Firewall Algorithm Approach for Diminution of DDOS Attack on Cloud

Vaishnavi Gawale[1] Varshapriya J N[2]
[1]P.G Student [2]Associate Professor
[1,2]Department of Computer Engineering & Information Technology
[1,2]Veermata Jijabai Technological Institute, Mumbai, India

Abstract— Although cloud computing practice has been increased extensively due to its more flexible and expanded infrastructure resources, using one of its major virtual (machine) instance service has become the centre of activity. These virtual instances are abstracted from public as well as private cloud and though the cloud security is treated as a prime issue still these cloud instance are vulnerable to DDOS attack. DDOS attack is second most influential and powerful attack seen on the cloud platform. The proposed solution emphasizes on preventing cloud instance from DDOS attack by implementing a new Firewall Algorithm. Here, the packets are categorized either as malicious or genuine. If packets are identifies as coming from attacker, then the respective IP addresses are blocked and thus, resulting in mitigating of DDOS attack. This paper frames the detailed execution of algorithm, attack study and staging results using statistics and graphs.

Key words: Cloud Computing, DDOS Attack, Firewall Algorithm, Openstack, Cloud Instance

I. INTRODUCTION

Cloud Computing works as a space which starts from computation and ends with providing services, data storage, operating system and applications [1]. Cloud can be further explained as the 'virtual' resource which provides computing services for independent software development. It allows users to deploy and use their own virtual machine on cloud which is called as 'cloud instance'. This service is termed as an Amazon Elastic Compute Cloud by the Amazon Web Services [2].

The main aim of Denial of Service (DOS) attack is to over saturate the range of victim machines and overwhelm its capacity making them unavailable for users [3]. If the attack performed from single source then it is categorized as DOS attack while, the attack coming from distributed sources is called as Distributed Denial of Service (DDOS) attack.

Distributed Denial of Service (DDOS) HTTP Flood attack is a volumetric attack that overloads the targeted machine with HTTP request resulting into failure of machine. HTTP Flood attack is categorized into two types: HTTP GET attack sends multiple requests for images, files and applications, while HTTP POST attack is related to the data that is being stored or fetched through web pages [4].Attackers are in search of different techniques to impose the attack and try to crack the security system. Meanwhile, the researchers update their methodologies for preventing these attacks [5]. Risk assessment can be made depending upon below five aspects – source, vector, target, impact and defense [6].

Though different methods are followed for mitigation of DDOS attack, still the single server virtual machine are lacking for its security. Therefore, the proposed model in this paper prevents the DDOS HTTP Flood attack by implementing a new Firewall Algorithm. This algorithm is structured in such a ways that by running a single bash script it will stop the illegitimate packets request. This algorithm works into two modules. The detection module analyzes and categorizes the packet whether they are malicious or genuine, while prevention module terminates the unwanted packets. We have created the private cloud environment by using Openstack, launched a virtual machine instance on Openstack, performed a DDOS HTTP Flood attack and prevented it by using proposed Firewall Algorithm.

This paper is organized as follows: Section 2 describes our related work, Section 3 consist information about cloud environment setup, and Section 4 is of our proposed model with two sub sections – Categorizing packets and Algorithm. Its performance evaluation is done in Section 5 with statistics and graph. Lastly, we end this research with conclusion and future scope in Section 6.

II. RELATED WORK

Over a decade many of the researchers came up with different DDOS detection and prevention methodologies and succeeded in improvising them.

DOS attack occurs as our system fails to process each and every request due to its less capacity and resource. A dynamic resource allocation method was proposed against the DDOS attack in [7]. Here, cloud service consists of queue and Intrusion Prevention system (IPs). The requests are buffered in queue and passed through IPs for its detection and prevention. When DDOS attack is performed, the queue is flooded immensely with requests. Hence, to process each request more resources are allocated and Intrusion Prevention systems are cloned dynamically and processed each request separately.

CS_DDOS system was discovered by Aqeel Sahi in [8] where, incoming packets are detected and prevented by using different classifier model system. Classifiers such as LS—SVM and K-nearest are used to test the CS_DDOS systems performance. In [9] a new form of DDOS attack is presented. There is a monitoring agent located in subnet of the cloud infrastructure. When system find available bandwidth is getting degraded it sends UDP packets asking for help to monitoring agent. When monitoring agent receives notification it calls the migration application. It creates a new standby subnet and transfer the application to new network.

