基于谓词时序逻辑的恶意代码行为描述及检测

金 然 范荣荣 顾小琪
(江南计算技术研究所 无锡 214083)

摘 要 基于行为的判别已成为恶意代码检测技术研究的主流方向,现有方法容易受到拟态攻击或影子攻击的影响。针对这些问题,提出了一种全新的使用谓词时序逻辑描述恶意代码行为的方法,该方法能够同时刻画一组函数调用之间的逻辑组合、时序、参数依赖和主客体关联等关系,因此能更准确细致地描述恶意代码行为。在此基础上,提出了相应的恶意行为检测算法,通过实例测试验证了该方法的有效性。

关键词 行为,逻辑,恶意代码
中图法分类号 TP393.08 **文献标识码** A

Predicate Temporal Logic Based Description and Detection of Malware Behavior

JIN Ran FAN Rong-rong GU Xiao-qi
(Jiangnan Computing Technology Institue, Wuxi 214083, China)

Abstract The behavior based security has become main-stream in the research of malware detection techniques. Although there have been some behavior based malware detection methods introduced in public papers, they are prone to suffer from mimicry attack or shadow attack. Towards these problems, a novel technique using predicate temporal logic to describe malware behavior was proposed in this paper. A variety of relations among system function calls, such as logic combination, precedence, parameter decencies and subject-object associations, can be depicted by one logic formula, therefore our method can describe malware behavior more subtlety and accurately. An algorithm of detecting malware behavior based on the logic was given and its feasibility was justified through real example test.

Keywords Behavior, Logic, Malware

1 引言

恶意代码检测技术一直是信息安全领域的重点研究内容。由于传统的基于特征码的识别方法很难有效应对快速增长的恶意代码变种以及未知特征的恶意代码,使得基于行为的检测方式成为了恶意代码检测技术研究的主流[1,2]。

代码运行时与系统进行交互,对系统上的文件、注册表、进程、网络等各种对象进行操作,需要调用系统提供的各种函数,因此根据代码的系统函数调用情况来研究其行为是绝大部分相关研究采用的方法[3]。

目前在基于行为检测恶意代码的研究领域内,一些方法通过分析正常代码与恶意代码在系统函数调用先后关系或调用参数上各自的统计特性来建立分类器,然后在此基础上识别恶意代码[4-6]。不过这些方法容易受到拟态攻击的影响[7],即恶意代码可通过插入额外系统函数调用或改变函数调用方式等手段来模拟正常代码所具有的函数调用特性。与之不同的是,其它一些方法利用有向图或树来描述恶意代码在表现其恶意行为时所调用的一组具有参数依赖关系的关键系统函数,以此来刻画恶意代码行为并将其作为检测恶意代码的基础[8,9,12]。这些方法不易受到拟态攻击的影响,不过由于它们所采用的方法不能刻画系统函数调用主客体之间的关联关系,因此容易受到影子攻击的影响[10],即恶意代码可以通过影子程序来调用实施其恶意行为的系统函数,例如在合法进程中注入线程,然后由线程来调用部分关键函数[11],或者直接利用系统自带的合法程序来达到其恶意目的,如创建服务、修改注册表等。

针对上述问题,本文提出了一种全新的基于谓词时序逻辑的恶意代码行为描述及检测方法。该方法使用逻辑公式来描述恶意代码在表现其恶意行为时所调用的一组关键系统函数,能同时刻画这些函数调用之间的逻辑组合、时序、参数依赖和主客体关联等关系,因此能更准确细致地描述恶意代码行为。此外,与有向图相比,逻辑公式更易于表述和存储管理。在此基础上,提出了检测代码恶意行为的算法,并介绍了其相关实例测试。

2 恶意行为描述

为了将恶意代码行为表示为一个逻辑公式,首先引入如下谓词,用以描述构成代码行为的原子操作:系统函数调用。

到稿日期:2012-11-12 返修日期:2013-03-18

金 然(1979-),男,工程师,主要研究方向为信息安全,E-mail:loolle@126.com;范荣荣(1979-),女,工程师,主要研究方向为信息安全;顾小琪(1983-),女,工程师,主要研究方向为信息安全。

$predicate ::= syscallname(sub, \{obj_1, \cdots, obj_n\}, \{adv_1, \cdots, adv_m\})$

