软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(11):3436−3447 [doi: 10.13328/j.cnki.jos.005863]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

子图相似性的恶意程序检测方法*

汪 洁, 王长青

(中南大学 计算机学院,湖南 长沙 410083)

通讯作者: 汪洁, E-mail: jwang@csu.edu.cn

摘 要: 动态行为分析是一种常见的恶意程序分析方法,常用图来表示恶意程序系统调用或资源依赖等,通过图挖掘算法找出已知恶意程序样本中公共的恶意特征子图,并通过这些特征子图对恶意程序进行检测.然而这些方法往往依赖于图匹配算法,且图匹配不可避免计算慢,同时,算法中还忽视了子图之间的关系,而考虑子图间的关系有助于提高模型检测效果.为了解决这两个问题,提出了一种基于子图相似性恶意程序检测方法,即 DMBSS.该方法使用数据流图来表示恶意程序运行时的系统行为或事件,再从数据流图中提取出恶意行为特征子图,并使用"逆拓扑标识"算法将特征子图表示成字符串,字符串蕴含了子图的结构信息,使用字符串替代图的匹配.然后,通过神经网络来计算子图间的相似性即将子图结构表示成高维向量,使得相似子图在向量空间的距离也较近.最后,使用子图向量构建恶意程序的相似性函数,并在此基础上,结合 SVM 分类器对恶意程序进行检测.实验结果显示,与其他方法相比,DMBSS 在检测恶意程序时速度较快,且准确率较高.

关键词: 恶意程序检测;神经网络;子图分布式表示;图相似函数

中图法分类号: TP311

中文引用格式: 汪洁,王长青.子图相似性的恶意程序检测方法.软件学报,2020,31(11):3436−3447. http://www.jos.org.cn/1000-9825/5863.htm

英文引用格式: Wang J, Wang CQ. Malware detection method based on subgraph similarity. Ruan Jian Xue Bao/Journal of Software, 2020,31(11):3436−3447 (in Chinese). http://www.jos.org.cn/1000-9825/5863.htm

Malware Detection Method Based on Subgraph Similarity

WANG Jie, WANG Chang-Qing

(School of Computer Science and Engineering, Central South University, Changsha 410083, China)

Abstract: Dynamic behavior analysis is a common method of malware detection. It uses graphs to represent malware's system calls or resource dependencies. It uses graph mining algorithms to find common malicious feature subgraphs in known malware samples, and detect unknown programs through these features. However, these methods often rely on the graph matching algorithm, and the inevitable calculation of the graph matching is slow, and the relationship between the subgraphs is also neglected in the algorithm. It can improve the detection accuracy of the model if the subgraphs' relationship is considered. In order to solve these two problems, a sub-graph similarity malware detection method called DMBSS is proposed. It uses the data flow graph to represent the system behavior or event of the running malicious program, and then extracts the malicious behavior feature subgraph from the data flow graph, and uses "inverse topology identification" algorithm to represent the feature subgraph as a string, and the string implied the structural information of the subgraph, using a string instead of the matching of the graph. The neural network is then used to calculate the similarity between the subgraphs and to represent the subgraph structure as a high dimensional vector, so that the similar subgraphs' distance is also shorter in the vector space. Finally, the subgraph vector is used to construct the similarity function of the malicious program, and based on this, the SVM classifier is used to detect the malicious program. The experimental results show that compared with other methods, DMBSS is faster in detecting malicious programs and has higher accuracy.

* 基金项目: 国家自然科学基金(61202495)
Foundation item: National Natural Science Foundation of China (61202495)
收稿时间: 2018-12-10; 修改时间: 2019-01-17, 2019-03-23; 采用时间: 2019-04-22

Key words: malware detection; neural network; subgraph distributed representation; graph similar function

当今,恶意程序仍是信息安全的最大威胁之一,每年政府和企业因恶意攻击造成的损失就高达数百万元,甚至可以说,恶意程序已成为一种可以盈利的商业模式[1,2].高额的利润聚集着大量的非法人员,使得恶意程序愈发复杂,与之对应各种分析技术也相继出现,目前,恶意程序的分析方法可以分为静态特征分析方法和动态行为分析.

- 静态特征分析方法主要是利用恶意程序二进制文件进行分析和检测,通过反编译恶意程序,提取指令序列或系统调用等信息,并以此为特征建立恶意程序分析模型.然而,静态分析方法容易受到混淆技术的干扰[3,4],造成检测精度的下降.
- 动态行为分析方法弥补了静态特征分析方法的不足.动态行为分析方法是通过捕获恶意程序运行时的行为特征,主流方式是使用系统调用[5,6]或资源依赖[7]来表示行为特征,并以此构建行为分析图,之后再使用图挖掘算法提取出恶意样本中的恶意特征或子图,这些恶意特征能够有效地区分出恶意和正常程序.如:Christodorescu 等人[8]率先提出了挖掘存在于恶意程序而非正常程序图的特有行为子图;Park 等人[9]提出了基于 HotPath 如最大公共子图来捕获恶意程序行为特征.

移动平台上同样存在着静态[4,10]和动态分析[7,11].Fan 等人[12]通过构建频繁子图来表示同一家族的常见行为,并将其用于安卓恶意程序家族分类.上述动态分析方法在挖掘特征子图过程中大多以频率[12,13]来确定子图的效用,这意味着特征子图的效用取决于该子图出现在恶意程序样本中的次数.然而这些方法过多地强调了特征子图出现的频率,而忽视了子图本身的属性.在这个问题上,Wuchner 等人[14]提出了基于压缩的恶意程序行为检测,子图的效用不仅考虑了子图出现在恶意样本中频率,同时还考虑子图本身结构复杂性.作者在文中使用 Scoring 函数 MDC(maximum data compression)来计算子图的效用,即:当一个特征子图上的数据流在恶意程序总数据流中所占的比重大,而在正常程序的总数据流中所占的比重小时,则认为该特征子图在检测恶意程序时越有用.

