Speed Enhancement of Snort Network Intrusion Detection System

Safaa O. Al-Mamory
College of Computer Technology, University of Babylon

Abstract

Network Intrusion Detection System (NIDS) is a security technology that attempts to identify intrusions. Snort is an open source NIDS which enables us to detect the previously known intrusions. However, Snort NIDS has several problems one of them is the efficiency problem. We suggest using distributed environment in order to enhance it. We achieved this goal by enhancing the Snort's string matching engine through using a LAN of computers, where each computer in the LAN matching a subset of the monitored attacks. The experimental results show that it is possible to improve Snort's efficiency using distributed environment.

الخلاصة

ان اكتشاف الهجومات في شبكات الانترنت تستخدم مجموعة تقنيات تحاول معرفة الاختراقات الامنيه. برنامج سنورت هو برنامج مفتوح المصدر لاكتشاف الهجومات المعروفه مسبقا". مع ذلك، فأن برنامج سنورت يعاني من مجموعة مشاكل من ضمنها هي مشكلة السرعه. لقد اقترحنا استخدام المعالجه الموزعه لزيادة سرعة السنورت. قمنا بذلك من خلال تحسين محرك مطابقة السلاسل التابع لبرنامج سنورت من خلال استخدام شبكة حاسبات محليه. ان كل حاسبه في هذه الشبكه تقوم بمراقبة جزء من الهجومات. النتائج العمليه اثبتت كفاءة الطريقه المقترحه. اضافة الى ذلك، فأن عملية اختبار السنورت اصبحت اسهل.

1. Introduction

The study of Network Intrusion Detection System (NIDS) has become an important aspect of network security. Snort is an open source NIDS which enables the research community to easily do experiments. It uses the misuse detection method which uses a set of signatures to detect previously known attacks. Unfortunately, it has several weaknesses like providing unmanageable amount of alerts, vulnerability to flooding attacks, etc.

In this paper, the problem of efficiency enhancement of Snort NIDS is considered. We suggest using distributed processing to solve this problem. Every node in distributed environment will be responsible for detecting a partition of the overall monitored attacks. By distribute attacks' signatures across several computers (like a LAN of computers, as can be seen in Figure 1) we can enhance the attacks' detection speed.

The experimental results have shown that using a LAN of computers is enhancing Snort's efficiency and detection. In addition, three string matching algorithms are considered which are Kunth Morris Pratt (KMP) algorithm, Karp-Rabin algorithm, and Boyer-Moore algorithm. Experimentally, Boyer-Moore algorithm has been proved to be the best among these algorithms in distributed environment.

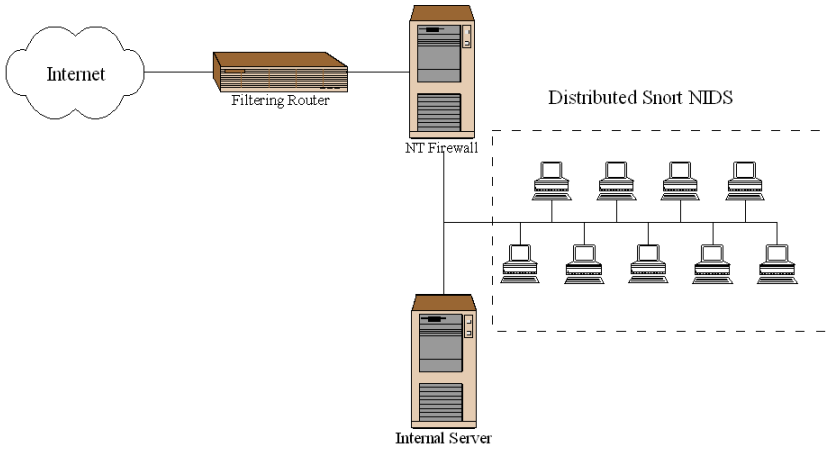

Figure 1. The position of the distributed Snort NIDS

This paper is organized as follows: Section 2 surveys the related work. The string matching algorithms that are considered in this paper have discussed in Section 3. Section 4 discusses how the efficiency of Snort NIDS can be enhanced. Section 5 presents the experimental results. Finally, Section 6 concludes this paper.

