Attacking Pattern Matching Algorithms Based on the Gap between Average-Case and Worst-Case Complexity

Yu Zhang, Ping Liu, Yanbing Liu, Aiping Li, Cuilan Du, and Dongjin Fan

***Abstract*—We have developed an effective method to generate attack data for widely-used pattern matching algorithms. Perceived to be a fundamental problem in the field of computer science, pattern matching has received extensive attention in research and has been used in a variety of fields to include information retrieval, computational biology, information security, etc. The extensive applications of pattern matching are mainly rooted in the fact that pattern matching algorithms, such as SBOM and WuManber, typically have a low time-complexity. However, in the worst case time-complexities of algorithms are very high, making pattern matching algorithms vulnerable to algorithmic complexity attacks (in other words, one might significantly slow down pattern matching algorithms simply by feeding them with specifically-designed text). In this study, we investigated this potential vulnerability by proposing a dynamic programming method designed to attack text having knowledge of patterns. Experimental results suggest that SBOM and WuManber run the specifically-designed text slower than random text (real text). Interestingly, it has been observed that the attacking method is still effective even when parts of patterns are only known, meaning that even a leak of small proportions has the potential to lead to severe attacks. Finally, we propose some suggestions to reduce the risks of these attacks. To the best of our knowledge, this is the first time that an attacking pattern matching approach is proposed based on algorithm complexity.**

***Index Terms*—Computer security, algorithmic complexity attacks, pattern matching, SBOM algorithm, WuManber algorithm.**

I. Introduction

Recognized as one of the most classic problems in the field of computer science, pattern matching (also called string matching) has received intensive attention in research. Until now, hundreds of efficient pattern matching algorithms have been proposed and widely used in various fields such as information retrieval, virus detection, computational biology, and so on. More recently, the development of network information security applications poses a high-performance requirement on multi-pattern matching technology.

Manuscript received January 25, 2013; revised April 10, 2013. This work was supported by the National 242 Information Security Projects (No.2010A029) and Chinese Academy of Sciences Strategic Technology Pilot Projects (No. XDA06030200).

Yu Zhang, Ping Liu, and Yanbing Liu are with Institute of Information Engineering, University of Chinese Academy of Sciences, 100093, Beijing, China (e-mail: zhangyu@iie.ac.cn, liuping@iie.ac.cn, liuyanbing@iie.ac.cn).

Aiping Li is with National University of Defense Technology, 410073, Changsha, China (e-mail: apli1974@gmail.com).

Cuilan Du and Dongjin Fan are with National Computer Network Emergency Response Technical Team/Coordination Center of China, 100029, Beijing, China (e-mail: dcl@isc.org.cn, fandongjin@126.com).

According to search direction, the popular pattern matching methods can be classified into two categories: prefix matching and suffix matching. The popular prefix matching algorithms include Aho-Corasick [1], while widely used suffix matching algorithms include SBOM (Set Backward Oracle Matching) [2], Set Horspool [3], and WuManber [4]. Due to an excellent average-case performance, suffix matching algorithms have been widely used.

A. Suffix Matching Algorithms

Before presenting our attacking approaches, we first provide a brief introduction to SBOM and WuManber algorithms. Generally speaking, both WuManber and SBOM algorithms consist of a preprocessing and matching phase.

In the preprocessing phase, a set of suffixes are selected from the patterns. Here, a suffix is designed to represent a shift operation when a mismatch between pattern and text occurs.

In the matching phase, a sliding window moves from left to right over text in the window as patterns are compared from right to left. The distance to move is determined by the suffix identified in the window. As a special case, the shift with distance of 0 denotes a possible match between patterns and text.

WuManber and SBOM are representatives of suffix matching algorithms; however, these two algorithms differ in both the preprocessing and matching steps. Specifically, WuManber selects as suffixes (denoted as B) all substrings of patterns with length b, and then constructs shift and hashing tables in the preprocess phase. Here, the shift table is used to determine how far a shift can safely take place when a mismatch occurs, and the hashing table records the patterns to be verified when a pattern matches text in the sliding window completely. An example of this process is shown in Fig. 1.