然而,基于图的动态行为分析方法存在两个问题:(1) 这些方法大都依赖于图匹配,图匹配是一个 NP 完全问题,不可避免地计算缓慢;(2) 方法中将挖掘出来的特征子图作为检测依据,却忽视了子图间的关系.子图作为分类属性并不能等同于一般的属性,这是因为子图间存在着联系,例如,子图添加一个节点和一条边就可以变成另一个子图.

针对这两个问题,本文提出了基于子图相似性的恶意程序检测方法,即 DMBSS.我们采用“逆拓扑标识”方法将特征子图表示成字符串,子图的结构信息蕴含子符串内,使用字符串代替图的匹配.同时也将子图间的相似性考虑进模型,这有利于提高模型的检测效果.子图间的相似性借鉴于神经语言模型,将图视作一种特殊的语言,类似于词组成句子而子图组成图,然后利用神经网络将子图结构映射成高维空间的向量,并使得相似子图所映射的向量在其向量空间距离也相近.这种映射主要是利用子图的“上下文”信息构建子图 Skip-Gram 模型,同时,本文使用子图向量构建图相似函数,结合 SVM 分类器可以恶意程序进行检测.

本文的贡献如下.

- 本文使用“逆拓扑标识”将特征子图表示成字符串,子图的结构信息蕴含在字符串内,字符串间的匹配比图更迅速.
- 本文基于神经网络实现了子图相似性的学习,使得相似子图在向量空间距离也相近.
- 实验结果表明,我们的方法精度与效率优于 MDC 方法.

本文第 1 节描述恶意程序检测方法的框架.第 2 节介绍恶意程序特征提取过程.第 3 节介绍特征子图嵌入即子图的向量化.第 4 节介绍恶意程序检测模型构建.第 5 节展示实验结果.第 6 节陈述结论.

1 方法框架

本节主要介绍基于子图相似性的恶意程序检测的框架.如图 1 所示,整个框架分成 3 部分,即特征子图库构

建及其向量表示、训练模型的构建过程、未知程序的检测过程.首先，将训练集中的恶意程序表示成数据流图,并从中进行特征子图提取来构建特征子图库;接下来,使用神经网络对子图进行相似性学习即子图向量化,其中,向量间的距离表示二者相似性.整个过程如图 1(a)所示,该过程是检测模型构建的前提.图 1(b)表示训练模型的构建过程,首先从训练样本(包含正常和恶意程序)中捕获运行时的行为特征并构建数据流图,之后基于特征子图库将数据流图映射成图向量,具体细节见下文.图向量结合子图向量来构建图相似函数,这同时兼顾了图的结构信息以及子图间的相似性.图相似函数可以计算任意两个图间的相似度,基于此,我们将样本中两两间进行图相似计算来构建样本的相似核矩阵,结合 SVM 算法训练出一个分类模型.图 1(c)表示未知程序的检测过程,即给定一个未知的程序,需判断其类别恶意还是正常.先将未知程序映射成图向量,再通过计算待测样本与训练样本的相似度(采用图相似函数计算)来构建相似向量,之后将相似向量放入 SVM 分类器,SVM 分类器将基于已知恶意和正常的信息进行类别判断.

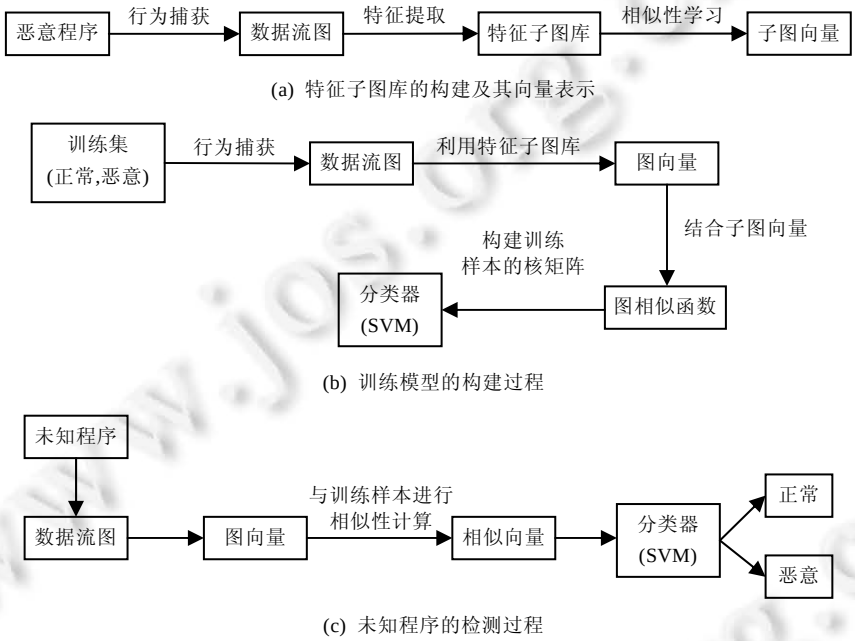


Fig.1 Framework of malware detection based on subgraph similarity

图 1 基于子图相似性的恶意程序检测框架

2 恶意程序特征提取

2.1 数据流图

在恶意程序行为检测方法中,常使用图来表示系统行为或资源依赖.现已证明,直接利用原始系统调用的方法检测对行为敏感[15],因而需使用更抽象的模型来构建系统行为图.本文中使用恶意程序的数据流图来表示系统行为图.

