Volume: 10 Issue: 02 Pages: 3776- 3781 (2018) ISSN: 0975-0290

Performance Evaluation of Different Pattern Matching Algorithms of Snort

Abhigya Mahajan
M.Tech. Student
Department of Computer Science & IT, University of Jammu, J & K, India
Email: abhigya121ma@yahoo.com
Alka Gupta
Research Scholar
Department of Computer Science & IT, University of Jammu, J & K, India
Email: alkagupta48@gmail.com
Lalit Sen Sharma
Professor
Department of Computer Science & IT, University of Jammu, J & K, India
Email: lalitsen.sharma@gmail.com

ABSTRACT

Snort is the most widely deployed Network Intrusion Detection System (NIDS) whose performance is dominated by the pattern matching of packets in the network. In this paper, we present an experimental evaluation and comparison of the performance of different pattern matching algorithms of Snort NIDS namely ac-q, ac-bnfa, ac-split, ac-banded and ac-sparsebands on Linux Operating System (Ubuntu Server 16.04). Snort's performance is measured by subjecting the server running Snort v2.9.9.1 to live malicious traffic and a standard dataset. The performance is calculated and compared in terms of throughput, memory utilization and CPU utilization.

Keywords - **Bnfa, D-ITG, NIDS, pattern-matching, Scapy, Snort, sparsebands**.

Date of Submission: July 10, 2018 Date of Acceptance: Aug 08, 2018

1. Introduction

1.1 Snort

Snort is an open source Network Intrusion Detection System (IDS) which has the ability to perform real-time traffic analysis and packet logging on Internet Protocol (IP) networks. It performs protocol analysis, content searching and matching. It can be configured in three main modes: sniffer, packet logger, and network intrusion detection. In sniffer mode, the system reads network packets and displays them on the console. In packet logger mode, it logs the packets to the disk whereas, in intrusion detection mode, the system monitors network traffic and analyzes it against a rule set defined by the security analyst. The program will then perform a specific action based on what has been identified. It uses a rule-driven language which combines the benefits of signature, protocol and anomaly based inspection methods.
Snort can be implemented as Host based IDS, that monitors a computer system on which it is installed to detect an intrusion or as Network based IDS, that scans network packets to detect an intrusion and/or misuse.
Snort implements Deep Packet Inspection (DPI) which inspects both packet headers and payloads to trace out and put off suspicious attacks. Created by Martin Roesch in 1998, it is a free and open source, rule based, network intrusion detection system (IDS) and network intrusion prevention system (IPS) [1]. It combines the benefits of signature, protocol and anomaly based inspection methods. With more than 3 million of downloads to date, *Snort* is the most widely deployed intrusion detection and prevention technology worldwide[2].

1.2 Architecture of Snort

Snort is a core and plug-ins system, with sniffer as the core component and the different available plug-ins are pre-processors, the detection engine and the output module. Fig. 1 represents the architecture of *Snort*. The sniffer collects the network traffic and identifies the packet structure. Packet then goes to the pre-processors which modify the data packets for faster detection by the detection engine. It also examines the behaviour of packet and can generate alert if an anomaly is found. The detection engine is responsible for rule-matching. It takes the pre-processed packet and compares with a set of available rules. If the rules match the data in the packet, alert is generated via output module.

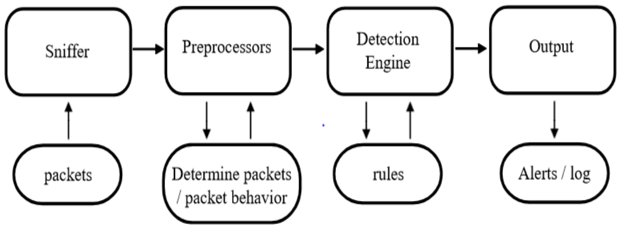

Fig.1. Architecture of *Snort*

1.2 Rule Syntax in Snort

The rule syntax in *Snort* is well-organized. Fig. 2 represents the rule syntax in *Snort*. The each rule consists of two parts:

- Rule Header: Rule header has a static definition and is composed of the 5 tuple as shown in Fig. 2. It must be present in every rule.
- Rule options: Rule options have variable definitions. It is not always present, and it has more than 50 options available to account for different requirements in the description of possible rules.

