A simple security policy enforcement system for an institution using SDN controller

Harsh Jain[1], Paritosh Dhotre[2], Anurag Kothari[3], Suresh Babu K. S[4]
[1,2,3,4]*Department of Information Technology, PCE, Navi Mumbai, India - 410206*

Abstract **- Basically, as we know (SDN) software-defined networking architectural framework eases the work of the n/w administrators by separating the data plane from the control plane. This provides easy network configuration by supporting a programmable interface for applications development related to security, management etc. and the centralized logical controller provides more control over the total network, which has complete network visibility. These SDN advantages exposes the network to vulnerabilities and the impact of the attacks is much severe when compared to traditional networks, where the network devices have protection from the attacks and limits the occurrence of attacks. Basically, Distributed Denial of Service (DDoS) attack is a DoS attack which utilizes multiple distributed attack sources. We know that every network in the system has an entropy and increase in the randomness causes entropy to decrease. For preventing this DDoS threat, we want to use POX for attack detection and want to provide a solution that is effective in terms of the resources used. More precisely, this project shows how DDoS attacks can consume controller resources and provide a solution to detect such attacks based on the entropy variation of the destination IP address. Now based on this entropy value, we shall block that specific port in the switch if it drops below certain threshold value, and then bring the port down.**

Index Terms **- DDoS, POX, SDN**

1.INTRODUCTION

Now-a-days cloud services are expanding, and large organizations are migrating towards SDN-based implementations of network. These virtual technologies provide manageability, predictability and quality of service. Security provision importance is relevant for centralized managed network and has become one of the concerns. Considering a centralized virtual server running as a controller, which installs and manages the flows in data plane through OpenFlow communication protocol. The use of OpenFlow makes the controller a primary victim for the attacker because of the following reasons.

1. In OpenFlow protocol there is no standard for security implementation and developers of products are implementing their own proprietary methods.
2. The programmable aspect of SDN also makes them much more vulnerable to numerous malicious attacks and code exploits.
3. The southbound interface can be targeted with denial of service and side channel attacks.
4. Errors related to configuration of SDN can be more serious than traditional network errors.
5. Establishment of trust is also crucial.

We worked on creating a DDoS attack and detection of the attack on the entropy base. And preventing the DDoS to occur.

2. PROPOSED WORK

What is a DDoS attack?

Basically, is a type of flood attack. Here many packets are sent to a network device for stopping the service or decreasing the performance of such a device. If the source addresses of incoming are spoofed, then the switch would not find a match and the packet needs to be forwarded to the controller. The collection of DDoS spoofed packets and legitimate packets can bind the controller into continuous processing, which exhausts them. Due to this, the controller is unreachable for the new incoming legitimate packets. This will bring the controller down causing loss to the SDN architecture. For a backup controller, the same challenge is to be faced. Such kind of attacks can be detected in the early stage by monitoring few hundred of packets considering changes in entropy. The early detection of DDoS attacks stops the controller from going down. The term ‘early’ is related to tolerance level and traffic being handled by the controller. Due to this, the impact of malicious packets flooding can be controlled. Such

a mechanism needs to be lightweight and high response time. The high response time saves the controller during the attack period for regaining the control by terminating the DDoS attack.

Why entropy?
The main reason for considering entropy is it's ability for measuring randomness in a network. The higher the randomness the higher is the entropy and vice versa. So, whenever the entropy is less than a threshold value we can say that a DDoS attack is occurred.

3. PROJECT IMPLEMENTATION

Basically, an OpenFlow controller is connected to a network. We then observe the entropy of the traffic related to the controller under normal and attack conditions. We used the POX controller for this project, because it runs on Python. The network emulator used is Mininet for creation of network topology. Packet generation is done with the help of Scapy. Where Scapy is used for generation of packets, sniffing, scanning, forging of packet and attacking. Scapy is used for generation of UDP packets and spoofing the source IP address of the packets. Here first we create a Mininet topology of 9 switches and 64 hosts as in fig below.