数据流图表示程序在一段运行时期内系统实体间的数据流,是通过捕获这段时期内系统实体间的行为或事件生成的.其中,系统实体包括进程 PROCESS,注册表 REGISTRY,文件 File 和网络地址 URL;而系统事件包括进程 ReadProcessMemory 和 WriteProcessMemory,文件 ReadFile 和 WriteFile,注册表 RegQueryValue 和 RegSetValue,网络 Send 和 Recv.虽然构建数据流图的实体和事件少,但 Wunchner 等人[16]证实了利用这些属性也是可以构建复杂的系统行为模型;同时,设计合适的方法可以进行高精度检测.我们使用图 2 来说明.图 2 表示系统实体进程 P 从文件 F 中读取数据流,使用 $e(src,dest,s,t)$ 表示这一事件,其中,src 表示数据流的发起方文件 F,$dest$

表示数据流的接受方进程 P,s 表示事件数据流大小即信息的字节数,t 表示事件发生的时间.为了简化模型,我们对同一对实体间的事件进行合并而不是创建各自的事件.在数据流图中具体表现,节点 A 流向另一个节点 B 的事件存在多起,即 $e_1(A,B,s_1,t_1)$与 $e_2(A,B,s_2,t_2)$,则合并为事件 $e(A,B,s_1+s_2,\min(t_1,t_2))$.这种简化无疑是精度向效率的妥协,我们认为这种妥协是可行.

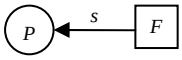


Fig.2 Data flow diagram

图 2 数据流示意图

本文使用 $G(N,E,\lambda)$来表示数据流图,其中,N 表示系统实体集合,E 表示系统事件,函数λ表示系统事件的更新.使用 $update(G,e)$来表示数据流图的生成过程,每次捕获系统实体间的事件就对数据流图进行更新.若该事件已存在,则合并流图 G 中该事件附加的信息;若不存在,则保存该事件信息到流图中.其表示如式(1)所示.

$$update(G,(src,dest,s,t)=\begin{cases}\begin{pmatrix}N\\E\\\lambda\begin{bmatrix}(e,size)\leftarrow(e,size)+s\\(e,time)\leftarrow(\min(\lambda(e,time),t)\end{bmatrix}\end{pmatrix}, & \text{if } e\in E\\ \begin{pmatrix}N\cup\{src,dest\}\\E\cup\{e\}\\\lambda\begin{bmatrix}(e,size)\leftarrow s\\(e,time)\leftarrow t\end{bmatrix}\end{pmatrix}, & \text{otherwise}\end{cases} \tag{1}$$

2.2 特征子图提取

上一节介绍了从恶意程序捕获系统实体间数据流生成数据流图,这一节我们将详细介绍从数据流图中提取特征子图的过程.我们使用 TreeWalk 算法提取数据流图的特征子图,该算法如算法 1 所示.

算法 1. *TreeWalk*(v,G,d).

输入:v 表示子图根节点且 $v\in G$,G 表示数据流图,d 表示提取深度.

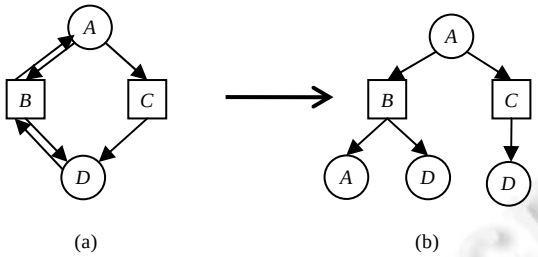
输出: sg_v^d 表示根节点 v、深度为 d 的子图.

1. sg_v^d 初始只有根节点 v.
2. 若 d 大于 0,则将图 G 中节点 v 的后继节点 N_v 作为树根节点的孩子节点,d 的大小减 1.
3. 判断 d 是否大于 0:若大于 0,则将树中叶子节点在图 G 中的邻接节点作为该叶子节点的孩子节点,图中已处理的节点对应的叶子节点除外.
4. 重复步骤 3,直至 d 的大小为 0,返回子图.

算法 1 表示从图 G 中提取根节点 v、深度为 d 的特征子图.我们使用图 3 来进行具体说明.图 3(a)是某个恶意程序的数据流图 G,图 3(b)是以 A 为根节点、深度为 2 提取的子图.其具体过程如下:初始时,子图只有根节点 A 且 d 为 2;然后,将图 G 中节点 A 的后继节点 B 和 C 作为子图根节点的孩子节点且 d 的大小减 1;之后,将树中叶子节点 B 和 C 在图 G 的后继节点作为该叶子节点的孩子节点,如节点 B 的孩子节点是 A 和 D,C 的孩子节点是 D,此时,d 为 0,则返回该特征子图.若初始 d 为 3,则需对图 3(b)中叶子节点 D 进行后继节点处理,而叶子节点 A 不处理.这是因为节点 A 的后继节点信息已存在子图中,这样可以减小子图的结构复杂性.

使用 TreeWalk 算法可以从图中以任意节点提取深度为 d 的特征子图,这些特征子图包含了图的恶意行为.然而,恶意程序间的差异性使得恶意行为也不同,仅通过深度为 d 的特征子图是不足以涵盖所有的恶意行为.因而,我们通过给定阈值 D,即最大提取深度来尽可能提取出恶意程序的恶意行为,即从图中以任意节点提取深度 $d\in\{0,1,\ldots,D\}$的特征子图,来尽可能地挖掘可能的恶意行为.

对于图 G,其所有的子图集表示为 $sg(G)=\bigcup TreeWalk(v_i,G,d),\forall v_i\in G,d\in\{0,1,...,D\}$,对于恶意程序数据流图集$\{G_1,G_2,\ldots,G_n\}$中提取每个图的子图集 $sg(G_i)$构造恶意程序的子图语料库 $sg_{cors}=\bigcup sg(G_i),i=1,2,...,n$,子图语料库包含了训练样本中恶意程序所有可能的恶意行为特征.



