IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-3, March 2019 E-ISSN: 2347-2693

A novel framework for combating network attacks using *Iptables*

Nikita Gandotra[1*], Lalit Sen Sharma[2]

[1,2]Dept. of Computer Science & IT, University of Jammu, Jammu (J&K), India

[*]*Corresponding Author: nkt_2201@yahoo.co.in*

DOI: https://doi.org/10.26438/ijcse/v7i3.226237 | Available online at: www.ijcseonline.org

Accepted: 20/Mar/2019, Published: 31/Mar/2019

Abstract— Network attacks pose as grievous threat to the stability of the Internet and are a major security concern as they can breach the security of the network or even make the victim unavailable. The network attack packets can intercepted by using *Iptables* before they can reach the victim machine. *Iptables* is the standard firewall included in Linux distributions for handling the kernel *Netfilter* modules. The effectiveness of the defense provided by the *Iptables* firewall mainly depends on its rules. In this paper, we have proposed a novel framework with new customized *Iptables* rules for mitigating fifteen types of network attacks which include port scanning; denial of service attacks, TCP, UDP, and ICMP based attacks etc. The performance of *Iptables* with these rules is evaluated with the real experiments for examining the competence of firewall in managing the network traffic and security when subjected to attack flow along with the normal traffic. The performance of *Iptables* is recorded in the terms of CPU utilization for processing and Logs generation, Frame Loss Ratio and Efficiency. The attack traffic is generated using *Scapy* for execution of the attacks whereas the normal traffic is generated using a traffic generator called D-ITG. It was found that *Iptables* could successfully detect the network attack and performed really well during the mitigation of such attacks.

Keywords—*Iptables*, *Netfilter*, *Scapy*, DITG, network-attacks

I. Introduction

Network attacks pose as threats to security and services provided by the servers connected to the Internet. The servers are continuously being targeted by intruders by different network attacks as these are extremely simple to execute in comparison to other types of cyber-attacks and can hamper the performance of the servers. In order to provide protection to the network, firewalls are usually deployed as these form the first line of defense without restricting the information exchange with the outside world. *Iptables* is an open source firewall that is included in most Linux distributions and has become extremely popular among researchers due to its reliability, robustness, flexibility and apparently infinite scope for customization. Since *Iptables* is an open source firewall and provides high dependability for handling burst traffic [1], thus it was chosen upon the other software firewalls. *Iptables* was developed under the *Netfilter* project [2]. The *Netfilter* module is a kernel space program that facilitates packet filtering, network address translation and several other packet mangling facilities. It examines the incoming and outgoing packets and allows or blocks them based on a set of rules that represents a set of conditions [3]. If all the conditions specified in a rule matches the packet, then the specified action is taken otherwise the default policy is applied. When a packet arrives at *Netfilter* hooks [4], the firewall searches the rule space sequentially, one by one and if a matching rule is found, first matching rule is applied against the packet.

This paper focuses on formulation of new customized *Iptables* rules for mitigating fifteen different network attacks [5] and also verifies their potential for their use in real life scenarios. The performance of *Iptables* is evaluated with these rules on an experimental testbed and the behavior of the firewall is observed when subjected to different network attack flows along with the normal traffic. The performance and security provided by *Iptables* is tested by performing real-time experiments and recording it in the terms of key parameters: CPU utilization for *Iptables* and Logs generation, frame loss ratio and efficiency. Key parameter represents the ability of firewall in managing the attacks along with network traffic. The network attacks were performed using *Scapy* [6] and their corresponding techniques are also discussed in the paper. Also, the normal network traffic was generated using D-ITG [7] to emulate the actual working condition of the firewall.

The main contributions of this work is as follows:

1. Performance of *Iptables* under attacks has not been evaluated before. The previous works have only reported the occurrence of attack without measuring its performance;

2. The rate limits proposed in this paper precisely represents the real life scenarios [19] as compared to previous works;
3. The performance of *Iptables* with *Scapy* has not been measured before;
4. The tests are conducted on high speed network of 22Mbps with attack rate of 2 Mbps; and
5. Both the framework and performance of *Iptables* is discussed for fifteen types of network attacks.

The organization of remaining paper is as follows: Section II briefly discusses the structure of *Iptables* while Section III presents an insight into the related works. The Experiment plan and the work done are discussed in Section IV. Section V and Section VI illustrate the observations and the results. Finally, Section VII summarizes and concludes this paper.

II. BACKGROUND

Iptables is a Linux administration tool used to access the *Netfilter* framework and allows the user access to the kernel firewall. The *Netfilter* module is implemented in the Linux as a kernel space program which is either included as a set of modules or compiled directly into the kernel. *Iptables* perform Stateful Packet Inspection (SPI) of the incoming and outgoing packets to filter out packets based on the security policy. It keeps track of every connection passing through it and has the ability to filter out packets based on the MAC address which makes it a formidable security solution.

Iptables use separate rule tables to support miscellaneous packet processing facility which are implemented as separate modules. The three primary tables used by *Iptables* are FILTER table, NAT table, and a MANGLE table [8]. These tables operate on different chains which contain the set of rules. Chains can be in-built or user defined [3] such as PREROUTING for NAT translation and checking the TOS, INPUT for incoming packets, FORWARD for redirecting incoming packets, OUTPUT for outgoing packets and POSTROUTING for NAT translation and services implemented on packet before leaving.

The FILTER table is primarily used for filtering packets where the actual action is taken against the packets whether to ACCEPT or DROP them. The decision depends upon the contents of packet header. It operates on INPUT, OUTPUT, FORWARD and user defined chains. The NAT table is used for network address translations and also for hiding the internal network using MASQUERADE target. It uses three types of chains: PREROUTING, OUTPUT and POSTROUTING. MANGLE table is a special packet handling module which operates on all chains and is used for enforcing TOS, TTL and security.

Firewall rules correspond to a set of conditions representing the situation for accepting or rejecting the packets. The different options [9] available for defining rules are: IP header fields, Device interfaces, Rate-limited packet matching, Current connection state, Addressing types, Layer 2 types, range of IP addresses, ICMP types, Length of the packet, Time of the arrival, Random packets, Packet sender's user, group, process, or process group ID, Type of Service field, Time to live and Targets.

III. RELATED WORK

Only a little work was available on the performance of firewall and their resilience under attacks. Most of the existing works laid much emphasis on reporting the occurrence of attacks [12-16] than the performance of the firewall under attack [17-18].

K. Chatterjee [10] studied the performance of CyberRoam, a commercial firewall under DoS flood. The system showed a poor performance due to high traffic and hence suggested the use of *Iptables* for restrictive forwarding and preventing DoS flood when the hardware could not be upgraded. B. Sharma and K. Bajaj [11] explored the use of *Iptables* for packet filtering in Linux. They discussed rules for blocking of HTTP traffic, specific URLs, ICMP traffic, SMTP traffic and P2P file sharing traffic.

