基于粗糙集约简的进程系统调用序列异常检测方法研究*)

鲜 明 张义荣 肖顺平 王国玉
(国防科技大学电子科学与工程学院 长沙 410073)

摘 要 提出了一种基于粗糙集约简的系统调用序列异常检测方法,其基本思想是利用粗糙集约简来对第 k 个系统调用位置进行预测,把前 $k-1$ 个位置视为条件属性集,第 k 个位置视为决策属性,通过 Rough 集约简方法得到一组预测第 k 个系统调用位置的最小规则集,进而可用于对实际进程的异常检测。基于合成的 UNM sendmail 系统调用数据的实验结果表明,本文所提出的异常检测算法性能好于 Forrest 等人的 tide 方法,与 Wenke Lee 等人的数据挖掘算法检测精度相当。但在选择较大的阈值时,漏报率更低。

关键词 异常检测,系统调用序列,粗糙集,约简,不一致推理

Research on Anomaly Detection of System Call Sequences of Process Based on Rough Set Reduction

XIAN Ming ZHANG Yi-Rong XIAO Shun-Ping WANG Guo-Yu
(School of Electronic Science and Engineering, National Univ. of Defense Technology, Changsha 410073)

Abstract An anomaly detection technique of system call sequence based on rough set reduction is presented in this paper. Its fundamental idea is that rough set reduction is utilized to predict the kth position of process system call trail, i.e., the kth position is viewed as the decision attribute and the previous $(k-1)$ positions are viewed as conditional attributes. The method of rough set reduction gives a set of minimal rules of predicting the kth system call position, thus it can apply to anomaly detection. The experiments based on synthetical sendmail system call sequences from UNM show that the proposed anomaly detection algorithm in the paper is superior to tide and comparable to the data mining algorithm of Wenke Lee, et al. in detection precision, moreover, achieves a lower negative positive rate when selecting a slightly large threshold.

Keywords Anomaly detection, System call sequence, Rough set, Reduct, Inconsistent reasoning

1 引言

在 Unix 系统中,进程运行过程中会产生一系列系统调用。通过对这些系统调用的分析,可以发现进程的异常运行状态,进而判断出该系统是否受到攻击。1996 年 Forrest[1] 等人在研究基于人工免疫的入侵检测中,发现可以通过对进程正常运行时的执行轨迹建模来刻画进程的正常运行状态,她们提出了所谓的时延嵌入序列(tide)方法,通过列举出现在训练数据中唯一的、预先定义的长度为 k 的连续序列来构造进程的正常行为轮廓。Wenke Lee[2] 的实验表明,机器学习方法在这样的系统调用短序列检测中起着重要的作用。进一步,Warrender、Forrest[3] 等人发现进程运行不正常时系统调用短序列具有局域特征,因此可以通过研究给定长度的局部区域中长度为 k 的短序列与进程正常轮廓库中不匹配的短序列数目来检测攻击,称之为序列 tide 方法(stide)。stide 方法进一步扩展为带(频率)门限的 stide 方法,即 t-stide 方法,它考虑了短序列出现的频率,而将稀有序列从进程的正常轮廓库中忽略。A. Wespi、M. Dacier 和 H. Debar[4] 基于一种生物序列模式发现方法-Teiresias,提出了一种可变长度的短序列的正常模式建立方法。研究结果表明,该方法明显好于定长序列的检测方法。G. Tandon 和 P. K. Chan[5] 则把系统调用参数整合进异常检测的学习规则中,他们的 M-LERAD 异常检测方法在检测新颖的异常攻击方面更为有效。

目前,在利用进程运行时产生的系统调用序列做异常检测方面,主要有这样几种思路:一是对系统调用短序列进行枚举,从而建立起正常系统调用短序列轮廓库,如 tide、stide 方法等;一是采用基于频率的方法,即对于系统调用出现的频率建模,其基本思想是正常进程运行时产生的系统调用出现的频率应该是稳定的,因为绝大多数程序是静态的,系统调用通常在代码的固定位置发生。相应的实现方法有 Y. H. Liao 等[6] 的 k 近邻分类器、t-stide 方法等;一是基于数据挖掘的方法,以便从大量数据中找到序列调用的重要特征,典型的有 Wenke Lee 等[2,7] 的采用"RIPPLE"程序的挖掘方法;一是基于有限状态机的方法,如采用 HMM 的检测[3] 等;还有一种是结合调用参数的系统调用短序列异常检测方法,如 A-LERAD、M-LERAD、M*-LERAD 等[5]。

