Cybersecurity Lab, CSE 3140
spring 2022

Network Protocols and Sniffing

Section # : 1
Team #: 1
Names: Jared Moore

Username: cse
Pass: 4uY;X?Xp

A key way for an attacker to gain access to an organization's resources is through the network, which is usually connected to the internet. Organizations try to prevent as many attacks as possible in a timely manner. Therefore, it is critical to understand the architecture of the system, the network design, communication flows and how to protect against possible attacks. In this lab we'll start to look at network security. Almost everything we do on computers requires speaking to some other computer over a network (often through the Internet). Attackers can observe this traffic and try to figure out what people are doing. ***Sniffing and Spoofing*** are core fundamental skills for most networking attacks. As data travels over the network, a sniffer program is capable of capturing and analyzing each protocol data unit. There are many programs that do sniffing and spoofing. An extensively used tool is Wireshark which we will be using during this lab. Wireshark is a software protocol analyzer, or "packet sniffer" application, used for network troubleshooting, analysis, software and protocol development, and education. At the most basic level, Wireshark monitors all network traffic that is visible to your computer. It has lots of features for making analysis of this traffic easier. You'll need to use MobaXterm for this lab to get X windows forwarding. However, it is also very important to be able to build your own tool. At some point you may have a specific goal that is not supported by any tool. In this lab you will learn basic networking concepts and how the OS handles your packet to be ready to go over the network. You are also going to learn how to write your own sniffing or spoofing program using python. For those of you that haven't taken a networking class this lab will naturally touch on networking concepts. We encourage you to look up concepts you don't understand and ask questions.

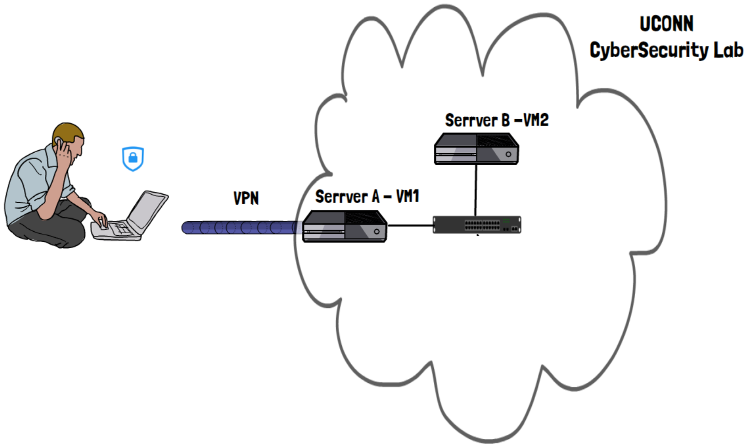


Fig.1: lab topology network

For this lab, we are providing the network configuration shown in fig 1. Note that these virtual machines are completely new, no changes that you made in previous labs persist to this environment. You have **2 VMs** assigned to your group. The main VM you have access to is **ServerA-VM$_1$**, see fig.1. **ServerA-VM$_1$** IP address is based on a formula that depends on both your section number and your group number. The overall IP is ***172.16.51.<20*section number + group id>***. So, a student in group 4 and section 3 will use IP address **172.16.51.64.** Start by SSHing into your assigned virtual machine (as before) the username: ***cse*** and the password: ***cse3140***, **change your password as soon as you login**.

Please add any script **Use screenshots whenever possible.**

Question 1 (10 points):

Sniffing and spoofing skills are important skills to learn. We will start by learning how to write a simple sniffing program using python and Scapy. Scapy is a powerful module for packet manipulation. It can help packet parsing, sniffing, spoofing, sending and receiving. The following example can help you write your first sniffing program. You can have more advanced skills if you tried the same programs in C. However, we are interested in learning the fundamental skill, so we will follow an easy way of implementation.

We will have your group co-operate with some other group **(only for this question)**, **group A** is required to write a listener programmer ***"listenerA.py"*** that will be receiving a secret message from **group B**. Group B will be writing a sender program **"senderB.py"** that will send that secret message. Next we will switch roles. Group A will write the sender program **"senderA.py"** to send a message to the listener program ***"listenerB.py"*** that groupB should be using to listen to the secret message. Do not share the exchanged messages, but you can acknowledge receiving a message. Report the string you sent and received from the other group. A helpful sample for sending or receiving using python is shown below:

```
#-------------------------------------------------------------------------------
# Sender

import socket

UDP_IP = "172.16.50.226"
UDP_PORT = 3300
MESSAGE = b'secret_message'
print("UDP target IP: %s" % UDP_IP)
print("UDP target port: %s" % UDP_PORT)
print("message: %s" % MESSAGE)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))

#-------------------------------------------------------------------------------
# Receiver

IP= "0.0.0.0"
Port = 3300
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP,Port))
while True:
    data, (ip,port) = sock.recvfrom(1024)
    print("Sender:{} and port:{}".format(ip,port))
    print("received message:{}".format(data))
```

```
cse@cse3140-HVM-domU:~/project2-cse$ python3 listenerA.py
Sender:172.16.51.27 and port:57742
received message:b'secret_message'
```

Received this transmission from group 7

Question 2 (10 points): CONDUCTING RECONNAISSANCE

Let's extract some networking information about your running VM. SSH into your assigned ***VM-serverA***. A nice utility to learn is ***ifconfig***. Use the **ifconfig** to verify your IPv4 address.

```
echoServer.py  helloClient.py  listenerA.py  python3  senderA.py
cse@cse3140-HVM-domU:~/project2-cse$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.51.21  netmask 255.255.252.0  broadcast 172.16.51.255
        inet6 fe80::90ab:9c80:29ce:4836  prefixlen 64  scopeid 0x20<link>
        inet6 fe80::a28e:d097:df3b:c5b7  prefixlen 64  scopeid 0x20<link>
        inet6 fe80::27a2:67d6:1c1b:e708  prefixlen 64  scopeid 0x20<link>
        ether 52:30:73:e9:9d:a1  txqueuelen 1000  (Ethernet)
        RX packets 135472627  bytes 8719878953 (8.7 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2222057  bytes 184745883 (184.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 806367  bytes 87536112 (87.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 806367  bytes 87536112 (87.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

cse@cse3140-HVM-domU:~/project2-cse$
```

***1)** What MAC **address** is assigned for this specific NIC?*

5230.73e9.9da1

***2)** What is your interface **(name)**?*

eth0

CONDUCTING RECONNAISSANCE, Before launching an active attack, an attacker usually conducts a reconnaissance practice to acquire more information on the target network/hosts. **Let's take a look at some examples of network scans next.** The **ping** command is a way of determining if a host exists on the network. Most computers will respond to a ping (this is turned off in many security conscious organizations). The command is: ***ping <ip_address>.*** Ping all of the machines on this subnet (write a script rather than doing this manually).

What ranges are occupied? What ranges did you not receive a response from?

172.16.48.1

172.16.48.8-10

172.16.48.18-35,37,40-46,48-55

172.16.48.99

172.16.50.1

172.16.50.100-144,151-177,191,216-220

172.16.51.18,19,21-32,41-45,47-53,55,60-76,80-95

Once the attacker identifies a host that is up and responding, the next step is to use a port scan to see whether the host has any open and listening. Usually, we scan the well-known ports that are in the range 1-1023; however, you can set the application to scan a wider range of ports.

You are now going to see what active services exist on the Server that you are SSHing to. You will be building a very simple port scanner (a very basic version of https://nmap.org). You can use the command ***telnet*** to check what ports are open on your server A. The basic command is ***telnet <ip_address> <port_number>***. Check ports between **1-1023**. When you find an open port, record it below. Look up what service that corresponds to. Record that below.

Note that ports are actually 16-bit numbers so they can extend to 65535. Extend your script to cover all possible ports. If you see ports above 1024 open, what do these likely correspond to? (Don't record all of them.)

After discovering open ports, we may seek to take the next step and connect with the host using the open port. To mitigate this vulnerability, the network administrator should disable any ports that aren't required.

Question 3 (10 points): SNIFFING NETWORK TRAFFIC (Wireshark)

Open up Wireshark through the SSH terminal. (We recommend launching it in the background by using the ***&*** at the end of the command.) To have the window launch on your computer requires **X** windows forwarding. See instructions at https://mobaxterm.mobatek.net/ if you receive an error about having no X11 server. Once Wireshark is open, start collecting all traffic.

ping server 172.16.50.1, then analyze the captured traffic. Did you notice the ARP packets?

Why does the PC send out a broadcast ARP prior to sending the first ping request? What MAC addresses did you notice? screenshot your VM's mac address and the mac address for the host (172.16.50.1)?

We noticed the ARP packets and screenshotted them. The PC sends out a broadcast ARP prior to the first ping request in order to determine the mac address of the target server.

You may need to clear your ARP cache in case you want to repeat this step. You can use **arp –n and arp –d commands** to display the arp cache and remove an entry in the cache**.**

There are filters for Wireshark that make understanding traffic much easier: https://wiki.wireshark.org/DisplayFilters. You may find it convenient to filter using the ***host <ip_address>*** filter which requires that host to be involved in the communication.

