文章编号: 1001-0920 (2013) 12-1855-04

一种快速的基于 BM 模式匹配的改进算法

马占飞[1], 杨树英[2], 郭广丰[1]

(1. 内蒙古科技大学 包头师范学院 计算机系，内蒙古 包头 014030;
2. 包头服务管理职业学校 信息技术系，内蒙古 包头 014030)

摘 要: 模式匹配算法是入侵检测系统 (IDS) 中非常重要的一种算法. 在研究和分析几种常用模式匹配算法的基础上, 提出一种快速的基于 BM (Boyer-Moore) 模式匹配的改进算法 —— IBM 算法. 该算法充分利用模式串的末字符和末字符所对应的文本串的后两字符的唯一性, 同时参考文本串本身的信息来提高模式串的移动量, 使得每次失配后, 在保证不丢失匹配成功可能性的前提下尽可能多地向后跳跃. 实验结果表明, 该算法相比其他模式匹配算法, 在检测性能和匹配效率上均具有很大优势, 并且能够有效地提高 IDS 的检测效率和性能.

关键词: 模式匹配算法；入侵检测系统；IBM 算法；模式串；文本串

中图分类号: TP393.08 **文献标志码:** A

A fast improved pattern matching algorithm based on BM

MA Zhan-fei[1], *YANG Shu-ying*[2], *GUO Guang-feng*[1]

(1. Department of Computer，Baotou Teachers College，Inner Mongolia University of Science and Technology，Baotou 014030，China；2. Department of Information Technology，Baotou Vocational School of Service and Management, Baotou 014030，China. Correspondent：MA Zhan-fei，E-mail：mazhanfei@163.com)

Abstract: The pattern matching algorithm is a very important method in the intrusion detection system(IDS). Through analyzing and studying the characteristics of the BM(Boyer-Moore) algorithm and BMHS algorithm, an improved algorithm based on BM(IBM) algorithm is proposed. IBM algorithm integrates the merits of other pattern matching algorithms. The IBM algorithm takes full advantage of the uniqueness of the last character of the pattern string and its corresponding to the next two characters of text string to determine the moving distance of the pattern string. Meanwhile, it also takes into account the information of text string itself to increase the amount of move of the pattern string as much as possible, when the text string and pattern string are mismatch. Theoretical analysis and experimental results show that the IBM algorithm can significantly decrease the moving time and shorten the pattern matching time than other pattern matching algorithms, and can effectively improve the detection efficiency and performance of the IDS.

Key words: pattern matching algorithm; intrusion detection system; improved Boyer-Moore algorithm; pattern string; text string

0 引 言

模式匹配是对字符串的一种重要操作[1], 即搜索指定模式串 P 在正文文本串 T 中的所有出现位置. 模式匹配算法在数据处理、数据压缩、文本编辑、信息检索、病毒检测、内容过滤、网络入侵检测和基因检测等方面得到了广泛应用[2-4]. 模式匹配算法根据一次可匹配模式的多少可分为单模式匹配算法和多模式匹配算法. 在单模式匹配算法中, 比较著名的有: BM (Boyer-Moore) 算法[5]和 KMP (Knuth Morris Pratt) 算法[6]. 一般情况下, BM 算法的检测效率比 KMP 算法快 3 ~ 5倍. 目前, 大多数基于规则的入侵检测系统 (IDS) 的检测引擎, 其核心是模式匹配算法, 而且这些 IDS 大多采用单模式匹配算法[7]. 在 IDS 中, 通过对网络数据包进行字符串匹配来判断入侵行为是一项非常耗时的工作, 尤其在大流量、高带宽的动态网络环境下, 模式匹配算法效率的高低对检测系统的实时性和准确性具有至关重要的作用[8]. 因此, 研究出一种更加高效、实用的模式匹配算法具有重要的实际意

收稿日期: 2012-09-04；**修回日期:** 2012-11-28.

