Web入侵检测系统高效多模式匹配算法*

范轩苗[1]，郑 宁[1]，范 渊[2]
（1.杭州电子科技大学 计算机学院，杭州 310018；2.亚龙（安恒）信息科技（杭州）有限公司，杭州 310035）

摘 要：针对 Web入侵检测系统中存在的攻击模式误匹配与效率问题，提出了一种高效的多模式匹配算法 MPMA。MPMA通过构建比较树，并在比较树的每个节点中记录下次比较的字符位置以提高比较效率，并利用（模式，偏移）信息对来搜索可能符合的匹配模式。详细的实验以及与现有算法的比较表明，提出的 MPMA不仅适合于 Web入侵检测系统，同时在时间、空间和匹配率性能上具有更高的效率。

关键词：入侵检测系统；多模式匹配；Web
中图分类号：TP393.08 文献标志码：A 文章编号：1001-3695(2009)04-1528-04

Efficient multi-pattern matching algorithm for Web intrusion detection systems

FAN Xuan-miao[1], ZHENG Ning[1], FAN Yuan[2]
(1. School of Computer Science, Hangzhou Dianzi University, Hangzhou 310018, China; 2. Hangzhou DB Appsecurity Information Technology Co., Ltd, Hangzhou 310035, China)

Abstract: To overcome the defects of false pattern matching and time-and-space efficiency in Web intrusion detection systems (IDSs), this paper proposed an efficient multi-pattern matching algorithm called MPMA. With building comparison tree, every tree node had a position value which could tell you where an octet comparison should be made next, and MPMA used (pattern, offset) pair to find possible matching patterns. Detailed experimental results and comparison with existed algorithms prove that the proposed MPMA not only fits Web IDS, but also outperforms current state-of-the-art schemes in terms of time efficiency, space efficiency and matching ratio.

Key words: intrusion detection systems(IDS); multi-pattern matching; Web

引言

由于管理者疏忽、系统漏洞、新的攻击手法层出不穷等各种因素，使得 Web服务器遭受网络攻击的事件频繁发生[1]。其中以 Web应用类型的攻击最多，而大部分 Web应用并没有采取专门有效的防护措施来应对。导致 Web应用进一步面临威胁的另一个因素是，Web服务器与应用底层架构的变化以及使用非安全开放源代码组件现象的增加[2]。Web服务器与 Web应用已经从最初提供简单的静态内容演变到提供丰富的动态内容；除了可以创建动态页面与启动应用程序外，还可以与数据库进行通信以生成对用户有用的内容。大多数 Web服务器平台都将应用程序与服务器捆绑在一起，即使最简单的网站也会与 Web应用进行交互，这就为攻击提供了更多的机会。

由于 Web服务通常会自动打开，访问控制机制的应用在有些情况下是不切实际的。此外，Web服务器在结构方面很复杂，在一个位置安装所有保护机制也不可能达到理想效果，在那些防御机制有效的地方，需要的配置变化依赖于引起攻击的根源和特殊的 Web服务结构。有一些策略可以使入侵者绕过 IDS或使 IDS不起作用。例如，入侵者可以试图使网络溢出或在虚拟包中插入一些新的恶意包来实现上述策略[3]。针对 Web攻击，目前主要的措施是采用入侵检测系统（IDS）尽快、尽可能可靠地检测出各种入侵行为[4]。IDS同时能对数据帧的帧头与负载进行检测，并与已知的所有攻击方式进行比较，找出可能的攻击方式。也就是说，如果将数据包流当做一个很长的字符串，IDS的工作就是检测该字符串中的子字符串是否与已知的攻击模式相符，并根据预定的方案采取相应的防范措施，以阻挡使用者或以报警的方式进行处理。针对字符串扫描检测，文献[5,6]分别采用哈希与基于签名的方式，虽然这些方法能快速地找出可能的攻击模式，但是无法避免错误匹配问题。而 Snort[7]方法在字符串检测上花费高达 70%的执行时间和 80%的指令执行时间，效率并不理想。

本文针对 Web入侵检测系统中存在的攻击模式误匹配与效率问题，提出了一种高效的多模式匹配算法 MPMA。该算法基于比较树，利用（模式，偏移）信息对来搜索可能的匹配模式，比较树通过每个节点存储相应的位置信息，记录下次比较的字符位置以提高搜索效率。

