文章编号:1001 - 9081(2005)07 - 1554 - 04

基于本体的协同式入侵检测系统

陈 刚[1],陈 伟[2]
(1. 中国联通有限公司 广州分公司,广东 广州 510655; 2. 武汉大学 计算机学院,湖北 武汉 430072)
(chenganggz@unicomgd.com)

摘 要:经过对现有的入侵检测系统的分析,认为多点协同检测能够使入侵检测系统更加准确、有效地检测入侵。提出一种基于本体的模式匹配方法,同时对协同式入侵检测的体系结构与协调方法进行了讨论,它可以使检测工作更加灵活,另外也提供了全局的信息定位以支持协同检测。

关键词:入侵检测;本体;协同检测
中图分类号: TP393.08 **文献标识码**:A

Ontology based cooperative intrusion detection system

CHEN Gang[1], CHEN Wei[2]
(1. *Guangzhou Branch, China Unicom, Guangzhou Guangdong* 510655, *China*;
2. *Computer School, Wuhan University, Wuhai Hubei* 430072, *China*)

Abstract: After a survey of present IDSs, it was concluded that more accurate and efficient detection result could be obtained by using multi-sensor cooperative detection. A matchmaking method based on ontology was given to improve flexibility of detection. Cooperative detection framework based on the ontology was also discussed.

Key words: intrusion detection; ontology; cooperative detection

0 引言

自从入侵检测的概念在 20 世纪 80 年代中期引入以来[1],入侵检测系统(Intrusion Detection System, IDS)已经经历了近三十年的发展。但是,高的误检率阻碍了入侵检测系统的实际应用。在对入侵检测系统高误检率的原因进行分析之后,我们认为,入侵检测系统效率不高的原因可以部分归结为不充分的检测数据源和缺少协同工作。许多 IDS 只依赖一个数据源:网络数据或基于主机的数据。但是很多入侵行为在这两类数据源中都有特征显示。如果更多探测器捕获的数据能够用来分析入侵行为,那么入侵检测的准确性将有所提高。

关键问题是如何关联多探测器获取的信息来评估被监控系统。在本文中,我们讨论了基于本体的协同检测体系结构。本体为入侵检测系统提供了对于多探测器收集的异种统计数据的共享概念以及提供它们之间关联。基于本体的概念,我们提出了协同检测的系统结构来关联多探测器收集的信息数据,并且给出了检测入侵行为的一种灵活、高效的匹配算法。

1 相关工作

一些入侵检测系统同时利用基于主机的数据和基于网络的数据来进行分析检测。DIDS[2](Distributed Intrusion Detection System)是一个分布式入侵检测系统,是较早采用分布式协同检测框架的系统,在它的框架里面同时使用了 Haystack 和 NSM 两种检测系统。它由三个部分组成:一个是在每个主机上的主机监视器,进行本地异常事件检测,同时记录本地异常事件检测结果并将统计数据传送至 DIDS 控制器;第二部分是在局域网安置的网络监听器,监听网络流量;第三部分是一个 DIDS 中心控制器,负责分析从主机监视器与网络监视器送来的数据,并将最终检测结果显示到安全管理员的控制台。中心控制器由通信管理器与专家系统组成。

EMERALD (Event monitoring enabling responses to anomalous live disturbances)[3]是一个分布式的基于主机与网络的入侵检测框架,具有良好的可扩展性。EMERALD 具有多层结构,它将多个高度分布、独立的监视器分布在企业网络中,这些监视器对事件流同时进行误用检测和基于统计的异常检测,为本地提供实时安全保护。EMERALD 还可以将分布的监视器的信息收集起来进行全局检测,以检测蔓延在整个网络中的入侵。

Frincke 在文献[4]中针对支持多域的协同方式入侵检测的系统结构提出了一些原理,还提出了协同式数据共享系统的原型,通过举例方式阐明了这些原理。

本体是对一些概念和关系的明确描述,它被应用于很多研究领域,如互联网语义、知识管理、人工智能等。很少有文献说明将本体应用于入侵检测领域。在文献[5]中,Pinkston 给出了一个以目标为中心的本体,用于描述入侵检测域内部概念以及它们之间关系。他们通过 DAML + OIL 语言实现了这种本体模型,但是似乎没有充分利用这些互相关联的和继承的关系进行检测工作。

2 基于本体的协同式入侵检测系统

2.1 系统结构

基于本体的协同式入侵检测系统采用混合式的体系结构,图 1 显示了体系结构中的三层结构:多探测器层,协同检

收稿日期:2005 - 01 - 04;**修订日期**:2005 - 03 - 24

作者简介:陈刚(1979 -),男,湖北黄冈人,硕士,主要研究方向:网络通信、网络安全; 陈伟(1979 -),男,江苏淮阴人,博士研究生,主要研究方向:软件工程、数据挖掘.

