Lab 6. Host Hijacking Attacks in SDN

Contents

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.

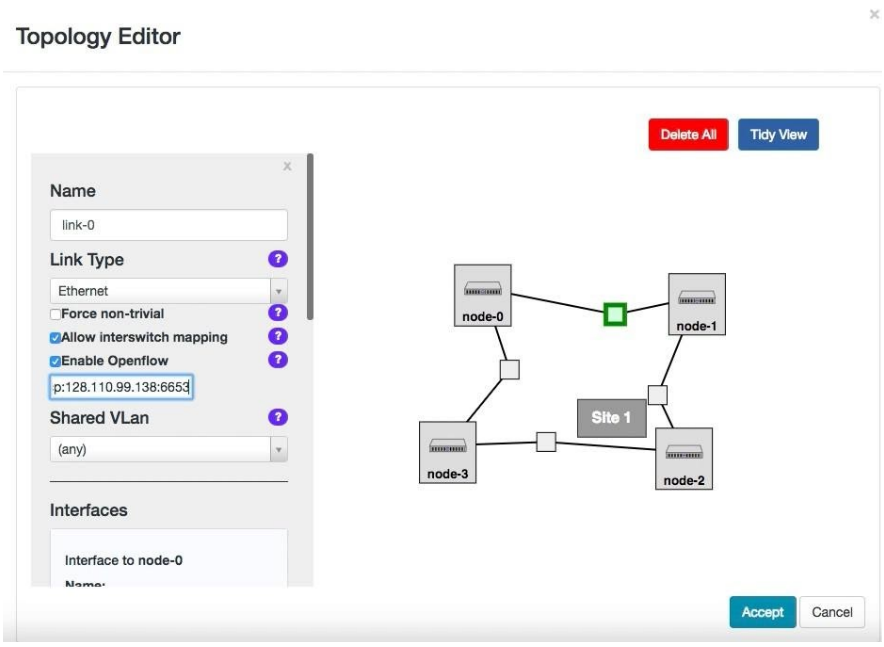
Students can refer to the link (http://docs.cloudlab.us/cloudlab-tutorial.html) for more details about creating profiles on CloudLab. Students should have an account of either CloudLab or GENI or any other federated services like EmuLab to access CloudLab. CloudLab login page: https://www.cloudlab.us/login.php

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

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



Topology Editor

Name

link-0

Link Type

Ethernet

□ Force non-trivial

☑ Allow interswitch mapping

☑ Enable Openflow

p:128.110.99.138:6653

Shared VLan

(any)

Interfaces

Interface to node-0

Name:

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

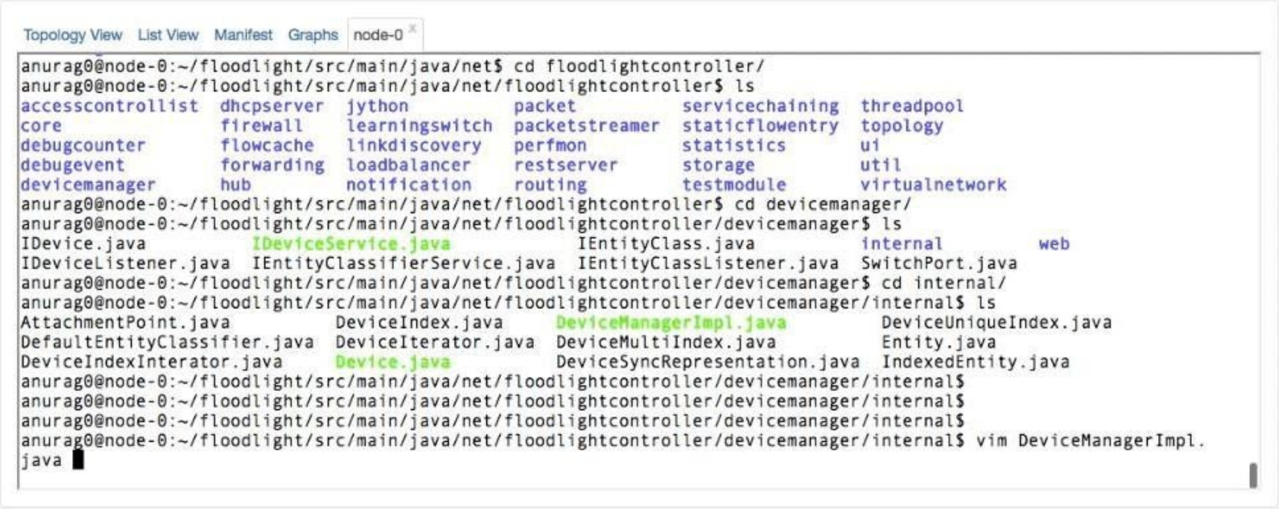
```
apt-get install build-essential ant maven python-dev
```