综合以上基于系统调用序列的异常检测研究思路,其共同点是需要在已知长度为 k 的系统调用序列的前 $k-1$ 个系统调用的情况下,对第 k 个系统调用或状态进行预测,并把预测结果与实际结果进行比较,通过设置一定的异常度阈值来判定异常。根据这种研究思路,本文提出了一种基于粗糙集约简的系统调用序列异常检测方法,它的基本思想是利用粗糙集约简来对第 k 个系统调用位置进行预测,把前 $k-1$ 个位置视为条件属性集,第 k 个位置视为决策属性,从而建立起信息系统决策表。Rough 集理论的约简方法确保了能得到一组预测第 k 个系统调用位置的最小规则集,从而可以利用这组规则来对实际进程进行异常检测。基于 UNM 的合成 sendmail 系统调用数据的实验结果表明,本文所提出的异常检测

*)国家自然科学基金项目(60372039)和"十五"国防预研基金项目(41329080101)。鲜 明 博士、副教授,主要研究方向为信息安全与电子对抗;张义荣 博士研究生,主要研究方向为信息安全和智能信息处理;肖顺平 教授、博士生导师,主要研究方向为电子系统建模、仿真与评估;王国玉 研究员、博士生导师,主要研究方向为信息对抗与目标识别。

算法性能好于 Forrest 等人的 tide 方法，与 Wenke Lee 等人[2]的检测精度相当。但在选择较大的阈值时，漏报率更低。

2 粗糙集知识约简概述[8]

2.1 决策表系统与粗糙集

定义 1 一个决策表是一个信息表知识表达系统 $S=\langle U,R,V,f\rangle$，其中，$R=C\cup D$ 为属性集合，子集 C 和 D 分别称为条件属性集和结果属性集，$D\neq\varnothing$；$V=\bigcup_{r\in R}V_r$ 是属性值的集合，V_r 表示属性 $r\in R$ 的属性值范围；$f:U\times R\to V$ 是一个信息函数，它指定 U 中每一个对象 x 的属性值。条件属性 C 和结果属性 D 的等价关系 $IND(C)$ 和 $IND(D)$ 的等价类分别称为条件类和决策类。

定义 2 令 $X\subseteq U$，当 X 能用属性子集 B 确切地描述(即是属性子集 B 所确定的 U 上的不分明集的并)时，称 X 是 B 可定义的，B 可定义集也称作 B 精确集，B 不可定义集称为 B 粗糙(Rough)集。

定义 3 令决策表系统为 $S=\langle U,R,V,f\rangle$，$R=P\cup D$ 为属性集合，子集 $P=\{a_i\mid i=1,\cdots,m\}$ 和 $D=\{d\}$ 分别为条件属性集和决策属性集；$U=\{x_1,x_2,\cdots,x_n\}$ 是论域，$a_i(x_j)$ 是样本 x_j 在属性 a_i 上的取值。$C_D(i,j)$ 表示可辨识矩阵中第 i 行、第 j 列的元素，则可辨识矩阵 C_D 定义为：

$$C_D(i,j)=\begin{cases}\{a_k\mid a_k\in P\wedge a_k(x_i)\neq a_k(x_j)\} & ,d(x_i)\neq d(x_j)\\ 0 & ,d(x_i)=d(x_j)\end{cases}$$

2.2 决策表属性约简

属性约简就是要从条件属性集合中发现部分必要的条件属性，使得根据这部分条件属性形成的相对于决策属性的分类和所有条件属性形成的相对于决策属性的分类一致，即和所有条件属性相对于决策属性 D 有相同的分类能力。

定义 4 设 U 为一个论域，P 和 Q 为定义在 U 上的两个等价关系簇，Q 的 P 正域记为 $POS_P(Q)$，定义为 $POS_P(Q)=\bigcup_{X\in U/Q}P_-(X)$。

进一步，若 $POS_P(Q)=POS_{(P\setminus\{r\})}(Q)$，则称 r 为 P 中相对于 Q 可省略的。若 P 中每一 r 都是 P 中相对于 Q 不可省略的，则称 P 相对于 Q 独立。

定义 5 设 U 为一个论域，P 和 Q 为定义在 U 上的两个等价关系簇，若 P 的 Q 独立子集 $S\subset P$，满足：$POS_S(Q)=POS_P(Q)$，则称 S 为 P 的 Q 约简，记为 $RED_Q(P)$。P 的所有 Q 约简的交称为 P 的 Q 核，记为 $CORE_Q(P)$。