B. Q. M. AL-Musawi [12] discussed the various types of DoS attacks and suggested *Iptables* rules for preventing only SYN flood, UDP flood and ICMP flood. The attack traffic was generated by hping and its occurrence was reported by using wireshark. Also, the rate limits proposed in rules were far away from real life scenario. Likewise, S. Mirzaie et al. in [13] proposed *Iptables* rules for mitigating SYN flood attack. They rate-limited the number of TCP connections from a single IP address. No further evaluation has been done in order to evaluate the efficiency of the suggested rule.

M. Šimon et al. [14] studied the use of IP6Tables and *Iptables* for IPv6 and IPv4 networks for mitigating HTTP-GET Flood. The test setup consisted of P2P grid and the performance of *Iptables* rules was measured in the form of network load, memory usage and network usage (data sent and received). Certain important parameters such as CPU Utilization and number of packets dropped by *Iptables* were not considered.

In [15], A. Balobaid studied the impact of DoS attacks and its mitigation on cloud (OpenStack- Icehouse) environment. The SYN flood and UDP flood were executed using Hping3 and *Iptables* was used for its mitigation. It was seen that *Iptables* significantly reduced the impact of the attack but the rules used were not discussed and the performance of *Iptables* was not evaluated.

M. Y. Arafat in [16] discussed an approach using *Iptables* to mitigate DNS Amplification Attack. The suggested rules were discussed in the paper but performance evaluation has not been done to check the effectiveness of the rules.

K. Salah et al. [17] discussed about the performance of network firewall and proposed an analytical model based on Markov chain. The performance of Linux *Netfilter* firewall was analyzed for 10000 dummy rules for both normal traffic and DoS traffic. The DoS traffic was generated by KUTE and the performance was recorded in terms of packet loss, throughput, packet delay, and CPU utilization. No *Iptables* rules for DoS attack were discussed in the paper.

T. Hayajnch et al. in [18] evaluated the performance of network and personal firewalls for security and performance. The performance evaluation was done on the basis of throughput, jitter, delay and packet loss while for security evaluation, different types of attacks that could be blocked by the respective firewall were reported. In particular, Cisco ASA 5510, Packet Filter, Comodo and ZoneAlam were considered and was inferred that Cisco ASA achieved better performance in comparison to other firewalls.

IV. EXPERIMENT PLAN

A. *Experimental Setup*

The experimental setup consists of a LAN of four PCs to assess the performance of *Iptables* under different types of network attacks. Figure 1 shows the setup together with the corresponding IP addresses.

The performance of *Iptables* is measured under attack along with normal traffic. For generating normal traffic, D-ITG (Distributed Internet Traffic Generator) is used. It is an open-source traffic generator with additional facility of network analysis, measurement and monitoring. It can be used to generate traffic by varying packet size and packet rate [20] and monitor the performance on the link without affecting the experiments. The normal traffic is generated at a rate of 6000 packets per second with the packet size of 256 bytes.

The traffic for different attacks were created using *Scapy* [6]. It is a packet manipulation tool written in Python that enables the user to send, sniff and forge the packets in a computer network. It provides the facility to describe a packet or a set of packets with user defined field values. The attacks were executed for duration of 120 seconds. The performance is measured in terms of following key parameters:

i. CPU Utilization by *Iptables* computed by averaging the amount of CPU utilization by the kernel during its execution and is measured by TOP command.
ii. CPU Utilization for Logs Generation computed by averaging the amount of CPU used by the SYSLOG during its execution.
iii. Frame Loss Ratio furnishes the total packets dropped by the firewall:

$$FLR = \frac{\text{Packets Dropped by Iptables} \times 100}{\text{Total Packets received by NIC}}$$

iv. Efficiency computed as the ratio of total packets received by *Iptables* to the packets received by the machine.

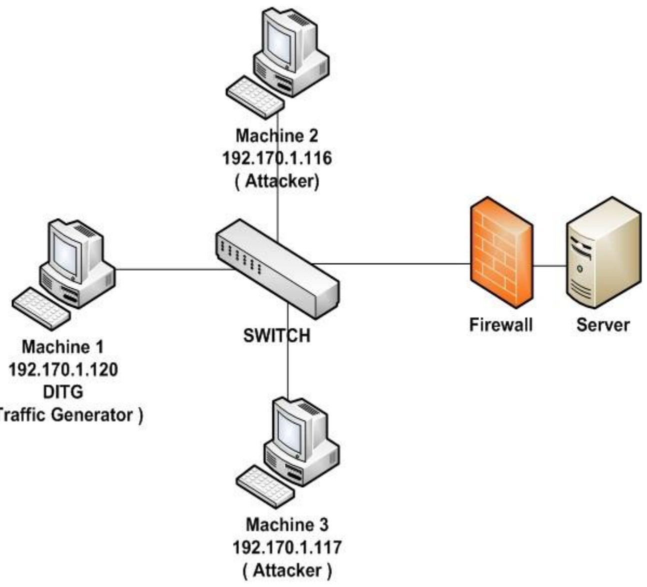

Figure 1. Experimental Setup

All the results are evaluated on machine with processor Intel® Core™ i3-3110 CPU @ 2.40 GHz along with 8 GB RAM space. Ubuntu-14.04 version was used as the operating system.

B. *Proposed Work*

A firewall can be positive filtering or negative filtering. Positive filtering firewall is set to allow all the traffic unless there is a rule to block it while in latter, no traffic is allowed unless there is a clear rule to allow it. This behaviour is defined by the choice of default policy of the firewall. Here, the default policies for all the chains in FILTER table are set to DROP. This is done to provide maximum protection to the system. The *Iptables* rules are written in a shell script to combat all the attacks and to allow traffic on certain ports. Total number of rules in the shell script is 210. The lines of code used for fixing the default policy are as follows:

```
iptables --policy INPUT DROP
iptables --policy FORWARD DROP
iptables --policy OUTPUT DROP
```

The following taxonomies briefly discusses the different types of attacks for which new customised *Iptables* rules were formulated:

TCP BASED ATTACKS

1. SYN Attack:

One of the major protocols of the internet is TCP. It provides reliable and connection-oriented service between two hosts. In order to create a connection, TCP uses three-way handshake:

i. Request where the client sends a SYN request to make a connection
ii. The server responds with an ACK (acknowledgement) and a SYN message, which accepts the client's request
iii. The client transmits back an ACK to the server, establishing the connection

