软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(3):522−533 [doi: 10.3724/SP.J.1001.2011.03727]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

提升多维特征检测迷惑恶意代码*

孔德光[1,2+], 谭小彬[1], 奚宏生[1], 官 涛[1], 帅建梅[1]

[1](中国科学技术大学 自动化系,安徽 合肥 230027)

[2](Cyber-Security Laboratory, The Pennsylvania State University, University Park, State College, 16801, USA)

Obfuscated Malware Detection Based on Boosting Multilevel Features

KONG De-Guang[1,2+], TAN Xiao-Bin[1], XI Hong-Sheng[1], GONG Tao[1], SHUAI Jian-Mei[1]

[1](Department of Automation, University of Science and Technology of China, Hefei 230027, China)

[2](Cyber-Security Laboratory, The Pennsylvania State University, University Park, State College, 16801, USA)

+ Corresponding author: E-mail: kdg@mail.ustc.edu.cn

Kong DG, Tan XB, Xi HS, Gong T, Shuai JM. Obfuscated malware detection based on boosting multilevel features. *Journal of Software*, 2011,22(3):522−533. http://www.jos.org.cn/1000-9825/3727.htm

Abstract: To cope with the problem of the low accuracy in detecting obfuscated malware, an algorithm to detect obfuscated malware based on boosting multi-level features is presented. After a disassembly analysis and static analysis for the obfuscated malware, the algorithm extracts features from three dimensions: opcode distribution, a function call graph, and a system call graph, which combines the statistic and semantic features to reflect the behavior characteristic of the malware, and then gives out the decision result based on weighted voting for a different feature analysis. It has been proven by experiment that the algorithms have a much higher accuracy on the testing dataset.
Key words: malware detection; multi-feature; obfuscate; boosting

摘 要: 针对迷惑恶意代码识别率较低的问题,提出一种基于提升多维特征的迷惑恶意代码检测算法.该算法在对迷惑恶意代码反汇编后进行静态分析,从Opcode分布序列、调用流图特征、系统调用序列图这3个特征维度对恶意代码家族特征进行归纳和分析,结合统计和语义结构特征表现恶意代码“行为”特性,从而对分类结果加权投票后给出迷惑恶意代码家族判定信息.实验结果表明,该方法对迷惑恶意代码家族检测准确率较高.
关键词: 恶意代码检测;多维特征;迷惑;提升
中图法分类号: TP309 **文献标识码**: A

恶意代码是在一定环境下执行对计算机系统或网络系统机密性、完整性、可用性产生威胁,具有恶意企图的代码序列,例如virus,worm,backdoor等.据Symantec公司2008年发布因特网安全公告称,2007年全球检测到新恶意代码攻击数量达711 912个,比2006年同期增长468%[1].对于恶意代码的检测与防范,按照检测位置差别分为基于主机和基于网络两种方式[2]:基于主机检测有基于特征码签名方式[3−5]、校验和方式[6]、启发式数据挖掘方法[7,8]等;基于网络检测有基于Deep packet Inspection[9]以及基于HoneyPot[10]的检测方式.按照恶意代码检

* 基金项目: 国家高技术研究发展计划(863) (2006AA01Z449)
收稿时间: 2009-03-19; 修改时间: 2009-06-01; 定稿时间: 2009-08-28

© 中国科学院软件研究所 http://www.jos.org.cn

测时代码状态的差别,分为静态和动态恶意代码检测方式.静态方式[3,4]通过对恶意代码进行静态分析来判定其是否有恶意行为,基于特征码签名方式的校验和检验都是静态检测.在动态检测的方式中,通过对代码在虚拟机[11,12]或仿真器中虚拟执行或者仿真执行获得代码相应执行行为,获得代码执行路径和相关语义信息进行检测与分析,区分恶意与非恶意代码.静态方式检测有受到静态代码分析技术不可判定性的局限;而动态检测技术检测过程开销较大,占用资源较多.从攻击者角度出发,出现各种代码隐藏和变换技术来逃避检测器的检测,实现其恶意目的.多态(polymorphic)[13]和变形(metamorphic)技术是通过迷惑恶意代码变换提高生存性的两种主要技术,通过加壳(加密、压缩)或指令变换逃避恶意代码检测器检测.多态代码通过加密恶意代码有效载荷,代码执行时解密;在解密循环中,使用各种保持语义等价的代码变换技术,如通过改变指令执行顺序、插入 NOP 指令、插入条件跳转 jump 指令、寄存器重新分配、变量重命名等来迷惑解密循环.变形恶意代码通过迷惑整个恶意代码序列本身,在代码被复制或传播时,通过各种代码变换技术(例如改变条件指令、寄存器重新分配等)达到语义等价和行为等价的目的.迷惑变换通过对恶意代码进行各种变换来对抗恶意代码的检测.一个恶意代码经多态或变形后,与源代码仍旧是同源恶意代码,代码表现形式有略微或较大不同,而实现相同恶意行为,可以称为一个恶意代码家族.另外,恶意代码家族在进化过程中,新的行为也有可能加入到已有的恶意代码中.例如蠕虫家族 Sobig 中,从版本 Sobig.A 到 Sobig.F[14,15],该蠕虫病毒只是增加或者减少一些特性,其行为近似相同.蠕虫家族 Beagle worm 则从版本 A 到版本 C 进行了不断的演化和升级,包括增加了后门、增加了阻碍本地安全机制的代码、增添了更好的在已有进程中隐藏蠕虫的机制.总之,一个家族的恶意代码行为或者完全相同,或者大致相同略有进化和改变.

如何在较低系统开销下提高检测识别率,降低误报和漏报率,成为恶意代码检测尤其是迷惑恶意代码检测的一个难点.由于传统方法大多都针对恶意代码进行检测,对变换后的家族恶意代码检测率较低.本文提出一种基于多维特征提升的迷惑恶意代码家族检测算法,用于解决迷惑恶意带检测识别困难的问题.该算法在对迷惑恶意代码反汇编后,从 Opcode 分布序列、调用流图特征、系统调用序列图 3 个维度对恶意代码家族特征进行提取、分析和归纳.在此基础上,根据恶意代码特征对结果进行加权投票决策,从而给出迷惑恶意代码家族判定信息.检测过程中借助于代码静态分析技术,并充分利用代码本身统计和语义结构特征,以静态的方式表现出代码“行为”特性,从而在较低系统开销下达到较高的识别效果.

该算法的主要特点:1) 主要针对迷惑恶意代码问题提取每个家族迷惑恶意代码特征,而不仅仅急是针对每个恶意代码特征;2) 代码的统计特征(opcode 出现的频度)和代码的语义特征相结合(相关控制流图和 API 调用信息),而不仅仅是类似于 fingerprint 匹配中的代码字节信息.

