Pattern Matching Algorithms for Intrusion Detection and Prevention System: A Comparative Analysis

Vibha Gupta
Computer Science and Engineering
Thapar University
Patiala, India
vibha.gupta08@gmail.com

Maninder Singh
Computer Science and Engineering
Thapar University
Patiala, India
msingh@thapar.edu

Vinod K. Bhalla
Computer Science and Engineering
Thapar University
Patiala, India
vkbhalla@thapar.edu

Abstract— **Intrusion Detection and Prevention Systems (IDPSs) are used to detect malicious activities of intruders and also prevent from the same. These systems use signatures of known attacks to detect them. Signatures are identified through pattern matching algorithm which is the heart of IDPSs. Due to technological advancements, network speed is increasing day by day, so pattern matching algorithm to be used in IDPS should be fast enough so as to match the network speed. Therefore choice of pattern matching algorithm is the critical to the performance of IDS and IPS. Several pattern matching algorithms exist in literature, but which pattern matching algorithm will give best performance for IDPS is not known at hand. So in this work four pattern matching algorithms namely Brute-force, RabinKarp, Boyer-Moore and Knuth-Morris-Pratt has been selected for the analysis. These single keyword matching algorithms are mainly used. Performance of pattern matching algorithms is analyzed in terms of run time by varying number of patterns and by varying size of network captured (pcap) file.**

Keywords—Intrusion Detection and Prevention Systems (IDPSs); Pattern Matching; Rabin-Karp; Knuth-Morris-Pratt (KMP); Boyer-Moore

I. Introduction

Internet is the best development in the domain of communication industry. It provides a lot of services in terms of tele-working, e-commerce, e-banking, advertisement, manage large database of the confidential information for the companies. With increasing usage of internet increase its complexity of implementation which leads to security vulnerability to be exploited. Intrusion Detection System (IDS) is very confident tool for providing security against intrusions that are increasing day by day. The main aim of IDS is to identify misuse, unauthorized access and disruption of information system from both internal and external penetrators [1, 2]. It is done by collecting packets from inflow and outflow traffic and looking for malicious contents and behaviors, after finding attacks generates an alarm for the network security administrator. Whereas Intrusion Prevention System (IPS) first of all detects the attacks, on the basis of detection it can drop the malicious packet, change the malicious content to the legitimate content, resetting the sessions or can block the traffic from malicious Internet Protocol (IP) address. IPS is reactive in nature. In many organizations IPSs are combined with Firewalls to provide security against many web based attacks. Advanced IPS technique does not drop the packet but redirect it to the honeypot to know more about the attacker.

There are basically three intrusion detection methodologies which are used to identify attacks [1, 2]:

A. Anomaly-based IDSs:

Anomaly-based detection technique detects intrusion by comparing definitions and data of legitimate user with the events those show deviations from their normal behavior. Anomaly-based IDSs generate profiles by monitoring the characteristics of network traffic and compares them with the thresholds belong to that profile [1, 2].

B. Signature-based IDSs:

A Signature is a pattern of attack that is predetermined. Signature-based detection systems compare signatures with the monitored network traffic [1, 2]. After finding a match IDSs generate alarm and sometime take appropriate action. Signatures can be present any part of the network packet according to the nature of attack. Signatures can be match with the any content of the network packet.

C. Stateful Protocol analysis:

Stateful protocol analysis is process of examine protocols of network, transport and application layer like TCP and UDP protocols contain HTTP, FTP, SMTP, SNMP protocols [1, 2]. These systems must know how these protocols are supposed to work. So these are capable to detect attack with in protocol application data. Disadvantage is that there are lots of overheads and are not able to detect attacks that show normal characteristics of acceptable protocol behavior.

In this work signature based intrusion detection system and comparison of various signature matching algorithms are presented.

II. Problem Description

Because of rapid growth in network traffic for past few years leads to more attacks on the networks. Previous protective measures like Packet filtering firewall are not capable against such network security attacks [3]. Therefore intrusion detection systems are developed with signature database. There were 854 rules in the Snort IDS in 2000 where as in 2003 there were 2000 rules which became 7256 in 2007 [3, 4]. The increasing rate of rules that are used for attack detection is the problem for the detection engine. To match network packet with large database of signatures need a fast pattern matching algorithm which increase the speed of IDS. It has been found that most of the IDS spent 30% - 60% processing time on pattern matching [2, 3]. So the choice of better pattern matching algorithm becomes bottleneck of the performance.

A. Pattern Matching

The pattern Matching is a problem to find the first occurrence of given pattern in a stream of given text but the match should be exact which is known as Exact Pattern Matching.

