International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-4, April 2019 E-ISSN: 2347-2693

Mitigation of *DoS* and *Port Scan* Attacks Using *Snort*

Alka Gupta[1*], Lalit Sen Sharma[2]

[1,2]Department of Computer Science and IT, University of Jammu, Jammu, India

Corresponding Author: alkagupta48@gmail.com, Tel.: +91-9419148180

DOI: https://doi.org/10.26438/ijcse/v7i4.248258 | Available online at: www.ijcseonline.org

Accepted: 14/Apr/2019, Published: 30/Apr/2019

***Abstract*:-** Network attacks persist to pose a major threat to the internet. Various techniques are suggested for its mitigation from time to time but newer procedures of performing network attacks are continuously being promulgated by the intruders. The mitigation process becomes really difficult when it comes to highly distributed attacks performed using botnets. These attacks pose a major challenge to both the legitimate users as well as the infrastructure and to protect them, early discovery of the attacks is important. In this paper, Intrusion Detection and prevention System (IDPS) *Snort* is presented as a solution to identify different Network Attacks. *Snort* has been evaluated in a high-speed real network for different *DoS* and *Port Scan* attacks to examine its behaviour and capacity in detecting them. A set of custom rules have been proposed which show promising results in detecting the attacks but it still has scope for improvement.

Keywords: *NIDS, Snort v2.X, D-ITG, Scapy, DoS attacks, flooding, Port Scan*.

I. INTRODUCTION

Network attacks pose a major threat to any network as they are launched every hour of every day and evolve at a striking pace. Two major types of network attacks are denial of service and *Port Scan*. A denial of service (*DoS*) attack prevents authorised and legitimate users from accessing computing or memory resource from the network by making the network too busy or too full. It works by overloading the server with a huge number of requests so that system shutdown occurs because of the overload [1]. *DoS* attacks fall into one of three categories [2]:

- **Volume Based Attacks**

The attacker sends a huge number of packets to the targeted, more than the target can handle, so that the target's bandwidth is saturated and it can no longer accept any new requests. This prevents any legitimate user from accessing the server. For example TCP flood and UDP flood.

- **Protocol Attacks**

It includes sending some particularly crafted packets to the vulnerable server or application leading to crashing of host machine or ending the service respectively. This type of attack consumes the server resources making it unable to process the legitimate requests. Example of such an attack is Ping of Death.

- **Application Layer Attacks**

This type of attacks exploit weakness in Layer 7 of protocol stack. These attacks are most difficult to identify and mitigate. The attacker establishes a connection with the target and then exhaust the server resources by monopolizing processes.eg HTTP flood and DNS flood.

DoS attacks occupy the bandwidth of the victim server by flooding the server with huge amount of packet data and so are also known as bandwidth attacks [3]. They aim at exhausting critical server resources like CPU capacity, internet link capacity, stack space in network etc. so that server is not able to provide services to authentic users. A typical *DDoS* attack is carried out in two steps. The first step involves compromising vulnerable systems in the internet and installing bots in them, turning them into zombies. In the second step, the attacker uses these zombies to launch an attack against the victim server.

*Port Scan*ning is used by attackers to discover services that can be exploited on target system or to break into them. Server systems have certain services running in them which are binded to particular port numbers. Servers are continuously listening for clients on those ports. By *Port Scan*ning, the attacker tries to find out the open ports and services, the underlying operating system, whether unauthenticated log-ins are supported etc. on the targeted servers. It is executed by sending flagged packets to different ports of the target and monitoring its response to find out if the port is vulnerable and can be exploited to break into the system.

Although Network attacks exist from a long time, but there still exists a risk as hackers invent new techniques to perform these attacks all over again [4]. This paper describes existing taxonomies for understanding different *DoS* and *Port Scan* attacks and then mitigating them by an open source IDS *Snort*. We have set-up a real network to analyse the efficiency of *Snort* in detecting attacks on a busy server machine. We have developed our own custom rules for *Snort* as per the generated traffic. *Scapy* has been used to perform the attacks and *D-ITG* has been used to generate the traffic which emulates the real-time traffic scenario at the servers.

An intrusion detection system (IDS) is a software application that monitors network or system activities for intrusion attempts, produces alerts and logs information about them for the administrator. An intrusion prevention system (IPS) is an IDS with an additional capability of blocking intrusions by either dropping the malicious packets, blocking the malicious IP address or resetting the connection [5]. An IPS acts faster on the threat whereas IDS only gets a copy of the network traffic and merely creates an alert for evaluation by the administrator (when the packet is probably already delivered) [6]. Collectively they are known as Intrusion detection and prevention system (IDPS). A good IDS is defined by its ability to identify true attacks, less number of false alerts and low value of drop packets [7].

1.1 *Snort* 2.X

Snort is a free and open source network IDPS software. It performs many functions from protocol analysis to content /matching, and can detect a variety of attacks and probes. It was initially launched as a lightweight cross-platform packet sniffer and was later upgraded to an IDS in 2003. *Snort* 2.X is a single-threaded user-level application which uses deep packet inspection (DPI) for examining packets wherein it first inspects the packet header for any malicious content and then goes on to examine the packet payload [7] . It is a developmental open source software and its latest stable release is 2.9.9.11. It is easy to configure, install and use and has around 5 million downloads till date. It works well in Windows, Linux and FreeBSD opera ting systems. The three operating modes of *Snort* are:

- **Packet sniffer;** displays the real time network traffic.
- **Packet logger;** saves the network traffic to the disk for traffic debugging later.
- **Network intrusion detection and prevention;** matches the network traffic against signatures and performs the specified actions.

Snort checks the incoming packet against all its rules and if the rule matches, an alert is generated and logged onto the disk and if specified, some preventive actions are also taken.

