第32卷 第16期 计 算 机 工 程 2006年8月
Vol.32 № 16 Computer Engineering August 2006
·软件技术与数据库· 文章编号：1000—3428(2006)16—0060—03 文献标识码：A 中图分类号：TP311

基于栈结构的支持度计算及其应用

费洪晓，郭球辉，刘 勇，谢文彪，裘方敏
(中南大学信息科学与工程学院，长沙 410075)

摘 要：提出了一种间断序列分析方法，实现了应用序列模式分析提取网络环境下用户正常行为模式。并在此基础上提出了基于栈结构的支持度计算算法，该算法仅对数据库进行一次扫描，即可实现候选序列的计数操作。实验证明该算法具有较好的实用性。
关键词：间断序列分析；入侵检测；栈结构

Support Computing Based on Stack Structure and Its Application

FEI Hongxiao, GUO Qiuhui, LIU Yong, XIE Wenbiao, QIU Fangmin
(School of Information Science and Engineering, Central South University, Changsha 410075)

【Abstract】 The paper presents a break sequences analysis method for such situation and implements extracting the users normal behavior patterns by using sequential analysis in the network environment. Then it presents a support computing algorithm based on stack structure, which can count the candidate sequences by scanning the database only once. The tests show that the algorithm is practical.
【Key words】 Break sequences analysis；Intrusion detection；Stack structure

入侵检测系统(Intrusion Detection Systems, IDS)是指对于面向计算机资源和网络资源的恶意行为识别和响应的网络安全系统，包括技术、人、工具3方面的一个整体[1,2]。然而随着攻击技术、方法与攻击工具的发展和变化，传统的依靠手工和经验的方式建立的基于专家系统的IDS在扩展性、适应性、分布式攻击检测和数据协同分析方面已经暴露出明显的不足。近年来，新的入侵检测产品不断出现，新的入侵检测方法理论不断被提出，尤其是将数据挖掘理论引入入侵检测系统，为入侵检测系统的研究开拓了新的领域[2]。

本文利用数据挖掘中的序列分析，结合入侵检测实际运行环境对网络和主机审计记录挖掘，挖掘出用户的行为模式，提出了对间断序列模式的挖掘，由于很多入侵手段是在经典的攻击步骤中插入一些欺骗动作[3]，通过间断序列模式的分析对这种入侵就能很好地检测。本文还就间断序列分析特性提出了一种基于栈结构的序列模式的支持度计算，该算法仅对数据库进行一次扫描，即可实现候选序列的计算计数操作，由于使用的是栈结构，其存储空间也较少，因此实用性较高。

1 序列模式挖掘在入侵检测中的应用

1.1 序列挖掘的相关定义

序列就是由不同的元素按顺序有序排列的项集。

假定项集中的项由一些连续整数代替，这样一个项集i可以表示$(i_1,i_2,\ldots,i_m)$，i_j 代表一个项。一个序列s则可以表示为$<s_1,s_2,\ldots,s_n>$，s_j 代表一个项集。

定义1 假定两个序列$a<a_1,a_2,\ldots,a_n>$ 和$b<b_1,b_2,\ldots,b_m>$，如果存在整数$i_1<i_2<\ldots<i_n$且$a_1\subseteq b_{i1}$，$a_2\subseteq b_{i2},\ldots,a_n\subseteq b_{in}$ ，则称序列a 包含于序列b。在一个序列集中如果序列s不包含于任何其他序列，则称序列s为最大项序列[2]。

序列模式挖掘就是找出所有的频繁序列，即该子序列在时序数据库中的出现频率不低于用户指定的最小支持度阈值。但目前多数基于频繁序列挖掘的算法都着重研究连续的序列挖掘，即序列模式的元素在时序中必须是连续的，这意味着序列模式具有完全的规律性。如在某段时间内，有数据特征A出现，然后出现了特征B，而后特征C又出现了，即序列A→B→C，这条序列在整个事务库中的支持度大于最小支持度阈值，则是一个频繁序列，但是如果出现A→B→D→E→C这样的序列则不能用连续的序列挖掘方法进行。在现实生活中，存在大量的这种间断的序列模式，因此，我们提出一种时序模式的挖掘。将某个事务的所有操作转换成某个时间段内的操作序列，即一条时序记录。这样就形成了整个时序数据库，时序数据库是由一条条时序记录组成的，挖掘出时序记录中的间断的频繁模式。其支持度计算如定义2。

定义2 给定时序数据库$S=\{S_1S_2\ldots S_n\}$及序列s，如果序列s在时序S_i中的匹配数为t_i，定义时序数据库S对序列s的支持度$support(s)=\Sigma t_i$。

