计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(2):181−189 [doi: 10.15888/j.cnki.csa.008904]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于异常的终端级入侵检测①

熊文定, 罗凯伦, 李　睿

(东莞理工学院 网络空间安全学院, 东莞 523808)
通信作者: 李　睿, E-mail: ruili@dgut.edu.cn

摘　要: 入侵检测技术作为计算机防护的主要技术手段, 因具有适应性强、能识别新型攻击的优点而被广泛研究, 然而识别率和误报率难以保证是该技术的主要瓶颈. 为了提升异常检测技术的识别率并降低误报率, 提出了一种终端级入侵检测算法 (terminal-level intrusion detection algorithm, TL-IDA). 在数据预处理阶段把终端日志切割成连续的小块命令序列, 并引入统计学的常用指标为命令序列构建特征向量, 再使用 TL-IDA 算法通过特征向量对用户建模. 在此基础上, 还提出了一种滑动窗口判别法, 用于判断系统是否遭受攻击, 从而提升入侵检测算法的性能. 实验结果表明, TL-IDA 算法的平均识别率和误报率分别达到了 83% 和 15%, 优于同类的基于异常技术的终端级入侵检测算法 ADMIT、隐马尔可夫模型法等.

关键词: 计算机安全; 异常技术; 动态聚类; 终端级入侵检测; 滑动窗口判别法

引用格式: 熊文定,罗凯伦,李睿.基于异常的终端级入侵检测.计算机系统应用,2023,32(2):181–189. http://www.c-s-a.org.cn/1003-3254/8904.html

Anomaly-based Terminal-level Intrusion Detection

XIONG Wen-Ding, LUO Kai-Lun, LI Rui

(School of Cyberspace Security, Dongguan University of Technology, Dongguan 523808, China)

Abstract: As the main technical means of computer protection, intrusion detection technology has been widely studied due to its advantages of strong adaptability and ability to identify new types of attacks. However, the recognition rate and false alarm rate are difficult to guarantee, which is the main bottleneck of this technology. To improve the recognition rate and reduce the false alarm rate of anomaly detection technology, this study proposes a terminal-level intrusion detection algorithm (TL-IDA). In the data preprocessing stage, the terminal log is cut into continuous and small-block command sequences, and common statistical indicators are introduced to construct feature vectors for the command sequences. Then TL-IDA is applied to model users through the feature vectors. On this basis, a sliding window discrimination method is also proposed to judge whether the system is under attack, so as to improve the performance of the intrusion detection algorithm. The experimental results show that the average recognition rate and false alarm rate of the TL-IDA are 83% and 15%, respectively, which are superior to those of similar terminal-level intrusion detection algorithms based on anomaly technology such as ADMIT and hidden Markov model.

Key words: computer security; anomaly technology; dynamic clustering; terminal-level intrusion detection; sliding window discrimination method

1　引言

计算机系统的安全性广泛受到人们的关注. 根据数据显示, 2020 年上半年, 我国境内感染计算机恶意程序的主机数量约 304 万台, 同比增长 25.7%, 位于境外

① 基金项目: 国家重点研发计划 (2021YFB3101300); 国家自然科学基金面上项目 (61972089)
收稿时间: 2022-06-12; 修改时间: 2022-07-11; 采用时间: 2022-07-16; csa 在线出版时间: 2022-09-01
CNKI 网络首发时间: 2022-11-16

的约 2.5 万个计算机恶意程序控制了我国境内约 303 万台主机. 因此, 从终端层面对计算机的安全性进行检测变得越发重要. 入侵检测系统作为防火墙、强认证和用户特权等预防性技术的补充, 可以监控网络或其他系统的异常行为, 其对于网络和终端的检测能力, 已经成为互联网技术安全管理的重要组成部分[1]. 然而, 由于网络流量和终端类型日益复杂化、多样化, 传统的基于签名技术的入侵检测系统越来越难以胜任入侵检测的任务. 因此, 基于异常技术的入侵检测系统是目前的重点研究方向.

目前入侵检测的相关研究主要集中在网络层面. 如 Ahmim 等人[2] 提出的基于决策树和规则概念的入侵检测方法, Lin 等人[3] 提出的一种称为聚类中心和最近邻的检测方法, Cui 等人[4] 提出的一种用于系统日志异常检测的卷积自动编码器方法, 还有基于卷积神经网络的字符级入侵检测方法[5], 均用于网络流量检测. 然而, 仅依靠网络级的安全检测无法保证终端的安全性, 因为授权用户的密码可能会被入侵者破解[6], 此时网络级的检测将不起作用.