Attackers exploit this process of TCP connection by generating fake messages to busy the servers for authenticated users. In a SYN attack, the system is targeted by multiple SYN packets with different spoof IP addresses which cause the system to respond with ACK/SYN messages and it continues to wait for the ACK from the spoof IP address which never arises. The server removes the waiting ACK from the concurrent connections, either when an ACK is returned or the connection interval timer is worn out which terminates the connection. The problem begins here because the server can handle only a limited number of concurrent connections. So, all the incoming requests are ignored when the service queue is full causing denial of service to the legitimate clients. To combat it, the rate of TCP requests is limited to 2500/sec. The decision of the limit is taken by considering the capabilities of the server to respond to concurrent TCP requests [19]. The following rules are formulated for this attack:

1) iptables -N TCPflood
2) iptables -A TCPflood -p tcp --syn -m limit --limit 2500/s --limit-burst 3000 -j RETURN
3) iptables -A TCPflood -p tcp -j LOG --log-prefix "*Iptables* TCP SYN Flood: " --log-level 7
4) iptables -A TCPflood -p tcp -j REJECT --reject-with tcp-reset
5) iptables -A INPUT -p tcp -m state --state NEW -j TCPflood

The above code creates a new chain called TCPflood. The limit match option matches the rate of incoming SYN requests to 2500/s and burst of 3000 i.e. the rate limit is enforced once the size of 3000 is reached. If the rate of incoming packets is less that 2500/sec then the control is returned to the calling chain (INPUT chain) and no action is taken here and the rule searching is resumed from the INPUT chain. While in case the rate of incoming requests is greater than the limit, then the packets more than the limit is dropped just like in token bucket technique. In order to execute the SYN attack, the SYN packets are generated by the following Python code:

send (IP (dst="192.170.1.119", src = RandIP ( ))/ TCP(dport = 80, flags= "S"),loop=1, inter=0.001)

This generates continuous TCP requests directed towards the server residing at the firewall on the HTTP port with an inter departure time of 0.001 sec.

2. ACK attack

It is a TCP based attack where the attacker sends packets with the ACK flag enabled and a forged IP address. On receiving the packet, the victim machine searches all the established TCP connections to find out the active connection to which it may belong. As the source contains spoofed IP address, the victim eventually drop these packets but causes resource exhaustion at victim machine as every packet has to be processed resulting in loss of service to the authenticated users. To combat the ACK attack, all the incoming TCP packets whose connection state are new but are not SYN requests are dropped here:

1) iptables -A INPUT -p tcp ! --syn -m state --state NEW -j LOG --log-prefix "*Iptables* Invalid Packet" --log-level 7
2) iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP

The ACK attack traffic was generated by the following Python code:
send(IP(dst="192.170.1.119",src=RandIP( ))/TCP(dport= 80, flags= "A"), loop=1, inter=0.001)

3. RESET Attack

TCP uses flags to indicate the type of packet. One of the flags used in TCP connections, is the "RST" flag which indicates that the connection should be terminated. This pre-empts the communication instead of completing with FIN flag. In a RESET attack, the attacker takes over the connection between two hosts or floods the host with massive number of RST packets which leads to re-initialisation of TCP connection by server. In order to block excessive RST packets, they are rate-limited to 400/sec when they occur in large frequencies by following lines:

1) iptables -A INPUT -p tcp -m tcp --tcp-flags RST RST -m limit --limit 400/s --limit-burst 400 -j ACCEPT
2) iptables -A INPUT -p tcp -m tcp --tcp-flags RST RST -j LOG --log-prefix "*Iptables* TCP RESET: " --log-level 7
3) iptables -A INPUT -p tcp -m tcp --tcp-flags RST RST -j DROP

The attack traffic was generated by following:
send (IP (dst = "192.170.1.119", src = RandIP ( )) / TCP (dport = 80 , flags = "R"),loop=1, inter = 0.001)

4. XMAS Tree Scanning

It is an attack where special packets are sent to the server that have either all the flags or FIN, URG and PSH flags enabled in the TCP header. These are often used for port scanning and performing denial of service attack as it requires greater processing than the usual packets, so the server has to allocate significant amount of resources to it. These packets derive their name Christmas Tree packets from its structure because all the fields of header are "lightened up" like a Christmas tree. In order to block XMAS type attack, following are used:

1) iptables -A INPUT -p tcp --tcp-flags ALL FIN, URG, PSH -j LOG --log-prefix "*Iptables* XMAS packet:" --log-level 7
2) iptables -A INPUT -p tcp --tcp-flags ALL FIN, URG, PSH -j DROP
3) iptables -A INPUT -p tcp --tcp-flags ALL ALL -j LOG --log-prefix "*Iptables* Xmas Tree type scanning: " --log-level 7
4) iptables -A INPUT -p tcp --tcp-flags ALL ALL -j DROP

The above rule no.2 matches TCP type packets with flags: FIN, URG, PSH enabled while rule no.4 matches TCP type packets where all the TCP flags enabled. Christmas tree packets are generated by using following commands in *Scapy*:

send (IP (dst = "192.170.1.119", src = RandIP ( ))/ TCP (dport=80, flags=X) , loop=1, inter=0.001) where X="FPU"; 0x0ff

5. IP Half Scan

In this attack, instead of establishing a complete TCP connection, the attacker only sends initial or final packets in order to avoid detection by port scan detectors. The attacker starts the SYN message with the server but do not complete it. IP Half scan attack is also known as half-open scans or FIN scans. There are various software (like Jakal) available that conducts half scans (stealth scan). In order to prevent this attack, TCP packets with only SYN and FIN flag enabled are filtered out using following:

1) iptables -A INPUT -p tcp --tcp-flags SYN, FIN SYN, FIN -j LOG --log-prefix "*Iptables* IP Half scan:" --log-level 7
2) iptables -A INPUT -p tcp --tcp-flags SYN, FIN SYN, FIN -j DROP

The attack is generated using the following script:

send (IP (dst="192.170.1.119", src = RandIP( ))/ TCP (dport=80, flags= "SF"), loop=1, inter=0.001)

6. Mail Bomb

Simple Mail Transfer Protocol is used for transferring emails. TCP is the underlying protocol used for SMTP on port number 25. This attack overwhelms a mail server, triggering denial of service to its users. It is achieved by sending an enormous number of e-mails to the server systems. So, to combat flood of emails, the traffic on this port is rate limited to 1000/sec using the following rules:

1) iptables -N Mailflood
2) iptables -A Mailflood -p tcp -m limit --limit 1000/s --limit-burst 1000 -j RETURN
3) iptables -A Mailflood -p tcp -j REJECT --reject-with tcp-reset
4) iptables -A INPUT -p tcp -m tcp --dport 25 -j Mailflood

The email flood is generated by using the following statement in *Scapy* or some bulk emailing services [21] [22] can be used to generate the email flood:

send (IP (dst = "192.170.1.119", src= RandIP ())/ TCP (dport=25, flags= "R"), loop=1, inter=0.001)

UDP BASED ATTACKS

7. UDP Flood

In this attack, the attacker sends off a hefty number of packets (UDP) to different ports of the targeted machine. If no application is listening on a port, the victim machine responds with an ICMP type 3 packets. This whole process of receiving, checking and responding results in victim being overwhelmed and unreachable.