Unlike WuManber posing length limits on suffixes, the SBOM algorithm uses all substrings of patterns as suffixes. Then the trie technique is employed to organize reversed suffixes into a factor oracle. This way, all factors of the patterns can be recognized using the resulting factor oracle. In the matching phase, the text is scanned backwards with the factor oracle — if the oracle fails to recognize a factor at a given position, the pattern will be shifted beyond that position. An example of this process is shown in Fig. 2.

By employing the suffix structure to express shift distance, suffix algorithms can move the sliding window as far as possible without concerns of missing any matches; thus, the number of comparisons is significantly reduced, leading to a considerable performance improvement. On the other hand, if one finds ways to make the algorithm shift as few times as

DOI: 10.7763/JACN.2013.V1.45

possible, the verification operations will be performed much more frequently. In this case, the suffix algorithm performance will inevitably decline.

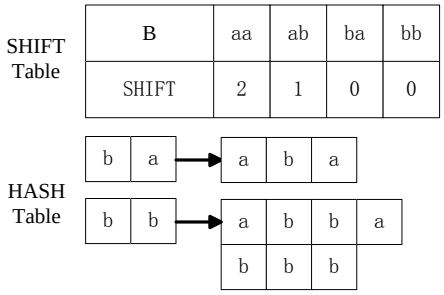


Fig. 1. An example of SHIFT table and HASH table used in WuManber algorithm. Pattern set: *{aba, abba, bbb}*. Character size: *2*. *b = 2*.

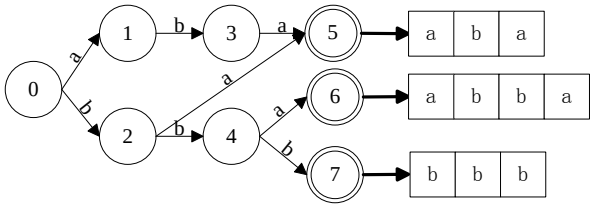


Fig. 2. An example of SBOM factor oracle. Pattern set: *{aba, abba, bbb}*. Character size: *2*.

B. Related Work

Currently, efficient pattern matching algorithms serve as the core of multiple practical systems, where performance guarantee is highly emphasized. To keep these systems free of attacks, intensive research has been conducted to analyze the vulnerability of the system and to propose corresponding protection strategies.

In addition to the standard types of attacks which aim at system leaks (such as hacking), a new type of attacking approaches has appeared. This approach actually targets the weakness of algorithms themselves. In other words, a system could potentially be slowed down simply by forcing the core algorithm to run in its worst-case scenario. One of the pioneering works following this idea is given by [5] to attack a hashing algorithm. In this work, the vulnerability of the hash table in the Squid Web cache, DJB DNS server, and Perl language were analyzed. A specifically-designed text was then constructed to attack the hashing algorithm. Experiments on the Bro system showed that simple feeding of specific text into the system resulted in expending the entire CPU resources in 6 minutes, and an overall packet loss rate of 71%.

The pioneering work on hashing algorithms has simulated a variety of research projects on algorithm security. For example, an NIDS evasion method [6] was proposed through algorithmic complexity attacking the rule matching algorithm, making inspection times 1.5 million times slower. In [7], the skewness in network traffic distribution was utilized to yield effective attacks. Recently, attacks based on algorithmic complexity were proposed for Snort [8], [9] and regular expression matching systems [10].

To resist these new algorithmic complexity attacks, a set of corresponding research has been conducted. A reverse engineering tool was used to analyze weakness of network-based intrusion detection systems [11] and to finally produce non-malicious traffic. Interestingly, the analysis also demonstrated that withholding signatures does not necessarily increase the resistance of a system to evasion and over-stimulation attacks.