其中，*syscallname* 表示所调用函数的名称，*sub* 表示调用主体，$\{obj_1, \cdots, obj_n\}$表示函数调用所处理的对象客体，当 $n=0$ 时表示无处理对象，$\{adv_1, \cdots, adv_m\}$表示对函数调用的限定，包括方法、类型等，当 $m=0$ 时，表示无限定。例如，*NtDeviceIoControlFile*(*pname*:"a. exe", {*dip*:"211. 158. 2. 45"}, {*type*:AFD_CONNECT})表示进程 a. exe 调用 *NtDeviceIoControlFile* 访问地址 211. 158. 2. 45，方式为 AFD_CONNECT(即连接)；而 *NtWriteFile*(*pid*:1196, {*fname*:"c:\temp. exe"})表示 *id* 号为 1196 的进程调用 *NtWriteFile* 对文件 c:\temp. exe 进行处理。

在上述谓词基础上，通过引入"与""或""非"逻辑关系符，以及表示以前操作的时序模态词，同时在 *predicate* 中引入变量，就可以构造出能够描述一组具有多种相互关系(包括逻辑组合关系、时序关系、参数依赖和主客体关联关系等)函数调用操作的逻辑公式，进而可描述复杂的恶意代码行为。下面给出逻辑公式的语法描述。

$\varphi ::= predicate \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond\varphi$

其中，*predicate* 可能包含主体变量、客体变量、限定变量；$\Diamond$ 属于时序模态词，表示"过去一段时间里有"。以下给出了一种利用 Office Word 漏洞植入木马的恶意行为描述实例。

φ_{word}:

$\Diamond$(($\Diamond$NtWriteFile(pname:"winword. exe", {fname}) $\wedge$
NtCreateProcessEx(pname:"winword. exe", {fname}))($\wedge$
NtDeviceIoControlFile(fname, {dip}, {type: AFD_CONNECT})

上述逻辑公式表述的是 winword. exe 创建了一个会连接某地址的进程，并且该进程的映象文件是在调用 *NtCreateProcess* 之前的某个时间点由 winword. exe 生成的。

为了更准确地描述由上述语法构造的逻辑公式 φ 的含义，下面给出其语义形式定义。主要从 φ 在何种条件下为真，即一系列函数调用何时满足 φ(或包含 φ 所表达行为)的角度来进行。

首先说明一些符号。设 $H=e_1e_2\cdots e_N$ 为目标代码在运行期间按时间先后顺序所观察到的系统中各进程的函数调用序列，$(H,i)(1\leqslant i\leqslant N)$表示 H 中前 i 个函数调用，即$(H,i)=e_1e_2\cdots e_i$。令 *e. name* 表示函数调用名，*e. para* 表示所调函数 *e* 的参数集，*Anc*(*e. sub*)表示由 *e* 函数调用主体的祖先进/线程构成的集合(包括 *e. sub* 自身)。由于 φ 可能含有变量，因此在讨论其真值时需引入解释 I:*VAR*→*VAL*。I 将 *VAR* 中的任一变量映射为 *VAL* 中的某一常量，令 $I(\varphi)$表示将 φ 中变量替换为由 I 所映射到的相应常量而得到的新公式，则 $I(\varphi)$一定不包含变量。

基于以上符号，下面用结构归纳法给出了 φ 的语义定义。

(1)$(H,i)\vDash^I predicate$ 当且仅当 $e_i.name=predicate.syscallname$ and $I(predicate).sub\in Anc(e_i.sub)$ and $e_i.para\supseteq I(predicate).\{obj_1,\cdots,obj_n\}\cup I(predicate).\{adv_1,\cdots,adv_m\}$

(2)$(H,i)\vDash^I\neg\varphi$ 当且仅当$(H,i)\nvDash^I\varphi$

(3)$(H,i)\vDash^I\varphi_1\wedge\varphi_2$ 当且仅当$(H,i)\vDash^I\varphi_1$ and $(H,i)\vDash^I\varphi_2$

(4)$(H,i)\vDash^I\varphi_1\vee\varphi_2$ 当且仅当$(H,i)\vDash^I\varphi_1$ or $(H,i)\vDash^I\varphi_2$

(5)$(H,i)\vDash^I\Diamond\varphi$ 当且仅当 $\exists j.(1\leqslant j\leqslant i$ and $(H,j)\vDash^I\varphi)$