本文第 1 节给出迷惑代码分析和检测的一些背景知识.第 2 节给出迷惑恶意代码检测框架.第 3 节给出代码检测的核心算法.第 4 节是实验结果分析.第 5 节介绍相关工作.最后是总结和展望.

1 预备知识

1.1 迷惑恶意代码定义和检测标准[16]

定义 1.1(迷惑恶意代码定义和检测). 令 P 为程序的集合,一个迷惑过程可以看成一个程序变换函数 $o:p\to p'(p,p'\in P)$.所有的迷惑变换集合记为 $O(o\in O)$.一个恶意代码检测器 D 用于判定任意一个程序 p 是否是被一个恶意代码 $m(m\in P)$所感染,记为 $D:P\times M\to\{true,false\}$,定义

$$D(p,m)=\begin{cases}true, & \text{if } D \text{ recognize } p \text{ is infected by } m\\ false, & \text{other}\end{cases}.$$

当一个程序 p 被一个恶意代码 m 所感染,则定义为 $m\mapsto p$.如果一个恶意代码检测器检测出的程序都是被感染的程序,则称其为健壮的(sound),误报率为 0.如果一个恶意代码检测器能检测出所有被感染的程序,则称为完整的(complete),漏报率为 0.如果一个恶意代码检测器是完整的,则对于一个迷惑变换 $o\in O$,当且仅当

$$\forall m\in P, o(m)\mapsto P\Rightarrow D(p,o(m))=true.$$

如果一个恶意代码检测器是健壮的,则对于一个迷惑变换 $o\in O$,当且仅当

© 中国科学院软件研究所 http://www.jos.org.cn

$$\forall m \in P, D(p, o(m)) = true \Rightarrow o(m) \mapsto P.$$

1.2 *N*-perm算法

n-gram 算法最早用于文本分类和信息检索中[17],核心思想是计数连续 *n* 个词出现的频率.*n*-perm 算法是 *n*-gram 算法的变形,不同之处在于统计连续 *n* 个词出现的频率时忽略了次序.以图 1 所示代码为例,在对可执行文件反汇编后提取操作码(GetOpcode)操作之后,统计每条指令中 opcode 出现的频率,使用(操作码(序列),出现频率)二元组作为特征向量.使用 1-perm 算法,得到模式({*push*,*mov*,*pop*,*jmp*},{2,3,1,1});使用 2-perm 算法,得到模式({*push*-*mov*,*jmp*-*mov*,*mov*-*pop*},{3,2,1});使用 3-perm 算法,得到模式({*push*-*mov*-*push*,*mov*-*push*-*mov*,*push*-*mov*-*jmp*,*mov*-*jmp*-*mov*,*jmp*-*mov*-*pop*},{1,1,1,1,1}).当 *n*=4,5 时,算法相同.

```
call 0x1c
inc eax
pop esi
xor dword[esi+0xe], 0xfd4cdb57
sub esi, 0x4
loop 0x1f
db 0xC7
```

Fig.1 Polymorphic code
图 1 多态代码

```
push eax
mov[edi], 0x04
push ecx
mov ecx, 0x04
jmp 0x3456
mov[edi], ecx
pop ecx
```

Fig.2 Assemble code for an executable
图 2 可执行文件反汇编代码

2 迷惑恶意代码检测原理

2.1 检测系统架构

由于恶意代码分家族,对于每类(家族)恶意代码,选取迷惑前后的多个样本进行分析.整个系统分为恶意代码特征提取流程和待检测迷惑代码分析流程两个过程.其中:恶意代码归纳流程用于提取每族恶意代码的特征;待检测恶意代码通过分析流程,与恶意代码知识库中的特征进行匹配,对检测结果加权投票后判决该恶意代码家族或判定为非恶意代码.其中:恶意代码特征能够提取过程输入为恶意代码家族,主要由反汇编与静态分析、特征提取与选择、分类器构建 3 个模块组成;待检测代码分析流程输入是欲检测恶意代码,由反汇编与静态分析、特征提取与选择、分类器识别 3 个模块组成(如图 3 所示).

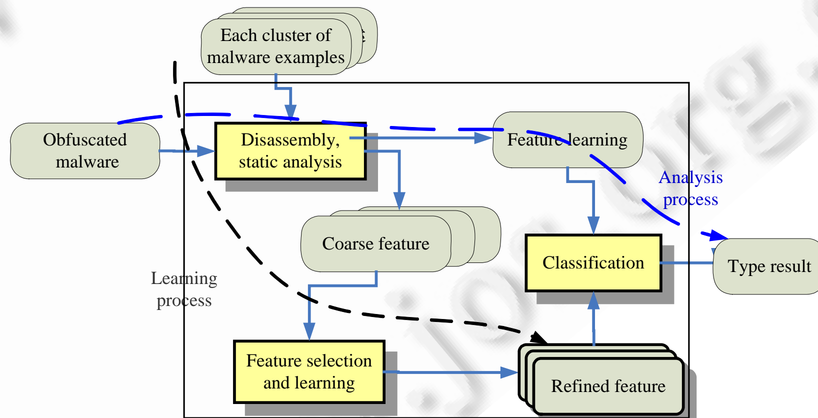

Fig.3 Architecture for the obfuscated malware detection system design
图 3 迷惑恶意代码检测系统架构设计

以下介绍特征提取与选择模块(如图 4 所示),我们假定迷惑后的恶意代码都能够正确进行反汇编.对可执行文件,经过文件装载器分析后,判断文件是否加壳:如果加壳,要经过脱壳操作后,对脱壳后的文件进行反汇编分析;否则,直接对文件进行反汇编分析.我们使用 IDA pro 工具[23]对预处理后的可执行文件进行反汇编,然后对汇编代码静态分析,提取的主要特征包括:1) 操作符 Opcode 频度(*n*-perm 算法);2) 可执行文件的函数调用图(call

© 中国科学院软件研究所 http://www.jos.org.cn

graph);3) 可执行文件系统调用(system call)序列.该模块输入是第 i 类恶意代码的 K 个样本集合,A_{ij} 表示第 i 类恶意代码的第 j 个样本($1 \leqslant j \leqslant K$).

