LAB 2 Olumuyiwa Ojo - 0232763507

As described in the lab's scope, this lab helped to understand potential threats in SDN environments, monitor traffic, detect anomalies and apply measures to mitigate the threats.

As Included in the section on the tools required, the following tools were used to complete the exercise;

- Mininet VM
- Windows 11 (Host environment)
- Virtualbox Manager
- Docker
- OpenFlow (for communication between switches and controllers) – included in MininetVM
- Open Network Operating System [ONOS] (as the SDN controller)
- Spiger (for Pentesting)
- Wireshark & tcpdump (for capturing and monitoring network traffic)

Monitor Traffic

Due to the CLI interface of the mininet VM, the tcpdump tool was used to capture traffic. Figure 1 shows the description of the command used and the interface of interest. A bridge interface is used because of the need to access the docker container from the host PC.

```
mininet@mininet-vm:~$ sudo tcpdump -i br-05d3bba537f9 -w capt3.pcap
tcpdump: listening on br-05d3bba537f9, link-type EN10MB (Ethernet), capture size 262144 bytes
^C1383 packets captured
1383 packets received by filter
0 packets dropped by kernel
```

Figure 1.

Figure 2 is an example of an ICMP normal traffic pattern. IP address **10.0.0.10** is the address of the Controller while **10.0.0.1** is the gateway IP for the Southbound API

icmp

No.	Time	Source	Destination	Protocol	Lengtl	Info
91	5.682276	10.0.0.1	10.0.0.10	OpenFl…	206	Type: OFPT_PACKET_IN
93	5.686304	10.0.0.10	10.0.0.1	OpenFl…	204	Type: OFPT_PACKET_OUT
253	15.714791	10.0.0.1	10.0.0.10	OpenFl…	206	Type: OFPT_PACKET_IN
255	15.718773	10.0.0.10	10.0.0.1	OpenFl…	204	Type: OFPT_PACKET_OUT
414	25.985955	10.0.0.1	10.0.0.10	OpenFl…	206	Type: OFPT_PACKET_IN
416	25.987273	10.0.0.10	10.0.0.1	OpenFl…	204	Type: OFPT_PACKET_OUT
418	25.987815	10.0.0.1	10.0.0.10	OpenFl…	206	Type: OFPT_PACKET_IN
424	25.989375	10.0.0.10	10.0.0.1	OpenFl…	204	Type: OFPT_PACKET_OUT
425	25.989840	10.0.0.1	10.0.0.10	OpenFl…	206	Type: OFPT_PACKET_IN
430	25.992345	10.0.0.10	10.0.0.1	OpenFl…	204	Type: OFPT_PACKET_OUT
431	25.992803	10.0.0.1	10.0.0.10	OpenFl…	206	Type: OFPT_PACKET_IN

Figure 2.

Anomaly Detection

To be able to inject malicious traffic, the Spiger tool was used. The tool has implementations of different attacks.

mininet@mininet-vm: ~/Spiger

```
__/ Spiger \________
__/ Spiger \________
| 1 App            |
| 2 info           |
| 3 DoS            |
| 4 MiTM           |
| 5 DDoS           |
| 6 Exit Spiger    |
+------------------+
Enter choice number?
```

Figure 3. Spiger Interface

Information Gathering

From the interface in Figure 3, choice number 2 performs reconnaissance to identify open networks or ports. Given a range of ip addresses in Figure 4, the results are described in Figure 5 & 6.

```
__/// Active  \_____
| 1 Who is up?       |
| 2 Port scan        |
| 3 Back             |
|                    |
|                    |
+-------------------+
Enter yuor choice? 1
hosts from ip: 10.0.0.0
to ip        : 10.1.0.0
Begin emission:
..Finished sending 1 packets.
```

Figure 4. Active Scan

```
Received 15 packets, got 0 answers, remaining 1 packets
10.0.0.9
Begin emission:
.*Finished sending 1 packets.

Received 2 packets, got 1 answers, remaining 0 packets
10.0.0.10
Begin emission:
..........Finished sending 1 packets.
```