决策表属性约简为一 NP-hard 问题，相应的算法有一般约简算法、基于可辨识矩阵和逻辑运算的约简算法、归纳属性约简算法、基于互信息的属性约简算法和基于特征选择的属性约简算法等。

2.3 决策表值约简

属性约简只是在一定程度上去掉了决策表中的冗余信息，但是还没有充分去掉决策表中的冗余信息。值约简是在属性约简的基础上对决策表的进一步简化。与属性约简中的属性核一样，值约简中也可以定义相应的值核。

决策表 $S=\langle U,R,V,f\rangle$，对于任意的 $x\in U$，用 d_x 表示决策规则，即 $d_x:des([x]_C)\Rightarrow des([x]_D)$，$d_x(a)=a(x)$，$a\in C\cup D$ 且 $d_x\{C,d_x\{D$ 分别称为 d_x 的条件和决策。

定义 6 考虑一个相容知识表达系统 S，对决策规则 d_x，有 $[x]_C\subseteq[x]_D$。若 $\forall r\in C$，有 $[x]_{C\setminus\{r\}}\not\subset[x]_D$，则 r 为 d_x 的核值属性，r 为 d_x 中不可省略的；反之，若 $[x]_{C\setminus\{r\}}\subseteq[x]_D$，则 r 不是 d_x 的核值属性，r 为 d_x 中可省略的。

同决策表属性约简一样，值约简通常也采用启发式方法。值约简算法有一般值约简算法、归纳值约简算法、启发式值约简算法和基于决策矩阵的值约简算法等。

2.4 不一致推理

对决策表进行属性约简和值约简后，得到一系列约简后的决策规则。然而，这些决策规则可能存在不一致的问题。即在一定的前提条件下，多个规则得到满足，而它们的结论却不相同。造成这种不一致的原因有：决策表包含冲突样例、数据预处理(补齐、离散化)产生的不一致、样例不足等。为了对决策规则进行冲突消解，需要进行不一致推理，相应的推理策略有加权综合法、规则后修剪法、高信任度优先法、多数优先原则和少数优先原则等。

3 基于粗糙集理论的异常检测模型

设 X 是某一进程在正常运行状态下所能产生的所有系统调用序列的集合。用一个长度为 k 的窗口沿着 X 中的每一条序列滑动，得到一系列长度为 k 的系统调用短序列，所有这些系统调用短序列组成了正常系统调用序列段集合(论域) U。对于 U 中的每一个系统调用短序列，前 $k-1$ 个系统调用称为一般位置属性，它们组成了一般位置属性集 C；第 k 个系统调用称为系统调用短序列的末位置属性，构成了决策属性集 $\{d\}$。一般位置属性和末位置属性取值范围为 V，V 为所有系统调用组成的集合(通常 UNIX 的系统调用个数不超过 180)。建立正常进程行为模型的基本思想就是根据 X 中的前 $k-1$ 个系统调用预测第 k 个系统调用，即找到条件属性集 C 确定决策属性 d 的最小决策规则集。

定义 7[9] 基于系统调用行为的进程正常模型 $M(L)$ 是正常系统调用短序列集合 U 中描述一般位置属性集 C 与末位置属性 d 之间关系的预测规则集。

一旦训练得到了进程行为的正常模型，可以根据进程运行产生的系统调用短序列判定进程的运行状态是否正常。一条规则同一个长度为 k 的短序列匹配，应满足：

(1)规则适用的长度为 k；

(2)规则前件中描述的序列调用短序列各个位置上的系统调用同序列段中的情况一致。

由于可能出现不一致规则的情况，蔡忠闽等在文[9]中采用了简单的最大投票(Max-voting)策略来进行异常检测，其过程如下。

Step1：用一个长度为 k 的窗口在待分析的系统调用序列上滑动，步长为 1，每次从系统调用序列上截取一个长度为 k 的短序列；

Step2：在正常模型中，寻找同此短序列匹配的规则；

Step3：若不存在这样的规则，训练集中最常出现的预测结果被选作基于短序列中前 $k-1$ 个系统调用的预测；

Step4：若存在一条以上的规则同短序列匹配，采用投票的方式来决定最后的预测结果，每一条规则为其预测的结果投一票，票数最多的预测结果被选为最后的结果；

Step5：若预测的结果同该短序列的第 k 个系统调用相同，预测成功，返回 Step1；

Step6：若预测的结果同该短序列的第 k 个系统调用不同，则预测失败，返回 Step1。

对于可能出现决策规则不一致的情况，我们认为蔡忠闽等人采用的最大投票策略不一定是最佳推理策略，因为在 Step4 中最后的匹配结果仍可能出现多条规则同一个进程运行轨迹匹配的情况，即多条规则的票数一样；其次，在规则匹