In addition, theoretical analysis of the LDM (linear DAWG matching) [12] algorithm suggests that LDM has an O(m) worse-case time complexity and an average case complexity of O(n(log m)/m). This algorithm would be the best choice if we only consider the algorithmic complexity attacks.

II. Constructing Attack Text

A. Problem Statement

According to the principle of suffix matching algorithms, time is mainly consumed in the following two steps of the matching phase: 1) calculating the shift distance in the current sliding window (as denoted by w_1 and w_2) verifying candidates when there is a possible match (as denoted by w_2). The ratio of these two types of time costs varies in different algorithms. Thus, the following formula is used to describe the total time cost:

$$w = p\sum w_1 + q\sum w_2 \quad (1)$$

where p and q denote the weights of these two steps. An inherent problem with the generation of attack data involves the design of text to maximize w.

Before describing our construction method, we briefly introduce the following notations: Let $\Sigma(|\Sigma|=\sigma)$ denote the character set. $P=p_1, p_2, ..., p_r$ denote the pattern set. The length of the shortest pattern is denoted as lmin while the length of text T is n, suffix set is π, and suffix in current window is B. The length of B is b ($b\epsilon$[1, lmin]). The shift distance of B is $s(B)$. If $s(B)$ is equal to 0 , then the number of candidates is L(B). Thus, the problem of constructing attack data can be described as follows: for a given P, T must be constructed over which total w can be maximized.

B. CAM: Construction Attacks to Matching Algorithms

For any k ($k\epsilon$ [lmin, n]), all text of length k are denoted by U_k ($\Sigma Uk= U$). According to each suffix B ($B\epsilon\pi$), Uk can be divided into many subsets. A subset of U is denoted by U_k, which further represents a subset composed of text with length of k and B at the end. There are three properties which can be derived from this definition:

1) For any k ($k\epsilon$ [lmin, n]), $UZ_{k,B}= U_k(B\epsilon\pi)$.
2) $X\cap Y=\phi$, if $X,Y\epsilon U$ and $X\neq Y$.
3) For any k($k\epsilon$ [lmin, n]), a text set constructed by adding $s(B)$ characters to the end of each text in $Z_{k,B}$ belongs to $Z_{k+s(B),B'}$. B' is a suffix of the $s(B)$ characters.

A problem with constructing attack data essentially involves the proper positioning of characters throughout the text to cost the most time during the matching phase. This problem can be depicted by the longest path problem in a DAG (Directed Acyclic Graph) with vertex set V and an edge set E.

1) V: For each $Z_{k,B}$, we define a corresponding vertex $v_{k,B}$.
2) E: For any$v_{k,B}$ and $B'(|B'|=b')$, an edge is defined between vertex $v_{k,B}$ and vertex $v_{k+s(B)\ ,\ B'}$. The edge is assigned

with a weight $c(e(v_{k,B}, v_{k+s(B),B'}))$ and a mark denoted as $p(e(v_{k,B}, v_{k+s(B),B'}))$ according to the following formulas:

$$c(e(v_{k,B}, v_{k+s(B),B'})) = \begin{cases} p \times b', if\ s(B) > b', \\ p \times b' + q \times L(B'), if\ s(B) = b', \\ \phi, otherwise. \end{cases} \quad (2)$$

$$p(e(v_{k,B}, v_{k+s(B),B'})) = \begin{cases} t_1 t_2 ... t_{s(B)-b'} B', if\ s(B) > b', \\ B' if\ s(B) = b', \\ \phi, otherwise. \end{cases} \quad (3)$$

For the convenience of presentation, we add v_{source} and v_{sink} to G. there are edges between v_{source} and $v_{l\min,B}(B \epsilon \pi)$. If $s(B) = 0$, weight of the edge is $pb + qL(B)$ and mark is B; otherwise, weight is set as 0 and mark represents any string length of which is lmin and end is B. There are edges between $v_{k,s}(k > n - l\min)$ and v_{sink}; the weights of these edges are 0 and mark is any string with length $n - k$.