Figure 5. Result from terminal

146	10.440847	10.0.0.1	10.0.0.9	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
147	10.504015	02:42:54:72:ca:a4	Broadcast	ARP	42 Who has 10.0.0.10? Tell 10.0.0.1
148	10.504046	02:42:0a:00:00:0a	02:42:54:72:ca:a4	ARP	42 10.0.0.10 is at 02:42:0a:00:00:0a
149	10.526144	10.0.0.1	10.0.0.10	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 150)
150	10.526206	10.0.0.10	10.0.0.1	ICMP	42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 149)
151	10.580622	02:42:54:72:ca:a4	Broadcast	ARP	42 Who has 10.0.0.11? Tell 10.0.0.1
152	10.763696	10.0.0.10	192.168.178.34	TCP	56 8181 → 59376 [PSH, ACK] Seq=15 Ack=13 Win=83 Len=2

Figure 6. Result from Wireshark

The attack is a broadcast ARP request sent on the network with active hosts or networks replying to the request.

In addition, two pieces of information from the packet capture. We know that 10.0.0.10 is up, and we also know the gateway 10.0.0.1. We were unaware of this information when we started the recon attack from Figure 4.

Man-in-The-Middle

Based on the information received from recon, we can now perform MiTM attack scenarios. This attack was used to get the Mac address of the victim, which, in this scenario, is the controller as shown in Figure 7. The attack, seen from the wireshark capture in Figures 9 and 10, led to spurious retransmission and duplicate traffic acknowledgement.

Figure 7.

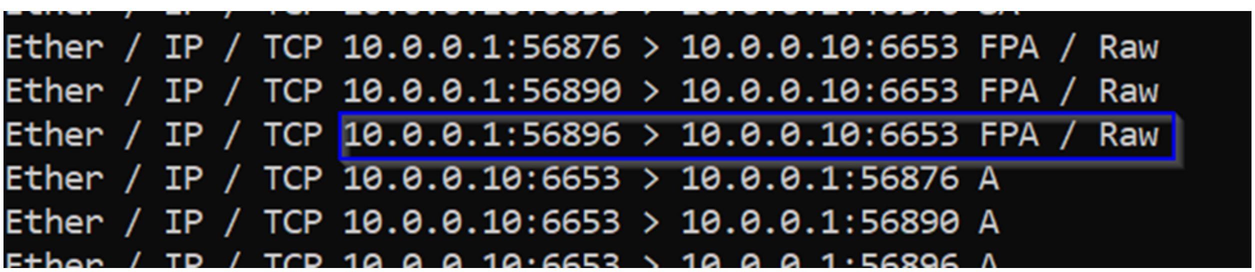


Figure 8. Attack Execution

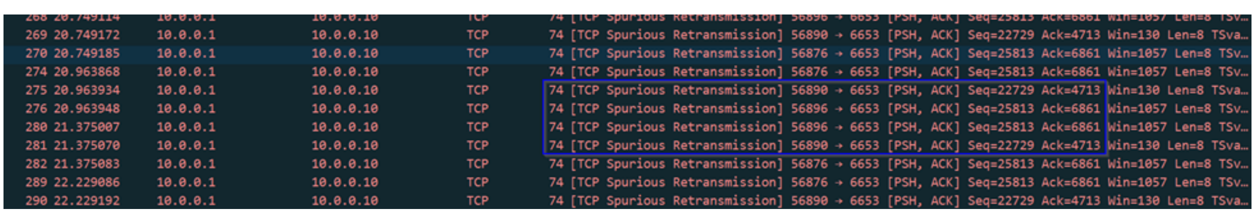


Figure 9. Spurious Retransmission (Wireshark Packets)

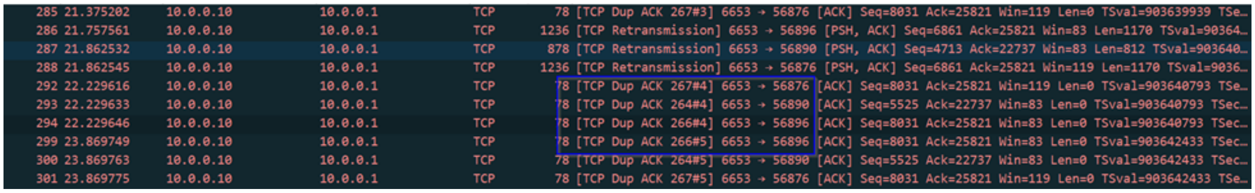


Figure 10. Duplicate Acknowledgement

After the execution of the attack, it was observed that connections to the hosts were lost.