某些早期的终端级入侵检测方法的检测对象是系统调用系列. 如 Cabrera 等人[7] 提出通过长度固定的系统调用序列来检测终端是否存在异常行为. Lee 等人[8] 把该问题当作分类问题, 采用机器学习分类器把系统调用序列区分为正常和异常 2 种类别. 此外, 宋海涛等人[9] 提出基于模式挖掘的异常识别算法. 然而, 系统调用级别的颗粒度太细, 不仅导致过大的系统开销, 检测准确性也难以保证. 此外, 一些入侵检测方法把终端的操作日志作为检测对象. 在这方面作出重要贡献的是 Ryan 等人[10], 他们首先提出用人工神经网络在终端提取用户特征的方法. 此外, DuMouchel[11] 创建了连续的基于命令序列的概率转移矩阵作为用户模型. 最后, Schonlau 等人[12] 测试了建立用户模型的各种统计方法. 与本文密切相关的方法是由 Lane 和 Brodley[13,14] 提出的, 他们使用基于实例学习和隐马尔可夫模型技术, 把用户的行为划分为若干个聚簇. 由于基于实例学习技术在归类用户的行为类别时只参考部分相近的实例, 从而限制了问题的复杂性. 然而, 该方法的前提条件是异常实例占比很小, 并且该方法存在需要为实例设置标签、运行时开销较大的缺陷. 在文献 [13,14] 的基础上, Sequeira 等人[15] 以命令相似度为指标, 使用动态聚类算法对数据集进行分类来建立用户模型. 他们的方法放宽了数据预处理的限制, 然而, 他们没有考虑命令表达的多样性问题. 据作者所知, 现有的终端级入侵检测方法大多需要为训练数据预先设置标签, 并且需要大量的训练数据集和较长的训练时间. 此外, 这些方法均没有解释聚簇的含义.

本文研究的是基于异常技术的终端级入侵检测问题, 提出了一种终端级入侵检测算法 TL-IDA, 并提出了对应的滑动窗口判别法. TL-IDA 算法依靠终端的操作日志对用户进行建模, 模型接收终端输入并采用滑动窗口判别法判断是否为异常操作. 为了实现该算法, 在数据预处理阶段构建特征向量时引入了统计学的常用指标, 并设计了符合本问题的动态聚类算法. 最后, 本文总结了操作日志存在的 4 种行为类别. 在实验部分, 采用 K 折交叉验证法和控制变量法对数据集进行了测试, 实验结果表明 TL-IDA 算法比类似的算法更好. 本文的主要贡献有:

(1) 提出了终端级入侵检测算法 TL-IDA.

(2) 提出了滑动窗口判别法用于识别异常行为.

(3) 总结了操作日志的行为类别.

2 TL-IDA 算法

本文提出的 TL-IDA 算法包括数据预处理和模型训练 2 个部分. 第 3 部分使用训练好的模型结合滑动窗口判别法对输入进行数据分类. 图 1 是入侵检测方法的处理流程.

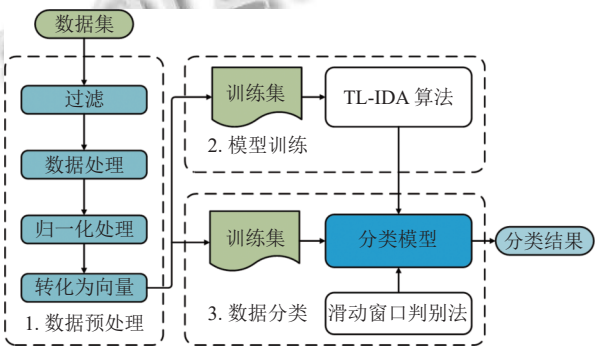

图 1 入侵检测方法的处理流程

2.1 数据预处理

操作日志由大量历史命令组成, 需要对日志进行预处理后才能用于建模. Sequeira 等人[15] 认为, 日志中的单个命令不能保留用户的使用习惯, 但连续的小块命令序列可以, 并且, 把日志转化成连续的小块命令序列集还可以生成大量样本用于模型的训练和测试, 因

此, 本文借鉴了该思想. 此外, 为了尽可能去除噪音干扰, 在分割日志的命令序列前, 还需要进行数据清理, 把不存在的命令删除. 表 1 展示了长度为 4 的命令序列, 设定子命令序列长度为 2、步长为 1 时的切割结果.

表 1 原长度为 4 的序列, 设子序列长度为 2, 步长为 1 时的切割结果

序列名称	切割结果
原命令序列	S={t_0=*ls–l*, t_1=*vi*, t_2=*ps–ef*, t_3=*vi*}
子命令序列	S_0={t_0=*ls–l*, t_1=*vi*}
	S_1={t_1=*vi*, t_2=*ps–ef*}
	S_2={t_2=*ps–ef*, t_3=*vi*}

得到子命令序列集后, 需要进行数值化处理, 把子命令序列逐个转化为特征向量, 为此, 本文引入了统计学的常用指标来保留用户特征. 首先, 定义两个子命令序列的相似度为这两个序列的最长公共子序列 (longest common subsequence, LCS). 假设命令序列 S_0={t_0=*ls–l*, t_1=*vi*, t_2=*ps–ef*, t_3=*vi*}, S_1={t_0=*vi*, t_1=*ps–ef*, t_2=*vi*, t_3=*touch*}. 根据定义, S_0 和 S_1 的 LCS 为{*vi*, *ps–ef*, *vi*}, 相似度为 3. 然后, 对操作日志统计以下数据:

(1) 所有命令的出现次数.

(2) 单个命令的出现次数.

(3) 单个命令的所有可选参数的出现次数.

(4) 单个命令的单个可选参数的出现次数.

(5) 用户的特征序列.

对于 (5), 先统计每个子命令序列与其他的子命令序列的累计相似度, 再以累计相似度最高的前 20% 子命令序列作为用户的特征序列. 最后, 定义特征:

(1) x_1: 命令的平均出现频率.

(2) x_2: 可选参数的平均出现频率.

(3) x_3: 与特征序列的最大相似度除以序列长度.

则每个子命令序列的特征向量为 (x_1, x_2, x_3), 在这个特征向量中, x_1 和 x_2 保留了命令的频率特征, x_3 保留了命令的顺序特征.

