International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 6 4177 – 4180

Performance Study of the Running Times of well known Pattern Matching Algorithms for Signature-based Intrusion Detection Systems

S. Manohar Naik, Research Scholar,
Department of Computer Science & Technology,
Sri Krishnadevaraya University,
Anantapur - 515003, India.
manoharamen@gmail.com

Dr. N. Geethanjali, Associate Professor,
Department of Computer Science & Technology,
Sri Krishnadevaraya University,
Anantapur - 515003, India.
geethanjali.sku@gmail.com

Abstract— Intrusion detection system (IDS) is the basic component of any network defense scheme. Signature based intrusion detection techniques are widely used in networks for fast response to detect threats. One of the main challenges faced by signature-based IDS is that every signature requires an entry in the database, and so a complete database might contain hundreds or even thousands of entries. Each packet is to be compared with all the entries in the database. This can be highly resource-consuming and doing so will slow down the throughput and making the IDS vulnerable. Since pattern matching computations dominate in the overall performance of a Signature-based IDS, efficient pattern matching algorithms should be used which use minimal computer storage and which minimize the searching response time. In this paper we present a performance study of the running times of different well known pattern matching algorithms using multiple sliding windows approach.

Keywords- *Network security, Intrusion detection, Pattern matching, Network performance*

*****

I. INTRODUCTION

In the past, we have seen computer worms spread themselves without any human interaction and launched the most destructive attacks against computer networks [2]. As an example, in January 2003, the SQL Slammer worm, also known as sapphire, was released into the Internet exploiting a weakness into Microsoft SQL servers. In only 10 minutes the worm spread worldwide consuming massive amount of bandwidth and bringing down 5 of the 13 root DNS servers [3]. Amongst worm defensive mechanisms, IDS are the most widely deployed techniques that utilize the self-duplicating repetitive and recurring nature of computer worms to detect the patterns and signatures of theses malicious codes in the network traffic. Consequently IDSs have become an integral part of the security infrastructure of organizations. These systems based on the parameters used for detection, are broadly divided to signature-based and anomaly-based IDSs [4].

Signature-based detection is normally used for detecting known attacks. There are different definitions of attack signatures. No knowledge of normal traffic is required but a signature database is needed for this type of detection systems.

Signature based IDS systems require that their data bases need to be updated regularly at different time intervals so as to detect the imminent threats generated on the network. This process is a quite time consuming and requires a quick underlying system to update the database. If the signature database is not updated timely, then new threats will not be detected using this model. Therefore one of the main challenges of Signature-based IDS is to control the huge traffic volume where each packet needs to be compared to with the known signatures database and reduce the comparison time of signatures in it.

The rest of the paper is organized as follows. Section II describes the contributed characterization of pattern matching in network security such as signature-based NIDS; Section III describes survey on the existing pattern matching algorithms; Section IV covers the performance analysis of some pattern matching algorithms using multiple sliding windows approach. Finally, the conclusion and the future work are presented in Section V.

II. CHARACTERIZATION OF PATTERN MATCHING IN NIDS

Signature matching is one of the most computationally intensive tasks of IDS. Performance of IDS suffers as signatures or rules grow in data volume. In order to increase the performance of NIDS, one of approach is to improve the performance of signatures detection engine by increasing the efficiency of pattern matching algorithm.

Pattern matching is a pivotal theme in computer research because of its relevance to various applications such as web search engines, computational biology, virus scan software, network security and text processing [5].

Pattern matching focuses on finding the occurrences of a particular pattern P of length ‘m’ in a text ‘T’ of length ‘n’. Both the pattern and the text are built over a finite alphabet set called Σ of size σ.

Generally, pattern matching algorithms make use of a single window whose size is equal to the pattern length. The searching process starts by aligning the pattern to the left end of the text and then the corresponding characters from the pattern and the text are compared. Character comparisons continue until a whole match is found or a mismatch occurs, in either case the window is shifted to the right in a certain distance [6]. The shift value, the direction of the sliding window and the order in which comparisons are made varies in different pattern matching algorithms.