1 多模式匹配算法 MPMA

1.1 问题描述

对于检测的数据包形成的字符串，定义为 T，T内每个字符表示为 $t_1, t_2, \cdots, t_{|T|}$。其中 $|T|$表示 T的长度。查找的 n个不同的模式定义为 $P_i (i \in [1, n])$，$P_y = p_1^{(y)} p_2^{(y)} \cdots p_{|P_y|}^{(y)} (y \in [1, n])$。多模式匹配算法 MPMA的目的就是要找出上一个位置 x和 P_y，满足以下条件：

$$t_{x+i} = p_i^{(y)} \quad i \in [1, |P_y|] \tag{1}$$

收稿日期：2008-07-10；修回日期：2008-08-29 基金项目：浙江省自然科学基金资助项目（Y106176）

作者简介：范轩苗（1983-），男，浙江绍兴人，硕士研究生，主要研究方向为网络安全（fanxuanmiaoweb2@163.com）；郑宁（1961-），男，浙江杭州人，研究员，博导，主要研究方向为信息系统与信息处理、信息安全等；范渊（1975-），男，浙江金华人，硕士研究生，主要研究方向为网络安全。

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

MPMA并不直接进行模式比较，而是先将模式转换为（模式，偏移）信息对，然后从比较树的树根开始进行比较。此时所有的（模式，偏移）对均为候选项，即所有的（模式，偏移）对都可能出现在 T上。接着读取当前所在节点的信息决定要对比的位置，经过比较后，根据结果跳转到特定的子节点上，重复上述比较步骤直到叶子节点，便可知 T中可能包含哪些（模式，偏移）对。最后再针对这些可能符合的（模式，偏移）对进行检测匹配。

以图 1(a)中所示的比较树为例，其中 P={FAT，FARM，TUTOR}，偏移为 0～2。比较树每个节点内的数字表示所要比较的字符在 T中的位置，树根的 3表示开始时要比较的位置是 T的第三个字符。当对第三个字符比较完后，根据比较结果决定选择哪个子节点。例如目前的第三个字符为 A，则将选择最左的子节点进行比较，而此时可能达到的叶子节点有（1，1）和（0，1），即经过第一次比较，过滤掉了（1，1）和（0，1）以外的（模式，偏移）对。接着比较第四个字符，如果为 R，则只需检测其是否与（1，1）符合即可。

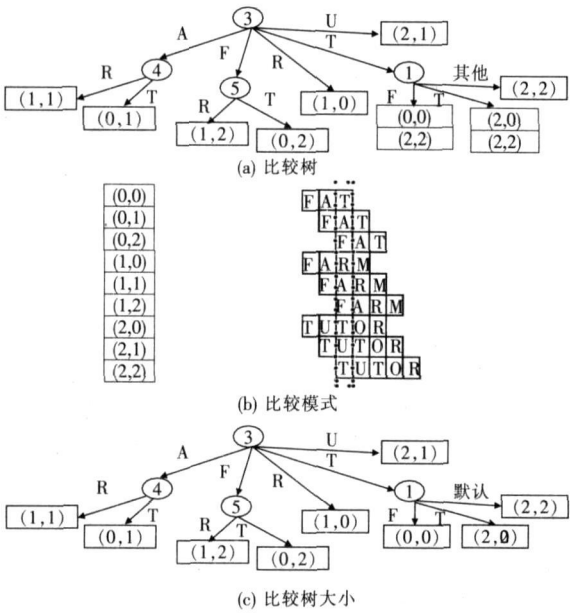

图 1　MPMA 模式比较示例

很明显，上述方法存在两个缺陷：a)模式匹配量大，当取偏移为 0～2时，由图 1(b)可知，需要处理的（模式，偏移）对是原模式个数的 3倍，因此计算量和空间开销都将增加；b)在进行（模式，偏移）对比较时，存在所有可能符合的（模式，偏移）对匹配不完全问题。由图 1(b)可知，当要比较第一个字符时，只有（0，0）、（1，0）和（2，0）三个（模式，偏移）对能比较到。其他（模式，偏移）对由于没有字符出现在该位置，将成为后续比较的候选项。