A group testing approach proposed byYing Xuan in [10] which is deployed on backend server. The GT model measures the size and distributes the client request to different virtual machines meanwhile, processing it separately. Many DDOS prevention methods fail to fast and early detection of attack. With the objective of this, Bio-Inspired Anomaly based application layer DDOS attack detection system is proposed in [11].Bio- inspired bat algorithm is experimented by using bench marking dataset.

In [12], analytical queuing model based on embedded Markov chain is proposed to examine the performance of Rule-based firewall. It analyzes the incoming and outgoing packets through the network. Performance of rule based firewall is simulated both at the time of normal traffic and attacking packets. The paper [13] explores cloud security challenges and proposes cloud native scalable security solution.

Rank Correlation based Detection (RCD) was proposed by Wei Wei in [14]. It inspects basic traffic pattern at the victim side and propose detection method for calculating suspicious flaws in the network. In [15] , Confidence-based method and HTTP Flooding detection method was proposed to overcome DDOS attack .Here, resources were allocated dynamically so that it automatically synchronize with available resource. Then Confidence Based Filtering score is calculated to discard the packets. In [16], New Cracking algorithm was implemented where MAC generator distinguish the attacking IP address.

III. CLOUD ENVIRONMENT

We have created a private cloud environment by using Openstack to implement and test our proposed system. Openstack enlarged into large community with more than 9000 organizations and 500 companies [17]. Openstack can be defined as open source software platform which behaves as a cloud and provides computing facilities specially infrastructure service, where customers can deploy their own instance (virtual machine) on cloud [18].

Openstack is made up of different components such as Compute, Network and Storage etc. These components can be installed separately in different PC's or all within one single PC. We have used three different PC's of Centos operating system with 8GB RAM configuration for making it Three Node Openstack installation – Controller node, Compute node and Network node. [19].

As proposed algorithm is written in bash script hence, for implementation we will be requiring Ubuntu operating system. Therefore, launching the Ubuntu instance will be the next step after installation of Openstack [20].

IV. PROPOSED SYSTEM

When communication between two PC's take place, streams of packets are sent or received by an IP address. Our algorithm analyzes the packet and categorizes whether to pass it or terminate it. Once it is categorized, prevention part of algorithm blocks the respective IP addresses of wrong packets and allows right packets to communicate. Below Figure (1) shows the block diagram of system.