Denial of Service

This attack led to different scenarios. These scenarios are described in Figures 12, 13, & 14.

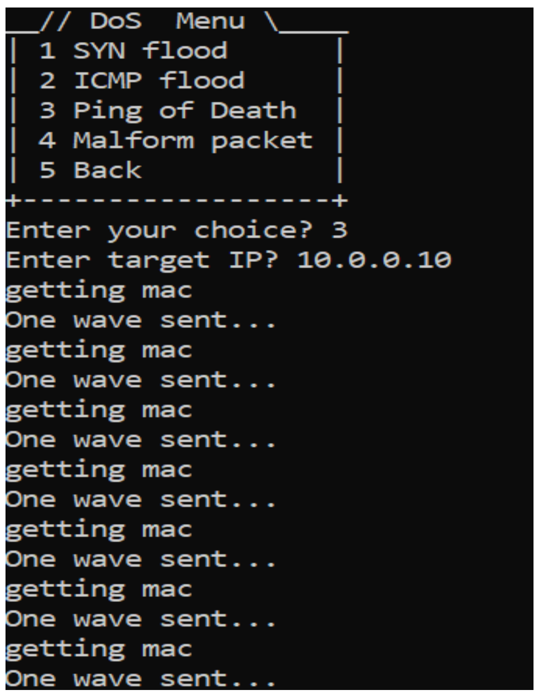


Figure 11. DoS Menu

Figure 12 depicts a lot of duplicate address frames due to DDOS attacks. The flag arp.duplicate-address-frame was used to list the packets affected. It was observed that an unusual number of flows are being removed. This could be as a result of flow tables not having enough memory space to accommodate the traffic. The attack also led to the generation of malformed packets as shown in Figure 14.

arp.duplicate-address-frame

No.	Time	Source	Destination	Protocol	Length	Info
1544	60.493852	10.0.0.1	10.0.0.10	OpenFl…	150	Type: OFPT_PACKET_IN
1546	60.495307	10.0.0.10	10.0.0.1	OpenFl…	230	Type: OFPT_PACKET_OUT
1548	60.495371	10.0.0.10	10.0.0.1	OpenFl…	230	Type: OFPT_PACKET_OUT
1550	60.495792	10.0.0.10	10.0.0.1	OpenFl…	148	Type: OFPT_PACKET_OUT
1552	60.495941	10.0.0.1	10.0.0.10	OpenFl…	150	Type: OFPT_PACKET_IN
1554	60.496821	10.0.0.10	10.0.0.1	OpenFl…	148	Type: OFPT_PACKET_OUT
1556	60.498132	10.0.0.10	10.0.0.1	OpenFl…	230	Type: OFPT_PACKET_OUT
1558	60.500419	10.0.0.10	10.0.0.1	OpenFl…	230	Type: OFPT_PACKET_OUT
1586	60.520461	10.0.0.1	10.0.0.10	OpenFl…	150	Type: OFPT_PACKET_IN
1598	60.529259	10.0.0.10	10.0.0.1	OpenFl…	230	Type: OFPT_PACKET_OUT

Figure 12. Duplicate Address Frames

openflow_v5.type == 11

No.	Time	Source	Destination	Protocol	Length	Info
412	17.329107	10.0.0.1	10.0.0.10	OpenFl…	146	Type: OFPT_FLOW_REMOVED
414	17.329138	10.0.0.1	10.0.0.10	OpenFl…	146	Type: OFPT_FLOW_REMOVED
416	17.329157	10.0.0.1	10.0.0.10	OpenFl…	146	Type: OFPT_FLOW_REMOVED
418	17.329174	10.0.0.1	10.0.0.10	OpenFl…	146	Type: OFPT_FLOW_REMOVED
430	17.335358	10.0.0.1	10.0.0.10	OpenFl…	146	Type: OFPT_FLOW_REMOVED
431	17.335381	10.0.0.1	10.0.0.10	OpenFl…	146	Type: OFPT_FLOW_REMOVED
445	17.336338	10.0.0.1	10.0.0.10	OpenFl…	146	Type: OFPT_FLOW_REMOVED
447	17.336367	10.0.0.1	10.0.0.10	OpenFl…	146	Type: OFPT_FLOW_REMOVED
449	17.336384	10.0.0.1	10.0.0.10	OpenFl…	146	Type: OFPT_FLOW_REMOVED