To reduce the number of comparisons, the matching process is usually divided into two phases. The pre-processing phase and the searching phase. The pre-processing phase determines the distance (shift value) that the pattern window will move. The searching phase uses this shift value while searching for the pattern in the text with as minimum character comparisons as possible.

A good pattern matching algorithm aims to decrease the searching phase during each attempt and to increase the shifting value of the pattern. Hence several pattern matching algorithms have been developed with a view to enhance the

ISSN: 2321-8169
4177 – 4180

searching processes by minimizing the number of comparisons performed [7-9].

III. Survey on the Existing Algorithms

This section includes survey of the main pattern matching algorithms. Table I summarize and compare these algorithms. String matching algorithms can be classified into seven categories according to the preprocessing function in the algorithm [10].

The first category, e.g. the Brute Force algorithm (BF) [11], shifts the pattern only one position at each attempt. The second category, which includes the Boyer-Moore algorithm (BM) [12]-[14] and the Fast Search algorithm (FS) [15], uses two preprocessing functions. The third category, a good example of which is the Boyer Moore Horspool algorithm (BMH) [16]-[18], uses one preprocessing function based on the rightmost character in the current window. The fourth category, e.g. the Quick Search algorithm (QS) [19], uses one preprocessing function based on the character next to the current window. The fifth category, such as the Berry–Ravindran algorithm (BR) [20], uses one preprocessing function based on the next two characters to the current window. The sixth category, e.g. the Karp-Rabin algorithm (KR) [21] and the Zhu Takaoka algorithm (ZT) [22], uses a preprocessing hashing function. The final category uses hybrid algorithms and includes the SSABS [23], TVSBS [24], ZTMBH [25], BRFS [26], BRBMH [27], BRQS [10], TSW [28], OE [29], RS-A [30], ERS-A [31] and the EERS-A [32] algorithms.

IV. Performance Analysis

In this section, we evaluate the performance of four algorithms, i.e., Fast-Search, TVSBS, SBNDM and FSBNDM algorithms. Table II, Table III, Table IV and Table V shows experimental results obtained by comparing multiple sliding windows variants of the above algorithms. Running times are expressed in mille-seconds and best values are underlined and boldfaced.

TABLE II. Running time of multiple sliding windows variants of the Fast-Search Algorithm

m	2	4	8	16	32	64
σ = 16						
FS-W(1)	16.20	10.93	8.27	7.19	6.95	6.63
FS-W(2)	**13.56**	**9.43**	**7.32**	**6.57**	**6.27**	**6.16**
FS-W(4)	15.56	10.54	8.02	7.0	6.66	6.57
FS-W(6)	15.84	10.67	8.17	7.2	6.76	6.67
FS-W(8)	15.96	10.78	8.69	7.32	6.93	6.78
σ = 32						
FS-W(1)	14.81	10.14	7.46	6.42	6.06	6.10
FS-W(2)	**12.30**	**8.15**	**6.73**	**6.02**	**5.85**	**6.01**
FS-W(4)	13.35	9.15	7.17	6.38	6.33	6.21
FS-W(6)	12.96	9.12	7.12	6.41	6.22	6.33
FS-W(8)	14.10	10.13	7.59	6.73	6.5	6.44
Σ = 128						
FS-W(1)	16.52	9.46	6.91	6.01	6.01	6.15
FS-W(2)	**9.85**	**7.17**	**6.1**	**5.8**	**5.77**	**5.97**
FS-W(4)	11.06	8.11	6.46	6.06	6.37	6.09
FS-W(6)	10.91	7.9	6.45	6.13	6.54	6.07
FS-W(8)	11.17	8.03	6.74	6.21	6.18	6.12

Table II and Table III shows that the best results are obtained for k = 2, and k = 6 respectively. Table IV and Table V shows that the best results are obtained for k = 4.

