International Journal of Engineering & Technology, 7 (2.6) (2018) 46-49

SPC

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research Paper

Mitigation of DDoS attack instigated by compromised switches on SDN controller by analyzing the flow rule request traffic

Sanjeetha R[1]*, Shikhar Srivastava[2], Rishab Pokharna[3], Syed Shafiq[4], Dr. Anita Kanavalli[5]

[1,2,3,4,5]*Ramaiah Institute of Technology, affiliated to VTU*
**Corresponding author E-mail: sanjeetha.r@msrit.edu*

Abstract

Software Defined Network (SDN) is a new network architecture which separates the data plane from the control plane. The SDN controller implements the control plane and switches implement the data plane. Many papers discuss about DDoS attacks on primary servers present in SDN and how they can be mitigated with the help of controller. In our paper we show how DDoS attack can be instigated on the SDN controller by manipulating the flow table entries of switches, such that they send continuous requests to the controller and exhaust its resources. This is a new, but one of the possible way in which a DDoS attack can be performed on controller. We show the vulnerability of SDN for this kind of attack. We further propose a solution for mitigating it, by running a DDoS Detection module which uses variation of flow entry request traffic from all switches in the network to identify compromised switches and blocks them completely.

Keywords: *Controller; DDoS; flow table; hard time out; idle timeout; SDN.*

1. Introduction

DDoS attacks are induced on primary servers like web servers, file servers, chat servers etc., in traditional networks. Identifying the DDoS attack becomes difficult as the server may not be able to differentiate between high normal traffic and attack traffic. In SDN, the controller has the global view of the entire network, it collects the statistics of number of packets sent by hosts and switches at any point of time and can identify a DDoS attack on any such primary servers easily based on these statistics. Many existing research papers discuss such DDoS attacks on the primary servers and propose few feasible solutions using controller.

In our paper we discuss the case when the centralized controller is itself under DDoS attack. This can be instigated by compromising switches in SDN, such that they send huge amount of flow table requests to the controller which acts as the attack traffic.

2. Background

SDN is a new network architecture that separates the data plane from the control plane. The SDN controller implements the control
plane and is responsible for making forwarding decisions and installing rules in the flow table. The network devices like hubs, switches and routers implement the data plane. They simply forward the packets based on the rules installed in their flow table.

In SDN, when a host wants to send packet to any destination, it first sends it to the switch to which it is connected. When the Switch receives the packet through its input port, it checks its flow table for any entries that matches the fields in the incoming packet like IP address, MAC address, port number etc., if a match is found it simply performs the actions specified in the matching rule which includes forwarding the packet to a specified output port, forwarding packet on all outgoing ports or dropping the packet. If no match is found, the packet is immediately forwarded to the controller.

The controller after receiving the unmatched packet from the switch decides how to handle it using different algorithms, applications or protocols. The controller then creates a new flow rule for this packet and installs this rule on the requesting switch and all other switches under its control. The unmatched packet is then sent back to the switch. The requesting switch now refers to the new rule and performs the specified action on that packet.

This very fundamental working principle of SDN switches can be exploited by attackers to manipulate the switch such that it sends continuous flow table requests to the controller and inflicts a DDoS attack on it.

A flow table is maintained by every switch in SDN. The flow table consists of match fields and actions. The packet received by the switch is compared with the match field specified in the flow table and the corresponding action (forward to specific port, forward to all ports and drop) is taken.

The flow entry can be matched upto 12 fields viz., VLAN priority, Ethernet source address, Ethernet destination address, Ethernet frame type, IP source address, IP destination address, IP protocol, IP Type of Service (ToS) bits, TCP/UDP source port and

TCP/UDP destination port. The rule need not have values for all fields. Suppose the rule is specified only based on destination IP address, all other field values will have wild cards.

A timeout field is specified for each flow table rule. There are two types of time out. First is the idle timeout which is used to remove flow table rule if there are no matching packets for the specified amount of time. Second is the hard timeout which is used to remove flow table rule after specified time, irrespective of whether matching packets arrive or not [1].

