一种面向入侵检测的 BM 模式匹配改进算法*

徐　成[1,3], 孙　伟[1], 戴争辉[1], 喻　飞[2]

(1. 湖南大学 计算机与通信学院, 湖南 长沙 410082; 2. 浙江大学 人工智能研究所, 浙江 杭州 310027; 3. 武汉理工大学 信息学院, 湖北 武汉 430070)

摘　要: 在分析了目前常用的模式匹配算法的基础上, 提出了一种改进的 BM 算法。实验结果表明改进的模式匹配算法能减少比较次数, 有效地提高了匹配速度。

关键词: 入侵检测; 模式匹配; BM算法

中图法分类号: TP312　　**文献标识码**: A　　**文章编号**: 1001-3695(2006)11-0089-03

Improved Algorithm of BM for Intrusion Detection

XU Cheng[1,3], SUN Wei[1], DAI Zheng-hui[1], YU Fei[2]

(1. *College of Computer & Communication, Hunan University, Changsha Hunan* 410082, *China*; 2. *Institute of Artificial Intelligence, Zhejiang University, Hangzhou Zhejiang* 310027, *China*; 3. *College of Information, Wuhan University of Technology, Wuhan Hubei* 430070, *China*)

Abstract: Based on the discussions of the most fashionable pattern matching algorithms at present, an improved algorithm of BM is presented. Experimental data shows the improved algorithm can decrease the times of comparing and enhance the matching speed effectively.

Key words: Intrusion Detection; Pattern Matching; BM Algorithm

随着人类对 Internet 需求的日益增长, 网络安全逐渐成为 Internet 及各项网络服务和应用进一步发展所需解决的关键问题。入侵检测技术是一种主动的网络安全防御措施, 它不仅可以通过监测网络实现对内部攻击、外部攻击和误操作的实时保护, 有效地弥补了防火墙的不足, 而且还能结合其他网络安全产品, 对网络安全进行全方位的保护, 具有主动性和实时性的特点, 是防火墙重要的和有益的补充。自 20 世纪 90 年代初, 人们就开始研究网络入侵检测技术, 目前大多数入侵检测系统采用的均是基于规则的模式匹配策略[1]。检测引擎是入侵检测系统的关键部分, 而数据分析又是检测引擎部分的重要模块。数据分析模块涉及到两个问题, 即如何描述入侵行为与使用什么样的算法来快速准确地检测出入侵行为。本文针对第二个问题对入侵检测中常用的模式匹配算法进行了深入分析, 并提出一个新的 BM 模式匹配改进算法。

1　入侵检测中的模式匹配算法

单模匹配算法主要有 BF 算法、BM 算法及其改进算法。BF 算法又称蛮力匹配算法[2], 该算法实现简单, 但匹配效率很低, 目前已较少应用。下面主要分析在入侵检测系统中应用广泛的 BM 算法及其改进算法。

1.1　BM算法

Boyer 和 Moore 于 1977 年提出了一个全新的模式匹配算法——BM 算法[3]。下面简要说明 BM 算法的原理及实现过程。算法根据预先计算好的两个数组将模式向右移动尽可能远的距离, 这两个数组为 Shift 和 Skip。假设有长度为 n 的文本字符串 $T = T_1 T_2 \cdots T_n$ 和长度为 m(m≤n) 的模式字符串 $P = P_1 P_2 \cdots P_m$, BM 算法的基本思想是:

(1) 匹配从右至左进行。

(2) 若匹配失败发生在 $P_j \neq T_i$, 且 T_i 不出现在模式 T 中, 则将模式右移直到 P_1 位于匹配失败位置 T_i 的右边第一位(即 T_{i+1}位); 若 T_i 在 P 中有若干地方出现, 则选择 $j = \max\{K \mid P_K = T_i\}$, 即通过 Skip 函数计算文本字符 T_i 失配时模式向右移动的距离, 也称坏字符启发。

(3) 若模式后面 k 位与文本 T 中一致的部分有一部分在 P 中其他地方出现, 则可以将 P 向右移动, 直接使这部分对齐, 且要求这一部分尽可能大, Shift 函数通过对已经匹配部分的考查决定模式向右移动的距离, 也称好后缀启发。