2. Related Work

In this section, we will review a part of the existing methods for enhancing the efficiency of Snort NIDS. These methods can be classified into one of the following: (a) reducing the number of rules, (b) enhancing the string matching algorithm, (c) reducing the monitored packet volume, (d) using faster computer, (e) turning off preprocessors, (f) using parallel computer, (g) using distributed environment, (h) using load balancing hardware. Depending on the selected method, we have different advantages and disadvantages. For example, using a parallel computer will be too expensive. In this paper, we have merged the solutions of class (b) and class (g).

The first direction is to improve the speed of matching process by using ad-hoc techniques. One straightforward idea in this direction is to put the rules with the similar criteria in the same group. During detection, the common constraints of a rule group need only be checked once. Kruegel et al. (Kruegel, 2003) have used machine learning clustering techniques to improve the matching process. They used decision trees to optimize the rules-to-input comparison process (Kruegel, 2003).

As another direction, NIDS load-balancing hardware is available that evenly distributes traffic to a group of sensors. This ensures that no single sensor will be overwhelmed with traffic. The load balancer can support up to eight sensors monitoring the same network. For a load balancer to be used in an NIDS, setting it must support stateful mirroring. Stateful mirroring ensures that a sensor that is tracking a session continues to receive the same session data as the load balancer divides up traffic. If the load balancer were to randomly distribute packets, streams would be broken up and NIDS would be severely hampered. For this reason, it is imperative to have stateful mirroring technology present in any load balancer considered. The major disadvantage of load balancing is its cost. A load balancer with a gigabit tap can easily be more expensive than Snort sensor group utilizing it (Koziol, 2003).

3. Snort's String Matching Engine

In this section, we present three string matching algorithms. Kunth Morris Pratt (KMP) algorithm will be presented in Subsection *A*. Subsection *B* presents Karp-Rabin algorithm. Boyer-Moore algorithm has been presented in Subsection *C*. We have selected these algorithms because they are the most popular algorithms in the field of string matching.

The component of Snort NIDS appears in Figure 2. Snort NIDS receives packets from the network, then it triggers an alert for any suspicious activity. As can be seen from Figure 2, the detection engine is the central component of Snort NIDS. The detection engine depends on string matching algorithms to achieve its goal. Therefore, in the squeal of this section, we will study the behavior of three string matching algorithms.

A. Kunth Morris Prat Algorithm

Knuth et al. (Knuth, 1977) have proposed a string matching algorithm that turns the search string into a finite state machine, and then runs the machine with the string to be searched as the input string. KMP algorithm uses information about the characters in the searched string to determine how much to 'move along' that string after a mismatch occurs. This can be implemented by associating each element position in the searched string with the amount that can be safely moved forward. If there is a mismatch in a certain position, then the search can be moved forward by the amount associated with that position. KMP algorithm achieves a running time of $O(n + m)$, where n is the length of text and m is the length of pattern (Fide, 2006).

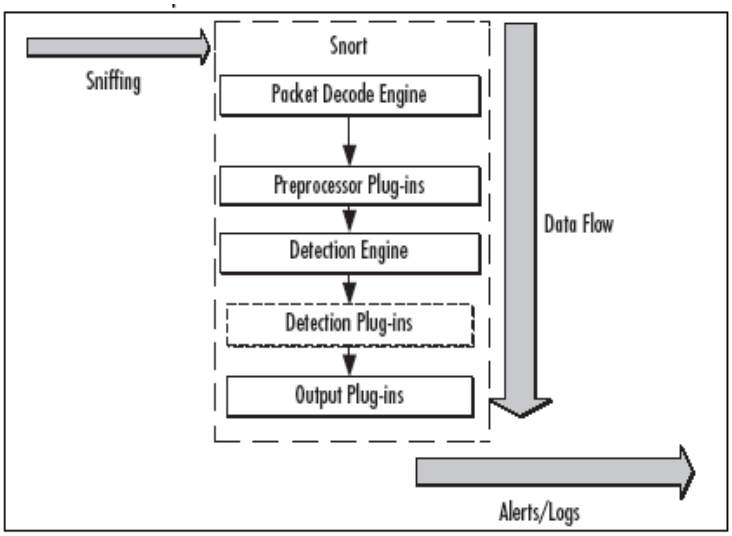

Figure 2. Snort Components Overview (Beale, 2003)