对某个时序数据库，给定最小支持度，则实际支持度大于给定的最小支持度的序列称为频繁序列。挖掘序列模式的任务就是在给定的用户时序数据库中，根据指定参数发现所有的频繁序列，由挖掘出的频繁序列生成相应用户的正常行为模式以及各种攻击方式的攻击行为模式[3]。

1.2 序列挖掘的应用

为了开发系统化自动化的 IDS，在近年来发展的新型智能化入侵检测技术中，数据挖掘技术运用得很多。将数据挖掘技术应用于处理网络审计数据，从中分别提炼出正常和入侵情况下的用户行为模式，再由生成的模式库匹配IDS所采集到的数据以捕获网络入侵。

在入侵检测中，根据孤立的事件或属性判断是否是入侵行为是很困难的，必须把用户在一次登录甚至几次登录过程

基金项目：国家自然科学基金资助项目(60173041)；湖南省自然科学基金资助项目(02JJY2094)
作者简介：费洪晓(1967—)，男，硕士、副教授，主研方向：网络管理与网络安全；郭球辉、刘 勇、谢文彪、裘方敏，硕士
收稿日期：2005-11-14 **E-mail：**hxfei@csu.edu.cn

中的行为作为一个整体分析。因此，运用数据挖掘中的序列模式分析方法对对审计数据进行序列挖掘，即从网络事件数据库中挖掘出用户正常行为以及入侵行为的频繁序列模式。可以根据用户执行的 shell 命令以及一些习惯性操作来判断用户的身份，假定某用户的登录主机固定，登录时间为下午，经常执行 su 命令以获取 root 权限，访问系统的口令文件，从而可以判断出该用户为系统管理员，如果某天该用户突然在晚上登录，则可以视为一次异常。

2 数据预处理

挖掘用户的正常行为模式，首先对审计记录进行预处理，本文以用户在 Telnet 会话过程中提交的 shell 命令为例来进行分析。首先针对每个用户构建其正常的行为模式(history)，然后在实际检测过程中，对用户的每一次 Telnet 会话都挖掘出其中包含的行为模式(present)，将当前模式和历史模式进行比较，计算出二者的相似度，相似度越高，说明该用户的当前行为模式与历史模式越吻合，出现异常的可能性也就越小。

对数据预处理主要分两步：收集用户 Telnet 会话记录，清除脏数据、无用数据和根据会话记录生成时序数据库。

一般地，通常是用Bro程序来对网络中传输的数据包进行分析的，为了进行用户行为的异常检测，对Bro程序中Telnet会话的事件处理程序进行修改，可以获取用户向Telnet主机递交的所有shell命令。收集用户每次登录使用的shell命令集，清洗脏数据和无用数据，按照操作时间顺序处理，生成如表 1 所示的审计记录[4]。

表 1 用户 shell 命令集

UserName	Timestamp	Command	Param
A	10:15:30	vi	.c
A	10:17:12	su	
A	10:20:30	gcc	- g – o
	…	…	…
A	10:25:10	gdb	

由于我们是挖掘时序模式，因此将一个用户一次登录所有的操作综合看成一个事件组成的时序，将这个时序用一个编号用以标注，作为时序数据库中的一条记录。对于表 1 生成的时序是(A,1，“ vi,su,gcc,…,gdb”)。将一个事务中所有用户的操作序列，按照此方法进行预处理，建立所有用户的审计记录，对其进行时序分析，挖掘所有用户的行为模式。

3 基于栈结构的序列分析

3.1 算法描述

序列模式的挖掘框架与采用关联规则挖掘Apriori算法相似的思想，将上轮产生的频繁序列作为本轮的候选序列。首先扫描数据库，统计出现元素的个数，将支持度大于最小支持度阈值作为频繁一项集L_1，然后产生候选 2 项集C_2，以此类推，直到产生所有的频繁序列。在此给出序列挖掘框架：

输入：时序数据库 S，最小支持度 min_support，序列模式最大长度 n

输出：频繁序列集

```
Begin
{
L1={large 1-sequences};                    //查找频繁 1 项序列
For (k=2;Lk-1≠Ø;k++)do
{
Ck=apriori-candidate-generation(Lk-1); //从上轮的Lk-1输出本轮的
//候选集Ck
Lk=ComputerSupport(Ck) //检查候选集，输出满足最小支持度的
//频繁集Lk
      }
Answer=Maximal sequences in ∪kLk    }
End
```

算法对各个函数的描述如下：

(1) 查找频繁 1 项序列：扫描整个时序数据库，统计出现的元素个数，将支持度大于min_support的元素加入到频繁L_1序列集中。

(2) apriori-candidate-generation(L_{k-1})函数，从上轮的L_{k-1}输出本轮的候选集C_k。构造频繁C_{k-1}集分为两步：连接步和剪枝步。

