# Lab 6. Host Hijacking Attacks in SDN

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment Task Design:</td>
<td>2</td>
</tr>
<tr>
<td>Submission:</td>
<td>2</td>
</tr>
<tr>
<td>Conducting the lab</td>
<td>2</td>
</tr>
<tr>
<td>Step 1: Create a profile for the network and the SDN controller</td>
<td>3</td>
</tr>
<tr>
<td>Step 2: Floodlight Installation in controller node.</td>
<td>3</td>
</tr>
<tr>
<td>Step 3: Install OpenVSwitch and setup bridge on all 4 nodes of the network.</td>
<td>5</td>
</tr>
<tr>
<td>Step 4: Test the connectivity</td>
<td>6</td>
</tr>
<tr>
<td>Step 5: Spoof an inactive node’s MAC address</td>
<td>6</td>
</tr>
<tr>
<td>Step 6: Results</td>
<td>7</td>
</tr>
<tr>
<td>Reference</td>
<td>9</td>
</tr>
</tbody>
</table>
Experiment Task Design:

**Problem Definition:** The controller has a vast amount of important data, such as topological information, device information, and link information, all of which can be compromised by attackers. To accomplish this, attackers exploit the host tracking service in the controller. They can tamper with host location information to break through the controller and impersonate the target host. In that case, all traffic on the target host will be routed to the attacker’s host. TopoGuard [5] was the first to demonstrate network poisoning attacks designed to compromise the network topology information based on the Link Layer Discovery Protocol (LLDP). This is one example of the many possible network poisoning attacks in SDN.

**Submission:**

Take screenshots of all the steps involved and explain in one or two paragraphs. Describe why the attack can be performed.

Students can refer to the link ([http://docs.cloudlab.us/cloudlab-tutorial.html](http://docs.cloudlab.us/cloudlab-tutorial.html)) for more details about creating profiles on CloudLab. Students should have an account of either CloudLab or GENI or any other federated services like EmuLab to access CloudLab. CloudLab login page: [https://www.cloudlab.us/login.php](https://www.cloudlab.us/login.php)

**Conducting the lab**

**Step 1: Create a profile for the network and the SDN controller**

Students can use the link below to copy the profile for the Controller and make changes in it. Once instantiated, install the dependencies for floodlight and install Floodlight version 1.2 [https://www.cloudlab.us/p/2da0db8c-63f4-11e8-b228-90e2ba22fee4](https://www.cloudlab.us/p/2da0db8c-63f4-11e8-b228-90e2ba22fee4)

Students can use the link below to copy the profile for the network and make changes in it. Make sure to provide the IP address of the controller for each link under Enable Openflow field. [https://www.cloudlab.us/p/cbf5cc34-630e-11e8-b228-90e2ba22fee4](https://www.cloudlab.us/p/cbf5cc34-630e-11e8-b228-90e2ba22fee4)
Step 2: Floodlight installation in Controller node

1) Open a shell window of the controller node.

   **Note:** Use Floodlight version 1.2

2) Install Floodlight using the following steps:

   **Get sudo user privileges**
   
   `sudo su`

   **Update APT repo**
   
   `apt-get update`

   **Install java 8**
   
   `apt-get install default-jdk`
   `apt-get install default-jre`

   **Install dependencies**
   
   `apt-get install build-essential ant maven python-dev`

   **Install Floodlight**
   
   `git clone https://github.com/floodlight/floodlight.git -b v1.2`
   `cd floodlight`
   `git submodule init`
   `git submodule update`
   `ant`
   `sudo mkdir /var/lib/floodlight`
   `sudo chmod 777 /var/lib/floodlight`

   At this stage, there are some required changes in one of the floodlight modules. Once the changes have been implemented, build again using “ant” and then run the controller.

3) `cd to /floodlight/src/main/java/net/floodlightcontroller/devicemanager/internal/`
4) Open DeviceManagerImpl.java

5) Locate the isEntityAllowed() method. Before the return statement add a print statement to observe the output every time it’s invoked.
6) cd into `/floodlight`

7) Run “`ant`”

8) Run “`java -jar target/floodlight.jar`”

**Step 3: Install OpenVSwitch and setup bridge on all 4 nodes of the network.**

1) Open a new terminal in **all** nodes of the network

2) Run “`sudo apt-get install -y openvswitch-switch`” to install Open VSwitch in **all** nodes of the network

3) Use the following commands to setup a bridge on each node and connect it to SDN controller. The only change will be in last command for every host.

   Node 0: `ifconfig br0 10.10.5.1 netmask 255.255.0.0 up`
   Node 1: `ifconfig br0 10.10.6.1 netmask 255.255.0.0 up`
   Node 2: `ifconfig br0 10.10.7.1 netmask 255.255.0.0 up`
   Node 3: `ifconfig br0 10.10.8.1 netmask 255.255.0.0 up`

   In the command, **“`ovs-vsctl set-controller br0 tcp:128.110.99:141:6653`”** change the IP address to the IP of controller.

   ```
   sudo su
   ovs-vsctl add-br br0
   ovs-vsctl add-port br0 eth1
   ovs-vsctl add-port br0 eth2
   ifconfig eth1 0
   ifconfig eth2 0
   ovs-vsctl set-controller br0 tcp:128.110.99:141:6653
   ifconfig br0 10.10.5.1 netmask 255.255.0.0 up
   ```
Other OVS commands:

ovs-vsctl: Used for configuring the ovs-vswitchd configuration database (known as ovs-db)
e.g. To delete a bridge: “ovs-vsctl del-br ovs-lan1”

ovs-ofctl: A command line tool for monitoring and administering OpenFlow switches
e.g. To print the OVS flow rules “ovs-ofctl dump-flows ovs-lan2 -O OpenFlow13”

Step 4: Test the connectivity

1) Ping node 2 from node 0

2) Use “tcLOGINit -i br0” on node 2 to see ping packets

Step 5: Spoof an inactive node’s MAC address

In our scenario, Node 2 will be disconnected from the Controller. Which means Node 2 is no longer part of the network.

We will configure Node 1 and change the IP/MAC address of br0 of Node 1 to IP/MAC of br0 of Node2.

On Node 2, perform the following tasks:
  - ifconfig
  - Note the IP and MAC address of br0
  - “sudo ovs-vsctl del-br br0”
On Node 1, perform the following tasks:

- Run this command: `ovs-vsctl set bridge br0 other-config:hwaddr="/MAC_ADDRESS_OF_BR0_OF_NODE2/"
- Run `ifconfig br0 IP_ADDR_OF_BR0_OF_NODE2 netmask 255.255.0.0 up`

Step 6: Results

Once node 1 is configured, ping again from node 0 to the old IP of node 2 (in our case 10.10.7.1). Start `tcpdump` on node 1's br0. You will notice the packets are received by node 1. This is obvious because we changed the IP/MAC address of node 1 to that of node 2.
Throughout this process if you notice the output of floodlight, you will see the print statement you added in the `isEntityAllowed`, every time you add or remove a device from the network.

**Explanation:**

If you analyze the code in `DeviceManagerImpl` class for `isEntityAllowed()` method, you will notice that it always returns true. This method is also used by the Routing module to make routing decisions. The purpose of that method is to keep a check on spoofing attacks. But logic has not been written for it. Because of this, it is very easy for an attacker to impersonate another host’s IP/MAC address and get access to all the packets destined to that location. All controllers are required to provide device migration service where a host changes its location and still gets its packets. But they are also supposed to provide spoofing protection service.
Reference