由于篇幅有限, 这里不给出 Shift 数组和 Skip 数组的算法, 只给出 BM 算法的 C 语言描述:

```
BM Algorithm
k = m - 1;
while( k < n) {
j = m - 1;
while( ( j > = 0) &&( pat[ j] = = text[ k] ) ) {
j - - ;
k - - ;
}
if( j = = 0)  then match found at text[ k]
else k = k + max( skip[ text[ k] ] , shift[ k] ) ;
}
not match found
```

匹配过程如下:

模式字符串为 EXAMPLE

收稿日期: 2005-07-18; 修返日期: 2005-10-11

基金项目: 国家发展和改革委员会资助项目(计高技 2034); 湖南省自然科学基金资助项目(03jjy3103)

文本字符串为 HERE IS A SIMPLE EXAMPLE

(1) 从右至左匹配。在搜索时先将 P_1 和 T_1 对齐。匹配从 P 的最右端字符开始,比较 P_m 和 T_m,如果匹配成功,则向左移动,依次比较直至 P 全部匹配成功或有不匹配的情况出现,如下所示:

```
EXAMPLE
HERE IS A SIMPLE EXAMPLE
```

(2) 坏字符启发。第一次匹配后发现 $P_m = E$ 与 $T_m = S$ 不匹配,且字符 'S' 没有在模式中出现,根据算法思想(2),模式应右移 m 个位置使 P_1 与 T_{m+1} 对齐,如下所示:

```
        EXAMPLE
HERE IS A SIMPLE EXAMPLE
```

(3) 比较字符 'P' 和 'E' 导致不匹配,但是字符 'P' 在模式中的位置 5 上出现,于是模式向右移动两个位置,如下所示:

```
          EXAMPLE
HERE IS A SIMPLE EXAMPLE
```

(4) 好后缀启发。从右至左逐个比较直到发现文本字符 'I' 与模式中对应的字符 'A' 不匹配,此时注意到后缀 "MPLE" 已经匹配,虽然在模式中没有 "MPLE" 的其他出现,但是模式中的一个前缀 'E' 匹配 "MPLE" 末端,于是模式向右移动六个位置,如下所示:

```
              EXAMPLE
HERE IS A SIMPLE EXAMPLE
```

(5) 比较字符 'P' 和 'E' 导致不匹配,但是字符 'P' 在模式中的位置 5 上出现,于是模式向右移动两个位置,如下所示:

```
                EXAMPLE
HERE IS A SIMPLE EXAMPLE
```

然后从右至左进行匹配,此时,匹配完全成功。

上例使用 BM 算法,需进行 12 次比较,比普通的搜索算法快两倍。实践应用表明,BM 算法已经是比较快的匹配算法,BM 算法预处理时间复杂度为 $O(m+s)$,空间复杂度为 $O(s)$,s 是与 P, T 相关的有限字符集长度,搜索阶段时间复杂度为 $O(mn)$。最坏情况下要进行 $3n$ 次比较,最好情况下的时间复杂度为 $O(n/m)$。

1.2 BMH 算法

BM 算法为了确定在不匹配的情况下最大的可行移位距离而使用了两种启发,即坏字符和好后缀启发。两种启发均能导致最大为 m 的移动距离,但是由于好后缀启发的预处理非常难以理解和实现,Horspool 于 1980 年发表了改进与简化 BM 算法的论文,即 BMH 算法[4]。该算法在移动模式时仅考虑了坏字符启发,即只使用 Skip 函数。

```
BMH Algorithm
k = m - 1;
while( k < n)
{ j = m - 1;
   while( ( j > = 0) && ( pat[ j] = = text[ k] ) ) {
   j - - ;
   k - - ;
}
if( j = = 0) then match found at text[ k]
else k = k + skip[ text[ k] ] ;
}
not match found
```