基金项目: 国家自然科学基金项目(61163025)；内蒙古自治区自然科学基金项目(2010BS0904)；内蒙古自治区高等学校科学研究基金重点项目(NJ10162)；内蒙古自治区高等学校科学研究基金项目(NJZY07116).

作者简介: 马占飞(1973−), 男, 教授, 博士, 从事计算机网络技术与信息安全、人工智能等研究；杨树英(1971−), 女, 副教授, 硕士, 从事计算机网络技术与信息安全的研究.

义.

本文在研究和分析IDS中典型的单模式匹配算法——BM算法和BMHS算法(简化的BM算法)的基础上, 提出一种新的基于BM算法的改进的模式匹配算法——IBM (Improved BM)算法. 该算法充分利用模式串的末字符和末字符所对应的文本串的后两字符的唯一性, 同时考虑文本串本身的信息来提高模式串的移动量, 减少窗口比对次数, 使得每次匹配失败后, 在保证不丢失匹配成功可能性的前提下, 尽可能多地向后跳跃, 使模式串获得更大的右移距离, 从而加快模式匹配的速度, 以便更好地适应IDS对模式匹配算法高效性的要求.

1 模式匹配算法分析

目前, 关于模式匹配的算法有很多, BM算法和BMHS算法是目前较为著名的单模式匹配算法, 在IDS中的应用也较为广泛[9]. 模式匹配算法描述如下.

对于给定的两个模式匹配字符串, 文本串T是长度为n的字符串, 记为

$$T = T_1T_2\cdots T_n,\ T_i \in \Sigma, i = 1, 2, \cdots, n; \quad (1)$$

模式串P是长度为m的字符串, 记为

$$P = P_1P_2\cdots P_m,\ P_j \in \Sigma, j = 1, 2, \cdots, m. \quad (2)$$

一般情况下$m \ll n$. 若能在文本串T中查询到模式串P, 则称其为匹配成功, 否则称其为匹配失败.

1.1 BM算法

BM算法, 即Boyer-Moore算法, 是由Boyer等[5]于1977年提出的一种单模式匹配算法. 该算法通过构造坏字符(bad-char)跳跃函数和好后缀(good-suffix)跳跃函数, 采用自右向左扫描文本串T的方式与模式串P的字符进行匹配. 在最坏情况下, BM算法的搜索时间也呈线性分布, 因此它被认为是当今应用较为广泛、性能较好的算法. 该算法的基本思想如下.

1) 模式串P与文本串T比较时, 首先从左端对齐, 即字符P_1与字符T_1对齐; 然后从右向左进行匹配, 即先判断字符P_m与字符T_m是否匹配, 若匹配成功, 则继续判断字符P_{m-1}与字符T_{m-1}是否匹配, 直至字符$P_1 \sim P_m$都匹配成功或者出现不匹配的字符为止.

2) 坏字符跳跃函数构造. 在BM算法的匹配过程中, 若发现文本串T中某个字符x与模式串P中的某字符不匹配, 即出现了坏字符, 则利用该坏字符在模式串P中出现的情况来移动模式串P. 坏字符跳跃构造的移动函数如下:

$$\mathrm{Dist}[x] = \begin{cases} m,\ x \neq P_j (1 \leqslant j \leqslant m),\ \text{即 } x \text{ 在 } P \text{ 中未出现;} \\ m - j,\ j = \max\{j | P_j = x, 1 \leqslant j \leqslant m\}, \\ \qquad \text{即 } x \text{ 在 } P \text{ 中出现.} \end{cases} \quad (3)$$

其中: $\mathrm{Dist}[x]$为模式串P的右移距离, m为模式串P的长度, j为字符x在模式串P中的最右位置, $x \neq P_j (1 \leqslant j \leqslant m)$表示$x$在$P$中未出现, $j|P_j = x (1 \leqslant j \leqslant m)$表示$x$在$P$中出现.