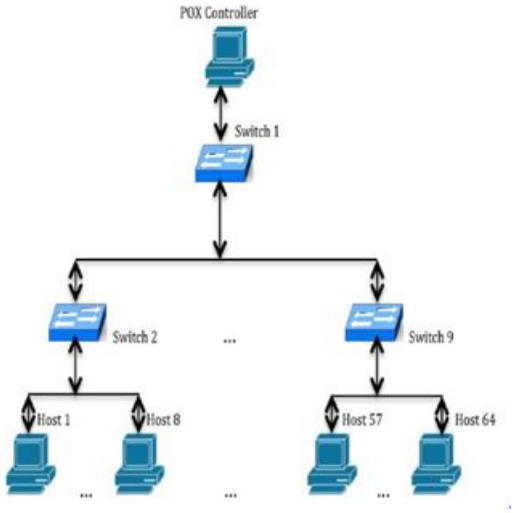


Fig. 1 Network Topology

We then started pox controller from another terminal and then we can see 9 openflow links are connected for 9 open switches.

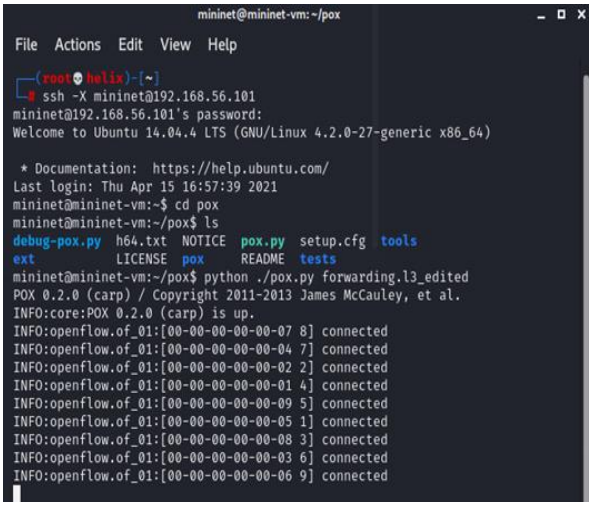


Fig. 2 Launching Controller

Now the project is divided into two parts
1) Packet generation and Detection
2) Attack and Detection.

1. Packet generation: At first run the packet generation program from one of the host which generates random source ip and send the packets to random destinations. We then able to see the entropy in controller terminal.
2. Attack: Here we run the attack from few selected hosts to a selected target. Now the entropy value in the controller decreases. After that for every 5 sets of process i.e., 250 packets entropy is below the threshold value then DDoS is detected and those ip's are blocked.

A. Launch traffic code: (Packet Generation)
Action: In the launch traffic code, we basically, generates random source IP and send the packets to random destinations between the host. We generally give start and end value from the command prompt of one of the node for example h1.

Modules imported:
We import sys module of Python for accessing system specific parameters and functions. The getopt module helps scripts to parse the command line arguments. We use os module to import Popen to fetches shell commands into python program. Scapy module is used to import functions such as sendp, IP, UDP, Ethernet and TCP. The random module is used to import randrange function.

B. Launch attack code:
Action: Here we run the attack from few selected hosts to a selected target. Now the entropy value in the controller decreases. Here we give the target ip address from the command prompt of the hosts which are acting like a botnet.

Modules used:
The Launch Attack code is like Launch Traffic code. The time module provides various time related functions. The logging module helps implement logging system for applications. The logging getlogger function will suppress all messages that have a lower level of seriousness than error messages, before importing scapy.

Code explanation:
Here we get the IP address of target from the command prompt of the bonet armies. As in the same way of traffic launch we create the packet with the random src ip and the send it out through the eth0 with help of sendp function of scapy.

C. Attack Detection: -
Action: here first we make a count of 50 packets in the window and then we calculate the entropy and compare it with threshold that we set and make a count of consecutive entropy value lower than threshold. If this count reaches 5 then we can say that DDoS had occurred otherwise not. For this we written a code for detection and did some changes in the l3_learning module of the pox controller so that it can detect the ddos. These changes are explained below.