The UDP flood can also be generated by exploiting different UDP services such as CHARGEN (measurement and debugging tool that generates a sequence of characters on receipt of every packet and operates on port 19). The attack is usually performed by sending a packet with a spoofed IP address to a system that has CHARGEN enabled. This can also be used along with ECHO operating on port 7 to generate massive traffic causing congestion. In addition to port 7, an attacker can also use other services such as the quotd (quote-of-the-day) functioning on port 17, or port 13 for the daytime service. These services echo packets on receipt of packets and when used in combination can cause service-denying congestion in the network. The UDP flood can be battled by rate limiting the incoming UDP traffic by following:

1) iptables -N UDPFLOOD
2) iptables -A UDPFLOOD -p udp -m limit --limit 2500/s --limit-burst 2500 -j RETURN
3) iptables -A UDPFLOOD -p udp -j LOG --log-prefix "*Iptables* UDPFLOOD: " --log-level 7
4) iptables -A UDPFLOOD -p udp -j REJECT
5) iptables -A INPUT -p udp -m state --state NEW -j UDPFLOOD

A new chain is created for new UDP traffic called UDPFLOOD. The rate of incoming packet is limited to 2500/sec. The UDP flood is generated by the following code in *Scapy*:
send (IP (dst = “192.170.1.119”, src = RandIP ( )) / UDP() , loop=1, inter=0.001)

8. UDP Snork Attack And Other UDP Services

It is analogous to the UDP flood but is performed by exploiting some selected services. It uses either the source port ECHO (on port 7) or CHARGEN (on port 9), with 135 as the destination port. The result is a flood of unnecessary transmissions which slows down the performance and can crash the involved systems. The different UDP services can be blocked by following:

1) iptables -A INPUT -p udp -m udp --dport X:X -j LOG --log-prefix “*Iptables* Port X" --log-level 7
2) iptables -A INPUT -p udp -m udp --dport X:X -j DROP

where X=7, 9, 13, 17 and 19

Attack is generated by following in *Scapy*:
send (IP (dst=“192.170.1.119”, src = RandIP ( ))/ UDP (dport=X) , loop=1, inter=0.001)

9. DNS DoS Attack

It is the most common UDP based attack. The difference of the size a DNS query and a response is exploited causing tying up the bandwidth of the network by counterfeit DNS queries. It is also called DNS Amplification Attack as the DNS servers are exploited to act as “amplifiers” to swell up the network traffic. Attack takes place by sending tiny queries to the DNS servers by spoofing the IP address of the targeted system. In response to these small sized queries, much larger DNS response is returned. A great number of responses will be returned at the same time if a number of requests are made to the server, thus, congesting the network and a DoS attack will take place. DNS service uses UDP and operates on port 53.The DNS DoS attack can be mitigated by rate limiting the DNS requests by following:

1) iptables -N DNSFLOOD
2) iptables -A DNSFLOOD -p udp -m limit --limit 1000/s --limit-burst 1000 -j RETURN
3) iptables -A DNSFLOOD -p tcp -m limit --limit 500/s --limit-burst 500 -j RETURN
4) iptables -A DNSFLOOD -p udp -j LOG --log-prefix “*Iptables* DNS DOS: " --log-level 7
5) iptables -A DNSFLOOD -p udp -j REJECT
6) iptables -A DNSFLOOD -p tcp -j LOG --log-prefix “*Iptables* DNS DOS: " --log-level 7
7) iptables -A DNSFLOOD -p tcp -j REJECT
8) iptables -A INPUT -p udp -m udp --sport 53 -j DNSFLOOD
9) iptables -A OUTPUT -p udp -m udp --dport 53 -j DNSFLOOD
10) iptables -A INPUT -p tcp -m tcp --sport 53 -j DNSFLOOD

DNS typically uses UDP for query and response. But in case the size of response packet is greater than 512 bytes, then TCP is used. Thus, the above rules are defined for both UDP and TCP. DNS DoS traffic can be generated by following code in Python:
answer = sr1 (IP (dst = “192.170.1.119”, src= RandIP ( )) / UDP (dport = 53) / DNS (rd = 1, qd = DNSQR (qname = “www.testrules.com”)) , verbose = 0, loop=1, inter=0.001)

ICMP BASED ATTACKS

10. Ping Flood

The Internet Control Message Protocol is used for transmitting error messages. The most common function implemented in ICMP is the ping function. It is used to determine if there exists a path between two hosts by using ECHO request and ECHO reply messages. While pinging a host, an ECHO request message is sent and in its response, an ECHO reply is received. In case no ECHO reply is received, this implies that either the other host is not available or does not support the ping functionality.
The ICMP flood targets the victim with ICMP Echo Request packets sent at a high-speed rate without waiting for a reply from the targeted device. This overwhelms the victim’s resources resulting in a consumption of both incoming and outgoing bandwidth. ICMP flood can also be referred as ping storm. So, the number of ECHO requests are rate limited to 1000/sec for combating the attack. The packets to be dropped are logged before so as to keep record of the dropped packets. The following rules are used to combat against this attack:

1) iptables -N ICMPFLOOD
2) iptables -A ICMPFLOOD -p icmp --icmp-type echo-request -m limit --limit 1000/second --limit-burst 1200 -j RETURN
3) iptables -A ICMPFLOOD -p icmp -j LOG --log-prefix “*Iptables* PING FLOOD: " --log-level 7
4) iptables -A ICMPFLOOD -p icmp -j REJECT
5) iptables -A INPUT -p icmp --icmp-type echo-request -m state --state NEW -j ICMPFLOOD

Ping requests are generated by following code:
send (IP (dst = “192.170.1.116”, src = RandIP ( )) / ICMP ( ), loop=1, inter=0.001)

11. Fraggle Attack:

It is a variation of the ICMP flood where instead of directly sending ping requests (ECHO requests) to the victim machine, ping (ECHO) requests with its IP address (address of the victim machine) are sent to a subnet, causing all the systems to answer with ECHO reply messages to the victim triggering flood of packets and thus consuming the bandwidth of the victim [23]. So, the replies are rate limited to a reasonable amount by following rules:

1) iptables -N Fragglestop
2) iptables -A Fragglestop -p icmp --icmp-type echo-reply -m limit --limit 300/s --limit-burst 300 -j RETURN
3) iptables -A Fragglestop -p icmp -j LOG --log-prefix "*Iptables* Fraggle Attack: " --log-level 7
4) iptables -A Fragglestop -p icmp -j REJECT
5) iptables -A INPUT -p icmp --icmp-type echo-reply -j Fragglestop

The attack is generated by sending spoofed ICMP ping requests with IP address of the firewall (192.170.1.119) to other computer systems on the network.