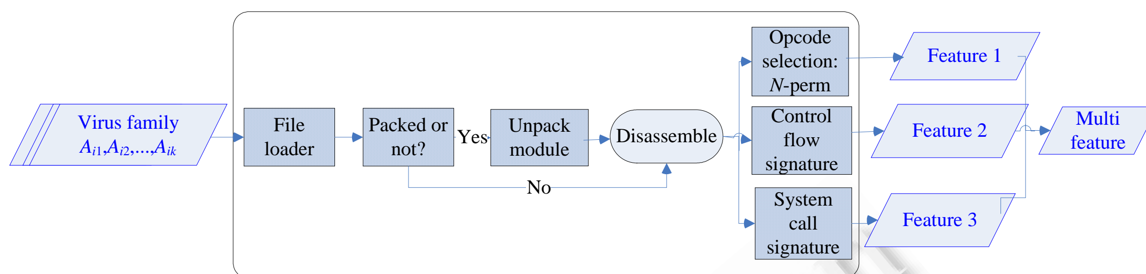

Fig.4 Feature selection process for obfuscated malware

图 4 迷惑恶意代码特征提取过程

2.2 待检测迷惑代码分析流程

对于新输入的迷惑代码,在得到已有恶意代码特征库后进行分类识别.利用第 2.1 节中的 n-perm,call graph, system call 特征,分别构建分类器 D_1,D_2,D_3(分类器个数 $L=3$,L 个分类器之间相互独立).利用集成学习的原理[21,22],将 3 个分类器决策结果加权投票后给出判决结果.集成学习通过组合不同分类器的多维特征,对单个分类器的缺陷进行优势互补,从而提升检测准确率.本文采用加权多数投票算法,充分发挥单个分类器的优势,用于提升恶意代码检测准确率.每一个分类器 $D_i(1 \leqslant i \leqslant L)$标记样本 x 为类别 $w_j,j=1,2,\ldots,c,c$ 为恶意代码类别总数,记类别集合$\Omega=\{w_1,w_2,\ldots,w_c\}$.记检测结果:

$$d_{ij}=\begin{cases}1, & \text{if } D_i \text{ labels } x \text{ as } w_j\\ 0, & \text{otherwise}\end{cases}.$$

类别 w_j 的决策函数 $g_j(x)$通过加权投票得到 $g_j(x)=\sum_{i=1}^{L} b_i d_{ij}$,其中,b_i 是分类器 D_i 的权重系数,满足 $\sum_{i=1}^{L} b_i=1$. 判定样本 $x \in w_k$,满足 $\sum_{i=1}^{L} b_i d_{ik}=\max_{j=1,2,\ldots,c} \sum_{i=1}^{L} b_i d_{ij}$.每个分类器 D_i 的检测准确率记为 p_i,为了最大化集成学习的准确率,权重 b_i 由 p_i 决定,满足 $b_i \propto \log \frac{p_i}{1-p_i}$.

定理 1. 当 $L=3$ 时,按照 $b_i \propto \log \frac{p_i}{1-p_i}$ 设定加权投票权值,集成学习准确率优于单个或两个分类器集成.

证明:首先说明这样设置加权投票权值 b_i 的原因,然后在此权值下进行分类器集成的准确率比较.

(1) 记 $s=[s_1,s_2,s_3]^t$ 表示集成分类器中每个 D_i 对样本 x 分类后所得结果组成的向量,$s_i \in \Omega$.那么,L 个分类器的 Bayes 最优决策函数 $g_j(x)=\log p(w_j)p(s|w_j),j=1,2,\ldots,c$.由条件概率独立性可得

$$\begin{aligned} g_j(x) &= \log p(w_j)\prod_{i=1}^{L} p(s_i \mid w_j) \\ &= \log p(w_j)+\log \prod_{i,s_i=w_j} p(s_i \mid w_j) \prod_{i,s_i \neq w_j} p(s_i \mid w_j) \\ &= \log p(w_j)+\log \prod_{i,s_i=w_j} p_i \prod_{i,s_i \neq w_j} (1-p_i) \\ &= \log p(w_j)+\log \prod_{i,s_i=w_j} \frac{p_i}{1-p_i}(1-p_i) \prod_{i,s_i \neq w_j} (1-p_i) \end{aligned}$$

© 中国科学院软件研究所 http://www.jos.org.cn

$$=\log p(w_j)+\log\prod_{i,s_i=w_j}\frac{p_i}{1-p_i}\prod_{i=1}^{L}(1-p_i)$$

$$=\log p(w_j)+\sum_{i,s_i=w_j}\log\frac{p_i}{1-p_i}+\sum_{i=1}^{L}\log(1-p_i).$$

由于最后一项 $\sum_{i=1}^{L}\log(1-p_i)$ 与分类器的决策无关,判决函数可以写成 $g_j(x)=\log p(w_j)+\sum_{i=1}^{L}d_{ij}\log\frac{p_i}{1-p_i}$.对比判决函数 $g_j(x)=\sum_{i=1}^{L}b_i d_{ij}$,从而得到 $b_i\propto\log\frac{p_i}{1-p_i}$.

(2) 分析两个分类器集成时,不妨设 $b_1\leqslant b_2$,那么 $\log\frac{p_1}{1-p_1}\leqslant\log\frac{p_2}{1-p_2}$,易知 $p_1\leqslant p_2$.同时,依据判决函数,此时集成分类器准确率为 p_2,与一个分类器 D_2 工作时准确率相同.下面分析 3 个分类器集成的情形,不妨设 $b_1\leqslant b_2\leqslant b_3$,由鸽巢原理可知,$b_1\leqslant b_2\leqslant\frac{1}{3}\leqslant b_3$ 或者 $b_1\leqslant\frac{1}{3}\leqslant b_2\leqslant b_3$ 中必有一式成立.

以下考察 $b_1\leqslant b_2\leqslant\frac{1}{3}\leqslant b_3$ 成立的情形,$b_1\leqslant\frac{1}{3}\leqslant b_2\leqslant b_3$ 的情形同理可证:A) 当 $b_1+b_2\leqslant b_3$ 时,集成分类器的准确率为 $p_3(1-p_1)(1-p_2)+p_1p_3(1-p_2)+p_2p_3(1-p_1)+p_3p_1p_2=p_3$,集成分类器准确率等同于单个分类器 D_3 准确率;B) 当 $b_1+b_2>b_3$ 时,由 $\log\frac{p_1}{1-p_1}+\log\frac{p_2}{1-p_2}>\log\frac{p_3}{1-p_3}$ 可得,$\frac{p_1}{1-p_1}\frac{p_2}{1-p_2}>\frac{p_3}{1-p_3}$.比较此时集成分类器的准确率与单个(两个集成)分类器的准确率(p_3)差值:

$$l=p_1p_2(1-p_3)+p_1p_3(1-p_2)+p_2p_3(1-p_1)+p_3p_1p_2-p_3=\prod_{i=1}^{3}(1-p_i)\left(\prod_{i=1}^{2}\frac{p_i}{1-p_i}-\frac{p_3}{1-p_3}\right)>0.$$