匹配过程如下:

```
EXAMPLE
HERE IS A SIMPLE EXAMPLE
(‘S’ not in pattern)
      EXAMPLE
HERE IS A SIMPLE EXAMPLE
(‘P’ in pattern)
         EXAMPLE
HERE IS A SIMPLE EXAMPLE
(‘I’ not in pattern)
            EXAMPLE
HERE IS A SIMPLE EXAMPLE
(‘X’ in pattern)
               EXAMPLE
HERE IS A SIMPLE EXAMPLE
```

上例使用 BMH 算法需进行 12 次比较。BMH 算法预处理时间复杂度为 $O(m+s)$,空间复杂度为 $O(s)$,搜索阶段时间复杂度为 $O(mn)$。在一般情况下,BMH 算法比 BM 有更好的性能,它只使用了一个数组,简化了初始化过程,省去了求最大值的计算。

2 SBM 算法

综合分析前面两种算法会发现 BM 算法考虑较全面,但它使用了两个数组,预处理时间开销较大,BMH 算法在其基础上进行了简化,使得算法更简单、高效。我们将两种算法的优点进行了综合并加以改进,得到一个新的模式匹配算法,即 SBM 算法。

在 BM 算法和 BMH 算法匹配过程中,常出现模式的一部分后缀与文本匹配,而模式的前缀却不匹配,在这种情况下,就进行了一些不必要的比较。因此在 SBM 算法中,我们在对模式串与文本字符串进行匹配时采用从模式两端向中间位置交替的匹配顺序,模式匹配先从模式最右端 P_m 开始进行。若 P_m 匹配不成功,则通过 Skip 函数计算出模式向右移动的距离,这与 BM 算法中坏字符启发思想相同;若 P_m 匹配成功,则比较模式 P_1 与文本中相应的字符。若 P_1 匹配不成功,则考查文本中与模式中 P_m 下一个字符对齐的字符,若该字符不出现在模式中,则模式可以向右移动 $m+1$ 个位置,若该字符出现在模式中,则计算其 Skip 函数,然后将模式向右移动相应的长度;若 P_1 匹配成功,则按上述方法依次对 $P_{m-1}P_2P_{m-2}P_3\cdots$ 进行匹配,依此类推,直到匹配过程完成。

算法描述如下:

```
SBM Algorithm
k = m - 1;
while( k < n)
{
while( r < j)
{   = m - 1;
  r = 0;
  i = k;
  t = k - m + 1;
  if( ( j > r) && ( pat[ j] == text[ k] ) )
  {
     j - - ;
     k - - ;
}
else {
   k = k + skip[ text[ k] ] ;
   break;
}
```

```
if( ( r < j) &&( pat[ r]  =  = text[ t] ) )
{
   r + + ;
   t + + ;
}
else {
   k = i + skip[ text[ i + 1] ] ;
   break;
   }
}
if( j < = r)  then match found
}
not match found
```

匹配过程如下:

```
EXAMPLE
HERE IS A SIMPLE EXAMPLE
(‘S’not in pattern)
        EXAMPLE
HERE IS A SIMPLE EXAMPLE
(‘P’in pattern)
            EXAMPLE
HERE IS A SIMPLE EXAMPLE
```

此时,比较模式最右端字符‘E’与文本中相应字符‘E’,匹配成功;然后比较模式中第一个字符‘E’与文本中相应字符,匹配不成功。因为文本中与模式最右端对齐的下一个字符(空格)未在模式中出现,于是模式跳过它向右移动八个位置后如下:

```
                EXAMPLE
HERE IS A SIMPLE EXAMPLE
```

同样的例子使用 SBM 算法,只需将模式移动三次,通过九次比较即可匹配成功,减少了比较次数;同时由于没有求最大值的过程,也减少了算法的时间开销。

算法的预处理时间复杂度为 $O(m+s)$,空间复杂度为 $O(s)$,搜索阶段时间复杂度为 $O(mn)$。该算法在比较右端字符失配时采用 BM 坏字符启发的思想,在比较左端字符失配时采用对文本中与模式最右端对齐的下一个字符进行考查的方法,使得在大多数情况下具有比 BM 算法更大的右移长度,从而有更好的平均性能。

