Proceedings of the Ninth International Conference on Machine Learning and Cybernetics, Qingdao, 11-14 July 2010

PROTECTING SMART-GRID WITH BRENO-SILVA FAST SINGLE PATTERN MATCH ALGORITHM FOR SMALL PATTERNS

BRENO SILVA PINTO[1], CHUN CHE FUNG[2]

[1]Open Information Security Foundation (OISF), Brasília, DF, Brazil
[2]School of Information Technology, Murdoch University, Australia
E - MAIL : [1]breno.silva@gmail.com, [2]l.fung@murdoch.edu.au

Abstract:

Security of future Electrical power supply and the Smart-Grid will rely on the integrity of the communication network and in particular the Internet. Intrusion Prevention System (IPS) is crucial to guard against malicious and intentional attacks. Network packet search and string processing tasks in high-speed networks are critical for secure IPS performance. Slower algorithms could mean a portion of the packets could have been bypassed and slip through the protective border, which leads to security vulnerability. A novel pattern matching algorithm is proposed in this paper and the new algorithm has worst, best, and average time complexities of O(n + 2m), O(2m), and O(n + m) respectively - when applied for large alphabets - while searching a m bytes pattern on a text of n bytes. Experiments on 256-bytes alphabet |∑| text files have been conducted to compare the performance of the new algorithm with other alternatives. In all tests, the proposed algorithm has demonstrated an improved performance of at least 50% better on the mentioned alphabet size.

Keywords:

Pattern matching algorithms Intrusion Prevention systems, Network Security, Smart Grid

1. Introduction

Electrical Power systems are undergoing revolutionary changes with advanced communication and computer technology. In particular, Smart Grid has the potential to provide new ways to control and manage the power system for improved efficiency and financial returns. However, fulfillment of such potential hinges on a reliable and secured communication system, in particular, the Internet. The integrity of the Internet is in turn depending on the efficiency and performance of intrusion detection systems to guard against malicious and intentional attacks that are normally identified in the forms of signatures or patterns.

In the last few years, many algorithms have been released to solve general problems in the pattern match area [1]. Finding the exact or approximate match of a given pattern within a string is the one of the major problems in computer science. Such studies and algorithms are applied to different disciplines for diverse purposes. Information Security is one of those areas that make use of this research. With the increasing network speed, better algorithms are required for faster traffic analysis. An example is the Intrusion Prevention Systems [2] which has its core engine based in pattern matching.

Network intrusion prevention systems provide proactive defense against security threats by detecting and blocking attack-related traffic. This task can be highly complex, and therefore, software-based network intrusion prevention systems have difficulty in handling high speed links, resulting in impact of network performance or unanalyzed data, because a large number of packets must be analyzed in terms of malicious content before being forwarded. In the case of the future Smart-Grid, it is expected that the traffic will grow due to the ongoing data exchange between the many entities involved within the power system.

Economic damages or losses due to attacks are very costly. A recent worm called Conficker caused a total economic cost (including the cost of efforts to combat the worm, the cost of purchasing counter-measure software) of $9.1 billion dollars [3]. Some of threats like Conficker can invade an internal network company if there is unanalyzed data bypassing the security solution. The indirect costs due to power system breakdown are expected to be magnified many times over.

A recent study released the SSEF [4] and BLIM [5] algorithms which seems to be good options for long pattern matching operations, but they do not work well with small patterns. From record, most of the known intrusion prevention systems usually inspect the network packets for patterns with 3 >= m <= 8 bytes, which are known as performance killers.

This research focuses on exact and fast matching of small patterns on random sequences of characters using a signature and filtering methodology. Instead of checking the occurrence of the pattern on all over the text, the method first detects slices of the text, on which the presence of the pattern is probable. After that, it starts a full verification on

978-1-4244-6527-9/10/$26.00 ©2010 IEEE

those text slices reported by the filtering phase using a word signature function. So, the first part of the algorithm is used to detect possible positions on the text which a match is possible without a deep investigation, and the verification phase will check the existence of the pattern on those positions. Section 2 of this paper provides the basics of the problem and then Section 3 provides a description of the proposed Breno-Silva algorithm. An analysis is provided in Section 4 followed by some experimental results in section 5. The paper is then concluded in Section 6.