测层和控制管理层。多探测器层和协同检测层使用了分布式体系结构,每一个探测器和检测器都能独立工作。控制层是集中式的体系结构,用于集中控制多探测器层和协同检测层,由于控制与管理命令相对来说数据量很小,采用集中方式也不会给系统带来很大负担。

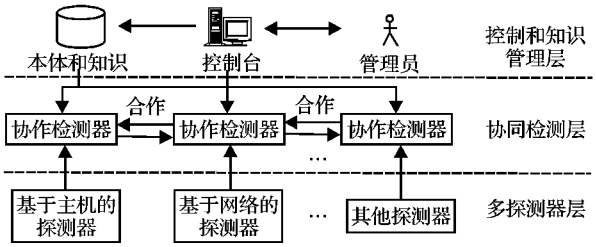

图 1　系统结构

多探测器层从网络、主机及网络其他设备,如防火墙、路由器等,收集原始数据。在我们的方法中使用分布式的多探测器,是因为不同设备上的数据能够为准确的检测提供更多的信息。例如,一个 DDoS 攻击者在网络上安装了攻击程序,他需要和这些攻击程序通信以发送攻击命令,如果他对这些命令加密,我们很难仅仅通过网络监听来迅速捕获这些攻击命令,但是这些攻击命令到达主机后会进行解密,然后调用一些异常的系统命令,这些信息容易在主机上捕获。所以如果同时使用主机和网络信息,就可以更准确地发现攻击特征。

协同检测层中,每个检测器可以在需要时向别的检测器发送查询命令。每个检测器可以根据本体的知识,找到自己感兴趣的信息。

控制管理层主要管理本体知识和控制多探测器层与协同检测层。当收到警报后管理员可以采取响应。本体知识存于本层中并及时更新,本体知识也存于协同检测器的本地数据库中,由控制管理的本体知识负责维持本体知识的一致性。

2.2　本体

术语本体的意思是某个概念的详细描述。一个本体就是对某一领域内的概念和概念之间关系的一种描述(类似一段程序的详细设计书)。在入侵检测系统中引入本体能够架构一个强有力的信息交流平台,方便协同检测的探测器之间交流机器可理解的信息。理解其他合作探测器发过来的信息以及对系统当前状态的描述对于协同工作是至关重要的。通过 IDS 规则以及由 Common Vulnerabilities and Exposures (CVE) 公布的安全漏洞进行分析研究之后设计了一个本体。与文献[5]给出的本体相比,我们设计的本体关注能侦测到的入侵特征,而不是像文献[5]中描述的,着重去分析、估计入侵者的动机。

一个完整的本体包括两种节点:值节点和属性节点。属性节点描述了所有能够被多探测器发现的特征,而值节点是一些属性节点的子节点,它们给出了其父亲属性节点可能存在的值。图 2 展示了我们设计的本体的一部分(仅仅包含属性节点),为了说明清楚,这里没有完全展示本体的结构。在本体的根的位置上是攻击特征。根节点的下一层节点是一些代表不同类入侵特征的属性节点,这些特征是来自不同类的探测器,如主机探测器、网络探测器、路由器日志等。本体中的高层节点代表更抽象的属性,并且是低层节点的父节点。当我们想要得到某个信息,通过本体的结构描述可以方便地查看到此信息属于哪个探测器。例如我们想知道总的内存使用量的信息,我们可以从本体中了解到这个信息能够从属于主机特征的系统状态探测器获得。这对于探测器协同工作很有用,因为它能够帮助定位信息源。

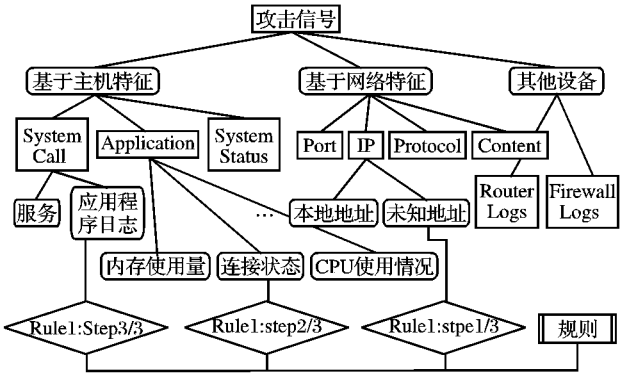

图 2　本体的结构

2.3　基于本体的模式匹配

入侵信号检测系统常常使用字符串匹配或专家系统来进行模式匹配。字符串匹配系统,如 snort,是对字符串进行简单的子串匹配。但是这样的工作机制不是很灵活,如果攻击的特征稍有改变,系统就无法检测这次攻击。例如:一个隐蔽的服务器利用端口“666”来同客户端通讯,如果服务器更改服务端口至“555”,如果不把一条新的规则加入到规则库,字符串匹配系统就不能检测到这个行为。专家系统功能强大,但是由于实现机制复杂,其处理效率比较低。本文提出的基于本体的模式匹配方法介于这两类方法之间,具有较好的灵活性并且有较高的效率。