12. Ping Of Death

In this attack, ping requests with massive payload (>65536) are sent to server which causes it to hang. So, the following rule is formulated to only accept packets upto size of 65535 bytes.

iptables -A INPUT -p icmp -m length --length 0:65535 - j ACCEPT

Ping of death is generated by:

*send(fragment (IP (dst = "192.170.1.119",src= RandIP( )) / ICMP( ) / ("Y"*60000)), loop=1, inter=0.001)*

OTHER ATTACKS

13. Null Scanning

It is a technique of locating "open" ports (for communication) on a computer system and gathering as much as possible information from it. TCP and UDP uses well known ports for different services and applications. Once information about listening ports is available with the attacker, it can be used to exploit the server for the typical service is enabled on the server. The null scanning uses TCP packets with no flags enabled for gathering the information about the listening TCP ports. So, null scanning can be stopped by using following rules:

1) iptables -A INPUT -p tcp --tcp-flags ALL NONE -j LOG --log-prefix "*Iptables* Port Scanning:" --log-level 7
2) iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP

Port scanners can be generated by following:

result, unans = sr (IP (dst = "192.170.1.119") / TCP(dport= (1,1024), flags= " "))

result. nsummary (lfilter = lambda (s, r) : (r.haslayer (TCP) and (r.getlayer (TCP).flags&2)))

14. ARP Spoofing

ARP (Address Resolution Protocol) is a Network layer protocol used for resolving IP addresses to machine MAC addresses. This mapping of logical address to physical address is stored in a table called ARP table. When a packet arrives at a network device, it searches the ARP table to locate the physical address of that host. In case no entry is found, ARP request is broadcasted to all the systems to find the MAC address of targeted host. The system which identifies the broadcasted IP address as its own responds with the ARP reply. This results in updating of ARP table and packet is redirected to the corresponding MAC address. ARP being a stateless protocol is often exploited by attacker to perform ARP spoofing.

In this attack, an attacker sends fake ARP messages over the network which results in linking of MAC address of the attacker and the IP address of a valid system in the ARP table. In order to prevent this attack, following rules are laid to drop the packets that do not match the MAC address and the IP address of the validated machine and can be used for all machines:

1) iptables -A INPUT -s 192.170.1.116 -m mac ! --mac-source 64:51:06:3c:8e:b1 -j LOG --log-prefix "*Iptables* Spoofing: " --log-level 7
2) iptables -A INPUT -s 192.170.1.116 -m mac ! --mac-source 64:51:06:3c:8e:b1 -j DROP

Spoofed packets were generated by the machines using following:

send(IP (dst ="192.170.1.119", src ="192.170.1.116")) / TCP( ) , loop=1, inter=0.001)

15. DoS Attack

Here, the objective is to make a network unapproachable to intended users by producing a bulky traffic that can overwhelm the resources of the network. It can be accomplished by tiring the resources (like CPU, Memory) of the server so that the server remains busy for actual users or by consuming the bandwidth of the network by flooding the network. The following rules to mitigate this type of attack are based on rate limiting all the incoming traffic irrespective of the type of traffic:

1) iptables -N DoSflood
2) iptables -A INPUT -j DOSFLOOD
3) iptables -A DoSflood -m limit --limit 8000/s --limit-burst 10000 -j RETURN
4) iptables -A DoSflood -j LOG --log-prefix "*Iptables* DoS attack:" --log-level 7
5) iptables -A DoSflood -j DROP

In order to perform this attack, a combination of different types of flood traffic (TCP, UDP and ICMP) is generated using *Scapy* from the systems in the network.

V. Results and Discussion

The key parameters recorded for the fifteen network attacks as discussed in previous section are shown in table I and are graphically depicted in figure 2 to figure 17. *Iptables* has performed well in mitigating the network attacks with the average CPU utilization for processing and Logging with 2.42 % and 19.45 % respectively. Likewise, the average efficiency of *Iptables* is 94.92% with frame loss ratio of 5.25% while mitigating the attacks.

Figure 2 to figure 5 compares the % CPU utilized by *Iptables* during the attacks. The average CPU Utilization for TCP and UDP based attacks is 2.55% and 2.78% while for ICMP based attacks and other network attacks, it is 1.53% and 1.98% respectively. Thus, the CPU Utilization by *Iptables* lies in the range 1-4 % which makes *Iptables* suitable for mitigating the network attacks without making system slower. Similarly, figure 6 to figure 9 shows the % CPU utilized by *Iptables* for Log Generation during the attacks. It lies in the range of 10-21%. Although, it is a significant amount but it can be reduced by rate limiting the log generated during the course of attack.

Figure 10 to figure 13 compares the percentage of packets dropped by the *Iptables* for different network attacks. *Iptables* performs excellently while mitigating TCP based attacks with average frame loss ratio of only 1.73 % with traffic of approximately 11,00,000 packets. Though, a significant amount of packets were dropped in case of ping flood but the average frame loss ratio in case of UDP and ICMP based attacks is 3.89% and 5.11% respectively. This indicates a good performance (except ping flood) with traffic of approximately 10,00,000 packets. The poor performance of *Iptables* in ping flood may be due to the fact that ICMP packets require higher processing at network layer because both Internet Control Message Protocol and Internet Protocol work at network layer. Correspondingly, *Iptables* drops a major amount of packets i.e. 30.75% which depicts the lowest performance for DoS attack as it comprised of Ping flood, TCP flood and UDP flood.

The efficiency of the firewall while combating the network attacks is depicted in figure 14 to figure 17. The average efficiency of the *Iptables* in case of TCP based attacks is 98.25% while the average performance in case of UDP and ICMP based attacks is 96.06% and 94.79% respectively. Likewise, the average efficiency in case of other network attacks is 90.01%. Thus, it can be inferred that the *Iptables* has efficiently blocked all the network attacks and can be reproduced in real-life scenarios.