TABLE III. Running time of multiple sliding windows variants of the TVSBS Algorithm

m	2	4	8	16	32	64
σ = 16						
TVSBS-(1)	15.71	14.48	10.23	8.23	7.58	7.24
TVSBS-(2)	17.9	12.49	10.3	8.33	7.48	7.2
TVSBS-(4)	**15.61**	**12.3**	10.92	8.61	7.62	7.68
TVSBS-(6)	16.96	13.1	**9.71**	**7.78**	**6.86**	**6.6**
TVSBS-(8)	20.12	14.38	10.54	8.56	7.83	7.94
σ = 32						
TVSBS-(1)	18.92	17.7	13.34	11.59	10.16	9.82
TVSBS-(2)	19.26	**15.88**	13.85	11.97	11.5	10.87
TVSBS-(4)	20.39	16.31	13.62	12.03	11.35	10.93
TVSBS-(6)	**18.48**	16.14	**13.0**	**11.21**	**10.12**	**9.63**
TVSBS-(8)	19.34	16.78	13.74	11.87	11.83	10.76
σ = 128						
TVSBS-(1)	20.82	16	11.53	9.39	**7.74**	**7.24**
TVSBS-(2)	18.7	14.69	11.82	9.7	8.8	8.05
TVSBS-(4)	16.78	14.07	10.98	9.32	8.57	7.98
TVSBS-(6)	**15.12**	**12.62**	**10.59**	**9.07**	8.34	8.02
TVSBS-(8)	16.55	13.47	12.15	9.49	8.48	7.96

TABLE IV. Running time of multiple sliding windows variants of the SBNDM Algorithm

m	2	4	8	16	32	64
σ = 16						
SBNDM–W(1)	17.92	12.90	**8.40**	**6.70**	**6.05**	**6.10**
SBNDM–W(2)	13.25	11.40	9.55	6.98	9.08	9.02
SBNDM-W(4)	**12.29**	**10.56**	9.96	7.89	8.32	14.33
SBNDM–W(6)	16.33	11.24	11.52	8.83	12.50	13.28
σ = 32						
SBNDM–W(1)	16.85	12.59	8.30	**8.09**	**5.90**	**6.98**
SBNDM–W(2)	14.06	8.77	8.19	8.14	7.67	9.41
SBNDM–W(4)	**11.26**	**7.18**	**8.09**	8.16	9.48	10.35
SBNDM–W(6)	11.34	8.92	10.60	8.71	14.01	10.56
σ = 128						
SBNDM–W(1)	12.94	13.31	8.82	6.28	7.86	6.39
SBNDM–W(2)	11.30	8.20	6.85	6.04	5.85	6.79
SBNDM–W(4)	**10.34**	**7.59**	**7.24**	**5.77**	**5.60**	6.39
SBNDM–W(6)	10.78	8.95	7.61	6.17	6.61	**6.35**

TABLE V. Running time of multiple sliding windows variants of the FSBNDM Algorithm

m	2	4	8	16	32	64
σ = 16						
FSBNDM–W(1)	12.32	9.73	7.73	6.73	6.13	6.14
FSBNDM–W(2)	**11.46**	**8.37**	7.78	6.74	6.15	6.54
FSBNDM–W(4)	11.48	10.37	**7.52**	**6.57**	**6.12**	**6.10**
FSBNDM–W(6)	13.23	9.75	8.46	7.12	6.28	6.36
σ = 32						
FSBNDM–W(1)	12.32	10.02	8.24	6.87	5.97	6.06
FSBNDM–W(2)	11.63	10.20	8.24	5.85	6.14	6.12
FSBNDM–W(4)	**10.26**	**9.94**	**8.14**	**5.83**	**5.81**	**5.70**
FSBNDM–W(6)	10.32	10.89	8.79	6.16	5.91	6.13
σ = 128						
FSBNDM–W(1)	9.93	7.52	6.33	6.09	5.76	6.30
FSBNDM–W(2)	8.32	8.06	6.29	5.69	5.86	6.05
FSBNDM–W(4)	**8.21**	**7.43**	**6.20**	**5.35**	**5.69**	**5.86**
FSBNDM–W(6)	8.56	7.58	6.22	5.89	5.98	5.97

