The improving pattern matching algorithm of intrusion detection

Zhaoyang Qu, Xiaobo Huang

Dept. of Information Engineering Institution, North East Dianli University, Jilin Jilin 132012, China

Abstract

Pattern matching algorithm is usually used in intrusion detection system. During the detection process, the efficiency of pattern matching algorithm determines the performance of the intrusion detection system. However the low efficiency has became the main shortcomings in the pattern matching algorithm. Based on this point, this paper proposed an improved algorithm, increased the skip distance of the text string pointer, and reduced the number of comparisons. Experimental results show that the algorithm can effectively improved the efficiency of string matching.

© 2011 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.

Selection and/or peer-review under responsibility of [CEIS 2011]

Keywords: pattern matching; intrusion detection efficiency; E-BM algorithm

1. Introduction

With the rapid development of Internet technology, network security situation is also becoming more severe and complex, the traditional firewall technology has been difficult to guarantee network security. Intrusion detection system (IDS), as a proactive security technology, can monitor network traffic and host logs automatically, then detection and response to suspicious events rapidly, which greatly improved the network security. The intrusion detection engine is a key part of the IDS. With the increasing of network
traffic, detection speed has become an important indicator to measure the performance of intrusion
detection engine. The principle of the detection engine, which is based on feature detection, is to match
with the network packet, then discover the network intrusion and alarm [1]. Thus the efficiency of pattern
matching became a bottleneck to improve intrusion detection efficiency. This article focuses on one of the
most widely used algorithm, BM algorithm, to better meet the efficiency requirements of IDS.

2. BM algorithm implementation

The basic process of BM algorithm: set the target text string as \( T \), mode string as \( P \). First, let \( T \) and \( P \)
left-aligned, then matched from right to left.

If mismatched, BM algorithm will use two heuristic rules, that is bad character and good suffix rules,
to calculate the distance that the text string pointer move to the right until the matching process finished
[2]. Bad character and good suffix was shown in Fig.1.

![Fig. 1. Bad character and good suffix](image)

In Fig.1, the first mismatched character (underlined character) is a bad character; the matched part
(bold part) is a good suffix.

- Bad Character
  - In the process of scanning from right to left, if a character \( X \) mismatched, then match by the following
two ways:
    1. If the character \( X \) does not appear in the pattern \( P \), the \( m \) characters starting from character \( X \)
is clearly not match successfully with \( P \), you can skip directly to the region.
    2. If \( X \) appears in the pattern \( P \), align with \( X \).

  Set \( \text{skip}(X) \) as the right skip distance of \( P \), set \( m \) as the length of pattern string \( P \), \( \text{max}(X) \) is the right
position of character \( X \) in pattern string \( P \). Mathematical formula is as follows:

\[
\text{skip}(X) = \begin{cases} 
  m - \text{max}(X) & \text{if } X \text{ does not appear in } P \\
  m - \text{max}(X) & \text{if } X \text{ appears in } P
\end{cases}
\]

Underlined part of the Fig.2.(a) there was a mismatching has happened. \( \text{skip} = 5 - 3 = 2 \), so \( P \) should
skip 2 positions to right. As shown in Fig.2.(b) after moving.

![Fig. 2. (a) A mismatching happened; (b) The position after moving](image)

- Good suffix rule
  - If there are some characters have matched before mismatch happened, analysis based on the following
two conditions:
    1. If the matched part \( P' \) appeared in a position \( t' \) again, and the previous character of the position \( t' \) is
       not the same as the previous character of the position \( t \), \( P \) will be shifted to the right so that \( t' \) can align to
       the location of \( t \).
If the matched part $P'$ didn’t appear again in $P$, find $X$, the longest prefix of $P$, which has matched with $P'$’s suffix $P''$, then move $P$ to right, so that $X$ can align to the location of $P''$.

Expressed as a mathematical formula: assuming shift $j$ is the distance which $P$ skips to right, $m$ is the length of pattern string $P$, $j$ is the current matched character position, $s$ is the distance between $t'$ and $t$ (shown in above I) or the distance between $X$ and $P''$ (shown in above II).

$$\text{shift}(j) = \min\{s \mid (p[j+1...m]=p[j...s+1...m...s]) \land \&\& (p[j] \neq p[j+s]), p[s+1...m]=p[1...m][j>s]\}$$

(2)

We interpret the process with the following examples.