配时没有考虑规则的可信度、覆盖范围等因素，然而它们是实时入侵检测的重要因素。因此，在本文的实验中，采用了不同的推理方法来做短序列匹配：高信任度优先法（记为 R1）、多数优先原则（记为 R2）和 LERAD[10]（记为 R3）。

另外，在文[9]中把系统调用序列的异常度定义为序列中正常模型规则集预测失败的短序列个数同序列中短序列总数之比。这种异常度定义没有充分考虑系统调用短序列的时间局域特性。攻击的出现一般具有突发性，当某个攻击发生时，在当前时刻之前的一段时间内的系统调用短序列也将与正常系统调用短序列不匹配，这意味着除了要考虑当前系统调用短序列的异常外，还要考虑毗邻的短序列的异常情况。因此，我们采用了一个大小为 $2n+1$、步长为 1 的窗口（局部帧，locality frame）沿执行轨迹滑动，计算在过去的 $2n+1$ 个短序列中不匹配序列的个数 m。如果 $m>n$，则此局部帧标记为异常，进程系统调用序列的异常度定义为调用序列中异常局部帧的比例。这样的异常度定义平滑了噪声和随机错误的影响，提高了检测的健壮性。

4 仿真实验与性能比较

4.1 实验数据源分析

为了便于性能比较，这里采用了两套 sendmail 系统调用数据集[11]。每个执行轨迹文件包含两列整数，第一列是进程的 ID 号，第二列是进程调用代号，这些代号可以通过一个索引文件转化成具体的系统调用名称。例如，代号“7”代表名称为“wait4”的系统调用，代号“9”代表名称为“link”的系统调用等。注意到，sendmail 进程能够产生分叉 fork（代号为“2”的系统调用），fork 系统调用创建父进程的精确副本并向父进程返回子进程的进程标识、对子进程返回零。fork 产生的子进程的系统调用序列被单独跟踪，并且它们也被包含到当前的 sendmail 进程序列中，用不同的 PID 号予以区分。

这里采用的是合成的 UNM sendmail 系统调用数据，它通过一个事先准备的脚本运行获得，程序的选项仅为使其运行可选，并不满足用户的任何真实请求。数据的获得是用 strace 调试工具记录 UNM 未打补丁的 Sun SPARC 工作站 SunOS 4.1.1 和 SunOS 4.1.4 内置的 sendmail 程序而来。异常 sendmail 调用序列包括 3 条 sscp 入侵轨迹、2 条 syslog-remote 入侵轨迹、2 条 syslog-local 入侵轨迹、2 条 decode 入侵轨迹、1 条 sm5x 入侵轨迹和 1 条 sm565a 入侵轨迹。

4.2 实验数据预处理

用长度为 k 的窗口将正常进程调用序列分割成一系列系统调用短序列集，从中随机选取 5% 用作正常训练集，其余作为正常测试集。同样地，用长度为 k 的窗口将入侵进程调用序列分割成一系列系统调用短序列集，此短序列集全部用作异常测试集。经过预处理后，得到 13 个决策表，其中包括 11 个入侵调用序列形成的决策表。

4.3 实验结果

实验采用波兰华沙大学与挪威科技大学联合开发的 Rosetta[12] 软件实现。实验中，本文采用了不同的推理方法来做短序列匹配：高信任度优先法（R1）、多数优先原则（R2）和 LERAD（R3）。局部帧设为 21。Wenke Lee 等人[2] 在实验中发现采用不同的局部帧对检测结果的影响并不大（没有数据集之间差别的影响大），因此这里只使用了大小为 21 的局部帧，并把步长取为 1。为了检查不同大小的滑动窗口分割系统调用序列对检测结果的影响，本文使用了不同大小的窗口 6，11，20 来分别分割进程系统调用序列。在每一窗口下，实验重复 8 次，平均后的实验结果如表 1、表 2、表 3 所示。

表 1 窗口长度为 6 时的异常度检测对比

执行轨迹	异常度（%）			
	Forrest 等[1]	R1	R2	R3
sscp	4.1	42.1	45.8	49.2
syslog-remote1	4.2	36.4	43.8	42.9
syslog-remote2	1.5	39.2	44.9	44.5
syslog-local1	4.2	27.1	28.2	26.3
syslog-local2	3.4	32.7	39.7	40.1
decode	0.3	6.7	7.2	8.5
sm565a	0.4	12.7	14.8	13.7
sm5x	1.7	26.4	22.5	20.8
入侵序列异常度平均值	2.48	27.9	30.9	30.8
sendmail	0	0.45	0.71	1.12

