分,即把产生的最长频繁序列加入到 Redis 队列头部,本文采用字典来记录 i-项频繁 API 序列的可扩展项信息,字典名为 extend_address,以"API_2":{s_1:address},{s_2:address},…,

{s_i:address},…,"API_n":{s_1:address},{s_2:address},…,{s_i:address}}的形式存储其信息,s_i 与 address 分别指包含该可扩展项的动态 AP 序列和其在对应动态 API 序列中的位置信息. 例如{1:{s_1:1,s_2:1,s_3:2,s_5:7},2:{ s_2:2,s_3:4,s_5:9}},其中的可扩展项在动态 API 序列编号为 1,2,可扩展项 1 分别在 s_1、s_2、s_3 和 s_5 动态序列的 1、1、2 和 7 位置,可扩展项 2 分别在 s_2、s_3 和 s_5 动态序列的 2、4、9 位置. 挖掘算法采用深度优先搜索的策略来遍历序列模式的搜索空间.

搜索过程如下:首先搜索所有 1-项频繁 API 子序列集并将其作为候选可扩展项,然后以 1-项频繁 API 子序列作为第一层树节点,最左边第一个树节点添加候选可扩展项作为后缀搜索 2-项频繁 API 子序列,把没有扩展项的最长频繁 API 序列存到数据库队列头部,把搜索得到的频繁二项序列作为第二层树节点,同理从第二层最左边节点继续搜索频繁三项序列,最后递归搜索直到搜索的所有 i-项频繁 API 子序列的扩展项都为"∅"时停止搜索.

最长频繁 API 序列挖掘算法的伪代码如算法 1.

算法 1 最长 API 频繁序列挖掘算法

输入 类别 $f=\{S_1,S_2,\cdots,S_n\}$,关键 API 序列 important_api=[api_1,api_2, …,api_n],初始第一层节点 init_seq,最小支持度 $Sup_{\min}$和最小不支持度 Sup_{non}

输出 Redis 数据库存储最长频繁 API 序列集 $f'=[s_1,s_2,\cdots,s_k]$

```
1) one_item_frequent=[]   # 全局变量
2) function main()   # 此为程序入口函数
3) extend_address = find_one_seq()
4) recusive_seq(init_seq, extend_address)
5) functionfind_one_seq ()   # 获取频繁一项集在动态 API 序列位置
6)      for api in important_api:
```

```
7)          for Si in f:
8) extend_address={Si:位置}    # 记录 api 在 Si首次出现的位置
9)          if the number of Si contains api> supmin:
10) one_item_frequent. append(api)    # 包含 api 的序列个数大于支持度
11)      return exend_address
12) function recusive _seq(init_seq, extend_address): # 递归得到所有的最长频繁序列
13) can_extension=find_extension_seq(seq, extend_address)
14)         if can_extension is not null:
15)             for eachseq in can_extension:
                #递归获得最长频繁序列
16) can_extension=find_extension_seq(seq, extend_address)
17) else :
18)             Redis. lpush(f', seq)
19) functionfind_extension_seq (extend_address) #得到序列的可扩展一项集
20) for item in one_item_frequent:
21)             for Si in extend_address:
22) Item_extend_address={Si:位置}    # 记录在动态 API 的位置
23)                 If (++non_sup) > Supnon
24)                     Break    # 候选可扩展项的不支持度大于 Supnon,结束循环
25) extend_address={item : 可扩展项的位置 }
26)      return extend_address
```

本文提出了最大不支持度概念(在挖掘算法第 23～24 行)，当不支持度大于最大不支持度时，提前结束本轮循环，在最小支持度较大，最大不支持度较小时，比如本文最小支持度为 94，最大不支持度为 6，能显著减少计算量.

第二部分对搜索过程中产生的最长频繁 API 序列进行过滤，由于本挖掘算法采用深度优先策略，挖掘过程中会产生大量较短的子序列. 本文从 Redis 数据库的队尾依次取出最长频繁序列进行过滤，过滤队列为 f'，过滤策略如下.

(1) 超集检测：对一个最长频繁序列 s，通过查找其在 f' 中是否存在一个最长频繁 API 序列 S_a，使得 $s \subseteq S_a$，如果存在，则 s 不是最长频繁 API 序列. 如果不存在，s 就是目前为止已经挖掘到的最长频繁 API 序列，并把它添加到 f' 中.

(2) 子集检测：如果序列 s 通过了超集检测，即它为目前已经挖掘到的最长频繁 API 序列，那么需要删除 f' 中 s 所有的子集，设 s' 是 f' 中任意一个 API 序列，如果 $s' \subseteq s$，则其是 s 的子集，从 f' 中移除 s'.

