International Journal of Future Generation Communication and Networking
Vol. 9, No. 9 (2016), pp. 169-180
http://dx.doi.org/10.14257/ijfgcn.2016.9.9.15

Single Digit Hash Boyer Moore Horspool Pattern Matching Algorithm for Intrusion Detection System

Sakshi Sharma and Manish Dixit

Department of CSE& IT
MITS
Gwalior, India
Sharmasakshi1009@gmail.com and dixitmits@gmail.com

Abstract

During the past time information on internet increasing enormously which greed the attacker and invite them for attack. In order to provide protection from illegal access the concept of Intrusion Detection System (IDS) is introduced. Intrusion detection system recognized as a powerful tool for identifying malicious attacks over the network. IDS works on the concept of string matching with the help of Detection engine. Detection engine of IDS uses String matching algorithm for comparing against malicious activities. This comparison takes enough processing time nearly 70% of the whole IDS processing time by improving performance of searching algorithm. In this paper, we proposed an enhanced version of Hash-Boyer-Moore-Horspool string matching algorithm by adding a single digit hash function for reduced the number of character comparisons and improve the efficiency of IDS by reducing the number of false positive match.

***Keywords**: IDS, String Matching Algorithm, Hash Function, Malicious, Performance Degradation, Pattern, Single Digit Hash Function.*

1. Introduction

Network security is a largely growing area of concern for every network. The continued growth of network traffic results in performance degradation. IDS is a concept which tries to detect unwanted network traffic and malicious content such as Port Scanning, Spoofing, Denial of Service attack due to these kind of activity's performance of IDS became challenging and important [1].

IDS works on the two Detection method- Anomaly based Detection and Signature based Detection. Anomaly based Detection finds unusual packet based on their behavioral log activities [2]. It must be useful for new generated attack, but sometimes it result in performance bottlenecks due to false alarm rates. Signature based detection finds attacks based on the old log files so sometimes it fails to recognize a newly generated attack for overcome this problem we need to update the rule policies time to time [11]. Signature based IDS uses Pattern Matching algorithm for Deep packet examination. Packet Matching Algorithm is the principal of IDS. It inspects each arriving packet against the predefined rule policies this process results in slow down the system and time consuming [12]. As day by day new attacks are created by the attacker therefore we need to update our rule policies for this purpose, necessity of String Matching Algorithm occurs which captures a maximum number of unwanted packets without the degradation of system performance and take less time. Several Pattern Matching Algorithms present in current scenario [3]. In this paper, we introduced a Single Digit Hash Boyer Moore Horspool Algorithm to improve the performance by reducing the number of character comparison and reduces false match.

2. Related Work

2.1. Boyer Moore Algorithm

Boyer Moore Algorithm begins with the rightmost character and track pattern from right to left. If there is a mismatch occurs, it uses two precalculated functions for shift the window right. Two shift functions are- Good Suffix Shift and Bad Character Shift, but it is difficult to analyze because the bad character shift is not very efficient for small alphabets [4][5].

2.2 Boyer Moore Horspool Algorithm (BMH)

BMH is an enhanced version of the Boyer Moore Algorithm with a small improvement. BMH uses only bad character shift table despite the fact, Boyer Moore uses two tables- Good Suffix Shift and Bad Character Shift. Its search the pattern from left to right and search values depends on the pattern size [9]. If the size of the pattern is larger therefore shift value is also larger, due to this improvement BMH performs in a faster way as compare to Boyer Moore. It has the low space complexity and easy to implement [6].

2.3. Hash Boyer Moore Horspool Algorithm (HBMH)

HBMH works on the basis idea of BMH with a little enhancement. It combined hash fingerprinting with BMH. The hash fingerprints are calculated in the same way the Karp-Rabin Algorithm [10]. This method reduces the number of character comparison at each time and reduces the comparison time, but there is still require some enhancement to improve false alarm rate and its characters comparison speed with new hash function [7].

3. Proposed Method

In this segment, we describe the procedure for improving the HBMH algorithm by adding a single digit hash function to work along with the shift table of the BMH algorithm. This hash function converts the string pattern into a numeric value. It can be calculated using the following hash function.

$$\mathrm{hash}[w(0_n)] = [w11[1]*(m)2+w12[1]*(m+1)2+w21[2]*(m+2)2+w22[2]*(m+3)2+\ldots +wn1[n]*(m+n-1)2+wn2[n]*(m+n)2]$$

Where m=1, n= last index number

The single digit hash function helps to reduce the number of character comparison in overall string as compare to previous used hash function thus, it saves the comparison time. Here, we take the example for explain the main idea behind using the single digit hash function for this algorithm.