（因文[2]没有给出窗口长度为 6 时的结果，故无法与之比较）

表 2 窗口长度为 11 时的异常度检测对比

执行轨迹	异常度（%）				
	Forrest 等[1]	W. Lee 等[2]	R1	R2	R3
sscp	5.2	48.3	45.3	47.4	50.1
syslog-remote1	5.1	47.1	42.7	50.2	45.9
syslog-remote2	1.7	42.4	39.8	45.6	44.0
syslog-local1	4.0	27.9	28.7	26.9	25.4
syslog-local2	5.3	38.5	35.4	36.4	37.3
decode	0.3	7.6	8.2	9.1	10.3
sm565a	0.6	15	13.5	16.8	14.2
sm5x	2.7	22.8	20.7	22.5	21.7
入侵序列异常度平均值	3.11	31.2	29.3	31.9	31.1
sendmail	0	0	0.62	0.58	1.04

从表 1 可以看出，基于 Rough 集推理的异常检测方法比 Forrest 等[1] 的方法检测精度高。在窗口长度为 6 时，使用 R1 推理方法得到的入侵序列异常度平均值达到了 27.9%，且所有入侵序列异常度不低于 6.7%，而 sendmail 正常序列异常度只有 0.45%，这意味着只需使用一个合适的阈值（比如 5%）就可检测出所有的异常序列。注意到，在窗口长度为 6 时，文[1]报导的入侵序列异常度只有 2.48%，远低于本文得到的结果，这是因为本文的方法充分利用了攻击发生时系统调用短序列的时间局域特性。我们统计了大小为 21 的局域帧内异常调用短序列的个数，攻击发生时产生的不匹配系统调用短序列的时域集中特性显著提高了本文的算法的有效性。文[1]对 sendmail 正常序列检测的异常度为 0，这是因为它几乎使用了所有的 sendmail 正常系统调用短序列进行训练，但在实际检测中一般很难得到进程的全部正常系统调用短序列。另一方面，在本文的异常检测算法中，使用不同的决策规则推理方法得到的结果也有差异。从表 1 可以看出，多数优先原则（R2）、高信任度优先法（R1）和 LERAD（R3）方法得到的入侵序列异常度平均值分别为 27.9，30.9 和 30.8。多数优先原则（R2）推理方法要好于高信任度优先法（R1）和 LERAD（R3）方法，这是因为 R2 推理方法既考虑了决策规则的可信度，又考虑了规则的覆盖范围。R3 方法也获得了与 R2 推理方法相当的入侵序列异常度平均值，但它对 sendmail 正常序列的检测的异常度大于 R2 推理方法，因此综合性能要次于 R2 推理方法。另外，也要看到，尽管 R2 推理方法的综合检测性能优于 R1 和 R3，但对不同的数据源，不同的推理方法得到的结果并不一样。对于 sm5x 入侵序列，R1 推理方法得到的异常度比其它两种方法高，这意味着 R1 推理方法

更适合检测 sm5x 入侵序列；而对于 sscp，syslog-local2 和 decode 入侵序列，R3 推理方法得到的异常度比其它两种方法高，因此 R3 推理方法更适合检测这 3 种入侵序列。