连接阶段从频繁L_{k-1}序列集中生成C_k序列候选集。其操作是：对于两个频繁L_{k-1}序列集中的序列S_1和S_2，如果去掉序列模式S_1的第一个元素与去掉序列模式S_2的最后一个元素所得到的序列相同，则可以将S_1与S_2进行连接，即将S_2的最后一个元素添加到S_1末尾。例如频繁S_1序列abc，S_2序列bcd，经过连接后生成候选序列abcd。

裁减阶段用于削减候选集的规模。在序列挖掘中具有与关联规则挖掘Apriori算法相似的特性：如果一个L_k序列是频繁的，其所有的子序列也必定是频繁的。因此，候选集的裁减与Apriori算法相似：若某候选序列的某个子序列不是频繁序列，则此候选序列不可能是频繁序列，将它从候选序列集中删除。

(3) ComputerSupport(C_k)函数：在生成C_k序列候选集后，需要检验序列是否是频繁的。方法是扫描时序数据库，对每条时序计算候选C_k序列S的匹配数，并统计出候选序列的支持度是否大于min_support，删除不满足条件的候选序列，生成频繁L_k。

通过扫描数据库中每条时序记录，查找每条候选序列的在每条时序数据库记录中出现的次数，将候选序列在所有时序记录中出现的次数统计起来，计算其支持度。传统的计数算法都是多次扫描数据库，使得算法效率很低下，特别是用于在线统计时，不能满足实时的要求。考虑到扫描数据库的因素，并且挖掘的不仅仅是连续的序列模式，还有注重间断序列模式的挖掘，为此，提出了一种基于栈的思想的计数统计算法。其主要步骤如下：

(1)对于每一条候选序列 S 有 n 个项，即 S{S1,S2, S3,…,Sn}。构造 n-1 个栈 stack[i](1≤i≤n-1)用以为存放候选序列中的前 n-1 个项。初始化匹配个数 MatchNum 为 0。

(2)扫描时序数据库记录，扫描到与候选序列中某个项 Sj(1≤j≤n)相同的项，则将 Sj 压入栈 stack[j]，即 push (stack[j],Sj)。

(3)若扫描到某个项 Sm(1≤m≤n)与 Sn 相同,并且栈 stack[i](1≤i≤n-1)均不为空，则说明找到一个序列与该候选序列匹配，就将 stack[i](1≤i≤n-1)均出栈，并且 MatchNum 加 1。

(4)继续扫描，直到扫描结束。

当统计完一条候选序列的个数后，将栈清空，MatchNum 重新设置为 0，进行下一条候选序列的计数操作。

其支持度计算算法描述如下：

```
For(每一个候选序列)     //每条长度为n,即有n个项,依次为//S1,
S2,...,Sn
{
 MatchNum=0;                    //匹配数，初始值为 0
 构造n-1 个栈，stack[i];   //1≤i≤n-1,分别用来存时序列中出
//现的S1，S2...Sn
 For(每一条时序列)            //扫描每一条时序列
 {          if(在时序列中能找到候选序列中的任一项Si)
            push(stack[i],Si);
      if(在时序列中找到Sn，且stack[i]均不为空)
      {
            pop(stack[i]);
            MatchNum++;          }
 }  }
```

3.2 挖掘算法结果实例

将栈结构引入序列模式分析，进行支持度的计算，在整个匹配计算过程中，只对数据库进行一次扫描，提高了算法

的效率，并且出栈和入栈操作也较容易，提高了算法的实用性。在生成的候选序列中，设定了最大长度，该最大长度一般只有几个到几十个，当在时序列中找到一个与候选序列匹配的项时，才为之建立一个栈，这样对存储空间也较为节省。

表 2 是一个已经经过预处理后的时序数据库片断，CommandSequence 是用户某次操作按照时间顺序处理后的命令序列。现有 3 条候选序列 cde，bc，cd，用上述算法来统计它在数据库中出现的次数。

表 2 用户行为时序数据库

UserName	LogID	CommandSequence
A	1	abcdfe
A	2	becgfdecd
...	…	…

首先，统计 cde 出现的次数。

(1)初始化两个栈，分别用以存放在时序列中扫描到的 c 和 d，初始化 MatchNum=0，见图 1。

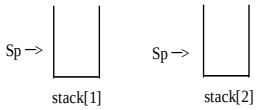

图 1 初始化两个栈

(2)依次扫描第 1 条时序列，扫描 a，b，c，将 c 入栈，继续扫描 d，将 d 入栈，继续扫描 f，e，扫描到了 e，e 为候选序列的最后一个项，并且 stack[1]和 stack[2]不为空，则 stack[1]和 stack[2]均出栈，且 MatchNum 加 1，此时 MatchNum 为 1。见图 2。