Using this transformation, it is clear that the longest path in the DAG corresponds to the text that forces the matching algorithm to run more slowly. Due to the fact that all edges have positive weights, the longest path can be easily identified using classical Dijkstra algorithm.

Input: Pattern set P, n, $l$min, σ, π.
Output: The text $T = t_1, t_2, ..., t_n$ to maximize w.
1:Add node $v_{k,B}$ ($k \epsilon [l\min, n]$, $B \epsilon \pi$) to G.
2:Add edge $e(v_{k,B}, v_{k+l\min-b,B'})$ ($k \epsilon [l\min, n-1]$, $B \epsilon \pi$, $B' \epsilon \pi$) to G According to the formula (2).
3:Calculate $c(e(vk,B, v_{k+l\min-b,B'}))$ and $p(e(v_{k,B}, v_{k+l\min-b,B'}))$ According to the formula (2) and formula (3).
4:Add starting node and ending node to G with their edges.
5:Find the longest path P according to Dijkstra Algorithm.
6: $T = P$.

III. Experiments and Analysis

The following experiments were performed to evaluate our construction method: 1) The matching performance was evaluated and compared against both attack and real text, 2) the factors affecting the attack effect were analyzed, and 3) we investigated whether having prior knowledge of a partial pattern set can be used to yield effective attacks.

All experiments were performed on a PC (OS: Red Hat Enterprise Linux Server release 5.2, CPU: Quad-Core AMD Opteron(tm) Processor 2376, Memory: 16GB). The matching time is acquired using Linux command "time". In the same environment, we use the WuManber and SBOM algorithms to detect the CT (CAM Text) and RT (Random Text or Real Text), respectively. The speed of CT is equal to CS and the speed of RT is RS. Thus, we were able to analyze the attack effect by detecting changes in the speed (SR, Speed Reduced). We calculate SR by formula 4.

$$SR = 1 - \frac{CS}{RS} \quad (4)$$

The experiments were performed over the following data sets:

Real text: In this experiment, we use two groups of patterns: one extracted from the Snort rules and the other from the virus list of ClamAntiVirus. In particular, we used the Snort2.8.3 version and selected 5000 keywords (which are longer than 4 bytes from the Snort rules) to constitute the patterns. We used ClamWinFreeAntiVirus0.90 version and selected 20000 keywords which are longer than 10 bytes from its virus list to constitute the patterns. As control, we used a set of real network data released by MIT for the evaluation of network intrusion detection systems. The data set can be obtained on http: //www.ll.mit.edu/IST/ideval/. We used all the data in mit_1999_training_week1_friday_inside.dat (64MB).

Random text: Random patterns data and random text data were created under the prerequisite concept that patterns and the characters in text are all independent and follow equal probability distribution. Thus, with random data the results of the experiment can show the average performance of the algorithm.

CAM text: To build attack data, the CAM text was constructed on given patterns using the CAM algorithm. To construct the CAM text, we selected different values of q and p depending on the algorithm. As the suffix length is actually fixed in the WuManber algorithm, most of the corresponding candidates in a pattern set are also related in the matching phase. As such, we set p equal to 1 and q equal to 0 in formula (1). As the suffix length is not fixed in the SBOM algorithm and the number of suffixes is typically large, most of the shift distance is related in matching phase. As such, we set p equal to 0 and q equal to 1 in formula (1).

A. Effectiveness of the Attack

TABLE I: Searching Speed Comparison between the CAM Text and Real Text with WuManber.

Pattern Set	MIT Speed (MBps)	CAM Text Speed (MBps)	Speed Reduced (%)
Snort	74.702	23.414	68.66
ClamAntiVirus	50.605	22.241	56.05

TABLE II: Searching Speed Comparison between the CAM Text and Real Text with SBOM.