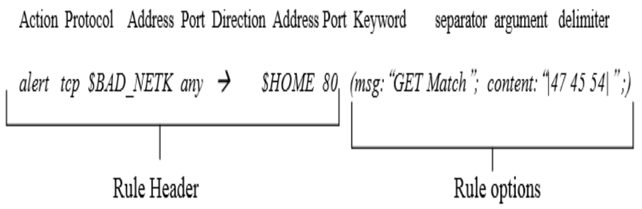
Action Protocol Address Port Direction Address Port Keyword separator argument delimiter
alert tcp $BAD_NETK any → $HOME 80 (msg: "GET Match"; content: "|47 45 54|";)
Rule Header
Rule options

Fig. 2. Rule Syntax in *Snort*

1.4 Algorithms

The pattern matching is the most computationally expensive test that *Snort* commonly performs on packets[3]. The basic pattern matching task is to match a number of patterns drawn from the *Snort* rules to each packet. The content field specifies a string to match in the packet. The number of patterns can be in the order of a few thousands. *Snort* version 2.9 contains over 9000 patterns. The following are the efficient and high-speed pattern matching algorithms which can match multiple patterns simultaneously.

- *Ac-q* is an optimized Aho-Corasick algorithm that uses a full matrix state transition table[2].
- *Ac-bnfa* is an optimized Aho-Corasick algorithm that uses binary NFA.
- *Ac-split* is an optimized Aho-Corasick algorithm which dynamically divides the pattern into blocks for efficient rule matching[4].
- *Ac-banded* is an optimized Aho-Corasick algorithm that uses a banded matrix state transition table.
- *Ac-sparsebands* is an optimized Aho-Corasick algorithm that uses a sparse-banded matrix state transition table.

The major contribution of this paper is evaluating experimentally and comparing the performance of different pattern matching algorithms of *Snort* under the Linux (*Ubuntu* Server 16.04). This will help in selecting the best pattern matching algorithm to boost and improve *Snort*'s performance. Our evaluation involves analysing the performance of different algorithms based on the parameters like CPU utilization, Memory utilization and throughput of *Snort*.

The rest of the paper is organized as follows. Section 2 summarizes the past work done in the same field. Section 3 describes the experimental setup and parameters used to carry out the research. Section 4 gives performance evaluation of the experiment.

2. Related Work

Sarang Dharmapurikar and John Lockwood in [5] presented a high-speed and scalable multi-pattern matching algorithm for Network Intrusion Detection Systems. They used hardware-based Bloom filters, which suppressed a large fraction of memory accesses and speeded up string matching.

Sarika Rameshwar Rathi in [6] presented a new framework for IDS. They used the Darpa Dataset and applied Apriori algorithm for intrusion detection which reduced its complexity and provided a high detection rate. Yaron Weinsberg et al. in [7] designed and implemented a full NIPS system based on a novel pattern matching algorithm, called RTCAM. Their solution achieved line-speed rates. For about 60% of real network traffic, an average line speed of 12.35 Gbps was achieved. Their system was fully compatible with *Snort*'s rules syntax which was capable of importing *Snort*'s database easily.

R. Hamsaveni and Dr. G. Gunasekaran in [8] identified the number of promising algorithms and provided an overview of recent developments in the single keyword pattern matching for IDS. They compared Boyer-Moore Algorithm which uses two tables and matching starts with right to left, with Horspool which uses only one table and the matching is faster than the BoyerMoore. They also analysed Brute force algorithm, Knuth -Morris -Pratt algorithm and Karp – Rabin algorithm. They proposed LPM algorithm was compared with the exiting algorithms and the result showed that the algorithm was faster and more reliable in network security applications.

Qing-Xiu Wu in [9] studied network protocol analysis technique in *Snort*. The technique analyzed all header fields of various layers of the network as well as the data content of application layer. This information was made available to users to understand the full range of network packet information and to carry out in-depth analysis of network attacks.