2.2 TL-IDA 算法设计

完成数据预处理后, 需要设计算法对特征向量进行聚类. 显然, 传统的 K-means 算法不适合本问题, 原因在于无法预测 K 的值 (即合理聚簇个数). Ahamed 等人[16] 提出的动态聚类算法解除了必须提前设置 K 值的限制, 但对于本问题表现不够理想. 为此, 本文设计了 TL-IDA 算法, 它主要由 3 个函数组成. 在分析这 3 个函数之前, 先介绍 3 个工具函数如下.

(1) $dist(X,Y)=\sqrt{\sum_{i=1}^{m}(x_i-y_i)^2}, x_i\in X, y_i\in Y$, 用于计算特征向量 X, Y 的距离. 假设特征向量维度为m, 则函数的时间复杂度为 O(m).

(2) $inter(SC)=\min(dist(c_i,c_j)), c_i,c_j\in S_C, i\neq j$, 用于计算聚簇间的分离度. 其中, S_C 是聚簇的中心集, c_i, c_j 分别是第 i 个聚簇和第 j 个聚簇的中心. 假设 K 为聚簇个数, 则函数的时间复杂度为 O(K^2m).

(3) $intra(S_c,S_p)=\dfrac{\sum_{i=1}^{K}\dfrac{\sum_{j=1}^{n_i}dist(s_j,c_i)}{n_i}}{K}, S_{p_i}\in S_p$, 用于计算所有聚簇的整体紧凑度, 时间复杂度为 O(mn). 其中, S_p 是聚类后的样本集, S_{pi} 是第 i 个聚簇的样本集, n_i 是第 i 个聚簇的样本个数.

2.2.1 BaseClustering

首先介绍 BaseClustering 函数, 它实现了基础的聚类功能. 该函数流程分为 2 个步骤: (1) 选择 K 个样本作为聚簇的中心, 将样本逐个分配到与其最近的中心的聚簇中, 得到一个聚类结果; (2) 更新每个聚簇的样本均值作为聚簇的新中心. 重复以上步骤, 直到所有聚簇的中心不再变化为止. 算法伪代码如算法 1.

算法 1. BaseClustering(S, K)

输入: (S, K), S 是待聚类的样本集

输出: (S_c, S_p), S_c 是聚簇中心集, S_p 是聚簇样本集

```
1: Sc←Randomly select K samples
2: do
3:     for i←1; i≤K; i++ do
4:         Spi←∅
5:     end for
6:     for s in S do
7:         for ci in Sc do
8:             if dist(s, ci) is min then
9:                 Spi←Spi∪s
10:            end if
11:        end for
12:    end for
13:    recalculate Sc
14: while Sc change
15: return Sc, Sp
```

接下来通过例子模拟该函数的执行过程. 假设 S={s_0=(0.972, 0.716), s_1=(0.776, 0.501), s_2=(0.515, 0.314), s_3=(0.305, 0.675), s_4=(0.755, 0.923), s_5=(0.605, 0.704), s_6=(0.634, 0.088), s_7=(0.445, 0.932), s_8=(0.427, 0.571), s_9=(0.298, 0.238)}, K=2. 首先, 随机选取中心集 S_c={c_0=s_1, c_0=s_3}. 然后, 把样本分配到距离最近的中心

的聚簇 (第 8、9 行), 则第 1 次迭代得到:

S_p={S_{p0}={s_0, s_1, s_2, s_4, s_5, s_6}, S_{p1}={s_3, s_7, s_8, s_9}},

S_c={c_0=(0.368, 0.604), c_1=(0.709, 0.541)}.

由于 S_c 发生变化, 继续第 2 次迭代 (第 14 行), 得到:

S_p={S_{p0}={s_0, s_1, s_2, s_4, s_5, s_6}, S_{p1}={s_3, s_7, s_8, s_9}},

S_c={c_0=(0.368, 0.604), c_1=(0.709, 0.541)}.

由于 S_c 没有变化, 因此跳出循环, 聚类计算完成. 分析可知, 每次迭代必然导致样本到达所属聚簇中心的距离之和减少, 因此函数具备收敛性.

显然, BaseClustering 函数的基本操作为 *dist* (第 8 行), 它的时间复杂度为 O(m). 当样本个数为 n、指定聚簇个数为 K 时, 该函数执行一次while迭代的时间复杂度为 O(Kmn), 假设完成聚类计算需要执行δ次迭代, 则函数的复杂度为 O(δKmn).

2.2.2 DynamicClustering

为了解除 BaseClustering 函数需要设置 K 值的限制, 本文设计了 DynamicClustering 函数. 该函数通过聚簇间的分离度 (inter_val) 和聚簇的紧凑度 (intra_val) 来判断聚类是否停止. 算法伪代码如算法 2.