TABLE I. SUMMRY AND MAIN CHARACTERSTICS OF PATTERN MATCHING ALGORITHMS IN LITERATURE

Algorithm Name	Year	Comparison Order	Preprocessing Time	Searching Time	Main Characteristics
Brute Force Algorithm	Very Old	From left to right	N/A	O(mn)	Shifts the pattern only a position each attempt. Not an optimal algorithm since it does not use the information that could be gained from the last comparison made
Knuth-Morris-Pratt Algorithm	1974	From left to Right	O(m)	O(mn)	Uses the notion of the border of the string. It increases performance, decreases delay, and decreases searching time compared to the Brute Force algorithm. It is efficient for large alphabets
Boyer-Moore Algorithm	1977	From right to left	O(m+σ)	O(mn)	Uses two preprocessing functions; the good-suffix shift and the bad-character shift. It is not very efficient for small alphabets
Horspool Algorithm	1980	From left to Right	O(m+σ)	O(mn)	Uses the Horspool bad-character preprocessing function based on the rightmost Character in the current window. It is a simplification of the Boyer-Moore algorithm. It is faster than, and easier to implement than the Boyer-Moore algorithm
Karp-Rabin Algorithm	1984	From left to Right	O(m)	O(mn)	Uses the Karp-Rabin preprocessing hashing function. It is very effective for multiple pattern matching in one-dimensional string matching
Zhu-Takaoka Algorithm	1989	From right to left	$O(m+\sigma^2)$	O(mn)	Uses the Zhu-Takaoka preprocessing hashing function. It is very effective for multiple pattern matching in two-dimensional string matching
Quick-Search Algorithm	1990	From right to left	O(m+σ)	O(mn)	Uses the Quick-Search bad-character preprocessing function based on the next character to the current window. It is especially fast for short patterns
Berry–Ravindran Algorithm	1999	From left to Right	$O(m+\sigma^2)$	O(mn)	Uses the Berry-Ravindran preprocessing function based on the next two characters after the current window in order to increase the shifting value of the pattern
Fast Search Algorithm	2003	From right to left	$O(m+\sigma^2)$	O(mn)	Uses the bad-character function only if the mismatching character is the last character of the pattern, otherwise the good-suffix function is to be used. It is efficient in very short patterns
SSABS Algorithm	2004	From right to left	O(m+σ)	O(m(n-+1))	Uses the Quick-Search bad-character preprocessing function. It scans the pattern with the text starting from the right most then it scans the next last character and goes backward to the left
TVSBS Algorithm	2006	From right to left	$O(m+\sigma^2)$	O(m(n – m + 1))	A combination of the Berry-Ravindran and the SSBAS algorithms. It scans the pattern with the text using the searching phase of the SSABS algorithm. Uses the Berry-Ravindran preprocessing function
ZTMBH Algorithm	2008	From left to Right	$O(m+\sigma^2)$	O(mn)	A combination of the Zhu Takaoka and the Boyer-Moore Horspool algorithms. It scans the pattern using the searching phase of the BMH algorithm. Uses the Zhu-Takaoka preprocessing hashing function.
BRFS Algorithm	2008	From right to left	$O(m+\sigma^2)$	O(mn)	A combination of the Berry-Ravindran and the Fast Search algorithms. It scans the pattern using the searching phase of the FS algorithm. Uses the Berry-Ravindran Preprocessing Function.
BRBMH Algorithm	2008	From left to Right	O(m+σ)	O(mn)	Enhances the preprocessing Berry-Ravindran algorithm and combines it with the BMH algorithm. It scans the pattern using the searching phase of the BMH algorithm. Uses the enhanced Berry-Ravindran Preprocessing Function.
BRQS Algorithm	2008	From right to left	O(m+σ)	O(mn)	Uses the enhanced Berry-Ravindran Preprocessing Function and combines it with the QS algorithm. It scans the pattern using the searching phase of the QS algorithm
TSW Algorithm	2008	Two parallel windows from left to right and from right to left	$O(m+\sigma^2)$	O(mn)	TSW uses two sliding windows rather than using one sliding window to scan all text characters as in Berry-Ravindran algorithm. Uses two one dimensional arrays to store shift values rather two dimensional array in original BR algorithm.
OE Algorithm	2009	From right to left	O(m+σ)	O(mn)	Combines an enhanced Preprocessing phase from Berry-Ravindran algorithm with the proposed new searching phase procedure
RS-A Algorithm	2012	From right to left	$O(m+\sigma^2)$	O(mn)	After each attempt, the window is shifted to the left using the shift value computed for the four consecutive characters immediately.
ERS-A Algorithm	2013	Two parallel windows from left to right and from right to left	O(2(m-3))	O(mn)	Uses two sliding windows rather than one to scan all the text characters. Two one dimensional arrays are used to store the shift values.
EERS-A Algorithm	2015	Two parallel windows from left to right and from right to left	O(2(m-3))	O(mn)	Used two sliding window to scan the text from both sides at the same time; also comparisons between the pattern and the text happened from both sides of the pattern.