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.

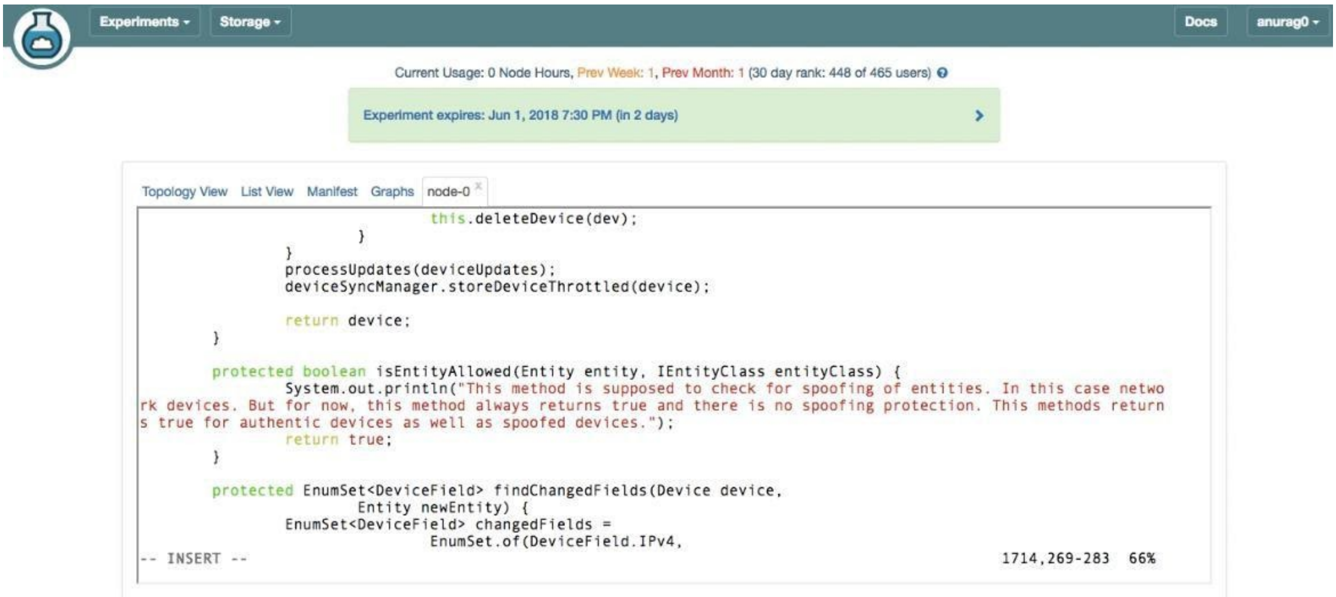
3) cd to **/floodlight/src/main/java/net/floodlightcontroller/devicemanager/internal/**



```
anurag0@node-0:~/floodlight/src/main/java/net$ cd floodlightcontroller/
anurag0@node-0:~/floodlight/src/main/java/net/floodlightcontroller$ ls
accesscontrollist  dhcpserver  jython          packet         servicechaining  threadpool
core               firewall    learningswitch  packetstreamer staticflowentry  topology
debugcounter       flowcache   linkdiscovery   perfmon        statistics       ui
debugevent         forwarding  loadbalancer    restserver     storage          util
devicemanager      hub         notification    routing        testmodule       virtualnetwork
anurag0@node-0:~/floodlight/src/main/java/net/floodlightcontroller$ cd devicemanager/
anurag0@node-0:~/floodlight/src/main/java/net/floodlightcontroller/devicemanager$ ls
IDevice.java          IDeviceService.java          IEntityClass.java          internal        web
IDeviceListener.java  IEntityClassifierService.java  IEntityClassListener.java  SwitchPort.java
anurag0@node-0:~/floodlight/src/main/java/net/floodlightcontroller/devicemanager$ cd internal/
anurag0@node-0:~/floodlight/src/main/java/net/floodlightcontroller/devicemanager/internal$ ls
AttachmentPoint.java          DeviceIndex.java     DeviceManagerImpl.java          DeviceUniqueIndex.java
DefaultEntityClassifier.java  DeviceIterator.java  DeviceMultiIndex.java           Entity.java
DeviceIndexInterator.java     Device.java          DeviceSyncRepresentation.java  IndexedEntity.java
anurag0@node-0:~/floodlight/src/main/java/net/floodlightcontroller/devicemanager/internal$
anurag0@node-0:~/floodlight/src/main/java/net/floodlightcontroller/devicemanager/internal$
anurag0@node-0:~/floodlight/src/main/java/net/floodlightcontroller/devicemanager/internal$
anurag0@node-0:~/floodlight/src/main/java/net/floodlightcontroller/devicemanager/internal$ vim DeviceManagerImpl.
java
```

