International Journal of Computer and Internet Security.
ISSN 0974-2247 Volume 5, Number 1 (2013), pp. 11-18

http://www.irphouse.com

Single-Keyword Pattern Matching Algorithms for Network Intrusion Detection System

K. Prabha[*] and Dr. S. Sukumaran

[1]Ph.D. Research Scholar, Erode Arts and Science College,
Erode–638009, Tamil Nadu, India.
E-mail: prabhaeac@gmail.com
[2]Associate Professor of Computer Science, Erode Arts and Science College,
Erode–638009, Tamil Nadu, India.
E-mail: prof.sukumaran1@gmail.com

Abstract

The Network Intrusion Detection System (NIDS) is an important part of any modern network. One of the important processes in NIDS is inspecting of individuals' packets in network traffic, deciding if these packets are infected with any malicious activities. This process, which is called content matching, is done via string matching algorithms. The content matching is considered the heart of NIDS. The content matching phase consumes most of the processing time inside the NIDS and slowed down around 70% of NIDS performance. In this case, it is difficult for NIDS to distinguish between normal network packets and abnormal network packets and consequently drop numbers of network packets. New algorithms are needed to enhance the matching since enormous packets are passing through the network every second. In this paper we presented a survey of single keyword pattern matching algorithms for NIDS.

Introduction

Over the years, pattern-matching has been routinely used in various computer applications, for example, in editors, retrieval of information (from text, image, or sound), and searching nucleotide or amino acid sequence patterns in genome and protein sequence databases. The present day pattern-matching algorithms match the pattern exactly or approximately within the text. An exact pattern-matching is to find all the occurrences of a particular pattern ($x= x_1 x_2 ... x_m$) of m-characters in a text ($y= y_1 y_2 ... y_n$) of n-characters which are built over a finite set of characters of an alphabet set. The direct way to this problem is to compare the first m-characters of the text and

the pattern in some predefined order and, after a match or a mismatch, slide the entire pattern by one character in the forward direction of the text. This process is repeated until the pattern is positioned at the (n-m+1) position of the text. This approach is commonly known as a brute-force method. To facilitate this task, several algorithms have been proposed, and these have their own advantages and limitations based on the pattern length.

Network intrusion detection systems are fundamental security applications that are growing in popularity in various network environments. The heart of almost every modern NIDS has a string matching algorithm. The NIDS uses string matching to compare the payload of the network packet and/or flow against the pattern entries of intrusion detection rules [1, 2]**.**

Single-Keyword Pattern Matching Algorithms

String matching algorithms are widely used in many applications which includes the NIDS [9]. These string matching algorithms are used to inspect the content of packets and identify the attacks signature in NIDS. String matching consists of finding one, or more generally, of all the occurrences of a search string in an input string. In NIDS applications, the pattern is the search string, while the payload is the input string. If more than one search string simultaneously matches against the input string, this is called multiple pattern matching. Otherwise, it is called single pattern matching. In this work, we have taken only the single keyword pattern matching algorithms.

The Boyer-Moore Algorithm (BM)

The Boyer-Moore algorithm is one of the famous exact string matching algorithms that used in single pattern matching and it considers very fast in its performance. The algorithm uses two tables or functions, which is used to move the sliding window to the right. The first table is called “bad character shift”, while the second table called “good suffix shift”. The algorithm is faster when it is working with small pattern size, but it is slower when it is working with large pattern size [11]. The BM algorithm is given below:

Algorithm BoyerMoore (T, P, S)

```
L=occFunction ()
i=m-1
j=m-1
while i > n-1
{
if T[i]=P[j]
if j=0
return i //match at i
else
i=i-1
j=j-1
```

```
else //character-jump
l=L[T[i]]
i=i + m–min(j, 1 + l)
j=m-1
}
return -1 //no match
void occFunction()
{
char a;
int j;
for (a=0; a < alphabetsize; a++)
occ[a] =- 1;
for (j=0; j < m; j++)
{
a=p[j];
occ[a]=j;
}
}
```

The algorithm preprocesses the pattern and creates two tables, which are known as Boyer-Moore bad character (bmBc) and Boyer-Moore good-suffix (bmGs) tables. For each character in the alphabet set, a bad-character table stores the shift value based on the occurrence of the character in the pattern. On the other hand, a good-suffix table stores the matching shift value for each character in the pattern. The maximum of the shift value between the bmBc (character in the text due to which a mismatch occurred) dependent expression and from the bmGs table for a matching suffix is considered after each attempt, during the searching phase. This algorithm forms the basis for several pattern-matching algorithms.