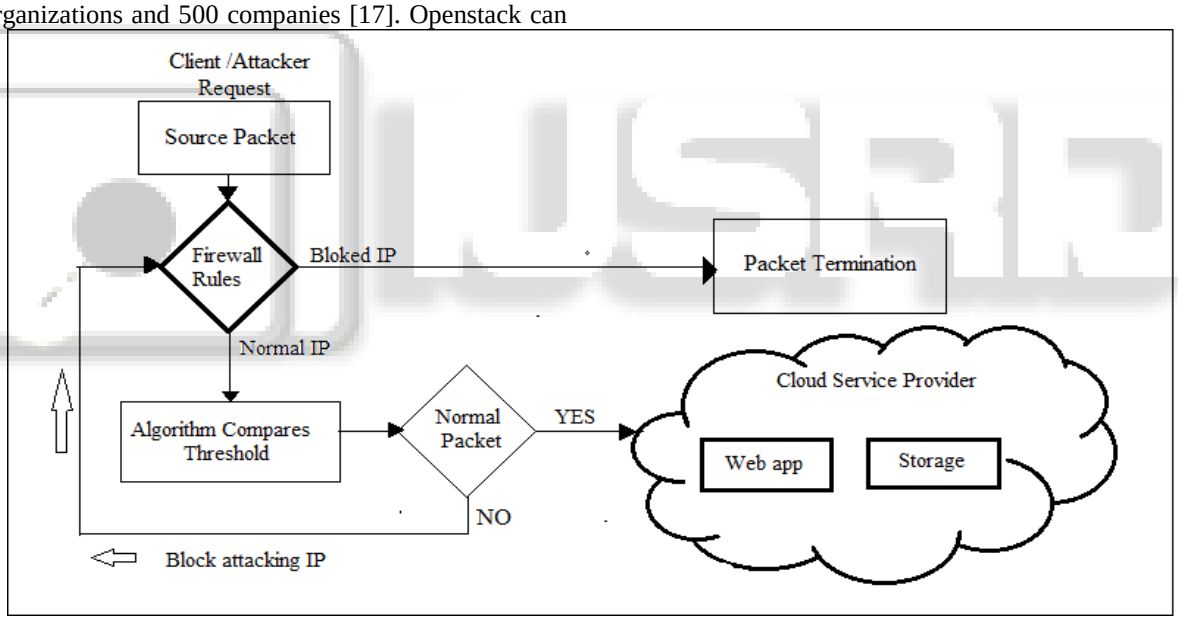


Fig. 1: Proposed system block diagram

A. Categorizing Packets

We request the target machine by using ping command. 172.18.31.61 is the IP address of our target machine. After sending the normal packets the target machine accepts and replies our request as shown in following Figure (2).

```
root@kali:~# ping 172.18.31.61
PING 172.18.31.61 (172.18.31.61) 56(84) bytes of data.
64 bytes from 172.18.31.61: icmp_seq=1 ttl=64 time=0.489 ms
64 bytes from 172.18.31.61: icmp_seq=2 ttl=64 time=0.640 ms
64 bytes from 172.18.31.61: icmp_seq=3 ttl=64 time=0.776 ms
^C
--- 172.18.31.61 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.489/0.635/0.776/0.117 ms
```

Fig. 2: Ping command

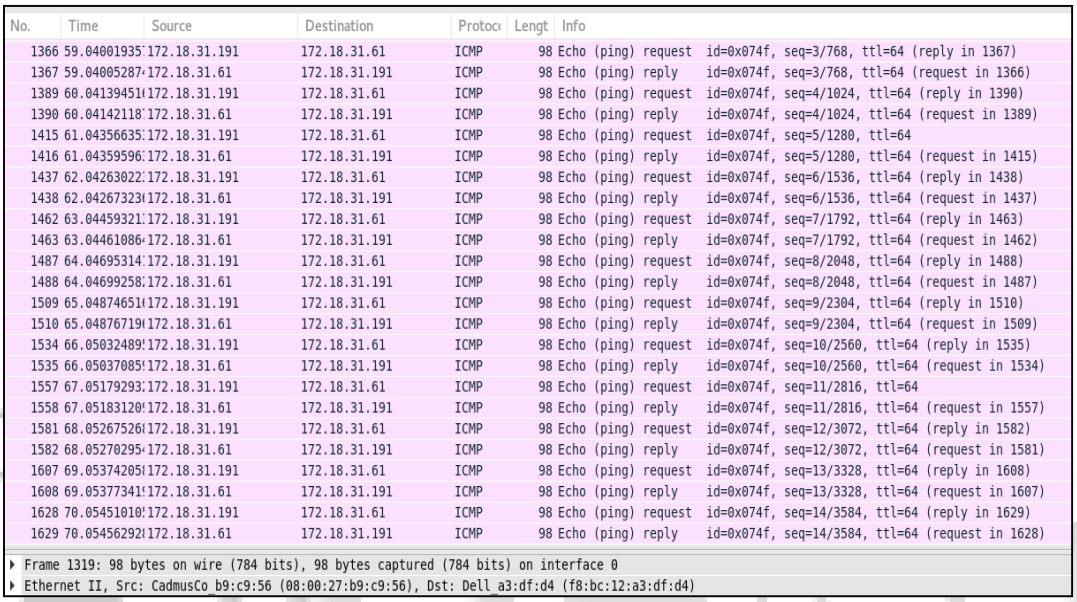
For categorizing whether the packet is genuine or came from an attacker, first we have to analyze the packet contents and statistics. For analyzing those things we are using Wireshark tool. When we send a normal request for an application or any particular IP address, the average packets per second sent or received can be notified within range of 0 - 40 packets/second. But in DDOS HTTP Flood attack packets captured starts from range 1000 –2000 per second. Below Figure (3) (a) shows the output of wireshark packet of normal request and its reply while, in Figure (3) (b) shows the statistic of average packets captured as 22.644 packets/second.

We then performed DDOS HTTP Flood attack by using Low Orbit Ion Cannon (LOIC) [21]. This tool starts flooding the targeted IP address until we stop it manually. It

gives us option to choose the speed of overwhelming packets on the targeted machine.