针对上述第一个问题，MPMA将采用虚拟节点压缩方式减少计算量和空间开销。针对第二个问题，MPMA采用两种不同的节点构建方法来处理：以速度优先的节点算法（Speed-MPMA）；以空间大小优先的节点算法（Size-MPMA）。

. - 算法

Speed-MPMA比较树 CT建立算法伪代码如下所示：

```
输入：模式 Pi(∈[1, n])与最大偏移量 max_offset
输出：Speed比较树 CT
for all patterns Pi(∈[1, n])
    for j=0 to max_offset
        root.candidate←((Pi, offset)←Pi)
    end for
end for
Stack=∅, Stack←root
while stack is not empty
    node←pop first element from stack; pos←Posparse(node)
    for all(pattern, offset)
        if get((pattern, offset), pos)!=NULL
            candidate_array[get((pattern, offset), pos)]
        else default_candidate←(pattern, offset)
        end if
    end for
    for all candidate
        if candidate_array[j]!=NULL
            node.child[j].candidate←candidate_array[j]+default_candidate
        else node.child[j].candidate←default_candidate
        end if
    end for
    for all node.child
        if(node.child[j] is not a leaf node)
            Stack←Stack+node.child[j]
        end if
    end for
end while
```

算法的 1～5行先将模式展开为（pattern，offset）形式，并把所有的（pattern，offset）设定为树根的候选项 candidate(pattern，offset)。其中偏移 offset的最大值为 max_offset。接着把树根压栈，并对栈进行检查，取出栈中节点，用子节点位置构建算法 Posparse产生节点位置。将该节点与所有的候选项进行比较，如果该（pattern，offset）在 Posparse产生节点的位置上，则将其放入候选数组 candidate_array[]中；否则将其存入默认的候选项 default_candidate中。然后将 candidate_array与 default_candidate的候选项与对应的子节点进行关联。最后将所有非叶子节点压栈，得到构建的比较树 CT。

子节点位置构建算法 Posparse如下：

```
输入：节点 node
输出：节点位置 pos
count[max_pattern_length+max_offset]=∅
for all(pattern, offset) po in node
    for i=po.offset to(po.offset+po.pattern.length)
        count[i]←count[i]+1
    end for
end for
pos[]←sort(count[])
for i=0 to max_pattern_length+max_offset
    if pos[i] can split(pattern, offset) from node
        return pos[q]
end for
return-1
```

子节点位置构建算法 Posparse的主要功能是根据当前的候选项 candidate(pattern，offset)，找出一个合适的位置作为建立子节点的依据。Posparse首先建立一个大小为最大偏移与最长模式的计数数组，接着检查所有节点的候选项，并将该候选项对应字符位置的计数加 1；然后将该数组按从大到小排序，从字符最多的位置开始，如果该位置可将任意两个候选项分开，则返回该位置。

Speed-MPMA_search搜索算法实现字符串的模式匹配功能。Speed-MPMA_search根据 Speed-MPMA算法建立的比较树 CT，从字符串 T的起始位置开始，对每个节点，根据 T[pos+

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

node.offset]的字符决定选择该节点的对应子节点，直到找到叶子节点为止。当到达叶子节点时，将当前的 T与该叶子节点的候选模式进行比较，直到找出所有的匹配模式为止。

Speed-MPMA_search搜索算法如下：

```
输入：数据包字符串 T 模式 P Speed-MPMA比较树 CT
输出：T所有匹配的模式
  pos← 0
  while pos< T.length
    node← CT.size.root
    while node is not a leaf node
      node← node.child[ T[ pos + node.offset] ]
    end while
    compare T+ pos with all node.candidate
    pos← pos+ CT.size.max_offset
  end while
```