B. Karp-Rabin Algorithm

Karp and Rabin (Karp, 1987) have proposed a string matching algorithm that searches a pattern within a text using hashing. The main idea is to use a hash function to convert every substring in the text to a numeric value (hash value). The algorithm exploits the fact that if two strings are equal, their hash values are also equal. However, the same hash value can be assigned to different substrings. So, if the hash values are the same, then it is necessary to verify that the substrings actually match, which can take a long time for long substrings. Therefore, it is important to come up with a good hash function to keep the

average search time good. For text of length *n* and pattern of length *m*, its average case running time is $O(n + m)$. However, the worst case performance is $O((n - m + 1)\ m)$ (Fide, 2006).

C. Boyer-Moore Algorithm

The detection engine in Snort NIDS depends on Boyer-Moore string matching algorithm. The Boyer-Moore algorithm (Boyer, 1977),(Cormen, 2002) is one of the early algorithms and is the most widely used algorithm for string matching. It is based on two heuristics: bad character heuristic and good suffix heuristic. The novel aspect of the Boyer-Moore algorithm and the reason for its effectiveness is that character matching is performed right-to-left. The bad character heuristic shifts the search string to align the mismatching character with the rightmost position at which the mismatching character appears in the search string. If the mismatch occurs in the middle of the search string, then there is suffix that matches. The good suffix heuristic shifts the search string to the next occurrence of the suffix in the string. For example, given a single pattern of length *n* to match, the input string is looked ahead by n characters. If the character at this point does not match with the pattern, the search pointer is moved ahead by $n + 1$ characters without inspecting the characters in between. If there is a match, the previous characters are compared. The Boyer-Moore algorithm shows a sub-linear performance in the average case (Fide, 2006).

4. Efficiency Enhancement Of Snort Nids

The main idea of this paper is the using of distributed environment to enhance the efficiency of Snort NIDS. To achieve this goal, we have used a LAN of computers (as a distributed environment) to run Snort NIDS. Snort NIDS has a set of rules representing a set of known attacks. Snort should match these rules against flow of packets. If a matching has been found, then an alert should be triggered. The following definitions state the main idea.

Definition 1: Let *A* be the set of attacks that Snort NIDS should monitor, where $A=\{A_1, A_2, \ldots, A_n\}$. The set *A* can be partitioned to disjoint subsets such that $A=\{S_1, S_2, \ldots, S_m\}$ where $S_1=\{A_1, A_2, \ldots A_i\}$, $S_2=\{A_{i+1}, A_{i+2}, \ldots, A_j\}$, …, $S_m=\{A_k, A_{k+1}, \ldots, A_n\}$.

The considered criteria in partitioning attacks can vary. We can cluster them according to attacks' type (like FTP attacks, web attacks, etc.), attacks transport layer protocols (TCP attacks vs. UDP attacks), etc. The definition of any node in the (used) distributed environment is as follows.

Definition 2: *Snort Node*$_i$ is any node (or computer) in distributed environment that have to monitor S_i (which appeared in Definition 1) of the set *A*.

Now, we are ready to describe the pseudo code describing in Figure 3. As we can see in the figure, the pseudo code has two parts which are network configuration part and packet flow monitoring part. According to network configuration, the number of Snort Nodes should be specified and given to variable K (line 2). Then, Snort NIDS will be installed on *K* Snort Nodes as in line 3. Hereafter, every Snort Node will have subset of signatures S_i (line 4). With respect to packet flow monitoring (line 5-11), every *Snort Node*$_i$ will monitor packet flow with S_i and will issue an alert if any suspicious activity will be noticed.

```
Input: Packet Flow, # of Snort Nodes
Output: Set of detected attacks (if any)
Begin
1.  // Network configuration part
2.      K= Number of Snort Nodes
3.      Setup the Snort NIDS on K nodes
4.      Install signature set S_i on Snort Node_i, ∀i i ≤ k
5.  // Packet flow monitoring; this part is in every Snort Node
6.      While(true)
7.          Snort Node_i monitors packet flow with S_i, ∀i i ≤ k
8.            If any detected attacks then
9.               Issue an Alert
10.           End if
11.     End while
End
```