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.



Current Usage: 0 Node Hours, Prev Week: 1, Prev Month: 1 (30 day rank: 448 of 465 users)

Experiment expires: Jun 1, 2018 7:30 PM (in 2 days)

```
                                        this.deleteDevice(dev);
                              }
                    }
                    processUpdates(deviceUpdates);
                    deviceSyncManager.storeDeviceThrottled(device);

                    return device;
          }

          protected boolean isEntityAllowed(Entity entity, IEntityClass entityClass) {
                    System.out.println("This method is supposed to check for spoofing of entities. In this case netwo
rk devices. But for now, this method always returns true and there is no spoofing protection. This methods return
s true for authentic devices as well as spoofed devices.");
                    return true;
          }

          protected EnumSet<DeviceField> findChangedFields(Device device,
                              Entity newEntity) {
                    EnumSet<DeviceField> changedFields =
                                        EnumSet.of(DeviceField.IPv4,
-- INSERT --                                                                              1714,269-283   66%
```

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:

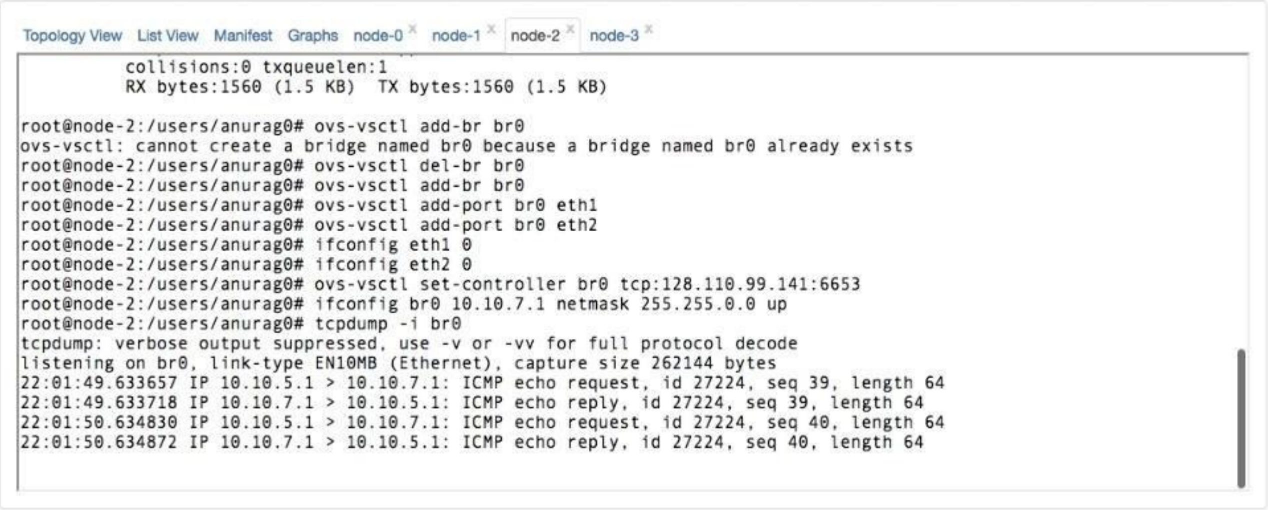
ovs-vsctl: Used for configuring the ovs-vswitchd configuration database (known as ovs-db)
e.g. To delete a bridge: “**ovs-vsctl del-br ovs-lan1**“