上述定义中(1)表述的是，函数调用序列在解释 I 下满足一个谓词当且仅当该谓词的 *syscallname* 与当前函数调用名相同，并且经过 I 对谓词中变量的替换，其客体对象和调用限定都出现在了当前函数调用的参数集中；同时谓词中指定的主体要么是当前函数调用的主体，要么是其祖先进/线程(如此设定可反映出恶意代码利用代理进/线程完成所需功能的行为)。

根据定义中第(5)项的表述，它还可有如下的等价表述方式：

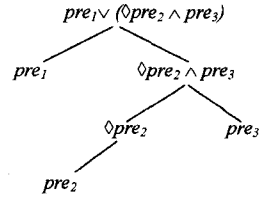
(5′)$(H,i)\vDash^I\Diamond\varphi$ 当且仅当$(H,i)\vDash^I\varphi$ or $(H,i-1)\vDash^I\Diamond\varphi$

3 恶意行为检测

根据上节给出的 φ 的语义定义可看出，检测目标代码是否具有 φ 所描述的恶意行为，需要通过观测目标代码运行时系统中的函数调用序列 H 来判断能否找到一个解释 I，并且在该解释下，能否使得 $H\vDash^I\varphi$。语义定义是通过将公式分解为子公式((5′)还考察了过去的公式真值)，利用结构归纳进行的。根据这样的思路，设计了一种动态规划的恶意行为检测算法，主要流程是先获取 φ 的子公式集，然后在顺序扫描 H 序列中的函数调用时，从谓词子公式开始，构造解释并判断真值，通过层层递推的方式，最终判断 φ 的真值情况。在给出算法描述前，先介绍两种操作。

①获取 φ 的子公式集

根据 φ 的语法，任何 φ 都可对应于一棵语法树，则 φ 的子公式就是该语法树的任一节点。例如假设公式 $pre_1\vee(\Diamond pre_2\wedge pre_3)$ 中的 pre_1，pre_2，pre_3 是谓词，则它对应的语法树为：



通过遍历这棵语法树就能获得 φ 的子公式集：$\{pre_1\vee(\Diamond pre_2\wedge pre_3), pre_1, \Diamond pre_2\wedge pre_3, \Diamond pre_2, pre_2, pre_3\}$。

②构造解释 I

根据 φ 的不同子公式形态，以下将从 5 个方面来说明如何构造解释 I，使得$(H,i)\vDash^I\varphi$。由于 φ 的子公式 φ' 不一定含有 φ 中的变量，即有 $\varphi.VAR\supseteq\varphi'.VAR$，为了计算方便，规定与 φ 各子公式相关的解释 I'，其定义域统一为 $\varphi.VAR$，同时在常量域中增加一个特殊常量"#"，对 $v\in\varphi.VAR-\varphi'.VAR, I'(v)=\#$。

(i)$\varphi=predicate$

当 H 中第 i 个函数调用 $e_i.name = predicate.syscallname$ 时，可通过将 $predicate$ 中各变量映射为 e_i 调用主体及其参数中的对应值来构造一个 I，此时若 $I(predicate).sub \in Anc(e_i.sub)$ 并且 $e_i.para \supseteq I(predicate).\{obj_1,\cdots,obj_n\} \cup I(predicate).\{adv_1,\cdots,adv_m\}$，则根据语义定义(1)可知，该解释可使得 $(H,i) \models^I \varphi$。

(ii) $\varphi = \neg\varphi'$

假设已构造出 I'，使得 $(H,i) \models^{I'} \varphi'$。根据语义定义(2)可知，此时只需令 $I = I'$，即可使 $(H,i) \models^I \varphi$。

(iii) $\varphi = \varphi_1 \wedge \varphi_2$

首先引入对解释 I' 和 I'' 的运算 $I' \Theta I''$，运算规则如下：

(a)若 $\exists v \in \varphi.VAR$，$v$ 为客体或限定变量，使得 $I'(v) \neq \#$、$I''(v) \neq \#$、$I'(v) \neq I''(v)$，或 v 为主体变量，使得 $I'(v) \neq \#$、$I''(v) \neq \#$ 且 $I'(v) \notin Anc(I''(v))$ 且 $I''(v) \notin Anc(I'(v))$，则 $I' \Theta I'' = \text{null}$，且表示结果为空，否则按(b)输出一个新解释 I；