上面提到的本体(图 2)的每个节点是体现收集到的信息的属性的特征节点。在完整的本体描述中,将包括值节点,它们展示这些特征的取值分布情况。值节点的父节点是拥有这些值的特征属性,如果属性的取值是连续的,那么须经过离散化的预处理。在上面的例子中,攻击者将与后门服务器通讯端口从可疑的“666”调整到很少被使用的“555”上,而且使用很少使用的 IP 地址建立连接。如图 3 所示,一条规则“Rule1:Step1/2”和被观察到的可疑数据节点“intrusion”被关联到图 3 显示的几个取值节点上。尽管检测到的数据“intrusion”并不能完全匹配给出的规则“Rule1:Step1/2”,但是我们知道这仍然是入侵行为。

为了评价规则和数据之间的关系以及相似程度,每个取值节点和其父节点之间的边被赋予一个权值,取值范围为[0,1]。权值代表同一个父节点下面不同取值节点之间的关系,1 表示它们之间有最大的关联性,而 0 表示最小的关联性。例如节点“Port”有三个值:“Suspicious”,“Seldom used”,“Frequent used”,而且我们采用以下的方法存储节点 N 的权值:$V_N(w_1, w_2, \cdots, w_n)$,$w_i$ 表示节点 N 同其兄弟节点 i 之间的权值。

表 1　端口的权值表

Weight	Suspicious	Seldom Used	Frequent Used
Suspicious	1	0.8	0.1
Seldom Used	0.8	1	0.5
Frequent Used	0.1	0.5	1

如表 1 所示,在上面的例子里,三个向量为:$V_{suspicious}(1)$, $V_{Seldom}(0.8,1)$, $V_{Frequent}(0.1, 0.5,1)$。

$V_{Seldom}(0.8,1)$中的0.8说明“Suspicious”和“Seldom used”之间的权值为0.8,0.8 这个值是通过专家的经验得出来的。“Suspicious”和“Seldom Used”之间的权值为 0.8 是因为很少使用的端口往往被入侵者用来进行恶意的行为。