Text T is a sequence of characters. An alphabet set ∑ is the set of alphabets that are used in a string. Pattern P is a string of length m and the prefix of P is substring of type P[0…k] and suffix of P is a substring of type P[k……m-1]. So the problem consists of finding a substring of T equal to P. An example of above statement is explained as follow:

Text: Buffer overflow attack is performed.
Pattern: attack

Then find the occurrence of pattern 'attack' in the text 'Buffer overflow attack is performed' is known as Exact Pattern Matching. Various algorithms used different techniques to find pattern in the text string.

III. Pattern Matching Algorithms

A. Brute-Force Search Algorithm:

The brute force algorithm is also known as Naive search algorithm. Brute force needs no pre-processing and use constant space. This algorithm searches the pattern through character by character matching. The brute force algorithm first compares every character in T with the first character in P then with the second and so on until the last character in P is reached. As in the algorithm there are two loops, first is looping m-n times and inner is looping n times so time complexity of brute force algorithm is O((m-n+1)n) in worst case[5].

NAIVE-SEARCH (T, P)

Input: T [1…….n] is a text string; P [1…….m] is a pattern string.
Output: Return the position of the substring of text matching P or -1

```
1. For i ← 0 to n-m do
2.         k ← 0
3.         While k<m && P[k]==T[i+k] do
4.                 k ← k+1
5.                 If k==m return i    //match occur
6. Return -1                          // match unsuccessful
```

B. Rabin-Karp Algorithm:

Michael O. Rabin and Richard M. Karp uses the technique of comparing hash values of the string in 1987 [6]. This pattern matching algorithm calculates a hash value for the pattern, and compares it with the hash value of same length subsequence of text string. If calculated hash values are not equal, the algorithm will calculate the hash value for next same length subsequence, therefore there is only single comparison per text subsequence. But if the hash values are equal, the algorithm will compare the pattern with that subsequence by following naive search algorithm [6].

RABIN-KARP (T, P)

Input: T [1…….n] is a text string; P [1…….m] is a pattern string
Output: Return the position of the substring of text matching P or -1

```
1. $h \leftarrow d^{m-1} \bmod q$
2. $p \leftarrow 0$
3. $t_0 \leftarrow 0$
         // Pre-processing
4. For k=1 to m do
5.       $p = (d\,p + P[k]) \bmod q$
6.       $t_0 = (d\,t_0 + T[k]) \bmod q$
         // Searching
7. For s=0 to n-m
8.       If $p==t_s$                  //hash value match occur
9.             If P[1…m]==T[s+1…….s+m]
10.                  Return s     // match occur
11.      If s < n-m
12.            $t_{s+1}=(d(t_s-T[s+1]h)+T[s+m+1]) \bmod q$
13. Return -1                          //match unsuccessful
```

C. Knuth-Morris-Pratt Algorithm:

This algorithm was discovered by Donald Knuth, Vaughan Pratt and James H. Morris in 1977 [7]. They found first linear time pattern matching algorithm by analyzing the brute force search. For this it calculate the prefix function. This function finds matches of prefixes of the pattern with the pattern itself. It helps to avoid backtracking on the text 'T'. Prefix function takes O(m) running time and matcher function takes O(n) running time, So total run time complexity is of O(m+n) [7].

KMP (T, P)

Input: T [1……n] is an array of text; P [1…….m] is an array of pattern
Output: Starting Return the position of the substring of text matching P or -1

```
1. f ← KMP-Failure-Function(P)    //build failure function
2. i ← 0
3. j ← 0
4. While i < n && j<m do
5.       If P[j] == T[i]then
```

```
6.                 i ← i + 1
7.                 j ← j + 1
8.         Else if j == 0
9.                 i ← i + 1
10.        Else
11.                j ← f(j-1)      // After matching prefix in P
12. If j==m return i-m             // Match occur
13. Return-1                       // Match unsuccessful
```

KMP-Failure-Function (P)

Input: P [1……m] an array of pattern
Output: Return a table of failure function for P

```
14. f (0) ← -1
15. For j=1 to m-1 do
16.        i ← f (j-1)
17.        While P[j] != P[i+1] && i>=0 do
18.                i← f (i)
19.        If P[j] == P[i]         // Matched i + 1 characters
20.                f (j) ← i + 1
21.        Else
22.                f (j) ← -1
```

D. Boyer-Moore Algorithm:

R. S. Boyer and J. S. Moore discovered an algorithm that searches for the index of the first occurrence of pattern in the text in 1977[8]. Boyer-Moore algorithm considered as the most efficient string-matching algorithm for many applications such as IDSs, text editors. BoyerMoore compares the pattern with text starting from right to left of the substring. This algorithm works on two heuristics.
Bad character heuristic: According to this if alphabet that does not match occurs somewhere else in the pattern then the pattern can be aligned to this text alphabet [8].
Good suffix heuristics: This approach work like KMP algorithm Pre-processing function. If suffix is matched then the pattern can be shifted until the next occurrence of suffix in the pattern is aligned to the text alphabets exactly same as suffix [8].