Huang Kun and Zhang Da Fang in [10] proposed the ISBF to achieve line-speed packet processing and reduce the memory requirements. They also proposed the lazy deletion algorithm and vacant insertion algorithm to reduce the update overhead of the ISBF and compared it with the TriBiCa and SFHT. Experimental results showed that compared to the TriBiCa and SFHT, the ISBF reduced the offchip memory accesses of insertion by 12.5 times and 4.6 times, deletion by 6 times and 5.8 times, respectively. The ISBF reduced the processing time of insertion by 45.9 times and 89 times, deletion by 10.4 times and 76.2 times respectively.

Vasudha Bhardwaj and Vikram Garg in [11] introduced a new algorithm EWM for multi pattern exact matching which proved to be an efficient version of the WM algorithm in both respect (time and memory). The algorithm used two shift table; one for single character and another for block of character. While in hash table they used AVL tree instead of linear data structure.

C. Jason Coit et al. in [12] implemented a "Boyer-Moore approach to exact set-matching" works well in practice to significantly reduce the computation time needed to match packet content to the ever-increasing rule set. They demonstrated that their version of *Snort* operates 1.02 times up to 3.32 times as fast as the current version depending on the number and type of content rules and the day to day speedup will depend on the network traffic and rule set used. Their algorithm worked efficiently in terms of time but used roughly 3 times the memory used by the standard version. They proposed the idea that replacing the current transition matrix of each node with splay tree will reduce the space needed for each node for an increase in running time.

Soumya Sen in [2] had studied the effect of varying packet size and rule set size on *Snort*'s performance and had suggested methods to reduce the dependency of *Snort*'s performance on increase in rule-sets.She proposed the use of sparse banded matrix data structure for implementing pattern-matching in Aho-Corasick algorithm of detection engine to increase both memory as well as time efficiency of the system. The reduction in memory requirements was intended to avoid cache misses and to make room for ever-growing number of rule-sets, which in turn would increase *Snort*'s performance.

N. Khamphakdee et al. in [13]described about Intrusion Detection System Based on *Snort* Rules for Network Probe Attack Detection. The MIT-DARPA 1999 dataset is tested and evaluated by MIT Lincoln Laboratory. The dataset consists of normal and abnormal connection which was recorded in many file formats. MIT-DARPA 1999 dataset has total 656 probe attacks. *Snort* IDSs can generate thousands of alarms in a day that flood network administrators. Many of these alarms are usually considered to be so called false alarms. As a result, network administrators run the risk of missing good alarms lost in the noise generated by the false alarms.

3. Experiment Plan

The aim of the experiment is to measure the performance exhibited by *Snort* under high traffic. *Snort* version 2.9.9.1 has been installed as a host based NIDS with the default configurations. We set the output methods for both alerting and logging with writing output to files located in the default log directory. Two experiments have been performed to evaluate the performance of *Snort*. In first experiment, live malicious traffic is send to the server with *Snort* and its performance is evaluated for all the five algorithms. In second experiment, the performance is calculated and compared for the *CDX-2009* dataset, which is a labled dataset captured by NSA with an aim to correlate IP addresses found in the PCAP files with the IP addresses to hosts on the internal USMA network [14].

3.1 Experimental set-up

The network setup comprises of four machines, which includes three attackers/traffic generators and a server, connected through a 1 Gbps Ethernet network. The receiver works on Linux operating system (*Ubuntu* server 16.04) and the senders are *Ubuntu* desktop 16.04 machines. Traffic is generated at the sender machines using three *Scapy* and three *D-ITG* (Distributed Internet Traffic Generator). *Scapy* is a powerful tool for packet manipulation written by Philippe Biondi in python[15], which is used to create and send custom packets of a wide number of protocols over the network. *D-ITG* is an open source platform capable to generate both IPv4 and IPv6 traffic at network layer, transport layer and application layer. ITGSend is the component responsible for generating traffic at packet level towards ITGRecv in *D-ITG*[16]. The setup shown in Fig. 3. is used for conducting Test 1.

