文章编号:1001 - 9081(2009)08 - 2223 - 04

基于报警序列的入侵场景自动构建

郭 帆[1],涂风涛[2],余 敏[1]
(1. 江西师范大学 计算机信息工程学院,南昌 330022; 2. 南昌师范高等专科学校 计算机系,南昌 330006)
(guofan771210@yahoo.com.cn)

摘 要:传统的入侵检测系统(IDS)由于其规则的抽象程度较低,导致一次攻击行为会产生大量重复和相关报警。研究表明,入侵场景可提供较高层次的抽象来表示攻击过程,但是已有研究方法均无法在线生成入侵场景。提出一种自动构建入侵场景的方法,将原始报警按照(源,目标)IP 对和优先级分类成不同超报警序列集合,从中挖掘频繁闭序列作为入侵场景。在 Darpa 数据集上的实验表明,该方法可以满足在线运行,并可有效发现攻击过程。

关键词:入侵检测;入侵场景;超报警序列;频繁闭序列
中图分类号: TP393.08 **文献标志码**:A

Automatic intrusion scenario construction by mining hyper-alert sequences

GUO Fan[1], TU Feng-tao[2], YU Min[3]
(1. *College of Computer Information and Engineering, Jiangxi Normal University, Nanchang Jiangxi* 330022, *China*;
2. *Department of Computer Science, Nanchang Normal Institute, Nanchang Jiangxi* 330006, *China*)

Abstract: Traditional Intrusion Detection System (IDS) always produces a great number of raw alerts for the same attack, due to lower abstract representation of the detection rules. Researchers use intrusion scenarios to describe complicated attack procedures at a high abstract level, while, to our best knowledge, none is able to produce the scenarios online. An automatic intrusion scenario construction method was proposed. According to the attributes of the raw alerts, the method firstly clustered them into different hyper-alert sequences. After that, frequent closed sequences were mined to construct scenarios. Experimental results on Darpa99 and Darpa2000 show the method can be used to run online and effectively detect attack procedures.

Key words: intrusion detection; intrusion scenario; hyper-alert sequence; frequent closed sequence

0 引言

入侵检测系统(Intrusion Detection System, IDS)是继防火墙等传统防御措施的新一代安全保障技术,通过分析与系统安全相关的数据来检测入侵活动。基于误用的 IDS 依赖预定义规则来判断相关事件是否攻击,但目前 IDS 规则的抽象级别较低,导致一次入侵过程可能引发几十甚至上百条报警。特别的,当黑客使用分布式或者协同网络攻击手段时,其攻击过程会隐身于海量报警信息中,使得安全人员难以发现黑客的真实意图。因此,如何从警报信息中分辨出黑客攻击流程并重构入侵场景,是近年来入侵检测领域的研究热点问题。

构建入侵场景是一种对 IDS 产生的报警进行再次分析的过程,主要通过报警聚合和关联算法来完成。报警聚合将针对同一安全事件产生的大量性质相同或相近的报警合并成一条报警,报警关联将属于同一攻击过程的每一步骤所产生的报警联系在一起并重建攻击过程[1]。

当前报警关联算法主要可分为三类。一是基于报警属性相似度的方法,该类方将相似的报警聚集在一起并重构入侵场景[2],Valdes 等人提出报警聚合需要比较的通用报警属性为源 IP 和端口、目标 IP 和端口、报警时间等,它把每类属性的比较结果赋予 0 到 1 之间的一个小数,通过为每类属性赋予不同权重并结合属性相似度值最终判断两个报警是否可聚合。这类算法通常可以在线运行,但是得出的关联结果往往不太可靠。二是依赖预定义的复合场景,需要安全专家事先定义好相关攻击场景。这类方法的缺点与基于误用的 IDS 一样,难以发现新的攻击过程。此类研究主要集中于设计复合场景的描述语言上,如 Cuppens 等人提出的 Lambda[3]。三是基于因果关系,Ning 等人[4]通过为每条报警定义其前因后果来完成报警关联,所有具有因果关系的攻击关联在一起,就重现了整个攻击过程,该方法可识别和报告不同攻击组合形成的新攻击过程,但是需要消耗大量计算资源,无法在线操作。Qin 等人[5]提出了基于统计时序的报警因果关联,他认为多步攻击所产生的报警属性之间具有统计的相似性,且攻击步骤存在因果关系,根据时序因果分析来获得报警关联知识。