算法 2. DynamicClustering(S)

输入: (S), S 是待聚类的样本集
输出: (S_c, S_p), S_c 是聚簇中心集, S_p 是聚簇样本集

```
1: K←1
2: new_inter←min float number
3: new_intra←max float number
4: do
5:     K←K+1
6:     S_c, S_p←BaseClustering(S, K)
7:     old_inter←new_inter
8:     old_intra←new_intra
9:     new_inter←inter(S_c)
10:    new_intra←intra(S_c, S_p)
11:    ConditionA←old_inter<new_inter
12:    ConditionB←old_intra>new_intra
13: while ConditionA and ConditionB
14: return S_c, S_p
```

使用第 2.2.1 节 BaseClustering 函数的输出结果来模拟 DynamicClustering 函数的执行过程. 首先, 根据 S_p 和 S_c, 可以通过工具函数 inter 和 intra 得到 inter_val=0.346, intra_val=0.563. 然后, 把 K 自增为 3, 再次调用 BaseClustering 函数, 得到:

S_p={S_{p0}={s_0, s_1, s_4, s_5}, S_{p1}={s_3, s_7, s_8}, S_{p2}={s_2, s_6, s_9}},

S_c={c_0=(0.777, 0.711), c_1=(0.392, 0.726), c_2=(0.482, 0.213)}.

计算得到 inter_val=0.384, intra_val=0.497. 由于 ConditionA 和 ConditionB 同时成立 (第 11、12 行), 表明 K=3 时的聚类效果更好, 因此保留本次修改并执行 K=4 时的迭代 (第 13 行). 同理可得:

S_p={S_{p0}={s_1, s_2, s_6}, S_{p1}={s_3, s_5, s_7, s_8}, S_{p2}={s_0, s_4}, S_{p3}={s_9}},

S_c={c_0=(0.641, 0.301), c_1=(0.446, 0.721), c_2=(0.864, 0.820), c_3=(0.298, 0.238)}.

计算得到 inter_val=0.349, intra_val=0.389. 由于 ConditionA 不成立, 表明 K=4 时的聚类效果并不优于 K=3 时的聚类效果, 因此放弃本次修改, 聚类计算完成. 该函数的极端情况为每个聚簇只保留一个样本, 因此该函数具有收敛性.

观察可知, DynamicClustering 函数迭代时计算量最大的操作为调用 BaseClustering 函数 (第 6 行). 在最极端的情况下, 每个聚簇只保留一个样本, 此时时间复杂度达到最坏. 由于单次聚类计算的时间复杂度为 O(δKmn), 因此最坏时间复杂度为 O(δKmn^2).

2.2.3 AdjustCluster

在聚类完成后, 为了提高聚簇的紧凑度和聚簇间的分离度, 设计了 AdjustCluster 函数. 该函数流程分为 2 个步骤: (1) 对于每个聚簇, 该函数把到中心距离大于阈值 T_{dist} 的样本移除; (2) 更新每个聚簇的中心. 重复以上步骤直到中心不再变化. 算法伪代码如算法 3.

算法 3. AdjustCluster(S_c, S_p, T_{dist})

输入: (S_c, S_p, T_{dist}), S_c 是聚簇中心集, S_p 是聚簇样本集, T_{dist} 是聚簇的半径
输出: (S_c, S_p), S_c 是聚簇中心集, S_p 是聚簇样本集

```
1: do
2:     for c_i in S_c do
3:         for S_pi in S_p do
4:             for s in S_pi do
5:                 if dist(s, c_i)>T_dist then
6:                     S_pi←S_pi−s
7:                 end if
8:             end for
9:         end for
10:    end for
11:    recalculate S_c
12: while S_c change
13: return S_c, S_p
```

接续 DynamicClustering 函数的输出结果来模拟

AdjustCluster 函数的执行过程. 假设 T_{dist} 为 0.2, 根据伪代码第 5 至 7 行, 对 S_p 和 S_c 的迭代计算得到:

$dist(s_0, c_0)=0.195<T_{dist}$, S_{p0} 保留 s_0

$dist(s_1, c_0)=0.210>T_{dist}$, S_{p0} 移除 s_1

$dist(s_4, c_0)=0.213>T_{dist}$, S_{p0} 移除 s_4

$dist(s_5, c_0)=0.172<T_{dist}$, S_{p0} 保留 s_5

$dist(s_3, c_1)=0.101<T_{dist}$, S_{p1} 保留 s_3

$dist(s_7, c_1)=0.213>T_{dist}$, S_{p1} 移除 s_7

$dist(s_8, c_1)=0.159<T_{dist}$, S_{p1} 保留 s_8

$dist(s_2, c_2)=0.106<T_{dist}$, S_{p2} 保留 s_2

$dist(s_6, c_2)=0.197<T_{dist}$, S_{p2} 保留 s_6

$dist(s_9, c_2)=0.186<T_{dist}$, S_{p2} 保留 s_9

因此:

$S_p=\{S_{p0}=\{s_0, s_5\}, S_{p1}=\{s_3, s_8\}, S_{p2}=\{s_2, s_6, s_9\}\}$,

$S_c=\{c_0=(0.789, 0.710), c_1=(0.366, 0.623), c_2=(0.482, 0.213)\}$.

由于 S_c 发生变化, 继续执行迭代 (第 12 行) , 同理可得:

$dist(s_0, c_0)=0.1835<T_{dist}$, S_{p0} 保留 s_0

$dist(s_5, c_0)=0.1835<T_{dist}$, S_{p0} 保留 s_5

$dist(s_3, c_1)=0.0801<T_{dist}$, S_{p1} 保留 s_3

$dist(s_8, c_1)=0.0801<T_{dist}$, S_{p1} 保留 s_8

$dist(s_2, c_2)=0.1058<T_{dist}$, S_{p2} 保留 s_2

$dist(s_6, c_2)=0.1967<T_{dist}$, S_{p2} 保留 s_6

$dist(s_9, c_2)=0.1859<T_{dist}$, S_{p2} 保留 s_9

$dist(s_9, c_2)=0.1859<T_{dist}$, S_{p2} 保留 s_9

观察发现, 本次迭代没有移除样本点, 因此 S_c 保持不变, 调整操作完成. 由于每个聚簇的样本有限, 因此该函数具备收敛性.