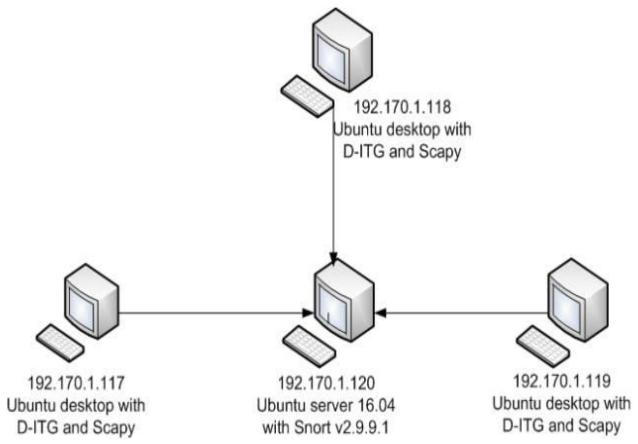

Fig.3. Experimental setup

3.2 Evaluation parameters

The following parameters are used for the evaluation of different pattern matching algorithms.

1. Throughput :- It is the total number of packets analysed in a given period of time.

   Throughput = $\frac{total\ number\ of\ packets\ analysed}{total\ time\ taken}$

   It is measured in Kpps(kilo Packets Per Seconds).

2. CPU Utilization :- CPU utilization refers to the processor utilization for *Snort*.

3. Memory Utilization :- Memory utilization refers to the total amount of memory utilised by *Snort* process.

4. Performance Measurements

The performance of Different pattern matching algorithms that are *ac-q*, *ac-bnfa*, *ac-banded* and *ac-sparsebands* are evaluated against the default algorithm that is *ac-split* in terms of throughput, memory utilization and CPU utilization.

4.1 Test 1: Snort's performance against real & malicious traffic

To compare the performance of different pattern matching algorithms of *Snort* under heavy traffic, two new rules have been added to *Snort*'s default rule-set. Packets of size 1024 bytes at 1 Gbps were send to the server for 120 seconds to study the behaviour of *Snort*.

The rule shown below check every incoming TCP packet for a payload containing the strings "malicious" and "bad_traffic". When a match occurs, a message "malicious data detected" and "bad data detected" is generated to the alert file stored in the default log directory with an identity of "1000001" and "1000002" respectively. The rules are inserted to a file called local.rules. The format of the rule is as follows:

```
alert tcp any any -> $HOME_Net any (msg : "malicious data
detected"; content : "malocious"; sid : 1000001; rev : 01)

alert tcp any any -> $HOME_Net any (msg : "bad data
detected"; content : "bad_traffic"; sid : 1000002; rev : 01)
```

The following commands is used in *Scapy* to generate traffic at the sender's end :

```
send(IP(dst="190.170.1.120")/TCP()/"bad_traffic", loop=1)
send(IP(dst="190.170.1.120")/TCP()/"malicious", loop=1)
```

On the basis of Memory utilization, CPU utilization, and Throughput, performance of different pattern matching algorithms of *Snort* against live normal and malicious traffic has been analyzed. Table 1. depicts the experimental values of different algorithms.

Algorithms/ Parameters	Memory	CPU	Throughput
ac-split	25.9	57.825	40.01804
ac-q	42	73.09	42.22628
ac-bnfa	7.1	69.38	41.56937
ac-banded	50.8	74.95	40.46659
ac-sparsebands	40.3	90.4	41.42213

Table 1. Experiment's result of Test 1

It has been analyzed that *ac-bnfa* algorithm has the best performance in terms of memory utilization as shown in Fig. 4. *Ac-split* and *ac-bnfa* gives better results than *ac-q*, *ac-banded* and *ac-sparsebands* in terms of CPU utilization as shown in Fig. 5. And in terms of throughput, *ac-q* has the best performance as shown in Fig. in 6.