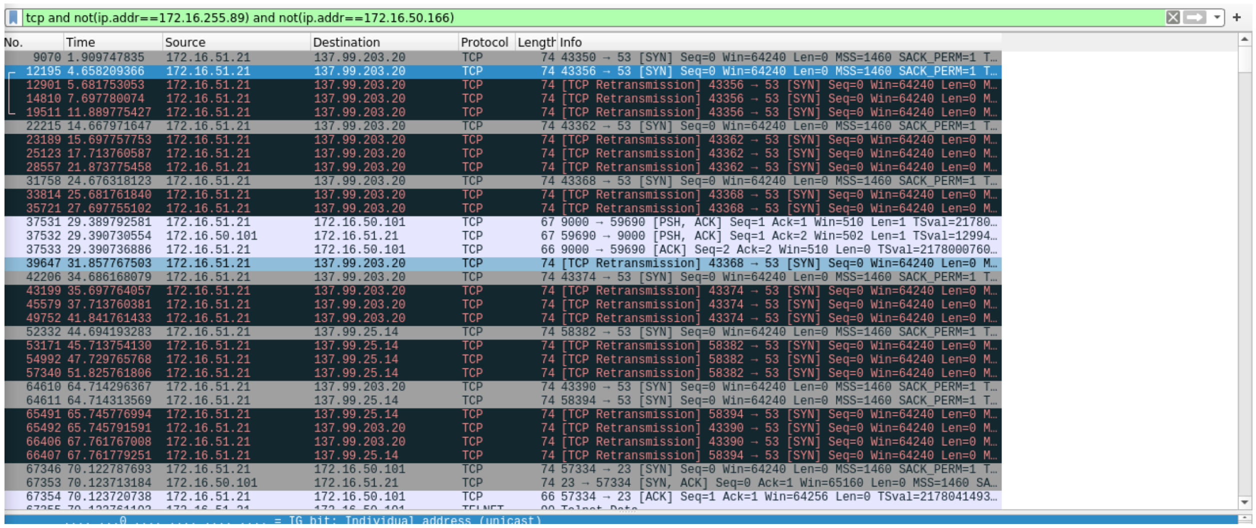
```
Frame 16993: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0
  Interface id: 0 (eth0)
  Encapsulation type: Ethernet (1)
  Arrival Time: Feb 21, 2022 11:10:44.815740594 EST
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1645459844.815740594 seconds
  [Time delta from previous captured frame: 0.001136416 seconds]
  [Time delta from previous displayed frame: 0.059209198 seconds]
  [Time since reference or first frame: 29.198855495 seconds]
  Frame Number: 16993
  Frame Length: 42 bytes (336 bits)
  Capture Length: 42 bytes (336 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:arp]
  [Coloring Rule Name: ARP]
  [Coloring Rule String: arp]
Ethernet II, Src: 52:30:73:e9:9d:a1 (52:30:73:e9:9d:a1), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  Source: 52:30:73:e9:9d:a1 (52:30:73:e9:9d:a1)
  Type: ARP (0x0806)
Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: 52:30:73:e9:9d:a1 (52:30:73:e9:9d:a1)
  Sender IP address: 172.16.51.21
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 172.16.50.1
```

```
Frame 17000: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface eth0, id 0
  Interface id: 0 (eth0)
  Encapsulation type: Ethernet (1)
  Arrival Time: Feb 21, 2022 11:10:44.816916825 EST
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1645459844.816916825 seconds
  [Time delta from previous captured frame: 0.000298805 seconds]
  [Time delta from previous displayed frame: 0.001176231 seconds]
  [Time since reference or first frame: 29.200031726 seconds]
  Frame Number: 17000
  Frame Length: 56 bytes (448 bits)
  Capture Length: 56 bytes (448 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:arp]
  [Coloring Rule Name: ARP]
  [Coloring Rule String: arp]
Ethernet II, Src: 02:95:3e:36:f3:b2 (02:95:3e:36:f3:b2), Dst: 52:30:73:e9:9d:a1 (52:30:73:e9:9d:a1)
  Destination: 52:30:73:e9:9d:a1 (52:30:73:e9:9d:a1)
    Address: 52:30:73:e9:9d:a1 (52:30:73:e9:9d:a1)
    .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Source: 02:95:3e:36:f3:b2 (02:95:3e:36:f3:b2)
    Address: 02:95:3e:36:f3:b2 (02:95:3e:36:f3:b2)
    .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Type: ARP (0x0806)
  Trailer: 0000000000000000000000000000
Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: 02:95:3e:36:f3:b2 (02:95:3e:36:f3:b2)
  Sender IP address: 172.16.50.1
  Target MAC address: 52:30:73:e9:9d:a1 (52:30:73:e9:9d:a1)
  Target IP address: 172.16.51.21
```

Question 4 (15 points): Now let's analyze the traffic captured using Wireshark. There is a web server running on **Server A** (the one you SSH to). "For remote students, [Configure the SSH tunneling to be able to open the web page on your machine *(Check instructions on huskyCT)"]. there is a continuous server talking to your server A, namely server B (VM_B). Server B is running an ftp server and has telnet protocol enabled. Reports the IP addresses for serverA, and serverB. Start running Wireshark then.*

Address of Server A: 172.16.51.21

Address of Server B: 172.16.50.101

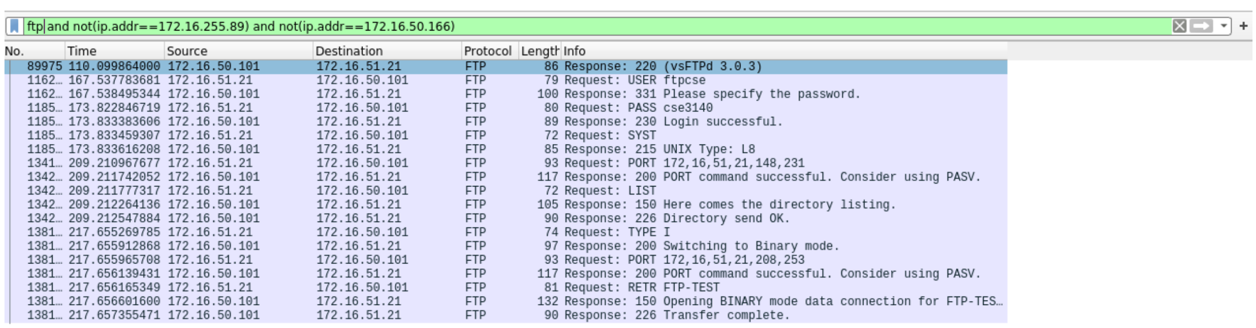


tcp and not(ip.addr==172.16.255.89) and not(ip.addr==172.16.50.166)

No.	Time	Source	Destination	Protocol	Length	Info
9070	1.909747835	172.16.51.21	137.99.203.20	TCP	74	43350 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
12195	4.658209366	172.16.51.21	137.99.203.20	TCP	74	43356 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
12901	5.681753053	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43356 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
14810	7.697780074	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43356 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
19511	11.889775427	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43356 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
22215	14.667971647	172.16.51.21	137.99.203.20	TCP	74	43362 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
23189	15.697757753	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43362 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
25123	17.713760587	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43362 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
28557	21.873775458	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43362 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
31758	24.676318123	172.16.51.21	137.99.203.20	TCP	74	43368 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
33814	25.681761840	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43368 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
35721	27.697755102	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43368 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
37531	29.389792581	172.16.51.21	172.16.50.101	TCP	67	9000 → 59690 [PSH, ACK] Seq=1 Ack=1 Win=510 Len=1 TSval=21780…
37532	29.390730554	172.16.50.101	172.16.51.21	TCP	67	59690 → 9000 [PSH, ACK] Seq=1 Ack=2 Win=502 Len=1 TSval=12994…
37533	29.390736886	172.16.51.21	172.16.50.101	TCP	66	9000 → 59690 [ACK] Seq=2 Ack=2 Win=510 Len=0 TSval=2178000760…
39647	31.857767503	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43368 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
42206	34.686168079	172.16.51.21	137.99.203.20	TCP	74	43374 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
43199	35.697764057	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43374 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
45579	37.713760381	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43374 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
49752	41.841761433	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43374 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
52332	44.694193283	172.16.51.21	137.99.25.14	TCP	74	58382 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
53171	45.713754130	172.16.51.21	137.99.25.14	TCP	74	[TCP Retransmission] 58382 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
54992	47.729765768	172.16.51.21	137.99.25.14	TCP	74	[TCP Retransmission] 58382 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
57340	51.825761806	172.16.51.21	137.99.25.14	TCP	74	[TCP Retransmission] 58382 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
64610	64.714296367	172.16.51.21	137.99.203.20	TCP	74	43390 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
64611	64.714313569	172.16.51.21	137.99.25.14	TCP	74	58394 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
65491	65.745776994	172.16.51.21	137.99.25.14	TCP	74	[TCP Retransmission] 58394 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
65492	65.745791591	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43390 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
66406	67.761767008	172.16.51.21	137.99.203.20	TCP	74	[TCP Retransmission] 43390 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
66407	67.761779251	172.16.51.21	137.99.25.14	TCP	74	[TCP Retransmission] 58394 → 53 [SYN] Seq=0 Win=64240 Len=0 M…
67346	70.122787693	172.16.51.21	172.16.50.101	TCP	74	57334 → 23 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
67353	70.123713184	172.16.50.101	172.16.51.21	TCP	74	23 → 57334 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SA…
67354	70.123720738	172.16.51.21	172.16.50.101	TCP	66	57334 → 23 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2178041493…