1.2 *Scapy*

Scapy is a packet crafting and manipulation tool written in Python language by Philippe Biondi [8]. It is a very flexible tool which gives users a platform to craft their own packets and study their behaviour in the network. User is free to put any value that is required in any field of network layer and stack them as per desire. It has the ability to forge, encode or decode packets, capture them from the network, send them to the network, match requests and replies and many more. It can be used to perform other tasks like trace routing, probing, scanning, network discovery and performing *DoS* attacks. We have used *Scapy* to generate *DDoS* and *DoS* attacks in the network at different speeds.

1.3 D-ITG

D-ITG (Distributed Internet Traffic Generator) is traffic generating platform which can be used to produce IPv4 and IPv6 packets [9] at application, transport and network layer. It replicates different application-level protocols to generate traffic as per stochastic models of inter Departure Time (*IDT*) and packet sizes (*PS*) for different time intervals. It has been experimentally proved that *D-ITG* can generate a packet rate of 75000, where size of each packet is 1024 bytes [10].

In this paper, we have suggested a set of new modified custom rules for *Snort* v2.9.11.1 and also tested their ability to mitigate network attacks in a network of 1GBps. We have designed a real-time network to measure the performance. The paper is organized as follows, Section I contains the introduction of intrusion detection system *Snort* versions 2.9.9.0 and traffic generator *D-ITG* and *Scapy*. Section II contain the related work done by other researchers to evaluate the performance of *Snort* under different network attacks, Section III contain the experiment plan, Section IV contains the proposed rules and observations regarding the behaviour of *Snort* under attack in real-time network, Section V discusses and infers the results obtained and finally Section VI concludes the research work.

II. RELATED WORK

C. chen in [11] has proposed a statistical model based on two-sample t-test for Distributed *DoS* evaluation. It identifies the difference between incoming SYN arrival rate (SAR) and normal SAR and counting the SYN and ACK packets send and received by the host. The method involves low computation overhead and has lower value of false negatives and false positives. It follows three main approaches: packet marking, proactive and reactive. In Packet marking, suspicious packets are marked with some bits at distributed routers, and then separate them if they are exceeding thresholds. However, this counting and reacting method will become computationally intensive when

deployed on a server receiving a huge number of legitimate requests per second.

In [12], a procedure to improve the detection ability of *Snort* has been suggested for network probe attack. They have first evaluated and studied the behaviour of *Snort* for detecting attack traffic of MIT-DARPA 1999 dataset and then used this information to propose new improved rules that would increase the detection ability of *Snort* and reduce the number of false positives. They showed that with the new improved rules, *Snort* showed more accurate results. However they have checked the accuracy on DARPA 1999 dataset only and no real time traffic and attack scenarios were considered.

In [13], FireCol is proposed and tested to detect *DDOS* attacks. A FireCol consists of multiple IPSs and is placed near the attacking source and as far as possible from the victim host in order to reduce the delay in detection. It was evaluated on real time data and DARPA-99 dataset. FireCol shows good detection ability and is robust with less computational and communication overhead. However, the tested it on age-old DARPA-99 dataset and the rules used are not specified.

Rule based methods like Fuzzy Inference System (FIS) and Decision Trees (DT) are suggested for signature based analysis in [14]. A test bed is prepared to generate *DDoS* attack in real time for Network, Transport and Application layers. The attacks are detected using *Snort* by adding custom rules to its rule-set in order to reduce the false positive rate. The datasets like *KDD* cup 99 and Shonlaus Truncated Command Sequences (*STCS*) are used for performance evaluation. They have developed an IDS ensemble tool which generates, detects and classifies real time attacks. The rules generated are specific to their test network and are not applicable in all network attack scenarios.

Simulation of *DoS* attack using UDP flooding has been presented in [15] and methods and tools to mitigate them have been proposed. UDP based flooding attacks like fraggle etc. have been simulated and methods have been suggested to mitigate them using IDS *Snort* and firewall. They have used UDP Flooder tool to generate huge amount of UDP packets for Windows OS. They have not simulated other *DoS* attacks based on TCP and ICMP which form a major proportion of network attacks.

In [16], authors have emulated four common *DoS* attacks using network packet generator tools and have detected them using *Snort* with their custom rules in their lab environment. The have performed Land, SYN flood, Smurf and UDP flood attacks using *visual packet builder* and *Frameip* tools.

In [17], performance comparison of FireCol and *Snort*-with modified rule set is presented. *Snort* rules are modified to more generalized rules so that even if one less condition is met, an alert of low priority is generated. The idea is that a packet with a small difference from known attack packet can be a variation of the attack. Results showed that new generalized rules have low overhead but require more time to process the same number of packets in comparison to FireCol. *Snort* based IDS proved out to be a feasible and efficient system for thwarting *DoS* attacks but has a greater number of false alarms.

M. Gandhi et.al. in [18] have created a signature based IDS for identifying network attacks. The IDS works in promiscuous mode to capture all the network traffic and compare it with *Snort* signatures to identify intrusions and report them to the entire network. The IDS can be operated in both HIDS and NIDS mode and was tested for four *DoS* and *Port Scan* attack. They have not explained any test statistics and no work is done to validate the proposed IDS.
In [19] different techniques like signature detection, anomaly detection and mining based classification are used for *DoS* attack detection and mitigation. Four detection systems viz. *Snort*, *Phad*, *Madam* and *Multops* are compared and evaluated. *Snort* showed best detection rate for known *DoS* vulnerabilities but performed poorly for unknown *DoS* vulnerability and *DoS* flood attacks. *Snort* has low false alarm rate but requires more time to write alerts and get them examined manually. The paper does not talk about the behaviour of *Snort* on receiving high speed traffic and dropping of packets.