Suppose we have the text (TSOSPOTRPOSTSTOPORV) and the pattern (POST). The HBMH starts by designing the Bad Characters Table bmBc by using given pattern just like used in BMH. Here, Bad character table used to hold the shifting value of each character. This gives the number of shift values, shifted by characters after each comparison executed between the text and pattern [8]. Consequently, the Algorithm is executed to find the match between the text and pattern. It matches the hash of the pattern from the hash generated for text, if the hash of text and pattern matches, then we match each character otherwise we skip the text. It reduces the number of comparison if there is mismatch so HBMH shift to the next comparison, according to the right most mismatches character which is (s=1) by using the Bad character shift table shown in fig.

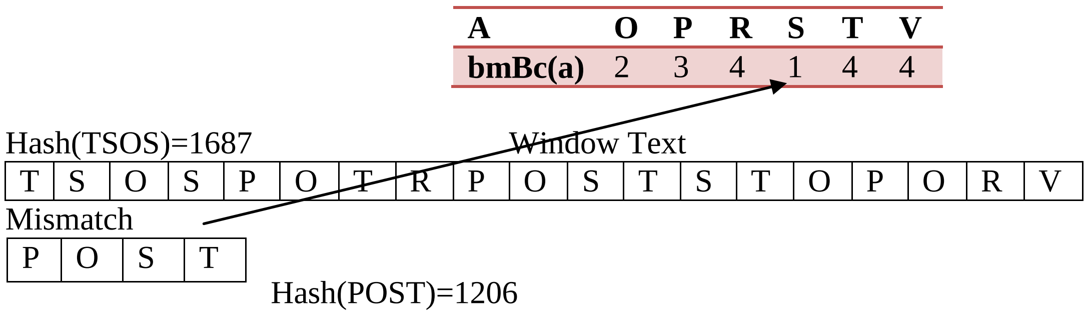


Figure 1. HBMH Matching Process for Text and Pattern

The algorithm starts the comparison of the hash of text(1687) against the hash of pattern(1206). As there is no matching between the hash, then algorithm move ahead without check the remaining text its move forward 1 character using the Bmbc table(s=1 shift value) again, it repeat the same procedure but hash of POTR(1206) match with the hash of POST(1206). Then it starts to compare characters of string which does not match. It results in false match and 4 comparisons. In the next step hash match occurs (POST against POST). This time true match occurs and 4 more comparisons counted. Same process runs consequently; in this case, Algorithm takes more time to check the false alarm and performs a large number of string matches with more time if the pattern size is large. As shown in figure.

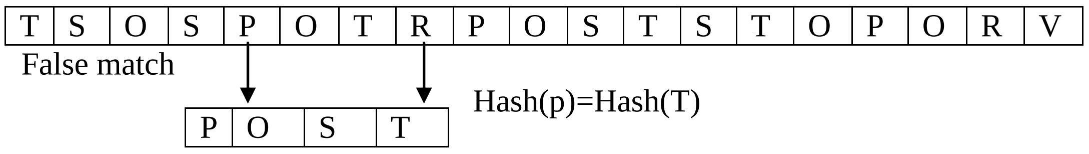


Figure 2. HBMH Matching Process for False Match (POTR)

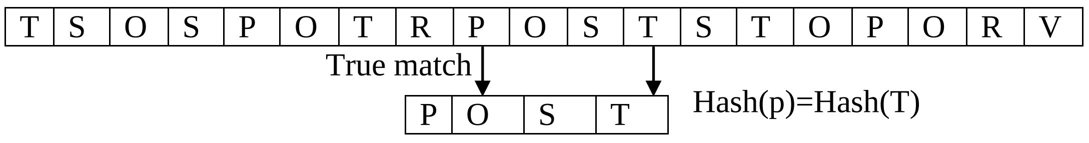


Figure 3. HBMH Matching Process for True Match (POST)

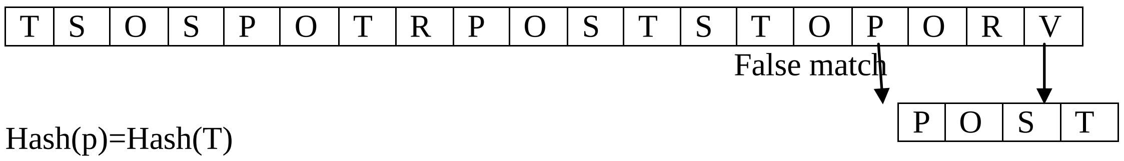


Figure 4. HBMH Matching Process for False Match (PORV)

From above example, we clearly found that there is wasted effort and time in each false comparison between the characters of the text (TSOSPOTRPOSTSTOPORV) and the pattern (POST).