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 “**tcpdump -i br0**” on **node 2** to see ping packets



```
          collisions:0 txqueuelen:1
          RX bytes:1560 (1.5 KB)  TX bytes:1560 (1.5 KB)

root@node-2:/users/anurag0# ovs-vsctl add-br br0
ovs-vsctl: cannot create a bridge named br0 because a bridge named br0 already exists
root@node-2:/users/anurag0# ovs-vsctl del-br br0
root@node-2:/users/anurag0# ovs-vsctl add-br br0
root@node-2:/users/anurag0# ovs-vsctl add-port br0 eth1
root@node-2:/users/anurag0# ovs-vsctl add-port br0 eth2
root@node-2:/users/anurag0# ifconfig eth1 0
root@node-2:/users/anurag0# ifconfig eth2 0
root@node-2:/users/anurag0# ovs-vsctl set-controller br0 tcp:128.110.99.141:6653
root@node-2:/users/anurag0# ifconfig br0 10.10.7.1 netmask 255.255.0.0 up
root@node-2:/users/anurag0# tcpdump -i br0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on br0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:01:49.633657 IP 10.10.5.1 > 10.10.7.1: ICMP echo request, id 27224, seq 39, length 64
22:01:49.633718 IP 10.10.7.1 > 10.10.5.1: ICMP echo reply, id 27224, seq 39, length 64
22:01:50.634830 IP 10.10.5.1 > 10.10.7.1: ICMP echo request, id 27224, seq 40, length 64
22:01:50.634872 IP 10.10.7.1 > 10.10.5.1: ICMP echo reply, id 27224, seq 40, length 64
```

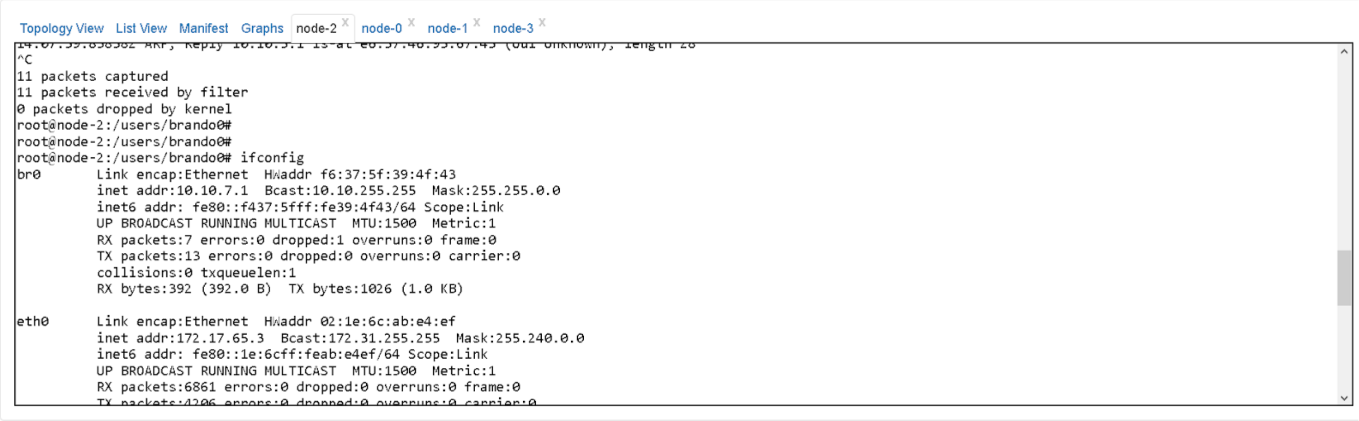
Step 5: Spoof an inactive node’s MAC address