Our work is different from all the previous work for various reasons: (1) Researchers have tested their rules on age-old DARPA data set whereas we have tested them on a real network (2) Packet c rafting tool *Scapy* has never been used to perform the attacks. (3) The traffic rate at which the experiments are carried out is high as compared to the related work. (4) The proposed rules are not limited to our own network but can be reutilized for other networks as well. (5) A total of 15 attacks including both *DoS* and *Port Scan* attacks are evaluated in this paper

III. EXPERIMENTAL SET-UP

We aim to evaluate *Snort* in host-based intrusion detection mode by analysing its performance for different network attacks on Ubuntu 16.04 server in a high speed network. *Snort v2.9.11.1* is installed in its default configurations and later custom rules are added to it for evaluation. *D-ITG* is used to generate high-speed traffic to emulate the traffic available to server. We are using two dedicated machines for traffic generation using *D-ITG*. Attacks are performed using *Scapy*. Two attackers are used to perform the attack so that *a*

large number of attack packets are generated and tests are performed accurately. Figure 1 shows our experimental set-up for performing network attacks. The system description and specifications are enlisted in table 1.

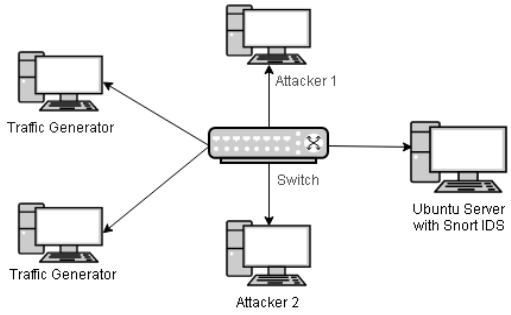

Figure 1. Experimental set-up

Table 1. System Specifications

Machine	Description	Specifications
Traffic generator 1 and 2	Dell Intel(R) core(TM) i3-3110M CPU @ 2.40GHz, 8 GB RAM	Ubuntu with *D-ITG* traffic generator
Attacker 1 and 2	Hp Intel(R) core(TM) i5-3210M CPU @ 2.40GHz, 8 GB RAM	Ubuntu with *Scapy*-python
Snort v2.9.9.0	Hp Intel(R) core(TM) i7-4770M CPU @ 2.40GHz, 8 GB RAM	Ubuntu 16.04 server

3.1. Evaluation parameters

Three evaluation parameters are used to evaluate the detection ability of *Snort*, after the addition of new rules to its database, under attacks the parameters are:

1. ***Snort* Efficiency:** It gives the value of number of packets analysed per second. More the *Snort* Efficiency more is its ability to detect attacks. It is measured in kilo packets analysed per second (kpps), using the formula

   Snort efficiency = $\frac{total\ packets\ analysed\ by\ Snort}{60*1000}$.

2. **Undetected Attack Packets Percentage:** It gives the value of attack packets that are not checked by *Snort* and are dropped by it. These packets go undetected and so more the undetected attack packet percentage, less is the performance of *Snort*. It is calculated using the formula

   $$\frac{total\ attack\ packets\ send - attack\ packets\ analysed}{total\ attack\ packets\ send} *100$$

3. **CPU Utilization:** It gives the percentage of CPU utilized under attack.

IV. PROPOSED SYSTEM

A new set of custom rules is proposed for *Snort* v2.9.11.1 for identifying various network attack. A total of 15 rules have been added in *Snort* Database to help identify attacks in a high speed network of 1 Gbps. Attack traffic along with normal network traffic was send using *Scapy* and *D-ITG* respectively for 60 sec. Two types of network attacks are considered i.e. *DoS* based attacks and *Port Scan* attacks.

4.1. *DoS* based attacks

A *DoS* attack attempts to utilize all of the computing and memory resource of the server so that it becomes too busy to handle legitimate requests and starts denying them. Some *DoS* attacks involve creating malformed packets, some try to change the flow of the packet, while others misemploy the basic elements of the packet [13]. A total of 11 *DoS* attacks have been discussed in this paper. We have created our own *Snort* rules to detect them and evaluated the performance of the Server under these *DoS* attacks.

The TCP is one of the main protocols of the Internet and provides reliable, ordered, and error-checked delivery packets between hosts communicating via an IP network. To establish a reliable connection between the hosts, a three-way handshake [20] is used. (1) Client sends a SYN packet to the server and sets the segment's sequence number to any random value (say X). (2) Server responses to the packet with a SYN-ACK. The acknowledgment number is one more than the received segment sequence number (X+1) and then server assigns another random sequence number to the packet (say Y) (3) Client sends an ACK back to the server and its sequence number is set to the received acknowledgement value (X+1) and the acknowledgement number is set to one more than the received sequence number (Y+1). Some of the TCP based *DoS* attacks exploit the three-way handshake mechanism to perform the attack.

1) SYN-flood

It is a denial of service attack in which attacker sends a huge number of TCP SYN packets from one or many spoofed source addresses to the server [21]. Server tries to respond to these SYN packets by sending ACK-SYN packets to the spoofed IP addresses, which would not respond back with expected Acknowledgements. This results in a half-open connection and all these waiting connections are stored in a queue for some time. When the number of waiting connections exceed the queue limit, all subsequent SYN requests are dropped leading to denial of service to authorized users. The aim of the attack is to utilize enough server resources to make the system unresponsive to authorized user requests.

We have used *Scapy* to generate a number of SYN packets with spoofed source addresses. *Snort* is used to monitor the traffic and is the number of SYN packets received by it exceed 2500 packets per second [22] , an alert is generated.