故当 L=3 时,集成学习分类器按照上述权值投票集成时,准确率得到提升. □

3 特征提取和检测中的核心算法

3.1 *N*-perm特征提取和检测算法

该算法通过对反汇编后的恶意代码进行分析,在提取汇编代码的操作符后,使用 *n*-perm 算法统计每个样本 A_{ij} 中操作符出现的频率,并存储在向量$(v_1,v_2,\ldots,v_n)$中.对 k 个样本特征按照权值取数学期望,从而使代表特征向量位于 k 个样本特征的中心,使该特征向量 $\bar{v}$ 能够最好地描述 k 个样本的性质.在每次迷惑代码的输入过程中,代表 k 样本聚类中心的特征不断发生改变.权值的确定在初始时设置相同数值,在分析阶段,通过训练进行调节.操作符提取 *N*-perm 模块训练时使用操作码序列(如(*push*,*call*,*ret*,*mov*,*xor*))作为特征,使用 SVM[18]算法对多类迷惑恶意代码进行分析和识别.令 c 表示恶意代码类别数目,类别标号记为 $w_1,w_2,\ldots,w_c$.由于是多类 SVM 分类器,使用 one-versus-all 方法来构建 c 个二类分类器.第 i 个分类器判决函数 f_i 使用来自类别 w_i 的恶意代码作为正例,来自非 w_i 类别的恶意代码作为反例进行训练.对于新的待判定恶意代码测试样本 x,判定 $x\in w_i$, $i=\arg\max_{j=1,2,\ldots,c} f_j(x)$.以下使用标准的 SVM 两类分类器算法构建 c 个二类分类器.x_i 是训练样本特征,$y_i\in\{1,-1\}$来标记该样本是 w_i 类的正例还是反例,$k(x_i,y_i)$是满足 Mercer 条件[18]的核函数将输入空间的样本映射到某一高维特征空间中,C 是正常数,$a_i>0$ 为拉格朗日乘子.训练过程中优化下列目标函数:

$$\max W(a_i)=\sum_i a_i-\frac{1}{2}\sum_{i,j}a_ia_jy_iy_jk(x_i,x_j) \tag{3.1}$$

其中,满足约束 $0\leqslant a_i\leqslant C$, $\sum_i a_iy_i=0$.对于样本 x 的决策函数定义如下:

$$f(x)=sign(w\cdot x+b)=sign\left(\sum_i a_iy_ik(x_i,x)+b\right) \tag{3.2}$$

© 中国科学院软件研究所 http://www.jos.org.cn

其中,w 是确定最优分类面的权系数向量,表示为训练样本向量的线性组合 $w=\sum_i a_i y_i k(x_i\cdot x)$;$b$ 是分类的阈值.在训练中选用了径向基核函数 $k(x_i,x)=\exp\left(-\frac{|x-x_i|^2}{\delta^2}\right)$($\delta$为参数);利用二次规划方法来求解目标函数公式(3.1)的最优解,得到最优拉格朗日系数 a,阈值 b 可通过任一支持向量使用公式(3.2)求得.若仅仅使用 N-perm 特征提取,易受到攻击者有效攻击(见附录 1),准确率较低.以下介绍提取 call-graph 和 system call 特征.

3.2 可执行文件函数调用流图分析算法

图 $G=(V,E)$,V 是顶点的集合,E 是边集合,$E=\{(v_i,v_j),v_i\in V,v_j\in V\}$.函数调用图中,$v_i$ 代表子过程(如 sub_42DC60),边代表函数间的调用关系.如果 sub_42DC60 调用 sub_42D583,则存在从结点 sub_42DC60 指向 sub_42D583 的边(sub_42DC60,sub_42D583).一个可执行文件由一系列的函数组成,其函数集合 $F=\{(v_1,v_2,\ldots,v_n),\forall v_i\in V\}$,$v_i.suc$ 和 $v_i.pre$ 分别记录该节点的前驱和后继的数目.基于观察和分析基础上认为,同族恶意代码相似性远大于不同族恶意代码的相似性.提取该图节点前驱和后继数目作为统计特性,调用流图在某种程度上反映语义特性,从而使恶意代码行为得到反映.对于函数调用流图,每个恶意代码样本 A_{ij} 都生成一个特定的调用流图签名 G_{ij}.每族 i 类共 k 个样本,构成了 k 个有向图.由于 k 个有向图都是同一族恶意代码的变体,其相似性远大于该样本和其他族中样本的相似性.求 k 个有向图的公共子图 G_i' 作为特征图,使用 $v.prenum$ 和 $v.sucnum$ 记录图中每个节点的前驱和后继的数目.

作为其特征图,求解 k 个有向图的公共子图 G_i' 的算法,取该公共子图中结点满足为所有子图中都出现的节点,即对于 $v\in G_i'$,寻找 j,使得$\forall j$ 满足 $1\leqslant j\leqslant k$ 并且$\exists v_l\in G_{ij}$,使 $v.prenum=v_l.prenum$ 和 $v.sucnum=v_l.sucnum$ 成立.待新输入恶意代码后,将恶意代码 G_{new} 与各个类别的特征代码子图进行匹配,得到其类型

$$type(G_{new})=\max_k\{score(G_{new},G_k)\}.$$

设 G_{new} 和 G_k 分别由 m 个和 n 个顶点,其中,$v_{newi}\in G_{new},v_{kj}\in G_k$,那么评分函数可以定义如下($\gamma_1$ 为相似性参数表示节点之间相似性,$0<\gamma_1<1$):

$$score_{call_graph}(G_{new},G_k)=\sum_{i=1}^{m}\sum_{j=1}^{n}\delta_{ij},\delta_{ij}=\begin{cases}1, & v_{newi}.prenum=v_{kj}.prenum\ \&\&\ v_{newi}.sucnum=v_{kj}.sunnum\\ \gamma_1, & v_{newi}.prenum=v_{kj}.prenum\ \|\ v_{newi}.sucnum=v_{kj}.sunnum\\ 0, & \text{else}\end{cases}.$$

3.3 系统调用流图特征提取和分析算法

与函数调用图图结构特征相同,但是语义不同.系统调用图中,v_i 代表系统调用名(如 RegOpenKeyExA),边代表系统调用间的前驱和后继关系.如果系统调用 2 在系统调用 1 后被执行,则存在从系统调用 1 指向系统调用 2 的边.$v.SystemCall()$表示该图 $G=(V,E)$中节点 v 所对应的系统调用名称.$v_a.pre\in G_a,v_b\in G_b,v_a.pre(i)$是图中节点 v_a 的第 i 个前驱节点,$v_a.prenum$ 表示节点 v_a 的前驱节点的个数,$v_a.suc(i)$是图中节点 v_a 的第 i 个后继节点.$v_a.prenumnum$ 表示节点 v_a 的后继节点的个数.迷惑前后的代码其系统调用特性一般说来改变较小,从而相同性和相似性较高.不同于以往工作是动态运行截获系统调用序列[24,25],我们使用静态分析方式获得系统调用流图,其难点在于生成过程间的系统调用序列,根据不同的控制流转移类型确定图中不同的系统调用节点的连接(见附录 2),较精确地生成了系统调用流图.