V. CONCLUSIONS

In this research we extensively studied and analyzed the various pattern matching algorithms and their running time performances for NIDS. We conclude from our general performance studies on pattern matching algorithms using multiple sliding windows approach and this approach turns out to be simple to implement and leads to very fast algorithms in practical cases, especially in the case of large alphabets and natural language texts as shown in the experimental results.

REFERENCES

[1] R.Bace and P.Mell. "Intrusion Detection Systems," National Institute of Standards and Technology (NIST), Special publication 800-31, 2001.

[2] S. Chen and Y. Tang, "Slowing down internet worms," in Proceedings of 24th International Conference on Distributed Computing Systems, pp. 312-319, 2004.

[3] M. Uddin, K. Khowaja, and A.A. Rehman, "Dynamic multi-layer signature based intrusion detection system using mobile agents," International Journal of Network Security & Its Applications (IJNSA), vol. 2, no. 4, pp. 129-141, 2010.

[4] G. Coretez, Fun with Packets: Designing a Stick, Endeavor Systems, Inc, 2002.

[5] Wang, Y. and H. Kobayashi, "High performance pattern matching algorithm for network security," IJCSNS, 6: 83-87, 2006.

[6] Charras, C. and T. Lecroq, Handbook of Exact String Matching Algorithms. First Edition. King's College London Publications. ISBN: 0954300645, 2004.

[7] Hume, A. and D. Sunday, Fast string searching. Software Practice Experience, 21: 1221-1248, 1991. doi: 10.1002/spe.4380211105.

[8] Lecroq, T., Experimental results on string matching algorithms. Software-practice and Experience, 25: 727-765, 1995. doi: 10.1002/spe.4380250703.

[9] Davies G., and Bowsher S., Algorithms for pattern matching, Software-Practice and Experience,16:575-601, 1996. doi:10.1002/spe.4380160608

[10] A. F. Klaib and H. Osborne, "BRQS Matching Algorithm for searching Protein Sequence Databases," unpublished.

[11] J. Mettetal. (2004, September 16). Brute Force Algorithms: Motif Finding. Available: http://ocw.mit.edu/NR/rdonlyres/Mathematics/18-417Fall-2004/8BA92AB3-A9CD-4719-A4AC-1AAFDB8AE5A0/0/lecture_03.pdf

[12] G. Plaxton, (Fall 2005). String Matching: Boyer-Moore Algorithm. Available: http://www.cs.utexas.edu/users/plaxton/c/337/05f/slides/StringMatching-4.pdf

[13] O. Danvy and H. K. Rohde, "Obtaining the Boyer-Moore String-Matching Algorithm by Partial Evaluation," Information Processing Letters, vol. 99, pp. 158–162, 2006.