***Snort* rule:**
alert tcp !$HOME_NET any -> $HOME_NET 80 (flags: S; msg: "Possible *DDoS* TCP attack"; flow: stateless; detection_filter: track by_dst, count 150000, seconds 60; sid:10000001; rev:001;)
***Scapy* statement to generate attack:**
send (IP (dst = "192.170.1.120", src = RandIP()) / TCP (dport=80, flags="S"), loop=1)

2) Land attack

It is another denial of service attack where the attacker sends a SYN packet with spoofed source address as that of destination. Thus, in such a packet both the source address as well as the destination address are the same [21]. These attacks are performed on some older TCP/IP implementations. On receiving such packets, the host system gets locked up and has to be physically turned on again. For detecting such attacks by *Snort* a keyword *sameip* is used in the rule which checks for same source and destination values.
***Snort* rule:**
alert tcp any any -> $HOME_NET any (sameip; msg:"LAND attack"; sid:10000002; rev:001;)
Attack statement:
send (IP (dst = "192.170.1.120", src = ”192.170.1.120”) / TCP(dport = RandShort()), loop=1)

3) Mail bomb attack

In this type of attack, the attacker sends a flood of e-mails to the victim server so that server's mail queue is overloaded causing system failure. For detecting a mail bomb attack, IDS may look for thousands of mail messages coming from or sent to a particular user within a short period of time. The rule shown below fires when the number of mails coming to SMTP port 25 for established TCP sessions exceed 1000 e-mails per second.
***Snort* rule**:
alert tcp any any - $SMTP_SERVER 25 (msg: "Possible Mail Bomb attack"; flags:A+; flow:established; detection_filter: track by_dst, count 60000, seconds 60; sid:1000003; rev:001;)
Attack statement:
send (IP(dst = "192.170.1.120", src = RandIP()) / TCP(dport = 25, flags = "AS"), loop =1)

4) HTTP flooding

In HTTP flooding, first the attacker establishes a connection with the server using three-way handshaking and then floods the server with numerous http requests. The attacker intends to exhaust the server side resources like I/O bandwidth, memory and CPU by sending numerous valid and invalid ‘GET’ requests to the server. *Snort* rule checks for the number of GET requests received by the server for an established TCP session. Our rule fires when the number of requests exceed 1500 per second. For testing, an Ubuntu mail server is used on which *Snort* is installed.
***Snort* rule:**
alert tcp !$HOME_NET any -> $HOME_NET 80 (flags:S; msg:"Possible http flood attack"; flow:established; content:"GET"; nocase; http_method; detection_filter: track by_dst, count 90000, seconds 60; sid:10000004; rev:001;)
Attack statement:
send (IP (dst = ”192.170.1.117”, src = RandIP())/ TCP (dport=80)/ “GET /HTTP/1.0\r\n\r\n”, loop=1)

5) TCP reset attack

After a successful TCP three way handshake is established, RST or FIN packets are exchanged between server and client to either restart or close the TCP-SYN session between them. The attacker utilizes this idea and sends a large number TCP RESET packets to the target server. The server when receive a large number of spoofed RESET packets, which do not belong to any of its current sessions, tries to process the invalid requests wasting its resources on them. The attack tries to exhaust the server’s resources making it unavailable to process legitimate requests. An IDS can detect such packets by searching for reset packets to the server that do not belong to any established TCP connection
***Snort* rule:**
alert tcp any any -> $HOME_NET 80 (flags:R; msg:"Possible *DDoS* TCP attack"; flow:stateless; sid:10000005; rev:001;)
Attack statement:
send (IP (dst = "192.170.1.120", src = RandIP()) / TCP(dport = 80, flags = "R"), loop=1)

6) Christmas tree attack

This attack is done using a Christmas tree packet in which all three FIN, URG, and PUSH flags are set. A large number of Christmas tree packets when send to the server utilizes all the server resources as they require much more processing than normal packets [23]. The server may get exhausted and is unavailable for legitimate requests. By observing how the server responds to such an odd packet, inferences can also be made regarding the server’s operating system.
***Snort* rule:**
alert tcp !$HOME_NET any -> $HOME_NET 80 (flags:FPU; msg:"Possible christmas tree *DoS* attack"; flow:stateless; sid:10000006; rev:001;)
Attack statement:
send (IP(dst="192.170.1.120", src=RandIP()) / TCP (dport = 80, flags = "FPU"), loop=1)

7) UDP flood

It is a *DoS* attack in which a flood of spoofed UDP packets is send by the attacker to random ports of the victim server. As UDP is a connectionless protocol, so no three-way handshaking is required like TCP. On receiving a UDP

packet on a particular port, the server determines which application is running on that port and if no application is available, an ICMP packet of destination unreachable is send to the spoofed source address. If a large number of UDP packets are received by the victim, system's performance goes down and it may become unavailable. *Snort* rule fires if number of UDP packets per second is more than 1500.

***Snort* rule:**
alert udp !$HOME_NET any -> $HOME_NET !53 (msg: "UDP-FLOOD detected"; flow: stateless; detection_filter: track by_dst, count 90000, seconds 60; sid:10000008; rev:001;)
Attack statement:
send (dst = "192.170.1.120", src = RandIP())/ UDP(dport = RandShort()), loop=1)

8) DNS flood
It is one of the toughest *DDoS* attacks to detect and prevent. The attacker sends a large amount of DNS requests from spoofed IP addresses to DNS server. The request packets emulate a real DNS request are difficult to be differentiated from legitimate ones. In order to serve all the incoming DNS requests, the server exhausts its resources. The attack consumes all available I/O bandwidth of the server until it is completely exhausted. *Snort* rule fires when the number of DNS requests to DNS server exceed 1000 per second.

***Snort* rule:**
alert udp !$HOME_NET any -> $HOME_NET 53 (msg:"DNS FLOOD"; detection_filter: track by_dst, count 60000, seconds 60; sid:10000009; rev:001;)
Attack statement:
send (IP (dst = "192.168.5.1")/ UDP() / DNS (rd =1, qd = DNSQR(qname="www.jammuuniversity.in")), loop=1)

9) ICMP flood
An ICMP flood attack is executed by overloading the victim server with thousands of ICMP ping requests from spoofed IPs. The victim would then use all of its resources in replying to these ping message until it can no longer process any valid request. *Snort* rule can detect this attack by counting the number of ICMP ping packets received per second. If the number of ping requests exceed 1500 per second, the rule fires.