Entropy formula: -
Here we detect the entropy with the help of 2 factors:
1. Destination IP and
2. No. of the times it repeated.

Here we used the window size to be 50 and Probability of a destination ip occurred in the window is given as pi :
pi=(xi)/n
where x is no. of event in the set and n to be the window size.
Now,
entropy H= - sum of all (pi)log(pi)
Where i is from o to n. The sequence of steps we use in the detection code is given as a flow below.

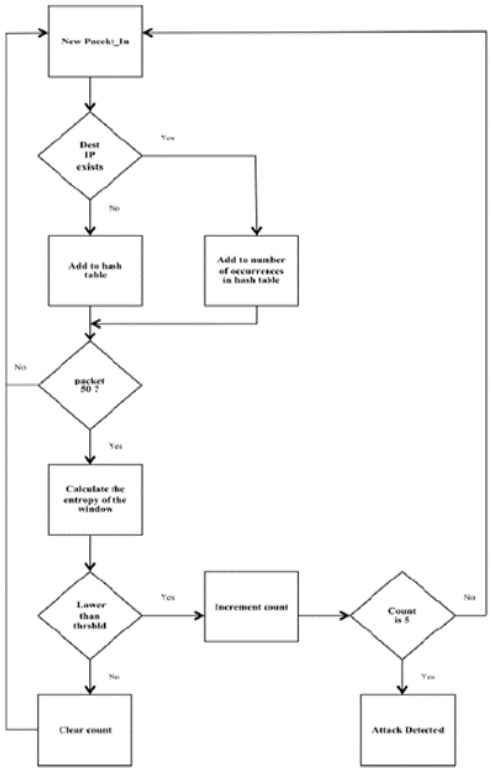


Fig. 3 Flow Diagram

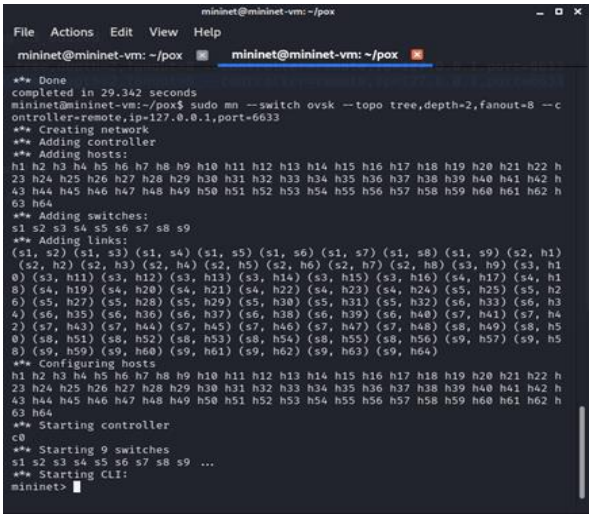
Modules imported in detection:
In the Detection Code, math module is imported for performing mathematical functions such as finding log values. The pox.core module comprises of POX's core API and functionalities. The core.getlogger function returns a logger, which is the root logger of the hierarchy.

Code explanation:
Here in the class Entropy, we defined the entropy dictionary, IP list and destination entropy are as null. The count and value are assigned values as 0 and 1 respectively. In the statcolect function, we basically collect statistics related to the detection of attack i.e. entropy. Here every packet in message is collected into iplist and count is incremented for every packet in. When the count value reaches 50, the 50 Packet_In messages would be parsed for their destination IP addresses in the hash table. If it is present in the table its value increments otherwise it is listed with 1. Entropy is calculated using this hash table values. In the entropy function we calculate the probability of destination ip occurred in the window and then we calculate the entropy by the formula

entropy H= - sum of all (pi)log(pi)
Then we store this value in entropy dictionary which is used in l3_learning module of pox to compare and say whether attack had occurred or not.