Figure 13. Unusual number of flows.

openflow_v5 && _ws.expert.severity == Error

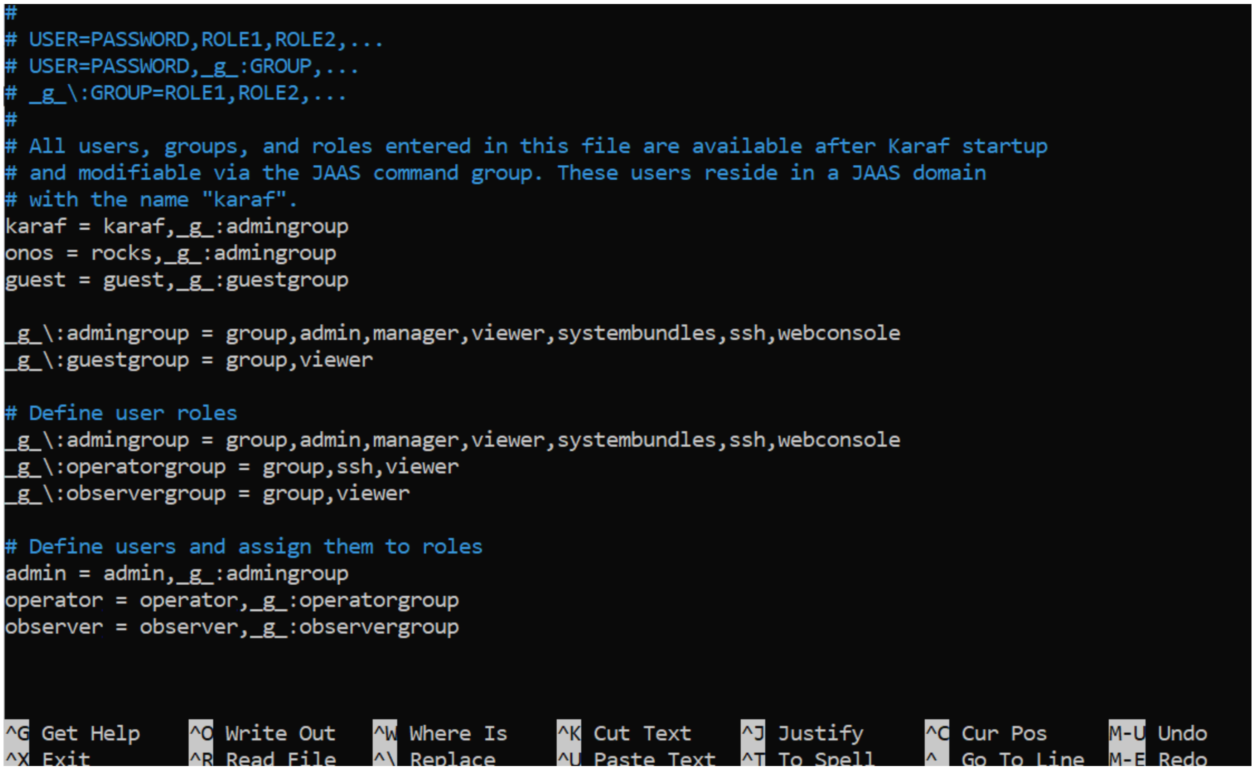
o.	Time	Source	Destination	Protocol	Length	Info
51	3.803185	10.0.0.1	10.0.0.10	OpenFl…	994	Type: OFPT_MULTIPART_REPLY[Malformed Packet]
54	3.811974	10.0.0.1	10.0.0.10	OpenFl…	1298	Type: OFPT_MULTIPART_REPLY[Malformed Packet]
57	3.839399	10.0.0.1	10.0.0.10	OpenFl…	1298	Type: OFPT_MULTIPART_REPLY[Malformed Packet]
137	8.803346	10.0.0.1	10.0.0.10	OpenFl…	994	Type: OFPT_MULTIPART_REPLY[Malformed Packet]
141	8.812330	10.0.0.1	10.0.0.10	OpenFl…	1298	Type: OFPT_MULTIPART_REPLY[Malformed Packet]
144	8.838970	10.0.0.1	10.0.0.10	OpenFl…	1298	Type: OFPT_MULTIPART_REPLY[Malformed Packet]
189	13.805265	10.0.0.1	10.0.0.10	OpenFl…	994	Type: OFPT_MULTIPART_REPLY[Malformed Packet]
193	13.810763	10.0.0.1	10.0.0.10	OpenFl…	1298	Type: OFPT_MULTIPART_REPLY[Malformed Packet]
195	13.840578	10.0.0.1	10.0.0.10	OpenFl…	1298	Type: OFPT_MULTIPART_REPLY[Malformed Packet]
286	18.803720	10.0.0.1	10.0.0.10	OpenFl…	994	Type: OFPT_MULTIPART_REPLY[Malformed Packet]