- *open the web browser, and type the IP address of server A to see the web service home page*
- run **ftp <serverB>**, [username:**ftpcse,** password**:cse3140]**, download file named **FTP-TEST.**
- run **telnet <serverB>**, [username:**cse,** password**:cse3140]**, then close the connection
- run **ssh <serverB>**, then close the connection

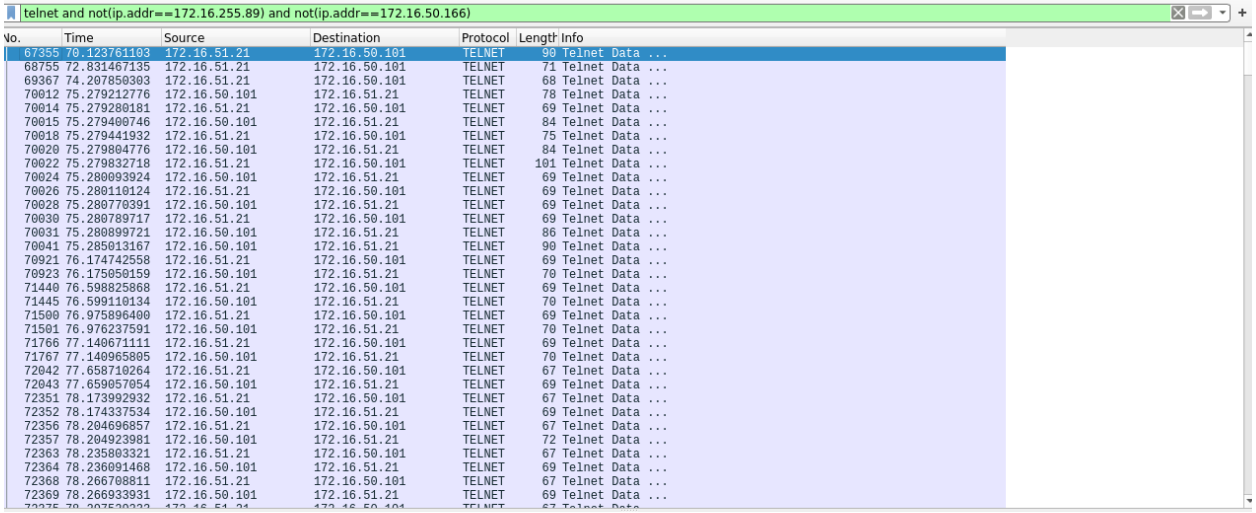
FTP:



ftp and not(ip.addr==172.16.255.89) and not(ip.addr==172.16.50.166)

No.	Time	Source	Destination	Protocol	Length	Info
89975	110.099864000	172.16.50.101	172.16.51.21	FTP	86	Response: 220 (vsFTPd 3.0.3)
1162…	167.537783681	172.16.51.21	172.16.50.101	FTP	79	Request: USER ftpcse
1162…	167.538495344	172.16.50.101	172.16.51.21	FTP	100	Response: 331 Please specify the password.
1185…	173.822846719	172.16.51.21	172.16.50.101	FTP	80	Request: PASS cse3140
1185…	173.833383606	172.16.50.101	172.16.51.21	FTP	89	Response: 230 Login successful.
1185…	173.833459307	172.16.51.21	172.16.50.101	FTP	72	Request: SYST
1185…	173.833616208	172.16.50.101	172.16.51.21	FTP	85	Response: 215 UNIX Type: L8
1341…	209.210967677	172.16.51.21	172.16.50.101	FTP	93	Request: PORT 172,16,51,21,148,231
1342…	209.211742052	172.16.50.101	172.16.51.21	FTP	117	Response: 200 PORT command successful. Consider using PASV.
1342…	209.211777317	172.16.51.21	172.16.50.101	FTP	72	Request: LIST
1342…	209.212264136	172.16.50.101	172.16.51.21	FTP	105	Response: 150 Here comes the directory listing.
1342…	209.212547884	172.16.50.101	172.16.51.21	FTP	90	Response: 226 Directory send OK.
1381…	217.655269785	172.16.51.21	172.16.50.101	FTP	74	Request: TYPE I
1381…	217.655912868	172.16.50.101	172.16.51.21	FTP	97	Response: 200 Switching to Binary mode.
1381…	217.655965708	172.16.51.21	172.16.50.101	FTP	93	Request: PORT 172,16,51,21,208,253
1381…	217.656139431	172.16.50.101	172.16.51.21	FTP	117	Response: 200 PORT command successful. Consider using PASV.
1381…	217.656165349	172.16.51.21	172.16.50.101	FTP	81	Request: RETR FTP-TEST
1381…	217.656601600	172.16.50.101	172.16.51.21	FTP	132	Response: 150 Opening BINARY mode data connection for FTP-TES…
1381…	217.657355471	172.16.50.101	172.16.51.21	FTP	90	Response: 226 Transfer complete.

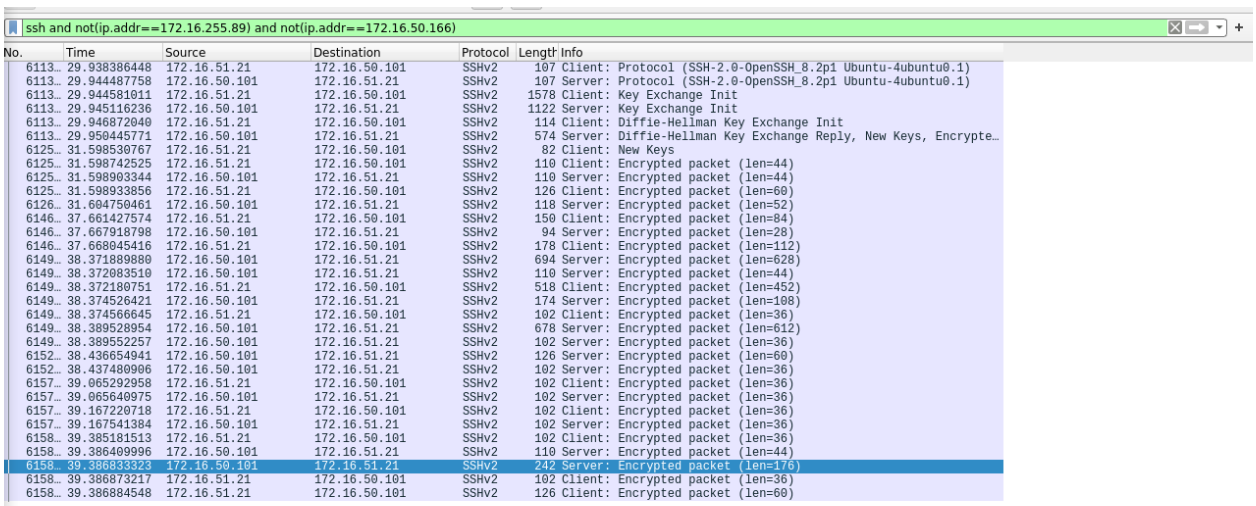
Telnet:



telnet and not(ip.addr==172.16.255.89) and not(ip.addr==172.16.50.166)