Table I. Observations for Key Parameters

ATTACKS	**Number Of Packets**				**Key Parameters**			
	SENT		**RECEIVED**		**% CPU Utilisation (Kernel)**	**% CPU Utilisation (Logging)**	**Frame Loss Ratio**	**Efficiency**
	Scapy	**DITG**	**NIC**	***Iptables***				
SYN	371K	770K	1141K	1120K	2.72	12.66	1.84	98.15
ACK	271K	745K	1026K	1016K	2.45	13.57	0.97	99.02
RESET	271K	735K	1022K	1001K	2.39	15.30	2.05	97.94
Half Scan	441K	727K	1168K	1146K	2.45	21.63	1.88	98.11
Xmas	271K	753K	1024K	1014K	2.64	15.59	0.97	99.02
Mail Bomb	446K	709K	1155K	1124K	2.65	14.78	2.68	97.31
Ping Flood	271K	721K	992K	840K	1.59	14.92	15.32	84.67
Fraggle	200K	723K	923K	918K	1.52	15.73	0.5	99.5
Ping Of Death	481K	1189K	1670K	1636K	1.49	10.01	2.03	97.96
UDP Flood	303K	710K	1013K	991K	2.47	10.07	2.07	97.82
Snork	455K	712K	1167K	1085K	2.06	21.47	7.03	92.96
DNS Dos	383K	743K	1126K	1097K	3.83	10.11	2.57	97.42
Null Scanning	216K	721K	937K	867K	1.51	15.42	7.5	93.02
DoS	543K	709K	1252K	867K	2.51	19.49	30.75	69.24
ARP Spoofing	301K	717K	1018K	1015K	1.93	20.22	0.33	99.5

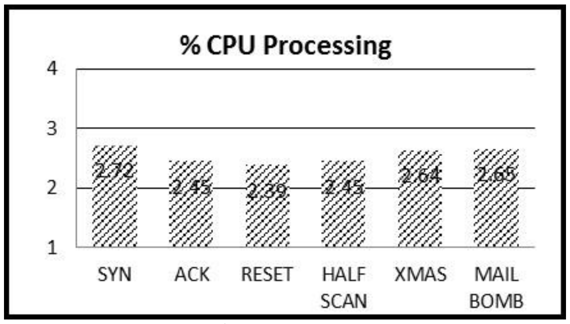

Figure 2. % CPU Utilized for Processing in TCP Based attacks

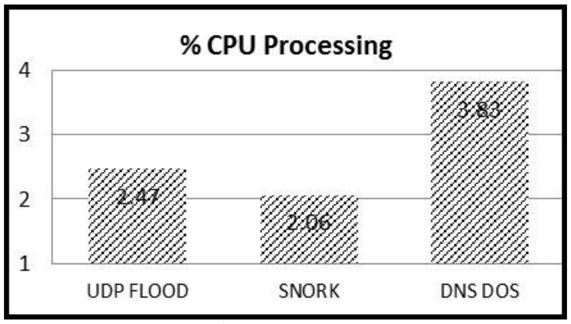

Figure 3. % CPU Utilized for Processing in UDP Based attacks

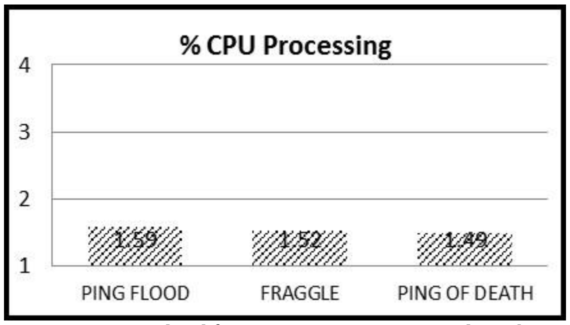

Figure 4. % CPU Utilized for Processing in ICMP Based attacks

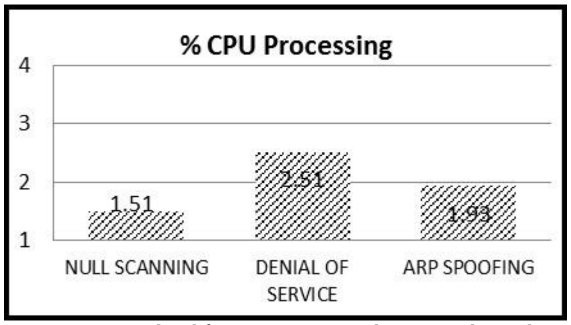

Figure 5. % CPU Utilized for Processing in Other Network attacks

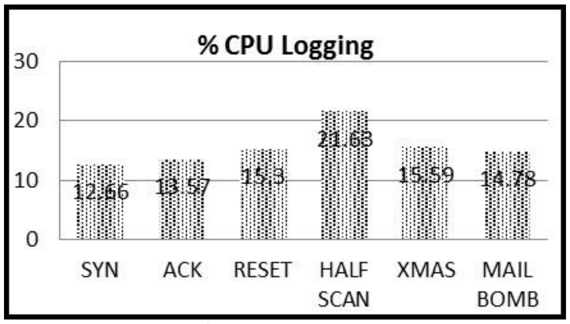

Figure 6. % CPU Utilized for Log Generation in TCP Based attack

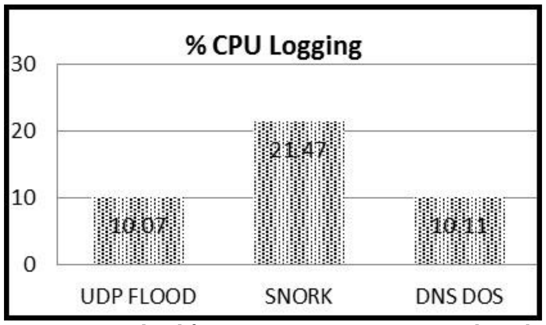

Figure 7. % CPU Utilized for Log Generation in UDP Based attack

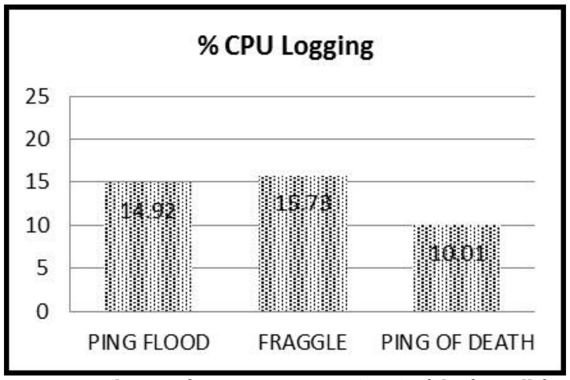

Figure 8. CPU Utilization for Log Generation (in %) of the firewall for ICMP Based attacks

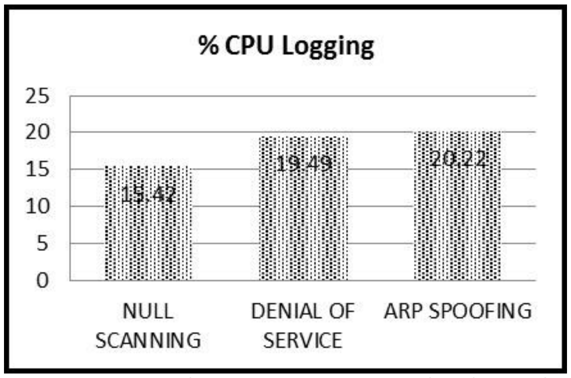

Figure 9. CPU Utilization for Log Generation (in %) of the firewall for Other Network Based attacks