3. Related Works

In [2] Fan et al. have proposed Source Router Preferential Dropping (SRPD) Scheme to detect and block DDOS attacks at their sources. SRPD only monitors the high rate outgoing flows (identified by Source and Destination IP address.). Routers here maintain different states of each flow. The rate limit of sending traffic is done by preferential dropping i.e., if average response time is greater than the minimum threshold the packets are dropped to reduce collateral damage to a very low level.

In [3] Mousavi et al, proposed early detection of DDoS attacks on SDN controllers by calculating the entropy which is the variation of destination IP address of the request packet. If the entropy increases beyond a specified threshold the packet is identified as an attack packet and is dropped.

In [4] Safko et al., have coded a software object named "bouncer" in Java that analyses request packets to identify whether they are malicious or legitimate based on frequency of requests and originating source and accordingly place them in a priority queue to mitigate their effect.

In [5] Jantila et al., use a hybrid mechanism to defend DDOS attacks on a web server in SDN with the help of controller. The web server regularly reports the behavior of its clients to the controller, based on which a trust value is calculated. The client behavior is analyzed using STRIDE threat model (Spoofing identity Tampering with data Repudiation Information disclosure DDOS Elevation of privilege). An entropy deviation is also calculated to identify an attacker who acts like a legitimate user initially to gain high trust value but misbehaves later. The controller authorizes each client based on trust and entropy deviation and then allows it to access the web server. If these values vary too much from normal, they are not allowed to communicate with the server.

In [6] Yoon et al., proposed a method to mitigate DDOS attacks on Critical internet sites. The DDoS for such sites are usually done using bot master that has a set of compromised hosts under its control. To mitigate such attacks, the IP addresses of previous logins are stored to create a white list of legitimate clients. When there is suspicion of a DDoS attack, traffic from white list clients only are allowed.

4. Implementations

4.1. Proposed DDoS attack

As discussed in introduction, each flow rule entry in a flow table has an idle and hard timeout associated with it. These timeouts specify the time duration till which a flow table entry is valid. After this time expires, the rule is deleted from the flow table. When a packet is received and the switch has a matching flow rule with a valid timeout value in its flow table, the packet is handled by switch only. Otherwise the packet is forwarded to the controller. Normally idle timeout value is set to 10seconds and hard timeout value is set to 30 seconds. Figure 1 shows these values in normal switch.

```
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:09.1 -> 00:00:00:00:00:08.3
DEBUG:forwarding.l2_learning:ofp_flow_mod
  header:
    version: 1
    type:    14 (OFPT_FLOW_MOD)
    length:  80
    xid:     962
  match:
    wildcards: nw_tos|tp_dst|tp_src (10000000000000011000000 = 2000c0)
    in_port: 1
    dl_src: 00:00:00:00:00:09
    dl_dst: 00:00:00:00:00:08
    dl_vlan: 65535
    dl_vlan_pcp: 0
    dl_type: 0x806
    nw_proto: 2
    nw_src: 10.0.0.9
    nw_dst: 10.0.0.8
  cookie: 0
  command: 0
  idle_timeout: 10
  hard_timeout: 30
```

Fig. 1: Idle timeout and hard timeout values in normal switch

To induce DDoS attack on controller, we compromise the switch by setting its idle and hard timeout values to 1 second as shown in Figure 2.

```
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.3 -> 00:00:00:00:00:09.1
DEBUG:forwarding.l2_learning:ofp_flow_mod
  header:
    version: 1
    type:    14 (OFPT_FLOW_MOD)
    length:  80
    xid:     1125
  match:
    wildcards: nw_tos|tp_dst|tp_src (10000000000000011000000 = 2000c0)
    in_port: 3
    dl_src: 00:00:00:00:00:04
    dl_dst: 00:00:00:00:00:09
    dl_vlan: 65535
    dl_vlan_pcp: 0
    dl_type: 0x806
    nw_proto: 2
    nw_src: 10.0.0.4
    nw_dst: 10.0.0.9
  cookie: 0
  command: 0
  idle_timeout: 1
  hard_timeout: 1
```