In Fig.3, the matched part $cab$ (the bold part) did not re-appear in pattern string $P$.

In Fig.4.(a), the suffix $T'$ (the bold part) match with $P'$ (the bold part), which is the prefix of $P$, then $P'$ skips to $T'$’s position(Fig.4.(b)).

BM algorithm takes the max value between $\text{skip}(X)$ and $\text{shift} \ j$ as the skip distance.

The time complexity of BM algorithm pretreatment is $O \ m \ s$, space complexity is $O(s)$, $s$ is the length of the limited character set related to $P$ and $T$. The time complexity of search phase is $O \ m \ n$. In the best case time complexity is $O \ n / m$, in the worst-case time complexity is $O \ m \ n$ [3].

BMH algorithm is an improved and simplified algorithm from BM. It only considered "bad character" strategy when moving pattern string. It will first compare the character which text string pointer pointed and the last character of pattern, if they are equal then compare the remaining $m-1$ characters. No matter which character caused the text to mismatch, there will be a character which alignment with the last position of the pattern string in the text, to determine the pattern string’s shift to the right.

If the pattern string mismatch with a character string, and the character which is aligned with the right character of the pattern string is not in the pattern string, the pattern string should skip to the right $m$ positions (assuming the pattern string’s length is $m$); If it was in the pattern string, the pattern string should skip to right with the distance which is calculated according to the skip array.

The following is an algorithm for calculating skip array [4]:

```c
int *GetSkip(char*p, int pLen)
{
    int i, *skip=(int*)malloc(|\Sigma|\sizeof(int));
    for(i=0;i<|\Sigma|;i++) skip[i]=pLen;
    for(i=0;i<pLen-1;i++) skip[p[i]]=pLen-i-1;
    return skip;
}
```

$|\Sigma|$ is the number of characters in the string, for example, $|\Sigma|$ is equal to 26 if a string is made up of alphanumeric.

The following is the BMH matching algorithm:
int BmhSearch(char *t, int tLen, char *p, int pLen, int *skip)
{
    // t is a character pointer to text string, p is a pointer to the address of pattern string, skip is an array of characters moving distance.
    int i=0, j;
    while(i <= tLen-pLen)
    {
        for(j=pLen-1; j>=0 && p[j] == t[j+i]; j--);
        if(j<0) return i;
        i+=skip[t[i]];
    }
    return -1;
}

In the worst case, the algorithm complexity of BMH is \( O(mn) \) and in the best case its time complexity is \( O(n/m) \) in theory. Under normal circumstances, BMH algorithm has better performance than the BM algorithm.

3. The improved BM algorithm

As BMH algorithm abandon good suffix completely, there always exists the case that a part of pattern string match with the main string while the prefix does not match in a real intrusion detection environment, so matching efficiency of bad character is lower than that of good suffix \[5\]. According to the study of BM algorithm and BMH algorithm, propose an improved algorithm. This algorithm first introduces an array \( F \) to label the frequency that every character appears in pattern string, then matching with the order of frequency, to increase mismatching probability, decrease match time and match numbers effectively.

3.1. Frequency calculation when character appears in pattern string

To calculate the frequency that pattern string appears, first to save them into an array \( Q \), then to order characters in accordance with the frequency of small to large, next to string matching with the sorting order\[6\].

In this way, the probability that characters mismatch could be increased, and matching time could be decreased. List the means that frequency calculation in pattern string character:
1) Statistics the frequency that every character appears in pattern string.
2) Order each character in accordance with the occurrence frequency of small to large. The smaller the frequency, the more priority to match.
3) The ending character takes precedence to match.
List the function to calculate the frequency that pattern string appears.

```c
void frequent_Q(char *P, int *Q){
int i = 0, j = 0, num = 0;
for (i=0; i < strlen(P); i++) {
    num = 0;
    for (j = 0; j<strlen(p)&&i!=j; j++) {
        if (P[i] == P[j]) num++;
    }
    Q[i] = num; // Save the frequency that corresponding character appears in pattern string into array Q.
}
```
3.2. E-BM algorithm

The algorithm is mainly improved the text string pointer’s skip distance when match failed. The following describe the calculation of distance skipped.

