

# Lab 4. Man-in-the-middle Attacks in the SDN Data Plane

# Contents

Objective	<b>3</b>
Submission	<b>3</b>
Prepare the Experiment	<b>3</b>
Prepare the Controller.	3
Prepare the Nodes	8
Conduct the Experiment	<b>12</b>
Scenario 1 (Normal Traffic)	12
Scenario 2 (MITM Attack)	16
Working:	18

# Objective

By using security protocols in the transport layer, students will learn how to establish a secure communication between the controller and switch. Students will understand OpenFlow protocol vulnerability.

Students will be able to launch the man-in-the-middle attack in SDN and understand how attackers can steal information. They will learn security protocols like TLS, IPSec, and SSH and their usage between the controller and the switches. They will learn authentication methods needed for all devices connected to the controller or switches to ensure secure communication.

# Submission

Copy and modify the profile, then start the experiment using your modified profile. Get screenshots of the following two scenarios.

- 1) The **Relay node** did not conduct the MITM attack. Use TCPDUMP to capture packets in **Switch6** or **Switch4**
- 2) The **Relay node** conducted the MITM attack. Use TCPDUMP to capture packets in the **Relay node**.

CloudLab Login page: <https://www.cloudlab.us/login.php>

# Prepare the Experiment

Prepare the Controller.

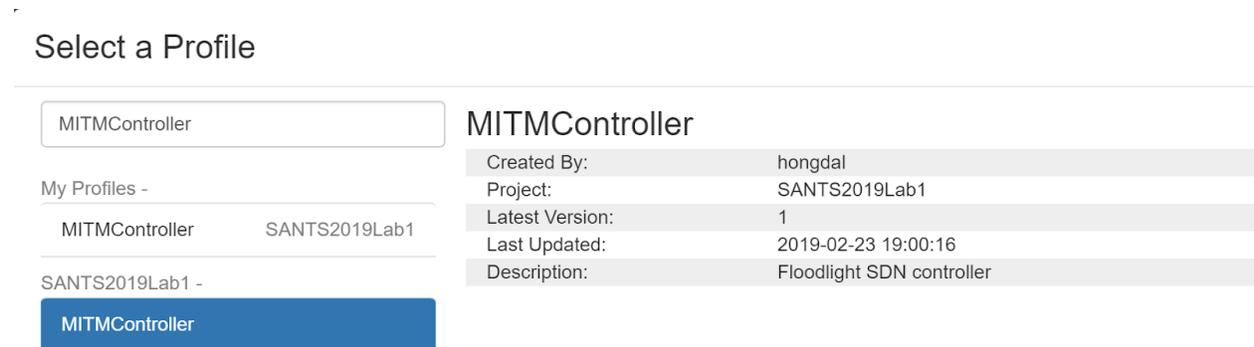


Figure 1: Start an experiment using the **MITMController** profile. Then Click “Select Profile” button to proceed.

1. Select a Profile    2. Parameterize    3. Finalize    4. Schedule

**Selected Profile:** MITMController

Floodlight SDN controller

Copy Profile    Show Profile    **Change Profile**

Previous    **Next**

Figure 2: Click "Next" button to proceed.

1. Select a Profile    2. Parameterize    **3. Finalize**    4. Schedule

**Profile:** MITMController    **Version:** 1    **Source**

Please review the selections below and then click Next.

**Name:** Optional

**Project:** SANTS2019Lab1

**Cluster:** Emulab

**Advanced Options**  
[Check Resource Availability](#)

**Previous**    **Next**

Figure 3: Choose a cluster and click "Next" to proceed.

1. Select a Profile
2. Parameterize
3. Finalize
4. Schedule

Please select when you would like to start this experiment and then click Finish.

Start immediately

or

Start on date/time

Experiment Duration

hours

Previous
Finish

Figure 4: Click “Finish” to proceed.

Your experiment is ready!

Name:	hongdal-QV47176
State:	ready
Profile:	MITMController
Started:	Feb 23, 2019 9:00 PM
Expires:	Feb 24, 2019 1:00 PM (in 16 hours)

Logs
Create Disk Image
Copy
Extend
Terminate

Profile Instructions

Topology View
List View
Manifest
Graphs

pc812



Controller

Click on a node for more options. Click and drag to move things around.