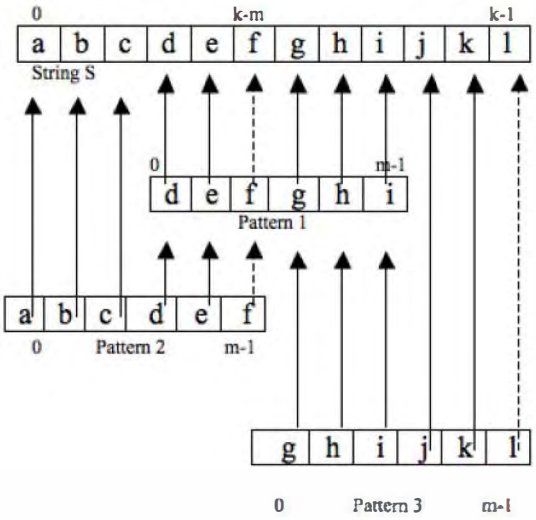
2. Preliminaries and Basics

For a given string S of k characters, they can be shown as:

$$S = S_0 S_1 S_2 \cdots S_{k-1} \quad (1)$$

Where k = m·2 and m the length of pattern P = $p_0 p_1 \cdots p_{m-1}$ to be searched. (2)

Assuming each character of a given pattern is not repeated and the pattern is present in the string S, the presence of P = $p_0 p_1 \cdots p_{m-1}$ can be easily found and its position determined looking at k-1 and k–m:



If the system is looking for *Pattern 2,* the k-1 position of string S will not give us any information, however at k-m will suggest the presence of the given pattern. Assuming at the k-m position, the algorithm can determine if letter "f" will represent the end of given pattern (*Pattern 2*) or the middle as in *Pattern 1*.

To make this decision possible an unique index number is determined during the preprocess stage:

0	Pattern 1				m-1
d	e	f	g	h	i
1	2	3	4	5	6

Index 1

0	Pattern 2				m-1
a	b	c	d	e	f
1	2	3	4	5	6

Index 2

Assuming the above example of index implementation the algorithm knows that letter "f" in *Pattern 1* is in the middle of pattern and it can be found at:

$$S = S_{k-m-2} \ldots S_{k-m+3} \quad (3)$$

On other side for *Pattern 2* the algorithm can find the pattern in S:

$$S = S_{k-m} \ldots S_{k-m-5} \quad (4)$$

However when a pattern with repeated characters is used the algorithm can identify the probable presence of a given pattern in the text portion, but cannot determine the character position. In this case the indexes will form a sequence of S that will be checked if all the members are true in the text portion.

3. The Breno-Silva algorithm

The proposed algorithm is described below as an exemplification of the mentioned filtering and signature methodology. The following sections will explain how the proposed algorithm selects a portion of the text and how it will check for the presence of a given pattern.

3.1 The Signature function

An important part of the algorithm is the signature function, it will be used to check if a pattern exists or not in a portion of text. As an elementary part of this function, an AlphaTable with a 256 16 bits numbers is defined using a polynomial number.

htab[256] <- {16bitsnum1, 16bitsnum2, ... 16bitsnum256 }

To create this AlphaTable a CRC [6] polynomial based function can be used. The *htab* will be accessed by a

Calsig() function to create the pattern signature, in other words, an unique number that will represent a pattern.