Figure 14. Malformed Packets

Mitigation Strategies

Role-Based Access Control (RBAC) for Controller's UI/API

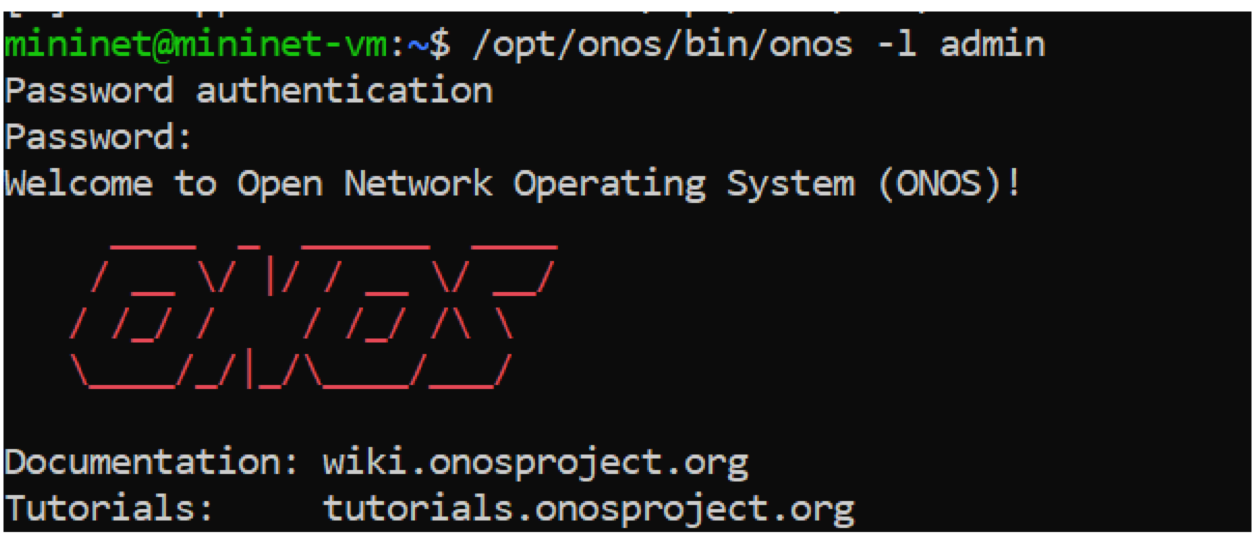
It was implemented by modifying the **user.properties** file in the **/opt/onos/apache-karaf-4.2.9/etc** folder. In Figure 15, it is shown that different users have been assigned to different groups based on their responsibilities.



```
#
# USER=PASSWORD,ROLE1,ROLE2,...
# USER=PASSWORD,_g_:GROUP,...
# _g_\:GROUP=ROLE1,ROLE2,...
#
# All users, groups, and roles entered in this file are available after Karaf startup
# and modifiable via the JAAS command group. These users reside in a JAAS domain
# with the name "karaf".
karaf = karaf,_g_:admingroup
onos = rocks,_g_:admingroup
guest = guest,_g_:guestgroup

_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh,webconsole
_g_\:guestgroup = group,viewer

# Define user roles
_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh,webconsole
_g_\:operatorgroup = group,ssh,viewer
_g_\:observergroup = group,viewer

# Define users and assign them to roles
admin = admin,_g_:admingroup
operator = operator,_g_:operatorgroup
observer = observer,_g_:observergroup

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line  M-E Redo
```

Figure 15. RBAC.