3　算法性能分析

为了对各算法的性能进行比较次数的测试,我们随机地选取了一段纯英文自然语言文本,指定匹配模式为文本末尾插入的长度为 20 个字符的特殊字符串,在同一台计算机上用不同算法进行 20 万次循环匹配,分别统计各算法进行 20 万次循环匹配所消耗的时间。测试结果如图 1 所示,从左至右依次为 BM 算法、BMH 算法、SBM 算法的消耗时间。

4　结束语

模式匹配是当前入侵检测系统中普遍采用的策略之一,其性能的好坏取决于所选用的匹配算法及其实现,高效的算法是提高系统性能的手段之一。在模式匹配中存在以下两个定理:任何字符串匹配算法在最坏情况下必须检查至少 $n-m+1$ 个 Text 中的字符;任何字符串匹配算法至少检查 n/m 个字符。由此可知没有一个算法比 BM 算法有更好的计算复杂度[5],但是可以通过改进来提高匹配算法的平均性能。根据以上的测试结果我们可以得出:SBM 算法在模式匹配过程中比较次数减少,同时又没有增加算法的复杂度,是一种快速的模式匹配算法。

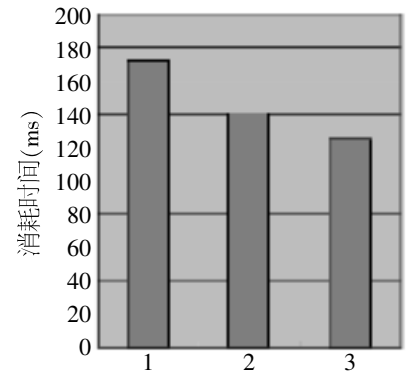

图 1　三种模式匹配算法的性能测试结果

参考文献:

[1] 唐正军. 入侵检测技术导论[M]. 北京:机械工业出版社, 2004. 1-16.

[2] Yan Weimin, Wu Weimin. Data Structure[M]. Beijing: Tsinghua University Press, 1997. 79.

[3] Robert S Boyer, J Strother Moore. A Fast String Searching Algorithm [J]. Communications of the ACM, 1977, 20(10): 762-772.

[4] Nigel Horspool R. Practical Fast Searching in Strings[J]. Software Practice and Experience, 1980, 10(6): 501-506.

[5] 李雪莹, 刘宝旭, 等. 字符串匹配技术研究[J]. 计算机工程, 2004, 30(22): 24-26.

作者简介:

徐成(1962-),男,副教授,主要研究方向为嵌入式系统设计与网络安全;孙伟(1982-),男,硕士研究生,主要研究方向为网络安全;戴争辉(1979-),男,硕士研究生,主要研究方向为网络安全;喻飞(1972-),男,博士研究生,主要研究方向为计算机网络安全。

(上接第 88 页)

[2] Hoffstein, J Pipher, J H Silverman. NTRU: A New High Speed Public Key Cryptosystem[C]. Proc. of Algorithm Number Theory (ANTS Ⅲ), Berlin: Springer-Verlag, 1998. 267-288.

[3] A Kiayias, M Yung. Breaking and Repairing Asymmetric Public-Key Traitor Tracing[C]. Berlin: ACM Workshop on Digital Rights Managements, Springer-Verlag, 2002. 207-224.

[4] K Kurosawa, Y Desmedt. Optimum Traitor Tracing and Asymmetric Schemes[C]. Berlin: EUROCRPYT, Springer-Verlag, 1998. 145-157.

[5] W G Tzeng, Z J Tzeng. A Public-Key Traitor Tracing Scheme with Revocation Using Dynamic Shares[C]. Berlin: PKC, Springer-Verlag, 1992. 207-224.

[6] Y Watanable, G Hanaoka, H Iami. Efficient Asymmetric Public-Key Traitor Tracing without Trusted Agents[C]. Berlin: CT-RSA, Springer-Verlag, 2001. 392-407.

作者简介:

杨文峰(1971-),男,山东昌邑人,助理研究员,硕士研究生,主要研究方向为网络安全;杨波(1963-),男,陕西临潼人,教授,博导,主要研究方向为网络安全、信息隐藏。