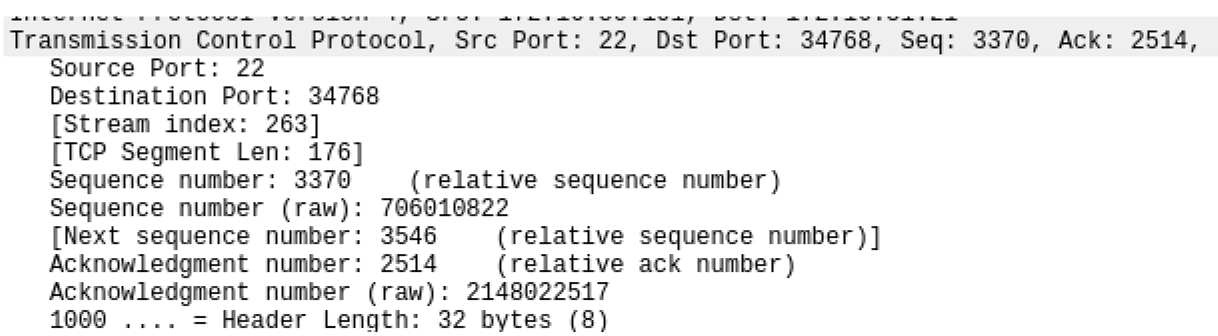
No.	Time	Source	Destination	Protocol	Length	Info
67355	70.123761103	172.16.51.21	172.16.50.101	TELNET	90	Telnet Data ...
68755	72.831467135	172.16.51.21	172.16.50.101	TELNET	71	Telnet Data ...
69367	74.207850303	172.16.51.21	172.16.50.101	TELNET	68	Telnet Data ...
70012	75.279212776	172.16.50.101	172.16.51.21	TELNET	78	Telnet Data ...
70014	75.279280181	172.16.51.21	172.16.50.101	TELNET	69	Telnet Data ...
70015	75.279400746	172.16.50.101	172.16.51.21	TELNET	84	Telnet Data ...
70018	75.279441932	172.16.51.21	172.16.50.101	TELNET	75	Telnet Data ...
70020	75.279804776	172.16.50.101	172.16.51.21	TELNET	84	Telnet Data ...
70022	75.279832718	172.16.51.21	172.16.50.101	TELNET	101	Telnet Data ...
70024	75.280093924	172.16.50.101	172.16.51.21	TELNET	69	Telnet Data ...
70026	75.280110124	172.16.51.21	172.16.50.101	TELNET	69	Telnet Data ...
70028	75.280770391	172.16.50.101	172.16.51.21	TELNET	69	Telnet Data ...
70030	75.280789717	172.16.51.21	172.16.50.101	TELNET	69	Telnet Data ...
70031	75.280899721	172.16.50.101	172.16.51.21	TELNET	86	Telnet Data ...
70041	75.285013167	172.16.50.101	172.16.51.21	TELNET	90	Telnet Data ...
70921	76.174742558	172.16.51.21	172.16.50.101	TELNET	69	Telnet Data ...
70923	76.175050159	172.16.50.101	172.16.51.21	TELNET	70	Telnet Data ...
71440	76.598825868	172.16.51.21	172.16.50.101	TELNET	69	Telnet Data ...
71445	76.599110134	172.16.50.101	172.16.51.21	TELNET	70	Telnet Data ...
71500	76.975896400	172.16.51.21	172.16.50.101	TELNET	69	Telnet Data ...
71501	76.976237591	172.16.50.101	172.16.51.21	TELNET	70	Telnet Data ...
71766	77.140671111	172.16.51.21	172.16.50.101	TELNET	69	Telnet Data ...
71767	77.140965805	172.16.50.101	172.16.51.21	TELNET	70	Telnet Data ...
72042	77.658710264	172.16.51.21	172.16.50.101	TELNET	67	Telnet Data ...
72043	77.659057054	172.16.50.101	172.16.51.21	TELNET	69	Telnet Data ...
72351	78.173992932	172.16.51.21	172.16.50.101	TELNET	67	Telnet Data ...
72352	78.174337534	172.16.50.101	172.16.51.21	TELNET	69	Telnet Data ...
72356	78.204696857	172.16.51.21	172.16.50.101	TELNET	67	Telnet Data ...
72357	78.204923981	172.16.50.101	172.16.51.21	TELNET	72	Telnet Data ...
72363	78.235803321	172.16.51.21	172.16.50.101	TELNET	67	Telnet Data ...
72364	78.236091468	172.16.50.101	172.16.51.21	TELNET	69	Telnet Data ...
72368	78.266708811	172.16.51.21	172.16.50.101	TELNET	67	Telnet Data ...
72369	78.266933931	172.16.50.101	172.16.51.21	TELNET	69	Telnet Data ...

SSH:



ssh and not(ip.addr==172.16.255.89) and not(ip.addr==172.16.50.166)

No.	Time	Source	Destination	Protocol	Length	Info
6113…	29.938386448	172.16.51.21	172.16.50.101	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.1)
6113…	29.944487758	172.16.50.101	172.16.51.21	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.1)
6113…	29.944581011	172.16.51.21	172.16.50.101	SSHv2	1578	Client: Key Exchange Init
6113…	29.945116236	172.16.50.101	172.16.51.21	SSHv2	1122	Server: Key Exchange Init
6113…	29.946872040	172.16.51.21	172.16.50.101	SSHv2	114	Client: Diffie-Hellman Key Exchange Init
6113…	29.950445771	172.16.50.101	172.16.51.21	SSHv2	574	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypte…
6125…	31.598530767	172.16.51.21	172.16.50.101	SSHv2	82	Client: New Keys
6125…	31.598742525	172.16.51.21	172.16.50.101	SSHv2	110	Client: Encrypted packet (len=44)
6125…	31.598903344	172.16.50.101	172.16.51.21	SSHv2	110	Server: Encrypted packet (len=44)
6125…	31.598933856	172.16.51.21	172.16.50.101	SSHv2	126	Client: Encrypted packet (len=60)
6126…	31.604750461	172.16.50.101	172.16.51.21	SSHv2	118	Server: Encrypted packet (len=52)
6146…	37.661427574	172.16.51.21	172.16.50.101	SSHv2	150	Client: Encrypted packet (len=84)
6146…	37.667918798	172.16.50.101	172.16.51.21	SSHv2	94	Server: Encrypted packet (len=28)
6146…	37.668045416	172.16.51.21	172.16.50.101	SSHv2	178	Client: Encrypted packet (len=112)
6149…	38.371889880	172.16.50.101	172.16.51.21	SSHv2	694	Server: Encrypted packet (len=628)
6149…	38.372083510	172.16.50.101	172.16.51.21	SSHv2	110	Server: Encrypted packet (len=44)
6149…	38.372180751	172.16.51.21	172.16.50.101	SSHv2	518	Client: Encrypted packet (len=452)
6149…	38.374526421	172.16.50.101	172.16.51.21	SSHv2	174	Server: Encrypted packet (len=108)
6149…	38.374566645	172.16.51.21	172.16.50.101	SSHv2	102	Client: Encrypted packet (len=36)
6149…	38.389528954	172.16.50.101	172.16.51.21	SSHv2	678	Server: Encrypted packet (len=612)
6149…	38.389552257	172.16.50.101	172.16.51.21	SSHv2	102	Server: Encrypted packet (len=36)
6152…	38.436654941	172.16.50.101	172.16.51.21	SSHv2	126	Server: Encrypted packet (len=60)
6152…	38.437480906	172.16.50.101	172.16.51.21	SSHv2	102	Server: Encrypted packet (len=36)
6157…	39.065292958	172.16.51.21	172.16.50.101	SSHv2	102	Client: Encrypted packet (len=36)
6157…	39.065640975	172.16.50.101	172.16.51.21	SSHv2	102	Server: Encrypted packet (len=36)
6157…	39.167220718	172.16.51.21	172.16.50.101	SSHv2	102	Client: Encrypted packet (len=36)
6157…	39.167541384	172.16.50.101	172.16.51.21	SSHv2	102	Server: Encrypted packet (len=36)
6158…	39.385181513	172.16.51.21	172.16.50.101	SSHv2	102	Client: Encrypted packet (len=36)
6158…	39.386409996	172.16.50.101	172.16.51.21	SSHv2	110	Server: Encrypted packet (len=44)
6158…	39.386833323	172.16.50.101	172.16.51.21	SSHv2	242	Server: Encrypted packet (len=176)
6158…	39.386873217	172.16.51.21	172.16.50.101	SSHv2	102	Client: Encrypted packet (len=36)
6158…	39.386884548	172.16.51.21	172.16.50.101	SSHv2	126	Client: Encrypted packet (len=60)



```
Transmission Control Protocol, Src Port: 22, Dst Port: 34768, Seq: 3370, Ack: 2514, L
    Source Port: 22
    Destination Port: 34768
    [Stream index: 263]
    [TCP Segment Len: 176]
    Sequence number: 3370    (relative sequence number)
    Sequence number (raw): 706010822
    [Next sequence number: 3546    (relative sequence number)]
    Acknowledgment number: 2514    (relative ack number)
    Acknowledgment number (raw): 2148022517
    1000 .... = Header Length: 32 bytes (8)
```

At the completion of this run, list the following:

1. all port numbers observed
   FTP: Port 21

Telnet: Port 23
SSH: Port 22

2. all IP addresses observed
   Our IP Address 172.16.51.21 and Server B 172.16.50.101

Create a display filter to show only web traffic involving ***VM-serverA*** (the one you are using for the SSH). What is the display filter? What type of traffic is present from that machine?

A display filter only shows the ip addresses within a given range. This filter excludes the origin machine and specifically looks for packets of a specific protocol to sniff.

Examine the traffic of a Telnet session to **VM-serverB**. You can telnet to any machine using the command **telnet <host_IP>,** apply a filter that only displays Telnet-related traffic. *Right-click one of the Telnet lines in the Packet list section of Wireshark, and from the drop-down list, select Follow TCP Stream.* The Follow TCP Stream window displays the data for your Telnet session with the VM. ***What do you notice?***

You can see everything you type immediately pop up in the wireshark terminal. It's almost scary. The text is stored as plain text.