Matcher (T, P)

Input: T [1……n] is an array of text; P [1…….m] is an array of pattern
Output: Return the position of the substring of text matching P or -1

```
1.  i ← m - 1
2.  j ← m – 1
3.  charTable[] ← BAD_CHARACTER_SHIFT_BM(P);
4.  offsetTable[] ← GOOD_SUFFIX_SHIFT_BM(P)
5.  While i < n-1 do
6.        If P[j] == T[i] then
7.              If j == 0 then
8.                      Return i               //Match occur
9.              Else i←i-1
10.                  j ← j - 1
11.       Else i ← i + max(j - OffsetTable(T[i]), charTable(j))
 // shift P relative to T
12.   Return -1
```

Preprocessing algorithms of the BoyerMoore algorithm is defined as follow:

BAD_CHARACTER_SHIFT_BM (P)

Input: P [1…….m] is a pattern string
Output: Return a table of all alphabets with valid shifts

```
1.  alphabet_size ← 256
2.  table ← new array[alphabet_size].
3.  For i ← 0 to alphabet_size-1 do
4.        table[i] ← m
5.  For i ← 0 to m-1 do
6.        table[P[i]] ← m-1-i
7.  Return table
```

GOOD _SUFFIX_SHIFT_BM (P)

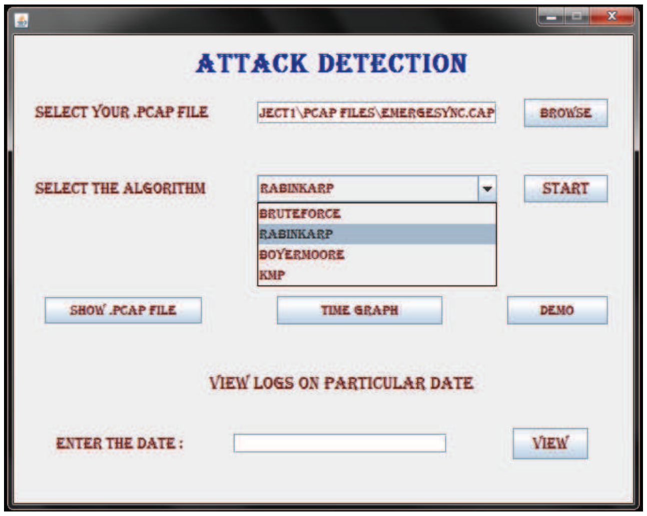
Input: P [1…….m] is a pattern string
Output: Return a table of patterns with valid shifts

```
1.  table ← int array[m]
2.  lastPrefixPosition ← m
      // check if i+1 a prefix of pattern
3.  For i ← m-1 to 0
4.  k ← i+1
5.     While( k < m)
6.      j ← 0
7.          if (pattern[i] == pattern[j]) then
8.               lastPrefixPosition ← k
9.       i ← i+1
10.      j ← j+1
11.  table[m - 1 - i] = lastPrefixPosition - i + m - 1;
12.  i ← i-1  // returns the maximum length of the substring
ends at i and is a suffix
13.  For i ← 0 to m
14.  len ← 0;
15.  j ← m-1
16.        While(i>=0)
17.              If(pattern[i]==pattern[j]) then
18.              len ← len+1
19.         i ← i-1
20.         j ← j-1
21.   table[len] ← m-1-i+len
22.   Return table
```

IV. Impelementation And Analysis

A. Implementation

Implementation of the selected algorithms is done using JAVA Netbeans platform and store patterns in MySql database. Implement four pattern matching algorithms Brute force, Boyer-Moore, KMP and RabinKarp. All the algorithms are able to detect attacks by searching signature of attacks in the packet stream. Read pcap file and convert pcap file into text file that is passed to the detection engine and we can also view the content of the pcap file. If matching occur calculates elapsed time by algorithms, extract source ip address from pcap file and attack name from database individually. By selecting particular date it shows the logs stored in database in the form of table.