(a) (b)

Fig.3 Schematic diagram of TreeWalk algorithm

图 3 TreeWalk 算法示意图

2.3 特征子图表示

在子图语料库构建过程中,面临着相同子图的合并即子图同构问题.在这个问题的处理上,我们借鉴了非常有名的 WL 重标识方法[17,18],将子图中节点的类型作为标识,采取“逆拓扑”的方式将子图映射成一个字符串,这样可以快速进行字符串比较.

其具体过程如下:初始时,子图中节点标识为节点类型{PROCESS,FILE,REGISTRY,URL}简写成{P,F,R,U};之后,将每个节点的后继节点标识排序后拼接在该节点标识后,再删除子图中出度为 0 的节点.重复上述过程,直至子图中只有一个节点.整个过程类似于拓扑的逆过程,我们称其为逆拓扑标识.我们使用图 4 来进行演示,其中,圆形表示进程,正方形表示文件,菱形表示注册表,六边形表示网络地址.初始时,节点 A~E 的标识分别为 P,R,F,P,U,而节点 A 的后继节点是节点 B 和 C,将节点 B 和 C 的标识排序后拼接在节点 A 的标识之后;对剩余节点同样处理,之后删除出度为 0 的节点 B 和 E;然后重复这个过程,直至只有一个节点.其具体过程见表 1,这里使用“#”进行分隔主要是为了便于查看,同时,每次拼接时,“#”的个数为迭代的次数.

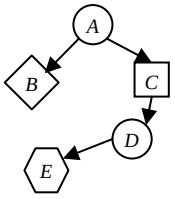


Fig.4 Diagram of data flow subgraph

图 4 子图的数据流图

Table 1 Subgraph inverse topology identification

表 1 子图逆拓扑标识

	节点	节点标识	节点出度
初始时状态	*A*	P	2
	B	R	0
	C	F	1
	D	P	1
	E	U	0
第 1 次迭代	*A*	P#F,R	1
	C	F#P	1
	D	P#U	0
第 2 次迭代	*A*	P#F,R##F#P	1
	C	F#P##P#U	0
第 3 次迭代	*A*	P#F,R##F#P###F#P##P#U	0

逆拓扑标识可以将一个子图映射成一个字符串,字符串间的比较相比图的匹配更简单、更迅速.同时,字符串中蕴含了子图的结构信息,子图结构不同其字符串表示也不同,并且本文中子图间的相似性学习并不是依据子图本身的结构而是子图的“上下文”信息.但必须注意的是,

1) 子图中存在环时方法将失效.子图语料库中的子图都是基于 TreeWalk 算法,而 TreeWalk 算法可以确保提取的子图中不存在环.
2) 子图字符串长度与子图的深度有关,当子图深度比较大时,字符串的长度会变得非常冗长.

然而,子图的最大深度取决于最大提取深度 D,从后续的实验中可以看出,最大深度 D 最佳取值为 5,因而子图表示长度仍在合理范围内.

2.4 特征子图库构建

子图语料库是以训练样本中恶意程序图中每个节点作为根节点,提取深度为 0~D 的子图,这会造成简单的子图频繁出现,例如单个节点、两个节点如进程读取注册表等,这些子图无论在恶意还是非恶意程序都是普遍存在的.然而这些简单的高频子图就类似于文本中高频出现的 a,the 等词,借鉴最新的神经语言模型处理高频词的方法[19],我们采取了简单的子采样方法,在子图语料库 sg_{cors} 中,每个子图 sg_i 具有以下概率被放弃,其计算公式为

$$P(sg_i)=1-\sqrt{\frac{t}{f(sg_i)}} \tag{2}$$

其中,$f(sg_i)$表示子图 sg_i 的频率;t 表示选择门槛,通常在 10^{-3} 左右.我们选择这个子采样公式是因为它能够对频率大于 t 的子图进行子采样,即频率愈大其被剔除的概率也愈大.虽然这个子采样公式是启发式的,不仅加速了学习速度而且提高子图的准确率,同时子图语料库 sg_{cors} 中低频的子图可以直接使用剔除的方法,即将子图语料库频率小于某一定值的子图剔除.实验中,我们取值为 3.子图语料库中对高频子图进行子采样,对低频子图进行剔除,剩余的子图就是普遍存在于训练样本恶意程序中,我们将子图语料库中剩余的子图称为特征子图库.

3 子图嵌入

本节的目的是将特征子图库中的子图向量化,这利用了神经语言模型 SkipGram[19−21]的思想(如图 5 所示).神经语言模型可以解决词相似性问题,即可以从文本中找出语境相似的词,如“苏维埃”与“俄罗斯”“语文”与“数学”等.其核心思想是,将每个词表示成多维向量,并且每次更新使得一个词与其上下文的词(相邻的词)相似而与其他词不相似.通过对文本不断地学习,会使得上下文相似的词相似,具体表现词向量的空间距离相近.我们将图视作为一种特殊的语言,类似于不同的词组成句子一样,不同的子图构成图,提出了 SubSkipGram 模型去学习子图间的相似性.