Let's now examine an SSH connection. Type **ssh <VM-serverB>**, to get a connection. Filter all SSH protocol connections with the remote serverC, ***what is the display filter?*** Right-click one of the SSHv2 lines in the Packet list section of Wireshark, and in the drop-down list, select the Follow TCP Stream option. Examine the Follow TCP Stream window of your SSH session. What do you notice, which do you prefer for a remote connection (SSH or TELNET)? Why?

63	0.004194606	172.16.51.21	172.16.255.89	SSH	150	Server: Encrypted packet (len=96)
64	0.004308768	172.16.51.21	172.16.255.89	SSH	150	Server: Encrypted packet (len=96)
65	0.004373909	172.16.51.21	172.16.255.89	SSH	150	Server: Encrypted packet (len=96)
66	0.004390520	172.16.255.89	172.16.51.21	TCP	56	63601 → 22 [ACK] Seq=193 Ack=3265 Win=6142 Len=0
67	0.004424934	172.16.255.89	172.16.51.21	TCP	56	63601 → 22 [ACK] Seq=193 Ack=3457 Win=6147 Len=0

You prefer SSH above telnet because SSH is more secure. The telnet sent the password in plain text, which was visible to the wireshark terminal. On the other hand, SSH encrypts the information before sending it, making it unreadable to the sniffing machine. This secure shell is typically preferable to telnet.

List all the hosts that are involved in an SSH connection that you can capture using Wireshark, Isolate one source and destination involved in SSH. Filter just those packets. What information can you confidently learn about the two computers involved in the connection? What can you not decipher? What is your best guess as to why you can't decipher everything?

```
▸ Frame 1: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface eth0, id 0
▸ Ethernet II, Src: Fortinet_fc:36:ba (04:d5:90:fc:36:ba), Dst: 52:30:73:e9:9d:a1 (52:30:73:e9:9d:a1)
▸ Internet Protocol Version 4, Src: 172.16.255.89, Dst: 172.16.51.21
▸ Transmission Control Protocol, Src Port: 63601, Dst Port: 22, Seq: 1, Ack: 1, Len: 96
▾ SSH Protocol
    Packet Length (encrypted): 636deba5
    Encrypted Packet: 1f65b53290b8c4d3fcaded95ca11cf393d6dee5dd12f3d21…
    [Direction: client-to-server]

0000  52 30 73 e9 9d a1 04 d5  90 fc 36 ba 08 00 45 00   R0s·····  ··6···E·
0010  00 88 57 da 40 00 7f 06  19 06 ac 10 ff 59 ac 10   ··W·@···  ·····Y··
0020  33 15 f8 71 00 16 26 29  53 ef 0e 19 34 87 50 18   3··q··&)  S···4·P·
0030  18 00 4d b7 00 00 63 6d  eb a5 1f 65 b5 32 90 b8   ··M···cm  ···e·2··
0040  c4 d3 fc ad ed 95 ca 11  cf 39 3d 6d ee 5d d1 2f   ········  ·9=m·]·/
0050  3d 21 de cb 7d a2 9d d6  58 ff 10 e9 ce ee 52 dd   =!··}···  X·····R·
0060  e5 a2 84 f0 cb 65 bd 3b  5f 3a d3 89 76 f0 94 ca   ·····e·;  _:··v···
0070  1f d8 20 a7 e1 87 5b 7c  3f fb 76 41 b3 15 9d ec   ·· ···[|  ?·vA····
0080  92 ec 73 06 94 48 0c 95  70 dc 1c 53 62 84 cc 86   ··s··H··  p··Sb···
0090  4a 04 17 a4 d1 dd                                  J·····
```

With the SSH Sniffing, you could find the source and destination involved in the SSH, and also know the size of the packet being sent and its encrypted value. You do not, on the other hand, get the keys it was encrypted with or the ability to decrypt it, meaning you cannot decipher the actual text of the commands or messages. The lengths also seem to all be 96, meaning even that gives you little information about what the computers are talking about. You simply know they are communicating.

Question 5 (10 points): Poison ARP cache

For now, you are probably wondering how these packets are sent over the internet. How does the TCP/IP protocol stack manage to know the location of the destination addresses? There are two types of addresses that are needed for any packet transfer; namely, physical address (MAC address) and logical address(IP address). So far, using the “ifconfig” command, we learned how to obtain the src IP and src MAC addresses. What about the destination MAC and the destination IP address for the VM I am willing to talk to?

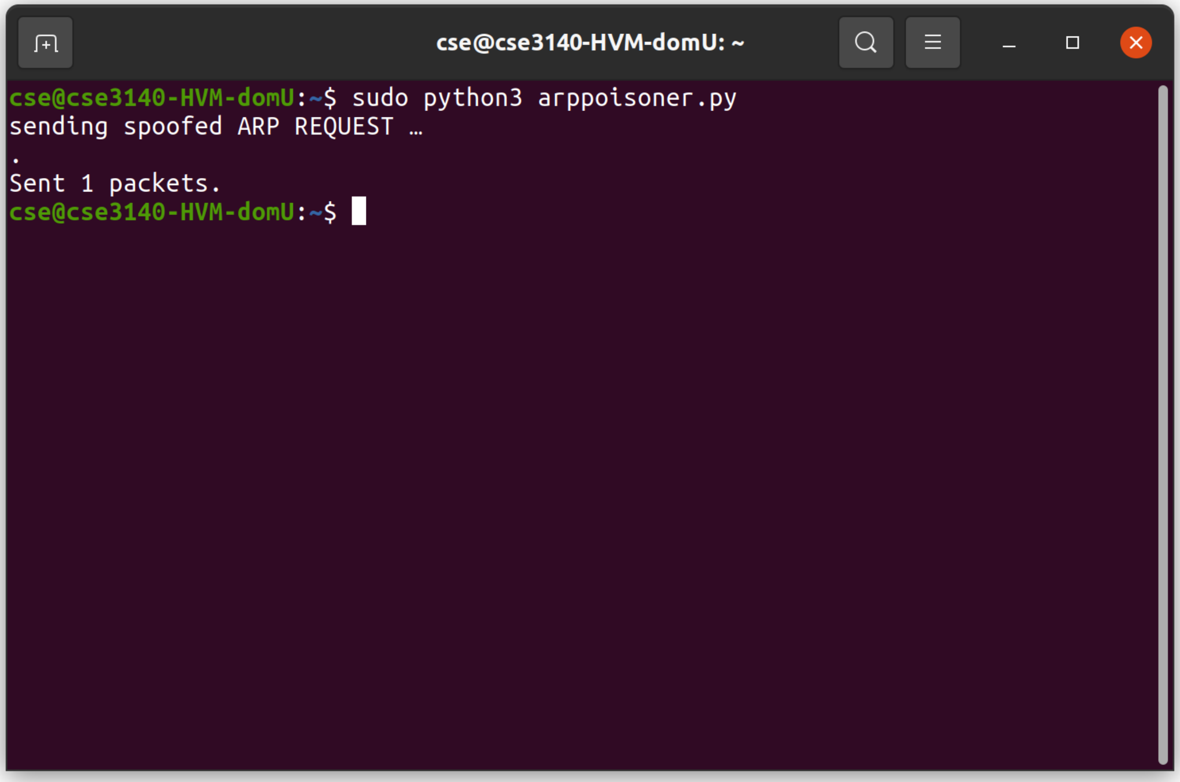
For any communication, the destination IP address can be obtained using an addressing service namely, the [Domain Name System (DNS)](). DNS is the hierarchical and decentralized naming system used to identify computers, services, and other resources reachable through the Internet or other Internet Protocol networks. For the physical address, [ARP]() protocol is used to obtain the physical address from a specific IP address. The IP addressing helps in end-to-end machine delivery. while the physical address decided the next step to take on the local network. In this question we will learn a famous attack that is using ARP messages, namely, “ARP cache poison attack”. The purpose of that attack is to poison the victims arp cache by injecting fake information about the physical address. For example, if the machine's B mac address is X, but you were able to update the victims arp cache to indicate that machine’s B mac address is Y. You can redirect the next step for the victim to go to Y instead of going to X.

ARP is a very simple lower layer protocol that does not have any protection like encryption or integrity checking. The ARP is also stateless, when it sends a request out if it forgets about that request, if it gets the reply it automatically updates the cache. Accordingly, the ARP cache information can easily get poisoned if an attacker manages to trick the victim's machine by updating it's ARP cache. **You need to update the victim's arp cache.** You can either send a spoofed ARP request or a spoofed ARP reply, even if a request was not triggered by the request.