Fig. 2: Idle timeout and hard timeout values in compromised switch

Because of this the compromised switch will keep forwarding all packets to the controller, requesting to install a flow rule into its flow table to handle them. Many switches in the network can be compromised this way to perform a DDoS attack. The controller now gets too much of packets and flow entry requests from all these compromised switches and eventually exhausts its resources.

Figure 3 shows the topology on which the proposed idea is implemented. It consists of a controller connected to 5 switches S1, S2, S3, S4 and S5. Each switch is connected to two hosts.
Figure 3 shows the traffic on controller in both scenarios, black color indicates traffic when all switches are normal and red color indicates attack traffic when switches (here S1 and S2) are compromised.

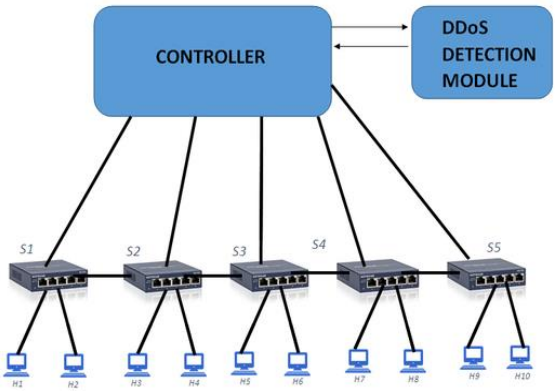

Fig. 3: SDN architecture used for proposed implementation.

4.1. Solution: DDoS detection and mitigation

We propose a solution to mitigate the DDoS attack by compromised switches in this section. A DDoS detection module is programmed and run on the controller to detect the attack. This module observes the number of flow table requests sent by each switch

to the controller and calculates the standard deviation of the overall traffic. It then compares the flow table requests from each switch with this calculated value, if the requests are greater than the standard deviation, the corresponding switch is identified as compromised. Figure 4 shows the pseudo code for DDoS detection module.

```
Begin
    Open source_switch.txt //trace file which contains source and destination ip address of
                                    from all switches
    Initialize  Si=0, where i=1 to N
    Loop in source_switch.txt until not empty
            Identify source switch sending flow tables requests
            Increment the request count for the identified switch i.e. Si=Si+1
    End Loop
    Total_Sum <- Sum{Si} , i=1 to N
    percentSi <-(Si*100)/Total_Sum
    Final_list<-[percentSi] , i=1 to N
    Mean <-(Total_Sum)/n
    Calculate Standard Deviation Std_dev
        Std_dev<-Sqrt (Sum ((percentSi-mean) ^2)/n)
    If (percentSi-mean>Std_dev)
        Detect Si as compromised switch
End
```

Fig. 4: Pseudo code for DDoS Detection Module

After identifying the compromised switch, the result is sent to the controller. The controller then blocks the requests from compromised switches by not processing their flow table requests.
Figure 5 shows the packets sent from hosts connected to normal switches are sent.

```
--- 10.0.0.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 199
rtt min/avg/max/mdev = 0.026/0.040/0.059/0.015 ms
./pingAllHost.sh: line 8: [0: command not found
node 10.0.0.8 is down
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=64 time=10.3 ms
64 bytes from 10.0.0.9: icmp_seq=2 ttl=64 time=0.359 ms
64 bytes from 10.0.0.9: icmp_seq=3 ttl=64 time=0.492 ms

--- 10.0.0.9 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 200
rtt min/avg/max/mdev = 0.359/3.726/10.328/4.668 ms
./pingAllHost.sh: line 8: [0: command not found
node 10.0.0.9 is down
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=64 time=30.5 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=64 time=0.531 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=64 time=0.088 ms
```

Fig. 5: Host H9 connected to normal switch S5 is reachable.