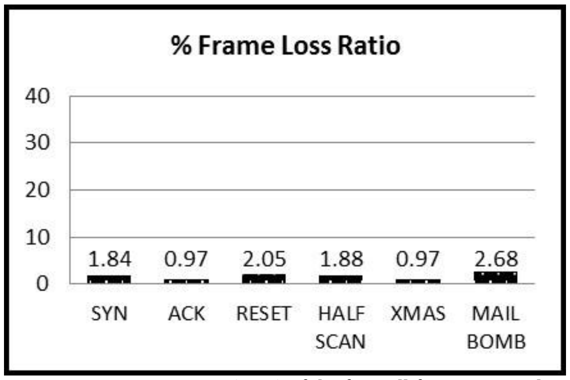

Figure 10. Frame Loss Ratio (in %) of the firewall for TCP Based attacks

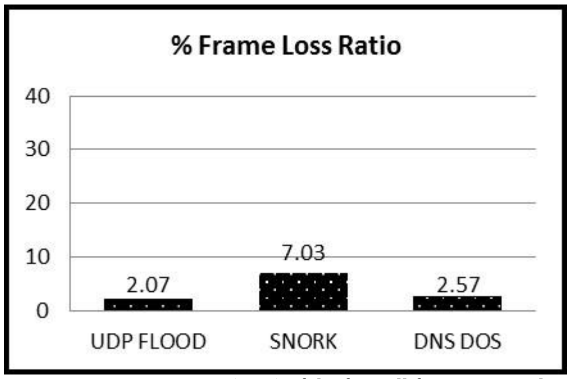

Figure 11. Frame Loss Ratio (in %) of the firewall for UDP Based attacks

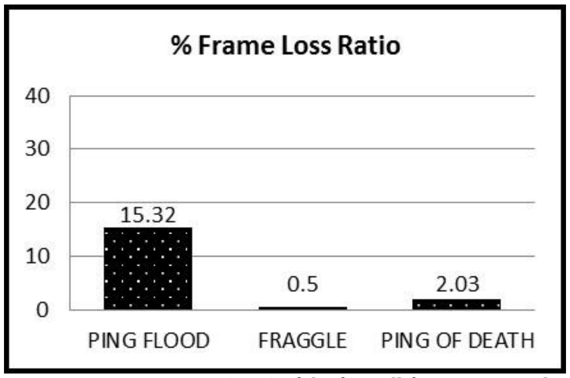

Figure 12. Frame Loss Ratio (in %) of the firewall for ICMP Based attacks

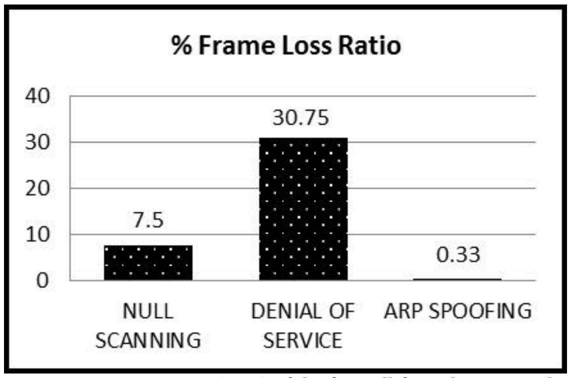

Figure 13. Frame Loss Ratio (in %) of the firewall for Other Network Based attacks

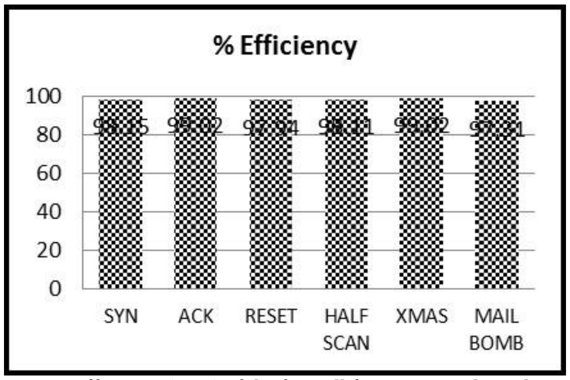

Figure 14. Efficiency (in %) of the firewall for TCP Based attacks

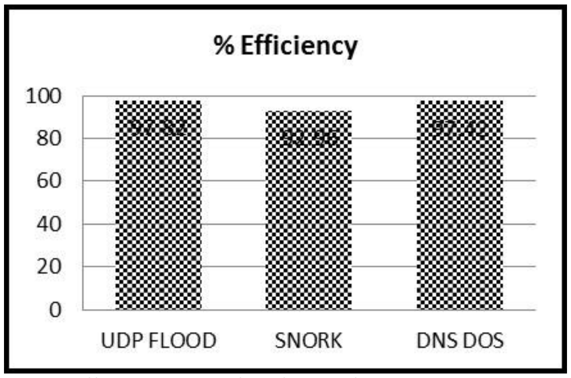

Figure 15. Efficiency (in %) of the firewall for UDP Based attacks

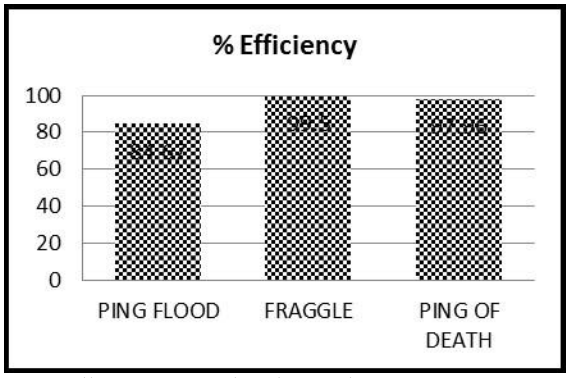

Figure 16. Efficiency (in %) of the firewall for ICMP Based attacks

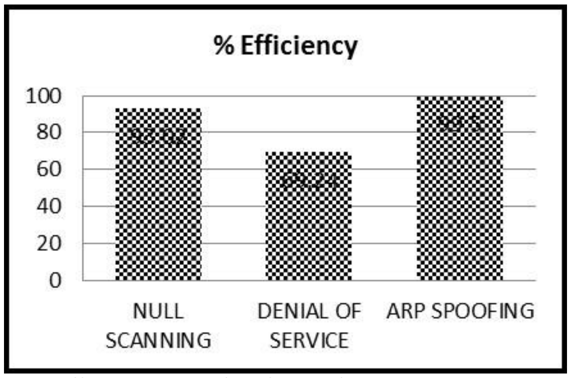

Figure 17. Efficiency (in %) of the firewall for Other Network attacks

VI. Conclusion and Future Scope

The research goal of this paper was to recommend new customised *Iptables* rules for mitigating fifteen network attacks. Correspondingly, the performance of *Iptables* was investigated by performing real time experiments and the behaviour of the firewall was observed under network attack along with normal traffic. This was recorded in the terms of CPU utilisation for processing and Logs generation, Frame Loss Ratio and Efficiency. To achieve the research goals, various rules were incorporated in the shell script to combat the network attacks. The designed script successfully Allow or Deny the traffic based on the type of packet (attack or normal). Therefore, this makes it possible to protect the system from a wide variety of network hazards and can be used for the providing security to the network.