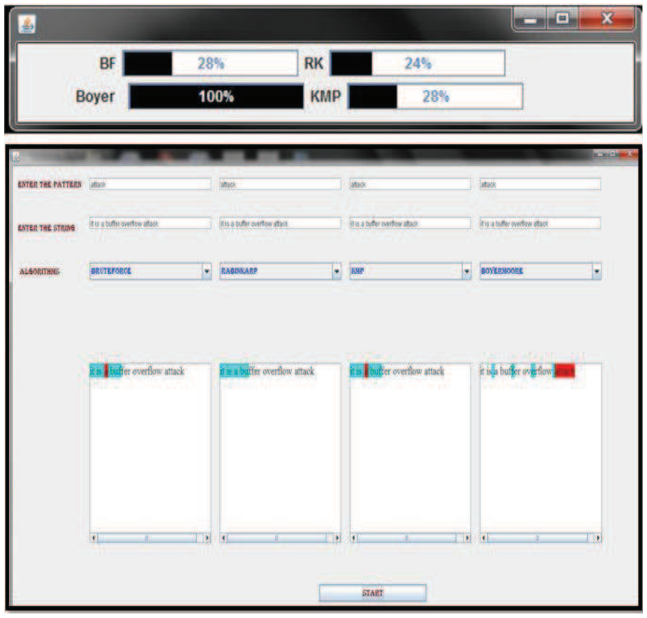


SELECT YOUR .PCAP FILE JECT1\PCAP FILES\EMERGESYNC.CAP

SELECT THE ALGORITHM RABINKARP

FIG 1. DEMONSTRATION OF THE WORKING OF ALGORITHMS

The four algorithms are implemented parallel using threads to demonstrate their working. By clicking on Demo button in above interface user can view the demonstration of the algorithms as shown in the following Fig 2.



BF 28% RK 24%
Boyer 100% KMP 28%

ENTER THE PATTERN attack attack attack attack

ENTER THE STRING it is a buffer overflow attack it is a buffer overflow attack it is a buffer overflow attack it is a buffer overflow attack

ALGORITHMS BRUTEFORCE RABINKARP KMP BOYERMOORE

FIG 2. DEMONSTRATION OF THE WORKING OF ALGORITHMS AFTER COMPLETION

Fig 1 shows that BoyerMoore has completed its searching before all other algorithms. Progressive bars are implemented to show that how much percentage of searching has been completed.

B. Analysis:

Running time taken by all the selected algorithms for different packet captured files against different number of patterns are recorded in the form of table. Table I shows the all recorded values.

TABLE I. RUNNING TIME OF ALGORITHMS TO DETECT ATTACKS

Size of file	No. of patterns	Running time of algorithms in sec			
		Brute-force	Rabin-Karp	KMP	Boyer-Moore
408,036 bytes	40	120	180	10	4
	100	296	360	19	11
	200	422	586	55	39
	400	689	720	108	92
	800	879	1003	252	186
	1000	1020	1245	339	223
1,609,728 bytes	40	240	390	19	9
	100	420	570	29	16
	200	596	774	67	52
	400	758	920	125	102
	800	960	1080	227	189
	1000	1130	1320	420	357
6,828,032 bytes	40	65	89	7	3
	100	158	186	10	6
	200	280	367	18	12
	400	389	508	39	26
	800	620	724	72	49
	1000	750	967	98	71
19,269,599 bytes	40	789	1030	73	48
	100	1290	1486	99	81
	200	1430	1703	127	102
	400	1890	2156	197	145
	800	2278	2589	345	232
	1000	2456	2834	410	308

On the basis of the above values a comparative analysis has been done by creating the graphs. Graphs are created by taking the running time of Brute force, RabinKarp, KMP and BoyerMoore algorithms for different size files and different number of patterns.

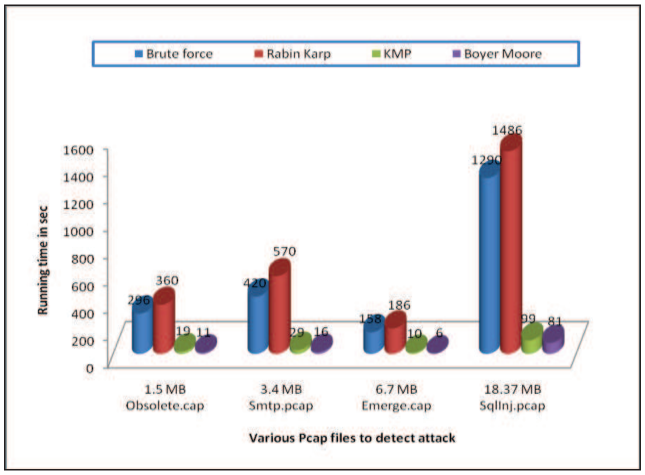


FIG 3. COMPARISON OF RUNNING TIME OF VARIOUS PATTERN MATCHING ALGORITHMS AGAINST FILE SIZE

Fig 3 shows a graph that is created by taking the running time of all the algorithms against the different size of attack files. Graph shows that RabinKarp algorithm takes more time than all other algorithms whereas BoyerMoore is fastest among all selected algorithms.