3.1 L3_learning module changes:
Here we import the entropy function from detection file and then make set_timer and defend DDoS as false as pox has a predefined method for defending DDoS we are making it off. In is instance class of pox we will extract the entropy values of the windows from entropy dictionary and compare it with a threshold value (we used 0.5 here) and whenever entropy is less than threshold, we implement preventing class otherwise we will set timer as false. In the preventing class in handle_openflow_packetin , we create a diction of switch id and port along with number of times it appeared. It looks like {switch id, (port, count)} and we create timer is true here. Now this diction is used to detect whether a ddos is occurred or not. In the timer function we check the count value and if it is greater than equal to 5 then we can say that a DDoS had occurred and we block the port of the switch by sending the: -
msg = of.ofp_packet_out(in_port=i)
core.openflow.sendToDPID(dpid ,msg)

3.2 Steps for performing the project task:
Finding the threshold of the usual traffic so first we will create a Mininet topology by entering the following command:
A.$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8--controller=remote,ip= 127.0.0.1, port=6633



B. In the Mininet terminal of virtual box enter the following command for running the pox controller:
$ python ./pox.py forwarding.l3_edited

C. Now opening xterm for a host by typing the following command:
mininet> xterm h1

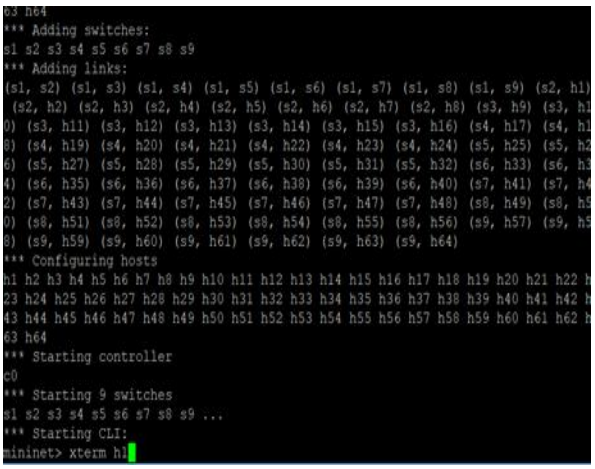


Fig 4. Creating Network

D. In the xterm window of host h1 running the following command:
-> python launchTraffic.py –s 2 –e 65

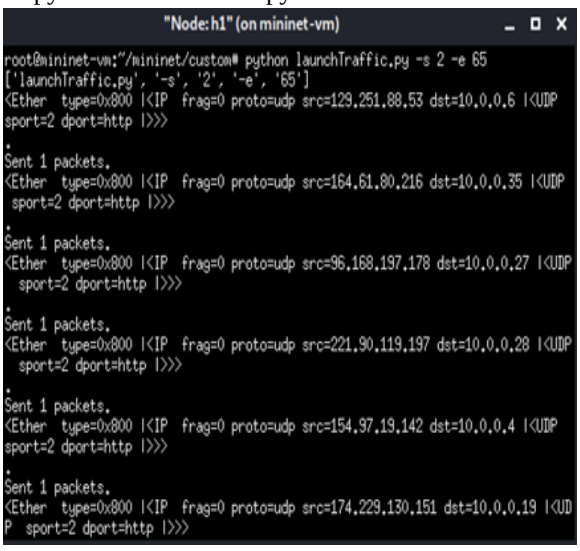


Fig 5. Generating Traffic

Detection of DDoS threat using the value of Entropy:
On xterm window of h64 entering the following commands:
> script h64.txt
> tcpdump –w h64 -eth0

E. Entropy value before the DDoS attack:

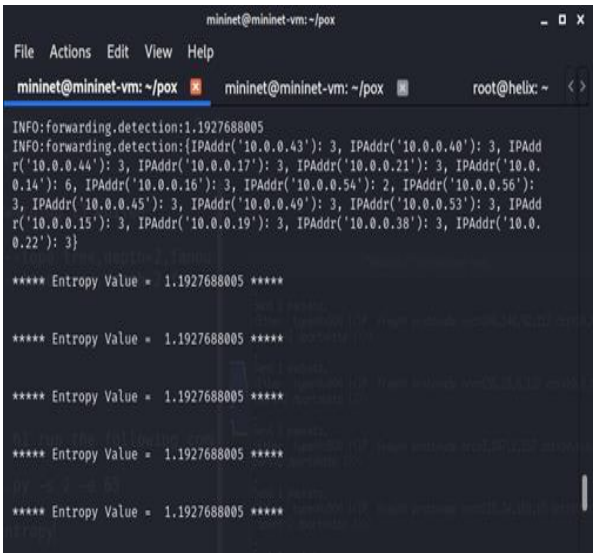


Fig. 6 Entropy Value Before DDoS attack.