Pattern Set	MIT Speed (MBps)	CAM Text Speed (MBps)	Speed Reduced (%)
Snort	68.946	8.408	87.80
ClamAntiVirus	89.431	10.885	87.83

For both real and random data, the performances of matching algorithms (WuManber algorithm and SBOM algorithm) were compared. The results, as shown in Table I and II, demonstrate that (in a real environment) these attacks lead to more than a 50% reduction in speed.

B. Factors Affecting the Attack

The main factors affecting the matching speed of algorithms are σ, r, and lmin. We changed the value of σ, r, and lmin when n=1024 and performed three groups of tests in order to observe the algorithm's performance under attack.

1) Size of Character Set

The first group of tests involved the size of character sets. Fig. 3-6 illustrates that the percentage by which overall speed

is reduced increases rapidly with the increasing of the size of character set. When σ is greater than 64, speed is reduced by over 80%. From this result we conclude that the suffix matching algorithm is suitable to work in a large size of character set, however also remains very vulnerable.

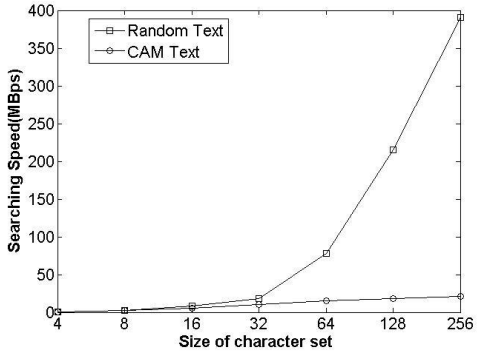


Fig. 3. Comparison of the attack performance with WuManber and different sizes of character sets. Pattern set size is 1000 with the length of the shortest pattern equal to 10.

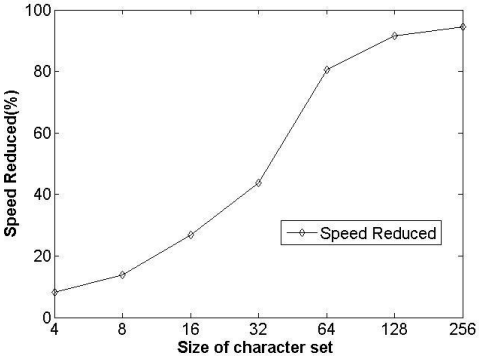


Fig. 4. Speed reduced with WuManber and different sizes of character sets. Pattern set size is 1000 with the length of the shortest pattern equal to 10.

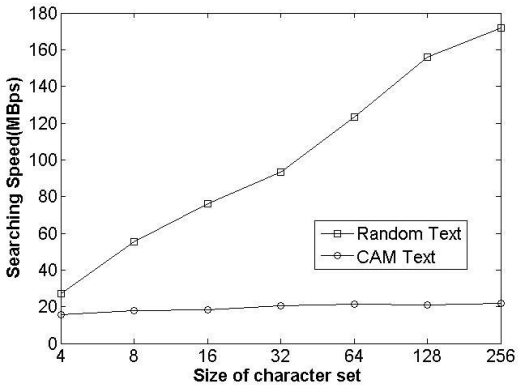


Fig. 5. Comparison of the attack performance with SBOM and different sizes of character sets. Pattern set size is 1000 with the length of the shortest pattern equal to 10.

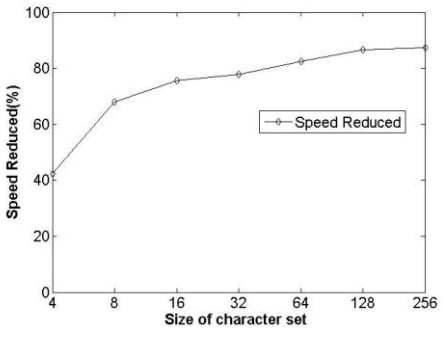


Fig. 6. Speed reduced with SBOM and different sizes of character sets. Pattern set size is 1000 with the length of the shortest pattern equal to 10.