但是,“Suspicious”和“Frequent used”之间的权值为 0.1,因为入侵者很少使用常用的端口进行攻击。

斜对角线上的值都是 1,因为每个值与其本身的相似度是最大的。

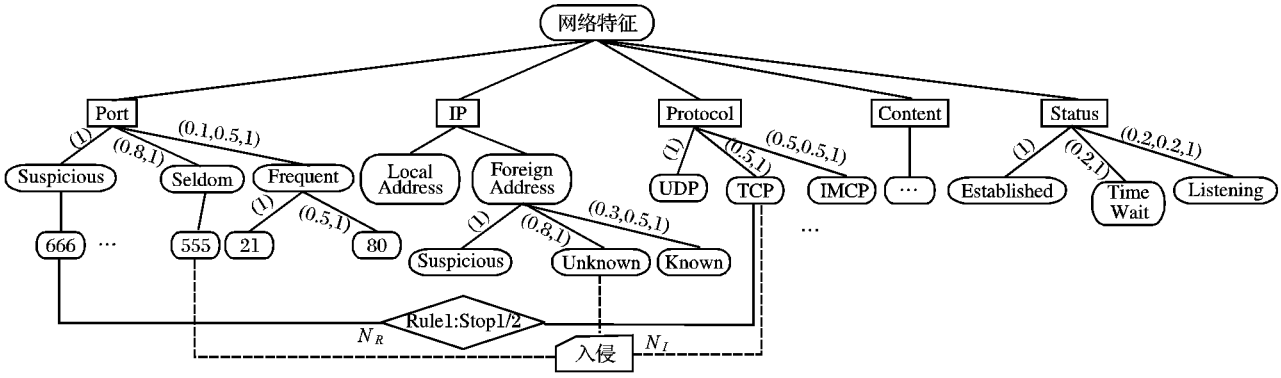

图 3 基于本体的模式匹配

将节点 $Rule(N_R)$ 和节点 $Intrusion(N_I)$ 附加到本体上之后,存在从节点 $Rule(N_R)$ 到节点 $Intrusion(N_I)$ 的一组路径,标记为 $path\_set(N_R, N_I)$,被用来评估两个节点之间的相似性。但是没有必要对这两个节点之间所有路径进行评估,只需要考虑那些在两个相邻节点之间的、具有相同父亲节点的路径。在上面的例子里,我们只考虑三条路径:“666”到“555”,“Suspicious”到“Unknown”以及“TCP”到“TCP”。如果节点 N_R 和节点 N_I 在一条路径上,是附属于同一个节点的,那么探测器会给出相似分值 1 分。如果不是,相似分值则取决于路径上它们所分别从属的节点 N_i, N_j 的权重,总相似分值通过所有路径的相似分值计算得到。

$$path\_simi\_score_{path(i,j)} = V_i(w_1, w_2, \cdots, w_n) * (0_1, 0_2, \cdots, e_j, \cdots, 0_n)' \quad (e_j = 1) \tag{1}$$

$$total\_score(N_R, N_I) = \frac{\sum_{path(i,j) \in path\_set(N_R, N_I)} path\_simi\_score_{path(i,j)}}{Count(path\_set(N_R, N_I))} \tag{2}$$

上面例子中的 $total\_score$ 为:

$$total\_score = \frac{(0.8, 1) * (1,0)' + (0.8, 1) * (1,0)' + 1}{3} = 0.8667$$

如果 $total\_score$ 超过了阈值,它表明被检测的数据与我们的规则相匹配,属于可能的入侵行为。在入侵检测过程中,大量的规则会被加入到规则库中,关联到本体上。每个统计数据相对相应规则的相似分值被函数 $total\_score$ 计算得出。为了让匹配工作更加有效,我们不需要计算每一个统计数据节点 N_{data} 和每个规则节点之间的总相似分值 $total\_score$。如果 $total\_score$ 超过阈值,我们必须有一个必要条件:路径集中至少有一个 $path\_simi\_score$ 超过阈值。基于这个必要条件,我们通过下面一个算法来过滤规则,避免大量执行 $total\_score$ 函数,以提高匹配效率。

基于本体的匹配算法如下:

```
Boolean: Intrusion_MatchMaking( Var RuleSet: Rst;
  Audit Data Node: N_data)
{
  Queue: Q;
  FOR each node N in AttachedNodeSet( N_data)
    // AttachedNodeSet gives all the nodes that N_data is
    //attached to
    FOR each rule R in Rst
      IF path_simi_score_path(N, Rj) > Threshold THEN
        insert( Q, R);
        // If the path_simi_score exceed the threshold
        //The rule R will be insert to queue
        continue;                    // Quit inner loop
      END IF END FOR
  END FOR
  FOR each R in Q
      // Calculate the total score of the N_data and rule in Q
    IF total_score( N_data, R) > Threshold THEN
      RETURN TRUE;
    END IF
  END FOR
  RETURN FALSE;
}
```

2.4 应用本体于协同检测

值节点和属性节点的关系能够用来进行更加准确,更加有效的入侵行为的模式匹配。属性节点之间的关系则提供了协同检测的基本信息,父亲节点指明了子节点的位置。例如,在协同检测过程中需要了解总的内存使用量,本体则告知检测系统从 Host 下的 System status 探测器获得。本体为协同检测提供了机器可理解的公共知识。

为了给出更加准确的告警,检测系统会在做出结论之前收集尽可能多的信息。当规则里的消息分散在不同的探测器,规则就变得十分复杂。例如,图 2 中的一条完整的规则由三个子规则组成。当检测系统完成了第一步 Network 探测器相关的检测,它还要进行 Host 探测器的 Connection status 检测。本体指示探测系统在哪里获得想要的信息,然后探测系统会发一个查询请求到 Host 探测器的 System Status 监视器。在进入到第三步之前,检测系统又会从本体获知 Application logs 的位置。这个情景在协同检测中很常见。如果入侵者通