2.4　恶意代码表示与检测

本文基于挖掘的最长频繁序列集合构造词袋模型，把样本文件的动态 API 序列转化为特征向量，使用随机森林进行分类器训练和恶意代码检测.

2.4.1　*最长频繁序列集合表征样本文件*　词袋(Bag of Words，BoW)模型最早出现在自然语言处理领域，广泛用于信息检索和文本分类，用于把文本内容向量化. 词袋模型也广泛应用于恶意代码的分类与聚类问题[15-17].

样本的动态 API 序列可以看作具有时间序列的一维文本，单个最长恶意频繁序列作为能表征文本类别的一个特征，因此本文用最长频繁 API 序列集合构造恶意行为的词袋模型，基于此模型将恶意代码样本和正常样本的动态 API 序列转化为向量.

对于恶意代码来说，恶意行为往往多次出现，因此最长频繁序列可能在样本文件的动态 API 序列中多次出现. 统计表明，大多数最长频繁序列在动态 API 序列中出现次数不超过 5 次. 因此，为了避免奇异性向量出现导致训练时间长，当该 API 序列在动态 API 序列出现的次数超过 5 次时，计为 5 次.

2.4.2　*分类器构建与恶意代码检测*　本文使用随机森林算法[18]建立分类器模型并进行恶意代码检测. 随机森林属于集成学习中的 Bagging 算法，通过组合多个弱分类器(决策树)，最终结果通过投票或取均值，使得模型有较高的精确度和抗过拟合性能.

检测阶段首先把训练样本的向量集进行归一化处理；然后进行标注，导入数据开始训练分类器；最后把处理好的测试集输入到已训练好的模型中进行检测.

3　实验结果和分析

本节通过一系列实验对本文提出的恶意代码

检测方法进行评估,并将其与已有的基于 API 调用序列的恶意代码检测方法进行比较. 本文实验采用多进程,利用 Redis 数据库的队列功能实现分布式挖掘,挖掘部分在 2 台 6 核 Intel(R)i5-8600k 机器上进行,去重部分在 6 核 Intel(R) Xren(R) CPU E3-1230 v3 上运行.

3.1 实验数据集及挖掘结果

论文实验中的实验样本来源于阿里云安全算法大赛使用的数据集,包括 7 个典型类别的恶意样本和正常样本. 实验的 7 类恶意代码分别是勒索病毒、挖矿程序、DDoS 木马、蠕虫病毒、感染性病毒、后门程序和特洛伊木马. 此数据集包含 301 个 API 函数,为便于处理,本文对每个 API 函数用编号替代.

由于此数据集包含逃避性样本,还有一些样本因其他原因未在沙箱环境成功运行[6,9],对最长频繁序列挖掘及后面的恶意代码检测造成干扰,因此剔除这部分样本,该数据集的蠕虫病毒样本数量较少,故额外选取 180 个蠕虫样本,他们来源于开源病毒库 Virushare,使用 Virustotal 检测和本地搭建 Avclass[19] 环境进行标注. 本文除去挖掘样本后,80%的样本用于模型训练,20%的样本用于测试. 筛选后的数据集样本空间如表 3 样本集所示.

表 3 样本集

Tab. 3 Sample set

恶意代码类别	挖掘样本数量	合计	支持度	最长频繁序列数量
勒索病毒	100	429	98	559
挖矿木马	100	1 031	91	354
DDoS 程序	100	739	94	161
蠕虫病毒	100	263	88	319
感染病毒	100	3 311	80	139
后门程序	100	419	84	1 483
特洛伊木马	100	770	82	3052
正常样本	700	6 962	0	0

为保证挖掘的最长频繁序列具有普适性,样本应该随机选取,且达到一定量. 所有类别采用 100 个样本进行挖掘. 由定义可知,支持度越小,时间和空间复杂度越大,挖掘到的最长频繁序列越多. 为保证挖掘算法的时间可实现性,而挖掘到尽可能多的最长频繁序列,支持度应合理选择. 本文各类别挖掘的支持度以及挖掘结果如表 3 所示.

3.2 实验结果分析

在分类问题中,评价指标有准确率(Accuracy,ACC)、精确度和召回率曲线(Precision Recall Curve,PRC)、特征曲线(Receiver Operating Curve,ROC)和曲线区域面积(Area Under Curve,AUC). 曲线面积 AUC 在测试集样本分布变化时仍然能够保持稳定. AUC 越大,表明分类器效果越好,因此测试结果的采用的主要指标有:ACC 和 AUC. 这些指标涉及机器学习中的四个检验指标:真阳性(TP)、真阴性(TN)、假阳性(FP)、假阴性(FN),如表 4 检验指标的含义.