2) Size of Pattern Set

The second group of tests involved the size of pattern sets. Fig. 7-10 illustrate that the percentage by which the speed of WuManber is reduced actually decreases with the increasing of the size of pattern set. Conversely, the percentage by which the speed of SBOM is reduced actually increases. According to our data, a different size of pattern sets has a small impact on the percentage of speed reduction, and is always greater than 80%.

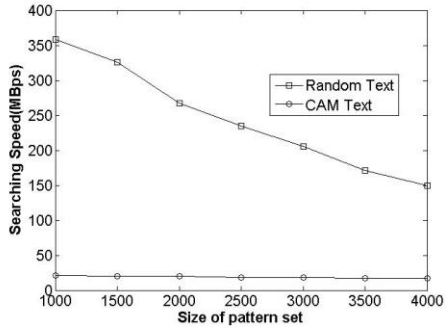


Fig. 7. Comparison of the attack performance with WuManber and different sizes of pattern sets. Character set size is 256 with the length of the shortest pattern equal to 10.

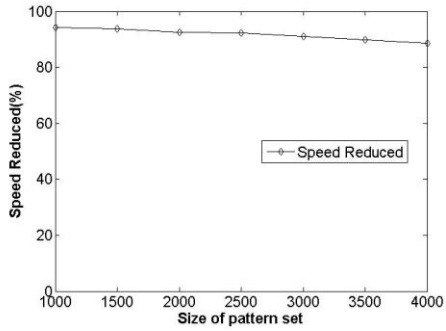


Fig. 8. Speed reduced with WuManber and different sizes of pattern sets. Character set size is 256 with the length of the shortest pattern equal to 10.

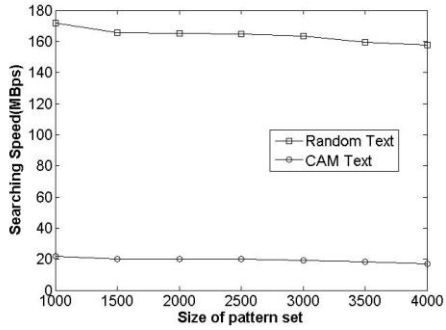


Fig. 9. Comparison of the attack performance with SBOM and different sizes of pattern sets. Character set size is 256 with the length of the shortest pattern equal to 10.

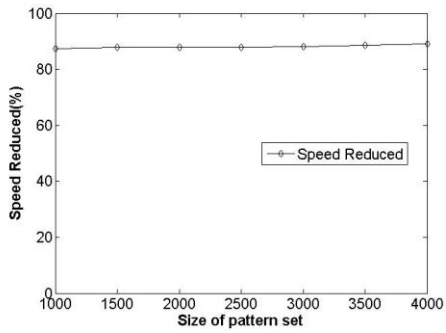


Fig. 10. Speed reduced with SBOM and different sizes of pattern sets. Character set size is 256 with the length of the shortest pattern equal to 10.

3) Length of the Shortest Pattern

The third group of tests involved the length of the shortest pattern. Fig.11-Fig. 14 illustrate that the percentage by which the speed of the two algorithms is reduced tends to increase with the increasing of the length of the shortest pattern. Similarly to the pattern size tests, the different length of the shortest pattern has a small impact on speed reduction, and is always greater than 80%.

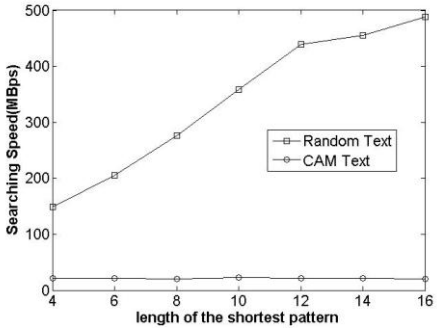


Fig. 11. Comparison of the attack performance with WuManber and different lengths of the shortest pattern. Character set size is 256 with the pattern set size equal to 1000.