Now let's consider **VM-serverA** and **VM-serverB**. We want to poison the cash of the target computer **VM-serverA**. Remember you have a sudo privilege only on **VM-serverB**. You will create an entry in **VM-serverA** that maps the mac address of Instructor's VM (192.16.51.100) to your **VM-serverB** mac address. You can display the arp cache entries using the command "arp -n". You will pretend to be the Instructor's VM with the IP 192.16.50.100, and you will send a unicast fake request to the victim machine **VM-serverA**. When **VM-serverA** receives this request it will update the arp cache to match the fake mac address. you can use the following code:

```
cse@cse3140-HVM-domU: ~
  GNU nano 4.8                          arppoisoner.py
from scapy.all import *
VM_target_IP = "172.16.51.25"
VM_target_MAC = "1a:cc:14:15:85:01"
VICTIM_IP = "172.16.50.100"
FAKE_MAC = "0a:36:b3:3c:ef:01"
print("sending spoofed ARP REQUEST …")
ether = Ether()
ether.dst = VM_target_MAC
ether.src = FAKE_MAC
arp = ARP()
arp.psrc = VICTIM_IP
arp.hwsrc = FAKE_MAC
arp.pdst = VM_target_IP
arp.op = 1
frame = ether/arp
sendp(frame)

                              [ Read 16 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell  ^_ Go To Line
```



Question 6 (10 points): Now, as you have learned how to send and receive through the provide OS Socket API. You should be wondering. How can I sniff packets that are not intended for me? How is Wireshark able to present such details about the packet headers? A nice feature that an OS provides is the use of **raw sockets**. The provided raw socket allows applications to get the data directly without going through each layer of the TCP/IP stack. If you register yourself as an application to the raw socket, the OS will make a copy of each packet received to the existing raw sockets. ***There is a hidden machine that is continuously pinging each VM in the network.*** You need to write a simple sniffing program on **<server B>** to sniff only ping packets and report the source and destination IP and Ethernet addresses. You will find the following example helpful. **[serverC (username:cse & password:cse3140)]**

cse@cse3140-HVM-domU: ~

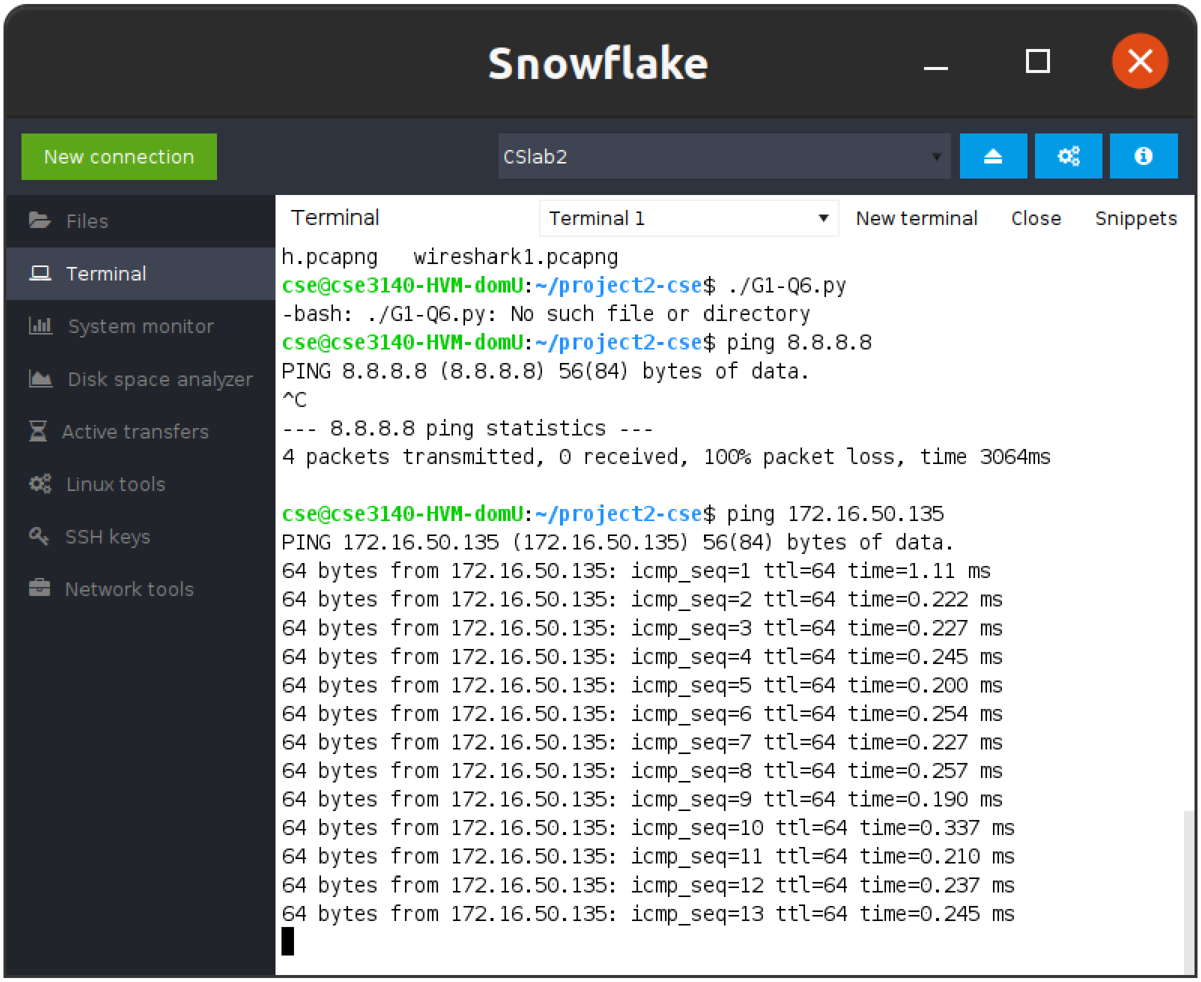
GNU nano 4.8 q6.py

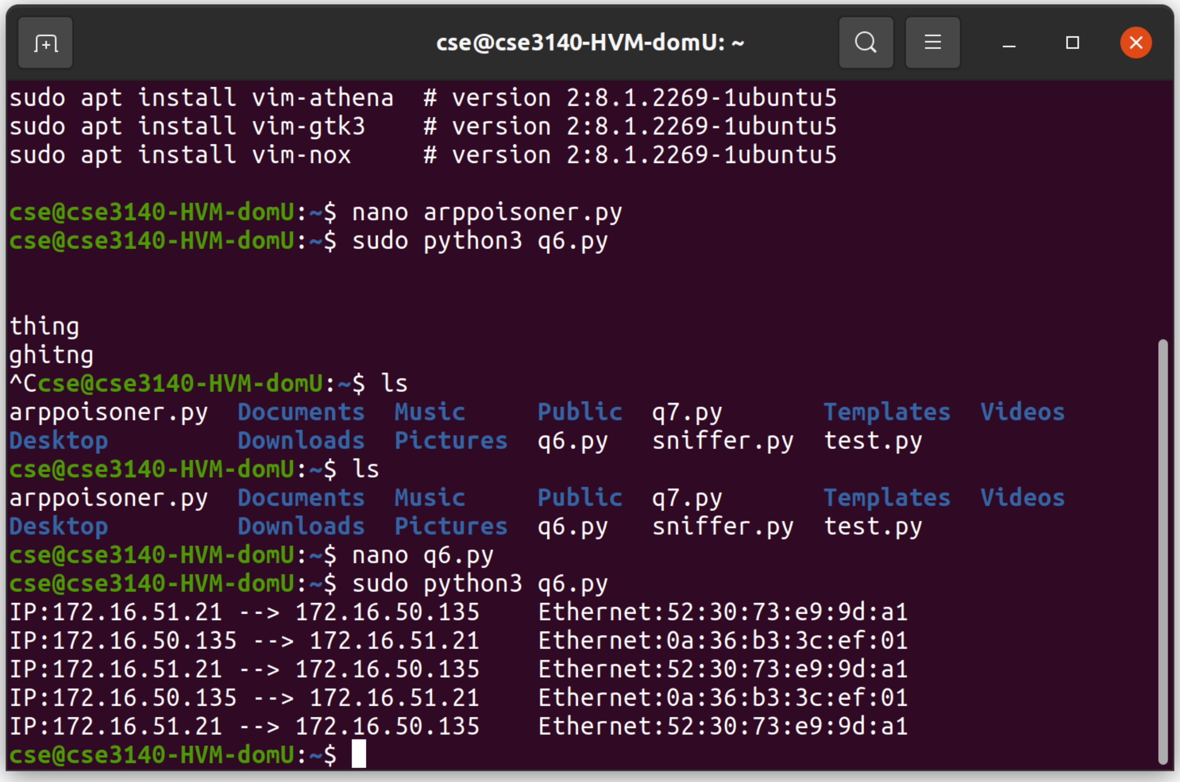
```
from scapy.all import *
def process_packet(pkt):
#  pkt.show()
  if pkt.haslayer(IP):
          ip = pkt[IP]
          print("IP:{} --> {}    Ethernet:{}".format(ip. src, ip.dst, pkt.src))
  elif pkt.haslayer(TCP):
          tcp = pkt[TCP]
          print("TCP ports:{} --> {}".format(tcp.sport, tcp.dport))
  elif pkt.haslayer(UDP):
          udp = pkt[UDP]
          print("UDP ports:{} --> {}".format(udp.sport, udp.dport))
  else:
          print("other protocol")

if __name__ == "__main__":
        sniff(iface="eth0", filter = "icmp", prn = process_packet, count =5)
```