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**”

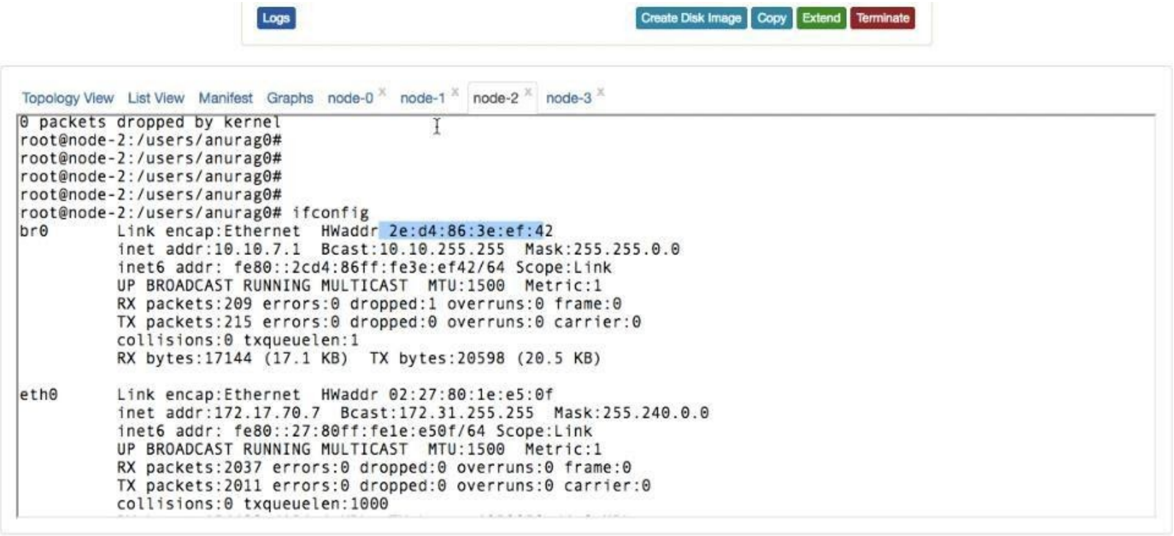


```
14:07:39.858382 ARP, Reply 10.10.5.1 is-at e6:37:46:95:67:43 (oui Unknown), length 28
^C
11 packets captured
11 packets received by filter
0 packets dropped by kernel
root@node-2:/users/brando0#
root@node-2:/users/brando0#
root@node-2:/users/brando0# ifconfig
br0       Link encap:Ethernet  HWaddr f6:37:5f:39:4f:43
          inet addr:10.10.7.1  Bcast:10.10.255.255  Mask:255.255.0.0
          inet6 addr: fe80::f437:5fff:fe39:4f43/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:1 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:392 (392.0 B)  TX bytes:1026 (1.0 KB)

eth0      Link encap:Ethernet  HWaddr 02:1e:6c:ab:e4:ef
          inet addr:172.17.65.3  Bcast:172.31.255.255  Mask:255.240.0.0
          inet6 addr: fe80::1e:6cff:feab:e4ef/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6861 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4206 errors:0 dropped:0 overruns:0 carrier:0
```