Reload Topo
Refresh Status

Figure 5: The controller is successfully running.

```
Topology View List View Manifest Graphs Controller x
controller:~> ifconfig
eth0      Link encap:Ethernet  HWaddr 02:28:df:cc:12:26
          inet addr:155.98.37.66  Bcast:155.98.39.255  Mask:255.255.252.0
          inet6 addr: fe80::28:dfff:fecc:1226/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4516 errors:0 dropped:120 overruns:0 frame:0
          TX packets:473 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:325952 (325.9 KB)  TX bytes:49941 (49.9 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
          TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1560 (1.5 KB)  TX bytes:1560 (1.5 KB)

controller:~> █
```

Figure 6: Start a shell. Type **“ifconfig”** to check the IP. This IP will be used when we configure the switches. Let’s note it as **controller\_ip**

```
Topology View List View Manifest Graphs Controller x
          RX packets:4516 errors:0 dropped:120 overruns:0 frame:0
          TX packets:473 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:325952 (325.9 KB)  TX bytes:49941 (49.9 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
          TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1560 (1.5 KB)  TX bytes:1560 (1.5 KB)

controller:~>sudo su -
root@controller:~#
root@controller:~#
root@controller:~#
root@controller:~# ls
root@controller:~# wget https://people.cs.clemson.edu/~hongdal/set_floodlight.sh .
```

Figure 7: setup controller.

- 1) Switch to root. Type **“sudo su -”**
- 2) Type **“wget [https://people.cs.clemson.edu/~hongdal/set\\_floodlight.sh](https://people.cs.clemson.edu/~hongdal/set_floodlight.sh) .”**

```
Topology View List View Manifest Graphs Controller x
Length: 310 [text/x-sh]
Saving to: 'set_floodlight.sh'

set_floodlight.sh          100%[=====]          310  ---KB/s   in 0s

2019-02-23 19:11:11 (7.78 MB/s) - 'set_floodlight.sh' saved [310/310]

--2019-02-23 19:11:11-- http://./
Resolving . (...)... failed: No address associated with hostname.
wget: unable to resolve host address '.'
FINISHED --2019-02-23 19:11:11--
Total wall clock time: 0.3s
Downloaded: 1 files, 310 in 0s (7.78 MB/s)
root@controller:~#
root@controller:~#
root@controller:~#
root@controller:~# ls
set_floodlight.sh
root@controller:~# chmod +x set_floodlight.sh
root@controller:~# ./set_floodlight.sh
```

Figure 8: Run the scripts.

- 1) Type ***"chmod +x set\_floodlight.sh"***
- 2) Type ***"./set\_floodlight.sh"***

```
Topology View List View Manifest Graphs Controller x
[copy] Copying 54 files to /root/floodlight/target/bin

compile-test:
[javac] Compiling 91 source files to /root/floodlight/target/bin-test
[javac] warning: [options] bootstrap class path not set in conjunction with -source 1.7
[javac] 1 warning

dist:
[echo] Setting Floodlight version: 1.2
[echo] Setting Floodlight name: floodlight
[jar] Building jar: /root/floodlight/target/floodlight.jar
[jar] Building jar: /root/floodlight/target/floodlight-test.jar

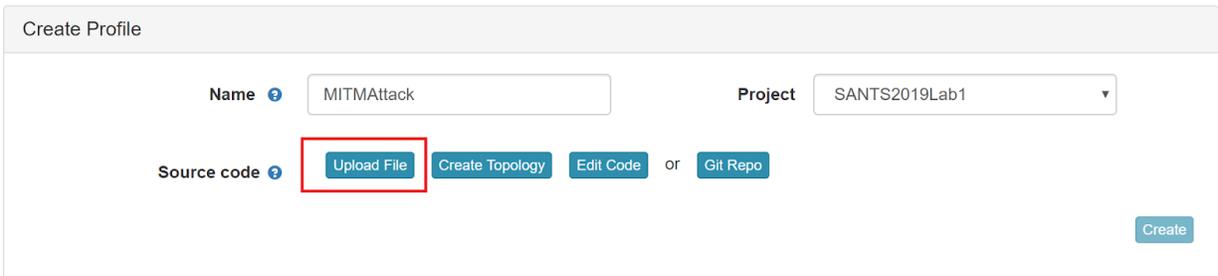
BUILD SUCCESSFUL
Total time: 26 seconds
root@controller:~#
root@controller:~# ls
floodlight set_floodlight.sh
root@controller:~# cd floodlight/
root@controller:~/floodlight# java -jar target/floodlight.jar
```

Figure 9: Launch the controller.