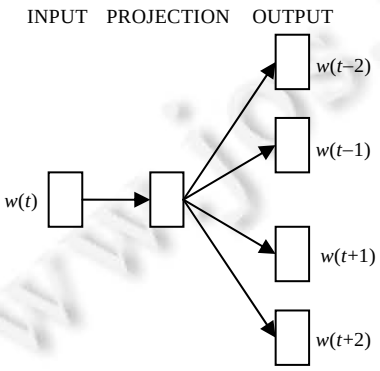


Fig.5 SkipGram model

图 5 SkipGram 模型

然而,图不能完全等同于文本,子图间不具有文本中词之间的线性关系.SkipGram 模型中将出现在目标词固定窗口内的词作为其上下文,这里,目标词与上下文中的词就具有线性关系.然而子图间并没有线性的上下文关系,所有的子图是从图中以某一节点作为根节点提取的.考虑到以根节点 v 的子图 sg_i 与以根节点 N_v 的子图 sg_j 存在一些相同特征,其中,N_v 表示节点 v 的后继节点,因此,子图 sg_i 相比其他子图与 sg_j 是更有关系的.基于这一点,我们将以后继节点 N_v 为根节点的子图作为子图 sg_i 的上下文 $context(sg_i)$.子图的根节点是唯一的,将根节点邻接节点子图作为其上下文,就好比文本中将邻近词作为目标词的上下文.关于子图上下文更多的信息,我们将在第 3.2 节讨论.

3.1 SubSkipGram模型

SubSkipGram 模型是计算在目标子图条件下,上下文子图概率的最大化.给定目标子图 sg_i 以及目标子图上下文 $context(sg_i)$,SubSkipGram 模型是求以下对数似然最大化.

$$\sum_{i=1}^{T}\log P(context(sg_i)\mid sg_i) \tag{3}$$

其中,概率 $P(context(sg_i)|sg_i)$是通过以下公式计算.

$$\prod_{sg_j\in context(sg_i)} P(sg_j\mid sg_i) \tag{4}$$

此外,概率 $P(sg_j|sg_i)$定义如下.

$$\frac{\exp(v_{sg_i}^{T}\cdot v'_{sg_j})}{\sum_{k=1}^{C}\exp(v_{sg_i}^{T}\cdot v'_{sg_k})} \tag{5}$$

其中,v_{sg_k} 和 v'_{sg_k} 是子图 sg_k 的输入和输出向量,C 表示子图库中子图的数量.

SubSkipGram 模型是神经语言 SkipGram 模型的延申.为了更好地理解上述式子,我们使用文本来解释.SkipGram 模型通过给定一个目标词来预测其周围词即给定输入向量来更新其上下文向量,而更新的依据来自公式(3),通过条件概率的最大化来反向更新其上下文向量.而公式(5)是用来计算上下文中的词出现的条件概率,即:给定目标词,某一词出现在该目标词的上下文的概率.其中,概率是通过词向量来计算的.

3.2 子图分布式学习

本节中将介绍目标子图上下文的构造过程以及其更新方式,具体如下.

算法 2. $SubSkipGram(\Phi,sg_v^d,G,D)$.

输入:Φ表示子图库中子图的向量矩阵,初始为 0; sg_v^d 表示根节点 v、深度为 d 的目标子图;D 表示最大提取深度.

1. $context(sg_i)=\varnothing$
2. **for each** $v'\in N_v$
3. **for each** $\theta\in\{d-1,d,d+1\}$ and $0\leqslant\theta\leqslant D$
4. $context(sg_v^d)=\bigcup TreeWalk(v',G,\theta)$
5. **for each** $sg_{cont}\in context(sg_v^d)$
6. $J(\Phi)=-\log P(sg_{cont}\mid\Phi(sg_v^d))$
7. $\Phi=\Phi-\alpha\dfrac{\partial J}{\partial\Phi}$

算法 2 中,步骤 1~步骤 4 表示子图 sg_v^d 上下文 $context(sg_v^d)$ 的构建过程:从根节点 v 的后继节点 N_v 提取深度为$\{d-1,d+1\}$的子图作为上下文,正如前面所说,子图多添加一个节点以及一条边可变成另一个子图,深度为 d 的目标子图 sg_v^d 不应只考虑邻接深度为 d 的子图作为其上下文,而深度$\{d-1,d+1\}$的子图也应考虑在内.步骤 5~步骤 7 表示根据目标子图来更新其上下文子图,具体是通过梯度下降来实现的,其中,α表示学习率.

然而,算法 2 只是对目标子图上下文的一次更新.而神经网络需要对所有的子图进行反复学习,即:需要对子图库中的子图每出现一次在训练集的数据流图时,就对该子图的上下文进行一次更新,通过不断地学习,会使得语境相似(上下文)的子图其向量表示距离也相近.并且在算法具体实现过程中,我们从数据流图提取子图的同时也保存了该子图的上下文信息,因而后续的过程只需要读取相关信息即可,这可以减小大量的时间开销.

4 恶意程序检测模型构建

在上一节介绍了子图间相似性的学习,这一节将详细介绍如何使用子图向量构建图相似性函数以及使用图相似函数来进行分类和检测.

4.1 图相似函数

本文使用深度图内核来计算图间的相似度,深度图内核[22]是一种非常流行并广泛应用于计算两个图间相似度,通过将一对图递归分成子结构,并定义子结构相似度函数来计算图间相似度.其表示如下.

$$S(G,G')=\Phi(G)^T\cdot M\cdot\Phi(G') \tag{6}$$

其中,$S(G,G')$是表示图 G 与 G'的相似度;$\Phi(G)$表示数据流图 G 的图向量;M 是个$|V\times V|$矩阵,本文中用来表示子图间的相似性矩阵,$|V|$表示子图库中子图的数量.