本文提出一种利用闭频繁序列挖掘技术自动构建入侵场景的方法,首先按照同源同目标将所有原始报警分类,在此基础上把原始报警聚合成超报警,然后构建报警序列,序列中的报警按时间和优先级递增排序。频繁发生的闭序列作为基于同源同目标的入侵场景候选,对所有不同分类中的入侵场景候选进一步挖掘闭频繁序列,得到同源同目标的入侵场景。该方法无须额外先验知识,实验表明应用该方法可以在线实现报警关联并有效发现复合攻击。

1 系统架构

首先定义本文使用的一些基本概念及相关符号表示。

定义 1 原始报警 E_i。由 IDS 上报的报警,用多元组

收稿日期:2009 - 03 - 02;**修回日期**:2009 - 04 - 07。
基金项目:国家 973 计划项目(2007CB316505);江西师范大学博士基金资助项目(2007)。
作者简介:郭帆(1977 -),男,江西于都人,副教授,博士,主要研究方向:信息安全、软件体系结构; 涂风涛(1976 -),男,江西南昌人,讲师,主要研究方向:网络安全; 余敏(1964 -),女,江西南昌人,教授,博士,主要研究方向:分布式计算、信息安全。

〈*Sensor*, *Time*, *srcIP*, *dstIP*, *srcPort*, *dstPort*, *Id*, *Priority*〉表示,*Sensor*表示其来源于哪个*IDS*,*Priority*表示报警优先级,*Id*表示报警的唯一标志,*Time*表示报警时间,*srcIP*、*dstIP*、*srcPort*和*dstPort*分别表示报警的源IP、目标IP、源端口和目标端口。

定义2 超报警*ME*。由多元组〈Γ_E, $Time_{\min}$, $Time_{\max}$, *srcIP*, *dstIP*, *Id*, *Priority*〉表示,Γ_E是原始报警集合〈$E_1,\cdots,E_n$〉,$\forall i,j,1 \leqslant i \leqslant j < n, E_i, E_j \in \Gamma_E$,有($ME.Time_{\min} \leqslant E_i.time \leqslant E_j.time \leqslant ME.Time_{\max}$) ∧ ($E_i.srcIP = E_j.srcIP = ME.srcIP$) ∧ ($E_i.dstip = E_j.dstip = ME.dstIP$) ∧ ($E_i.Id = E_j.Id = ME.Id$) ∧ ($E_i.Priority = E_j.Priority = ME.Priority$),$Time_{\min}$和$Time_{\max}$表示$\Gamma_E$中原始报警的最早和最晚发生时间,超报警中所有原始报警的源IP、目标IP、唯一ID号均相同。

定义3 超报警序列*SME*。由多元组〈*TimeGap*, *srcIP*, *dstIP*, Γ_{ME}〉表示,Γ_{ME}是超报警集合〈$ME_1,\cdots,ME_n$〉,$\forall i,j,1 \leqslant i \leqslant j \leqslant n$, $ME_i, ME_j \in \Gamma_{ME}$, *SME*满足($ME_i.Priority \leqslant ME_j.Priority$) ∧ ($ME_i.srcIP = ME_j.srcIP = SME.srcIP$) ∧ ($ME_i.dstIP = ME_j.dstIP = SME.dstIP$) ∧ ($ME_j.Time_{\max} - ME_i.Time_{\min} < SME.TimeGap$),即序列按照报警递增排序,且序列所包含的任何两条超报警,其最大时间间隔不能超过*TimeGap*所定义的时间阈值,所有超报警的源和目标IP均相同。