3) 好后缀跳跃函数构造. 在BM算法扫描过程中, 当发现文本串T中某个字符与模式串P不匹配时, 表明已有部分字符匹配成功, 此时应充分利用已经匹配好的后缀的信息来移动模式串P. 好后缀跳跃构造的移动函数如下:

$$\begin{aligned} &\mathrm{Skip}[j] = \\ &\min\{s | (P_{j+1}, \cdots, P_m) = (P_{j-s+1}, \cdots, P_{m-s}) \,\&\,\& \\ &(P_j \neq P_{j-s})(j > s), (P_{s+1}, \cdots, P_m) = \\ &(P_1, \cdots, P_{m-s})(j \leqslant s)\}. \end{aligned} \quad (4)$$

其中: $\mathrm{Skip}[j]$为模式串P的右移距离, m为模式串P的长度, j为当前所匹配字符的位置, s为两部分匹配子串的距离.

4) 当文本串T出现与模式串P不匹配的字符时, 根据坏字符跳跃函数和好后缀跳跃函数来计算偏移值, 取两者中的较大值作为模式串P的右移距离.

根据BM算法的思想, 假设文本串T =“substringsearching”, 模式串P =“ching”, 即$m = 5, n = 19$, 以此来分析BM算法的匹配过程(匹配失败的字符用黑体表示), 如表1所示.

表1 BM算法的模式匹配过程

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	s	u	b	–	**s**	t	**r**	i	n	g	s	e	a	r	**c**	h	i	n	g
1	c	h	i	n	**g**														
2						c	**h**	i	n	g									
3											c	h	i	n	**g**				
4															c	h	i	n	g

BM算法的匹配过程如下: 第1次匹配. 将模式串P与文本串T左端对齐, 从右向左进行匹配. 模式串P的末字符P_5与T_5匹配不成功, 且字符T_5在P中不存在, 则将模式串P移动到字符P_1与T_6对齐的位置. 第2次匹配. 字符P_2与T_7不匹配, 且字符T_7在P中不存在, 则将模式串P移动到字符P_1与T_{11}对齐的位置. 第3次匹配. 模式串P的末字符P_5与T_{15}仍然匹配不成功, 因为字符T_{15}在模式串P中存在, 所以将模式串P中相同的字符P_1移动到与T_{15}对齐的位置进行匹配, 此次匹配成功. 利用BM算法, 模式串P进行3次移动、11次匹配操作完成了匹配工作.

1.2 BMHS算法

1990年, Sunday[10]提出了简化的BM算法, 即BMHS算法. BMHS算法将字符失配与计算右移量独立, 即计算右移量函数时并不关心文本串T中哪个字符造成了失配, 而是利用文本串T当前匹配窗口后面

的一个字符T_{i+m}来决定模式串P的右移量. BMHS算法只采用了BM算法中的坏字符跳跃规则, 其坏字符跳跃函数定义如下:

$$\mathrm{Dist}[T_{i+m}]=\begin{cases} m+1,\ T_{i+m}\neq P_j(1\leqslant j\leqslant m),\ \text{即}\\ \qquad T_{i+m}\text{在}P\text{中未出现};\\ m-j,\ j=\max\{j|P_j=T_{i+m},1\leqslant j\leqslant m\},\text{即}\\ \qquad T_{i+m}\text{在}P\text{中出现}. \end{cases} \tag{5}$$

其中: $\mathrm{Dist}[T_{i+m}]$为模式串P的右移距离, m为模式串P的长度, j为字符x在模式串P中的最右位置$P_j \neq T_{i+m}(1\leqslant j\leqslant m)$表示$T_{i+m}$在$P$中未出现, $j\,|P_j = T_{i+m}(1\leqslant j\leqslant m)$表示$T_{i+m}$在$P$中出现.