图相似函数考虑了子图间的相关性,这可以有效缓解对角线优势问题,即给定的图形容易仅与自身相似.关于公式(6)必须要说明的是,$\Phi(G)$表示流图 G 的图向量,且图向量的维度与特征子图库的大小相同.那么如何将一个图映射成一个向量?我们将子图库中的子图与图向量的维度一一对应,同时图向量中使用 1 和 0 来表示该图是否包含这个维度对应的特征子图.假设特征子图库只有$\{a,b,c,d,e\}$这 5 个子图,若图 G 只包含子图 b 和 d,那么其图向量为(0,1,0,1,0),同时,矩阵 M 设计成上述 5 个子图的相似矩阵.这里有一个难点,就是判断图 G 中包含的特征子图.图与子图匹配过程是非常复杂并且耗时的,而我们采取的方法是非常迅速,从图中提取特征子图时,采取逆拓扑标识的手段表示成字符串并保存到文件中,一个图对应一份文件,那么每份文件就存储了一个图所有的子图信息.之后,只需要使用子图库中的子图字符串与文件进行字符串匹配,就可以判断该图所包含的特征子图.

矩阵 M 表示子图库中子图间相似性矩阵,M_{ij} 表示子图 sg_i 与 sg_j 的相似度.使用上述 5 个子图的示例,M_{13} 则表示子图 b 和 d 的相似度.关于矩阵 M 的计算我们提供了两种计算方法.

(1) 在第 3 节中,我们将“上下文”相似的子图映射成的向量在其向量空间也相近.基于这一点,我们设计子图相似性计算方法如下.

$$s(sg_i,sg_j)=\frac{1}{d(v(sg_i),v(sg_j))+1} \tag{7}$$

其中,s 表示子图的相似度,$d(v(sg_i),v(sg_j))$表示子图 sg_i 与 sg_j 的空间距离,且 $v(sg_i)$表示子图 sg_i 的向量.

(2) 深度图内核,即公式(8),可以用来计算 G 与 G'的相似度,同样可以用来计算子图 sg_i 与 sg_j 的相似度,因为子图也是图.其表示如下.

$$s(sg_i,sg_j)=v(sg_i)^T\cdot m\cdot v(sg_j) \tag{8}$$

为了便于区分,使用了符号 m.这里,矩阵 m 仍是未知的.我们设计矩阵 m 是一个对角矩阵,其中,m_{ij} 是通过计算$\langle v(sg_i),v(sg_j)\rangle$且 $m_{ij}=0,i\neq j$.

上述两种方式均可以计算 $M_{ij}=(sg_i,sg_j)$,这样我们可以计算出子图库中任意子图间的相似性,即矩阵 M.

4.2 分类和检测

分类任务是将图分成两类或者多类.给定恶意程序和正常程序的图集$\{G_1,G_2,\ldots,G_n\}$以及类标签集合 $Y=\{y_1,y_2,\ldots,y_n\}$,其中,标签使用 0 和 1 分别表示正常和恶意程序,图分类任务是希望找到一种模型,能够将图 G_i 映射到标签 y_i.本文使用图相似函数构建核矩阵 K,K_{ij} 表示图 G_i 与 G_j 的相似度,然后将核矩阵 K 放入核方法如 SVM 进行分类训练:

$$K=\begin{bmatrix} S(G_1,G_1),...,S(G_1,G_n) \\ S(G_2,G_1),...,S(G_2,G_n) \\ \vdots \\ S(G_n,G_1),...,S(G_n,G_n) \end{bmatrix} \to \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{9}$$

检测任务就是一个未知的程序需检测出其类别恶意还是正常.给定数据流图 G,根据特征子图库映射成图向量,将该图向量与训练样本进行相似性度量获得相似向量 $v=(S(G,G_1),\ldots,S(G,G_n))$,将该向量放入分类器,而分类器在训练阶段建立好核矩阵 K 到标签集 Y 的映射,据此来确定向量 v 的类别.

5 实 验

本文实验有两个目的:首先,评估我们方法的准确性和有效性;其次,我们探讨了方法中的各个参数对结果的影响.

实验中使用 Cuckoo Sandbox[16]捕获样本原始的系统行为记录,并从中提取数据流图.恶意程序集来自开放的恶意程序数据库 VXHeaven(VXHeaven:http://vxheavens.com/),我们选择蠕虫、后门和特洛伊木马等 4 类作为实验对象,这里的类别是依据程序名划分的,而最终是检测恶意和正常程序,即使出现类别错误,但并不影响检测结果.正常程序是来自信息安全课程学生提交的可执行 exe 文件.由于相关数据的缺乏,我们将对正常的、可执行文件的收集任务交由本科生完成.提交程序种类繁多,完全是可以模拟正常情况电脑的运行状况.鉴于一些程序并不能正常运行在虚拟平台中,实验最后,我们随机选择了可以完全运行的恶意和正常程序各 300 份,一共 600 份.

5.1 结果和讨论

在恶意和正常这个二分类问题:True Positive(TP)表示恶意程序样本正确分类为恶意,False Negative(FN)表示恶意样本错误分类为正常;同时,True Negative(TN)表示正常样本正确分类为正常,False Positive(FP)表示正常样本错误分类为恶意.其中,精度(accuracy)表示分类正确的样本数占样本总数的比例,而查准率(precision)、查全率(recall)和 $F1$ 分别定义为

$$\left.\begin{aligned} P&=\frac{TP}{TP+FP} \\ R&=\frac{TP}{TP+FN} \\ F1&=\frac{2\cdot P\cdot P}{P+R} \end{aligned}\right\} \tag{10}$$

与 MDC[11]方法对同一数据集进行了各种性能测试,数据集中随机选取 90%的数据用于训练,剩余 10%的数据用于测试,实验重复了 10 次且取平均值作为最终结果.