AdjustCluster 函数的基本操作为 *dist* (第 5 行). 假设样本个数为 n, 聚簇个数为 K, 迭代次数为γ, 则函数的时间复杂度为 $O(\gamma Kmn)$.

本文主要设计了 3 个函数来实现 TL-IDA 算法. 首先, BaseClustering 函数实现最基本的聚类功能, 其次, DynamicClustering 函数通过聚簇间的分离度和聚簇的紧凑度是否更好来判断聚类计算是否停止, 取消了需要提前输入 K 值的限制. 最后, AdjustCluster 函数用于清除边缘的样本, 使得聚类的结果更好. 对比 3 个函数的时间复杂度可知, TL-IDA 算法最差的时间复杂度为 $O(\delta Kmn^2)$, 通常情况下, 样本数 n 远大于聚簇个数K和向量维度 m, 因此 TL-IDA 算法的时间复杂度为 $O(n^2)$ 级别.

2.3 方法对比

本文方法的检测对象是终端级操作日志, 对比 Cabrera 等人[7]、Lee 等人[8] 和宋海涛等人[9] 对系统调用级数据进行分析的方法, 本文方法的系统开销更小. 同时, 与 Ryan 等人[10] 基于人工神经网络的入侵检测方法对比, 本文方法在很大程度上缩短了训练时间. 与本文密切相关的 Sequeira 等人[15] 对比, TL-IDA 算法的时间复杂度为 $O(n^2)$ 级别, 而他们的算法时间复杂度为 $O(n^3)$ 级别, TL-IDA 算法有更好的时间复杂度. 表 2 更为详细地对比了本文与其他 5 种终端级入侵检测方法的差别.

表 2 终端级入侵检测方法对比

文献	基本方法	复杂度	训练时间	检测对象
本文	动态聚类	$O(n^2)$	短	操作日志
[7]	统计分析	—	较长	系统调用序列
[8]	规则系统	—	较长	系统调用序列
[9]	序列检测	$O(n^3)$	一般	系统调用序列
[10]	人工神经网络	—	很长	操作日志
[15]	动态聚类	$O(n^3)$	一般	操作日志

由于文献 [7,8,10] 并未给出算法时间复杂度的推导, 并且该算法时间复杂度取决于模型的学习率、批次大小等超参数, 因此无法准确计算文献 [7,8,10] 的算法时间复杂度. 但通常而言, 在训练集数据量相同的情况下, 文献 [7,8,10] 的方法的训练时间更长.

3 滑动窗口判别法

由于数据集可能包含噪音, 直接通过单个子命令序列是否属于某个聚簇来判断是否产生异常是不可取的, 因为这将导致过高的误报率. 本文提出了一种滑动窗口判别法, 用于判断是否真正产生异常. 该方法定义了一个大小为 n (n>1) 的窗口, 每次以步长为 1 的幅度向前滑动, 当窗口内出现m个或以上异常序列 ($m<n$), 则判定产生了异常. 图 2 展示了 n 为 5, m 为 3 的滑动窗口判别法的工作方式.

3.1 有效性证明

首先证明滑动窗口判别法可以有效提高算法的识别率. 假设 TL-IDA 算法对单个子命令序列的识别率为 P_{rcg}, 通过单个子命令序列判断是否产生异常的识别率 $R_{sin}=P_{rcg}$, 而使用滑动窗口判别法的识别率 $R_{win}=\sum_{i=m}^{n} C_n^i P_{rcg}{}^i (1-P_{rcg})^{n-i}$ 当终端输入全为异常序列, 且 $P_{rcg}=0.7$, $n=7$, $m=4$ 时, 代入上述式子可得, $R_{sin}=0.7$,

R_{win}=0.874 1, R_{sin}<R_{win}, 因此使用滑动窗口判别法的识别率更高, 算法性能更好.

然后证明滑动窗口判别法可以有效降低算法的误报率. 假设 TL-IDA 算法对单个子命令序列的误报率为 P_{fa}, 通过单个子命令序列判断是否产生异常的误报率 $R_{sin}=P_{fa}$, 而使用滑动窗口判别法的误报率 $R_{win}=\sum_{i=m}^{n} C_n^i P_{fa}{}^i(1-P_{fa})^{n-i}$. 当终端输入全为正常序列, 且 P_{fa}=0.4, n=7, m=4 时, 代入上述式子可得, R_{sin}=0.4, R_{win}=0.2897, R_{sin}>R_{win}, 因此使用滑动窗口判别法的误报率更低, 算法性能更好.