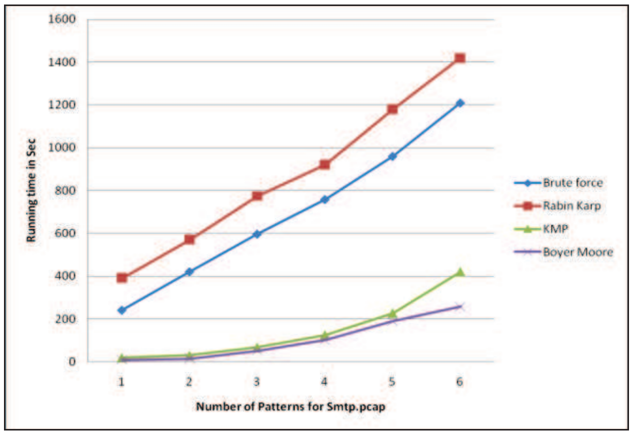


FIG 4. COMPARISON OF RUNNING TIME OF VARIOUS PATTERN MATCHING ALGORITHMS AGAINST NUMBER OF PATTERNS

Fig 4 shows another graph that is created by taking the running time of all the algorithms to find the attack against the different number of patterns to be searched. Graph shows that with the increasing numbers of patterns, the performance of BoyerMoore algorithm is also increases.

TABLE II. COMPARISON OF PATTERN MATCHING ALGORITHMS

Parameters	**Algorithms**			
	Brute-Force	***Rabin-Karp***	***KMP***	***Boyer-Moore***
Pre-processing	No	O (m)	O(m)	O(m+|∑|)
Search Type	Left to Right	Left to Right	Left to Right	Right to Left
Running Time	O(nm)	O (n+m)	O(n+m)	O(n/m)
Search Ideas	Search by matching all characters	Compares hash values of the text and the pattern	Constructs an automaton from the pattern	Bad-character and good-suffix heuristics to find valid shift
Approach	linear search	hashing based	heuristics based	heuristics based

V. CONCLUSION AND FUTURE SCOPE

This paper compares various pattern matching algorithms to observe their performance for intrusion detection systems and gives a view to choosing the efficient algorithms. After implementing the algorithms it has been analyzed that KMP and Boyer-Moore algorithms are efficient practically. Boyer-Moore algorithm is faster in case of large text and pattern. KMP reduces the time of searching as compared to brute force algorithm. Rabin-Karp algorithm whose time complexity is O(n+m) is not faster than naive search algorithm. Using a good hash function, Rabin-Karp can be efficient and it is easy to implement. Table II shows the comparative analysis of algorithms against various parameters. Exact pattern matching can solve many problems in computer science and to provide security to the systems. So any innovative concept in pattern matching algorithms can play an important role to get efficient performance in intrusion detection and prevention systems.

References

[1] S. Patil, P. Rane, Dr. B. B. Meshram, "IDS vs IPS", International Journal of Computer Networks and Wireless Communications, Vol. 2, No. 1, 2012.

[2] K. Scarfone, P. Mell, "Guide to Intrusion Detection and Prevention Systems(IDPS)" NIST Publication, 2007.

[3] J. Aldwairi, M. Monther, and D. Alansari. "Exscind: Fast pattern matching for intrusion detection using exclusion and inclusion filters." Next Generation Web Services Practices (NWeSP), 7th International Conference on. IEEE, 2011.

[4] B. Jakub, P. Buciak, and P. Sapiecha. "Building dependable intrusion prevention systems." Dependability of Computer Systems, International Conference on. IEEE, 2006.

[5] T.H. Corman, C.E. Leiserson, R.L. Rivest and C. Stein, "Introduction to Algorithms", The MIT Press, 2009, Cambridge, Massachusetts London, England.

[6] M. R. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms." IBM Journal of Research and Development, Vol. 31, No. 2, 1987.

[7] D.E. Knuth, J.H. Morris, and V.R. Pratt. "Fast pattern matching in strings." SIAM journal on computing 6.2,pp. 323-350,1977.

[8] R. S. Boyer and J. S. Moore. "A fast string searching algorithm." In Communications of the ACM, Vol. 20, pp. 762–772, October 1977.