表 2 进一步验证了从表 1 得出的结论。从表 2 可以看出，在窗口长度为 11 时，本文的基于 Rough 集推理的异常检测方法比 Forrest 等[1]的方法检测精度高，使用 R1 推理方法得到的入侵序列异常度平均值达到了 29.3%，且所有入侵序列异常度不低于 8.2%，而 sendmail 正常序列异常度只有 0.62%，这意味着只需使用一个合适的阈值（比如 5%）就可检测出所有的异常序列。再一次看到，在本文的异常检测算法中，使用不同的决策规则推理方法得到的结果不同。从表 2 可以看出，多数优先原则（R2）、高信任度优先法（R1）和 LERAD（R3）方法得到的入侵序列异常度平均值分别为 29.3，31.9 和 31.1，而对 sendmail 正常序列检测的异常度分别为 0.62%，0.58%和 1.04%，多数优先原则（R2）推理方法仍要好于高信任度优先法（R1）和 LERAD（R3）方法。R3 方法获得了比 R2 推理方法略低的入侵序列异常度平均值，但它对 sendmail 正常序列的检测的异常度大于 R2 推理方法，这和从表 1 得到的结论一致。同时，另外也看到，对于 syslog-local1 入侵序列，R1 推理方法得到的异常度比其它两种方法高，这意味着 R1 推理方法更适合检测 syslog-local1 入侵序列；而对于 sscp，syslog-local2 和 decode 入侵序列，R3 推理方法得到的异常度仍比其它两种方法高，因此 R3 推理方法更适合检测这 3 种入侵序列。最后，在同 Wenke Lee 等[2]的算法比较上，本文采用的基于 R2 推理的异常检测方法要好于 Wenke Lee 等的方法，R2 推理方法得到的入侵序列异常度平均值为 31.9%，而 Wenke Lee 等的结果是 31.2%。尽管 Wenke Lee 等采用的数据挖掘方法对 sendmail 正常序列检测的异常度为 0，但其得到的最小入侵序列异常度值为 7.6%，小于本文得到的 9.1%，这意味着我们在选择检测门限时比 Wenke Lee 等更灵活，在选择较大的阈值时（如 8%），本文算法的漏报率要低于 Wenke Lee 等人的方法。

由于文[1]和文[2]没有给出窗口大小为 20 时的检测结果，因此表 3 只列出了本文采用不同推理方法时得到的检测结果。从表 3 可以得出与从表 1、表 2 相似的结论：多数优先原则（R2）推理方法要好于高信任度优先法（R1）和 LERAD（R3）方法。纵向看，对于同一种推理方法，采用不同大小的窗口分割进程系统调用序列时得到的检测结果也不相同，其中窗口大小为 11 时结果最好。但不同窗口得到的检测结果相差不大，一般在 2%之内。这意味着，对于 sendmail 进程系统调用序列的异常检测而言，数据源和决策规则推理方法之间的差别要大于窗口选择的差别。

表 3　窗口长度为 20 时的异常度检测对比

执行轨迹	异常度(%)		
	R1	R2	R3
sscp	49.8	48.8	52.0
syslog-remote1	31.4	45.8	43.9
syslog-remote2	35.8	43.4	44.2
syslog-local1	29.1	27.2	27.0
syslog-local2	34.5	37.7	36.8
decode	7.5	8.9	12.3
sm565a	11.9	14.2	12.9
sm5x	24.2	23.6	22.8
入侵序列异常度平均值	28.0	31.2	31.5
sendmail	0.69	0.61	0.97

（因文[1]、[2]没有给出窗口长度为 20 时的结果，故表中未列出其检测结果）

为了直观上显示表 1、表 2 和表 3 中各种检测方法对 sendmail 进程系统调用序列异常检测结果的差异，作出了同一大小窗口分割下不同方法检测结果的直方图，如图 1、图 2 所示。同时，为了纵向比较不同窗口下本节 3 种基于 Rough 集推理方法检测结果的差异，我们也作出了不同窗口下同一方法检测结果的直方图，如图 3、图 4 所示。