(1) Compare $t[m]$ with $P[m]$ from right to left, and execute (2) or (3) according to comparison.

(2) If the match is successful, comparing other correspond character continually accord with the frequency priority, and check if the frequency of that character is 1 at the same time, that is to say that character makes the value of $Q_i$ equal to 1. If it is, save the pointer $i$ that first matched with a variable flag until mismatched. When match fails, altering the text string pointer to flag + m, and then to begin a new matching. If none of the value in $Q_i$ is 1 before match failure, it indicates that the characters which have matched appeared multiple times, to execute (3).

(3) If the matching is failure, combine the character which is align with the last character of pattern string and its next character in text string to an entirety character $M$, then check if it appears in pattern $P$. If it does, according to the offset of pre-calculated to move the text string pointer, to make the last right character $M$ in pattern $p$ align with character $M$ in pattern $T$. If not, look that whether the second character of that combination appears in pattern $P$. If it does, the text string pointer's skip distance is $M$, else is $m+1$.

3.3. The time complexity analysis of E-BM algorithm

The most important factor that affects the efficiency of matching algorithm is not the maximum amount of shift, but the probability of producing the maximum times of shift and times of character comparison. If the character in text string, which aligning with the ending one in pattern string, does not appear in pattern string, both BM algorithm and BMH algorithm has the maximum times of shift [7]. As for E-BM algorithm this will happen in three cases. First is the character that does not appear in pattern sting becomes the ending character's next one in text string. Second is the character appears only once in pattern string before matching failure. The third is the ending character and its next one is not appear in pattern string. So, E-BM algorithm has higher probability in producing maximum shift and fewer character comparison times than BM algorithm and BMH algorithm. As the pattern string is the fixed rules of IDS, we do not need to calculate the character frequency of all rules before every matching, as long as calculate once and then save into an array. In this case, time spend on calculating the character frequency can be ignored. In the best case, time complexity of BM algorithm and BMH algorithm are $O(n/m)$ and $O(n/m+1)$[8]. But considering the maximum shift, the probability to produce the maximum shift and character comparison times, E-BM algorithm has higher matching rate than BM algorithm and BMH algorithm.

4. Algorithm testing and evaluation

Experimental environment: operating system is Microsoft Windows XP Professional SP3, CPU is Intel 2.00GHz, and memory is 2G. Each algorithm run 50 times, take the averaged running time. The data used in the experiment were selected from a natural English language text, it has 327,000 characters, 293.7 KB, and the pattern string has 14 characters. Test the running time and the text pointer moves times with BM, BMH and E-BM algorithm.

Test results: the comparison times, shown in Fig.5.(a).the running time, shown in Fig.5.(b).

The experiment results show that E-BM algorithm has better performance than the BM algorithm and BMH algorithm. On the times of character comparisons, E-BM algorithm is less than the BM algorithm and BMH algorithm, reduced by 16.67% and 9.87%; E-BM algorithm is increased by 14.29% and 4.25%
than the BM algorithm and BMH algorithm in time performance. The foregoing analysis shows that, E-BM algorithm can significantly reduce the character matching times and pattern strings’ moving times; it can also improve the matching speed without increasing the space complexity. E-BM algorithm’s performance is higher than the BM algorithm and BMH algorithm, therefore, with great practical value.

5. Conclusion

As the development of network technology is rather rapidly today, there is a serious performance bottleneck in the IDS which based on BM pattern matching algorithm, the network environment can not meet the requirements of high flow, the packet loss rate and the false negative rate is also rising. This paper improved the BM algorithm after a detailed study. Experimental results show that the improved algorithm is more efficiency than the original one. It is provide a valuable reference for the future IDS developer to design the next more efficiency intrusion detection system.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (No.51077010) and Natural Science Foundation of Jilin Province (No.20101517).

References