There are a large number of incoming packets and rule policies used in IDS. It's a time consuming process for applying check on each incoming packet, so there is a need of the algorithm which performs check with minimum time and false alarm rate. For this purpose, we implement a single digit HBMH (SDHBMH) to overcome the drawback of HBMH algorithm. This single digit hash function algorithm converts each character of text and pattern into digit. By using an improved hash function which results in reduce number of character comparison and false alarm rate, which improve the performance of the IDS.

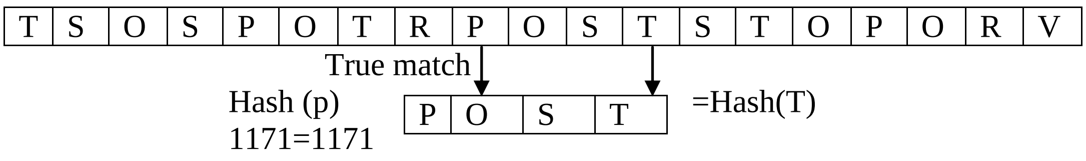


Figure 5. SDHBMH Matching Process for True Match

As shows in the above example, the hash of text (POST) is 1171 and the hash of pattern (POST) is 1171. So the hash of text and pattern are equal, then we compare the characters of the text and pattern. There are match of characters occurs. It reduced the number of false match as compared to HBMH which reduce the false alarm rate and results in performance improvement of IDS.

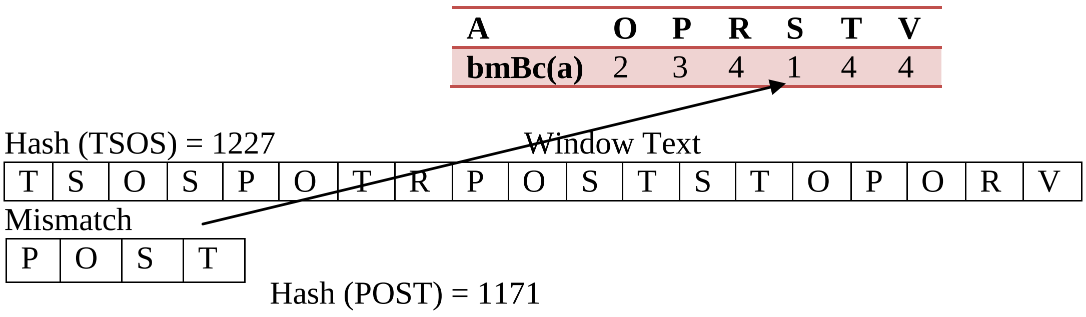


Figure 6. SDHBMH Matching Process for Text and Pattern

From above figure, we see that there are comparison between the window text (TSOS) and pattern (POST). The SDHBMH will shift the pattern by 1 position by using the value of S given in shift table. After that, SDHBMH will calculate the hash value for next text window (SOSP) and then comparison occurs between the text and pattern this process continue until the text window ended. In the given table we show the matching process comparison of HBMH and SDHBMH.

Table 1. Shows the Matching Comparison of HBMH and SDHBMH

Number of steps	Text data	Matching result for HBMH	Matching result for SDHBMH
1	TSOS	Not match	Not match
2	SOSP	Not match	Not match
3	POTR	False match	Not match
4	POST	Match	Match
5	STOP	False match	Not match
6	PORV	False match	Not match

3.1. Algorithm for SDHBMH Pattern Matching

```
Step 1: Capture the packet.
Step 2: Perform decoding on the packet.
Step 3: Applying Preprocessing on the packet.
Step 4: Take the text string and pattern string from the packet based on the rule and Computes
        the Single Digit Hash Function using function $hash[w(0_n)]$.
Step 5: Apply Boyer Moore Horspool Algorithm.
Step 6: If both hash value are not equal.
Step 7: Goto step 4
        Else
Step 8: Compare the String.
Step 9: If both string are not equal.
Step 10: Goto step 4
         Else
Step 11: Generate the Alert.
```