1.3 Size-MPMA

Size-MPMA与 Speed-MPMA的最大区别在于候选项的产生方式。在 Speed-MPMA中，如果子节点位置构建算法 Posparse所选定的位置上有任何候选项没有字符存在时，就会指定到默认子节点上。这将导致所有的子节点被复制，这种方法将产生巨大的空间消耗。Size-MPMA则针对该缺陷，将原来分布在每个子节点上的候选项集中到一个特殊的默认子节点上，并把一个节点的候选项 candidate(pattern, offset)根据子节点位置构建算法产生的位置分为两部分：一部分是该位置有字符的 candidate(pattern, offset)；另一部分是该位置无字符的 candidate(pattern, offset)。有字符的候选项作为一般子节点处理，无字符的部分作为默认子节点处理。这种方式可以大量降低比较树占用的空间消耗。

Size-MPMA比较树建立算法伪代码与 Speed-MPMA比较树建立算法基本相同，只需要将 Speed-MPMA比较树建立算法伪代码的第 18行删除即可。也就是说，当候选项数组为空时，该子节点不会被设定为任何值。默认候选项 default_candidate会被节点设置到默认子节点 default_child上。

Size-MPMA_search搜索算法伪代码如下：

```
输入：数据包字符串 T模式 P Size-MPMA比较树 CT
输出：T所有匹配的模式
 pos← 0  stack← ∅
 while pos< T.length
   stack← CT.size.root
   while stack is not empty
    node← pop first element form stack
    while node is not a leaf
     node← node.child[ T[ pos+ node.offset] ]
     stack← stack+ node.default_child
    end while
    compare T+ pos with all node.candidate
    end while
    pos← pos+ CT.size.max_offset
  end while
```

2 混合算法 Hybrid-MPMA

Speed-MPMA与 Size-MPMA具有很大的相似性，在比较树 CT的构建上，如前所述，其差异仅仅在子节点候选项的设定上。当子节点位置构建算法 Posparse所选定的位置上有任何候选项无字符时，两种算法的处理方式不同：Speed-MPMA会把这些候选项复制到所有的子节点内，而 Size-MPMA则把这些候选项集中放入默认子节点中。在搜索算法中，Speed-MPMA经过一个节点时，只需要其位置 pos与 T相比，即可知道下一个要处理的节点；而 Size-MPMA除了要处理系统的子节点外，还需要处理默认子节点。因此混合算法 Hybrid-MPMA充分利用两个算法的优点，对常用的节点使用 Speed-MPMA，而对少用的节点使用 Size-MPMA。由于处理比较树 CT的顺序是从根节点开始，假设 T内的字符服从统一分布，则每个子节点被处理的概率为父节点的 1/256。当节点越深，其被处理的概率越小。因此，在 Hybrid-MPMA中，只需要设定一个合适的深度 L作为阈值，当深度小于等于 L时，采用 Speed-MPMA构建节点；否则采用 Size-MPMA构建节点。

根据比较树 CT的特点，可以采用虚拟节点压缩方式在保障时间效率基本不受影响的前提下，提高空间效率。对于图 1(a)和(c)中所示的比较树，树根左半边的两个子节点的候选项 candidate(pattern, offset)非常相似，其中 pattern部分完全相同，只有 offset有定差。此时只需要建立其中的一个节点，另一个节点则建立虚拟节点，如图 2所示。

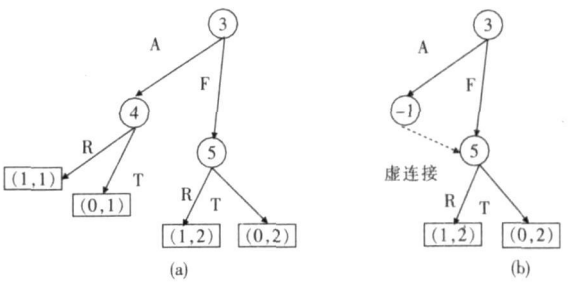

图 2 压缩比较树

图 2(a)为图 1(a)中所示的比较树的左边部分；(b)为经过虚拟节点压缩后的子树。虚拟节点记录了两个信息，一是与其相似的节点的位置，二是两者之间的偏移差。这种方式在很大程度上节省了空间消耗，而产生的时间额外开销非常小。

实验分析

为了验证提出的 Hybrid-MPMA的性能，实验中对 Path AC算法[8]、FNP算法[9]的性能进行了分析。系统采用 Linux，2.4 GHz CPU，内存 512 MB，测试数据为 858 MB，分为 2 000个数据流。模式采用 Snort的默认规则集。为了考察最短模式长度(LSP)的影响，将 LSP分成 1～4共四组。