[14] R. S. Boyer, J. S. Moore, "A fast string searching algorithm," Communications of ACM, vol. 20, no. 10, pp.762–772, 1977.

[15] Cantone, S. Faro, "Fast-search: A new efficient variant of the Boyer–Moore string matching algorithm", Lecture Notes in Computer Science, vol. 2647, pp. 47–58, 2003.

[16] M. Régnier and W. Szpankowski, "Complexity of Sequential Pattern Matching Algorithms" Lecture Notes in Computer Science, vol. 1518, pp. 187-199, 2004.

[17] T. RAITA, "Tuning the Boyer–Moore–Horspool String Searching Algorithm," Software-Practice and Experience, vol. 22, pp. 879-884, 1992.

[18] R. N. Horspool, "Practical fast searching in strings," Software-Practice and Experience, vol. 10, no. 6, pp. 501-506, 1980.

[19] Sunday, "A very fast substring search algorithm," CommonACM, no. 33, pp. 132–142, 1990.

[20] Berry and Ravindran, "Fast string matching algorithm and experimental results,", Proceedings of the Prague Stringology Club, pp. 16–26, 2001.

[21] C. Charras, T. Lecroq, Handbook of exact string matching Algorithms. Available: http://www-igm.univ-lv.fr/~lecroq/string/.

[22] R.F. Zhu, T. Takaoka, "On improving the average case of the Boyer-Moore string matching algorithm," Journal of Information Processing, vol. 10, no. 3, pp. 173–177, 1987.

[23] S. S. Sheik, S. K. Aggarwal, A. Poddar, N. Balakrishnan and K. Sekar, "A fast pattern matching algorithm," Journal of Chemical Information and Computer Sciences, no. 44, pp. 1251–1256, 2004.

[24] R. Thathoo, A. Virmani, S. S. Lakshmi, N. Balakrishnan and K. Sekar, "TVSBS: A fast exact pattern matching algorithm for biological sequences," Current Science, vol. 91, no. 1, pp. 47–53, 2006.

[25] Y. Huang, X. Pan, Y. Gao, and G. Cai, "A Fast Pattern Matching Algorithm for Biological Sequences," IEEE, pp. 608 – 611, 2008.

[26] Y. Huang, L. Ping, X. Pan, and G. Cai, "A Fast Exact Pattern Matching Algorithm for Biological Sequences," International Conference on BioMedical Engineering and Informatics, pp.8-12, 2008

[27] A. F. Klaib and H. Osborne. "Searching Protein Sequence Database Using BRBMH Matching Algorithm," International Journal of Computer Science and Network Security (IJCSNS), vol. 8, no. 12, pp. 410-414, 2008.

[28] Hudaib, A., R. Al-khalid, D. Suleiman, M. Itriq and A. Al-Anani, "A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW)" Journal of Computer Science 4 (5): 393-401, 2008

[29] A. F. Klaib and H. Osborne, "OE Matching Algorithm for Searching Biological Sequences". In: International Conference on Bioinformatics, Computational Biology, Genomics and Chemoinformatics (BCBGC-09). ISRST, Orlando, Florida, pp. 36-42. 2009

[30] Senapati, K.K., S. Mal and G. Sahoo, "RS-A Fast Pattern Matching Algorithm for Biological Sequences" , International Journal of Engineering and Innovative Technology (IJEIT), 1(3): 116-118.

[31] Suleiman, D., Hudaib, A., Al-Anani, A., Al-Khalid, R. and Itriq, M. "ERS-A Algorithm for Pattern Matching". Middle East Journal Scientific Research, 15, 1067-1075, 2013

[32] Suleiman, D., Itriq, M., , A., Al-Anani, Al-Khalid, R,, and Hudaib, A., "Enhancing ERS-A Algorithm for Pattern Matching (EERS-A)", Journal of Software Engineering and Applications, 8, 143-153, 2015.