对于系统调用流图,每个恶意代码样本 A_{ij} 都生成一个特定的系统调用流图签名 G_{ij}.每族 i 类共 k 个样本,构成了 k 个有向图.由于 k 个有向图都是同一族恶意代码的变体,其相似性远大于该样本和其他族中样本的相似性.求 k 个有向图的公共子图 G_i' 作为其特征子图,使用 $v.prenum$ 和 $v.sucnum$ 记录系统调用图中每个节点的前驱和后继的数目.作为其特征图,求解 k 个有向图的公共子图 G_i' 的算法,即该公共子图中结点满足为所有子图中都出现的节点,即 $v\in G_i'$ 等价于对于$\forall j,1\leqslant j\leqslant k$,都$\exists v_l\in G_{ij}$ 使得 $v.prenum=v_l.prenum$ 并且 $v.sucnum=v_l.sucnum$ 和 $v.systemcall=v_l.systemcall$.分别使用不同谓词 *same*,*simi*,*diff* 描述不同节点之间相同、相似和不同(见附录 2).在获得两个系统调用流图的相似性矩阵,使用如下评分函数对相似性进行打分:

$$score_{sys_call}(A,B)=\sum_{i=1}^{m}\sum_{j=1}^{n}\delta_{ij},\delta_{ij}=\begin{cases}1, & same(A_i,B_j)=true\\ \gamma_2, & simi(A_i,B_j)=true\\ 0, & diff(A_i,B_j)=true\end{cases},$$

其中,γ_2为相似性度量参数,$0<\gamma_2<1$.选择类型 $type(G_{new})=\max_k\{score(G_{new},G_k)\}$ 作为最终恶意代码类型.

4 实验结果分析

4.1 实验设置

实验平台选取为测试环境:Pentium® 4,3.0G Processor,512M 内存,Microsoft Window XP Professional,IDA Pro 5.0[23],Matlab7.1,Visual Studio.net2003.本文选取的二进制恶意代码实验样本来自 VX Heavens(http://vx.netlux.org),同时,选取正常代码 400 个(例:winzip,daemon, AcroRead 以及计算 $n!$等用户非恶意可执行代码).实验中所选取恶意代码家族 99 个(例:一族 Backdoor.Win32.Iroffer.1303 共选取了后缀 a 至 m 共 12 个样本作为测试样本,其中,至少留一个未经训练样本作为测试样本,后缀 o 用于测试),恶意代码总样本数数 542.测试过程中,99 类恶意代码加上 1 类正常代码作为训练集合进行训练.由于每类恶意代码的样本数至少为 4 个,我们进行了 4 重交叉验证,保证用于不同训练样本改变某一个进行训练时,所使用的测试样本都不同.定义准确率 ACC 为测试样本中能够被正确分类的样本在所有测试样本中所占的比例;误报率 $FP1$ 为第 i 类恶意代码被误报为第 j 类恶意代码的样本在所有测试样本中所占的比例,误报率 $FP2$ 为正常代码为误报为恶意代码的样本在所有测试样本中所占的比例;漏报率 NP 为第 i 类恶意代码样本被漏报为正常代码的测试样本在所有测试样本中所占的比例.分别对 n-perm,call graph,system call graph 以及 boosting 算法的准确率进行比较和测试.

在基于 n-perm 的分类实验中,提取的特征为出现频度最多的 10 个操作符;在构造分类器时,采取了构造 99 个二类恶意代码分类器 SVM 的方法,每个二类分类器由一族恶意代码样本组成一类训练样本 i,剩余恶意代码组成另一类训练样本,进行训练.另外一个二类分类器用于训练正常样本,其正例为正常样本,反例为恶意代码样本.构建分类器后,根据第 3.1 节中判决函数给出判决结果.在计算识别率时,给出了 4 重交叉验证的最终结果.使用 one-versus-all 方法训练分类器时,训练数据中存在极大不平衡,分类器性能下降较大;采用多次重复恶意代码样本的方法增加恶意代码样本的数量;另外,使用一些迷惑变换(寄存器重命名,无效指令插入等)产生同族一些恶意代码,从而平衡训练数据集合.在提取调用流图时,提取了 99 类恶意代码中每类调用流图的特征.由于正常代码之间差别较大,从而导致图结构形态各异难以发现其公共子图,故没有对正常代码的特征流图和系统调用流图进行提取.调用流图特征识别的准确率依赖于参数γ_1,调节参数γ_1 的不同值对分类结果产生不同影响.在训练中,我们分别取$\gamma_1=\{0.3,0.5,0.7,0.9\}$,并对识别结果进行了比较.在取某一$\gamma_1$ 值时,如果某测试恶意代码分类结果不能确定,这时再次调节γ_1 值,使γ_1 增大或者减小一个数值量Δ(实验中选取了$\Delta=0.2$)后再进行比较,直至确定该恶意代码类别.在提取系统调用流图时,也是只考虑了恶意代码的系统调用流图,其方法与调用流图相似,并且它依赖于参数γ_2.在训练恶意代码系统调用图过程后确定恶意代码类别,调节并选取合适的γ_2 值,直至其类别确定.设 $threshold_1$ 和 $threshold_2$ 分别为函数调用流图和系统调用流图的阈值,则当

$$\max_k\{score_{call_graph}(G_{new},G_k)\}<threshold_1,\ \max_k\{score_{sys_call}(G_{new},G_k)\}<threshold_2$$

分别满足时,该代码被识别为正常代码.

4.2 实验结果

表 1 给出了函数调用流图特征检测中,不同参数设置($threshold_1,\gamma_1$)的检测结果.表 2 给出了系统调用图特征检测中,不同参数设置($threshold_2,\gamma_2$)的检测结果.表 3 对不同特征的检测结果和 Boosting 算法的检测结果进行了比较,*call graph 参数选取为(0.5,4.3),system call 参数选取为(0.6,2.9).表 4 给出了不同特征检测算法的效率(在数据机上的所有样本训练时间和样本测试时间)比较.表 5 给出了在 VX Heavens 测试样本集合上,集成方法与两个商业杀毒软件(诺顿、瑞星)之间的比较结果.其中,Norton Antivirus 病毒库版本为 i32.exe(2009.2.20),Rav2009

© 中国科学院软件研究所 http://www.jos.org.cn

病毒库版本是 21.10.30.00.