- 1) Type ***"cd floodlight"***
- 2) Type ***"java -jar target/floodlight.jar"***

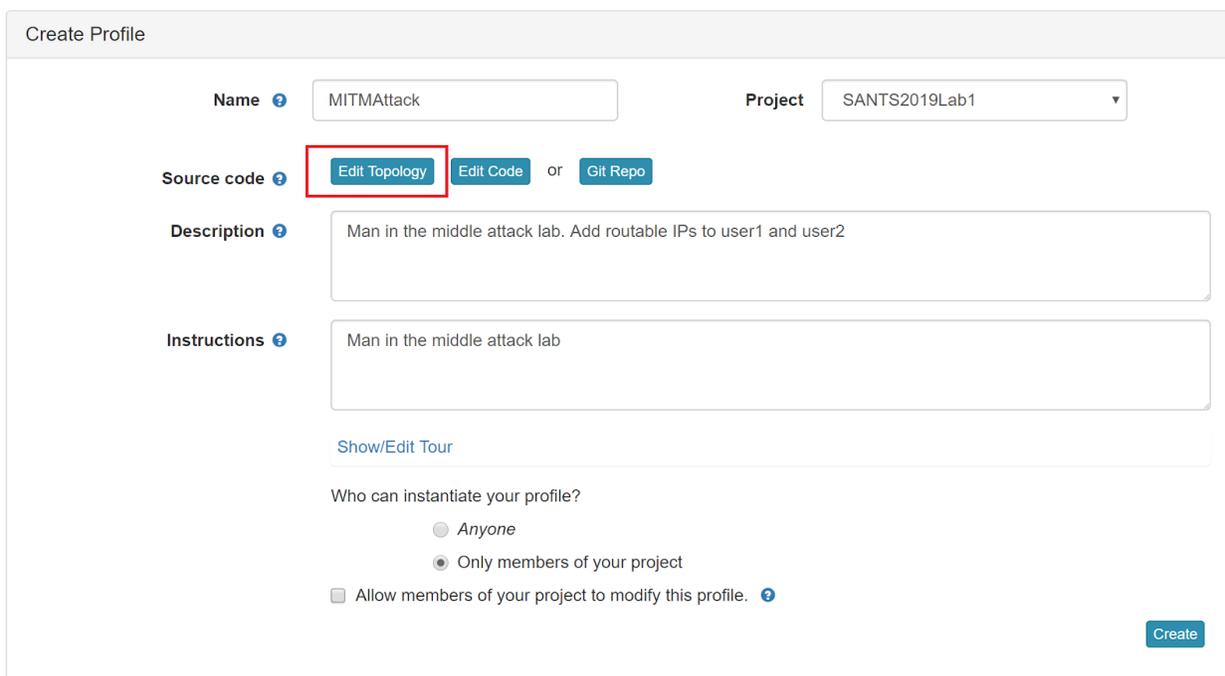
## Prepare the Nodes

Download a topology from here: <https://people.cs.clemson.edu/~hongdal/mitm-attack.xml>  
Create a new profile using the download topology.



The screenshot shows the 'Create Profile' form. The 'Name' field contains 'MITMAttack' and the 'Project' dropdown is set to 'SANTS2019Lab1'. Under the 'Source code' section, the 'Upload File' button is highlighted with a red rectangular box. Other buttons visible are 'Create Topology', 'Edit Code', and 'Git Repo'. A 'Create' button is located at the bottom right of the form.

Figure 10: Upload the topology when creating the profile.