定义4 入侵场景*AS*。由四元组〈*TimeGap*,Γ_{id}, Γ_{AS}, Γ_{AS}^{-1}〉表示。*TimeGap*表示该场景跨越的最大时间间隔,$\Gamma_{id} = \{Id_1 \rightarrow Id_2 \rightarrow \cdots \rightarrow Id_n\}$,$\forall 1 \leqslant i \leqslant j \leqslant n, Id_i.Priority \leqslant Id_j.Priority$,if $i \neq j$,then $Id_i \neq Id_j$,场景中不存在相同ID号,场景中的报警按优先级递增排序。Γ_{AS}表示场景*AS*可能关联的场景集合,$\Gamma_{AS} = \{AS_1,\cdots,AS_n\}$,$\Gamma_{AS}^{-1}$表示所有可以关联场景*AS*的场景集合$\Gamma_{AS}^{-1} = \{AS_{m+1},\cdots,AS_{m+k}\}$。

定义5 场景实例$Inst_{as}$由三元组〈SME_i, Γ_{AS},*Time*〉表示,*Time*是实例中最近一条原始报警时间,SME_i是超报警序列,$\forall AS \in \Gamma_{AS}$,满足:$SME_i.TimeGap \leqslant AS.TimeGap$;$\forall 1 \leqslant j \leqslant n, ME_j \in SME_i.\Gamma_{ME} \rightarrow ME_j.Id \in AS.\Gamma_{id}$;假设$AS.\Gamma_{id} = \{Id_1 \rightarrow Id_2 \rightarrow \cdots \rightarrow Id_m\}$,$\forall 1 \leqslant i < j \leqslant n, \exists 1 \leqslant k < l \leqslant m$, $ME_i.Id = Id_k, ME_j.Id = Id_l$,即$SME_i$不存在重复超报警。$\Gamma_{AS}$是$SME_i$可能匹配的场景集合。

定义6 场景子序列$\triangleright SubSeq(AS_1, AS_2)$。当且仅当$AS_1.\Gamma_{id} = \{Id_{11},\cdots,Id_{1n}\}$, $AS_2.\Gamma_{id} = \{Id_{21},\cdots,Id_{2m}\}$, $\forall Id_{1i} \in AS_1.\Gamma_{id}$, $\exists j, j \geqslant i$, $Id_{2j} \in AS_2.\Gamma_{id} \wedge Id_{1i} = Id_{2j}$。

整个系统体系结构如图1所示。

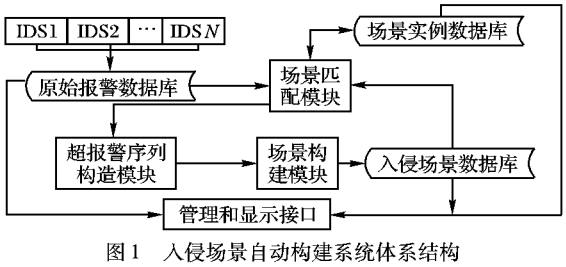

图1 入侵场景自动构建系统体系结构

首先来自各个IDS的原始报警经过统一格式转换后,按报警产生的时间顺序放入原始报警数据库中。场景匹配模块初始化时从场景数据库中将所有场景装入内存,运行时从原始报警数据库中按时间顺序取出原始报警,如果该报警是已有场景实例的一部分则更新实例的状态,如果不是则创建新的实例。原始报警最后由超报警序列模块处理,该模块负责将报警分类到不同源和目标IP对中,并构造超报警序列,经过一定时间周期(或者报警总数达到某个阈值)后,将所有超报警序列提交给场景构造模块处理,场景构造模块使用闭频繁序列挖掘技术得到频繁报警序列作为入侵场景。

2 算法描述

自动场景构建系统的算法框架如下:

输入:E_k、Γ_s、Γ_{SME}、Γ_{inst}和*Thr*。

输出:Γ_s、Γ_{SME}、Γ_{inst}。

```
AutoScenario( E_k, Γ_s, Γ_SME, Γ_inst, Thr )
  if ( Thr.EventCount >= Thr.EventThr ||
    GetCurrentTime( ) - Thr.Time >= Thr.TimeThr)
    UpdateScenario( Γ_s, Γ_SME )
    Γ_SME = ∅; Γ_inst = ∅;
    Thr.EventCount = 0; Thr.Time = GetCurrentTime( )
  endif
  Thr.EventCount ++
  if ( ! ExistInstance( Γ_inst, E_k ))
    CreateInstance( Γ_s, Γ_inst, E_k )
  endif
  UpdateSMESet( Γ_SME, E_k );
```

系统初始化时,Γ_s、Γ_{SME}和Γ_{inst}均为空集。Γ_s只有在报警数量或者时间周期超过一定阈值时才会被更新,每条原始报警如果可以与某个场景匹配,则要么是已有实例的一部分要么创建新实例。每条原始报警最后都将加入到超报警序列中,用于场景自动更新,每次更新Γ_s时,Γ_{inst}和Γ_{SME}将重新初始化为空集。下面分别描述各个具体算法过程。

2.1 超报警序列

对于每条原始报警E_k,系统首先寻找所有满足两个条件的SME_i,并将其放入集合中:1)SME_i的源和目标IP与E_k必须相同;2)SME_i的最小时间与E_k的时间差不能超过给定阈值。这样做的主要目的是防止超报警序列变得无限长。如果为空集,即没有找到相应的*SME*,则通过AllocateME和AllocateSME过程分别新建立一条ME_j和SME_m,其中ME_j仅包含E_k一条报警,而SME_m仅包含ME_j一条超报警。否则从选择$Time_{\max}$属性最大的SME_j,并考虑其中最后一条超报警ME_n。超报警序列构建算法形式描述如下:

```
UpdateSMESet( Γ_SME, E_k, Thr )
  Γ'_SME = {SME_i ∈ Γ_SME:
    SME_i.srcIP = E_k.srcIP ∧ SME_i.dstIP = E_k.dstIP ∧
      E_k.Time - SME_i.Time_min < SME_i.TimeGap}
    if ( Γ'_SME ≠ ∅ )
      SME_k = Max_Tmemax( Γ'_SME )
      ME_n = Last element of SME_k.Γ_ME
      if ( E_k.Id = ME_n.Id && E_k.Time - ME_n.Timemin <
        Threshold.METimeGap)
        ME_n.Γ_E = ME_n.Γ_E ∪ { E_k }   // 聚合进已有超报警
        ME_n.Time_max = E_k.Time
      else                                  //创建新超报警
        ME_j = AllocateME( E_k )
        if ( E_k.Priority >= ME_n.Priority )
          SME_k.Γ_ME = SME_k.Γ_ME ∪ { ME_j }
        else
          SME_m = AllocateSME( ME_j )
          Γ_SME = Γ_SME ∪ {SME_m}
        endif
      endif
    else                  //如果报警的源目标IP对没有处理过
      ME_j = AllocateME( E_k );
      SME_m = AllocateSME( ME_j );
      Γ_SME = Γ_SME ∪ {SME_m};
```

endif

如果 E_k 想要被聚合进 ME_n,则 E_k 必须满足:1) 时间不能超出阈值;2) 报警 ID 必须与 ME_n 相同。如不能被聚合,则需要新分配一条仅包含 E_k 的超报警 ME_{n+1},此时决定 E_k 是否可以加入到超报警序列 SME_j 的判断条件是 E_k 的优先级属性。根据 Qin 的假设[5],大部分复合攻击都是按照报警优先级顺序有序进行,比如攻击者通常按照探测系统、扫描系统漏洞、攻击漏洞、放置后门和清除现场等步骤进行,这些攻击步骤对应的报警优先级递增有序。因此我们构造的超报警序列,满足所有超报警按优先级顺序递增。如果 E_k 的优先级低于 ME_n,则认为 E_k 是另一次攻击的开始,因此另外构造一个新的 SME,仅包含 ME_{n+1}。超报警序列算法最终产生严格满足定义 3 要求的 SME 集合。在实现时按 IP 对保存各 SME 序列,并记录 $Time_{max}$ 属性值最大的 SME 序列,则算法复杂度为 $O(1)$。