Table 1 Detection result based on function call graph in different parameters ($threshold_1,\gamma_1$)

表 1 基于函数调用流图的检测结果($threshold_1,\gamma_1$)

Algorithm parameters ($threshold_1$, γ_1)/Metric	*ACC*	*FP*1	*FP*2	*NP*
0.3, 3.2	0.640	0.210	0.095	0.190
0.5, 4.3	0.670	0.155	0.090	0.215
0.7, 5.8	0.620	0.215	0.115	0.255
0.9, 6.5	0.580	0.315	0.140	0.205

Table 2 Detection result based on system call graph in different parameters ($threshold_2,\gamma_2$)

表 2 基于系统调用流图的检测结果($threshold_2,\gamma_2$)

Algorithm parameters ($threshold_2$, γ_2)/Metric	*ACC*	*FP*1	*FP*2	*NP*
0.2, 2.1	0.800	0.125	0.015	0.085
0.4, 2.6	0.790	0.115	0.020	0.125
0.6, 2.9	0.825	0.070	0.015	0.140
0.8, 3.3	0.750	0.255	0.035	0.125

Table 3 Detection result comparison based on different algorithms

表 3 不同特征检测算法的比较结果

Algorithm/Metric	*ACC*	*FP*1	*FP*2	*NP*
n-perm	0.730	0.135	0.050	0.155
*Call graph	0.670	0.155	0.090	0.215
*System call graph	0.825	0.070	0.015	0.140
Ensemble method	0.845	0.045	0.010	0.035

Table 4 Efficiency comparison of different detection algorithms (training time, testing time)

表 4 不同特征检测算法效率比较(training time, testing time)

Algorithm/Calculation cost	Training time (s)	Testing time (s)
n-perm	1.234	0.345
Call graph	3.465	0.278
System call graph	5.213	0.306
Ensemble method	9.386	0.457

Table 5 Detection result comparison with commercial anti-virus software

表 5 与商业杀毒软件之间的比较结果

Software/Metric	*ACC*	*FP*2	*NP*
Boosting	0.845	0.010	0.035
Symantec Norton antivirus	0.645	0.010	0.360
Rav2009	0.720	0.005	0.450

4.3 分析和不足

分析实验中产生漏报的样例是由于迷惑变种后的代码与同族的恶意代码在代码操作符序列或者系统调用序列中差别较大,从而被误分为正常代码;而实验中产生的误报的样例是由于一些正常代码行为与恶意代码样本有较多的相似性.例如实验中一组测试数据恰好为加密代码,而训练数据是简单语义等价变换后的代码(非加密),这时,从 *n*-perm 和 call graph 训练得到的分类器都不能够正确识别;而只有系统调用序列可以正确识别,除了增加一个系统调用序列外仍然保持了原来的其他系统调用序列.通过对实验样本进行分析和特征提取我们发现,大多数迷惑变种病毒只是改变了部分操作序列或者加密迷惑了恶意代码文件中的部分程序,而不是全部改变,这也为我们进行特征提取提供了可能.对于寄存器替换、插入 NOP 指令这类简单的代码迷惑技术不改变代码的结构特征,在分析中,call-graph 和 system call 一般较少发生改变.另外,由于此类变化通常作用于代码的局部,*n*-perm 分析有时也能获得较高的准确率.深入分析检测失败的原因,从本质上讲是由于迷惑变种前后的代码行为和代码序列发生了较大改变,从而导致分类器特征学习发生偏差.通过分析表 1、表 2 可以发现,误报率 *FP*2 随着参数组($threshold_1,\gamma_1$),($threshold_2,\gamma_2$)的组合而不断变化,由于阈值 $threshold_1$ 与 γ_1,$threshold_2$ 与 γ_2 分别正相关;我们同时增加两者数值发现,误报率先变小又变大,漏报率先变大又变小,而检测率变化分别是先变大再变小和

© 中国科学院软件研究所 http://www.jos.org.cn

先变小再变大再变小,趋势不是那么直观地完全按照比例变化.在步进参数调整过程中,我们最终选取了使误报率达到最小的那一组参数作为集成方法的分类器使用,同时有较高的准确率.由表 3 可以得到,集成方法的准确率优于单个分类器,这也验证了第 2.2 节的结果(满足 $b_1+b_2>b_3$);并且在该测试数据集上误报率和漏报率都有较大程度的下降,但是下降的比例并不能直观地得到,依赖于不同分类器的性能以及测试和训练数据集本身的特性.集成方法的误报率 $FP2<2\%$,还是可以接受的,但还是需要进一步降低误报率,以便在实际中可以使用.表 4 中,运行时间是数据集合上所有数据构造分类器的时间总和,还是比较令人满意.由于训练时间包含了对可执行文件静态分析,其中反汇编操作就花费了较多的时间,单单用于构建分类器的时间并不多.本质上是由于该方法在静态分析层次上对恶意代码进行分析,从而相比于动态分析(虚拟执行)而言,系统开销较小,训练和测试时间较快;不需要执行恶意代码及其变种程序.表 5 中,通过与现有杀毒软件的比较可以看出,该方法在检测变种迷惑恶意代码方面的准确率有较大提高,并且漏报率也降低,能够检测出一些商业软件(如诺顿,瑞星)所检测不出的变形病毒.

不足之处是,该算法依赖于训练样本所形成恶意代码特征库,能对同类恶意代码变形代码进行分类识别,但对新出现恶意代码检测效果不能保证;同时,对于迷惑反汇编的恶意代码无能为力.对过于复杂的恶迷惑变化和差别情况较大的变换,其特征变化较多(例如系统调用序列有较多增加,在保持语义的同时进行了大量的代码等价替换),从而导致分类器对恶意代码错误分类或者漏判.需要进行更深层次的语义分析或者程序等价性和相似性分析来提取最能够反映恶意代码特征的代码序列或行为序列,从而进一步提升迷惑恶意代码检测准确率.

5 相关工作比较

对于迷惑恶意代码的静态检测主要有两种方式:一是基于语义的恶意代码检测技术,利用迷惑代码语义变换前后的等价性和相似性,检测具有一定特征的恶意代码.例如 Jha[4]提出一种基于语义模板匹配的迷惑恶意代码检测算法,Gerald[13]提出一种利用上下文无关语法的语义检测算法,Debray[16]还给出了基于模板匹配的语义检测理论框架与有效性证明,Sung[20]利用控制流,数据流等语义信息对恶意代码进行检测;二是基于统计机器学习的迷惑恶意代码,对恶意代码特征提取后识别并进行检测,大多是基于 n-gram 算法[17].例如 Kolter[7]利用字符和字节序列信息对恶意代码和正常代码进行区分,Eskin[8]通过对已有恶意代码字节频度学习归纳后对未知恶意代码进行检测.现有的工作大多是区分正常和恶意代码,鲜有工作专门针对于恶意代码家族进行检测.本文从统计特性出发并结合代码语义特性(函数调用流图和系统调用流图),对家族恶意代码进行检测,实现了相对较好的效果.

6 总结和展望

由于在理论和实际中判定迷惑恶意代码在语义上的等价性是困难的,本文提出了一种提升迷惑恶意代码多维特征的恶意代码相似性比较框架和算法.通过集成方法,有效提升了恶意代码检测的准确率;并且相比于动态分析(虚拟执行)而言,系统开销较小,训练和测试时间较快,不失为一种有效的检测方法.下一步将结合提取更深层次的代码语义特征,对迷惑恶意代码进行分析和检测,进一步改进识别率和算法效率.