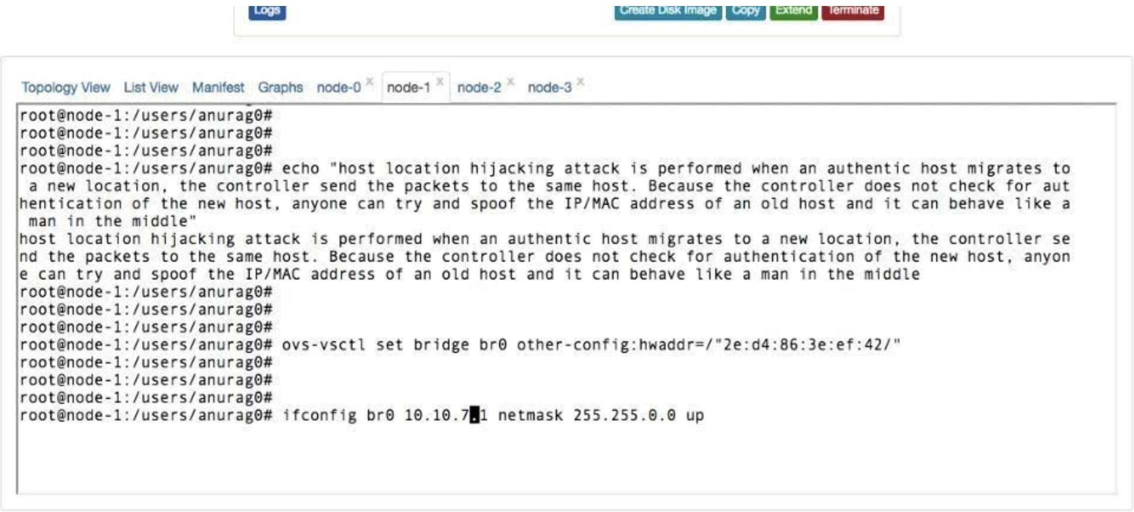
On Node 1, perform the following tasks:

- 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**



```
0 packets dropped by kernel
root@node-2:/users/anurag0#
root@node-2:/users/anurag0#
root@node-2:/users/anurag0#
root@node-2:/users/anurag0#
root@node-2:/users/anurag0# ifconfig
br0       Link encap:Ethernet  HWaddr 2e:d4:86:3e:ef:42
          inet addr:10.10.7.1  Bcast:10.10.255.255  Mask:255.255.0.0
          inet6 addr: fe80::2cd4:86ff:fe3e:ef42/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:209 errors:0 dropped:1 overruns:0 frame:0
          TX packets:215 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:17144 (17.1 KB)  TX bytes:20598 (20.5 KB)

eth0      Link encap:Ethernet  HWaddr 02:27:80:1e:e5:0f
          inet addr:172.17.70.7  Bcast:172.31.255.255  Mask:255.240.0.0
          inet6 addr: fe80::27:80ff:fe1e:e50f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2037 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2011 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
```

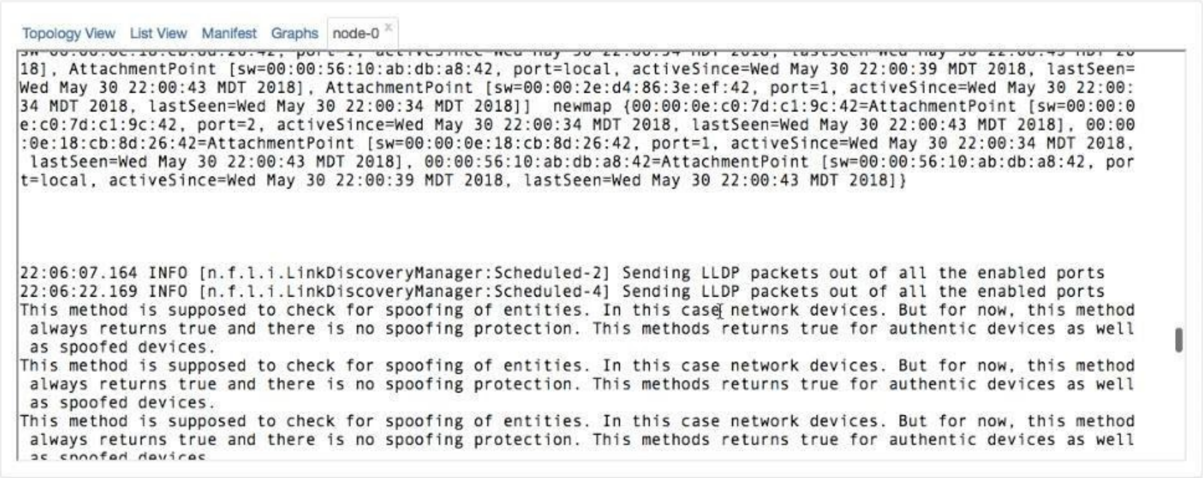


```
root@node-1:/users/anurag0#
root@node-1:/users/anurag0#
root@node-1:/users/anurag0#
root@node-1:/users/anurag0# echo "host location hijacking attack is performed when an authentic host migrates to
 a new location, the controller send the packets to the same host. Because the controller does not check for aut
hentication of the new host, anyone can try and spoof the IP/MAC address of an old host and it can behave like a
 man in the middle"
host location hijacking attack is performed when an authentic host migrates to a new location, the controller se
nd the packets to the same host. Because the controller does not check for authentication of the new host, anyon
e can try and spoof the IP/MAC address of an old host and it can behave like a man in the middle
root@node-1:/users/anurag0#
root@node-1:/users/anurag0#
root@node-1:/users/anurag0#
root@node-1:/users/anurag0# ovs-vsctl set bridge br0 other-config:hwaddr=/"2e:d4:86:3e:ef:42/"
root@node-1:/users/anurag0#
root@node-1:/users/anurag0#
root@node-1:/users/anurag0#
root@node-1:/users/anurag0# ifconfig br0 10.10.7.1 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 n**ode 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.