As a proof of concept, Figure 16 shows the login success of the admin user through ssh. In contrast, the observer user failed in Figure 17 due to the role description.



```
mininet@mininet-vm:~$ /opt/onos/bin/onos -l admin
Password authentication
Password:
Welcome to Open Network Operating System (ONOS)!

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
```

Figure 16. Login Success

```
mininet@mininet-vm:~$ /opt/onos/bin/onos -l observer
Password authentication
Password:
Password authentication
Password:
Password authentication
```

Figure 17. Failed Authentication

Source-Based Access Control for Controller's Management Interface

The access control was implemented using an accepted list of IP addresses. Every IP that is included in the list will be able to have access to the Management Interface. The Python script used to implement the access control verifies the IP address (Figure 19), if it is in the whitelist and displays the list of devices or hosts upon authentication (Figure 20). Figure 18 depicts the list of acceptable IP addresses.

```
# Define a list of whitelisted IP addresses
WHITELISTED_IPS = [
    "10.0.0.10",
    "10.0.0.1",
    "10.0.1.1",
    "10.0.1.2",
    "10.0.2.1",
    "10.0.2.2",
    "10.0.3.1",
    "10.0.3.2"
]
```

Figure 18. Whitelist Addresses

```
mininet@mininet-vm:~$ sudo python whilelisting.py
Access denied for IP: 10.0.0.2
```

Figure 19. IP denied

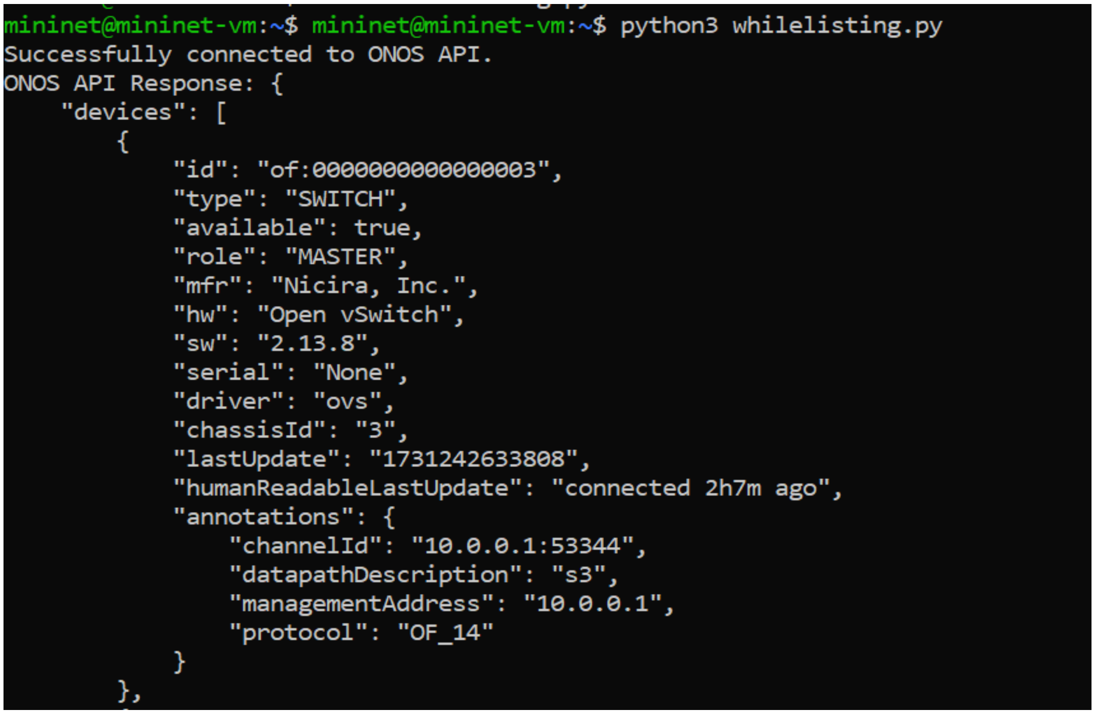


Figure 20. List of Devices

Figure 21. List of Hosts

Rate-Limiting Mechanism

The implementation of the rate limiting mechanism was done by a python script. This Python script configures a traffic rule on the controller. It defines two functions:

1. **create_meter:** Creates a meter on the device (S3) because it connects to the controller directly with a defined rate limit (in kilobits per second) and burst size (also in kbps). This meter restricts the data flow for traffic passing through it.
2. **add_flow_with_meter:** Adds a flow rule to the device. This rule matches incoming traffic that's both ICMP (based on EtherType and IP protocol number) and destined for the controller. The flow rule then applies the previously created meter to this specific traffic, limiting the bandwidth for ICMP packets going to the controller.