References:

[1] http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiii_04-2008.en-us.pdf

[2] Idika N, Mathur AP. Survey of malware detection techniques. Technical Report, Department of Computer Science, Purdue University, 2007.

[3] Christodorescu M, Jha S. Static analysis of executables to detect malicious patterns. Technical Report, Computer Science Department, University of Wisconsin, 2003.

[4] Christodorescu M, Jha S, Seshia S, Song D, Bryant R. Semantics-Aware malware detection. In: Proc. of the 2005 IEEE Symp. on Security and Privacy. 2005. 32−46. http://www.cs.berkeley.edu/~dawnsong/papers/semantic-aware.pdf [doi: 10.1109/SP.2005.20]

© 中国科学院软件研究所 http://www.jos.org.cn

[5] Milenkovic M, Milenkovic A, Jovanov E. Using instruction block signatures to counter code injection attacks. ACM SIGARCH Computer Architecture News, 2005,33:108−117. [doi: 10.1145/1055626.1055641]

[6] Varney D. Adequacy of checksum algorithms for computer virus detection. ACM SIGSMALL/PC Notes, 1991,17(1):27−29. [doi: 10.1145/122459.122462]

[7] Kolter JZ, Maloof MA. Learning to detect malicious executables in the wild. In: Proc. of the 10th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD 2004). New York: ACM Press, 2004. 470−478. [doi: 10.1145/1014052.1014105]

[8] Schultz MG, Eskin E, Zadok E, Saolfo SJ. Data mining methods for detection of new malicious executables. In: Proc. of the IEEE Symp. on Security and Privacy. 2001. 38−49. [doi: 10.1109/SECPRI.2001.924286]

[9] Lin PC, Lin YD, Lai YC, Lee TH. Using string matching for deep packet inspection. Computer, 2008,41(4):23−28. [doi: 10.1109/MC.2008.138]

[10] Kreibich C, Crowcroft J. Honeycomb—Creating intrustion detection signatures using honeypots. In: Proc. of the 2nd Workshop on Hot Topics in Network. 2003. http://www.icir.org/christian/publications/honeycomb-hotnetsII.pdf [doi: 10.1145/972374.972384]

[11] Jiang X, Xu D. Collapsar: A VM-based architecture for network attack detention center. In: Proc. of the 2004 USENIX Security Symp. 2004. http://www.cs.purdue.edu/research/technical_reports/2006/TR%2006-001.pdf

[12] Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection. In: Proc. of the 2003 Network and Distributed System Security Symp. (NDSS). 2003. http://suif.stanford.edu/papers/vmi-ndss03.pdf [doi: 10.1002/bltj.v12:3]

[13] Thompson GR, Flynn LA. Polymorphic malware detection and identification via context-free grammar homomorphism. Bell Labs Technical Journal, 2007,12(3):139−147.

[14] LURHQ Threat Intelligence Group. Sobig.a and the spam you received today. Technical Report, LURHQ, 2003. http://www.lurhq.com/sobig.html/

[15] LURHQ Threat Intelligence Group. Sobig.e—Evolution of the worm. Technical Report, LURHQ, 2003. http://www.lurhq.com/sobig-e.html/

[16] Preda MD, Christodorescu M, Jha S, Debray S. A semantics-based approach to malware detection. In: Proc. of the POPL 2007. 2007. 377−388 [doi: 10.1145/1190216.1190270]

[17] Li W, Wang K, Stolfo S, Herzog B. Fileprints: Identifying file types by *n*-gram analysis. In: Proc. of the 6th IEEE Information Assurance Workshop. 2005. 64−71 [doi: 10.1109/IAW.2005.1495935]

[18] Muchnick SS. Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers, 1997.

[19] Tan XB, Wang WP, Xi HS, Yin BQ. Anomaly detection based on SVM. Journal of University of Science and Technology of China, 2003,33(5):599−605 (in Chinese with English abstract).

[20] Sung A, Xu J, Chavez P, Mukkamala S. Static analyzer of vicious executables (save). In: Proc. of the 20th Annual Computer Security Applications Conf. (ACSAC 2004). 2004. 326−334. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1377239 [doi: 10.1109/CSAC.2004.37]

[21] Kuncheva LI. Combining Pattern Classifiers: Methods and Algorithms. Wiley-Interscience, 2004. 123−133.

[22] Duin R. The combining classifier: To train or not to train? In: Proc. of the Int'l Conf. on Pattern Recognition (ICPR). 2002. http://www.computer.org/portal/web/csdl/doi/10.1109/ICPR.2002.1048415 [doi: 10.1109/ICPR.2002.1048415]

[23] IDA pro. http://www.datarescue.com/

[24] System call sequence. http://www.cs.unm.edu/~immsec/systemcalls.htm

[25] Lam LC, Li W, Chiueh TC. Accurate and automated system call policy-based intrusion prevention. In: Proc. of the Int'l Conf. on Dependable Systems and Networks (DSN 2006). 2006. 413−424. http://www.ecsl.cs.sunysb.edu/tr/TR193.pdf [doi: 10.1109/DSN.2006.10]

附中文参考文献:

[19] 谭小彬,奚宏生,王卫平,殷保群.基于支持向量机的异常检测.中国科学技术大学学报,2003,33(5):599−605.

© 中国科学院软件研究所 http://www.jos.org.cn

附录 1:*n*-perm 算法抗攻击性分析

分析基于 *n*-perm 算法的特征提取算法在受到攻击的情况下对分类结果的影响.假定攻击者已知所选取的算法流程与特征提取流程,那么攻击者产生一系列的迷惑代码,从而使恶意代码运算符的分布发生较大的改变,偏离正常的聚类中心,即产生大量的噪声数据干扰特征提取的过程.

设所提取的特征是一个 n 维向量 x,该向量由 $x_1,x_2,\ldots,x_k$,共 k 个迷惑代码的特征分析得来.设$\phi(x_i)$为表征向量 x_i 映射后的特征值,取 $x=\frac{1}{n}\sum_{i=1}^{k}\phi(x_k)$,那么聚类半径 $R=\max_{1\leqslant i\leqslant k}\{\|\phi(x_i)-x\|\}$.

设产生攻击点使聚类圆心发生改变,定义偏离聚类半径中的点为异常点(outlier)在产生时满足$\|y-x\|\geqslant R$.

那么在攻击过程中,可以不断产生某个代码的迷惑版本 $x_1,x_2,\ldots,x_m$,从而使得聚类的圆心不断发生偏移.每次产生的攻击点为了能够起到作用,要在原来聚类的基础上产生异常点,从而使聚类中心不断发生改变.

用 t 记录迷惑恶意代码版本产生次数,此过程迭代 T 次($1\leqslant t\leqslant T$),每次产生的攻击点个数为 n_t.