***Snort* rule:**
alert icmp !$HOME_NET any -> $HOME_NET any (msg:"ICMP-FLOOD"; itype:8; detection_filter: track by_dst, count 90000, seconds 60; sid:10000010; rev:001;)
Attack statement:
send (IP(dst = "192.170.1.120", src = RandIP()) / ICMP(type= 8), loop=1)

10) Ping of Death
It is a denial of service attack that affects many operating systems. It is launched by sending large sized ICMP ping packets to the victim. The size of the packet is larger than 65536 bytes, which is more than IP specification limit and will either crash, hang or reboot the victim machine. To identify such packets using *Snort* we measure the size of ICMP packet using *dsize* and if it's larger than 65535, the rule fires.

***Snort* rule**:
alert icmp !$HOME_NET any -> $HOME_NET any (msg:"ping of death detected"; dsize: >65535; itype: 8; icode:0; sid:10000011; rev:001;
Attack statement:
send (fragment (IP(dst = "192.170.1.120") / ICMP()/ ("X"*60000), loop=1)

11) Smurf Attack
In the "smurf" attack, a large amount of ICMP echo requests (ping) packets are send to the broadcast IP address of the network with spoofed source IP address of the intended victim. Every host machine present in the subnet of that broadcast address will receive these the ICMP echo requests and reply to the victim host with ICMP echo reply. The attacker amplifies the attack by using the broadcast address which multiplies the traffic by the factor of active hosts in the subnet. The maximum possible amplification factor is 255 [21]. The Smurf attack can be identified by monitoring the number of 'echo replies' being sent to the victim machine from many different places in a small time span.

***Snort* rule:**
alert icmp any any -> 192.170.1.120 any (msg:"Smurf Attack"; itype:0; detection_filter: track by_dst, count 50000, seconds 60; sid:10000012; rev:001;)
Attack statement:
send ((IP(dst="192.170.1.255", src = "192.170.1.120") / ICMP(), loop=1)

4.2. *Port Scan* attacks
A *Port Scan* attack is aimed to find out what ports are open on the victim machine. The attacker sends a number of packets to the victim machine by varying the destination port to find out what services are running on which port and also to get an idea of the victim's operating system [12]. Attacker may send TCP packets (with different flag values of the header) and observe the response of the victim, to know if a particular port is listening or not .A summary of TCP/IP implementation rules followed by host machines on receiving flagged TCP packets prior three way handshake is shown in table 2. We have performed four *Port Scan* attacks.

Table 2. TCP responses to flagged packets

S. no.	TCP Flag value	Receiving host	
		Open/listening port	*Closed port*
1	SYN	SYN-Ack is send for three-way handshake	Drop packet and send RST packet
2	RST	Drop packet	Drop packet
3	ACK	Drop packet and send RST message	Drop packet
4	FIN/PSH/ URG	Drop packet	Drop packet and send RST packet
5	Null	Drop packet	Drop packet and send RST packet

1) Ack scan

TCP provides reliability in delivery of data by assigning a sequence number to every byte that is to be transmitted. It also waits for an acknowledgment from the other end upon receipt of the data. Acknowledgement Number is a field in the TCP header which is 32 bits long and gives the information regarding the next packet sequence number the other end is expecting next [23]. However, this field is significant only when the ACK flag in the TCP header is set. In ACK scan, attacker sends TCP packets to different ports of the victim with ACK flag set but sequence number has value 0. But as per TCP rules such packets are not acceptable by the receiver, which drops such packets and sends back a RST packet. When attacker receives this RST packet, it learns that the service is open at that particular port.

***Snort* rule:**

alert tcp any any -> $HOME_NET any (flags: A; ack: 0; msg:"ACK Scan Detected"; sid:10000013; rev:001;)

Attack statement:

ans, unans = srloop (IP (dst = "192.170.1.120", src=RandIP())/ TCP(dport = (0,1024), flags="A"))

2) FIN scan

This scan technique uses FIN segments to probe victim ports. TCP packets with FIN flag set are send to different ports on the victim machine with spoofed source IP address and the responses are recorded. When such a packet arrives for a closed port, the victim machine drops the segment and sends back a RST packet. Otherwise, when it arrives for a listening port, it is simply dropped and no RST packet is sent. On receiving the responses, the attacker can make out which ports are open and which ones are closed.

***Snort* rule:**

alert tcp !$HOME_NET any -> $HOME_NET any (flags:SF; msg:"FIN scan"; flow: stateless; sid:10000014; rev:001;)

Attack statement:

ans, unans = srloop (IP (dst = "192.170.1.120", src=RandIP())/ TCP (dport = (0,1024), flags="SF"))

3) Null scan

The Null Scan is a variation of the above scan to identify open TCP ports. The attacker sends a series of TCP packets that contain a sequence number of 0 and no flags are set. In real traffic, there will never be a TCP packet that doesn't contain a flag and so such packets can penetrate firewalls and edge routers that filter incoming packets with particular flags. On receiving a Null packet, the open ports do not respond but the closed ports respond with a TCP RESET packet.

***Snort* rule:**

alert tcp !$HOME_NET any -> $HOME_NET any (flags:0; msg:"Null scan"; flow:stateless; sid:10000015; rev:001;)

Attack statement:

ans, unans = srloop (IP (dst = "192.170.1.120", src=RandIP())/ TCP (dport = (0,1024), flags=0))

4) FTP bounce scan

In FTP bounce scan, FTP protocol is exploited by attacker to gain access to ports of the victim machine via a middle machine. This technique can be used to *Port Scan* hosts discreetly, and to access specific ports that the attacker cannot access through a direct connection. The attacker first opens up a FTP connection to a FTP server (middle machine) and requests it to execute a file which opens up a new FTP connection to a specific port of victim machine [24]. After the FTP connection is established, the data connection is made by specifying data port and IP address of requesting machine (in this case, the middle machine). Here, the attacker tricks by spoofing the port number and IP address of middle machine with its own. To prevent such attacks using IDS, any packet which contains PORT command for connection to FTP server should be monitored and alerted. FTP server is activated on Ubuntu server for the attack.