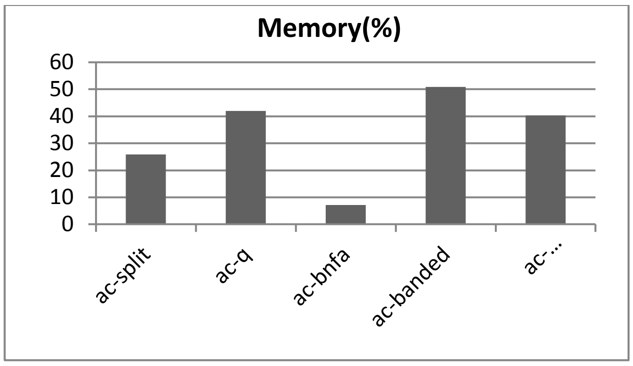

Fig. 4. Comparative analysis of algorithms in terms of Memory for Test 1

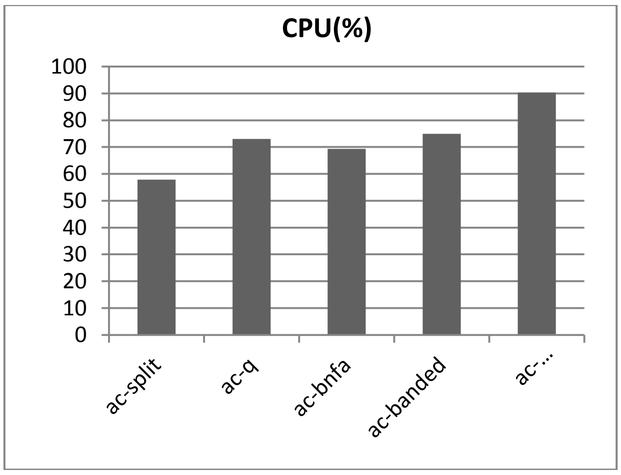

Fig. 5. Comparative analysis of algorithms in terms of CPU for Test 1

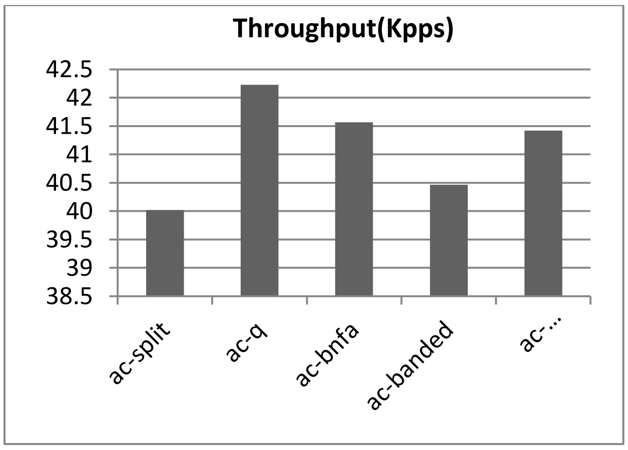

Fig. 6. Comparative analysis of algorithms in terms of throughput for Test 1

4.2 Snort's performance against CDX Dataset

To measure and evaluate the performance of *Snort* against *CDX-2009* dataset, the following read command is used to read the dataset and the results are measured for different pattern matching algorithms.

```
Snort –c /etc/Snort/Snort.conf - - pcap – dir /home/dataset
–A console
```

The performance of different pattern matching algorithms of *Snort* have also been measured and compared against *CDX-2009* dataset in terms of Memory and CPU utilization, and Throughput. Table 2. depicts the experimental result of different algorithms.

Algorithms/ Parameters	Memory	CPU	Throughput
ac-split	27.73	96.2	21.044
ac-q	52.4	94.29	20.175
ac-bnfa	22.06	97.21	19.071
ac-banded	77.7	50.05	8.083
ac-sparsebands	77.83	50.05	8.813

Table 2. Experiment's result of Test 2

In Fig. 7. it has been analyzed that *ac-bnfa* gives the best result in terms of memory utilization. In terms of CPU utilization both the algorithms *ac-banded* and *ac-sparsebands* gives better results as compared to *ac-split*, *ac-q* and *ac-bnfa* as shown in Fig. 8. And in case of throughput *ac-split*, *ac-q* and *ac-bnfa* gives the best results as shown in Fig. 9.