实验 1 对 Hybrid-MPMA是否采用虚拟节点压缩及采用不同偏移情况下的内存消耗性能，当节点深度 L设置为 2～10时，算法的内存消耗性能如图 3所示。

从图 3中可以看出，当采用虚拟节点压缩后，在偏移相同的情况下，Hybrid-MPMA的内存消耗将大大减少。如偏移为 2时，采用虚拟节点压缩的内存消耗比不采用压缩时平均要少 540 KB。当偏移量增加时，Hybrid-MPMA的内存消耗将有所增加。随着节点深度的增加，对于虚拟节点压缩情况下，偏移为 2、3时内存消耗的性能变化不大。但是有一点值得注意，当不采用虚拟节点压缩时，偏移为 3的内存消耗比偏移为 4的小，当节点深度为 5～10时，两种情况下的内存消耗基本相同；但是采用压缩后，如图 3中所示，当节点深度为 10，偏移为 4时的内存消耗比偏移为 3时的内存消耗小 680 KB。这是因为虚拟节点压缩是根据模式偏移对终端的偏移差建立虚拟节点，在最大偏移越大的情况下，可以找到的虚拟节点越多，所以当

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

这两个树经过虚拟节点压缩后，偏移为 4的树可以找到比偏移为 3更多的虚拟节点。经过压缩后，节点深度为 10时，偏移为 4的内存消耗反而比偏移为 3的低 680 KB。

图 4所示为节点深度 L变化时，Hybrid-MPMA的吞吐量性能。从图 4中可以看出，随着节点深度的增加，Hybrid-MPMA的吞吐量性能逐渐增大。偏移为 2时，算法的吞吐量从 18.3增加到 22.1 MBps；偏移为 3时，吞吐量从 22.4增加到 25 MBps；偏移为 4时，从 25.8增加到 26.3 MBps。值得注意的是：当节点深度 L小于 4时，三种偏移情况下的吞吐量增加迅速，而当节点深度大于 4后，吞吐量的增加并不明显。这是因为当节点深度增加时，搜索过程中节点被遍历的概率越小，系统的吞吐量增加并不明显。

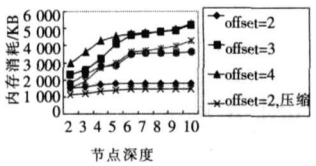

图 3　Hybrid-MPMA 内存消耗

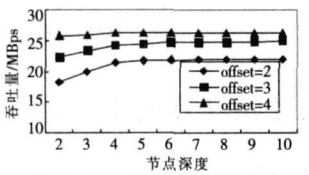

图 4　Hybrid-MPMA 吞吐量

实验 2　考虑三种算法在内存消耗与吞吐量上的性能。

图 5所示为三种不同算法在最短模式长度变化时的内存消耗性能。从图 5中可以看出，FNP消耗的内存基本上保持在 32 MB，是三种算法中内存消耗最大的；而 Path AC算法消耗的内存保持在 29 MB；提出的 Hybrid-MPMA的内存消耗为 13 MB。这也证明了 Hybrid-MPMA适合于不同最短模式长度情况下的 Web入侵检测系统，并能以较低的内存消耗实现多模式的并发匹配。

图 6所示为三种算法在最短模式长度变化时的吞吐量性能。从图 6中可以看出，FNP算法的吞吐量基本上保持在 4 MB，是三种算法中最低的；Path AC算法消耗的吞吐量保持在 11 MB；提出的 Hybrid-MPMA的吞吐量随着最短模式长度的增加而呈增长趋势，当最短模式长度增加到 4时，其吞吐量从 18.3增加到 29.8 MBps。这是因为最短模式长度越大，模式可以用来比较的字符越多，比较树的节点就可以选择较好的位置计算出子节点的位置，进而过滤未匹配模式的效率增加。这证明了提出的 Hybrid-MPMA适合于不同最短模式长度情况下的 Web入侵检测系统，并能以高吞吐量实现多模式的并发匹配，提高模式匹配的搜索效率。