仍然以假设文本串T="sub-stringsearching"、模式串P="ching"、$m=5$、$n=19$为例来分析BMHS算法的匹配过程, 如表2所示. 首先进行预处理, 生成BMHS算法的坏字符跳跃表, 如表3所示, 当前窗口的下一个字符用x表示, $\mathrm{Dist}[x]$表示模式串P的右移距离.

表 2　BMHS 算法的模式匹配过程

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	s	u	b	−	**s**	t	r	i	n	g	**s**	e	a	r	c	h	**i**	n	g
1	c	h	i	n	**g**														
2						c	h	i	n	**g**									
3													c	h	i	n	**g**		
4															c	h	i	n	g

表 3　BMHS 算法的坏字符跳跃表

x	c	g	h	i	n	其他字符
$\mathrm{Dist}[x]$	5	1	4	3	2	6

BMHS算法的匹配过程如下: 第1次匹配. 将模式串P与文本串T左端对齐, 从右向左进行匹配. 在文本串T的T_5处失配, 此时当前窗口下一个字符T_6为t, 查询坏字符表得$\mathrm{Dist}[\mathrm{t}]=6$, 所以模式串$P$向右移动6位. 第2次匹配. 在文本串$T$的$T_{11}$处失配, 当前窗口的下一个字符$T_{12}$为e, 查询坏字符表得$\mathrm{Dist}[\mathrm{e}]=6$, 所以模式串$P$向右移动6位. 第3次匹配. 在文本串$T$的$T_{17}$处失配, 当前窗口的下一个字符$T_{18}$为n, 查询坏字符表得$\mathrm{Dist}[\mathrm{n}]=2$, 故模式串$P$向右移动2位. 进行第4次匹配, 第4次匹配成功. 在BMHS算法的匹配过程中, 模式串P进行3次移动、8次匹配操作完成了匹配工作.

2　改进的模式匹配算法(IBM算法)

模式匹配算法的关键是通过模式串P的不断右移来完成对文本串T的查询[11-12]. 因此, 在匹配失败时, 模式串P每次都至少向右移动一个字符的距离, 据此可对BM算法思想进行改进, 使得每次匹配失败后, 在保证不丢失匹配成功可能性的前提下尽可能多地向后跳跃, 使模式串P获得更大的右移距离, 从而加快模式匹配的速度, 提高检测算法的效率. IBM (Improved BM) 算法思想如下.

1) 首先将模式串P与文本串T左对齐, 然后从右向左进行匹配, 如果模式串P与文本串T对应字符相同, 则继续进行下一个字符的比较.

2) 当发生失配时, 查看当前匹配窗口对应文本串T的后一位字符T_{i+m}在模式串P中是否存在, 如果存在, 则向右移动模式串P使其对应字符与T_{i+m}对齐; 如果不存在, 则再查看字符T_{i+m}的下一位字符T_{i+m+1}与模式串P首字符P_1是否相同, 如果相同, 则将模式串P向右移动$m+1$位; 如果不同, 便可以直接跳过该字符, 即将模式串P向右移动$m+2$位.

3) 按照以上规则进行匹配, 直至在文本串T中找到匹配的模式串P, 或者完成对文本串T的搜索后也未找到匹配的模式串P, 算法终止.

当模式串P在文本串T中的字符x处发生失配时, 根据IBM算法思想, 构造的移动函数如下:

$$\mathrm{Dist}[x]=\begin{cases} m+1,\ P_j\neq T_{i+m}\,\&\&\,T_{i+m+1}=P_1(1\leqslant j\leqslant m);\\ m+2,\ P_j\neq T_{i+m}\,\&\&\,T_{i+m+1}\neq P_1(1\leqslant j\leqslant m);\\ m-j,\ j=\max\{j|P_j=T_{i+m},1\leqslant j\leqslant m\}. \end{cases} \tag{6}$$

根据IBM算法思想, 同样以假设文本串T="sub-stringsearching"、模式串P="ching"、$m=5$、$n=19$为例来分析IBM算法的匹配过程(匹配失败的字符用黑体表示), 如表4所示.