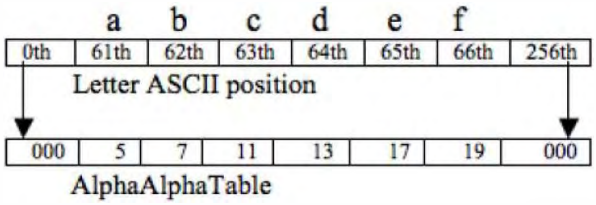
calcsig(x,y) (htab[((y SHIFT_RIGHT 8) AND 0xff)] XOR (y SHIFT_LEFT 8) XOR x)

It is important to note that this phase of Breno-Silva algorithm will be called once for any pattern and any text.

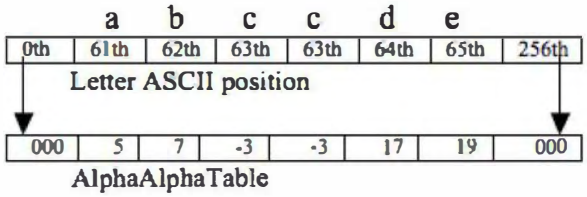
3.2 Preprocessing

The first phase is a preprocessing stage that will help the main search function to select a portion of the text to be checked for a pattern. Also this function will define an index number for each pattern letter in the *AlphaTable*, which will also help to determine the position of a pattern letter in the text.

Below is illustrated the AlphaTable for a pattern (abcdef) with no repeated letters using the *Pindex[5,7,11,13,17,19]*.



Also the example with repeated letter pattern (abccde) with *Rindex[-3,-2,-1,1,2,3]*.



The *Pindex* and *Rindex* arrays must have the same pattern's size and different number in each array position.

3.3 Main algorithm

The pseudo code of *Search* function is called after the *Preprocess* to inspect each $P \cdot 2$ and $P \cdot 2 - P$ portion of the text, assuming P is the pattern length.

Using the pattern index number *Pindex[]* array the algorithm knows the exact position of a pattern letter if it is not composed by repeated characters. So a specific K_p pattern length sequence combination around the index number will have its signature calculated by *Calcsig* function to determine if the pattern is present or not by using all members of K sequence and accessing data from AlphaTable[K_i]

Below is described a K sequence for 6-bytes pattern:

If pattern index number represent the first position
$K = 0\ \{ i, i+1, i+2, i+3, i+4, i+5 \}$
If pattern index number represent the second position
$K = 1\ \{ i-1, i, i+1, i+2, i+3, i+4 \}$
If pattern index number represent the third position
$K = 2\ \{ i-2, i-1, i, i+1, i+2, i+3 \}$
If pattern index number represent the 4th position
$K = 3\ \{ i-3, i-2, I-1, I, I+1, I+2 \}$
If pattern index number represent the 5th position
$K = 4\ \{ i-4, i-3, I-2, I-1, i, I+1 \}$
If pattern index number represent the 6th position
$K = 5\ \{ i-5, i-4, i-3, i-2, i-1, i \}$

The algorithm assumes a different flow when the pattern has repeated letters. It will directly test if each K sequence is true, if so it will compute the signature to check if the pattern exists in the inspected portion of text.

4. Complexity Analysis

The preprocessing stage will setup the index numbers for a pattern length m. Thus, the complexity of preprocessing is $O(m)$.

For the best case the algorithm will detect a specific index number on *AlphaTable[]* and execute m times the *calcsig* function, which is the most expensive operation, so the algorithm is $\Theta(2m)$.

In the worst case for small alphabets the algorithm will stop in each index number of $P \cdot 2$ and $P \cdot 2 - P$ and will check all K sequences in case of repeated letters. In both positions *calcsig* function will be called near to $n \cdot m$ times, so we have $O(n \cdot m + m)$. However the worst case for large alphabets is near to $O(n + 2m)$:

$J = n / P \cdot 2$, where J is the number of K sequences (5)

$T = m \cdot 2 \cdot J + m$, where m is the pattern length and *calcsig* execution time. (6)

$O(T) = O(n + m)$ (7)

Assuming pre-processing complexity:

$O(T + m) = O(n + 2m)$ (8)

Finally in the average case the search function will execute *calcsig* for n times, assuming $|\sum| = 256$ For each K sequence the *calcsig* will be executed m times at $P \cdot 2$ *and* $P \cdot 2 - P$:

$J = n / P \cdot 2$, where J is the number of K sequences (9)

$T = m \cdot 2 \cdot J$ (10)

where m is the pattern length and *calcsig* execution time.