The Horspool Algorithm (HP)

The Horspool algorithm is a derivative of Boyer-Moore and is easy to implement. Horspool algorithm is considered to be one of string matching algorithm that used in network intrusion detection system based on Boyer Moore algorithm. Horspool algorithm is easy and works in any order. Snort NIDS uses a modified version of the algorithm called Boyer-Moore-Horspool algorithm to maintain memory usage and speed up during searching phase. Unlike Boyer-Moore algorithm, which uses two tables; bad character shift and good suffix shift, the Horspool algorithm uses only one table (bad character shift).When the alphabet size is large and the length of the pattern is small, it is not efficient to use Boyer-Moore's bad-character technique. Instead, it is always enough to find the bad-character shift of the right-most character of the window to compute the value of the shift. These shift values are computed in the preprocessing stage for all the characters in the alphabet set. Hence, the algorithm is

more efficient in practical situations where the alphabet size is large and the length of the pattern is small.

The Quick-Search Algorithm (QS)

Quick search algorithm is the modified version of BoyerMoore algorithm. Like the Horspool algorithm it uses only one table called "bad-character shift", and working on one of two pattern shifting. Quick search algorithm is easy to implement and can apply on short and large patterns giving very fast results [14].

The Quick-search algorithm uses the Quick-search bad-character (qsBc) shift table, generated during the preprocessing stage. The shift value for a character in the qsBc table is defined as its corresponding position in the pattern from right to left order. If the character is not present in the pattern, then the shift value is equal to m+1. After an attempt, when the window is positioned on y[j.. j+m-1], the length of the shift is at least equal to one. Therefore, the character y[j+m] is necessarily involved in the next attempt and is used for the bad-character shift of the current attempt. During each attempt of the searching phase, the comparisons between the pattern and the text characters can be performed in any order.

Brute Force Algorithm (BF)

The most basic method of approaching the problem of pattern matching is the Brute Force (BF) algorithm. This technique is very simple and easy to follow. Let's assume we have text (input) T with length n and a pattern (keyword) P with size m. The algorithm begins by comparing the pattern to the text, scanning left to right, one character at a time, until there are no more matching characters. If a mismatch occurs, the algorithms shift the pattern one character to the right. The algorithm is given below,

Algorithm Brute Force (text, pattern)

```
{
n=length(text) // n is length of text
m=length(pattern) // m is length of pattern
for i=0 to (n-m)
{
j=0
while (j<m) and
(text(i+j)=pattern(j)) )
j++
if j=m
return i // match at i
}
return –l // no match
}
```

Karp-Rabin Algorithm (KR)

The Karp-Rabin Algorithm was created by Michael Rabin and Richard Karp. The main idea is that instead of using comparisons it involves mathematical computations which more specifically extends to the notion of hashing. The application of hashing

(converting each string into a numeric value) has always been a useful approach when it comes down to string matching because we can use it in order to test if two strings are the same. If both words have different hash values then we can be certain they are different [12]. But if their hash values are the same we cannot conclude they are the same string and will have to perform further comparisons (usually via Brute Force).

```
Algorithm Karp-Rabin(T,P,d,q)
n=length (T)
m=length (P)
h=d^(m-1) mod q
p=0
t_0=0
for i=1 to m //preprocessing
{
p=(d*p + P[i]) mod q //checksum of P
t_0=(d*t_0 + T[i]) mod q //checksum of T[1…m]
}
for s=0 to n-m //matching
{
if p=t_s
if P[1..m]=T[s+1..s+m]// Checksums match.
print “Pattern occurs with shift” s
if s < n-m
t_(s+1)=(d*(t_s-T[s+1]*h) + T[s+m+1]) mod q
}
```

Knuth-Morris-Prath Algorithm (KMP)

This algorithm was introduced by Don Knuth, Jim Morris, and Vaughan Pratt. It is quite similar to the Brute Force approach regarding scanning the text left to right, however we are now using information from the previously compared characters in order to determine the maximum possible shift of the pattern to the right. The idea is to avoid comparisons with elements from the text *T* that have previously been compared with some elements of the pattern *P*. In order to achieve this task, KMP preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself. The pre-calculation is done in time O(m) and is called the *next function F[j]* [16]. This function is an array that represents the size of the largest prefix of *P[0…j]* which is also a suffix of P[1…j]. The KMP algorithm states that the most we can shift the pattern in order to avoid redundant comparisons is namely the length of the *next function*.

```
Algorithm KMP (T, P)
F=nextFunction (P)
i=0
```

```
j=0
while i < n
{
if T[i]=P[j]
if j=m-1
return i–j //match
else
i++
j++
else
if j > 0
j=F[j-1]
else
i++
}
return -1 // no match
Void next Function (P)
F[0]=0
i=1
j=0
while i < m
{
if P[i] == P[j]
F[i]=j + 1
i++
j++
else if j > 0 then
j=F[j-1]
else
F[i]=0 //no match,
i++
}
```

Conclusion

Nowadays the network applications increased rapidly through Internet. Hence, there is a need to detect the malicious packets such as virus and worm in the network to support these applications. So the NIDS are deployed in the networks to detect these malicious activities.