***Snort* rule:**

alert tcp !$HOME_NET any -> $HOME_NET 21 (msg:"FTP Bounce scan"; content:"PORT"; nocase ; ftpbounce; sid:10000015; rev:001; pcre:"/^((\%70)|(p)|(\%50))((\%6f)|(o)|(\%4f))((\%72)|(r)|(\%52))((\%74)|(t)|(\%54)) /smi";)

V. RESULTS AND DISCUSSIONS

This section evaluates the custom *Snort* rules in a real network of 1 Gbps. *D-ITG* sends normal traffic at a rate of 1Gbps and *Scapy* performs the attack and sends attack packets at the rate of 3500-4000 packets per second. The

various packets received, analysed and dropped by *Snort* are recorded in table 3 and 4. The attacks packets that are dropped (excluding normal packets that are dropped) are also calculated to give a clear indication of the performance of the rule. We have compared the performance of *Snort* for server under attack with that of server under normal traffic with no attack packets.

***DoS* based attacks:**

From table 3 we can deduce that the rules performed very well for SYN-flood, TCP reset, Xmas tree, UDP flood, DNS flood, ICMP flood and smurf attacks as their values of Undetected Attack Packets percentage is less than 0.2%. It performed well for Land attack, Mail bomb and HTTP flood also but its performance was very low for Ping of Death as 90% of the packets were undetected. The reason for this behaviour in case of Ping of Death is that *Snort* takes more time in reassembling of fragments and so is not able to cope up with the high rate of attack traffic. Figure 3 shows the variation in undetected attack packets percentage for various attacks except Ping of Death.

Snort efficiency lies in the range of 35 – 56 for all the attacks with the minimum for that of Ping of Death and maximum for normal Traffic as in figure 2. This indicates that *Snort* performance is not affected to a greater extent even under attack and performs well in all the attacks except ping of death.

From figure 4, we can see that the CPU utilization is more under attack than under normal traffic. Also it is maximum for Ping of Death and minimum for TCP reset. For all others, it is in the range of 53-63%. The main aim of the attack is to exhaust the server resources like CPU and if IDS is also installed on the same machine than it may get less CPU time leading to undetected packets.

Table 3. Observations for *DoS* attacks

S. No.	Attack Type	Packets Received	Packets Analysed	Packets Dropped	Attack Packets Send	Attack Packets dropped	*Snort* Efficiency	Undetected Attack Packets Percentage	CPU Utilization
1.	SYN-flood	2885307	2885005	302	226889	302	48.08	0.133105	63.3
2.	Land attack	2779301	2770950	8351	213351	3351	46.18	1.570651	53.43
3.	Mail bomb	3080683	3068787	11896	206117	6551	51.15	3.178292	62.5
4.	HTTP flood	3105515	3102916	2599	214219	2289	51.72	1.068533	58.4
5.	TCP reset	2996648	2996152	496	2114278	0	49.94	0	45.43
6.	Xmas tree	3137510	3137205	305	227979	10	52.29	0.004386	59.65
7.	UDP flood	2837670	2837347	323	219404	326	47.29	0.148584	58.083
8.	DNS flood	2780308	2778606	1702	174794	1477	46.31	0.844995	57
9.	ICMP flood	2916154	2916154	0	221998	0	48.6	0	57.18
10.	Ping of death	2342191	2107802	234389	254103	230003	35.13	90.51566	70.3
11.	Smurf	2486160	2485891	269	254257	269	41.43	0.105798	58.82
12.	Normal traffic	3403692	3403692	0	0	0	56.73	0	39

Table 4. Observations for *Port Scan* attacks

S. No.	Scan Type	Packets Received	Packets Analysed	Packets Dropped	Attack Packets Send	Attack Packets dropped	*Snort* Efficiency	Undetected Attack Packets Percentage	CPU Utilization
1.	ACK scan	3085975	3085975	0	224115	0	51.43	0	61.37
2.	FIN scan	3094506	3081037	13469	229282	3790	51.35	1.652986	69.08
3.	Null scan	3150763	3150466	297	228201	0	52.51	0	55.77
4.	FTP Bounce	3024858	3023672	1186	194633	307	50.39	0.157732	74.1
5.	Normal traffic	3403692	3403692	0	0	0	56.73	0	39