[Read 17 lines]

```
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line
```


Snowflake
New connection
CSlab2
Files
Terminal
System monitor
Disk space analyzer
Active transfers
Linux tools
SSH keys
Network tools
Terminal
Terminal 1
New terminal
Close
Snippets
h.pcapng   wireshark1.pcapng
cse@cse3140-HVM-domU:~/project2-cse$ ./G1-Q6.py
-bash: ./G1-Q6.py: No such file or directory
cse@cse3140-HVM-domU:~/project2-cse$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3064ms
cse@cse3140-HVM-domU:~/project2-cse$ ping 172.16.50.135
PING 172.16.50.135 (172.16.50.135) 56(84) bytes of data.
64 bytes from 172.16.50.135: icmp_seq=1 ttl=64 time=1.11 ms
64 bytes from 172.16.50.135: icmp_seq=2 ttl=64 time=0.222 ms
64 bytes from 172.16.50.135: icmp_seq=3 ttl=64 time=0.227 ms
64 bytes from 172.16.50.135: icmp_seq=4 ttl=64 time=0.245 ms
64 bytes from 172.16.50.135: icmp_seq=5 ttl=64 time=0.200 ms
64 bytes from 172.16.50.135: icmp_seq=6 ttl=64 time=0.254 ms
64 bytes from 172.16.50.135: icmp_seq=7 ttl=64 time=0.227 ms
64 bytes from 172.16.50.135: icmp_seq=8 ttl=64 time=0.257 ms
64 bytes from 172.16.50.135: icmp_seq=9 ttl=64 time=0.190 ms
64 bytes from 172.16.50.135: icmp_seq=10 ttl=64 time=0.337 ms
64 bytes from 172.16.50.135: icmp_seq=11 ttl=64 time=0.210 ms
64 bytes from 172.16.50.135: icmp_seq=12 ttl=64 time=0.237 ms
64 bytes from 172.16.50.135: icmp_seq=13 ttl=64 time=0.245 ms



Question 7 (12 points): TCP is the transport protocol used for applications that require reliability. TCP connections are initiated using a three-way handshake: **SYN**, **SYN-ACK**, and **ACK**. Before a client attempts to connect with a server, the server must first bind its application to a continuous listening port that is ready to receive connections. A client usually starts the connection establishment by sending a SYN packet. During the data exchange all packets sent from one side are ACKed by the other side to ensure reliable transmission. A connection termination phase uses either three-way or four-way handshake. ***What is the difference between three-way or four-way handshake used for terminating the connection?***

A Three way handshake starts with a SYN packet, which lets the receiving server know it is trying to "Synchronize". This opens a 75 second window for the server to receive this packet and send back its own packet, a SYN-ACK packet. This packet "acknowledges" receipt of the SYN packet, and sends an SYN packet on its own to initiate the connection. Once the original machine receives the SYN-ACK packet, it returns an ACK to let the other machine know its SYN was received and open the channel of communication. In this way, it can be thought of as 4 packets of information moving (2 SYN's, and 2 ACK's, but the 2nd command was a combined packet) between two machines.

Similarly, the close command is 4 packets of information being sent between two machines. The difference with this, however, is that it can be either a 3 or 4 way handshake depending on the specifics of the machine and intended use case. For most use cases, one machine will send an FIN packet, and the other machine will send both an ACK of receipt for this packet, and a FIN packet of

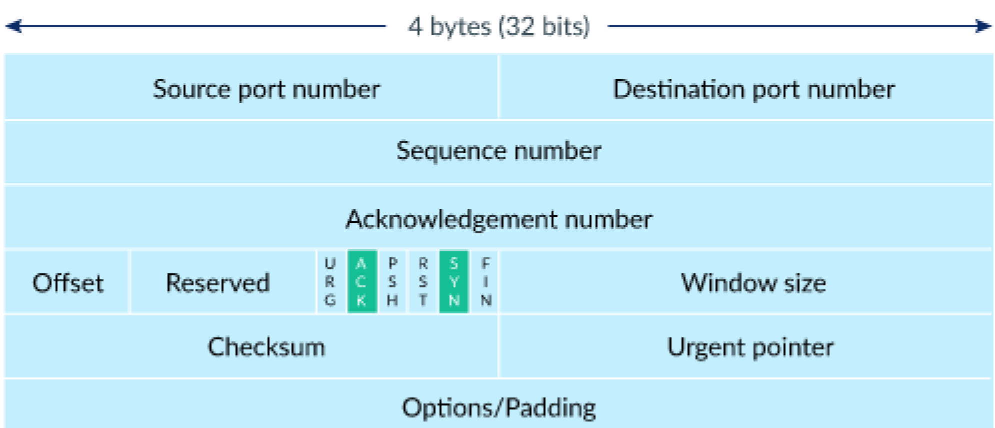
their own. These packets could be sent together(FIN-ACK), as in the initialization of connection protocol, but often are not because the second machine does not actually have to send an FIN on the network. Because of this, concatenating them together is less often done. This means instead the second machine sends two separate packets, an FIN, and an ACK, or just an ACK. If the machine sends both an FIN and an ACK, the first machine knows it received its FIN, and upon Receipt of an FIN will send back an ACK. Once the 4 way handshake is complete FIN-ACK-FIN-ACK, then the session is terminated.

Try to find a three way handshake for connection establishment and another three way handshake connection termination in your captured TCP. You can use the FTP connection from the previous question. ***Which layer does this TCP belong to in the TCP/IP model?***

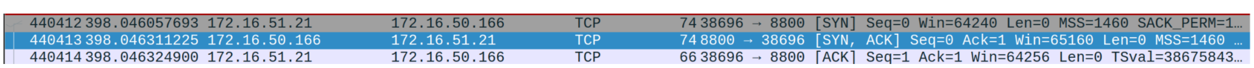
TCP belongs to the transport layer of the TCP/IP model.

What actual data is sent in these three packets (ignoring TCP headers)?

Syn and Ack are bits which can be represented in this diagram here. We can see it shows the source and destination ports, the sequence number, acknowledgement number, and sets the value of the syn and ack bit to 0 or 1 depending on their presence. This is also how a SynAck or FinAck can be sent in the same packet and put together. The actual data, which is stored below these headers, is made up of



For one TCP connection, can you identify TCP 3-way handshake easily?



440412	398.046057693	172.16.51.21	172.16.50.166	TCP	74	38696 → 8800 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1…
440413	398.046311225	172.16.50.166	172.16.51.21	TCP	74	8800 → 38696 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 …
440414	398.046324900	172.16.51.21	172.16.50.166	TCP	66	38696 → 8800 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=38675843…

Yes, if you search specifically for the SYN, ACK packet, you will find that in the middle of the 3 way handshake to establish a connection

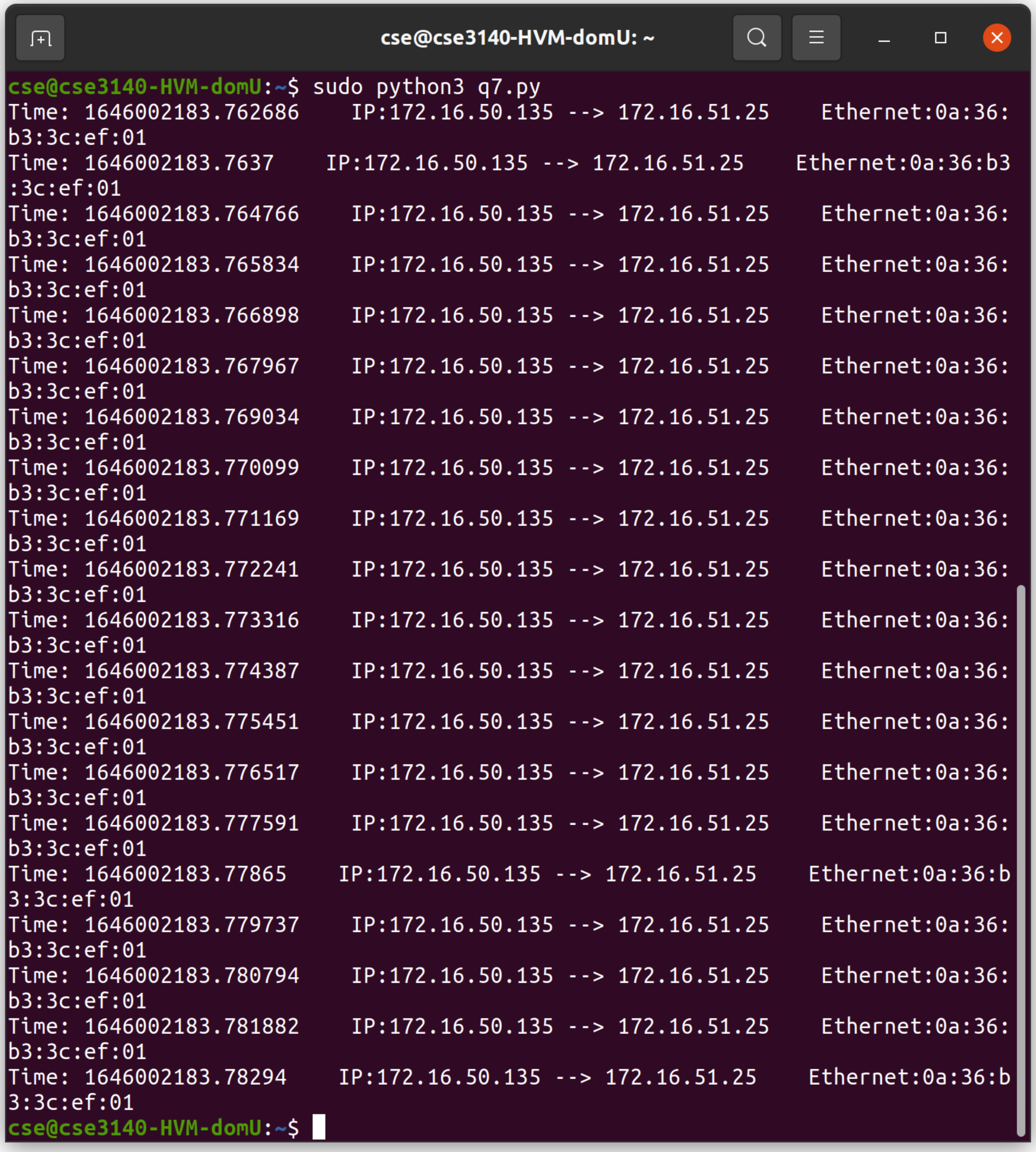
Fill in the following information for the following connection establishment 3-way handshake **(Consider the TCP connection sending packets of length 1566 bytes)**. Why does TCP need to specify a window size (Win)?