The screenshot shows the 'Create Profile' form with more fields filled out. The 'Name' is 'MITMAttack' and the 'Project' is 'SANTS2019Lab1'. The 'Source code' section now has the 'Edit Topology' button highlighted with a red rectangular box. The 'Description' field contains the text 'Man in the middle attack lab. Add routable IPs to user1 and user2'. The 'Instructions' field contains 'Man in the middle attack lab'. Below these fields is a 'Show/Edit Tour' link. At the bottom, there are radio buttons for 'Who can instantiate your profile?' with 'Anyone' and 'Only members of your project' options, and a checkbox for 'Allow members of your project to modify this profile.' A 'Create' button is at the bottom right.

Figure 11: Click “Edit topology” to configure the SDN controller.

## Topology Editor

The screenshot displays the 'Topology Editor' window. On the left is a configuration panel for a link named 'link-2'. The 'Link Type' is set to '(any)'. The 'Enable Openflow' checkbox is checked, and the 'tcp:155.98.37.66:6653' address is entered in the text field below it. The 'Shared Vlan' is also set to '(any)'. The 'Interfaces' section shows 'Interface to Relay-Node'. On the right is a network diagram for 'Site 1' showing a ring topology with nodes: User2, Relay-Node, User1, Switch6, and Switch4. A green square highlights the link between User2 and Relay-Node. At the top right are 'Delete All' and 'Tidy View' buttons. At the bottom right are 'Accept' and 'Cancel' buttons.

Figure 12: Update the IP address to the **controller\_ip** and select the **Link Type** as “Ethernet” all the 5 links in the profile. Then click “Accept”.

Modify Profile SANTS2019Lab1/MITMAttack

**Source code** ? [Edit Topology](#) [Edit Code](#) [Convert to geni-lib](#) **NEW!**

**Description** ?

**Instructions** ?

[Show/Edit Tour](#)

Who can instantiate your profile?

Anyone

Only members of your project

Allow members of your project to modify this profile. ?

[Delete](#) [Copy](#) [Share](#) [Instantiate](#) [Save](#) [Cancel](#)

Figure 13: Click “instantiate” to proceed.

1. Select a Profile    2. Parameterize    **3. Finalize**    4. Schedule

**Profile:** MITMAttack    **Version:** 0    [Source](#)

Please review the selections below and then click Next.

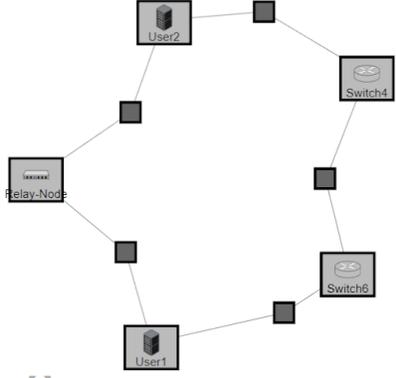
**Name:**

**Project:**

**Cluster:**  ●

[+ Advanced Options](#)

[Check Resource Availability](#)



[Previous](#)    [Next](#)

Figure 14: Choose a cluster. Click “Next” to proceed.

Your experiment is ready! >

---

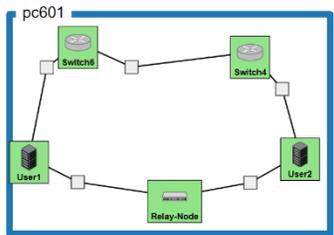
Name: hongdal-QV47178  
State: ready  
Profile: MITMAttack  
Started: Feb 23, 2019 9:18 PM  
Expires: Feb 24, 2019 1:18 PM (in 16 hours)

[Logs](#) [Create Disk Image](#) [Copy](#) [Extend](#) [Terminate](#)

---

[Profile Instructions](#) >

[Topology View](#) [List View](#) [Manifest](#) [Graphs](#)



Click on a node for more options. Click and drag to move things around.

[Reload Topo](#) [Run Linktest](#) [Refresh Status](#)

Figure 15: The nodes are running successfully.