Figure 3. The pseudo code of the working procedure

There are two variations of the deployment of Snort's signatures in distributed environment. In the first one, the same copy of Snort's signatures can be used in several nodes. This deployment is used to monitor the network in different locations (for example, before the firewall, after the firewall, before the router, in a DMZ). We can say that this deployment give us global view of attacks because it is used to monitor the network from different points of view. In the second variation, Snort's signatures are distributed over several nodes as mentioned in Definition 1. This variation gives us a local view of attacks because it focuses on solving the problem of flooding Snort. Table 1 shows a comparison between different deployments of Snort signatures (the comparison is not exhausted).

Table 1. The comparison between different deployments of Snort's signatures.

Signatures' Type	Single-PC	Multi-PC
Snort with Complete Signatures	Advantage • Give local view of attacks. Disadvantage • Vulnerable to flooding attacks	Advantages • Give global view of attacks. • High availability Disadvantages • Vulnerable to flooding attacks • High volume of generated alerts.
Snort with Subset Signatures	Advantages • Fast processing. • Immune against flooding attacks. Disadvantage • Low detection rate.	Advantages • Give global view of attacks. • High availability. • High detection rate. • Immune against flooding attacks.

5. Experiments And Results

In this section, we describe the experiments conducted to evaluate the proposed system. The proposed system was tested on an Intel processor 2.2 GHz Core 2 Duo with 1 GB of RAM running Windows XP. A LAN of nine computers was used in our

evaluation. In addition, Snort NIDS (Roesch, 1999), an open-source signature-based NIDS, has been used here as a tool to be enhanced. We have used Java Builder 6 as a programming language and have used Snort NIDS version 2.6.1in our experiments.

The string matching algorithm that is used in the search engine of Snort NIDS is Boyer-Moore algorithm. However, three different string matching algorithms were used in our experiments (KMP algorithm, Karp-Rabin algorithm, and Boyer-Moore algorithm) in order to check the efficiency and the behavior of these algorithms in the distributed environment. These algorithms have different behavior with respect to the length of the pattern as shown in Figure 4. As can be noted with Boyer-Moore algorithm, the longest the pattern to be searched for, the minimum execution time is. In addition, the other algorithms have not been affected by the length of the pattern.

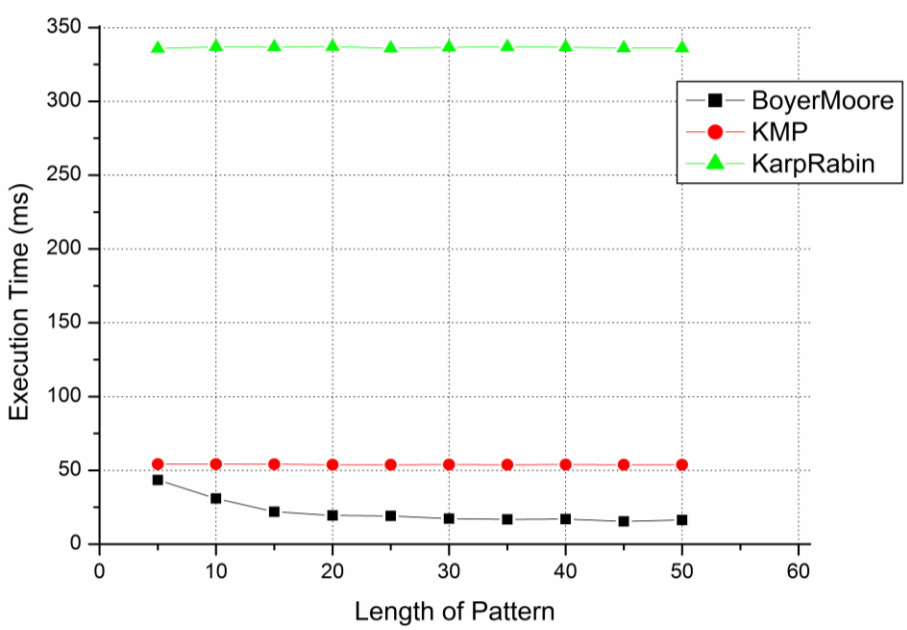

Figure 4. The relation between length of pattern and execution time.