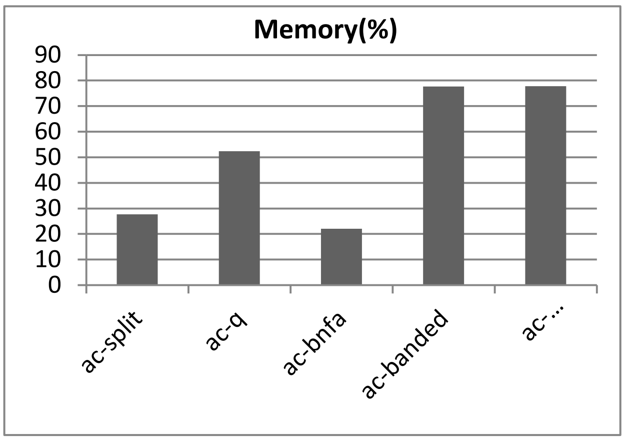

Fig. 7. Comparative analysis of algorithms in terms of memory for Test 2

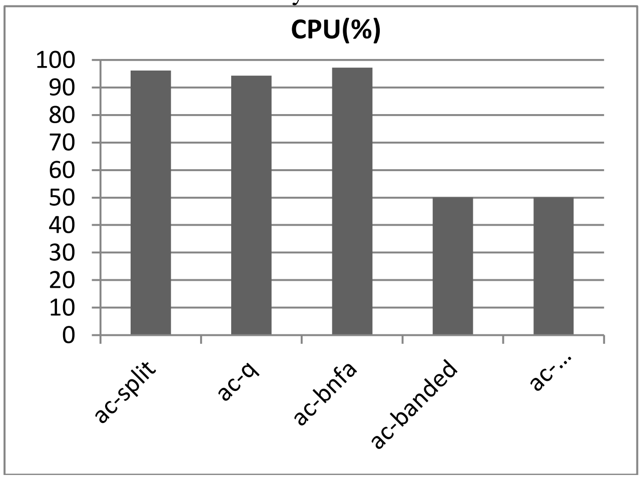

Fig. 8. Comparative analysis of algorithms in terms of CPU for Test 2

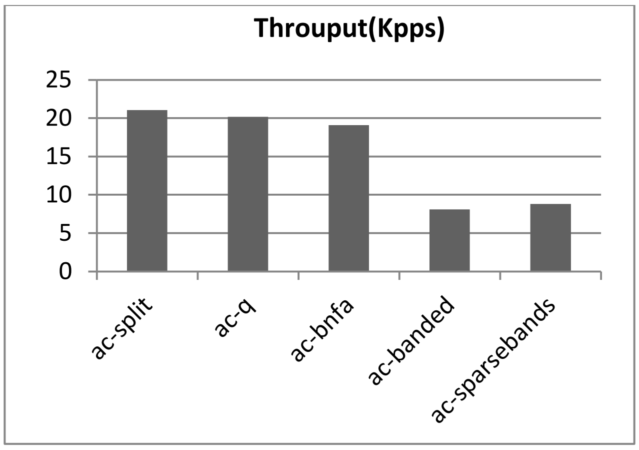

Fig.9. Comparative analysis of algorithms in terms of throughput for Test 2

5. Conclusion

In this paper, we have evaluated and compared the performance of different pattern matching algorithms of *Snort* NIDS on *Ubuntu* server 16.04. The performance was measured and compared in terms of throughput, CPU and Memory utilization by subjecting the server running Snort to both malicious traffic and *CDX-2009* dataset. It has been experimentally found that Snort with *ac-bnfa* algorithm gives the best results in terms of memory utilization for both the test cases as compared to other algorithms. It has also been observed that in terms of throughput *ac-split*, *ac-q* and *ac-bnfa* gives the best results in case of *CDX-2009* dataset and in case of real and malicious traffic, *ac-q* gives the best results. In terms of CPU utilization, *ac-bnfa* and *ac-split* have shown better results than other algorithms in case of real and malicious traffic, but in case of *CDX-2009* dataset, *ac-banded* and *ac-sparsebands* outperforms other algorithms.

6. References

[1].https://Snort-org-site.s3.amazonaws.com

[2].Soumya Sen, "Performance Characterization & Improvement of Snort as an IDS," Bell Labs Report, 2006.