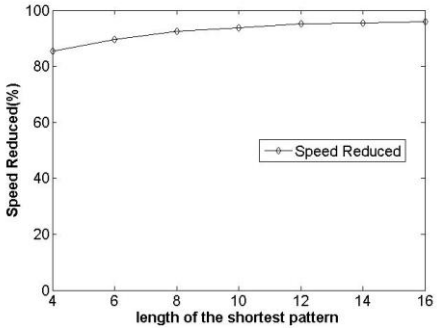


Fig. 12. Speed reduced with WuManber and different lengths of the shortest pattern. Character set size is 256 with the pattern set size equal to 1000.

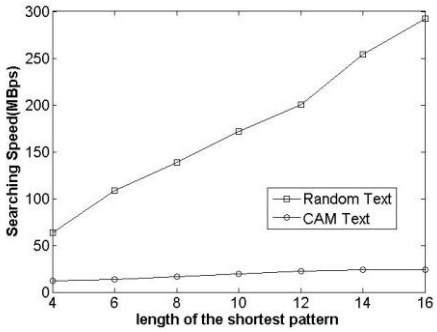


Fig. 13. Comparison of the attack performance with SBOM and different lengths of the shortest pattern. Character set size is 256 with the pattern set size equal to 1000.

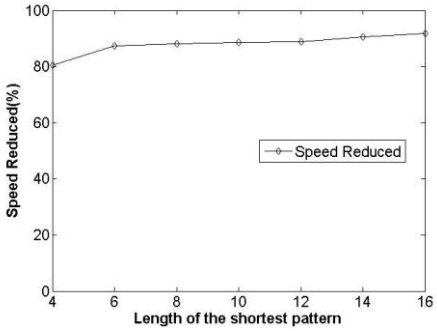


Fig. 14. Speed reduced with SBOM and different lengths of the shortest pattern. Character set size is 256 with the pattern set size equal to 1000.

The following conclusions were obtained as a result of these three groups of experiments:

1) Amongst the three factors tested, the size of a character set has the greatest impact on attack performance.
2) As the size of a character set increases, the attack has a greater effect.
3) Of the three factors tested, the size of a pattern set and the length of the shortest pattern both have a relatively small impact on attack performance. In contrast, our data shows that when the size of a character set is large, the effect of the attack is always significant.

C. Attack Using Part of Pattern Set

In real life, it is common to not know all components of a pattern set; thus, it is interesting to investigate whether this kind of attacks still works with only partial knowledge of patterns. Here, we use a part of a pattern set selected randomly to construct our attack data. Attack effects are displayed in Table III. It can be seen that attacks based on only a small part of the pattern set still have considerable effects. In addition, different patterns known to construct attack data can produce different results. In practice, by leaking only a partial piece of a pattern, we can use this method to test our system.

TABLE III: ATTACK USING PART OF PATTERN SET. CHARACTER SET SIZE IS 256. PATTERN SET SIZE IS 1000. THE LENGTH OF THE SHORTEST PATTERN IS 10

Part of Pattern Set (%)	Speed Reduced of WuManber (%)	Speed Reduced of SBOM (%)
Snort	74.702	23.414
ClamAntiVirus	50.605	22.241

IV. CONCLUSION AND FUTURE WORK

In this paper, we present an analysis of algorithmic complexity attack against suffix matching algorithms in an effort to raise awareness surrounding the issue of attack data construction, and to help find the optimal algorithm to address this issue. Our experiments show that (in most cases) our algorithm usually leads to a greater than 80% reduction in attack speed. In addition, our experimental results suggest that using only part of the algorithmic pattern is sufficient to yield effective attacks.