准确性:符号(1)、符号(2)分别表示矩阵 M 的计算的两种方法,从表 2 中可以明显看出,DMBSS(2)检测率高于 MDC 方法,而 DMBSS(1)略微低于 MDC 方法.造成这种现象的原因,我们认为,方法(1)对子图的相似度计算太过简单,而使用更加复杂的深度图内核来计算子图间的相似性,却使检测效果出现明显的提高.因而我们可以认为,子图间的相关性考虑对检测效果的提升是明显的.

Table 2 Performance comparison between two methods

表 2 两种方法的性能比较

	Accuracy	Precision	Recall	$F1$	Extract time(s)	Train time(s)	Detect time(s)
MDC	0.938±0.023	0.934±0.019	0.937±0.022	0.941±0.021	–	46.3±2.5	1.28±0.19
DMBSS(1)	0.932±0.021	0.931±0.016	0.935±0.025	0.933±0.012	13.5±1.4	18.3±2.1	1.07±0.04
DMBSS(2)	0.948±0.019	0.952±0.026	0.947±0.017	0.948±0.020	13.5±1.4	17.8±1.7	0.63±0.01

有效性:虽然我们的方法多了对子图的处理过程,即子图提取和表示,但总体上,我们的方法时间开销更小.因为我们通过提取待测程序数据流图的所有子图,之后按逆拓扑标识方法映射成字符串,然后与子图库中特征

子图比较来判断其包含的特征子图,这可以避免进行图匹配,从而使得整个算法效率能够明显提高.

5.2 参数最优化

本小节中将讨论实验中参数的最优化.图 6 列出了最大深度 D 以及选择阈值 t 的参数效果图,其中,符号(1)、符号(2)分别表示矩阵 M 的两种计算方法.从图 6(a)和图 6(b)可以看出,随着最大深度 D 的增大,其精度也在提高;但深度 $D\geqslant 5$ 后预测结果花费的时间(提取子图和训练的时间总和)却呈现大幅度增长,我们可以认为提取子图的深度最佳为 5.图 6(c)和图 6(d)表明,选择阈值 t 增加,其精度呈现下降趋势,而花费的时间呈上升趋势.这是因为 t 取值越大,剔除的子图越少,增加了子图库中的无关子图数,从而增加了时间开销.

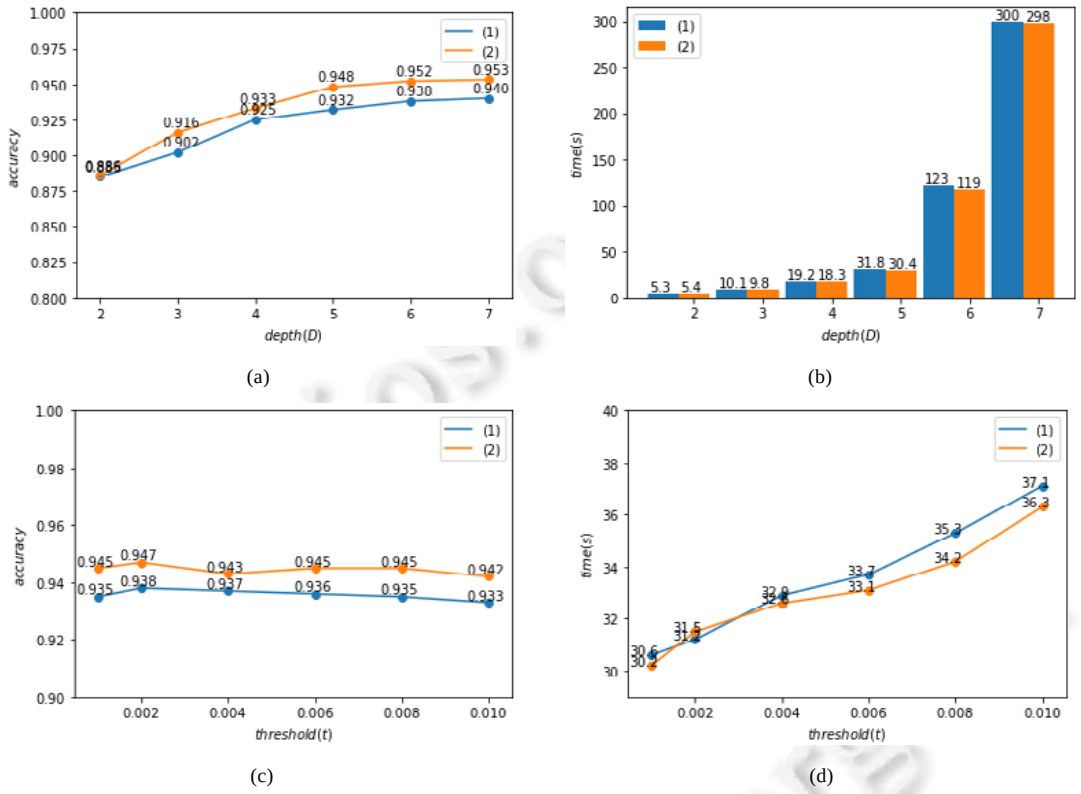


Fig.6 Effect chart of different parameters

图 6 不同参数的效果图

6 结 论

本文提出了基于子图相似性的恶意程序检测方法,利用神经网络学习数据流图中子图的分布式表示,使得上下文相似的子图其向量也相近;然后使用子图向量构建图相似性函数,同时,结合 SVM 分类器算法进行分检测;最后,实验证实我们的方法在恶意程序检测上的准确性和有效性.本文的下一步工作考虑数据流图中的数据量,即将子图的数据量也纳入进检测模型.引入子图的数据量,可以进一步优化检测模型.同时,我们的方法也可以向安卓平台进行拓展,问题的关键是构建安卓端的系统行为图.