3.3 Flow Chart of Proposed Algorithm

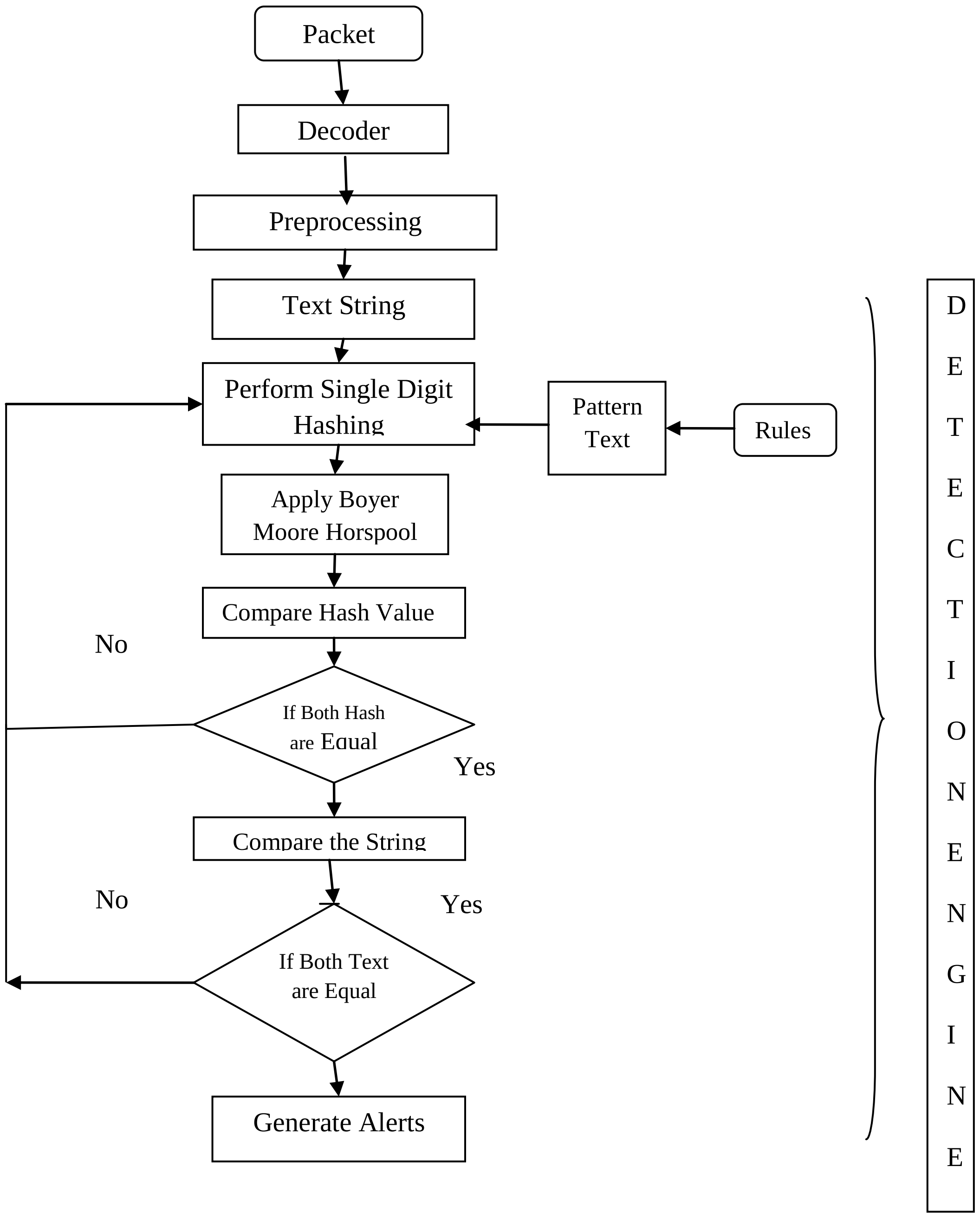


Figure 7. SDHBMH Flow Chart

Above flowchart is used to show the process flow algorithm in IDS aspect. Firstly decoder captures the packet from distinct network interfaces (like PPP, Ethernet etc.) and sends it for the preprocessing. Then preprocessor perform defragmentation on packets, categorization of data based on interfaces, decode HTTP URL after these processes preprocessor send the selected Text string to the Detection engine where we apply rule and give the pattern string in rule. Pattern string is the content which we want to find in our network data. After this we perform single digit hashing on the pattern and text data and apply Boyer Moore Horspool method for comparing the hash value of the text and pattern if both are matched, then we compare string character by character if both string are matched then generates the alert. And in the case of mismatching of hash value or string we move to the starting phase of detection engine where we move according to the Boyer Moore algorithm and again apply the hash function and repeat the process until whole data is detected.