第 t 次迭代前,特征记为 $\overline{x_{t-1}}$,那么第 t 次产生 n_t 个攻击点后,分别记为 $x_1^t,x_2^t,\ldots,x_{nt}^t$;有效攻击点为$\delta_t$个,分别记为 $x_1''^t,x_2''^t,\ldots,x_{\delta t}''^t\in\{x_1^t,x_2^t,\ldots,x_{nt}^t\}$,其特征量改变为 $\overline{x_t}=\frac{\left(\sum_{i=1}^{t-1}n_i\right)\overline{x_{t-1}}+\sum_{j=1}^{n_t}x_j^t}{\sum_{i=1}^{t}n_i}=\overline{x_{t-1}}+\frac{\sum_{j=1}^{n_t}(x_j^t-\overline{x_{t-1}})}{\sum_{i=1}^{t}n_i}$.

对于有效攻击点 $x_j^t(1\leqslant j\leqslant n_t)$,满足 $x_j^t-\overline{x_{t-1}}\geqslant R$,每次攻击点对生成特征产生改变.

无效攻击点 $x\in\{x_1^t,x_2^t,\ldots,x_{nt}^t\},x\notin\{x_1''^t,x_2''^t,\ldots,x_{\delta t}''^t\}$ 且 $x_j^t-\overline{x_{t-1}}<R$.

易知,当 $n_t=\delta_t$ 时攻击效果最好(不妨记 $x_j^t=\overline{x_{t-1}}+R$,以下都假设攻击序列在初始时半径为 0).

定理 1. 在攻击序列中,记 $P_{t-1}=\sum_{i=1}^{t-1}\sum_{j=1}^{n_i}x_j^i, N_{t-1}=\sum_{i=1}^{t-1}n_i, p_t=\sum_{i=1}^{n_t}x_i^t$,那么当满足 $\frac{p_t}{P_{t-1}}-\frac{n_t}{N_{t-1}}>0$ 时,攻击后半径不断增大.

证明:攻击后半径不断增大,即 $\overline{x_t}>\overline{x_{t-1}}$ 成立.

由 $\overline{x_t}=\frac{\sum_{i=1}^{t}\sum_{j=1}^{n_i}x_j^i}{\sum_{i=1}^{t}n_i}$,代入 $\overline{x_t}>\overline{x_{t-1}}$ 中即得 $\overline{x_t}-\overline{x_{t-1}}=\frac{N_{t-1}(P_{t-1}+p_t)-P_{t-1}(N_{t-1}+n_t)}{N_{t-1}(N_{t-1}+n_t)}$,故得 $\frac{p_t}{P_{t-1}}-\frac{n_t}{N_{t-1}}>0$,从而攻击后的半径不断增大. □

可见,只要按照一定的策略和算法产生攻击点,能够使特征提取中的聚类中心不断发生改变,从而达到敌手攻击的目的,使后端检测效果大受影响.

附录 2:系统调用流图中相似性度量

使用谓词 *same* 描述节点 v_a 与节点 v_b 相同.*same*(v_a,v_b)=*true* 等价于 v_a.*systemcall*=v_b.*systemcall* 并且 v_a.*prenum*=v_b.*prenum* 和 v_a.*sucnum*=v_b.*sucnum* 以及对于$\forall i$,都$\exists j$ 使得 v_a.*pre*(i).*systemcall*=v_b.*pre*(j).*systemcall* 并且同时满足对于$\forall i$,都$\exists j$ 使得 v_a.*suc*(i).*systemcall*=v_b.*suc*(j).*systemcall*.

谓词 *simi* 描述节点 v_a 和 v_b 相似.*simi*(v_a,v_b)=*true* 等价于 v_a.*systemcall*=v_b.*systemcall* 并且 v_a.*prenum*!=v_b.*prenum* 或者 v_a.*sucnum*!=v_b.*sucnum* 成立,并且同时$\exists i\exists j$,使得 v_a.*pre*(i).*systemcall*!=v_b.*pre*(j).*systemcall* 成立或者同时$\exists i\exists j$ 使得 v_a.*suc*(i).*systemcall*!=v_b.*suc*(j).*systemcall* 成立.

使用谓词 *diff* 描述节点 v_a 与节点 v_b 不同.*diff*(v_a,v_b)=*true* 等价于 v_a.*systemcall*!=v_b.*systemcall*.根据函数中语句的跳转关系,将由 *jmp* 和 *call* 标记的分支指令分为以下 4 类.其中,*scall* 用于标记系统调用,*pcall* 用于标记过程间调用,*jmp* 和 *cjmp* 分别对应无条件和条件跳转.

© 中国科学院软件研究所 http://www.jos.org.cn

```
输入:可执行文件的反汇编代码(PE/Elf 格式);
输出:系统调用流图 G=(V,E).
1  V←0,E←0,cur←0                                         //顶点和边为空,当前处理的系统调用为空
2  I←GetFirstInstuction                                   //获得第 1 条指令
3  while I.equal(lastInstruction)=false                   //不是最后一条指令
4      Opcode←I.GetOpcode();                              //获得该代码处的操作码
5      Pr←I.GetParameter();                               //获得该代码的操作数
6      if Type(Opcode).equal(scall)                       //是系统调用
7          if cur.equal(null)
8              cur←Pr                                     //记录当前系统调用
9          else
10             next←Pr, E.addedge(cur→next), cur←next     //增加系统调用边
11         endif
12     elseif Type(Opcode).equal(JMP)                     //对应无条件转移
13         next←GetFirstSystemCall(Pr);                   //获得跳转处的第 1 条系统调用
14         E.addedge(cur→next),   cur←null                //增加系统调用边
15     elseif Type(Opcode).equal(CJMP)                    //对应条件转移
16         next←GetFirstSystemcall(Pr), E.addedge(cur→next)   //获得跳转处的第 1 条系统调用,增加系统调用边
17     elseif Type(Opcode).equal(Pcall)                   //对应过程间跳转
18         next←GetFirstSystemcall(Pr)                    //获得子过程处的第 1 条系统调用
19         E.addedge(cur→next), cur←GetEndSystemcall(Pr)  //增加系统调用边,获得子过程处的最后一条系统调用
20     end if
21     r←GetNextInstruction()                             //取下一条指令
22 end while
```

孔德光(1983－),男,山东邹城人,博士生,主要研究领域为统计机器学习在信息安全中的应用.

宫涛(1982－),男,博士生,主要研究领域为信息安全.

谭小彬(1973－),男,博士,副教授,主要研究领域为信息安全.

帅建梅(1961－),女,高级工程师,主要研究领域为网络安全与信息安全.

奚宏生(1950－),男,教授,博士生导师,主要研究领域为离散事件动态系统,网络性能建模与分析,信息安全.

© 中国科学院软件研究所 http://www.jos.org.cn