2.2　场景构建算法

场景构建算法主要分为 3 个步骤,形式描述如下:

```
UpdateScenario( Γs, ΓSME )
  for each ( IP1, IP2 ) do                                  //步骤 1
    BIDE( Γas(IP1, IP2), ΓSME((IP1, IP2), 1)
    Unique( Γas(IP1, IP2))
    RemoveSubSequence( Γas(IP1, IP2))
    Γs1 = Γs1 ∪ Γas(IP1, IP2)
  end for
  BIDE( Γs1,Γs1 , 1)                                        //步骤 2
  Unique( Γs1 )
  RemoveSubSequence( Γs1 )
  for each ASk ∈ Γs1 do                                     //步骤 3
    if ¬ ∃ASj ∈ Γs,▷SubSeq(ASk,ASj)
      for each ASj ∈ Γs,▷SubSeq(ASj,ASk) do
        Γs = Γs - {ASj}
      endfor
      Γs = Γs ∪ {ASk}
    endif
  end for
```

首先获得基于 IP 对的 SME 集合,对每个集合分别进行闭频繁序列的挖掘,默认的最小支持度为 1,即挖掘结果将包含所有出现过的原始报警。我们在实现中采用双向扩展频繁闭序列挖掘算法(a frequent closed sequence mining algirothm with Bi-Directional Extension,BIDE)[6]算法进行序列挖掘,它相比传统算法速度要快一个数量级以上。由于得到的闭频繁序列中可能会包含 Id 相同的超报警,因此需要对得到的闭频繁序列进行唯一化操作。唯一化指对于每个 $\Gamma_{as}(IP_i,IP_j)$ 的 $\Gamma_{id} = \{Id_1 \rightarrow Id_2 \rightarrow \cdots \rightarrow Id_n\}$,$\forall 1 \leqslant i < j \leqslant n$,如果 $Id_i = Id_j$,那么 $\Gamma_{id} = \Gamma_{id} - \{Id_j\}$,唯一化后的场景候选满足定义 4 的入侵场景条件。接下来为了减少场景候选的数量,我们从 $\Gamma_{as}(IP_1,IP_2)$ 中删除子序列场景候选,即 $\forall AS_i \in \Gamma_{as}(IP_1, IP_2)$,$\exists AS_j \in \Gamma_{as}(IP_1,IP_2), i \neq j$,$\triangleright SubSeq(AS_i,AS_j)$ 成立,则 $\Gamma_{as}(IP_1,IP_2) = \Gamma_{as}(IP_1,IP_2) - \{AS_i\}$,最后将所有 $\Gamma_{as}(IP_i,IP_j)$ 放入临时场景集合 Γ_{s1} 中。假设有每个 IP 产生 N 个场景候选且每个候选平均有 M 条报警,则唯一化操作平均时间为 $O(N\log M)$,去除子序列操作平均时间为 $O(N\log N\log M)$。

第二步对 Γ_{s1} 进行闭频繁序列挖掘,把每个场景候选的 Γ_{id} 作为一条序列,然后再次进行唯一化和去除子序列操作。算法分为两步挖掘而不是对 Γ_{SME} 序列进行一次挖掘主要有两个目的:一是优化,二步挖掘可以有效节省挖掘算法所需内存空间;二是场景关联,分布式攻击过程往往包含多个 IP,而初次挖掘的每个 IP 对的场景候选是这类攻击过程的一部分,通过关联这些中间过程即可还原出分布式攻击。