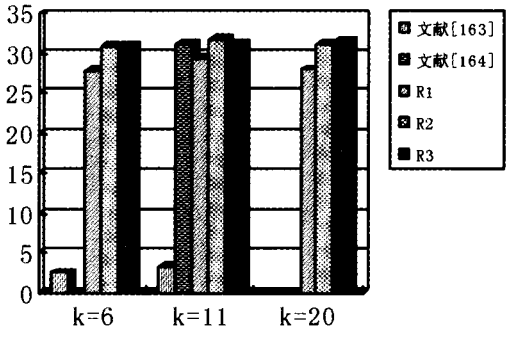

图 1　同一窗口下 sendmail 入侵序列平均异常度对比（%）：文[163]，文[164]，R1，R2，R3

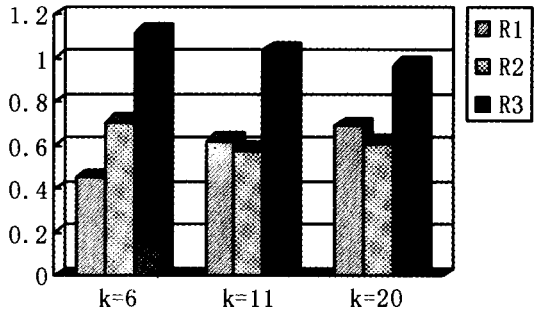

图 2　同一窗口下 sendmail 正常序列异常度对比（%）：R1，R2，R3

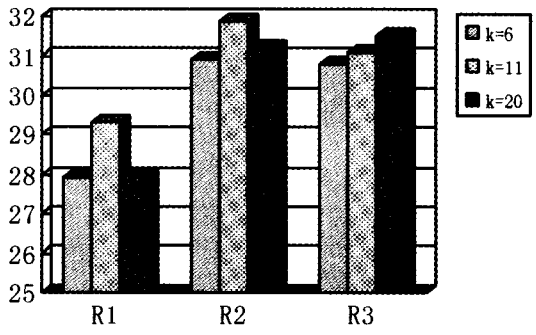

图 3　不同窗口下 sendmail 入侵序列平均异常度对比（%）：R1，R2，R3

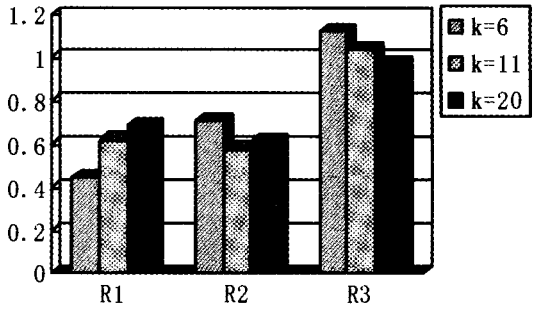

图 4　不同窗口下 sendmail 正常序列异常度对比（%）：R1，R2，R3

（下转封三）

驱动器说明；GetFolder：根据指定路径中的文件夹返回相应的 Folder 对象；GetParentFolderName：根据指定路径中的最后成分返回包含其父文件夹名称的字符串；GetSpecialFolder 方法返回指定的特殊文件夹对象，有 window、system、temp 三种；GetTempName：返回一个随机产生的临时文件或文件夹名，有助于执行那些需要临时文件或文件夹的操作；MoveFile：从一个位置向另一个位置移动一个或多个文件；MoveFolder：从一个位置向另一个位置移动一个或多个文件夹；OpenTextFile：打开指定的文件并返回一个 TextStream 对象，可以通过这个对象对文件进行读、写或追加。

3 系统功能及实现

Windows 系统监视程序主要用来获取网络信息、驱动器信息、文件夹信息、系统关键文件信息等。

为了获取网络信息，使用 WSH 自带的网络对象。具体编程代码如下：

```
Sub dispnetstat '显示网络状态的子程序
    '创建网络对象
    Set WshNetwork = Wscript. CreateObject ("Wscript. Network)
    '获取系统所有的网络驱动器
    Set oDrives=WshNetwork. EnumNetworkDrives
    '获取系统所有的网络打印机
    Set oPrinters=WshNetwork. EnumPrinterConnections
    '获取系统域名
    netinfo="域名:"+WshNetwork. UserDomain+vbcrlf
    '获取计算机名
    netinfo=netinfo+"计算机名:"+WshNetwork. ComputerName+vbcrlf
    '通过循环获取网络驱动器的名称
    netmap=""
    for i=0 to oDrives. Count()-1 step 2
        netmap=netmap+"驱动器"+oDrives. Item(i)+" ="
    +oDrives. Item(i+1)+vbcrlf
    next
    netprn=""
    '通过循环获取网络打印机的名称
    for i=0 to oPrinters. Count()-1 step2
        netprn=netprn+"端口:"+oPrinters. Item(i)+"="+
    oPrinters. Item(i+1)+vbcrlf
    next
    Set WshNetwork=nothing '释放网络对象变量
End Sub
```

为了获取驱动器信息、文件夹信息和系统关键文件信息等，必须通过使用 WSH 来调用 FSO 组件实现。其中，获取文件夹信息的具体代码如下，获取驱动器信息、系统关键文件信息等的方式与其类似，在此不具体给出。

```
Function folderatt(fo)'获取文件夹属性的函数
Dim att
On Error resume next '获取目录的名称
folderatt=folderatt+fo. name+"目录:"+vbcrlf
'获取目录的文件数
folderatt=folderatt+"该目录下的文件数:"+cStr(fo. Files. Count)
+vbcrlf
'获取目录的子目录数
folderatt=folderatt+"该目录下的子目录数:"+cStr(fo. SubFolders. Count)+vbcrlf
'获取目录的占用字节数
folderatt=folderatt+"该目录占用字节数:"+cStr(fo. Size/1024/1024)+"MB"+vbcrlf
'获取目录的最后修改时间
folderatt=folderatt+"该目录的最后修改时间:"+cStr(fo. DateLastModified)+vbcrlf
End Function
```