# Conduct the Experiment

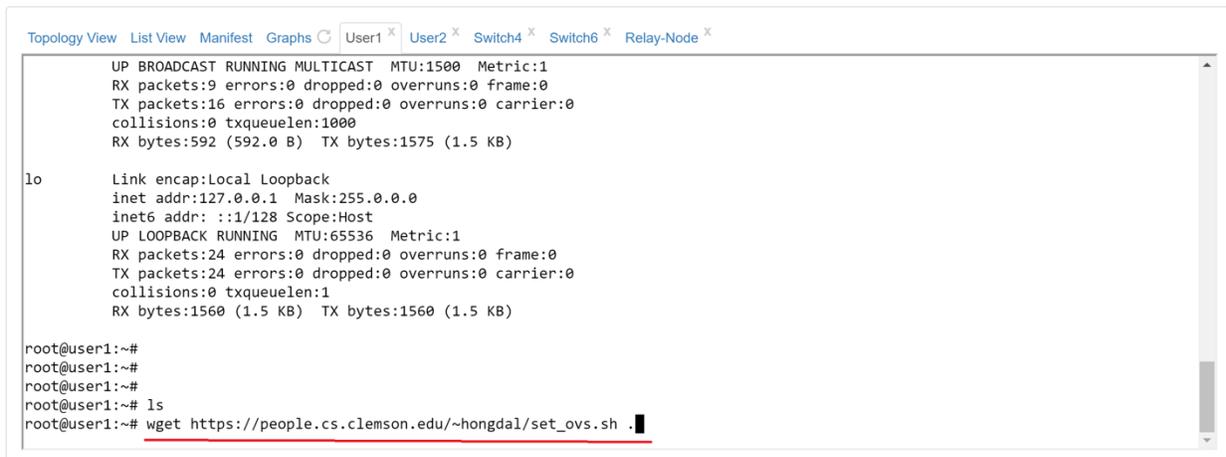
## Scenario 1 (Normal Traffic)

- 1) Open the shells for each of the node. Do steps 2) and 3) for each node. We will use **User1** as an example. The node IP should be different for different nodes.
- 2) Run “**sudo su**” and download the “**set\_ovs.sh**” script by running the command “**wget https://people.cs.clemson.edu/~hongdal/set\_ovs.sh**” on all the nodes.
- 3) change permission and run the script.

**“chmod +x set\_ovs.sh”**

**“./set\_ovs.sh eth1 eth2 155.98.37.66 10.10.10.1”**

The **red IP** is the **controller\_ip**, the **blue IP** is the node IP. Use different node IP for different nodes. E.g., 10.10.10.2 for **User2**, 10.10.10.3 for **Switch4**, and 10.10.10.4 for **Switch6**.

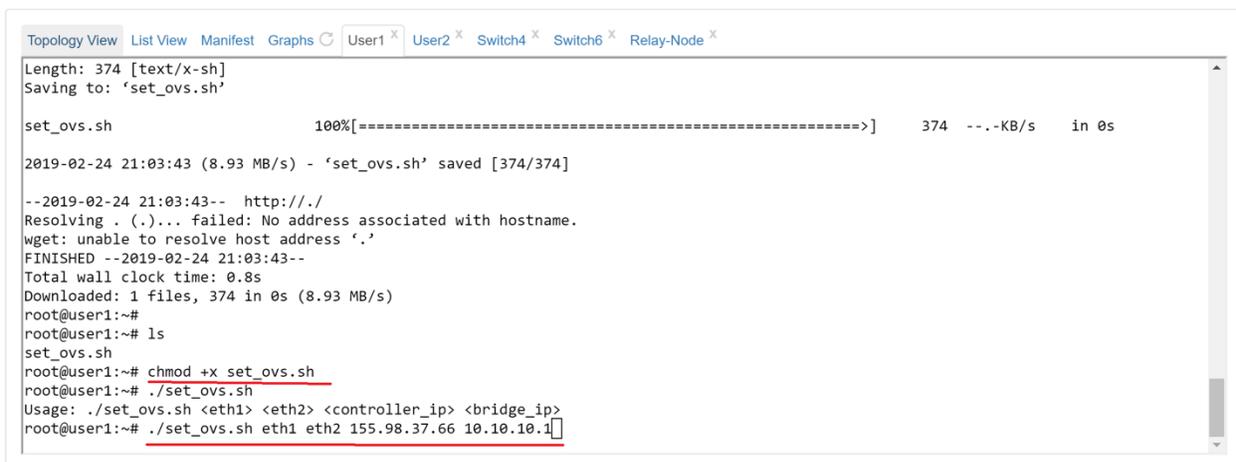


```
Topology View List View Manifest Graphs User1 x User2 x Switch4 x Switch6 x Relay-Node x
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:9 errors:0 dropped:0 overruns:0 frame:0
TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:592 (592.0 B) TX bytes:1575 (1.5 KB)

lo    Link encap:Local Loopback
      inet addr:127.0.0.1 Mask:255.0.0.0
      inet6 addr: ::1/128 Scope:Host
      UP LOOPBACK RUNNING MTU:65536 Metric:1
      RX packets:24 errors:0 dropped:0 overruns:0 frame:0
      TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1
      RX bytes:1560 (1.5 KB) TX bytes:1560 (1.5 KB)

root@user1:~#
root@user1:~#
root@user1:~#
root@user1:~# ls
root@user1:~# wget https://people.cs.clemson.edu/~hongdal/set_ovs.sh
```