第三步更新 Γ_s,包容最新入侵场景。对于每个新的场景 AS_k,如果它是某个原有场景 AS_j 的子序列,则不用考虑;如果存在原有场景 AS_j,是新场景 AS_k 的子序列,则从原有场景中删除 AS_j;如果新场景与任何原有场景均无关系,则直接把 AS_k 加入 Γ_s。该步骤实际上是频繁序列模式更新问题,这里只对该步骤的处理逻辑给出形式描述。

Γ_s 中的任何场景可认为是某个同源同目标的复合攻击过程,该过程按优先级递增顺序排列,每个场景反映了频繁发生的攻击过程,如果攻击者将来再使用该攻击过程,通过实例化该场景可以轻松还原攻击者的意图。

场景构建算法最消耗时间的部分在挖掘闭频繁序列算法 BIDE,该算法难以具体推算其时间复杂度,但是实验表明该算法在实际应用中具备较好性能,可以满足在线运行的要求。

2.3　实例化算法

当场景集合构建完毕后,系统对随后的每条原始报警进行场景匹配即场景实例化过程。对每条原始报警 E_k,算法首先得到 E_k 所属 IP 对的实例集合,然后判断每个实例的 $Inst_{as}.SME_i.\Gamma$id 集合在新加入 $E_k.Id$ 后,是否还是 Γ_s 中某个场景的子序列,如果是则可认为 E_k 属于实例 $Inst_{as}$。还有一种情况是 SME_i 中已经存在有 $E_k.Id$,那么 E_k 将被聚合进实例 $Inst_{as}$ 的超报警序列。从图 4 可以看出,不同场景可能存在相同的子序列,为避免 E_k 可能被加入到多个实例中,选择与当前报警 E_k 时间最相近的场景实例,即 $E_k.Time - Inst_{as}.Time$ 最小,保证 E_k 仅属于唯一一个实例,这种方式更容易发现使用自动工具攻击的入侵场景。当实例 $Inst_{as}$ 被 E_k 更新后,实例的 Γ_{AS} 同样被更新以满足定义 5 的要求。

如果 E_k 无法加入到当前实例集合中,那么需要新分配一个实例 $Inst_{as}$。首先找到 E_k 可能匹配的所有场景,即所有包含 $E_k.Id$ 的入侵场景 AS 都将放入 $Inst_{as}.\Gamma_{AS}$ 中,然后构造仅包含 E_k 的 SME 序列,将 $Inst_{as}.Time$ 设置为 $E_k.Time$ 即可。

实例化过程实际上是每个实例 $Inst_{as}.$ 的 SME 和 Γ_{AS} 属性不断变化的过程,随着新报警的加入,SME 序列不断增长,而 Γ_{AS} 将不断缩小,最好情况下 Γ_{AS} 将最终只包含一个场景即唯一匹配。但很多时候只能实例化某个场景的子序列,由于不同场景会存在共同子序列,因此实例可以是多个场景的实例。这是一种保守策略,即在无法确定某个具体场景时,确定所有可能的场景集合。

判断原始报警是否属于某个实例可以在 $O(1)$ 时间内实现,如果将所有场景集合用确定有限自动机 DFA 表示,原始报警作为输入,自动机的每个状态包含一个场景集合(每个实例可能实例化的场景集),就可以实时维护每个实例所处的状态。设场景数为 N,每个场景的平均序列长度是 M,自动机需要维护最多 $N*M$ 个状态。因此 ExistInstance 的时间性能与 E_k 匹配的实例数成线性关系,CreateInstance 的时间性能与 E_k 匹配的场景数成线性关系。

场景实例化算法如下:

```
ExistInstance( Γinst, Ek )
  Γinst(Ek) = ∅
  for each Instas ∈ Γinst( Ek.srcIP, Ek.dstIP)
    if ∃MEi ∈ Instas.SMEi && Ek.Id = MEi.Id
      Γinst(Ek) = Γinst(Ek) ∪ {Instas}
    else
```