(b) $v \in \varphi.VAR$，若 v 为客体或限定变量，则

$$(I'\Theta I'')(v) = \begin{cases} \#, & I'(v) = I''(v) = \# \\ I'(v), & I''(v) = \# \text{ and } I'(v) \neq \# \\ I''(v), & I'(v) = \# \text{ and } I''(v) \neq \# \\ I'(v), & I'(v) = I''(v) \neq \# \end{cases}$$

若 v 为主体变量，则

$$(I'\Theta I'')(v) = \begin{cases} \#, & I'(v) = I''(v) = \# \\ I'(v), & I''(v) = \# \text{ and } I'(v) \neq \# \\ I''(v), & I'(v) = \# \text{ and } I''(v) \neq \# \\ I'(v), & I'(v) \neq \# \text{ and } I''(v) \neq \# \\ & \text{and } I'(v) \in Anc(I''(v)) \\ I''(v), & I'(v) \neq \# \text{ and } I''(v) \neq \# \\ & \text{and } I''(v) \in Anc(I'(v)) \end{cases}$$

现假设已构造出 I' 和 I''，使得 $(H,i) \models^{I'} \varphi_1$ 且 $(H,i) \models^{I''} \varphi_2$。令 $I = I' \Theta I''$，若 $I \neq \text{null}$，则根据上述运算规则(b)以及语义定义(3)，新构造出的 I 可使得 $(H,i) \models^I \varphi$。

(iv) $\varphi = \varphi_1 \vee \varphi_2$

假设针对 φ_1 已构造出 I'，使得 $(H,i) \models^{I'} \varphi_1$，或者针对 φ_2 已构造出 I''，使得 $(H,i) \models^{I''} \varphi_2$。根据语义定义(4)可知，此时只需令 $I \in \{I', I''\}$，就可使 $(H,i) \models^I \varphi$。

(v) $\varphi = \Diamond\varphi'$

假设已构造出 I'，使得 $(H,i) \models^{I'} \varphi'$，或者也已构造出 I''，使得 $(H,i-1) \models^{I''} \varphi$。根据语义定义(5′)可知，此时只需令 $I \in \{I', I''\}$，就可使 $(H,i) \models^I \varphi$。

在上述①②两种操作基础上，图 1 给出了检测恶意代码行为的算法。算法中 B_{now}、B_{pre} 和 IS_{now}、IS_{pre} 分别用于记录在扫描当前函数调用以及前一函数调用时，各子公式的真值情况和能使子公式为真的解释集合，其索引号对应于子公式的序号，由于子公式 φ_0 就是公式 φ 本身，因此当算法执行到 $B_{now}[0]$ 为真时，则输出：目标代码具有 φ 所描述的行为。从算法可看出，该算法的最坏时间复杂度为 $O(N \times M)$，其中 N 为函数调用序列长度，M 为子公式的个数。

```
Detecting malware behaviour
Input: behaviou φ and func call seq H = e1…eN when executing object code
Output: does object code have behaviou φ?

1. get sub formula set {φ0, φ1, …, φM} of φ according to ①, φ0=φ;
2. allocate bool arrays Bpre[M]and Bnow[M] with initial value "false";
3. allocate interpretation set arrays ISpre[M]and ISnow[M];
4. for each function call e in H from e1 to eN do
5.   for each formula in {φ0, φ1, φ2, …, φM} do
6.     if(φi is predicate) then
7.       if(e.name = φi.syscallname) then
8.         construct Interpretation I for φi according to ②.(i);
9.         if(I != null) then
10.          Bnow[i] := true;
11.          ISnow[i] := {I};
12.    if(φi = ¬ φj) then
13.      Bnow[i] := ¬ Bnow[j];
14.      ISnow[i] := ISnow[j];
15.    if(φi = φj ∧ φk) then
16.      ISnow[i] := {Ij ⊙ Ik | Ij ∈ ISnow[j] and Ik ∈ ISnow[k]};
17.      if(ISnow[i] != φ)
18.        Bnow[i] := Bnow[j] ∧ Bnow[k];
19.    if(φi = φj ∨ φk) then
20.      Bnow[i] := Bnow[j] ∨ Bnow[k];
21.      ISnow[i] := ISnow[j] ∪ ISnow[k];
22.    if(φi = ◇φj) then
23.      Bnow[i] := Bnow[j] ∨ Bpre[i];
24.      ISnow[i] := ISnow[j] ∪ ISpre[i];
25.  Bpre := Bnow;
26.  ISpre := ISnow;
27.  if(Bnow[0] = true)
28.    output "object code has behaviour φ";
29. output "object code does not have behaviour φ";
```