Figure 16: Download the script



```
Topology View List View Manifest Graphs User1 x User2 x Switch4 x Switch6 x Relay-Node x
Length: 374 [text/x-sh]
Saving to: 'set_ovs.sh'

set_ovs.sh          100%[=====] 374 --.-KB/s  in 0s

2019-02-24 21:03:43 (8.93 MB/s) - 'set_ovs.sh' saved [374/374]

--2019-02-24 21:03:43-- http://./
Resolving . (...) failed: No address associated with hostname.
wget: unable to resolve host address '.'
FINISHED --2019-02-24 21:03:43--
Total wall clock time: 0.8s
Downloaded: 1 files, 374 in 0s (8.93 MB/s)
root@user1:~#
root@user1:~# ls
set_ovs.sh
root@user1:~# chmod +x set_ovs.sh
root@user1:~# ./set_ovs.sh
Usage: ./set_ovs.sh <eth1> <eth2> <controller_ip> <bridge_ip>
root@user1:~# ./set_ovs.sh eth1 eth2 155.98.37.66 10.10.10.1
```

Figure 17: Change permission and run

4) Check IP of **User1** and **User2**. Note that we will work with experimental IP (ovs-lan), not the public IP.

2) PING from **User1** to **User2** again.

Topology View List View Manifest Graphs  User1  User2  Switch4  Switch6  Relay-Node 

```
lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:24 errors:0 dropped:0 overruns:0 frame:0
        TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:1560 (1.5 KB)  TX bytes:1560 (1.5 KB)

ovs-lan Link encap:Ethernet  HWaddr a6:23:c7:ed:59:4c
        inet addr:10.10.10.1  Bcast:10.10.10.255  Mask:255.255.255.0
        inet6 addr: fe80::a423:c7ff:feed:594c/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:2 errors:0 dropped:0 overruns:0 frame:0
        TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:112 (112.0 B)  TX bytes:648 (648.0 B)

root@user1:~#
```

Figure 18: Check the experimental IP of User1.

```
Topology View List View Manifest Graphs User1 User2 Switch4 Switch6 Relay-Node X
lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING MTU:65536 Metric:1
        RX packets:24 errors:0 dropped:0 overruns:0 frame:0
        TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:1560 (1.5 KB) TX bytes:1560 (1.5 KB)

ovs-lan Link encap:Ethernet HWaddr b2:da:61:9c:69:49
        inet addr:10.10.10.2 Bcast:10.10.10.255 Mask:255.255.255.0
        inet6 addr: fe80::b0da:61ff:fe9c:6949/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:4 errors:0 dropped:0 overruns:0 frame:0
        TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:224 (224.0 B) TX bytes:648 (648.0 B)

root@user2:~#
```

Figure 19: Check the experimental IP of User2.