表 4　IBM 算法的模式匹配过程

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	s	u	b	−	**s**	t	r	i	n	g	s	**e**	a	r	c	h	i	n	g
1	c	h	i	n	**g**														
2							c	h	i	n	**g**								
3																c	h	i	n	**g**

IBM算法的匹配过程如下: 第1次匹配. 将模式串P与文本串T左端对齐, 从右向左进行匹配. 模式串P末字符P_5与文本串T的字符T_5不匹配, 查看文本串T窗口后继字符T_6; 因为字符T_6在模式串P中不存在, 所以继续查看T_6的后继字符T_7; 由于字符T_7与模式串P的首字符P_1不相同, 根据匹配规则, 将模式串P向右移动$m+2=7$位. 第2次匹配. 由于模式串P末字符P_5与文本串T的字符T_{12}不匹配, 当前匹配窗口后继字符T_{13}在模式串P中不存在, T_{13}的后继字符T_{14}与模式串P的首字符P_1也不相同, 根据匹配规则, 将模式串P向右移动$m+2=7$位. 第3次匹配, 第3次匹配成功. 根据IBM算法匹配过程, 模式串

P 在经过 2 次移动、7 次匹配操作便完成了匹配工作.

通过上述分析和比较, IBM 算法相比 BM 算法和 BMHS 算法, 不仅减少了模式串 P 的移动次数和字符比较次数, 而且算法效率有了一定提高. 特别当文本串 T 长度较大时, IBM 算法效率更加明显.

3 IBM 算法测试与分析

为进一步验证 IBM 算法的有效性, 本文采用如下实验平台: CPU为 Intel 英特尔酷睿 2 双核 E7500; 操作系统为 Windows XP Professional 32 位; 内存为 2GB; 编程语言为 Visual C++; 编译环境为 Visual Studio 2008, 并通过同一主程序调用.

实验 1 从网上选取一段经典的英文文学作品作为测试文本, 大小为 32 MB, 内容包括“student”和“learn”两个字符串, 分别定义为模式串 A 和模式串 B. 实验中选取这两个字符串作为模式串, 分别采用 BM 算法、BMHS 算法和 IBM 算法进行测试和分析, 测试结果如表 5 所示.

表 5 各算法的匹配次数及 CPU 占用时间

算法	模式串 A		模式串 B	
	匹配次数	CPU 时间 / ms	匹配次数	CPU 时间 / ms
BM	14 062	4.23	128 267	10.26
BMHS	13 587	3.88	121 429	9.79
IBM	11 208	1.89	106 922	7.85

由表 5 实验数据可以看出, 与 BM 算法和 BMHS 算法相比, IBM 算法在模式匹配过程中, CPU 占用时间和匹配次数均有较大幅度的提升. 其中: 模式串 A 的字符匹配次数分别比 BM 算法和 BMHS 算法提高了 20.30 % 和 17.51 %, 模式串 B 的字符匹配次数分别比 BM 算法和 BMHS 算法提高了 16.64 % 和 11.95%; 模式串 A 的 CPU 占用时间分别比 BM 算法和 BMHS 算法提高了 55.32 % 和 51.29 %, 模式串 B 的 CPU 占用时间分别比 BM 算法和 BMHS 算法提高了 23.49 % 和 19.82%.

实验 2 从某学习网站下载一测试数据集 Text (纯英文字符) 作为测试文本, 大小为 60 MB. 选用不同长度的字符串作为测试模式串 (模式串长度分别为 5, 15, 25), 分别采用上述 3 种算法进行模式匹配实验, 并记录各算法的运行时间, 如表 6 所示.