3.2. Code Implementation

```
int hash (int exval[], int c)
  {       // exval [], is an array which contains extracted single digit from string.
   int i,j;   // int c, count of total extracted digit from string
   j=1;
   int sum=0;
   for (i=c-1;i>=0;i--)
       {
        sum= sum+(exval[i]* pow(j,2));
        j++;
       }
  return sum;
}
```

4. Implementation of SDHBMH

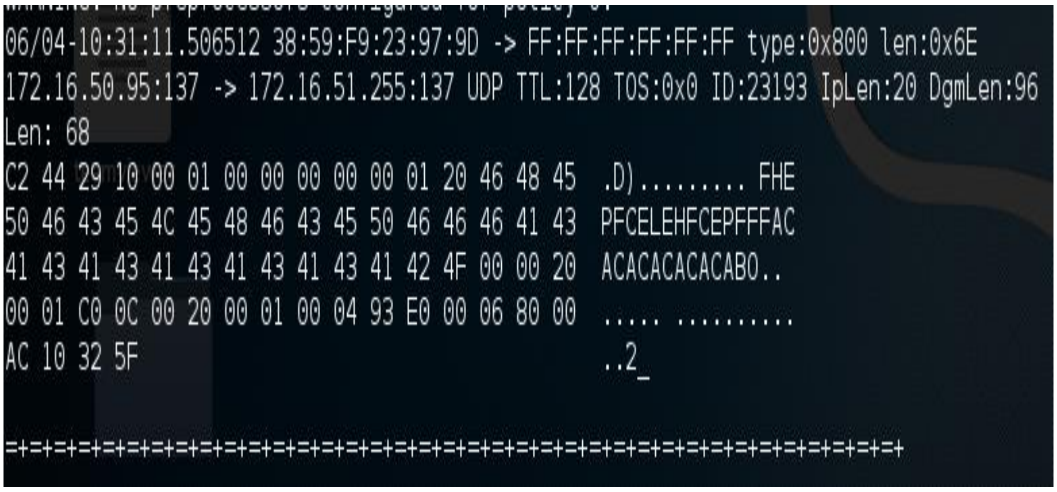
String matching algorithm is the heart of the ids. Its mainly consists of two phases; preprocessing phase and searching phase. SDHBMH also has these two phases [7].

Preprocessing phase
1. Change signature string to bytes in array form.
2. Create a 2-dimensional bmBc table.
3. Decides how to move a proper position in the search.
4. Values of table are fixed during the searching process.
5. Calculate the numeric value of text and pattern.

Searching phase
1. Calculate the hash function for value comparison.
2. Compare text from right to left at each point.
3. If a mismatched occurs.
4. Move to the next to the last character of the current text window for executing the next matching process.