$\Gamma_{id} = \{Inst_{as}.SME_i.ME_m.Id: 1 \leqslant m \leqslant n\} \cup \{E_k.Id\}$
$AS_k = (SME_i.TimeGap, \Gamma_{id}, \varnothing, \varnothing)$
if $\exists AS \in Inst_{as}.\Gamma_{AS}$ && $\triangleright SubSeq(AS_k, AS)$
$\Gamma_{inst}(E_k) = \Gamma_{inst}(E_k) \cup \{Inst_{as}\}$
end if
end for
$Inst_k = Inst_{as} \in \Gamma_{inst}(E_k)$ && $\max(E_k.Time - Inst_{as}.Time)$
if $\exists ME_i \in Inst_k.SME_i$ && $E_k.Id = ME_i.Id$
$Inst_k.SME_i.ME_i.\Gamma_E = Inst_k.SME_i.ME_i.\Gamma_E \cup E_k$
else
$Inst_k.SME_i.\Gamma_{ME} = Inst_k.SME_i.\Gamma_{ME} \cup$ AllocateME(E_k)
$Inst_k.\Gamma_{AS} = \{AS: \forall MEi \in SME.\Gamma_{ME}, ME_i.Id \in AS.\Gamma_{id}\}$
CreateInstance($\Gamma_s, \Gamma_{inst}, E_k$)
$\Gamma_{s1} = \{AS_j: AS_j \in \Gamma s \wedge E_k.Id \in AS_j.\Gamma_{id}\}$
if $\Gamma_{s1} \neq \varnothing$
SME_i = AllocateSME(E_k)
$\Gamma_{inst} = \Gamma_{inst} \cup \{(SME_i, \Gamma_{s1}, E_k.Time)\}$
Endif

3 实验评估

我们已经实现了系统的一个初步原型,并使用 Darpa99 数据集的第三周网内数据和 Darpa2000 数据集进行场景挖掘实验。实验环境是 Windows XP Professional SP2, Intel CPU 2.66 GHz,512 MB 内存,原始报警由 Snort2.8.2 的 Win32 版本产生,系统原型使用 VC ++6.0 开发。

我们使用 DARPA99 数据来验证场景构建算法的性能。由于 Darpa99 每天的报警都大致为 3 500 ~ 4 000 条,因此我们设定 *Thr.TimeThr* 为一天,每条超报警序列和场景的 TimeGap 属性同样设置为一整天。对于 3000 多条报警,场景构建算法所耗时间为 80 ~ 90 ms,如果对整周数据分析,时间在 400 ms 左右,每天场景为 30 ~ 40 个。

我们应用 Darpa2000 中内网在阶段 1 到阶段 5 收集的数据来示例说明识别攻击场景。应用场景构建算法分析得出共有场景 16 个点表示每个场景的 Γ_{id} 集合。图 2 中序列长度最长的场景 2 勾画了一幅完整的攻击场景,攻击者首先寻找是否存在 rpc sadmind 漏洞(1:1957),接下来对 RPC 请求解码(1:12626, 1:585, 1:12628, 1:2256),然后发起 exploit 攻击(1:1911)获得管理员权限,最后调用 rsh 得到远程 shell(1:610),此后攻击者可以在此基础上攻击其他机器。很容易看出,图中最复杂的 4 个场景均有共同子序列,因为在构建超报警序列时,由于 1:585, 1:12626 和 1:12682, 1:2256 这些报警的优先级相同而且它们在报警序列中以不同时间顺序出现过,在初始化时会被构建成不同的报警序列,序列挖掘后则形成不同场景,在实际应用过程中有可能造成场景数量过多,因此需要进一步研究场景的表示方法,将拥有共同子序列的多个场景合并为一个场景。

由于我们的实验仅使用 Snort 对 Darpa2000 的网络数据进行分析,对 Snort 产生的原始报警做进一步聚合与关联处理,因此实验结果只验证了Darpa2000数据的阶段1到阶段4的攻击场景关联,而最后一步的 DDoS 攻击(阶段 5),由于 Snort 没有任何报警,因此实验没有检测到该 DDoS 攻击。