This survey identifies a number of promising algorithms and provides an overview of recent developments in the single keyword string matching for NIDS. Algorithms like BM has two tables and matching starts with right to left, but in HP

and QS algorithms are used only one table and the matching is faster than the BM. In BF algorithm compared with the above three algorithms it is very simple and easy, the matching starts with left to right. Boyer-Moore algorithm is one of the efficient algorithms compared to the other algorithms available in the literature.

References

[1] Anagnostakis K G, Markatos E P, Antonatos S, Polychronakis M, "A domain-specific string matching algorithm for intrusion detection", Proceedings of the 18th IFIP International Information Security Conference (SEC2003). Athens, Greece: Kluwer Acadamic Pubhishers, 2003.

[2] Bi Kun., Gu. Nai-jie., Tu Kun., Liu. Xiao-hu. and Liu. Gang, "A Practical Distributed String Matching Algorithm Architecture and Implementation", Proceeding of world academy of science, engineering and technology, 2005.

[3] B. Watson, "The performance of single-keyword and multiple-keyword pattern matching algorithms", Eindhoven University of Technology. Department of Mathematics and Computing Science. 1994.

[4] Coit C J, Staniford S, McAlerney J, "Towards faster string matching for intrusion detection or exceeding the speed of Snort", Proceedings of the DARPA Information Survivability Conference and Exposition II (DISCEX'01). Los Alamitos, CA, USA: IEEE Comput. Soc., 2001

[5] D. M. Sunday, "A very fast substring search algorithm", Communications of the Association for Computing Machinery, 1990.

[6] Fisk M, Varghese G, "An analysis of fast string matching applied to content-based forwarding and intrusion detection", Technical Report CS2001-0670. San Diego: University of California, 2002.

[7] Gaston Gonnet and Ricardo Baeza-Yates, "An analysis of the Karp-Rabin string matching algorithm", Data Structuring Group. Department of Computer Science. University of Waterloo. May 1989.

[8] Nathan Tuck, Timothy Sherwood, Brad Calder, George Varghese, "Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection". Department of Computer Science and Engineering, University of California, San Diego. Department of Computer Science, University of California, Santa Barbara. 2004.

[9] Nilsen G, Torresen J, Sorasen O, "A variable word-width content addressable memory for fast string matching", Proceedings of the 22nd Norchip Conference. Osio, Norway: IEEE, 2004.

[10] Norton M, Roelker D,"The new Snort", Computer SecurityJournal, 2003.

[11] Oddiraju, Sriharsha. "Boyer Moore". Computer Science Department. Indiana State University. December 2011.

[12] Plaxton, Greg, "String Matching: Rabin-Karp Algorithm", Theory in Programming Practice. University of Austin, Texas. Fall 2005.

[13] P. S. Wheeler, "Techniques for improving the performance of signature based intrusion detection systems," Master's thesis, University of California Davis, 2006.
[14] Qingzhang CHEN, Yibo NIU, Zhehu WANG, Feng DU, "Improved BM Pattern Matching Algorithm for Intrusion Detection", 2010 Third International Joint Conference on Computational Science and Optimization. 2010.
[15] R.N. Horspool, "Practical fast searching in strings", Software-Practiceand Experience, Vol. 10, no. 6, 1980.
[16] Roesch M,"Snort: Lightweight intrusion detection for networks", Proceedings of the 13th System Administration Conference.
[17] R.S. Boyer and J.S.Moore,"A fast string searching algorithm", Communications of the Association for Computing Machinery (ACM),Vol. 20, no. 10, 1977.
[18] Wu S, Manber U, "A fast algorithm for multi-pattern searching",Technical Report TR-94-17, University of Arizona, 1994.
[19] Yuehui Chen, Abraham, Ajith and Crina Grosan, "Evolution of Intrusion Detection Systems", School of Computer Science and Engineering, Chung-Ang University, Korea. 2005.