Now repeating step D on h1 and parallelly entering the following commands to run the attack traffic from h4 and h6 xterm windows to attack on 56

> python launchAttack.py 10.0.0.56

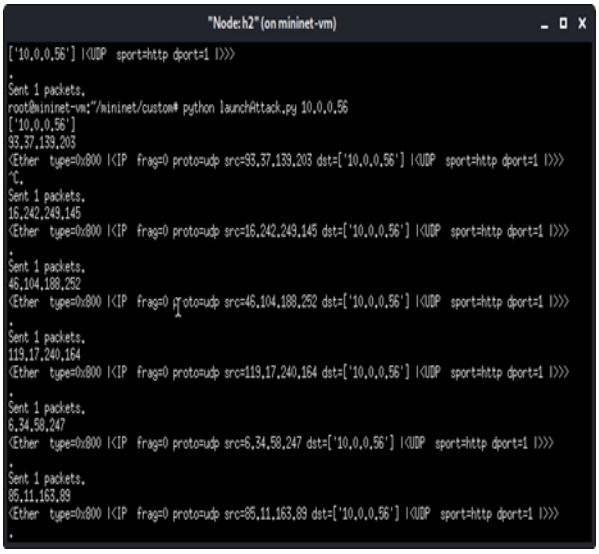


Fig. 7 Attacking the target host.

F. Detection of DDoS: -

The value decreases below the threshold value (which is equal to 0.5 here) for normal traffic. Thus, we can detect the attack within the first 250 packets of malicious type of traffic attacking a host in the SDN network. After the hosts stop sending attack packets, the switches are started again by the POX controller. We can see that controller terminal giving alert that DDoS is detected in the diagram below.

Fig. 8 Detection of DDoS attack.

4.CONCLUSION

For a network administrator, it is complicated to enforce security policy, manage updates and access control on traditional network. Security policy enforcement in traditional network is time consuming and error prone. SDN provide an ideal paradigm to address these challenges of traditional networks and it will be helpful to avoid masquerading attack. By blocking the malicious users, we can save our network form denial of service attack.

5.ACKNOWLEDGEMENT

We would like to thank our guide and mentor, Prof. Suresh Babu K. S, Pillai College of Engineering who mentored us throughout our “A simple security policy enforcement system for an institution using SDN controller” project and cleared our concepts and helped us understand all the topics. We would also like to thank the Head of Department of IT Dr. Satish Kumar Varma, Pillai College of Engineering for giving us an opportunity to understand this project, which helped us a lot in understanding deep concepts of Internet of Things, and how it works. We thank our principal Dr.Sandeep Joshi, Pillai College of Engineering for providing us with all the facilities and opportunities to explore our domain and for motivating us to do better.

REFERENCES

[1] Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2020).
[2] Antikainen, Markku, Tuomas Aura, and Mikko Särelä. "Spook in your network: Attacking a sdn with a compromised openflow switch." Nordic Conference on Secure IT Systems. Springer, Cham, 2019.
[3] McKeown, Nick, et al. "OpenFlow: enabling innovation in campus networks." ACM SIGCOMM Computer Communication Review38.2 (2018): 69-74.
[4] Casado, Martin, et al. "SANE: A Protection Architecture for Enterprise Networks." USENIX Security Symposium. Vol. 49. 2006.
[5] Casado, Martin, et al. "Ethane: Taking control of the enterprise." ACM SIGCOMM Computer Communication Review. Vol. 37. No. 4. ACM, 2007.