Figure 22 displays the successful implementation of the script

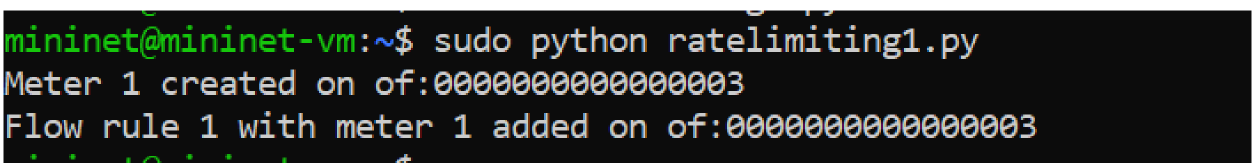
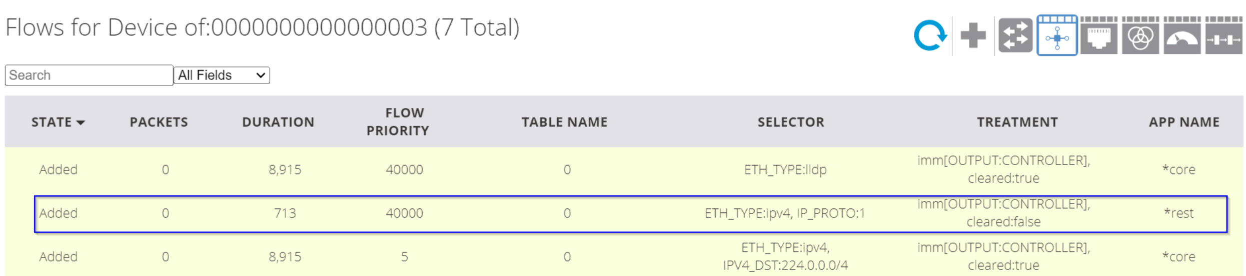


Figure 22. Rate-Limiting

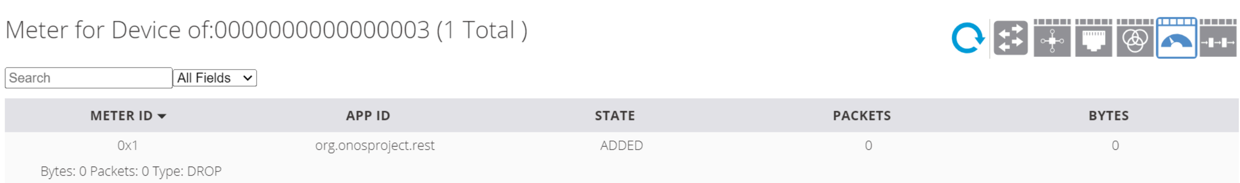


Flows for Device of:0000000000000003 (7 Total)

Search | All Fields

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	8,915	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	713	40000	0	ETH_TYPE:ipv4, IP_PROTO:1	imm[OUTPUT:CONTROLLER], cleared:false	*rest
Added	0	8,915	5	0	ETH_TYPE:ipv4, IPV4_DST:224.0.0.0/4	imm[OUTPUT:CONTROLLER], cleared:true	*core

Figure 23. Flow Added



Meter for Device of:0000000000000003 (1 Total)

Search | All Fields

METER ID	APP ID	STATE	PACKETS	BYTES
0x1	org.onosproject.rest	ADDED	0	0

Bytes: 0 Packets: 0 Type: DROP

Figure 24. Meter Added

Conclusion

This lab provided hands-on experience securing an SDN environment by monitoring traffic, detecting anomalies, and implementing mitigation strategies using the ONOS controller. We used Wireshark and tcpdump to analyze network traffic, with Spiger simulating attacks like reconnaissance, Man-in-the-Middle (MITM), and Denial of Service (DoS), exposing vulnerabilities in the SDN infrastructure. To mitigate these threats, Role-Based Access Control (RBAC) and Source-Based Access Control were applied to restrict access to ONOS's management interface. At the same time, a rate-limiting mechanism was used to control ICMP traffic directed at the controller, preventing potential overloads. These security measures highlighted the importance of layered defenses in SDN to maintain network integrity and resilience against unauthorized access and attacks.