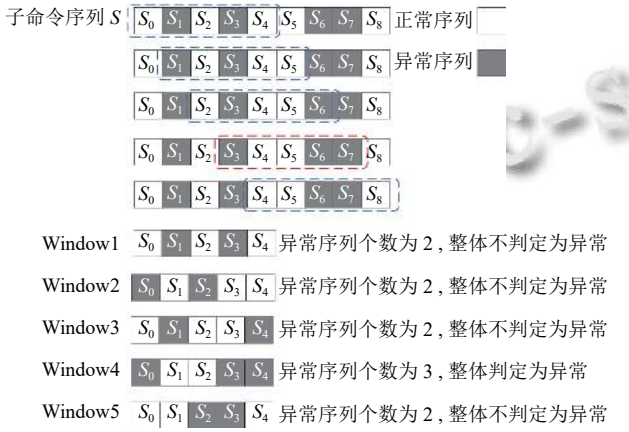

图 2 n 为 5, m 为 3 的滑动窗口判别法的工作方式

3.2 滑动窗口判别法设计

下面将通过 3 个公式介绍滑动窗口判别法. 首先是判断单个子命令序列 s 是否属于某个聚簇的公式, 其表达式如下:

$$F_1(s,c)=\begin{cases}1, & dist(s,c)\leqslant T_{dist}\\ 0, & dist(s,c)>T_{dist}\end{cases} \tag{1}$$

其中, c 为聚簇的中心. 该公式规定, 如果子命令序列 s 与聚簇的中心 c 的距离小于或等于 T_{dist}, 则该序列属于 c 代表的聚簇, 此时 F_1 的值为 1, 否则为 0.

有了式 (1) 之后, 可以进一步判断单个子命令序列在模型中是否异常, 判断公式如下:

$$F_2(s)=\begin{cases}1, & \sum_{i=1}^{K} F1(s,ci)<1\\ 0, & \sum_{i=1}^{K} F1(s,ci)\geqslant 1\end{cases} \tag{2}$$

其中, K 为模型的聚簇个数. 该函数使用式 (1) 依次判断 s 是否属于某个聚簇并累加式 (1) 的值, 如果 s 属于某个聚簇, 该累加值大于或等于 1, $F_2(s)$=0, 则 s 为正常序列. 反之, 当 $F_2(s)$=1 时, 表明 s 不属于任何聚簇, 则 s 为异常序列.

最后, 通过式 (2) , 可以得到滑动窗口判别法的公式, 该公式表达如下:

$$F_3(S_n)=\begin{cases}1, & \sum_{i=1}^{n} F_2(si)\geqslant m\\ 0, & \sum_{i=1}^{n} F_2(si)<m\end{cases} \tag{3}$$

其中, S_n 表示过去n个子命令序列集. 该公式规定, 对于过去n个子命令序列, 依次使用式 (2) 检测单个子命令序列是否异常, 再统计 S_n 中被标记为 1 的序列个数. 如果在过去 n 个子命令序列中, 有 m 个或以上序列被标记为异常, 则判定产生了异常.

本小节介绍了滑动窗口判别法的工作原理, 并介绍了该方法主要的 3 个函数. 首先, F_1 判断单个子命令序列是否属于某个聚簇, 它的衡量标准是该序列与聚簇中心的距离是否小于 T_{dist}. 然后, F_2 在 F_1 的基础上判断单个子命令序列是否为异常序列, 它通过判断子命令序列是否属于模型中的任意一个聚簇完成. 最后, F_3 查看了过去 n 个子命令序列的整体异常情况来判断是否产生异常. 本文通过推导证明了滑动窗口判别法比单个子命令序列为标准的方法性能更好.

4 实验结果及分析

本文使用普渡大学提供的数据集进行实验. 该数据集包含 9 组经过清理的用户数据, 这些数据来自普渡大学的 8 位 UNIX 计算机用户在长达 2 年的时间里的历史命令. 其中, USER0 和 USER1 是由同一个用户在不同的平台和不同的项目上工作生成的. 除非特别声明, 默认选择 USER0 的日志文件作为训练集, 其余的文件作为测试集. 同时, 为了便于理解, 采用聚簇的样本包含率而不是长度来描述 T_{dist}. 在本次实验中, 设置子命令序列长度为 5, 设置 T_{dist} 为包括约 85% 样本的距离, 设置滑动窗口 n 大小为 7, m 为 4.

4.1 识别率和误报率的实验结果

为了便于与现有的终端级入侵检测相关文献作对比, 本文采用和文献 [14,15] 相同的识别率和误报率来衡量 TL-IDA 算法的有效性. 该评价指标如下:

(1) 识别率 $=\dfrac{\text{用户A样本在模型B的被检出数}}{\text{用户A样本数}}$, 其中, 模型 B 通过用户 B 的样本训练得到.

(2) 误报率 $=\dfrac{\text{用户A样本在模型A的被检出数}}{\text{用户A样本数}}$, 其中, 模型 A 通过用户 A 的样本训练得到.

为了测试算法的识别率, 选择除 USER1 外的其余文件作为测试集, 用它们在以 USER0 为训练集的模型上的被检出率来衡量算法的识别率. 排除 USER1 的原因是它和 USER0 来自同一个用户. 在实验中, 测试集被平均分成 5 个片段, 依次对这些片段进行单独测试, 实验结果是所有单独测试的平均值. 得到的实验数据如图 3 所示.

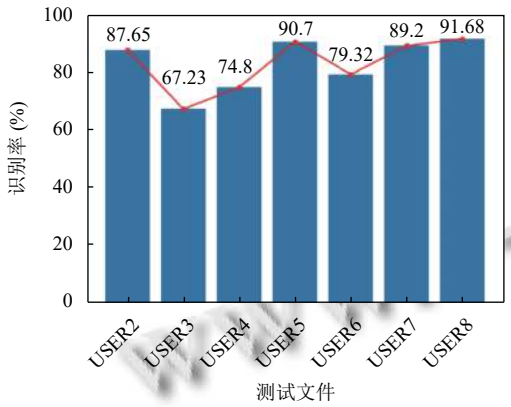

图 3 识别率的实验结果

本文采用 5 折交叉验证法测试算法的误报率. 单个用户文件被分割成 5 个片段, 每次选择 1 个片段用于测试, 其余 4 个片段用于训练. 测试单个文件时, 依次更换不同的片段作为测试集, 得到该文件的单次完整实验数据. 本次实验对每个用户文件进行 10 次完整测试, 实验结果是每个完整测试的平均值. 图 4 展示了实验数据.