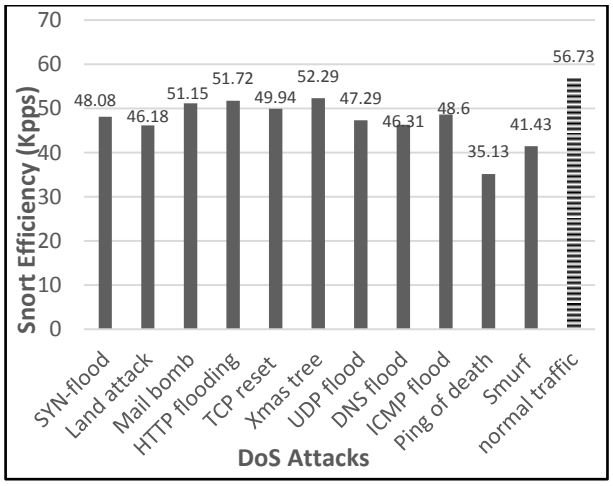

Figure 2. Comparison of *Snort* efficiency for different *DOS* attacks

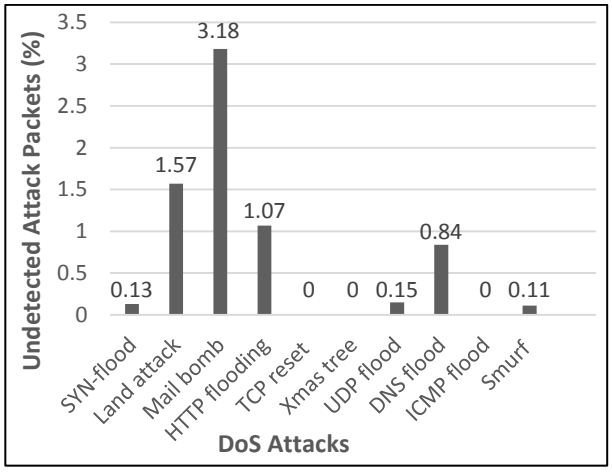

Figure 3. Undetected attacks packets percentage for *DoS* attacks

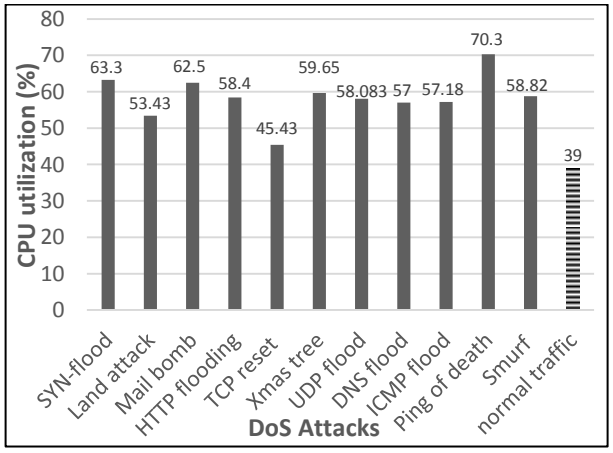

Figure 4. CPU utilization for different *DoS* attacks

***Port Scan* attacks**

Table 4 shows that all the *Snort* rules for *Port Scan* attacks performed well as the value of Undetected Attack Packets percentage for all of them is less than 0.2%. However, amongst them the maximum packet drop is shown by FIN scan (figure 6). *Snort* efficiency is similar for almost all the attacks and lies in the range of 51-53 Kpps but is less from that for normal traffic as shown by figure 5. This is because under attack *Snort* has to report the intrusions by writing alerts to disk which reduces its efficiency. The value of CPU utilization varies from 55-71 % for attacks which is very high as compared to CPU utilization for normal traffic (figure 7).

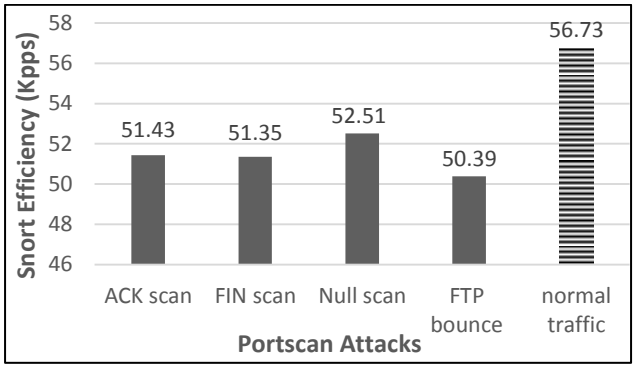

Figure 5. Comparison of *Snort* efficiency for different *Port Scan* attacks

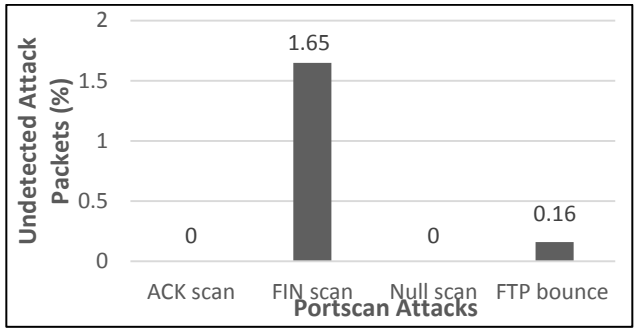

Figure 6. Undetected attacks packets percentage for *Port Scan* attacks

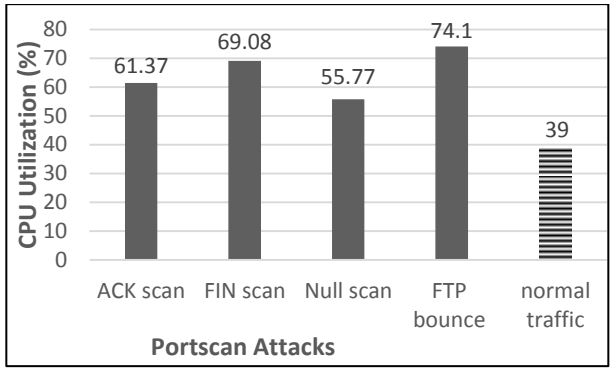

Figure 7. CPU utilization for different *Port Scan* attacks

VI. CONCLUSION

n this paper, we propose a number of *SNORT* rules to detect SQLIAs. The *SNORT* rules we present show a significant improvement in performance in detecting SQL injection attacks. In some cases the proposed rules perform 100% success in detection with zero false alarm, in particular rule 5 (Table 1).

n this paper, we propose a number of *SNORT* rules to detect SQLIAs. The *SNORT* rules we present show a significant improvement in performance in detecting SQL injection attacks. In some cases the proposed rules perform 100% success in detection with zero false alarm, in particular rule 5 (Table 1).

This paper attempts to improve the detection ability of *Snort* by proposing new set of custom rules to detect network attacks viz. *DoS* and *Port Scan* attacks. The results of our test-bed show that the new custom *Snort*-IDS rules show a significant improvement in correctly detected network attacks. For SYN-flood, TCP reset, Xmas tree, UDP flood, DNS flood, ICMP flood and Smurf attacks, the percentage of Undetected Attack Packets was less than .2%. Also ACK scan and Null scan showed 100% success as the value of undetected attack packets is zero. However, the rule for Ping of Death performed poorly and dropped 90% of the attack packets. Other than that all other rules prove to be efficient and accurate. For future, we will improve the *Snort* rules for network attacks of type U2R and R2L.

REFERENCES

[1] Q. Gu and P. Liu, "Denial of Service Attacks, Technical Report," http://s2.ist.psu.edu/paper/DDoS-Chap-Gu-June-07.pdf.

[2] S. Acharya and N. Tiwari, "Survey Of DDoS Attacks Based On TCP/IP Protocol Vulnerabilities," IOSR Journal of Computer Engineering, vol. 18, no. 3, pp. 68-76, 2016.

