

1.1 Scenario: Network Packet Sniffing and Security Analysis

Goal

The goal of this experiment was to analyze network packet flow and how attackers can exploit packet sniffing. Using Mininet and Wireshark, we saw how hosts transmitted packets.

Experiment

- Setup: A simple Mininet topology was created between H1 and H2 and a switch
- H1 was used to send ICMP packets to H2
- Wireshark was used to monitor the switch interface and capture packets
- The source and destination IP addresses were 10.0.0.1 (H1) and 10.0.0.2 (H2)
- ICMP protocol was used
- The captured packets included session IDs (Id=0x3917)

```
p4@p4: ~  
File Actions Edit View Help  
p4@p4: ~  
p4@p4:~$ sudo mn --controller=remote,ip=127.0.0.1,port=6633 --topo=tree,2,2  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2 h3 h4  
*** Adding switches:  
s1 s2 s3  
*** Adding links:  
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)  
*** Configuring hosts  
h1 h2 h3 h4  
*** Starting controller  
c0  
*** Starting 3 switches  
s1 s2 s3 ...  
*** Starting CLI:  
mininet> pingall  
*** Ping: testing ping reachability  
h1 -> h2 h3 h4  
h2 -> h1 h3 h4  
h3 -> h1 h2 h4  
h4 -> h1 h2 h3  
*** Results: 0% dropped (12/12 received)  
mininet> h1 ping h2  
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.  
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.74 ms
```

```
p4@p4: ~/pox  
File Actions Edit View Help  
p4@p4: ~/pox  
p4@p4:~/pox$ python3 pox.py forwarding.l2_learning  
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.  
WARNING:version:Support for Python 3 is experimental.  
INFO:core:POX 0.7.0 (gar) is up.  
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected  
INFO:openflow.of_01:[00-00-00-00-00-01 2] closed  
INFO:openflow.of_01:[00-00-00-00-00-01 5] connected  
INFO:openflow.of_01:[00-00-00-00-00-03 6] connected  
INFO:openflow.of_01:[00-00-00-00-00-02 7] connected  
[]
```

```
p4@p4: ~  
File Actions Edit View Help  
p4@p4: ~  
*** Results: 0% dropped (12/12 received)  
mininet> h1 ping h2  
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.  
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.74 ms  
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.397 ms  
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.077 ms  
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.067 ms  
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.254 ms  
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.090 ms  
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.082 ms  
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.161 ms  
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.088 ms  
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.091 ms  
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.121 ms  
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.124 ms  
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.123 ms  
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.072 ms  
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.072 ms  
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.075 ms  
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.103 ms  
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.075 ms  
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.085 ms  
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.068 ms  
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=0.076 ms  
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.075 ms  
64 bytes from 10.0.0.2: icmp_seq=23 ttl=64 time=0.110 ms
```

The image shows a Wireshark capture of ICMP traffic on interface s2-eth1. The packet list displays a series of Echo (ping) requests and replies between 10.0.0.1 and 10.0.0.2. Packet 337 is highlighted, showing a request from 10.0.0.1 to 10.0.0.2. The packet details pane shows the Ethernet II, Internet Protocol Version 4, and Internet Control Message Protocol (ICMP) fields. The packet bytes pane shows the raw data in hexadecimal and ASCII.

Packet 337 details:

- Ethernet II, Src: 3a:92:fc:f8:ee:57 (3a:92:fc:f8:ee:57), Dst: 56:45:89:cf:50:fd (56:45:89:cf:50:fd)
- Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
- Internet Control Message Protocol
- Type: 0 (Echo (ping) reply)
- Code: 0
- Checksum: 0x5e0e [correct]
- [Checksum Status: Good]
- Identifier (BE): 14600 (0x3908)
- Identifier (LE): 2105 (0x0839)
- Sequence number (BE): 159 (0x009f)
- Sequence number (LE): 40704 (0x9f00)
- [Request frame: 337]
- [Response time: 0.061 ms]
- Timestamp from icmp data: Mar 18, 2025 09:46:33.000000000 UTC
- Timestamp from icmp data (relative): 0.872236602 seconds
- Data (48 bytes)

Observations

ICMP packets were transmitted successfully between H1 and H2. Moreover, packet sniffing allows attackers to view network topology details, such as active IP addresses and protocol behavior.

Challenges

Capturing relevant packets. We resolved it by using display filters in Wireshark to focus on ICMP traffic.