The results clearly indicate that the *Iptables* performs really well in case of network attacks but drops a considerate amount of packets even though the processing is done by kernel. This could be due to linear searching of the list of rules which can potentially slow down the working of the firewall. To overcome this problem, Packet Classification algorithms can be implemented to improve the performance of the *Iptables*.

References

[1] W. Su and J. Xu, "Performance Evaluations of Cisco ASA and Linux *Iptables* Firewall Solutions," May 2013.

[2] "*Netfilter* Project," [Online]. Available: www.netfilter.org. [Accessed 01 October 2017].

[3] "*Iptables*," 2017. [Online]. Available: http://www.*Iptables*.info/en/structure-of-*Iptables*.html. [Accessed 7 September 2017].

[4] "Monitoring and Tuning the Linux Networking Stack: Receiving Data," May 2016. [Online]. Available: https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/. [Accessed 27 September 2017].

[5] R. K. C. Chang, "Defending against Flooding-Based Distributed Denial-of-Service Attacks:A Tutorial," IEEE Communications Magazine, pp. 42-51, October 2002.

[6] "*Scapy* and its Documentation," 6 Nov 2017. [Online]. Available: https://scapy.readthedocs.io/en/latest/. [Accessed 22 October 2017].

[7] A. Botta, W. Donato, A. Dainotti, S. Avallone and A. Pescapé. [Online]. Available:http://traffic.comics.unina.it/software/ITG/manual/. [Accessed 16 November 2017].

[8] O. Andreasson, 2001. [Online]. Available: http://onz.es/IpTables%20Tutorial.pdf. [Accessed 5 October 2017].

[9] M. Rash, Linux Firewalls- Attack Detection and Response, 2007.

[10] K. Chatterjee, "Design and Development of a Framework to Mitigate DoS/DDoS Attacks Using IPtables Firewall," International Journal of Computer Science and Telecommunications , vol. 4, no. 3, pp. 67-72, March 2013.

[11] B. Sharma and K. Bajaj, "Packet Filtering using IP Tables in Linux," International Journal of Computer Science Issues(IJCSI), vol. 8, no. 4, pp. 320-325, July 2011.

[12] B. Q. M. AL-Musawi, "Mitigating DoS/DDoS Attacks Using *Iptables*," International Journal of Engineering & Technology IJET-IJENS, vol. 12, no. 03, pp. 101-111, June 2012.

[13] S. Mirzaie, A. K. Elyato and D. A. Sarram, "Preventing of SYN Flood attack with iptables Firewall," in 2010 Second International Conference on Communication Software and Networks.

[14] M. Šimon, L. Huraj and M. Čerňanský, "Performance Evaluations of IPTables Firewall Solutions under DDoS attacks," Journal of Applied Mathematics Statistics and Informatics (JAMSI), vol. 11, no. 2, pp. 35-45, 2015.

[15] A. Balobaid, W. Alawad and H. Aljasim, "A Study on the Impacts of DoS and DDoS Attacks on Cloud and Mitigation Techniques," in 2016 International Conference on Computing, Analytics and Security Trends (CAST), College of Engineering Pune, India. Dec 19-21, 2016, 2016.

[16] M. Y. Arafat, M. M. Alam and F. Ahmed, "A Realistic Approach and Mitigation Techniques for Amplifying DDOS Attack on DNS," in Proceedings of 10th Global Engineering, Science and Technology Conference, BIAM Foundation, Dhaka, Bangladesh, 2-3 January, 2015.

[17] K. Salah, K. Elbadawi and R. Boutaba, "Performance Modelling and Analysis of Network Firewalls," IEEE Transactions on Network and Service Management, vol. 9, no. 1, pp. 12-20, March 2012.

[18] T. Hayajneh, B. J. Mohd, A. Itradat and A. N. Quttoum, "Performance and Information Security Evaluation with Firewalls," International Journal of Security and Its Applications, vol. 7, no. 6, pp. 355-372, 2013.

[19] S. M. Aaqib, "To Analyze Performance, Scalability & Security Mechanisms of Apache Web Server Vis-a-vis with contemporary Web Servers," University of Jammu. Available: [http://hdl.handle.net/10603/65175], Jammu, 2014.

[20] S. Mishra, S. Sonavane and A. Gupta, "Study of Traffic Generation Tools," International Journal of Advanced Research in Computer and Communication Engineering, (IJARCCE) Vol. 4, Issue 6, June 2015.

[21] "BULK Email," BESI Marketing Solutions, [Online]. Available: http://www.bulkemailsmsindia.com/. [Accessed 12 December 2017].

[22] "Bulk Email service," Mail Marketer, [Online]. Available: http://mailmarketer.in/.

[23] R. J. Shimonski, D. L. Shinder, T. W. Shinder and A. C.-. Henmi, Best Damn Firewall Book Period, Syngress, ISBN: 1-931836-90-6, 2003.

[24] S. Sharma, Y. Verma and A. Nadda, “Information Security: Cyber Security Challenges,” International Journal of Scientific Research in Computer Science and Engineering, Vol.7, Issue.1, pp.10-15, February (2019)

[25] P. Santra, “An Expert Forensic Investigation System for Detecting Malicious Attacks and Identifying Attackers in Cloud Environment,” International Journal of Scientific Research in Network Security and Communication, Volume-6, Issue-5, October 2018.

Authors Profile

Ms. Nikita Gandotra received her B.E. in Computer Engineering from the University of Jammu in 2010. She has also completed M. Tech. in Computer Science from University of Jammu (India) in 2014 which is one of the most prestigious universities in India and ranked 51st in University Category in NIRF-2018 in India. She is currently pursuing Doctorate of Philosophy (PhD) from Department of Computer Science & IT, University of Jammu. She is working on Packet Classification and security management. Her areas of Interest include communication and networking, network security, network and systems management, data structures and adhoc networks.

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 Six PhD scholars.