Furthermore, we have changed the size of the file to be searched in while the length of the pattern was 10 characters for the three algorithms. This can be seen in Figure 5 where all the three algorithms are affected by the file size but the super algorithm was the Boyer-Moore algorithm. The worst algorithm was Karp-Rabin algorithm while the behavior of KMP algorithm was acceptable.

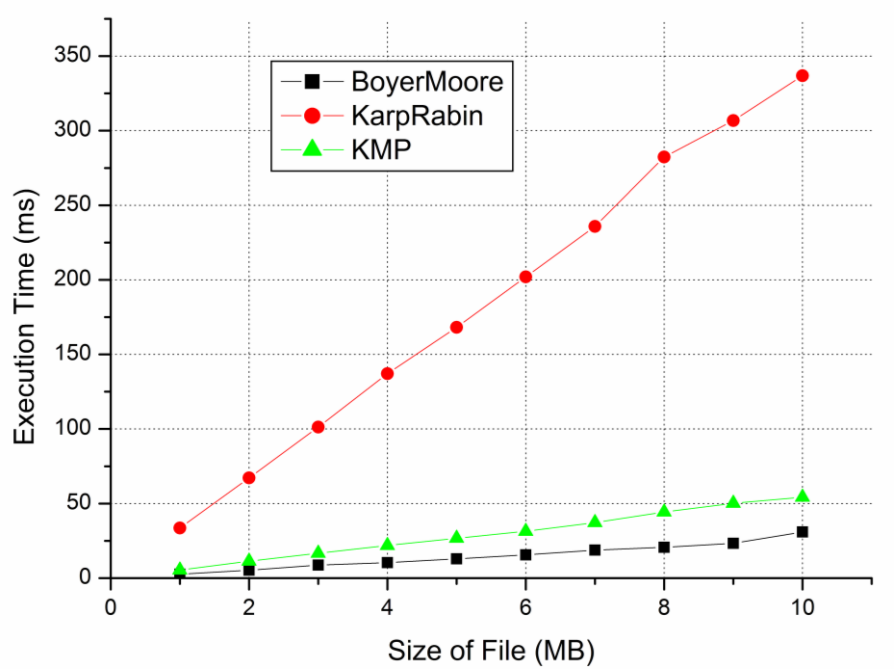

Figure 5. The relation between file size and execution time.

After applying the three algorithms in a distributed environment which is composed of a LAN of nine computers having the same hardware properties, different behavior has been noted from these algorithms. Nine computers have been used here because the LAN in our laboratory contains only these computers work properly. In addition, the main important fact is that we have done the experiments with more than one computer.