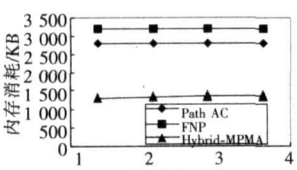

图 5　三种算法内存消耗比较

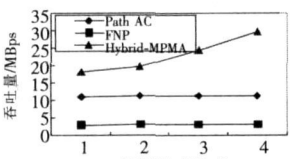

图 6　三种算法吞吐量比较

实验 3　对三种算法在不同 Web攻击下的模式匹配正确率进行性能比较。模拟器随机产生不同数量的攻击动作，并将时间设置为 30 s；三种算法在黑客攻击行为占正常行为不同百分比情况下的正确匹配率如图 7所示。从图 7中可以看出，当攻击动作所占比例增加时，FNP与 Path AC算法的性能有所下降，当攻击动作所占比例达到 15%时，这两种算法的正确匹配率分别为 93.1%和 93%。当攻击动作所占比例变化时，提出的算法的正确匹配率始终保持在 98%以上，是三种算法中正确匹配率最高的。这也充分证明了提出的算法对 Web攻击具有良好的正确匹配性能，同时也在很大程度上克服了误匹配问题。

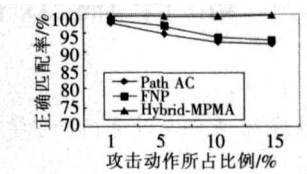

图 7　三种算法正确匹配率比较

结束语

针对 Web入侵检测系统中存在的攻击模式误匹配与效率问题，提出了一种有效的多模式匹配算法。该算法基于比较树，利用（模式，偏移）对来搜索可能的匹配模式，并通过比较树上的每个节点存储的位置信息，记录下次比较的字符位置以提高搜索效率。为了克服模式匹配量大与模式匹配不完全的问题，MPMA采用虚拟节点压缩方式减少计算量与空间开销，以速度优先和以空间大小优先的混合节点构建算法实现快速、低内存开销的模式匹配。实验性能分析表明提出的多模式匹配算法不仅具有较低的内存开销与较高的吞吐量性能，同时具有良好的误匹配性能。该算法易于在系统中实现，是一种切实可行的 Web入侵检测模式匹配算法。

参考文献：

[1] GARCIA A, JUAN J, PIKATZA A, et al. Intrusion detection in Web applications using text mining[J]. Engineering Applications of Artificial Intelligence, 2007, 20(4): 555-566.

[2] DJEMAIEL Y, REKHIS S, BOUDRIGA N. Intrusion detection and tolerance: a global scheme[J]. International Journal of Communication Systems, 2008, 21(2): 211-230.

[3] BRUSCHI D, PIGHIZZINI G. String distances and intrusion detection: bridging the gap between formal languages and computer security[J]. Theoretical Informatics and Applications, 2006, 40(2): 303-313.

[4] KAI Hong-mei, ZHU Hong-bing, KEIE, et al. A novel intelligent intrusion detection decision response system[J]. IEICE Trans on Fundamentals of Electronics, Communications and Computer Sciences, 2006, E89-A(6): 1630-1637.

[5] MARKATOS E P, ANTONATOS S. Exclusion-based signature matching for intrusion detection[C]//Proc of International Conference on Communications and Computer Networks. New Jersey: IEEE Press, 2002: 146-152.

[6] DHARMAPURIKAR S, KRISHAMURTHY P. Deep packet inspection using parallel bloom filters[J]. IEEE Micro, 2004, 24(1): 52-61.

[7] ANTONATOS S, ANAGNOSTAKIS K G. Generating realistic workloads for network intrusion detection systems[J]. ACM SIGSOFT Software Engineering Notes, 2004, 29(1): 207-215.

[8] TUCK N, CALDER T, VARGHESE B. Deterministic memory-efficient string matching algorithms for intrusion detection[C]//Proc of IEEE INFOCOM. New Jersey: IEEE Press, 2004: 2628-2639.

[9] LIU Rong-ting, HUANG Nan-fang. A fast string-matching algorithm for network processor-based intrusion detection system[J]. ACM Trans on Embedded Computing Systems (TECS), 2004, 3(3): 614-633.

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net