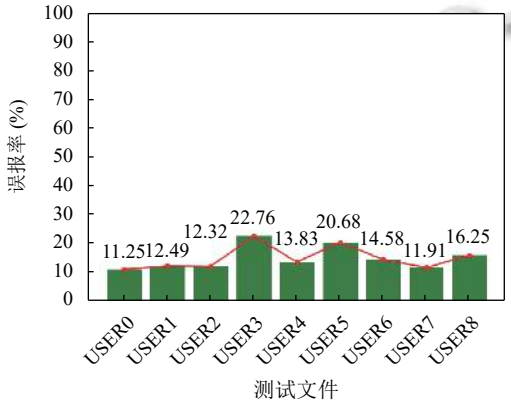

图 4 误报率的实验结果

4.2 子命令序列长度的影响

本文采用控制变量法测试不同子命令序列长度对应的识别率和误报率. 对于识别率, 选取 USER0 为训练集, USER8 为测试集. 对于误报率, 选取 USER0 作为 5 折交叉验证的数据集. 实验数据如表 3 所示.

表 3 子命令序列长度的实验结果

长度	识别率 (%)	误报率 (%)
2	76.50	60.91
3	81.68	32.55
4	84.76	17.71
5	87.61	14.25
6	76.01	15.69
7	71.86	16.79
8	71.98	13.98

分析数据可以发现, 随着子命令序列长度不断增加, 算法的识别率不断上升, 当子命令序列长度为 5 时达到峰值, 随后不断下降. 而误报率在子命令序列长度为 2 时非常高, 随着子命令序列长度增加, 它先迅速下降, 然后趋于稳定. 这是因为当长度过短时, 子命令序列的颗粒度太细, 几乎不包含用户操作的有效信息, 因此识别率和误报率的随机性很大. 当长度过长时, 识别率下降的原因是因为失去了用户操作的“时间局部性”信息, 而误报率保持在一个相对稳定的数值是因为特征向量中的 x_3 相对稳定, 使得特征向量被正确聚类. 图 5 更为直观地显示了实验结果.

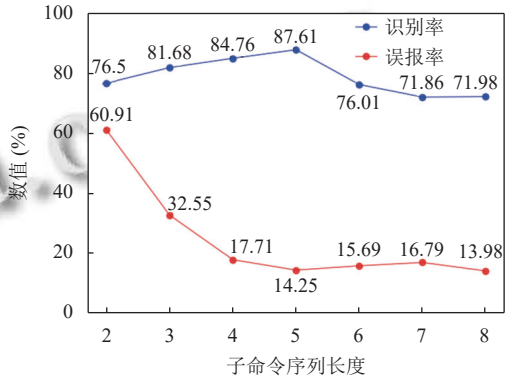

图 5 子命令序列长度影响的实验结果

4.3 实验结果对比

在同样的数据集下, TL-IDA 算法的异常检测率平均为 83% 而误报率低至平均 15%, 该成果对比 Lane 等人[14] 的成果 (分别为 74% 和 28%) 有较大的提升. 对比 Sequeira 等人[15] 的方法 (分别为 80% 和 15%), TL-IDA 算法除了进一步提升了识别率和降低了误报率之外, 同时考虑了命令表达多样性的情况. 图 6 直观地展示了实验结果对比.

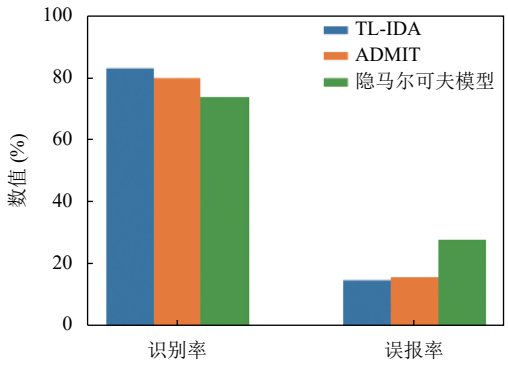

图 6 实验结果对比

4.4 聚簇类别分析

对终端的操作日志进行聚类, 应该至少得到用户常规行为类别 (I 类) 和用户偶然行为类别 (II 类) 2 种聚簇, 而代表用户新行为类别 (III 类) 和入侵者行为类别 (IV 类) 的聚簇可能不存在, 下面讨论 4 种聚簇类别的特点和分布.

用户常规行为类别 (I 类). 这类聚簇代表的是用户最常用的命令. 根据统计显示, 用户的常规行为占比是最高的, 因此这类聚簇应该是样本最多的聚簇, 并且大多数样本的特征向量 x_1、x_2 和 x_3 均不为 0.

用户偶然行为类别 (II 类). 这类聚簇代表的是由用户误操作和用户罕见操作组成的命令. 前者是因为用户输入了正确的命令同时输入了错误的可选参数导致 (错误的命令输入已经在数据预处理时删除了). 这导致特征向量的 x_2 和 x_3 接近 0 而 x_1 不为 0. 后者的样本数量占比较少, 由于命令出现次数较少, 因此特征向量表现为 x_1 和 x_3 接近 0 而 x_2 不为 0.