In Figure (4) (a) we can see the scenario after performing the attack where, enormous packets are bombarded on target machine with random IP addresses. And its statistics is calculated in Figure (4) (b) where average packets are 38952 packets/second. Comparing below two statistic figures we can clearly understand the huge difference between these two scenarios. Thus, using these values as a threshold point, classifying the packets origin will be further step in our algorithm which is explained precisely in next section.

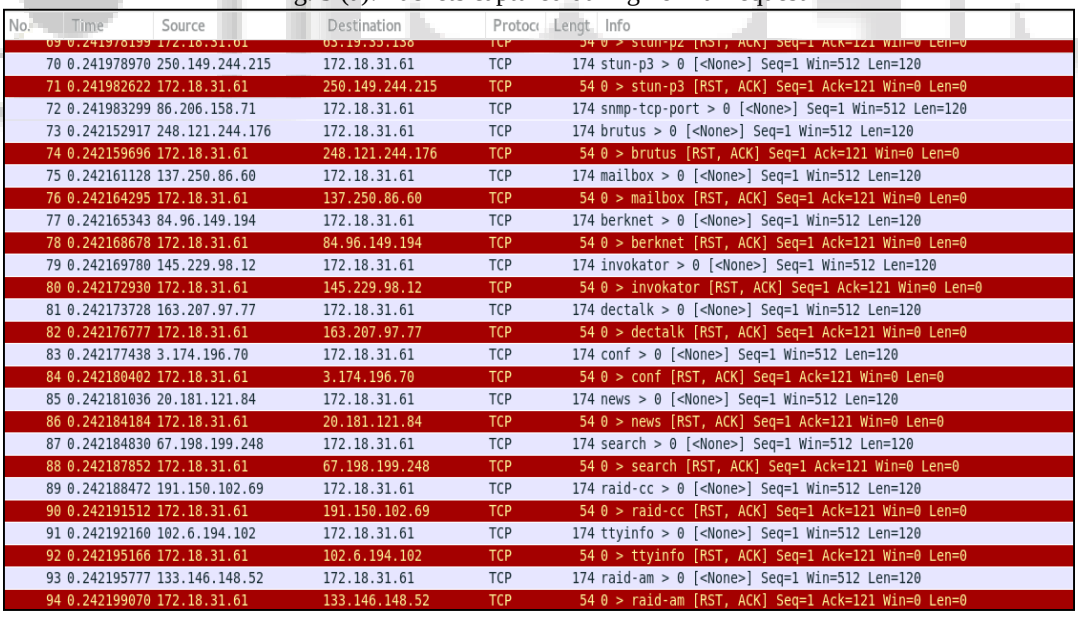
Now all these statistics are resulted through graphical tool. For making this result available through the programming, we are using Tshark tool – a command line version of Wireshark. [22]. It works as a command with providing various options of writing the live captured data to a specific file format and also reading the data from same format. It allows the user to choose the field whose output is supposed to be printed. It is also featured with auto stop condition based on time duration, number of records inserted, and file size. By using all these functionalities, Firewall Algorithm is structured, and explained below.



No.	Time	Source	Destination	Protoc	Lengt	Info
1366	59.04001935	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=3/768, ttl=64 (reply in 1367)
1367	59.04005287	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=3/768, ttl=64 (request in 1366)
1389	60.04139451	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=4/1024, ttl=64 (reply in 1390)
1390	60.04142118	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=4/1024, ttl=64 (request in 1389)
1415	61.04356635	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=5/1280, ttl=64
1416	61.04359596	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=5/1280, ttl=64 (request in 1415)
1437	62.04263022	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=6/1536, ttl=64 (reply in 1438)
1438	62.04267323	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=6/1536, ttl=64 (request in 1437)
1462	63.04459321	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=7/1792, ttl=64 (reply in 1463)
1463	63.04461086	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=7/1792, ttl=64 (request in 1462)
1487	64.04695314	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=8/2048, ttl=64 (reply in 1488)
1488	64.04699258	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=8/2048, ttl=64 (request in 1487)
1509	65.04874651	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=9/2304, ttl=64 (reply in 1510)
1510	65.04876719	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=9/2304, ttl=64 (request in 1509)
1534	66.05032489	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=10/2560, ttl=64 (reply in 1535)
1535	66.05037085	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=10/2560, ttl=64 (request in 1534)
1557	67.05179293	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=11/2816, ttl=64
1558	67.05183120	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=11/2816, ttl=64 (request in 1557)
1581	68.05267526	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=12/3072, ttl=64 (reply in 1582)
1582	68.05270295	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=12/3072, ttl=64 (request in 1581)
1607	69.05374205	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=13/3328, ttl=64 (reply in 1608)
1608	69.05377341	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=13/3328, ttl=64 (request in 1607)
1628	70.05451010	172.18.31.191	172.18.31.61	ICMP	98	Echo (ping) request id=0x074f, seq=14/3584, ttl=64 (reply in 1629)
1629	70.05456292	172.18.31.61	172.18.31.191	ICMP	98	Echo (ping) reply id=0x074f, seq=14/3584, ttl=64 (request in 1628)