4.1. Rule used in Snort for String Pattern Detection

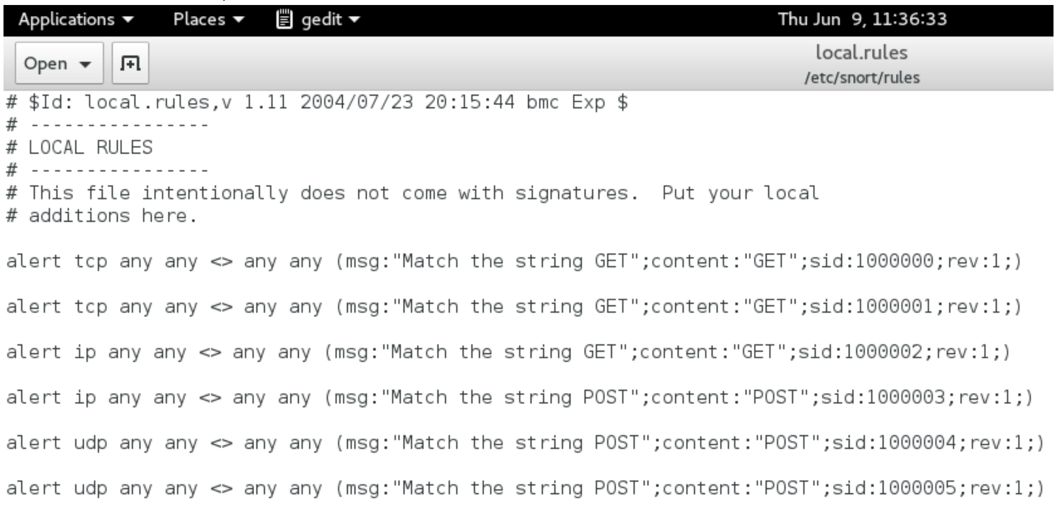
Given figure shows the incoming packet format in real world. We perform analysis on this kind of captured packet format.



```
06/04-10:31:11.506512 38:59:F9:23:97:9D -> FF:FF:FF:FF:FF:FF type:0x800 len:0x6E
172.16.50.95:137 -> 172.16.51.255:137 UDP TTL:128 TOS:0x0 ID:23193 IpLen:20 DgmLen:96
Len: 68
C2 44 29 10 00 01 00 00 00 00 00 01 20 46 48 45  .D)......... FHE
50 46 43 45 4C 45 48 46 43 45 50 46 46 46 41 43  PFCELEHFCEPFFFAC
41 43 41 43 41 43 41 43 41 43 41 42 4F 00 00 20  ACACACACACABO..
00 01 C0 0C 00 20 00 01 00 04 93 E0 00 06 80 00  ..... .........
AC 10 32 5F                                      ..2_

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Figure 8. Capture Packet Format in Snort

alert tcp $HOME_NET any <> $EXTERNAL_NET any (msg:"Match the string"; flow:from_client; content:"POST"; http_method; classtype:string-detect; priority:10; sid:1000000; rev:1;)

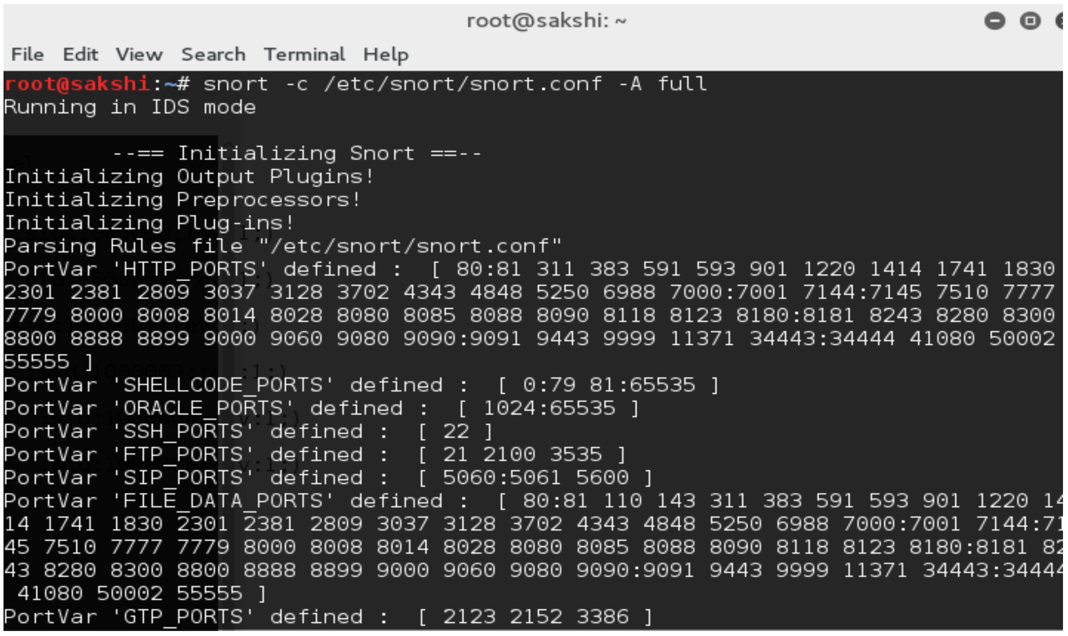


```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ----------------
# LOCAL RULES
# ----------------
# This file intentionally does not come with signatures.  Put your local
# additions here.

alert tcp any any <> any any (msg:"Match the string GET";content:"GET";sid:1000000;rev:1;)

alert tcp any any <> any any (msg:"Match the string GET";content:"GET";sid:1000001;rev:1;)

alert ip any any <> any any (msg:"Match the string GET";content:"GET";sid:1000002;rev:1;)

alert ip any any <> any any (msg:"Match the string POST";content:"POST";sid:1000003;rev:1;)

alert udp any any <> any any (msg:"Match the string POST";content:"POST";sid:1000004;rev:1;)