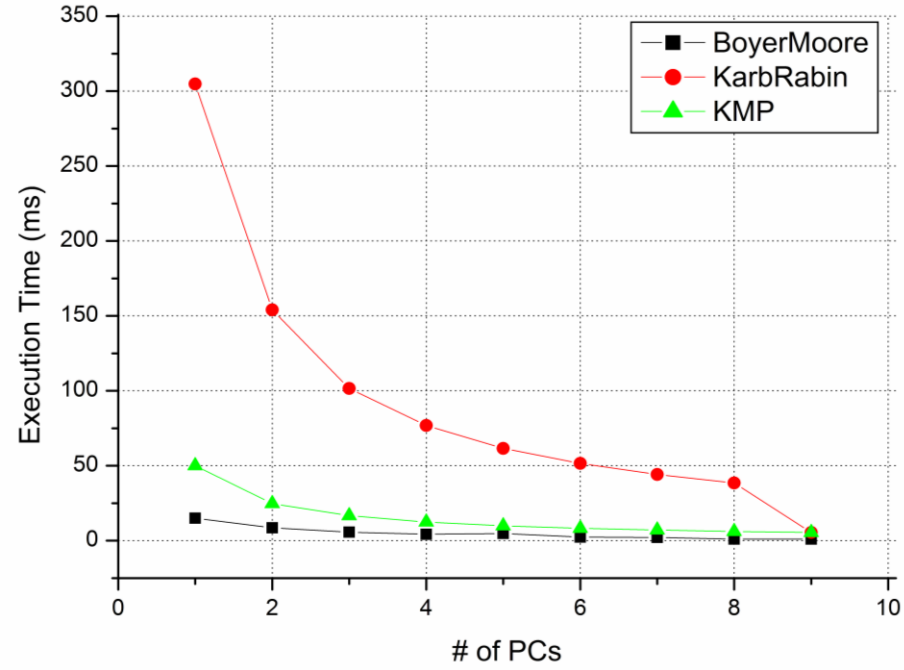

Figure 6. The relation between number of PCs and execution time.

The achieved speedup and efficiency are measures of the quality of a parallel implementation. The speedup achieved by a parallel algorithm running on P processors is defined as the ratio of the execution time of the parallel algorithm on a single processor and the execution time of the parallel algorithm on P processors. The efficiency is equal to the speedup divided by P. Therefore, the definitions for the parallel speedup *S(n,*P) and the parallel efficiency *E(n,P)* are described in Equations 1 and 2 (Dirk, 1995), respectively.

$$S(n,P) = \frac{T(n,1)}{T(n,P)} \qquad \text{.....(1)}$$

$$E(n,P) = \frac{S(n,P)}{P} \qquad \text{.....(2)}$$

where n denotes the problem size, *T(n,1)* and *T(n,P)* denote the execution times of the algorithm on one and P processors respectively.

To compute the parallel speedup and parallel efficiency of the considered algorithms, we have use Equations 1 and 2 respectively. The obtained results for the parallel speedup are depicted in Figure 7 while Figure 8 shows the computed parallel efficiency for these algorithms. As can be seen in Figures 6, 7, and 8, the most benefit algorithm from distributed environment was Karp-Rabin algorithm. However, the efficiency of Boyer-Moore algorithm and KMP algorithm is also enhanced. These results prove our claim that is: "*the efficiency of Snort NIDS can be enhanced using distributed environment*."