用户新行为类别 (III 类). 这类聚簇代表的是用户在终端上很少执行的一些命令, 可能由用户在新的项目或在不同的环境工作导致. 区分 III 类聚簇和 IV 类聚簇是提高识别率、降低误报率的关键. 在初始时, 用户新行为会被归类为 II 类聚簇, 随着命令使用次数增加, 它会逐渐往 I 类聚簇迁移, 当使用次数足够多时, 它将不再是“新行为”, 而是“常规行为”, 因此这类聚簇通常会分布在第 I 类和第 II 类聚簇之间.

入侵者行为类别 (IV 类). 这类聚簇代表的是入侵者使用的命令, 是需要识别的主要类别. 由于入侵者的操作习惯和合法用户的操作习惯很可能不一致, 因此它们的命令序列也会有差别, 所以这类聚簇的特点是样本数量较少并且与 I 类聚簇的距离较远.

5 结束语

本文研究的是终端级异常检测算法, 为了提升异常检测技术的识别率, 本文提出了一种终端级入侵检测算法 TL-IDA. 对比现有的终端级入侵检测算法, TL-IDA 算法的时间复杂度为 O(n^2) 级别, 为同类算法最好. 此外, 该算法的异常检测率平均为 83% 而误报率低至平均 15%, 在相同的数据集下优于文献 [14,15] 报告的结果, 并且在训练时不需要为训练集设置标签. 此外, TL-IDA 算法的另一个优势是需要设置的超参数更少, 这些超参数通常需要大量的测试才能确定, 更少的超参数意味着更短的训练时间. 最后, 本文归类了聚簇的类别并且总结了它们的特点, 这些归类将更有利于安全分析人员分析系统的行为. 下一步将考虑使用生成对抗网络把攻击行为伪造成正常行为, 通过改进算法识别这些伪装攻击, 提升入侵检测算法对伪装攻击行为的识别能力, 进一步完善入侵检测算法.

参考文献

1 Axelsson S. The base-rate fallacy and the difficulty of intrusion detection. ACM Transactions on Information and System Security, 2000, 3(3): 186–205. [doi: 10.1145/357830.357849]

2 Ahmim A, Maglaras L, Ferrag MA, *et al*. A novel hierarchical intrusion detection system based on decision tree and rules-based models. Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS). Santorini: IEEE, 2019. 228–233.

3 Lin WC, Ke SW, Tsai CF. CANN: An intrusion detection system based on combining cluster centers and nearest neighbors. Knowledge-based Systems, 2015, 78: 13–21. [doi: 10.1016/j.knosys.2015.01.009]

4 Cui Y, Sun YP, Hu JL, *et al*. A convolutional auto-encoder method for anomaly detection on system logs. Proceedings of 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Miyazaki: IEEE, 2018. 3057–3062.

5 Lin SZ, Shi Y, Xue Z. Character-level intrusion detection based on convolutional neural networks. Proceedings of 2018 International Joint Conference on Neural Networks (IJCNN). Rio de Janeiro: IEEE, 2018. 1–8.

6 陆悠, 李伟, 罗军舟, 等. 一种基于选择性协同学习的网络用户异常行为检测方法. 计算机学报, 2014, 37(1): 28–40.

7 Cabrera JBD, Lewis L, Mehra RK. Detection and classification of intrusions and faults using sequences of system calls. ACM SIGMOD Record, 2001, 30(4): 25–34. [doi: 10.

1145/604264.604269]

8 Lee W, Stolfo SJ. Data mining approaches for intrusion detection. Proceedings of the 7th USENIX Security Symposium. San Antonio: USENIX Association, 1998. 1–16.

9 宋海涛, 韦大伟, 汤光明, 等. 基于模式挖掘的用户行为异常检测算法. 小型微型计算机系统, 2016, 37(2): 221–226. [doi: 10.3969/j.issn.1000-1220.2016.02.006]

10 Ryan J, Lin MJ, Miikkulainen R. Intrusion detection with neural networks. Proceedings of the 10th International Conference on Neural Information Processing Systems. Denver: MIT Press, 1997. 943–949.

11 DuMouchel W. Computer intrusion detection based on Bayes factors for comparing command transition probabilities. Technical Report, Washington: National Institute of Statistical Sciences, 1999.

12 Schonlau M, DuMouchel W, Ju WH, *et al*. Computer intrusion: Detecting masquerades. Statistical Science, 2001, 16(1): 58–74.

13 Lane TD. Machine learning techniques for the computer security domain of anomaly detection [Ph.D. Thesis]. West Lafayette: Purdue University, 2000.

14 Lane T, Brodley CE. Temporal sequence learning and data reduction for anomaly detection. ACM Transactions on Information and System Security, 1999, 2(3): 295–331. [doi: 10.1145/322510.322526]

15 Sequeira K, Zaki M. ADMIT: Anomaly-based data mining for intrusions. Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Edmonton: ACM, 2002. 386–395.

16 Ahamed Shafeeq BM, Hareesha KS. Dynamic clustering of data with modified K-means algorithm. Proceedings of 2012 International Conference on Information and Computer Networks. Singapore: IACSIT Press, 2012. 221–225.

(校对责编: 孙君艳)