$O(T) = O(n)$ (11)

Assuming preprocessing complexity:

$O(T + m) = O(n + m)$ (12)

The reason for that is because the probability to have a K sequence true for pattern in a large alphabet is too small. On the other side for small alphabets ($|\sum| <= 4$) the average case is near to worst case, because the probability of all K sequences to be true is high.

5. Implementation and Experimental Results

The algorithm was implemented in C language and executed in a 32 bits computer Intel(R) Core(TM)2 Duo T7300 @ 2.00GHZ. In order to determine the algorithm's elapsed time, the GetTickCount() was used, which gave the time in milliseconds. Patterns with 4, 6 and 8 bytes were chosen for the tests and a large number of repetitions for each pattern (100, 1000, 1500, 5000, 10000, 15000, 20000, 50000, 100000, 500000, 1000000 and 5000000) were used to measure the performance during the execution, which is a real scenario for security network inspection operations. For packet simulation, the size selected was a 1408 bytes text. Four algorithms were compared and representative results are tabulated or shown in diagrams below.

- Breno-Silva (BS) Algorithm
- Boyer-Moore Algorithm (BM) [7]
- Knuth-Morris-Pratt Algorithm (KMP) [8]
- Shift-And Algorithm (SA) [9]

Existent pattern of 4-bytes length – time in miliseconds				
Repetitions	BS time	BM time	KMP time	SA time
100	0	63	156	47
500	92	156	408	313
1000	155	345	797	545
1500	156	376	1171	859
5000	717	1151	3288	2436
10000	1570	2863	7864	4678
15000	2082	4035	10758	7935
20000	3081	5643	15744	9558
50000	7783	14325	38830	25922
100000	13205	24622	67658	48840
500000	74608	136465	367389	235024
1000000	141618	270450	719088	459998
5000000	680765	1332767	3559202	2626901

Table 1. Tests using 100 4-bytes random existent patterns

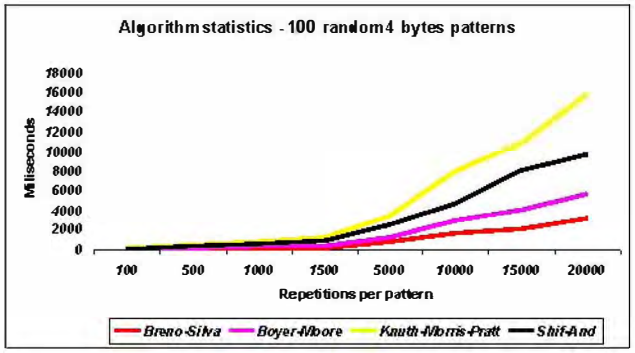


Figure 1. Algorithm statistics using 100 random 4–bytes patterns (100-20000 repetitions)

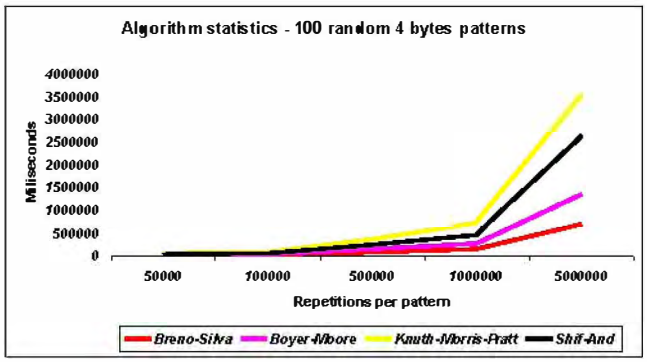


Figure 2. Algorithm statistics using 100 random 4–bytes patterns (50000-5000000 repetitions)