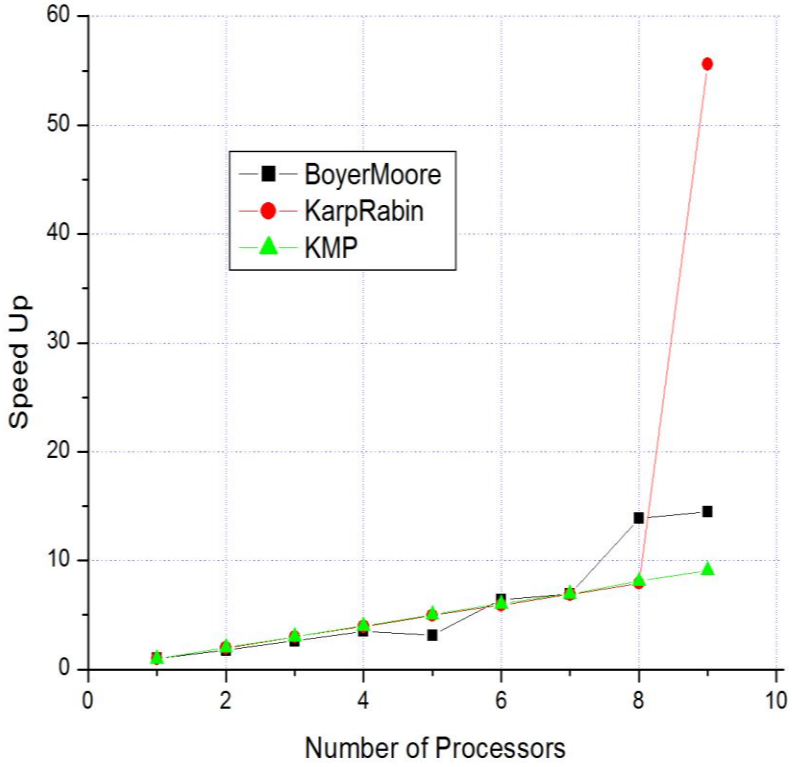

Figure 7. The relation between number of processors and Speedup

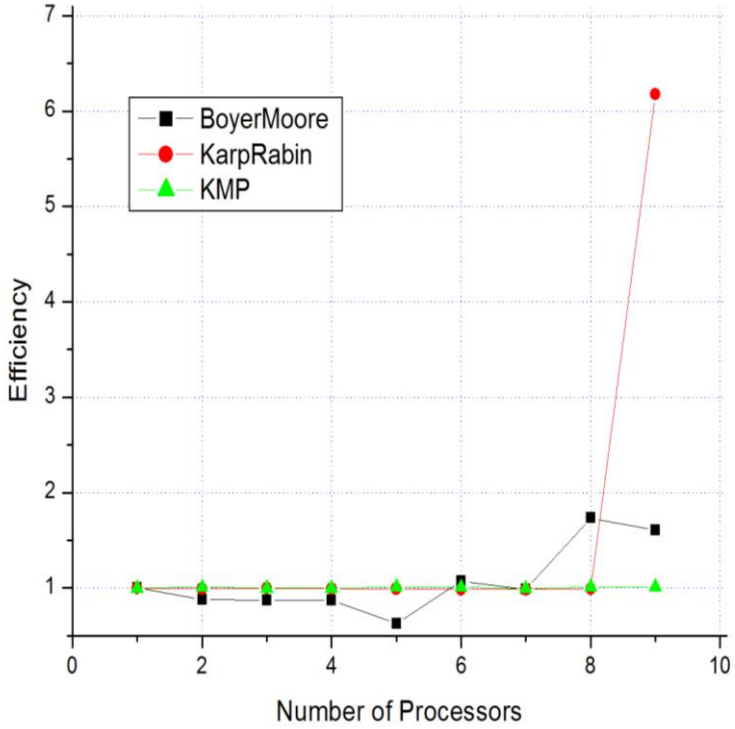

Figure 8. The parallel efficiency of three string matching algorithms on various numbers of processors

6. Conclusion

Snort is an open source NIDS which enable us to detect the previously known intrusions. We have suggested to use distributed environment in order to enhance it. We achieved this goal by enhancing the Snort's string matching engine through using a LAN of computers, where each computer in the LAN matching a subset of the monitored attacks. It is possible to improve Snort's efficiency using distributed environment. In addition, Snort's testability has been enhanced.

References

Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. (2002). Introduction to Algorithms, 2nd Edition, The MIT Press.

Dirk Roose, Rafael V. Driessche, (1995). "Parallel Computers and Parallel Algorithms for CFD: An Introduction," in the AGARD REPORT R-807 as Special Course on Parallel Computing in CFD.

Donald Knuth, James H. Morris, Vaughan Pratt. (1977) "Fast Pattern Matching in Strings," SIAM Journal on Computing, 6(2), pp.323–350.

Jack Koziol, (2003). Intrusion Detection with Snort, Sams Publishing.

Jay Beale, James C. Foster, Jeffrey Posluns, Brian Caswell, (2003). Snort 2.0 Intrusion Detection, Syngress Publishing, Inc..

Kruegel, C. ; Toth, T. (2003)"Using Decision Trees to Improve Signature-Based Intrusion Detection," In Proceeding of the 6th International Workshop on the Recent Advances in Intrusion Detection (RAID'2003), LNCS V. 2820.

R. S. Boyer, J. S. Moore, (1977). "A Fast String Searching Algorithm," Communications of the ACM, 20(10):762–772.

Richard M. Karp and Michael O. Rabin, (1987). "Efficient Randomized Pattern-Matching Algorithms," IBM Journal of Research and Development, 31(2):249–260.

Roesch, M, (1999). "Snort-lightweight intrusion detection for networks," In Proceeding of the 1999 USENIX LISA Conference, pp. 229–238.

Sevin Fide, Stephen Jenks, (2006). "A Survey of String Matching Approaches in Hardware," .