表 6 算法运行时间开销对比表 ms

模式串长度 / 位	BM	BMHS	IBM
5	11	9	7.6
15	10	8.2	6.7
25	7.1	4.5	3.3

由表 6 的实验数据可以看出: 对同一文本串进行模式匹配, 模式串的长度选取为 5 位字符长度时, 相比 BM 算法和 BMHS 算法, IBM 算法所用时间分别缩短了 30.9 % 和 15.6 %; 模式串的长度选取为 15 位字符长度时, 相比 BM 算法和 BMHS 算法, IBM 算法所用时间分别缩短了 33.0 % 和 18.3 %; 模式串的长度选取为 25 位字符长度时, 相比 BM 算法和 BMHS 算法, IBM 算法所用时间分别缩短了 53.5 % 和 26.7%.

通过以上两组实验数据的分析和对比, 在模式串长度相同的情况下, 与 BM 和 BMHS 两种算法相比, IBM 算法所用的时间开销和匹配次数是最低的, 而且随着模式串长度的增加, IBM 算法的效率更为明显.

4 结 论

本文通过对 BM 模式匹配算法和简化的 BM 模式匹配算法 (BMHS 算法) 的研究和分析, 提出了一种改进的、高效的模式匹配算法 —— IBM 算法. IBM 算法不仅能够有效地减少模式串的移动次数, 而且缩短了模式匹配时间, 从而使入侵检测系统 (IDS) 的检测性能有较大幅度的提升. 实验结果表明, 在相同的网络环境下, IBM 模式匹配算法相比传统的 BM 算法和 BMHS 算法, 在检测性能和匹配效率上都具有很大优势. 因此, 将 IBM 算法应用于入侵检测系统 (Snort) 的检测引擎中, 能够明显地提升 Snort 的检测性能. 然而, 由于该算法仍为单模式匹配算法, 每次只能对一种模式进行匹配, 相对于多模式匹配算法而言性能略低, 今后还需对算法做进一步改进, 以适应大流量、高带宽的动态网络入侵检测环境.

参考文献(References)

[1] Wang Minjie, Zhu Lianxuan. Research about pattern matching algorithm[J]. Advances in Intelligent and Soft Computing, 2012, 127: 27-32.

[2] külekci M O, Vitter J S, Xu B. Fast pattern-matching via k-bit filtering based text decomposition[J]. The Computer J, 2012, 55(1): 62-68.

[3] Ong C H L, Ramsay S J. Verifying higher-order functional programs with pattern-matching algebraic data types[C]. Proc of the 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. Austin, 2011: 587-598.

[4] Gawrychowski P. Optimal pattern matching in LZW compressed strings[C]. Proc of the 22th Annual ACM-SIAM Symposium on Discrete Algorithms. San Francisco, 2011: 362-372.

[5] Boyer R S, Moore J S. A fast string searching algorithm[J]. Communications of the ACM, 1977, 20(10):762-772.

[6] Knuth D E, Morris J H. Fast pattern matching in strings[J]. SIAM J on Computing, 1977, 6(2): 323-350.

[7] Zhong Qiuxi, Wan Hui, Xie Peidai, et al. An efficient packet pre-filtering algorithm for NIDS[J]. Lecture Notes in Electrical Engineering, 2012, 126: 113-120.

[8] Ouyang W, Tombari F, Mattoccia S, et al. Performance evaluation of full search equivalent pattern matching algorithms[J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2012, 34(1): 127-143.

[9] Wang Xiaoqiang. Study on genetic algorithm optimization for support vector machine in network intrusion detection[J]. Advances in Information Sciences and Service Sciences, 2012, 4(2): 282-288.

[10] Sunday D M. A very fast substring search algorithm[J]. Communications of the ACM, 1990, 33(8): 132-142.

[11] Martin S, Ouelhadj D, Beullens P, et al. A generic agent-based framework for cooperative search using pattern matching and reinforcement learning[R]. Hants: Department of Mathematics, University of Portsmouth, 2012.

[12] Lin Cheng hung, Chang Shih chieh. Efficient pattern matching algorithm for memory architecture[J]. IEEE Trans on Very Large Scale Integration Systems, 2011, 19(1): 33-41.