[3] T. PENG, C. LECKIE and K. RAMAMOHANARAO, "Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS problems," ACM Computing Surveys, vol. 39, no. 1, April 2007.

[4] S. Sharma, Y. Verma and A. Nadda, "Information Security: Cyber Security Challenges," International Journal of Scientific Research in Computer Science and Engineering, vol. 7, no. 1, pp. 10-15, 2019.

[5] M.Shivakumar, R.Subalakshmi, S. Shanthakumari and S. Joseph, "Architecture for Network-Intrusion Detection and Response in open Networks using Analyzer Mobile Agents," International Journal of Scientific Reearch in Network Security and Communication, vol. 1, no. 4, pp. 1-7, 2013.

[6] P. Innella, " An Introduction to IDS," 5 dec 2011. [Online]. Available: https://www.symantec.com/connect/articles/introduction-ids.. [Accessed Accessed Dec 2017].

[7] S. Chakrabarti, M. Chakraborty and I. Mukhopadhyay, "Study of Snort-Based IDS," in Proceedings of the International Conference and Workshop on Emerging Trends in Technology, ACM, 2010.

[8] S. Bansal and N. Bansal, "Scapy–A Python Tool For Security Testing," Journal of Computer Science & Systems Biology, vol. 8, no. 3, pp. 140-159, 2015.

[9] A. P. G. V. D. Emma, Analysis and experimentation of an open distributed platform for synthetic traffic generation, Suzhou, 2004, pp. 277-283.

[10] S. Avallone, S. Guadagno, D. Emma, A. Pescap and G. Ventre, "D-ITG Distributed Internet Traffic GeneratorS. Avallone S. Guadagno D. Emma A. Pescap `eG. Ventre," in 1st International Conference on Quantitative Evaluation of Systems , Enschede, The Netherlands, 27-30 September 2004.

[11] C.-L. Chen, "A new Detection Method for Distributed Denial-of-Service Attack Traffic based on Statistical Test", J," Journal of Universal Computer Science, vol. 15, no. 2, 2009.

[12] N. Khamphakdee, N. Benjamas and S. Saiyod, "Improving Intrusion Detection System Based on Snort Rules for Network Probe Attack Detection," in 2nd International Conference on Information and Communication Technology (ICoICT), 2014.

[13] J. François, I. Aib and R. Boutaba, "FireCol: A Collaborative Protection Network for Detection of Flooding DDoS Attack," IEEE/ACM Transactions on Networking, vol. 20, no. 6, pp. 1828 - 1841, 2012.

[14] S. T, Application of Machine Learning Algorithms for Real Time Intrusion Detection and Classification, chennai, Tamil Nadu: Shodhganga, 2011.

[15] L. Xiaoming, V. Sejdini and H. Chowdhury, Denial-of-Service (DoS) Attack with UDP Flood., Windsor, Ontario: School of Computer Science, University of Windsor:, 2007.

[16] Z. Trabelsi and L. Alketbi, "Using network packet generators and snort rules for teaching denial of service attacks," in Annual Conference on Innovation and Technology in Computer Science Education, 2013.

[17] M. Saritha and M. Chinta, "Countering Varying DoS Attacks using Snort Rules," International Journal of Advanced Research in Computer science and Software Engineering, vol. 3, no. 10, October 2013 .

[18] M. Gandhi and S.K.Srivatsa, "Detecting and preventing attacks using network intrusion detection System," International Journal of Computer Science and Security, vol. 2, no. 1, pp. 49-60, 2008.

[19] D. Lin, "Network Intrusion Detection and Mitigation against Denial of Service Attack," University of Pennsylvania, Philadelphia, 2013.

[20] F. Hsu, Y. Hwang, C. Tsai, W. Cai, C. Lee and K. Chang, "TRAP: A three-way handshake server for TCP connection establishment," Appl. Sci., vol. 6, no. 11, 2016.

[21] K. Kendall, "Intrusion Detection Attacks Database," 1999.

[22] S. M. Aaqib, "To analuse performane, scalability and security mechanisms of apache web server vis-a-vis with contemporary web servers," University of Jammu, 2014.

[23] M. d. Vivo, L. Ke, G. Isern and G. O. d. Vivo, "A review of port scanning techniques," ACM SIGCOMM Computer Communication Review , vol. 29, no. 2, pp. 41-48, 1999 Computer Communication Review. .

[24] T. S. Buddy, "What is FTP Bounce Attack ?," 7 march 2017. [Online]. Available: https://www.thesecuritybuddy.com/vulnerabilities/what-is-ftp-bounce-attack/. [Accessed 1 feb 2018].

[25] "Study of Snort based IDS," in Proceedings of the International Conference and Workshop on Emerging Trends in Technology, ACM, 2010.

[26] M. Roesch, "SNORT 3 User manual," SourceFire.Inc, 2017. [Online]. Available: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/. [Accessed Nov 2017].

[27] T. W. Shinder, The Best Damn Firewall Book Period, Second ed., Syngress, 2007.

AUTHORS PROFILE

Dr. Lalit Sen Sharma has obtained Master of Science in Mathematics and MCA from Guru Nanak Dev University, Amritsar (India). He has also obtained Doctorate of Philosophy (PhD) from Guru Nanak Dev University in 2008. Currently, he is working as a Professor and Head of Department in the department of Computer Science and Information Technology in University of Jammu, India. He has been teaching to postgraduate students of computer applications for fifteen years. He is a member of Indian Science Congress Association, Institute of Electronics and Communication Engineer, India and National HRD network, India.

Alka Gupta has obtained her B.E. in Computer Science from University of Jammu, India and has received her M. Tech in Computer Science from Shri Mata Vaishno Devi University Katra, J&K, India she has been pursuing her Doctorate of Philosophy (PhD) from Department of Computer Science and IT, University of Jammu since 2016. Her areas of interest include Computer Networks, Network Security, Data structures and mobile computing.