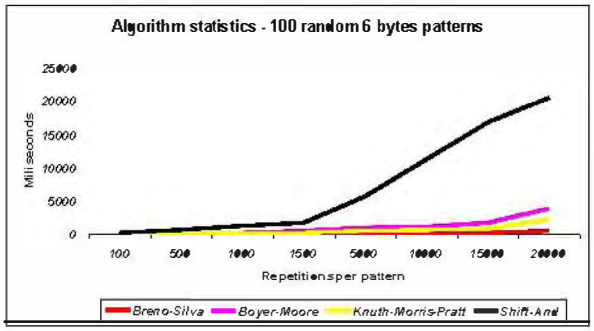


Figure 3. Algorithm statistics using 100 random 6–bytes patterns (100-20000 repetitions)

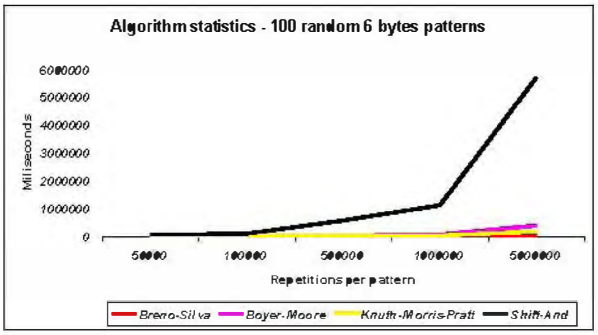


Figure 4. Algorithm statistics using 100 random 6–bytes patterns (50000-5000000 repetitions)

Existent pattern of 6-bytes length – time in miliseconds				
Repetitions	BS time	BM time	KMP time	SA time
100	0	16	15	188
500	0	93	48	625
1000	16	172	31	1188
1500	15	501	172	1794
5000	78	843	407	5534
10000	139	1115	596	11140
15000	205	1624	828	16892
20000	487	3827	2046	20408
50000	960	8145	3489	57521
100000	1723	16122	7452	115293
500000	4347	40017	10973	577592
1000000	8516	79500	38156	1137611
5000000	42420	398609	190627	5690767

Table 2. Tests using 100 6-bytes random existent patterns

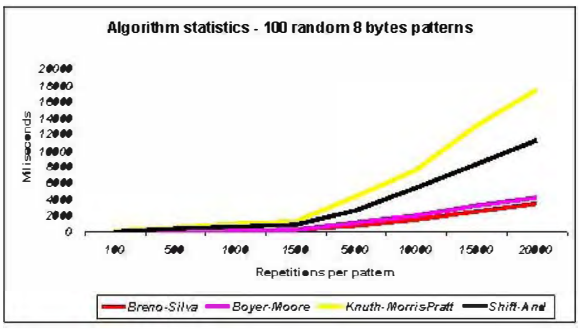


Figure 5. Algorithm statistics using 100 random 8–bytes patterns (100-20000 repetitions)

Existent pattern of 8-bytes length – time in miliseconds				
Repetitions	BS time	BM time	KMP time	SA time
100	16	61	110	47
500	78	124	486	374
1000	125	170	971	622
1500	215	299	1220	873
5000	733	1107	4300	2590
10000	1401	1891	7520	5332
15000	2421	3171	12986	8215
20000	3439	4138	17361	11155
50000	7589	9882	40217	29606
100000	15359	20857	87472	54305
500000	86405	108379	464560	288472
1000000	163604	207648	861014	556332
5000000	856734	1084331	4501794	2810794