```
18], AttachmentPoint [sw=00:00:56:10:ab:db:a8:42, port=local, activeSince=Wed May 30 22:00:39 MDT 2018, lastSeen=
Wed May 30 22:00:43 MDT 2018], AttachmentPoint [sw=00:00:2e:d4:86:3e:ef:42, port=1, activeSince=Wed May 30 22:00:
34 MDT 2018, lastSeen=Wed May 30 22:00:34 MDT 2018]]  newmap {00:00:0e:c0:7d:c1:9c:42=AttachmentPoint [sw=00:00:0
e:c0:7d:c1:9c:42, port=2, activeSince=Wed May 30 22:00:34 MDT 2018, lastSeen=Wed May 30 22:00:43 MDT 2018], 00:00
:0e:18:cb:8d:26:42=AttachmentPoint [sw=00:00:0e:18:cb:8d:26:42, port=1, activeSince=Wed May 30 22:00:34 MDT 2018,
 lastSeen=Wed May 30 22:00:43 MDT 2018], 00:00:56:10:ab:db:a8:42=AttachmentPoint [sw=00:00:56:10:ab:db:a8:42, por
t=local, activeSince=Wed May 30 22:00:39 MDT 2018, lastSeen=Wed May 30 22:00:43 MDT 2018]}



22:06:07.164 INFO [n.f.l.i.LinkDiscoveryManager:Scheduled-2] Sending LLDP packets out of all the enabled ports
22:06:22.169 INFO [n.f.l.i.LinkDiscoveryManager:Scheduled-4] Sending LLDP packets out of all the enabled ports
This method is supposed to check for spoofing of entities. In this case network devices. But for now, this method
 always returns true and there is no spoofing protection. This methods returns true for authentic devices as well
 as spoofed devices.
This method is supposed to check for spoofing of entities. In this case network devices. But for now, this method
 always returns true and there is no spoofing protection. This methods returns true for authentic devices as well
 as spoofed devices.
This method is supposed to check for spoofing of entities. In this case network devices. But for now, this method
 always returns true and there is no spoofing protection. This methods returns true for authentic devices as well
 as spoofed devices
```

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

Sungmin Hong*, Lei Xu*, Haopei Wang, Guofei Gu. "Poisoning Network Visibility in Software- Defined Networks: New Attacks and Countermeasures." In Proc. of 22nd Annual Network & Distributed System Security Symposium (NDSS'15), San Diego, CA, USA. February 2015. (*co-first author) http://faculty.cs.tamu.edu/guofei/paper/TopoGuard_NDSS15.pdf