[3].Martin Roesch, "Snort - lightweight intrusion detection for networks," in Proceedings of the 13th Systems Administration Conference. 1999, USENIX.

[4].https://s3.amazonaws.com/Snort-org-site/production/document_files/files/000/000/122/original/Snort_2.9.9.x

[5].Sarang Dharmapurikar and John Lockwood, "Fast and Scalable Pattern Matching for Network Intrusion Detection Systems", in *IEEE Journal on Selectedd Areas in Communications*, vol. 24, no. 10, pp. 1781 - 1792, 2006.

[6].Sarika Rameshwar Rathi, "Detecting Attack Packets by Using Darpa Dataset on Intrusion Detection System" in *International Journal Of Engineering And Computer Science, International Journal Of Engineering And Computer Science*, vol. 4, no. 2, pp. 10567-10569, 2015.

[7].Yaron Weinsberg, Shimrit Tzur-David, Danny Dolev and Tal Anker High, "Performance String Matching Algorithm for a Network Intrusion Prevention System (NIPS)", in *IEEE workshop on High Performance of Switching and Routing*, 2006.
[8].R. Hamsaveni and Dr. G. Gunasekaran, "A Secured Pattern Matching Technique for Intrusion Detection System in Wireless Sensor Network", in *International Journal of Computer Networks and Wireless Communications*, vol. 6, no. 3, pp. 34-41, 2016.
[9].Qing-Xiu Wu, "The Network Protocol Analysis Technique in Snort", in *ELSEVIER on InternationalConference on Solid State Devices and Materials Science*, 2012.
[10]. Huang Kun and Zhang DaFang, "An index-split Bloom filter for deep packet inspection", in *ACM Journal on Science China Information Sciences*, vol. 54, no. 1, pp. 23-27, 2011.
[11]. Vasudha Bhardwaj and Vikram Garg, "Efficient Wu Manber String Matching Algorithm for Large Number of Patterns", in *International Journal of Computer Applications*, vol. 132, no. 17, pp. 29-33, 2015.
[12]. Christopher V. Kopek, Errin W. Fulp and Patrick S. Wheeler, "Distributed Data Parallel Techniques for Content-Matching Intrusion Detection Systems", in *proc. of IEEE on Military Communications Conference*, 2007.
[13]. N. Khamphakdee, N. Benjamas and S. Saiyod" Improving Intrusion Detection System Based on *Snort* Rules for Network Probe Attack Detection", in *IEEE on 2nd International Conference on Information and Communication Technology (ICoICT)*, 2014.
[14]. https://www.usma.edu/crc/SitePages/DataSets.aspx
[15]. https://*Scapy*.net
[16]. http://www.grid.unina.it/software/ITG

Authors Biography

Abhigya Mahajan is a student of Master of Technology (MTech) in Computer Science in the Department of Computer Science & IT, University of Jammu, Jammu & Kashmir, India. He has received his Bachelor's of Engineering (B.E) degree in the discipline of Computers from University of Jammu. His research interests are in the field of Data Structures and Network Security.

Alka Gupta is a Ph.D. scholar in the Department of Computer Science and IT, University of Jammu. She has received her M.Tech degree in Computer Science from Shri Mata Vaishno Devi University, Katra, Jammu & Kashmir, India. She has done her B.E. in Computer Science and Engineering from Government College of Engineering and Technology, Jammu, Jammu & Kashmir, India. She also holds a teaching experience of 3 years. Her research interests are network security, Data structures and computer graphics.

Lalit Sen Sharma received his doctorate in Computer Science and Engineering from Guru Nanak Dev University Amritsar, Punjab, India. He also holds master's degree in Mathematics and Computer Applications from the same university. He has been teaching to post graduate students in computer applications of University of Jammu for more than 20 years. He is a life member of India Science Congress Association, Computer Society of India, Institute of Electronics and Communication Engineers and National HRD Network, India. He is specialized in Data Communication and Network, Internet and WWW and Data Structures. He has completed one major research project to trace out vulnerabilities in network applications funded by University Grants Commission, Ministry of Human Resource Development, Govt. of India. He has produced one PhD and is currently supervising Five PhD scholars.