▸ Frame 1319: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▸ Ethernet II, Src: CadmusCo_b9:c9:56 (08:00:27:b9:c9:56), Dst: Dell_a3:df:d4 (f8:bc:12:a3:df:d4)

Fig. 3 (a): Packets captured during normal request



No.	Time	Source	Destination	Protoc	Lengt	Info
70	0.241978970	250.149.244.215	172.18.31.61	TCP	174	stun-p3 > 0 [<None>] Seq=1 Win=512 Len=120
71	0.241982622	172.18.31.61	250.149.244.215	TCP	54	0 > stun-p3 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
72	0.241983299	86.206.158.71	172.18.31.61	TCP	174	snmp-tcp-port > 0 [<None>] Seq=1 Win=512 Len=120
73	0.242152917	248.121.244.176	172.18.31.61	TCP	174	brutus > 0 [<None>] Seq=1 Win=512 Len=120
74	0.242159696	172.18.31.61	248.121.244.176	TCP	54	0 > brutus [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
75	0.242161128	137.250.86.60	172.18.31.61	TCP	174	mailbox > 0 [<None>] Seq=1 Win=512 Len=120
76	0.242164295	172.18.31.61	137.250.86.60	TCP	54	0 > mailbox [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
77	0.242165343	84.96.149.194	172.18.31.61	TCP	174	berknet > 0 [<None>] Seq=1 Win=512 Len=120
78	0.242168678	172.18.31.61	84.96.149.194	TCP	54	0 > berknet [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
79	0.242169780	145.229.98.12	172.18.31.61	TCP	174	invokator > 0 [<None>] Seq=1 Win=512 Len=120
80	0.242172930	172.18.31.61	145.229.98.12	TCP	54	0 > invokator [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
81	0.242173728	163.207.97.77	172.18.31.61	TCP	174	dectalk > 0 [<None>] Seq=1 Win=512 Len=120
82	0.242176777	172.18.31.61	163.207.97.77	TCP	54	0 > dectalk [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
83	0.242177438	3.174.196.70	172.18.31.61	TCP	174	conf > 0 [<None>] Seq=1 Win=512 Len=120
84	0.242180402	172.18.31.61	3.174.196.70	TCP	54	0 > conf [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
85	0.242181036	20.181.121.84	172.18.31.61	TCP	174	news > 0 [<None>] Seq=1 Win=512 Len=120
86	0.242184184	172.18.31.61	20.181.121.84	TCP	54	0 > news [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
87	0.242184830	67.198.199.248	172.18.31.61	TCP	174	search > 0 [<None>] Seq=1 Win=512 Len=120
88	0.242187852	172.18.31.61	67.198.199.248	TCP	54	0 > search [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
89	0.242188472	191.150.102.69	172.18.31.61	TCP	174	raid-cc > 0 [<None>] Seq=1 Win=512 Len=120
90	0.242191512	172.18.31.61	191.150.102.69	TCP	54	0 > raid-cc [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
91	0.242192160	102.6.194.102	172.18.31.61	TCP	174	ttyinfo > 0 [<None>] Seq=1 Win=512 Len=120
92	0.242195166	172.18.31.61	102.6.194.102	TCP	54	0 > ttyinfo [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
93	0.242195777	133.146.148.52	172.18.31.61	TCP	174	raid-am > 0 [<None>] Seq=1 Win=512 Len=120
94	0.242199070	172.18.31.61	133.146.148.52	TCP	54	0 > raid-am [RST, ACK] Seq=1 Ack=121 Win=0 Len=0