Table 3. Tests using 100 8-bytes random existent patterns

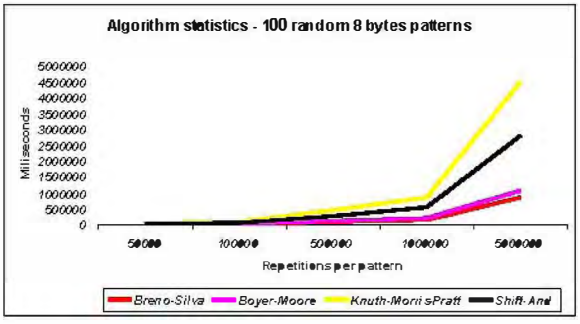


Figure 6. Algorithm statistics using 100 random 8–bytes patterns (50000-5000000 repetitions)

In all cases, the Breno-Silva algorithm has demonstrated better performance, in both large and small numbers of search operations. In other words, while the other reported algorithms are capable of inspecting only a portion of traffic in high-speed networks, the proposed Breno-Silva algorithm is able to inspect a larger traffic

volume without severely affecting its performance. It can be inferred that the proposed algorithm, when incorporated in an Intrusion Prevention System, a large number of packets can be inspected in a lesser time. This is particularly important for Smart-Grid operations in ensuring the integrity and safety of the network.

6. Conclusion and future work

The proposed algorithm as demonstrated in this study is a good option for string search operations with small patterns. The use of this algorithm can represent a performance improvement for security network inspect products like Intrusion Prevention Systems during the packet processing stage, helping the industry to minimize the losses caused by attacks. In particular, it will be suitable for the protection of essential utilities such as power systems and the Smart Grid. According to security specialists, billions of dollars are lost every year repairing attacked systems [10]. The major losses are caused by known threats, which in many situations can be detected and blocked by an Intrusion Prevention System. However this security solution must be prepared to inspect all the network traffic and definitely a better pattern match algorithm will help to achieve this goal, as the costs will be much higher due to breakdown or interruption of the power system.

As a future complement of this study, longer patterns will the used to compare this algorithm with others, as well, tests with small alphabets texts. Also a multi-pattern-match implementation of this algorithm in examining typical data and traffic pattern on electrical power systems will be carried out.

References

[1] C. Charras and T. Lecroq: Handbook of exact string matching algorithms, King's Collage Publications, 2004.

[2] "Guide to Intrusion Detection and Prevention Systems", http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf

[3] "Cyber Secure Institute on the Conficker Controversy", http://cybersecureinstitute.org/blog/?p=15

[4] M. O˘guzhan K¨ulekci: Filter Based Fast Matching of Long Patterns by Using SIMD Instructions", Proceedings of the Prague Stringology Conference 2009.

[5] M. O. K¨ulekci: A method to overcome computer word size limitation in bit-parallel pattern matching, in Proceedings of 19th International Symposium on Algorithms and Computation, ISAAC'2008, S.-H. Hong, H. Nagamochi, and T. Fukunaga, eds., vol. 5369 of Lecture Notes in Computer Science, Gold Coast, Australia, December 2008, Springer Verlag, pp. 496–506.

[6] "Cyclic redundancy check", http://en.wikipedia.org/wiki/Cyclic_redundancy_check#Designing_CRC_polynomials

[7] R. Boyer and J. Moore: A fast string searching algorithm. Communications of the ACM, 20(10) 1977, pp. 762–772.

[8] Knuth, D. E., Morris, J. H., & Pratt, V. R. (1977). Fast pattern matching in strings. SIAM Journal on Computing, 6(2), 323-350.

[9] U. Manber. A text compression scheme that allows fast searching directly in the compressed _le. In Proc. 5th Annu. Symp. Combinatorial Pattern Matching, volume 807 of Lecture Notes in Computer Science, pages 113{124. Springer-Verlag,1994.

[10] "State of internet 2009: A Report on the Ever-Changing Threat Landscape", http://ca.com/files/SecurityAdvisorNews/2009threatreportfinalfinal_224176.pdf