References:

[1] Caballero J, Grier C, Kreibich C, Paxson V. Measuring pay-per-install: The commoditization of malware distribution. In: Proc. of the Usenix Security Symp. 2011. 13–31.

[2] Holz T, Engelberth M, Freiling F. Learning more about the underground economy: A case-study of keyloggers and dropzones. In: Backes M, Ning P, eds. Proc. of the Computer Security (ESORICS 2009). Springer-Verlag, 2009. 1−18.

[3] Liu J, Su PR, Yang M, He L, Zhang Y, Zhu XY, Lin HM. Software and cyber security—A survey. Ruan Jian Xue Bao/Journal of Software, 2018,29(1):42−68 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5320.htm [doi: 10.13328/j.cnki.jos.005320]

[4] Miao XC, Wang R, Xu L, Zhang WF, Xu BW. Security analysis for Android applications using sensitive path identification. Ruan Jian Xue Bao/Journal of Software, 2017,28(9):2248−2263 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5177.htm [doi: 10.13328/j.cnki.jos.005177]

[5] Fredrikson M, Jha S, Christodorescu M, Sailer R, Yan X. Synthesizing near-optimal malware specifications from suspicious behaviors. In: Proc. of the 2010 IEEE Symp. on Security and Privacy (SP 2010). Washington: IEEE Computer Society, 2010. 45−60.

[6] Eskandari M, Hashemi S. A graph mining approach for detecting unknown malwares. Journal of Visual Languages and Computing, 2012,23(3):154−162.

[7] Martinelli F, Saracino A, Sgandurra D. Classifying Android malware through subgraph mining. In: Garcia-Alfaro J, Lioudakis G, Cuppens-Boulahia N, Foley S, Fitzgerald W, eds. Proc. of the Data Privacy Management and Autonomous Spontaneous Security. Berlin, Heidelberg: Springer-Verlag, 2014. 268−283.

[8] Christodorescu M, Jha S, Kruegel C. Mining specifications of malicious behavior. In: Proc. of the 1st India Software Engineering Conf. New York: ACM, 2008. 5−14.

[9] Park Y, Reeves DS, Stamp M. Deriving common malware behavior through graph clustering. In: Proc. of the 6th ACM Symp. on Information, Computer and Communications Security. New York: ACM, 2011. 497−502.

[10] Fan M, Liu J, Wang W, Li HF, Tian ZZ, Liu T. DAPASA: Detecting Android piggybacked apps through sensitive subgraph analysis. IEEE Trans. on Information Forensics and Security, 2017,12(8):1772−1785.

[11] Andriatsimandefitra R, Tong VVT. Detection and identification of Android malware based on information flow monitoring. In: Proc. of the 2015 IEEE 2nd Int'l Conf. on Cyber Security and Cloud Computing. New York: IEEE, 2015. 200−203.

[12] Fan M, Liu J, Luo X, Chen K, Chen T, Tian Z, Zhang X, Zheng Q, Liu T. Frequent subgraph based familial classification of android malware. In: Proc. of the 2016 IEEE 27th Int'l Symp. on Software Reliability Engineering (ISSRE). IEEE, 2016. 24−35.

[13] Karbalaie F, Sami A, Ahmadi M. Semantic malware detection by deploying graph mining. Int'l Journal of Computer Science Issues, 2012,9(1):373−379.

[14] Wuchner T, Cislak A, Ochoa M, Pretschner A. Leveraging compression-based graph mining for behavior-based malware detection. IEEE Trans. on Dependable and Secure Computing, 2017,16(1):99−112.

[15] Banescu S, Wuchner T, Guggenmous M, Ochoa M, Prertschner A. An empirical evaluation framework for malware behavior obfuscation. In: Proc. of the 10th Int'l Conf. on Malicious and Unwanted Software. Washington: IEEE Computer Society, 2015. 40−47.

[16] Wuchner T, Ochoa M, Pretschner A. Malware detection with quantitative data flow graphs. In: Proc. of the 9th ACM Symp. on Information, Computer and Communications Security. New York: ACM, 2014. 271−282.

[17] Shervashidze N, Schweitzer P, Leeuwen EJV, Mehlhorn K. Weisfeiler-Lehman graph kernels. Journal of Machine Learning Research, 2011,12:2539−2561.

[18] Narayanan A, Chandramohan M, Chen LH, Liu Y, Saminathan S. Subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. In: Proc. of the Computing Research Repository. 2016. 1−8.

[19] Mikolov T, Sutskever I, Chen K, Greg C, Dean J. Distributed representanitons of words and phrases and their compositionality. In: Proc. of the Computer Science. 2013. 3111−3119.

[20] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. In: Proc. of the Workshop at ICLR. 2013. 1301−1312.

[21] Mikolov T, Le QV, Sutskever I. Exploiting similarities among languages for machine translation. In: Proc. of the Computer Science. 2013. 1−10.

[22] Yanardag P, Vishwanathan SVN. Deep graph kernels. In: Proc. of the 21th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM, 2015. 1365−1374.

附中文参考文献:

[3] 刘剑,苏璞睿,杨珉,和亮,张源,朱雪阳,林惠民.软件与网络安全研究综述.软件学报,2018,29(1):42−68. http://www.jos.org.cn/1000-9825/5320.htm [doi: 10.13328/j.cnki.jos.005320]

[4] 缪小川,汪睿,许蕾,张卫丰,徐宝文.使用敏感路径识别方法分析安卓应用安全性.软件学报,2017,28(9):2248−2263. http://www.jos.org.cn/1000-9825/5177.htm [doi: 10.13328/j.cnki.jos.005177]

汪洁(1980－),女,博士,副教授,CCF 专业会员,主要研究领域为信息与网络安全.

王长青(1994－),男,硕士,主要研究领域为恶意程序检测.