Fig. 4 (a): Packet captured during DDOS attack

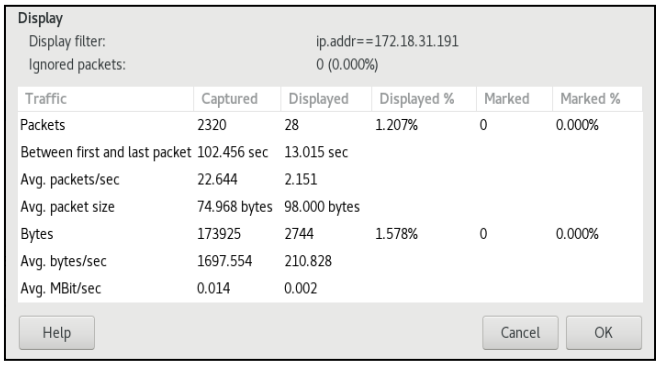


Fig. 3(b): Statistics of normal packets.

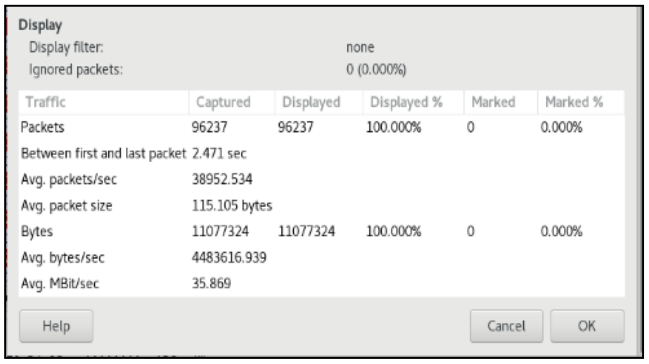


Fig. 4(b): Statistics of attacking packets

B. Algorithm

Detection.sh:-

```
#!/bin/bash
while true
 do
  tshark –i p4p1 –a duration:2  -a filesize:120
                          –w /home/ip.pcap
   tshark –r /home/ip.pcap –T fields
                          –e ip.src > output.txt
  ./Prevention.sh
> /home/ip.pcap
> /output.txt
done
```

Prevention.sh:-

```
#!/bin/bash
var=$ ( capinfos –x /home/ip.pcap )
packet=$(echo $var | cut –d ‘ ’  -f7)
 echo $packet
if (( $packet > 40 ));  then
    head -5 output.txt |
  while read a;  do
    iptables –A INPUT –s $a  -j  DROP
    echo “blocked  - $a”
  done
 fi
```

Where:

- i is interface
- a is auto condition
- w write data to file
- r read data from file
- T to use display filters
- e specific field
- x is capture statistic
- A is append rule
- s is source address

For executing the above file we only need to run the Detechtion.sh bash script on our Ubuntu launched instance of Openstack. We can run this script by just typing “./Detection.sh” in terminal (command prompt of Ubuntu Operating system). Prevention.sh script will be automatically called through the Detection.sh script.

In Detection.sh script the first tshark command captures data from p4p1 interface and writes data to ip.pcap file (.pcap is extension file format for captured data). This command stops capturing data after 2 seconds and when the file size of ip.pcap reached 120kb.

As tshark command which works like tcpdumb command, it dumps all the information regarding captured packets to ip.pcap file. Hence, to extract only Source IP address from that file we are using second tshark command that reads data from ip.pcap and write it to text file called output.txt. In next step it starts executing the Prevention.sh script.

Capinfos –x command store the output of Average packet/second[23] in variable var as like-

1) “Average packet rate: 38952 packets/sec”

By using the second line of script, we are cutting down the output of variable var and only storing the field 7(-f7) that is “38952” into packet variable and printing it. Next most important part is to compare the packets/sec value with the threshold value. We consider here 40 as our threshold value, if value of $packet exceeds 40 then the IP address stored in output.txt are considered to be attackers. They are traversed in loop and are blocked using Iptables Firewall rules. It appends the rule and DROP the packets by specifying it as a source address.

After executing all the lines from Prevention.sh script, the control transfer back to previous script. The last two lines of Detection.sh script format the ip.pcap and output.txt. The reason behind using these commands is to delete the previous captured data and update those files with the new data each time when script is run. All the commands in the script are executed in loop to update the files automatically for every 2 second and continuously captures the live data in background.