Figure 6 shows dropping of packets sent by hosts connected to compromised switches.

```
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.

--- 10.0.0.8 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2016ms

./pingAllHost.sh: line 8: [1: command not found
node 10.0.0.8 is down
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.9 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2000ms
pipe 3
./pingAllHost.sh: line 8: [1: command not found
node 10.0.0.9 is down
```

Fig. 6: Host H2 connected to compromised switch S1 is reachable.

Figure 7 shows the overall methodology used to implement the solution.

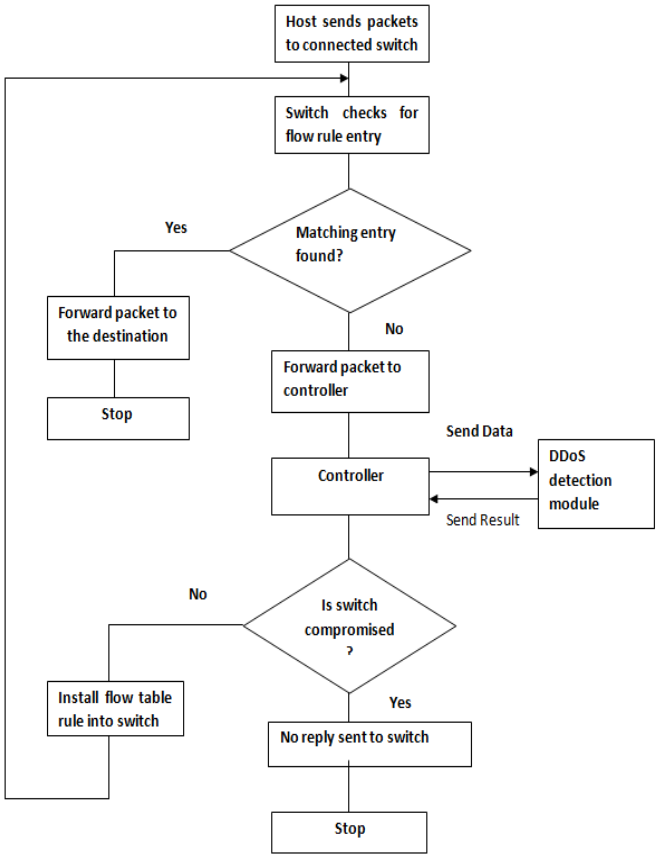

Fig. 7: Methodology for proposed work.

5. Simulation Results

Mininet is used to implement the proposed attack and solution. A customized topology is created using python and the controller used is POX controller, which is in-built in mininet.
The traffic is analyzed by running wire shark and the graph is plotted using its IO graph feature.

Figure 8 shows the graph of normal traffic shown in black line and DDoS attack traffic when the switch is compromised in red line. The graph is plotted with X axis showing time from 0 second to 240 seconds and Y axis showing the number of packet sent during the specified time period.

Figure 9 shows the graph of load on controller (flow requests and packets) from switches S1, S2 and S3 during DDoS attack shown by red line and the load on controller after mitigating the DDoS attack by blocking compromised switch shown by black line. It can be observed that the load on controller reduces significantly after blocking the attack traffic from compromised switches S1 and S2.