结束语 通过 WSH 可以充分利用脚本来实现计算机工作的自动化，但不可否认，也正是它的这一特点，使系统又有了新的安全隐患。许多计算机病毒制造者正在热衷于用脚本语言来编制病毒，并利用 WSH 的支持功能，让这些隐藏着病毒的脚本在网络中广为传播。因此，对于来历不明、尤其是邮件附件里的一些脚本文件还是应该保持戒备。

参 考 文 献

1 张志民，王强，武港山. 基于 WSH 的脚本开发. 计算机应用研究，2000，(7)

2 哈节棍. 访问注册表之 WSH 篇. 中文信息：程序春秋，2002(9)

3 范青山. WSH，批处理技术的新武器. 软件，2002(6)

4 邵方，刘宗田. 脚本语言发展研究. 计算机科学，2000，27(1)

5 孙康生. 应用 FSO 对象模型实现文件查找的探讨. 开封大学学报，2004(4)

6 王军号，等. 基于 Web 的文件搜索引擎的设计与实现. 科技情报开发与经济，2005(4)

(上接第 284 页)

结束语 在 Unix 系统中，通过对进程正常运行时的执行轨迹进行分析来刻画进程的正常运行状态，是一种重要的异常检测技术。在利用进程运行时产生的系统调用短序列建立入侵检测模型方面，有枚举法、基于频率的方法、基于数据挖掘的方法、基于有限状态机的方法和结合调用参数的系统调用短序列异常检测等方法。本文提出了一种基于粗糙集约简的系统调用序列异常检测方法，它根据进程系统调用的前 $k-1$ 个位置，利用粗糙集约简来对第 k 个位置进行预测。Rough 集理论的约简方法确保了能得到一组预测第 k 个系统调用位置的最小规则集，从而可以利用这组规则来对实际进程进行异常检测。基于合成的 UNM sendmail 系统调用数据的实验结果表明，本文所提出的异常检测算法性能好于 Forrest 等人的 tide 方法，与 Wenke Lee 等人的数据挖掘算法检测精度相当。但在选择较大的阈值时，漏报率更低。

参 考 文 献

1 Forrest S, Hofmeyr S A, Somayaji A, et al. A sense of self for Unix process [J]. In: Proceedings of 1996 IEEE Symposium on Computer Security and Privacy, 1996. 120～128

2 Lee Wenke, Stolfo S, Chan Phil. Learning Patterns from Unix Process Execution Traces for Intrusion Detection [A]. AAAI Workshop: AI Approaches to Fraud Detection and Risk Management, July 1997

3 Warrender C, Forrest S, Pearlmutt B. Detecting Intrusion Detection Using System Calls: Alternative Data Model [J]. In: Proceedings of 1999 IEEE Symposium on Computer Security and Privacy, 1999. 133～145

4 Wespi A, Dacier M, Debar H. Intrusion Detection using variable-length audit trail patterns [J]. RAID, 2000. 110～129

5 Tandon G, Chan P. Learning Useful System Call Attributes for Anomaly Detection [A]. In: Proc 18th Intl FLAIRS Conf, 2005. 405～410

6 Liao Y H, Vemuri V R. Use of K-Nearest Neighbor classifier for intrusion detection [J]. Computers & Security, 2002, 21(5): 439～448

7 Lee Wenke, Stolfo S J, Mok K W. A Data Mining Framework for Building Intrusion Detection Models [A]. 1999 IEEE Symposium on Security and Privacy, Oakland, California, May. 1999

8 王国胤. Rough 集理论与知识获取[M]. 西安：西安交通大学出版社，2001

9 蔡忠闽，管晓宏，邵萍，等. 基于粗糙集理论的入侵检测新方法. 计算机学报[J]. 2003，26(3)：361～366

10 Mahoney M, Chan P. Learning Rules for Anomaly Detection of Hostile Network Traffic [A]. In: Proc. Third IEEE Intl Conf on Data Mining(ICDM), 2003. 601～604

11 UNM Sequence-based Intrusion Detection data set [EB/OL]. http://www.cs.unm.edu/～immsec/data/. Cited 2005

12 Rosetta [EB/OL]. Knowledge Systems Group, Dept of Computer and Info Science, Norwegian University of Science and Technology, Trondheim, Norway and Group of Logic, Inst of Mathematics, University of Warsaw, Poland. http://rosetta.lcb.uu.se/general/