The line “echo blocked - $a” is for sake of user understanding that prints which IP addresses are blocked. The performance and accuracy of above algorithm can be examined in the following section. It shows number of capturing packets and blocked IP addresses.

V. PERFORMANCE EVALUATION

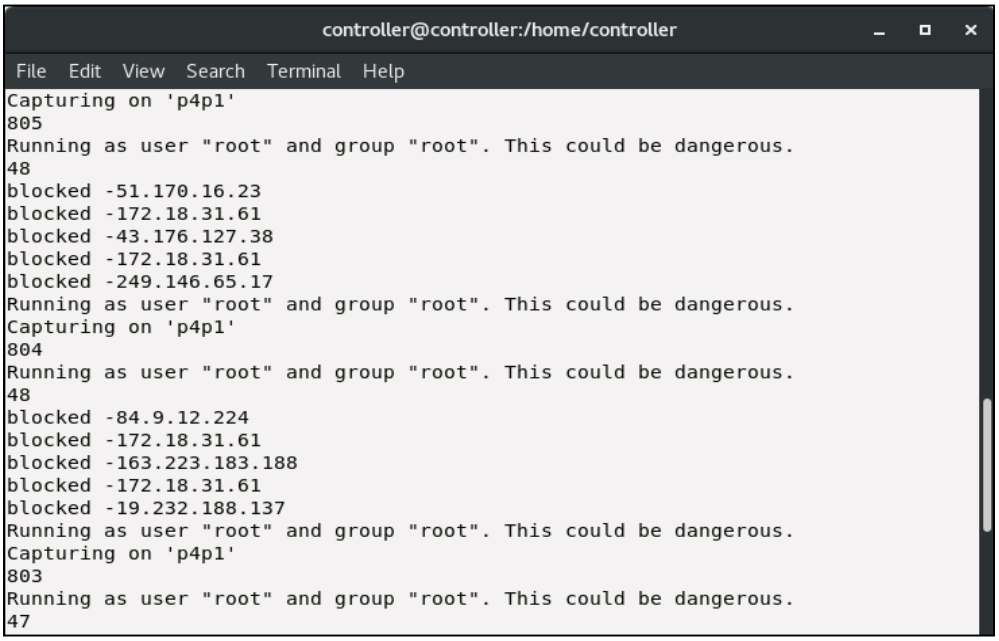


Fig. 5: Output of proposed algorithm

Here, this output is traced from terminal while the script was executing in background. We can verify whether these IP addresess are blocked or not by using the Firewall commands – sudo iptables –L

It displays all the rules added and blocked IP addresses by Firewall [24]. In order to verify again, we can test by pinging the target machine from attacker machine. As attacker's IP address has been blocked by the targeted machine, it doesn't reply any request as shown in below Figure (6).

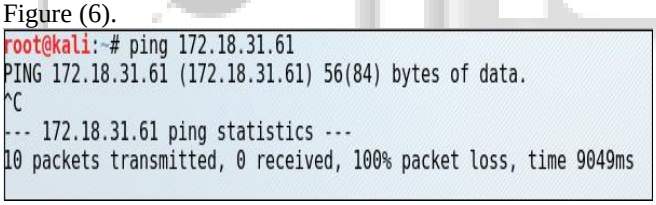


Fig. 6: Output of ping after attack

We can see packets are transmitted, but 0 packets are received resulting 100% packet loss. Thus, from above results we claim that our proposed Firewall algorithm approach works efficiently, identifies the attacking packets and prevent from DDOS attack.

VI. CONCLUSION

Cloud Computing has been a widespread technology in many sectors. Hence its security has become a major issue now-a-days. Though different mechanisms were proposed for prevention of second most powerful DDOS attack, still the single server machine were vulnerable. This Firewall algorithm approach was proposed to prevent the virtual instance of cloud by identifying the malicious packets sent by an attacker and terminating those packets by blocking their respective IP addresses. The algorithm is built in two phase-Detection and Prevention. In Detection phase, algorithm captures the statistic and compares the output with default set threshold value. Based on this comparison it categorize whether the packet is normal or malicious. In Prevention phase it blocks the respective IP addresses of malicious packet by using the Firewall Rules. The results presented in Performance Evaluation section shows the expected working of proposed algorithm. Thus, the Algorithm effectively improves the security from DDOS attack.