3) Observe the packets from **Switch4** or **Switch6**. Type “*tcpdump -i eth1 -nq icmp*” in their shells.

```
Topology View List View Manifest Graphs User1 User2 Switch4 Switch6 Relay-Node X
collisions:0 txqueuelen:1
RX bytes:1560 (1.5 KB) TX bytes:1560 (1.5 KB)

ovs-lan Link encap:Ethernet HWaddr a6:23:c7:ed:59:4c
        inet addr:10.10.10.1 Bcast:10.10.10.255 Mask:255.255.255.0
        inet6 addr: fe80::a423:c7ff:feed:594c/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:2 errors:0 dropped:0 overruns:0 frame:0
        TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:112 (112.0 B) TX bytes:648 (648.0 B)

root@user1:~# ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=182 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=2.11 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=1.50 ms
64 bytes from 10.10.10.2: icmp_seq=4 ttl=64 time=1.74 ms
64 bytes from 10.10.10.2: icmp_seq=5 ttl=64 time=1.46 ms
```

Figure 20: PING from User1 to User2.

```
Topology View List View Manifest Graphs User1 x User2 x Switch4 x Switch6 x Relay-Node x
ovs_version: "2.5.5"
root@switch4:~#
root@switch4:~#
root@switch4:~#
root@switch4:~# tcpdump -i eth1 -nq icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
20:16:45.451014 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 20628, seq 1, length 64
20:16:45.458367 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 20628, seq 1, length 64
20:16:46.278842 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 20628, seq 2, length 64
20:16:46.279816 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 20628, seq 2, length 64
20:16:47.279927 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 20628, seq 3, length 64
20:16:47.280453 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 20628, seq 3, length 64
20:16:48.281814 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 20628, seq 4, length 64
20:16:48.282365 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 20628, seq 4, length 64
20:16:49.283499 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 20628, seq 5, length 64
20:16:49.284001 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 20628, seq 5, length 64
20:16:50.285153 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 20628, seq 6, length 64
20:16:50.285647 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 20628, seq 6, length 64
20:16:51.286794 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 20628, seq 7, length 64
20:16:51.287211 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 20628, seq 7, length 64
```

Figure 21: Observe the packets at Switch4 or Switch6.

4) Observe the packets from Relay node. Type ***"tcpdump -i eth1 -nq icmp"*** in its shell.

```
Topology View List View Manifest Graphs User1 x User2 x Switch4 x Switch6 x Relay-Node x
root@relay-node:/users/priganta# tcpdump -i eth1 -nq icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
█
```

No traffic is observed on Relay node.

## Scenario 2 (MITM Attack)

1) Stop the PING at **User1**

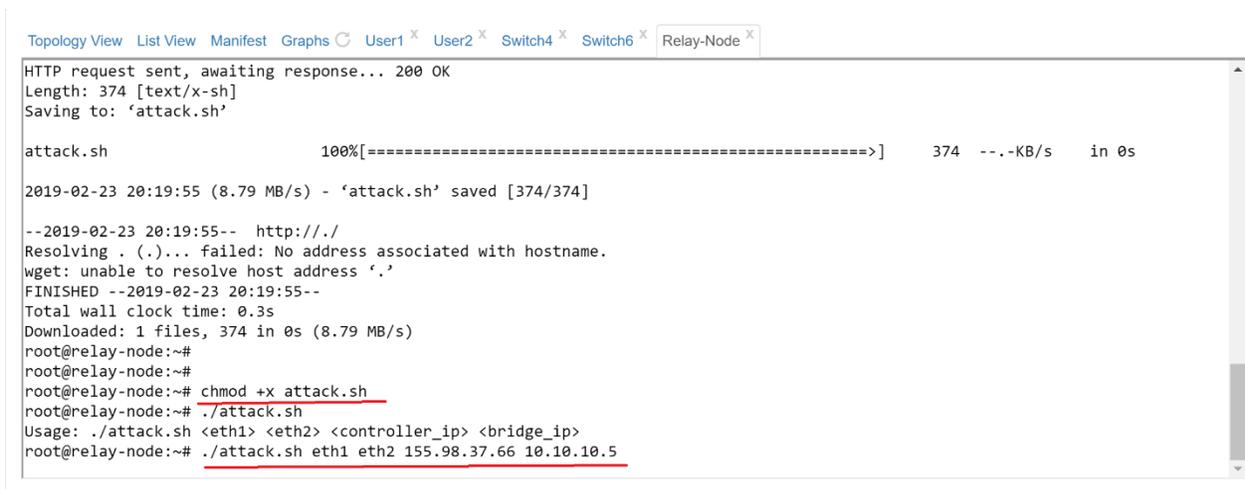
2) Conduct attack.