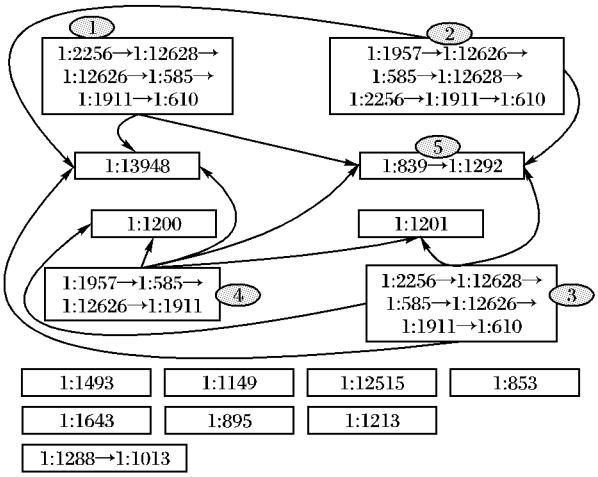

图 2 Darpa2000 数据的场景集合

4 结语

本文提出了一种基于闭频繁序列挖掘算法来进行入侵场景自动构建的方法,并给出了所有相关算法的逻辑描述。该方法可根据设定的阈值从已有原始报警集合提炼出所有可能的攻击场景集合,然后用这些场景去聚合与关联随后的原始报警。在 Darpa99 和 Darpa2000 数据集上的实验表明,该方法适合在线运行,并且不需要额外先验知识,能有效发现复合攻击场景。未来工作主要包括:1)研究频繁模式更新算法,解决场景集合的快速自适应更新问题;2)当前生成的场景还仅仅是报警序列,导致场景集合中存在很多冗余信息,需要进一步研究如何将这些具有相同子序列的场景合并为一个场景。

参考文献:

[1] 穆成坡,黄厚宽,田盛丰. 入侵检测系统报警信息聚合于关联技术研究综述[J]. 计算机研究与发展,2006,43(1):1 - 8.

[2] VALDES A, SKINER K. Probabilistic alert correlation[C]// The 4th International Symposium on Recent Advances in Intrusion Detection: RAID 2001, LNCS 2212. Berlin: Springer, 2001: 54 - 68.

[3] CUPPENS F, ORATOL R. LAMBDA: A language to model a database for detection of attacks[C]// Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection. London: Springer-Verlag, 2000: 197 - 216.

[4] NING P, CUI Y, REEVES D S. Analysing intensive intrusion alerts via correlation[C]//Recent Advances in Intrusion Detection 2002, LNCS 2516. Berlin: Springer-Verlag, 2002: 74 - 94.

[5] QIN X, LEE W. Statistical causality of INFOSEC alert data[C]// Recent Advances in Intrusion Detection 2003, LNCS 2820, Berlin: Springer-Verlag, 2003: 73 - 94.

[6] WANG J Y, HAN J W. BIDE: Efficient mining of frequent closed sequences[C]// Proceedings of 20th International Conference on Data Engineering. Washington, DC: IEEE Computer Society, 2004: 79 - 90.

(上接第 2222 页)

[6] BONEH D, BOYEN X. Short signatures without random oracles and the SDH assumption in bilinear groups[J]. Journal of Cryptology, 2008, 21(2): 149 - 177.

[7] BOYEN X, WATERS B. Full - domain subgroup hiding and constant - size group signatures[C]// Proceedings of the 10th International Conference on Practice and Theory in Public-Key Cryptography, LNCS 4450. Berlin: Springer, 2007: 1 - 15.

[8] DELERABLEE C, POINTCHEVAL D. Dynamic fully anonymous short group signatures[C]// Vietcrypt' 06, LNCS 4341. Berlin: Springer-Verlag, 2006: 193 - 210.

[9] GROTH J. Fully Anonymous group signatures without random oracles [C]// Advances in Cryptology - ASIACRYPT 2007, LNCS 4833. Berlin: Springer, 2008: 164 - 180.