过加密的指令与服务器通讯,那么探测器仅仅能够发现一些通过未知端口通讯的连接,而不能解密通讯的内容。因此,解密的命令只能从主机上的应用程序日志中获得。

面对多步的规则,各个探测器分别执行单步的检测,并且把检测结果临时存储到本地数据库,每一个子告警都有一个TTL(生存期)标记,当子告警过期时,将它从数据库中删除。当最后一步检测结果完毕之后,探测器开始协同工作,并从相关的本地数据库查询每个子步骤的检测结果。整个协同检测的过程描述如下:

```
CooperatingDetection( Var RuleSet: Rst; Audit Data Node: N_data)
{ Boolean: Alert = FALSE;
  FOR each step S_i in multi-step rule
    IF Intrusion_MatchMaking( S_i, N_data) = TRUE THEN
      // Perform each sub-rule detection individually
      IF S_i is not last step in multi-step rule THEN
        INSERT sub-alert( S_i) in local database;
        // If not the last sub-rule, just store sub-alert in db
      ELSE
        FOR j = i TO 1 STEP -1
          // If the last step, perform cooperation
          IF sub-alert( S_j) = FALSE THEN
            // detector find locality of sub-alert by
            //ontology
            BREAK ;
            // If one of sub-alert is false, not give alert
          END IF
        END FOR
        IF j = 0 THEN Alert = TRUE;
      END IF
    END IF
  END FOR
  IF Alert = TRUE THEN sendalert()
}
```

3　试验结果

设计了一个试验来验证我们的方法。本体的节点和边存储在关系数据库里。

表 2　本体中节点的表结构

字段	描述
NodeID	节点 ID
Parent	父节点 ID
Type	类型
Value	值
Level	节点所在层,根节点在 0 层

表 3　本体中边的表结构

字段	描述
EdgeID	边 ID
Parent	父节点 ID
Child	子节点 ID
Weight	权值

由于条件限制,我们仅仅实现了基于主机和网络的协同检测。一个基于 WinPcap 的工具被用来收集网络层的数据包,基于 NT 系统的工具 Strace 被用来在 Windows2000 Server 操作系统上跟踪 NT 系统调用。一些主机日志,如内存使用情况和网络连接状态也被用来检测入侵。根据 Snort 和 CVE 系统的规则,我们定义了 21 条规则,其中大部分是 R2L(远端到近端)和 U2R(用户到根)的检测规则,因为 R2L 和 U2R 的入侵比较难于检测,而我们通过协同检测的方法同时分析主机和网络统计数据对这种入侵是适合的。这些规则中的大部分是有 2 个子步骤的,分别包含基于主机的和基于网络的子步骤。在试验中采用入侵工具 Netbus 来评估我们的系统原型。在匹配过程中,改变了 Netbus 的通讯端口来检验基于本体的入侵匹配算法。Netbus 的默认通讯端口为 20034,我们分别将它改变为 20038 和 80,同时也运行了 Snort 2.1.1 来检测模拟的入侵并比较其与原型的检测率情况。

表 4　Snort 和原型的检测结果比较

Netbus	检测结果		误测次数	
	Snort	原型	Snort	原型
Default Port	Y	Y	8	2
Port: 20038	N	Y	12	3
Port: 80	N	Y	18	2

表 4 比较了 Snort 和原型的检测结果和误检次数。试验结果表明了我们提出的原型优越于 Snort。从表 4 的第 2 行,可以看出原型的灵活性大于 Snort,因为原型中采用了基于本体的规则匹配算法。但是,如果入侵行为变化得太大,原型系统就显示了它的局限性。由于结合了 Host 和 Network 协同检测,误检率大大降低。

4　结语

本文提出了本体的概念来描述多探测器发现的入侵行为特征之间的关系。本体包含两种节点:取值节点和属性节点。通过给属性节以及其父亲属性节点之间的边指定权值,提出了一种更灵活的入侵规则匹配算法。同时,属性节点和其父亲节点之间的关系也从另一方面指明了所需信息的位置。本文还提出了一个基于本体的协同检测方案。

参考文献:

[1] DENNING DE. An Intrusion Detection Model[J]. IEEE Transactions on Software Engineering, 1987, SE-13(2): 222 - 232.

[2] SNAPP SR, SMAHA SE, TEAL DL, *et al*. The DIDS (distributed intrusion detection system) prototype[A]. Proceedings of the Summer USENIX Conference[C]. San Antonio, Texas, 1992. 227 - 233.

[3] PORRAS PA, NEUMANN PG. EMERALD: Event monitoring enabling responses to anomalous live disturbances[A]. Proceedings of the 20th National Information Systems Security Conference[C]. Baltimore, Maryland, USA, 1997. 353 - 365.

[4] FRINCKE D, MORCONI J, MCCONNELL J, *et al*. A Framework for Cooperative Intrusion Detection[A]. Proceedings of the 21st National Information Systems Security Conference[C], 1998. 361 - 373.

[5] PINKSTON J, UNDERCOFFER J, JOSHI A, *et al*. A Target-Centric Ontology for Intrusion Detection[A]. The 18th International Joint Conference on Artificial Intelligence[C], 2003.