alert udp any any <> any any (msg:"Match the string POST";content:"POST";sid:1000005;rev:1;)
```

Figure 9. Rules File in Snort

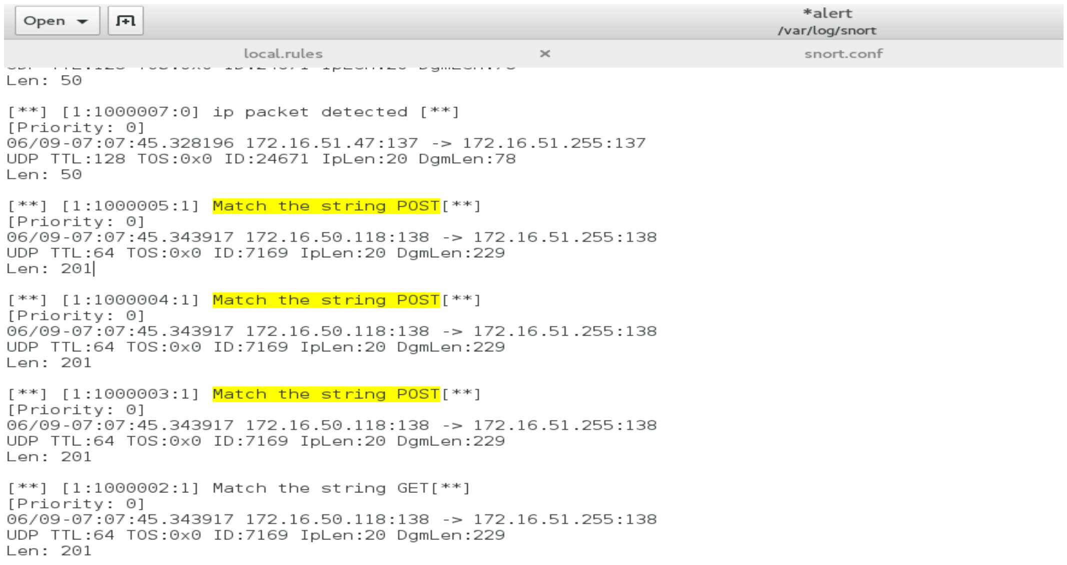
The above rule is used to detect the string in snort packet, and generates the output which shown below.

Then these rules are running in IDS mode, as shown in given figures. Which capture the whole network traffic and generates the alert file for the matched string.



```
root@sakshi:~# snort -c /etc/snort/snort.conf -A full
Running in IDS mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file "/etc/snort/snort.conf"
PortVar 'HTTP_PORTS' defined :  [ 80:81 311 383 591 593 901 1220 1414 1741 1830
2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7000:7001 7144:7145 7510 7777
7779 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180:8181 8243 8280 8300
8800 8888 8899 9000 9060 9080 9090:9091 9443 9999 11371 34443:34444 41080 50002
55555 ]
PortVar 'SHELLCODE_PORTS' defined :  [ 0:79 81:65535 ]
PortVar 'ORACLE_PORTS' defined :  [ 1024:65535 ]
PortVar 'SSH_PORTS' defined :  [ 22 ]
PortVar 'FTP_PORTS' defined :  [ 21 2100 3535 ]
PortVar 'SIP_PORTS' defined :  [ 5060:5061 5600 ]
PortVar 'FILE_DATA_PORTS' defined :  [ 80:81 110 143 311 383 591 593 901 1220 14
14 1741 1830 2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7000:7001 7144:71
45 7510 7777 7779 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180:8181 82
43 8280 8300 8800 8888 8899 9000 9060 9080 9090:9091 9443 9999 11371 34443:34444
 41080 50002 55555 ]
PortVar 'GTP_PORTS' defined :  [ 2123 2152 3386 ]
```

Figure 10. Snort Run in IDS Mode for Packet Capturing



```
Len: 50

[**] [1:1000007:0] ip packet detected [**]
[Priority: 0]
06/09-07:07:45.328196 172.16.51.47:137 -> 172.16.51.255:137
UDP TTL:128 TOS:0x0 ID:24671 IpLen:20 DgmLen:78
Len: 50

[**] [1:1000005:1] Match the string POST[**]
[Priority: 0]
06/09-07:07:45.343917 172.16.50.118:138 -> 172.16.51.255:138
UDP TTL:64 TOS:0x0 ID:7169 IpLen:20 DgmLen:229
Len: 201

[**] [1:1000004:1] Match the string POST[**]
[Priority: 0]
06/09-07:07:45.343917 172.16.50.118:138 -> 172.16.51.255:138
UDP TTL:64 TOS:0x0 ID:7169 IpLen:20 DgmLen:229
Len: 201

[**] [1:1000003:1] Match the string POST[**]
[Priority: 0]
06/09-07:07:45.343917 172.16.50.118:138 -> 172.16.51.255:138
UDP TTL:64 TOS:0x0 ID:7169 IpLen:20 DgmLen:229
Len: 201

