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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>3</td>
</tr>
<tr>
<td>Submission</td>
<td>3</td>
</tr>
<tr>
<td>Prepare the Experiment</td>
<td>3</td>
</tr>
<tr>
<td>Prepare the Controller.</td>
<td>3</td>
</tr>
<tr>
<td>Prepare the Nodes</td>
<td>8</td>
</tr>
<tr>
<td>Conduct the Experiment</td>
<td>12</td>
</tr>
<tr>
<td>Scenario 1 (Normal Traffic)</td>
<td>12</td>
</tr>
<tr>
<td>Scenario 2 (MITM Attack)</td>
<td>16</td>
</tr>
<tr>
<td>Working:</td>
<td>18</td>
</tr>
</tbody>
</table>
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.

Select a Profile

<table>
<thead>
<tr>
<th>MITMController</th>
<th>Created By: hongdal</th>
</tr>
</thead>
<tbody>
<tr>
<td>My Profiles -</td>
<td>Project: SANTS2019Lab1</td>
</tr>
<tr>
<td>MITMController</td>
<td>Latest Version: 1</td>
</tr>
<tr>
<td>SANTS2019Lab1 -</td>
<td>Last Updated: 2019-02-23 19:00:16</td>
</tr>
<tr>
<td>MITMController</td>
<td>Description: Floodlight SDN controller</td>
</tr>
</tbody>
</table>

Figure 1: Start an experiment using the MITMController profile. Then Click “Select Profile” button to proceed.
Figure 2: Click “Next” button to proceed.

Figure 3: Choose a cluster and click “Next” to proceed.
Figure 4: Click “Finish” to proceed.

Figure 5: The controller is successfully running.
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`.

Figure 7: setup controller.
1) Switch to root. Type “`sudo su -`”
2) Type “`wget https://people.cs.clemson.edu/~hongdal/set_floodlight.sh`”
Figure 8: Run the scripts.
1) Type 
   \texttt{chmod +x set\_floodlight.sh}  
2) Type 
   \texttt{./set\_floodlight.sh}  

Figure 9: Launch the controller.
1) Type \texttt{cd floodlight}  
2) Type \texttt{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.

Figure 10: Upload the topology when creating the profile.

Figure 11: Click “Edit topology” to configure the SDN controller.
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.”
Figure 13: Click “instantiate” to proceed.

Figure 14: Choose a cluster. Click “Next” to proceed.
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 “
   ```bash
   wget https://people.cs.clemson.edu/~hongdal/set_ovs.sh
   ```
   on all the nodes.
3) change permission and run the script.
   ```bash
   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.
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.

```
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)
```

Figure 18: Check the experimental IP of User1.
### Figure 19: Check the experimental IP of User2.

```
lo  Link encap:Local Loopback  
inet addr:127.0.0.1  Mask:255.0.0.0  
in6 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@user2:~#
```

### Figure 20: PINF from User1 to User2.

```
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=2.12 ms  
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=1.58 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=1.66 ms  
```

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

```
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.

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.
Figure 23: Packets observed from Relay Node

```
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 24: Packets observed from Switch4 or Switch6

```
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
21:55:21.290876 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2752, seq 12, length 64
21:55:22.291839 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2752, seq 12, length 64
21:55:22.292335 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2752, seq 13, length 64
21:55:23.292607 IP 10.10.10.2 > 10.10.10.1: ICMP echo reply, id 2752, seq 13, length 64
21:55:23.293120 IP 10.10.10.1 > 10.10.10.2: ICMP echo request, id 2752, seq 14, length 64
6 packets captured
6 packets received by filter
0 packets dropped by kernel
```

```c
^C
```
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.