表 4 检验指标的含义

Tab. 4 Meaning of the inspection index

性能指标	TP	TN	FP	FN
样本类型	恶意	正常	正常	恶意
检测结果类型	恶意	正常	恶意	正常

ROC 是一个二维坐标轴中的曲线,横坐标是假阳性率(False Positive Rate,FPR),纵坐标是真阳性率(True Positive Rate,FPR),由表 4 得到 TPR 和 FPR 定义. 由一个确定的分类器和测试集可得到一组 FPR 和 TPR,设置不同的 TPR 可得到一系列的 FPR 和 TPR 构成 ROC 曲线,AUC 是 ROC 曲线好坏的有效度量方式,TPR、FPR 和 ACC 的定义如下.

$$TPR=\frac{TP}{FN+TP}\times 100\% \tag{3}$$

$$FPR=\frac{TP}{TN+FP}\times 100\% \tag{4}$$

$$ACC=\frac{TP+TN}{TN+FN+TP+FP}\times 100\% \tag{5}$$

选取不同的信息增益的序列,可得到一系列的 AUC 和 ACC 值,如图 2 不同最长频繁数量的准确率和 AUC 值所示.

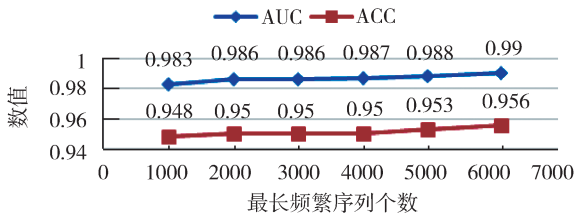

图 2 不同最长频繁数量的准确率和 AUC 值

Fig. 2 The accuracy and AUC value of different sequence number

从图 2 可以看出，选取前 6131 个信息增益较大的最长 API 序列，AUC 曲线面积最大为 0.99，分类器的准确率为 95.6%. 而芦效峰等人[13]提出的方法使用信息熵筛选 API 序列特征，组合随机森林和 LSTM 网络进行恶意代码检测. 如表 5 为本文方法芦效峰等人的组合模型的对比.

表 5　本文方法和组合框架方法的检测结果

Tab. 5　The method comparison between this paper and combination frame

本文 Method		组合模型	
准确率/%	AUC	准确率%	AUC
95.6	0.99	93.5	0.971

表 5 的实验结果表明，本文提出的恶意代码检测方法相对于芦效峰等人的组合模型对恶意代码检测的效果更佳. 这是因为芦效峰等人把所有样本作为一个恶意样本集进行 API 序列的筛选，没有考虑不同种类的恶意代码的功能差异性导致的 API 序列的差异性.

文献[8,13]指出序列长度越长，对恶意样本和正常样本有更强的区分能力，而文献[8]提出的 MACSPMD 方法挖掘的序列长度大于等于 5 的才 174 个，大于等于 7 的序列才 8 个，文献[14]采用 4-Gram 方法进行 API 序列的分割，因此检测准确率存在局限性. 本文提出的方法挖掘的 6 131 个序列长度分布如图 3 所示.

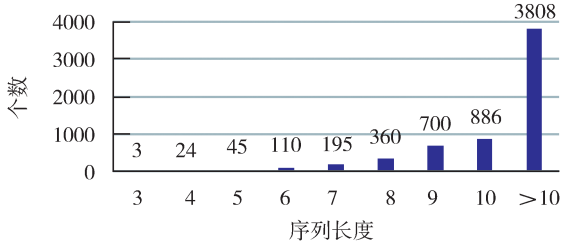

图 3　最长频繁序列不同长度的个数分布图

Fig. 3　The number of longest frequent API sequence distribution of the length

如图所示最长频繁序列的长度大部分大于 6，因此，本文挖掘的最长频繁序列能更好的表示恶意代码的某种潜在恶意行为，并且相对于其他特征形式(API 调用集和 N-Gram 模式)，本文挖掘的最长频繁序列作为特征有更强的解释性. 另外本文不同类别的恶意样本分布不均衡，仍能取得较好的检测效果，说明本文提取的特征更有效，因此对恶意样本和正常样本有更好的区分能力.

4　结　论

本文基于恶意代码的动态 API 调用序列，采用最长频繁序列挖掘算法，提出了一种新的恶意代码检测方法. 本文提出的恶意代码检测方法首先获取 PE 文件在沙箱中运行产生的 API 调用序列；然后，选取区别恶意样本和正常样本的关键 API 保留其调用顺序，并使用本文提出的分类别的最长频繁序列挖掘算法挖掘出能代表这一类恶意代码的最长 API 调用序列；使用挖掘出的最长频繁 API 序列集合构造词袋模型；用特征向量表征样本文件；最后采用随机森林构造分类器用于恶意代码检测. 实验结果表明，本文提出的恶意代码检测模型能有效检测恶意代码.