[**] [1:1000002:1] Match the string GET[**]
[Priority: 0]
06/09-07:07:45.343917 172.16.50.118:138 -> 172.16.51.255:138
UDP TTL:64 TOS:0x0 ID:7169 IpLen:20 DgmLen:229
Len: 201
```

Figure 11. Alert File Generation

5. Result Analysis

As we shown in the given example. If we apply HBMH and SDHBMH algorithm on (TSOSPOTRPOSTSTOPORV). We notice that the performance of SDHBMH is better as Compare to HBMH. SDHBMH reduces the number of comparison and reduce the false alarm rate with minimum time. There are 12 comparisons using the HBMH and SDHBMH has only 4 comparisons. This shows that SDHBMH improve the performance of network security applications such as IDS in a faster way.

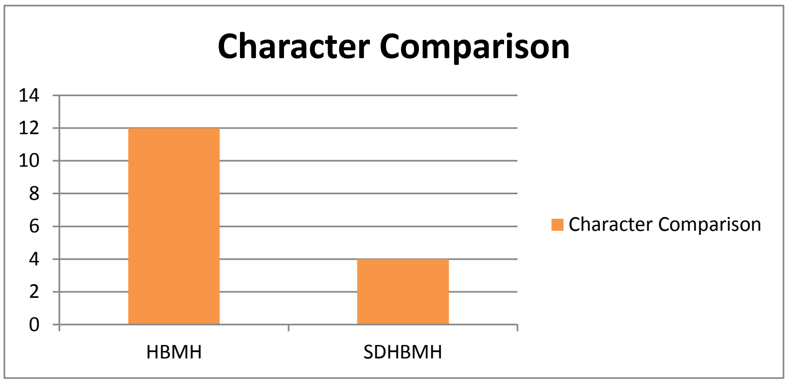


Figure 12. Performance Comparison of HBMH and SDHBMH

6. Conclusion

As the use of internet growing constantly and traffic on the network increases tremendously and uses network security systems so there is need of developing an efficient and fast string matching algorithm arises for finding the malicious patterns in the network traffic. Internet is the main need of every organization so security of the internet is also a major issue. We can solve this issue by using IDS. IDS analyze the incoming packets against the malicious activities. It uses pattern matching algorithm for this purpose and apply rule policies. In this work, we have suggested a new approach for string matching which improve the IDS performance and efficiency by reducing the character comparison time and also minimize the false match. Proposed work gives a single digit hash function which results in performance improvement over the HBMH. SDHBMH execution is faster than the HBMH algorithm. SDHBMH is good for IDS because it provides a faster method for packet content comparison in real world with minimum wrong alert.

References

[1] Mahdinia, Payam, Mehdi Berenjkoob, and Hedayat Vatankhah. "Attack signature matching using graphics processors in high-performance intrusion detection systems." In Electrical Engineering (ICEE), 2013 21st Iranian Conference on, pp. 1-7. IEEE, 2013.
[2] Hasan, Awsan A. "Multi-Pattern Boyer-Moore-Horspool Algorithm based Hashing Function for Intrusion Detection System." Lecture Notes on Information Theory Vol 1, no. 2 (2013).
[3] Gupta, Vibha. "Pattern Matching Algorithms for Intrusion Detection and Prevention Systems." PhD diss., Thapar University Patiala, 2014.
[4] Zhao, Dongcan, Xiaomin Zhu, and Tong Xu. "Improvement of algorithm for pattern matching in intrusion detection." In Broadband Network & Multimedia Technology (IC-BNMT), 2013 5th IEEE International Conference on, pp. 281-284. IEEE, 2013.
[5] Boyer, Robert S., and J. Strother Moore. "A fast string searching algorithm."Communications of the ACM 20, no. 10 (1977): 762-772.

[6] Horspool, R. Nigel. "Practical fast searching in strings." Software: Practice and Experience 10, no. 6 (1980): 501-506.
[7] Awsan Abdulrahman Hasan and Nur Aini Abdul Rashid " Hash - Boyer-Moore - Horspool String Matching Algorithm for Intrusion Detection System." IPCSIT vol.35(2012) , Singapore.
[8] Qiao, Jiaxing, and Hua Zhang. "Improvement of BM algorithm in intrusion detection system." In Software Engineering and Service Science (ICSESS), 2015 6th IEEE International Conference on, pp. 652-655. IEEE, 2015.
[9] Charras, Christian, and Thierry Lecroq. Handbook of exact string matching algorithms. King's College, 2004.
[10] Karp, Richard M., and Michael O. Rabin. "Efficient randomized pattern-matching algorithms." IBM Journal of Research and Development 31, no. 2 (1987): 249-260.
[11] Sharma, Sakshi, and Manish Dixit. "A Review on Network Intrusion Detection System Using Open Source Snort." International Journal of Database Theory and Application 9, no. 4 (2016): 61-70
[12] Kelly, James. "An Examination of Pattern Matching Algorithms for Intrusion Detection Systems." PhD diss., Carleton University Ottawa, 2006.