图 1 检测恶意代码算法

4 实例测试

由于真实系统环境下，应用层系统函数众多，对其进行监视效率较低，因此在内核中监视系统服务函数调用来检测代码行为是比较合适的选择。我们利用 SSDT 挂钩技术实现了对 Windows 内核服务函数的监视器，每个被监视函数的参数在其调用前后都进行了相应的处理和过滤，例如根据文件句柄获取对应的文件名等，这样能获得全面的关键参数信息，从而方便应用上述方法检测恶意代码行为。

通过对恶意代码行为的研究归纳，我们利用前述谓词时序逻辑总结定义了若干恶意代码行为，同时选择在现实环境中收集捕获到的恶意文档和可执行程序来测试检测算法。结果显示，凡是用谓词时序逻辑公式描述行为的恶意代码均能被检测出来，而没有描述行为的代码没有产生报警。限于篇幅，这里只以第 2 节给出的 φ_{word} 为例具体说明恶意代码行为检测算法。当双击打开目标 Word 文档 target.doc 后，监控得到如图 2 所示的函数调用序列。

采用前述检测算法，首先将 φ_{word} 分解为如下子公式，其变量有 $fname$ 和 dip。

φ_0：◇(◇ NtWriteFile(pname:"winword.exe",{fname}) ∧
NtCreateProcessEx(pname:"winword.exe",{fname})) ∧
NtDeviceIoControlFile(fname,{dip},{type:AFD_CONNECT})

φ_1：◇(◇ NtWriteFile(pname:"winword.exe",{fname}) ∧
NtCreateProcessEx(pname:"winword.exe",{fname}))

φ_2：◇ NtWriteFile(pname:"winword.exe",{fname}) ∧
NtCreateProcessEx(pname:"winword.exe",{fname})

φ_3：◇ NtWriteFile(pname:"winword.exe",{fname})

φ_4：NtWriteFile(pname:"winword.exe",{fname})

φ_5：NtCreateProcessEx(pname:"winword.exe",{fname})

φ_6：NtDeviceIoControlFile(fname,{dip},{type:AFD_CONNECT})

随后，当扫描到第 676 号函数调用时，对 φ_4，根据算法 6－11

行，可得将 *fname* 映射为“C:\DOCUME~1\Administrator\LOCALS~1\Temp\tt.exe”的解释，并使 $B_{new}[4]$=true；对 φ_3，根据 22－24 行，可得与 φ_4 相同解释，使 $B_{new}[3]$=true，此时，对其它子公式，其对应的 B_{new} 值都为 false。当扫描到第 690 号函数调用时，对 φ_5、φ_3、φ_2 和 φ_1，可得与前述相同的解释，使它们的 B_{new} 值都为 true。最后当扫描到第 1135 号函数调用时，对 φ_6，可得将 *fname* 映射为“C:\Program Files\Internet Explorer\iexplore.exe”、将 dip 映射为“192.168.0.23”的解释，使 $B_{new}[6]$=true。不过由于根据第 1064 号函数调用可知，此时的调用主体是被 tt.exe 所创建，因此最终根据算法 15－18 行，可得到将 *fname* 映射为“C:\DOCUME~1\Administrator\LOCALS~1\Temp\tt.exe”、将 dip 映射为“192.168.0.23”的解释，并使 $B_{new}[0]$=true。从而可判定目标文档 target.doc 打开后含有 φ_{word} 所描述的恶意行为。