本文提出的挖掘算法在挖掘过程中要根据候选项多次迭代，产生了大量冗余的子序列，这也是当前序列挖掘算法的一大难点，希望在以后的工作中能够优化数据结构和算法，避免大量冗余的序列产生，加快挖掘速度，节约存储空间. 本文考虑到子序列的数量级较大，未将最长频繁 API 序列还原成 API 子序列，在恶意行为的表征方面缺乏细粒度的表示，下一步提出优化策略完全还原所有的 API 子序列，并筛选增益较大的 API 子序列来表征恶意代码的恶意行为. 另外本文选取的特征较为单一，在加入 API 总数、API 个数、单个 API 所占比例统计特征，API 语义，API 参数及返回值等特征，结合自然语言处理和深度学习算法应该能进一步提高准确率.

参考文献：

[1]　Damodaran A, Troia F D, Visaggio C A, *et al*. A comparison of static, dynamic, and hybrid analysis for malware detection [J]. J Comput Virology Hacking Tech, 2017, 13: 1.

[2]　王怀军，房鼎益，李光辉，等. 基于变形的二进制代码混淆技术研究[J]. 四川大学学报：工程科学版，2014，46：14.

[3]　Ahmadi M, Sami A, Rahimi H, *et al*. Malware detection by behavioural sequential patterns [J]. Comput Fraud Secur, 2013, 2013: 11.

[4]　Cho I K, Kim T G, Shim Y J, *et al*. Malware similarity analysis using API sequence alignments [J]. J Internet Serv Inf Secur, 2014, 4: 103.

[5]　Ki Y, Kim E, Kim H K. A novel approach to de-

tect malware based on API call sequence analysis [J]. Int J Distrib Sens N, 2015, 2015: 1.

[6] Pirscoveanu R S, Hansen S S, Larsen T M T, *et al*. Analysis of malware behavior: Type classification using machine learning [C]// Proceedings of the 2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA). Londun: IEEE, 2015.

[7] Kolosnjaji B, Zarras A, Webster G, *et al*. Deep learning for classification of malware system call sequences [C]//Proceedings of the Australasian Joint Conference on Artificial Intelligence. Cham: Springer, 2016.

[8] 荣俸萍, 方勇, 左政, 等. MACSPMD: 基于恶意 API 调用序列模式挖掘的恶意代码检测[J]. 计算机科学, 2018 ,2018: 131.

[9] Nikolopoulos S D, Polenakis I. A graph-based model for malware detection and classification using system-call groups [J]. J Comput Virology Hacking Tech, 2017, 13: 29.

[10] Ding Y, Xia X, Chen S, *et al*. A malware detection method based on family behavior graph [J]. Comput Secur, 2018, 73: 73.

[11] Uppal D, Sinha R, Mehra V, *et al*. Malware detection and classification based on extraction of API sequences [C]// Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI). New Delhi: IEEE, 2014.

[12] 李盟, 贾晓启, 王蕊, 等. 一种恶意代码特征选取和建模方法[J]. 计算机应用与软件, 2015, 32: 266.

[13] 芦效峰, 蒋方朔, 周箫, 等. 基于 API 序列特征和统计特征组合的恶意样本检测框架[J]. 清华大学学报: 自然科学版, 2018, 58: 500.

[14] Hsu M C. FreeSpan: frequent pattern-projected sequential pattern mining [C]//Proceedings of the Acm Sigkdd International Conference on Knowledge Discovery & Data Mining. Boston: ACM, 2000.

[15] Salehi Z, Sami A, Ghiasi M. MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values [J]. Eng Appl Artif Intell, 2017, 59: 93.

[16] 冯亚玲. 基于系统调用的恶意软件检测技术研究[J]. 信息安全研究, 2016, 2: 367.

[17] Széles G J, Coleşa A. Malware clustering based on called API during runtime[C]//Proceedings of the International Workshop on Information and Operational Technology Security Systems. Cham: Springer, 2018.

[18] Ho T K. Random decision forests[C]// Proceedings of the 3rd International Conference on Document Analysis and Recognition. Montreal, Quebec: IEEE Computer Society, 1995.

[19] Sebastián M, Rivera R, Kotzias P, *et al*. Avclass: A tool for massive malware labeling [C]// Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses. Cham: Springer, 2016: 230.

引用本文格式:

中 文: 黄琨茗, 张磊, 赵奎, 等. 基于最长频繁序列挖掘的恶意代码检测[J]. 四川大学学报: 自然科学版, 2020, 57: 681.

英 文: Huang K M, Zhang L, Zhao K, *et al*. Malware detection based on longest frequent API sequence [J]. J Sichuan Univ: Nat Sci Ed, 2020, 57: 681.