(1) Download the attack script by running the command “**wget** <https://people.cs.clemson.edu/~hongdal/attack.sh>”

The “**attack.sh**” script creates a ovs-switch on the relay node and connects it to the controller which is very similar to the “**set\_ovs.sh**” script that is run in the remaining nodes.

(2) Type “**chmod +x attack.sh**”

(3) Run **attack.sh**. “**attack.sh eth1 eth2 155.98.37.66 10.10.10.5**”. The **controller\_ip** is use here.



```
Topology View List View Manifest Graphs User1 User2 Switch4 Switch6 Relay-Node
HTTP request sent, awaiting response... 200 OK
Length: 374 [text/x-sh]
Saving to: 'attack.sh'

attack.sh          100%[=====]           374  --.-KB/s   in 0s

2019-02-23 20:19:55 (8.79 MB/s) - 'attack.sh' saved [374/374]

--2019-02-23 20:19:55-- http://./
Resolving . (...)... failed: No address associated with hostname.
wget: unable to resolve host address '.'
FINISHED --2019-02-23 20:19:55--
Total wall clock time: 0.3s
Downloaded: 1 files, 374 in 0s (8.79 MB/s)
root@relay-node:~#
root@relay-node:~#
root@relay-node:~# chmod +x attack.sh
root@relay-node:~# ./attack.sh
Usage: ./attack.sh <eth1> <eth2> <controller_ip> <bridge_ip>
root@relay-node:~# ./attack.sh eth1 eth2 155.98.37.66 10.10.10.5
```

Figure 22: Download **attack.sh**, change permission, and run the **attack.sh** script.

5) Wait for a minute to let the attack take effect.

6) PING from **User1** to **User2** again.

7) Observe the packets from the **Relay node**.

8) Observe the packets from **Switch6** or **Switch4**.

```
Topology View List View Manifest Graphs User1 x User2 x Switch4 x Switch6 x Relay-Node x
inet addr:10.10.10.5 Bcast:10.10.10.255 Mask:255.255.255.0
inet6 addr: fe80::600d:f5ff:fe03:c44d/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:4937 errors:0 dropped:0 overruns:0 frame:0
TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:276472 (276.4 KB) TX bytes:648 (648.0 B)

root@relay-node:~# tcpdump -i eth1 -nq icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
21:57:28.856157 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2760, seq 1, length 64
21:57:28.867933 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2760, seq 1, length 64
21:57:29.852329 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2760, seq 2, length 64
21:57:29.852983 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2760, seq 2, length 64
21:57:30.852932 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2760, seq 3, length 64
21:57:30.853406 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2760, seq 3, length 64
21:57:31.854138 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2760, seq 4, length 64
21:57:31.854626 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2760, seq 4, length 64
21:57:32.855269 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2760, seq 5, length 64
21:57:32.855766 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2760, seq 5, length 64
```

Figure 23: Packets observed from Relay Node

```
Topology View List View Manifest Graphs User1 x User2 x Switch4 x Switch6 x Relay-Node x
ovs_version: "2.5.5"
root@switch4:~# tcpdump -i eth1 -nq icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
21:55:21.290176 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2752, seq 12, length 64
21:55:21.290682 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2752, seq 12, length 64
21:55:22.291839 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2752, seq 13, length 64
21:55:22.292335 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2752, seq 13, length 64
21:55:23.292607 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2752, seq 14, length 64
21:55:23.293120 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2752, seq 14, length 64
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
root@switch4:~#
root@switch4:~#
root@switch4:~# tcpdump -i eth1 -nq icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Figure 24: Packets observed from Switch4 or Switch6

### Working:

In the second scenario, when the relay node is connected to the controller, the controller tends to route the packets through the shortest path which is through the relay node. In this case the packets are routed from source to destination and the relay node is capable of sniffing the traffic flowing between the two nodes. Hence, this acts as Man-in-the-middle, making no changes to the traffic flow, but capable of sniffing it.