#	pid	tid	process_name	nativeAPI	parameters
			.....		
383	1624	1836	explorer.exe	NtOpenProcess	pid:1624, fname:C:\WINDOWS\explorer.exe
384	1624	1204	explorer.exe	NtCreateFile	fhandle:1172, fname:C:\Program Files\Microsoft Office\OFFICE11\winword.exe
385	1624	1204	explorer.exe	NtCreateProcessEx	pid:1212, fname:C:\Program Files\Microsoft Office\OFFICE11\winword.exe
			.....		
674	672	968	winlogon.exe	NtCreateThread	pid:672, tid:212
675	1212	1876	winword.exe	NtCreateFile	fhandle:696, fname:C:\DOCUME～1\Administrator\LOCALS～1\Temp\tt.exe
676	1212	1876	winword.exe	NtWriteFile	fhandle:696, fname:C:\DOCUME～1\Administrator\LOCALS～1\Temp\tt.exe
			.....		
690	1212	1876	winwrod.exe	NtCreateProcessEx	pid:1296, fname:C:\DOCUME～1\Administrator\LOCALS～1\Temp\tt.exe
			.....		
1060	1296	1440	tt.exe	NtOpenProcess	pid:1424, fname:C:\Program Files\Internet Explorer\iexplore.exe
1061	648	1704	csrss.exe	NtCreateFile	fhandle:244, fname:C:\WINDOWS\WinSxS\Policies\x86_Policy.6.....
1062	648	1704	csrss.exe	NtCreateFile	fhandle:244, fname:C:\WINDOWS\WinSxS\Policies\x86_Microsoft.W.....
1063	648	1704	csrss.exe	NtCreateFile	fhandle:244, fname:C:\WINDOWS\WinSxS\Policies\x86_Microsoft.W.....
1064	1296	1440	tt.exe	NtCreateThread	pid:1424, tid:1576
			.....		
1135	1424	1576	iexplore.exe	NtDeviceIoControlFile	iocode:AFD_CONNECT, dip:192.168.0.23
			.....		

图 2　目标文件打开时观察到的系统服务函数调用

实际上，根据图 2 的函数调用序列可知，target.doc 被打开后会先释放 tt.exe 并创建其进程，该进程在 iexplorer.exe 中远程注入的线程将试图连接 192.168.0.23。尽管 tt.exe 并没有直接连接网络，而是利用一个远程代理线程进行操作，但使用本文方法依然能检测到它的恶意行为 φ_{word}，表明该方法具有抗影子攻击的特点。

结束语　本文通过引入谓词时序逻辑来描述恶意代码行为，不仅能刻画系统函数调用间的组合关系、时序关系、参数依赖关系，也能表达出函数调用间的主客体关联关系，从而可准确地在函数调用层次上描述复杂的恶意代码行为。在此基础上，本文给出了相应的恶意代码行为检测方法，通过实例测试验证了该方法的有效性。进一步丰富主客体之间的关系描述是本文今后的研究方向。

参 考 文 献

[1] Idike N, Mathur A P. A Survey of Malware Detection Techniques[R]. Technical Report. Purdue University, 2007

[2] Geer D. Behavior-Based Network Security Goes Mainstream[J]. Computer, 2006, 39(3): 14-17

[3] Canali D, Lanzi A, Balzarotti B, et al. A Quantitative Study of Accuracy in System Call-Based Malware Detection[C]// Proceedings of the the International Symposium on Software Testing and Analysis. 2012

[4] Sami A, Rahimi H, Yadegari B, et al. Malware Detection Based on Mining API Calls[C]// ACM Symposium on Applied Computing. 2010

[5] Alazab M, Venkatraman S, Watters P. Malware Detection Based on Structural and Behavioural Features of API Calls[C]// 1st International Cyber Resilience Conference. Edith Cowan University, Perth Western Australia, 2010

[6] 李鹏，王汝传，高德华. 基于模糊识别和支持向量机的联合 Rootkit 动态检测技术研究[J]. 电子学报，2012，40(1)：115-120

[7] Parampalli C, Sekar R, Johnson R. A Practical Mimicry Attack Against Powerful System-Call Monitors[C]// ACM Symposium on Information, Computer and Communications Security (AsiaCCS). Japan, 2008: 156-167

[8] Kolbitsch C, Comparetti P M, Kruegel C, et al. Effective and Efficient Malware Detection at the End Host[C]// Proceedings of 18th USENIX Security Symposium. 2009

[9] Martignoni L, Stinson E, Fredrikson M, et al. A Layered Architecture for Detecting Malicious Behaviors[C]// Proceedings of the 11th international Symposium on Recent Advances in intrusion Detection. 2008

[10] Ma W, Duan P, Liu S, et al. Shadow Attacks: Automatically Evading System-Call-Behavior Based Malware Detection Based Malware Detection[J]. Journal in Computer Virology, 2012, 8(1/2): 1-13

[11] Harbour N. Stealth Secrets of the Malware Ninjas[EB/OL]. https://www.blackhat.com/ presentations/bh-usa-07/Harbour/Presentation/bh-usa-07-harbour.pdf, 2012-09-20

[12] 杨彦，黄浩. 基于攻击树的木马监测方法[J]. 计算机工程与设计，2008，29(11)：2711-2714