In future, we aim to extend to Firewall Algorithm by using the IPsets for storing large amount of firewall rules and improvising the algorithm for getting more faster results.

REFERENCES

[1] Santosh Kumar and R. H. Goudar, "Cloud Computing – Research Issues, Challenges, Architecture, Platforms and Applications: A Survey," International Journal of Future Computer and Communication,Vol. 1, No.4, December 2012.
[2] Shufen Zhang, Shuai Zhang, Xuebin Chen and Shangzhuo Wu, "Analysis and Research of Cloud Computing System Instance," Second International Conference on Future Networks,2010.
[3] DOS attck information – <http://www.cloudfare.com/ /learning/ddos/glossary/denial-of-service/.
[4] DDOS attack Information – http://www.cloudflare.com.
[5] Jelena Mirkovic and Peter Reiher "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms" ACM SIGCOMM Computer Communications Review, Volume 34, Number 2: April 2004
[6] Nina Viktoria Juliadotter and Kim-Kwang Raymond Choo Digtal Integrated Circuits— "Cloud Attack and Risk Assessment Taxonomy" Ieee Cloud Computing Published By The Ieee Computer Society 2325-6095/15/$31 .00 © IEEE,2015.
[7] Shui Yu, Yonghong Tian, Song Guo and Dapeng Oliver Wu, "Can We Beat DDoS Attacks in Clouds?", IEEETransactions On Parallel And Distributed Systems.
[8] Aqeel Sah, David Lai, Yan Li2, and Mohammed Diykh, "An Efficient DDoS TCP Flood Attack Detection and

Prevention System in a Cloud Environment" IEEE Digital Object Identifier 10.1109/ACCESS.2017.2688460, VOLUME 5, 2017.
[9] Huan Liu, "A New Form of DOS Attack in a Cloud and Its Avoidance Mechanism", ACM 978-1-4503-0089-6/10/10 Chicago, Illinois, USA CCSW October 8, 2010.
[10] Ying Xuan, Incheol Shin, My T. Thai, and Taieb Znati, "Detecting Application Denial-of-Service Attacks: A Group Testing-Based Approach", IEEE Transactions On Parallel And Distributed Systems, VOL. 21, NO. 8, AUGUST 2010.
[11] Indraneel Sreeram and Venkata Praveen Kumar Vuppala, "HTTP flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm", Applied Computing and Informatics 15 59–66,2019.
[12] Khaled Salah, Khalid Elbadawi, and Raouf Boutaba, "Performance Modeling and Analysis of Network Firewalls", IEEE Transactions On Network And Service Management, Vol. 9, No. 1, MARCH 2012.
[13] Deepak R Bharadwaj, Anamika Bhattacharya and Manivannan Chakkaravarthy "Cloud Threat Defense – a Threat Protection and Security Compliance Solution", IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), 2018.
[14] Wei Wei, Feng Chen, Yingjie Xia, and Guang Jin, "A Rank Correlation Based Detection against Distributed Reflection DoS Attacks", IEEE COMMUNICATIONS LETTERS, VOL. 17, NO. 1, JANUARY 2013.
[15] Dr. V. Naga Lakshmi and Shameena Begum "DDos Defense: Enhanced Flooding Detection and Confidence-Based Filtering Method", Advances in Computational Sciences and Technology ISSN 0973-6107 Volume 10, Number 8, 2017.
[16] V.Priyadharshini and Dr.K. Kuppusamy "Prevention of DDOS Attacks using New Cracking Algorithm", International Journal of Engineering Research and Applications (IJERA) Vol. 2, Issue 3, May-Jun 2012.
[17] Openstack information - <https://vmokshagroup.com /tag/openstack/.
[18] Openstack Information - < https://en.wikipedia.org/ wiki/OpenStack.
[19] https://www.techsupportpk.com/2016/12/installing-openstack-on-multi-node-in-linux.html
[20] https://www.linuxtechi.com/launch-instance-from openstack -dashboard/
[21] https://en.wikipedia.org/wiki/Low_Orbit_Ion_Cannon
[22] https:/www.wireshark.org/docs/manpages/tshark.html
[23] https://www.wireshark.org/docs/manpages/capinfos.html
[24] https://www.geeksforgeeks.org/how-to-setup-firewall-in-linux/