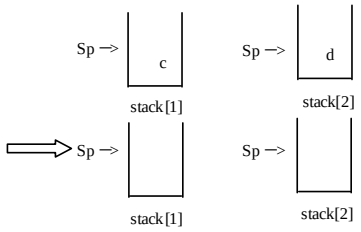

图 2 依次扫描第 1 条时序列

(3)依次扫描第 2 条时序列，扫描 b，e，c，将 c 入栈，继续扫描 g，f，d，将 d 入栈，继续扫描到 e，e 为候选序列的最后一个项，并且 stack[1]和 stack[2]不为空，则 stack[1]和 stack[2]均出栈，且 MatchNum 加 1，此时 MatchNum 为 2，扫描完毕(见图 3)。

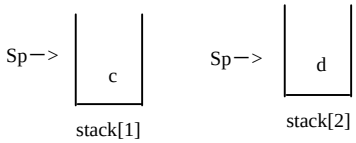

图 3 依次扫描第 2 条时序列

通过这样的扫描，得出 cde 在这个时序数据库片断中出现的次数为 2，用同样的方法可以得出候选序列 bc 和 cd 出现的次数分别是 2 和 3。

4 小结

利用数据挖掘中的序列分析，结合入侵检测实际运行环境，根据用户 Telnet 会话记录挖掘出用户的行为模式。本文提出了对间断序列模式的挖掘，并就其特性提出了基于栈结构的序列模式的支持度计算。试验证明，对间断序列模式的挖掘，更能检测出用户的异常行为。基于栈结构的支持度算法，节省了存储空间，由于减少了对数据库的访问也大大提高了效率，此外，由于计数是通过入栈和出栈来进行的，该算法的实用性和可操作性也很强。

参考文献

1 Lee W, Stolfo S, Chan P, et al. Real Time Data Mining-based Intrusion Detection [C]. The 2001 DARPA Information Survivability Conference and Exposition (DISCEX II), Anaheim, CA, 2001-06.

2 戴英侠, 连一峰, 王 航. 系统安全与入侵检测[M]. 北京 :清华大学出版社, 2002: 99-137.

3 陈望斌, 王力生, 廖根为. 基于序列模式挖掘的入侵检测技术研究[J]. 小型微型计算机系统, 2004, 25(5): 878-881.

4 宋世杰, 胡华平, 胡笑蕾等. 数据挖掘技术在网络型异常入侵检测系统中的应用[J]. 计算机应用, 2003, 23(12): 20-23.

(上接第 59 页)

(4)TCB* time_slice_schedule(TCB *tcb)时间片调度

time_slice_schedule 实现时间片调度，当该任务执行的时间超过应该连续执行的时间时，将该任务放置在就绪队列中的同一优先级下的任务最末端，并返回排在最前端的任务。

```
TCB* time_slice_schedule(TCB *tcb)
{ if (tcb != NULL && tcb->slicetime > 0 ) {
      tcb->slicecnt += TIMER_PERIOD;
      if (tcb->slicecnt > tcb->slicetime) {
         tcb = ready_queue_move_last (&ready_queue, tcb);
         /*将用尽时间片的任务放于就绪对列尾*/  }  }
 return tcb;  }
```

slicetime 为该任务可执行的时间片，slicecnt 为该任务当前实际已执行的时间，TIMER_PERIOD 为系统定时器周期。

在 T-Kernel 中的其它一些操作中也进行了任务调度，如对内核对象锁定及解锁操作等,这里就不一一介绍。

4 T-Kernel 调度策略与机制的实时性能分析

为了达到实时的目的，T-Kernel 采用了基于优先级的抢占调度算法，并可动态地改变每个任务的优先级。

一方面 T-Kernel 在任务的调度机制上引入了就绪队列这个数据结构(RDYQUE)，其中的 top_priority 字段能快速地定位就绪任务中的最高优先级任务，从而避免了遍历查找的过程，大大地减少了任务调度的时间，也避免了任务调度时间随着任务数的增多而增大。

在另一方面 T-Kernel 系统的调度时机较多，有利于保证实时进程在其截止期限之前完成；还有 T-Kernel 在调度之前先判断是否处于系统临界区之中，能在不破坏核心数据结构的前提下，使优先级较高的实时任务及时得到调度。

5 结束语

经过对 T-Kernel(Version 1.01.00)源代码分析，对其任务调度策略及机制的深入探讨，我们看到它在任务调度上采用了一些特殊的方法以改善系统的实时性，减少任务的抢占延迟，具有较强的实时处理能力。

参考文献

1 Sakamura K. T-Kernel Sepcification(Version 1.B0.02)[Z]. T-Engine Forum, 2003.

2 T-Engine Specification(Version 1.A0.00)[Z]. T-Engine Forum, 2003.

3 郑宗汉. 实时系统软件基础[M]. 北京: 清华大学出版社, 2002.

4 Liu J W S. 姬孟洛, 李 军, 王 馨等译. 北京: 清华大学出版社, 2003.