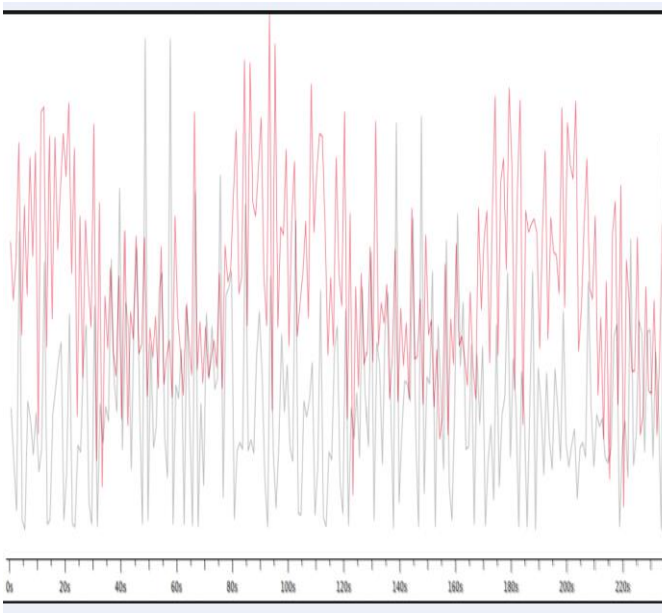

Fig. 8: DDoS attack traffic vs. Normal traffic in network.

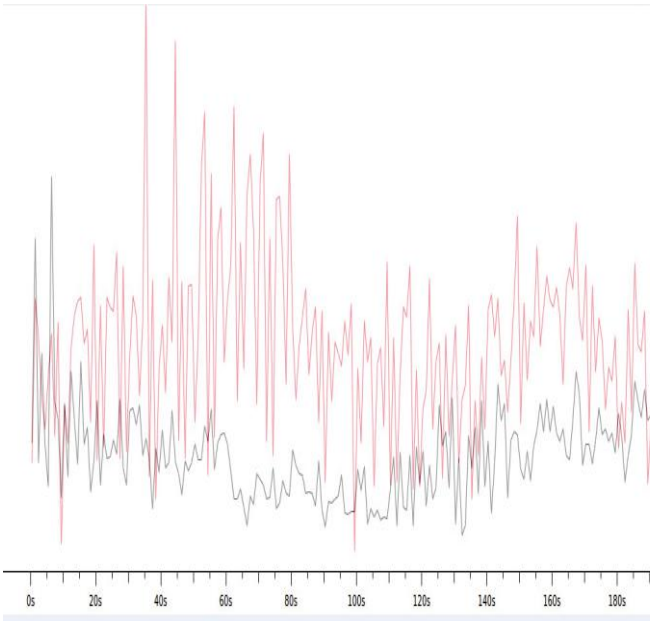

Fig. 9: DDoS attack traffic vs. mitigated traffic on controller.

6. Conclusion & Future Scope

In this paper we show how a new DDoS attack can be instigated on SDN controller by compromising switches to send lot of flow table requests such that its resources get exhausted. We also propose a solution for mitigating the same by detecting the DDoS attack and mitigating it immediately.

The paper can be extended to use other mathematical models to identify the deviation in flow requests from switches and a comparison can be done between them to check its accuracy.

7. Acknowledgements

We would like to acknowledge Ramaiah Institute of Technology, an autonomous institute affiliated to Visveswarya Technological University for their support and encouragement.

References

[1]] Goransson, Paul, Chuck Black, and Timothy Culver. *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[2] Fan, Yinghong, Hossam Hassanein, and Pat Martin. "Proactively defeating distributed denial of service attacks." In *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, vol. 2, pp. 1047-1050. IEEE, 2003.

[3] Mousavi, Seyed Mohammad, and Marc St-Hilaire. "Early detection of DDoS attacks against SDN controllers." In *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pp. 77-81. IEEE, 2015.

[4] Safko, Gregory. "Defending against Denial of Service Attacks using a Modified Priority Queue: Bouncer." In *SoutheastCon, 2006. Proceedings of the IEEE*, pp. 114-119. IEEE, 2005.

[5] Jantila, Saksit, and Kornchawal Chaipah. "A Security Analysis of a Hybrid Mechanism to Defend DDoS Attacks in SDN."*Procedia Computer Science* 86 (2016): 437-440.

[6] Yoon, MyungKeun. "Using whitelisting to mitigate DDoS attacks on critical internet sites." *IEEE Communications Magazine* 48, no. 7 (2010).