Future works include:

1) Designing an anti-attack algorithm: It is prudent for the entire field to improve matching algorithms, enabling them to resist this kind of attack. Randomization might be an effective technique for this objective.
2) Algorithm security in network: The security of the algorithms, and how to properly evaluate and validate this factor, is another question that needs to be addressed.

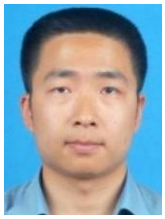
REFERENCES

[1] A. Aho and M. Corasick, and B. Laboratories, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 8, 1975, pp. 333-340.
[2] C. Allauzen, M. Crochemore, and M. Raffinot, "Efficient experimental string matching by weak factor recognition," in *Proc. 12th Annu. Symp. On Combinatorial Pattern Matching*, 2001, pp. 51-72.
[3] G. Navarro and M. Raffinot, *Flexible Pattern Matching in Strings: Practical on-line search algorithms for texts and biological sequence.* Cambridge: Cambridge University Press, 2002.

[4] S. Wu and U. Manber, "A fast algorithm for multi-pattern searching," Dept. of Computer Science, University of Arizona, Tucson, AZ, TR-94-7, 1994, 1051
[5] S. A. Crosby and D. S. Wallach, "Denial of service via algorithmic complexity attacks," in *Proc. 12th USENIX Security Symposium*, 2003, pp. 29-44.
[6] R. Smith, C. Estan, and S. Jha. "Backtracking Algorithmic Complexity Attacks against a NIDS," in *Proc. 22nd Annual Computer Security Applications Conference*, 2006, pp. 89-98
[7] A. El-Atawy, T. Samak, E. Al-Shaer *et al.*, "On using online traffic statistical matching for optimizing packet filtering performance," in *Proc. IEEE INFOCOM'07*, 2007, pp. 866-874.
[8] S. Rubin, S. Jha, and B. P. Miller, "Automatic generation and analysis of NIDS attacks," in *Proc. 20th Annual Computer Security Applications Conference*, 2004.
[9] T. Tung, A. Issam, E. Al-Shaer *et al.* (2008). Evasion Attack on Stateful Signature-based Network Intrusion Detection, University of Waterloo. [Online]. Available: http://www.cs.uwaterloo.ca/research/tr/2008/CS-2008-18.pdf
[10] M. Becchi, M. Franklin, P. Crowley, "A workload for evaluating deep packet inspection architectures," in *Proc. the 2008 IEEE International Symposium on Workload Characterization (IISWC)*, 2008.
[11] C. Kruegel, D Mutz, W. Robertson, G Vigna *et al.*, "Reverse engineering of network signatures," in *Proc. AusCERT Asia Pacific Information Technology Security Conference*, 2005.
[12] L. He, B. Fang, X. Yu, "A time optimal exact string matching algorithm," *Journal of Software*, vol. 16, no. 5, 676-683, 2005 (In Chinese).

Yu Zhang was born in 1987. He has been a Ph.D. candidate since 2012 in the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. He received his Master degree from the Institute of Computing Technology, the Chinese Academy of Sciences, Beijing, China. His current interests include string matching algorithms and network information security.

Liu Ping was born in 1972. She is a senior engineer at the Institute of Information Engineering, the Chinese Academy of Sciences, Beijing, China. Her main research interests include information security, data stream processing and string matching algorithms.



Yanbing Liu was born in 1981, Ph.D. He is an assistant researcher ar the Institute of Information Engineering, the Chinese Academy of Science, Beijing, China. His main research interests include string processing algorithms and information security.

Aiping Li was born in 1974, Ph.D. He is an associate researcher at National University of Defense Technology, Changsha, China. His current research interests include artificial intelligence, database and distributed computing.

Cuilan Du was born in 1966. She is an associate senior engineer at National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China. Her main research interests include network information security.

Dongjin Fan was born in 1983, Ph.D. She is an engineer at National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China. Her main research interests include computer application and image processing.