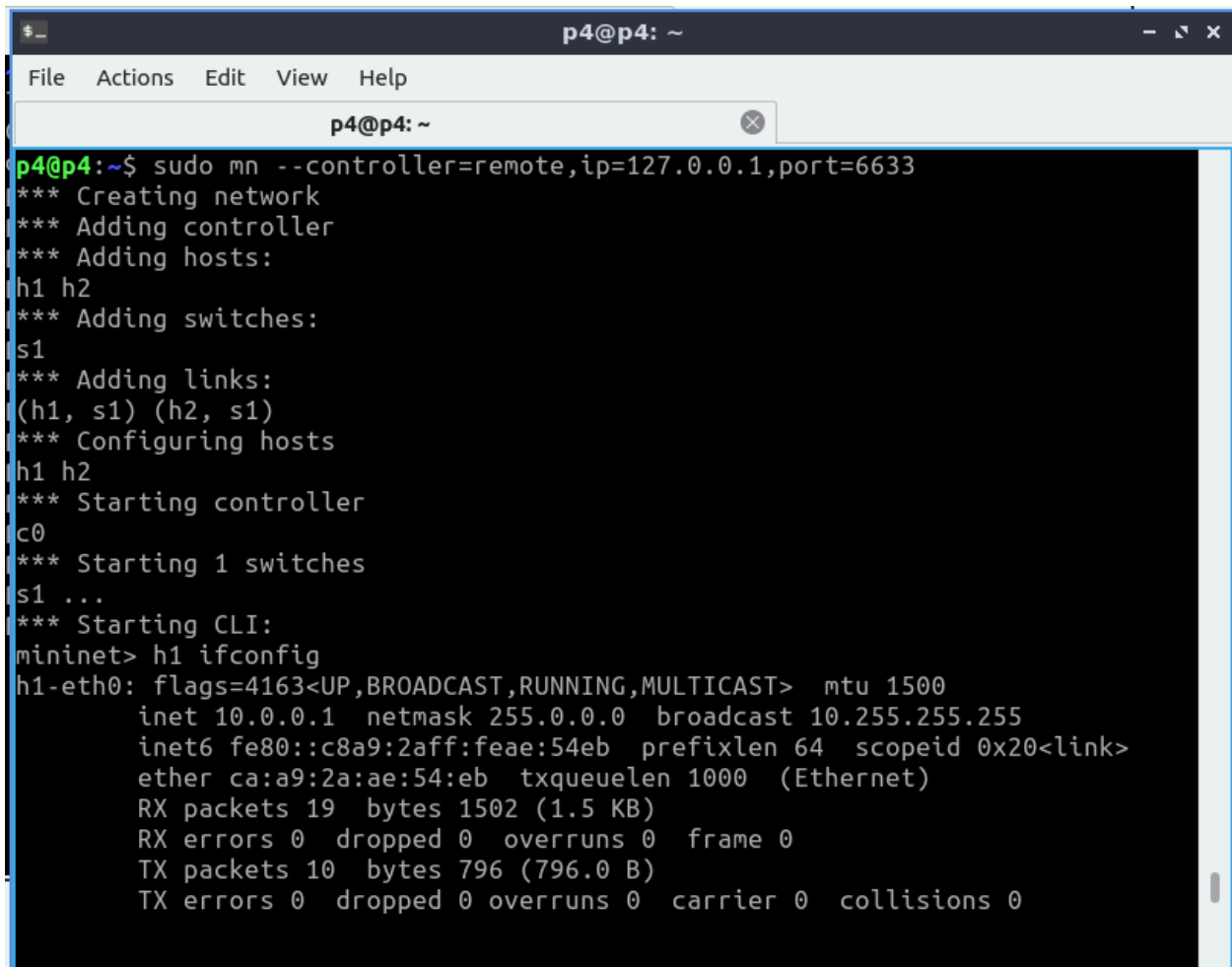
1.3 Scenario: Simulating MAC Spoofing in a Network

Goal

The goal of this experiment was to show how MAC address spoofing can be used to bypass security based on MAC filtering. Altering a host's MAC address can allow attackers to disguise their identity and evade network restrictions.

Experiment

- Setup: We configured a Mininet topology with 2 hosts and a switch
- We recorded the default MAC address of h1, which we then changed to 00:00:00:00:00:03
- We then verified that the MAC address of h1 was indeed changed



```
p4@p4: ~  
File Actions Edit View Help  
p4@p4: ~  
p4@p4:~$ sudo mn --controller=remote,ip=127.0.0.1,port=6633  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2  
*** Adding switches:  
s1  
*** Adding links:  
(h1, s1) (h2, s1)  
*** Configuring hosts  
h1 h2  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ...  
*** Starting CLI:  
mininet> h1 ifconfig  
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255  
    inet6 fe80::c8a9:2aff:feae:54eb prefixlen 64 scopeid 0x20<link>  
    ether ca:a9:2a:ae:54:eb txqueuelen 1000 (Ethernet)  
    RX packets 19  bytes 1502 (1.5 KB)  
    RX errors 0  dropped 0  overruns 0  frame 0  
    TX packets 10  bytes 796 (796.0 B)  
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```

mininet> h1 ifconfig h1-eth0 hw ether 00:00:00:00:00:03
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::cec:82ff:feec:c576 prefixlen 64 scopeid 0x20<link>
    ether 00:00:00:00:00:03 txqueuelen 1000 (Ethernet)
    RX packets 19 bytes 1502 (1.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 796 (796.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=13.4 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.475 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.079 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.062 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.114 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.094 ms
^Z
[8]+  Stopped                  sudo mn --controller=remote,ip=127.0.0.1,port=6
633
p4@p4:~/tutorials/exercises/basic$

```

Observations

The MAC address change was successfully applied, allowing h1 to assume a new identity on the network. If any MAC filtering mechanisms solely relied on source MAC addresses, they would be ineffective against spoofing. Hence, networks should use more dynamic security measures such as anomaly detection to detect and prevent such attacks.

Conclusion

Our experiments highlight that packet sniffing shows how information like network topology can be revealed by analyzing traffic, while MAC spoofing allows for identity evasion. Security implementations should take these vulnerabilities into account and must include measures like encryption, authentication, and anomaly detection to mitigate these inherent risks.