The TCP window size needs to be specified so that the receiving machine knows how much data to expect and look for. This allows the receiving machine to not only know that a package was received, but that it was the right packet of the right size with data of the right size.

cse@cse3140-HVM-domU: ~

```
  GNU nano 4.8                            q7.py
from scapy.all import *
def process_packet(pkt):
  if len(pkt) == 1566:
    #pkt.show()
    thistime = pkt.time
    print("Time: {}     ".format(thistime), end='')
    if pkt.haslayer(IP):
      ip = pkt[IP]
      print("IP:{} --> {}    Ethernet:{}".format(ip. src, ip.dst, pkt.src))
    elif pkt.haslayer(TCP):
      tcp = pkt[TCP]
      print("TCP ports:{} --> {}".format(tcp.sport, tcp.dport))
    elif pkt.haslayer(UDP):
      udp = pkt[UDP]
      print("UDP ports:{} --> {}".format(udp.sport, udp.dport))
    else:
      print("other protocol")

if __name__ == "__main__":
                                     [ Read 22 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line
```

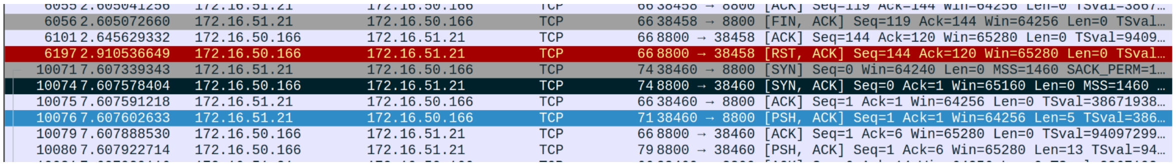


```
cse@cse3140-HVM-domU: ~
cse@cse3140-HVM-domU:~$ sudo python3 q7.py
Time: 1646002183.762686    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.7637    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:b3
:3c:ef:01
Time: 1646002183.764766    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.765834    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.766898    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.767967    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.769034    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.770099    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.771169    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.772241    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.773316    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.774387    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.775451    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.776517    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.777591    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.77865    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:b
3:3c:ef:01
Time: 1646002183.779737    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.780794    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.781882    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:
b3:3c:ef:01
Time: 1646002183.78294    IP:172.16.50.135 --> 172.16.51.25    Ethernet:0a:36:b
3:3c:ef:01
cse@cse3140-HVM-domU:~$
```

```
cse@cse3140-HVM-domU: ~
  GNU nano 4.8                                sniffer.py
from scapy.all import *
def process_packet(pkt):
#  pkt.show()
        if pkt.haslayer(IP):
                ip = pkt[IP]
                print("IP:{} --> {}".format(ip.src, ip.dst))
        elif pkt.haslayer(TCP):
                tcp = pkt[TCP]
                print ("TCP ports:{} --> {}".format(tcp.sport, tcp.dport))
        elif pkt.haslayer(UDP):
                udp = pkt[UDP]
                print ("UDP ports:{} --> {}".format(udp.sport,udp.dport))
        if pkt.haslayer(Ether):
                eth = pkt[Ether]
                print("Ether ports:{} --> {}".format(eth.src, eth.dst))
        else:
                print("other protocol")
packets = sniff(iface = "eth0", filter = "icmp", prn = process_packet, count = 5)
print(packets)
                                     [ Read 19 lines ]
^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit         ^R Read File    ^\ Replace      ^U Paste Text   ^T To Spell     ^_ Go To Line
```

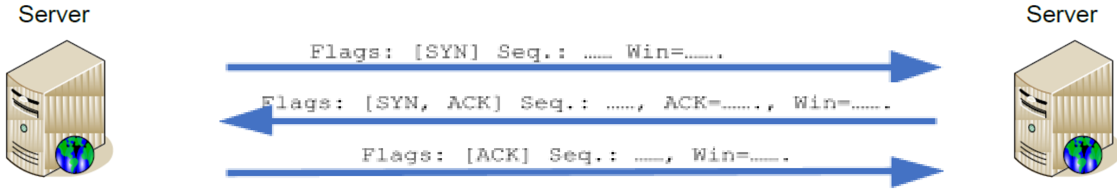
```
cse@cse3140-HVM-domU: ~
cse@cse3140-HVM-domU:~$ ls
arppoisoner.py  Documents  Music     Public  q7.py       Templates  Videos
Desktop         Downloads  Pictures  q6.py   sniffer.py  test.py
cse@cse3140-HVM-domU:~$ sudo python3 sniffer.py
IP:172.16.48.9 --> 172.16.50.135
Ether ports:32:92:ce:f0:76:3d --> 0a:36:b3:3c:ef:01
IP:172.16.50.135 --> 172.16.48.9
Ether ports:0a:36:b3:3c:ef:01 --> 32:92:ce:f0:76:3d
IP:172.16.48.9 --> 172.16.50.135
Ether ports:32:92:ce:f0:76:3d --> 0a:36:b3:3c:ef:01
IP:172.16.50.135 --> 172.16.48.9
Ether ports:0a:36:b3:3c:ef:01 --> 32:92:ce:f0:76:3d
IP:172.16.48.9 --> 172.16.50.135
Ether ports:32:92:ce:f0:76:3d --> 0a:36:b3:3c:ef:01
<Sniffed: TCP:0 UDP:0 ICMP:5 Other:0>
cse@cse3140-HVM-domU:~$
```

You can see that there were sets of 3 or 4, depending on if I was establishing the connection as in the first one, or dropping the connection as in the second one.



6055	2.605041256	172.16.51.21	172.16.50.166	TCP	66 38458 → 8800 [ACK] Seq=119 Ack=144 Win=64256 Len=0 TSval=3867…
6056	2.605072660	172.16.51.21	172.16.50.166	TCP	66 38458 → 8800 [FIN, ACK] Seq=119 Ack=144 Win=64256 Len=0 TSval…
6101	2.645629332	172.16.50.166	172.16.51.21	TCP	66 8800 → 38458 [ACK] Seq=144 Ack=120 Win=65280 Len=0 TSval=9409…
6197	2.910536649	172.16.50.166	172.16.51.21	TCP	66 8800 → 38458 [RST, ACK] Seq=144 Ack=120 Win=65280 Len=0 TSval…
10071	7.607339343	172.16.51.21	172.16.50.166	TCP	74 38460 → 8800 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1…
10074	7.607578404	172.16.50.166	172.16.51.21	TCP	74 8800 → 38460 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 …
10075	7.607591218	172.16.51.21	172.16.50.166	TCP	66 38460 → 8800 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=38671938…
10076	7.607602633	172.16.51.21	172.16.50.166	TCP	71 38460 → 8800 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=5 TSval=386…
10079	7.607888530	172.16.50.166	172.16.51.21	TCP	66 8800 → 38460 [ACK] Seq=1 Ack=6 Win=65280 Len=0 TSval=94097299…
10080	7.607922714	172.16.50.166	172.16.51.21	TCP	79 8800 → 38460 [PSH, ACK] Seq=1 Ack=6 Win=65280 Len=13 TSval=94…

Connection establishment

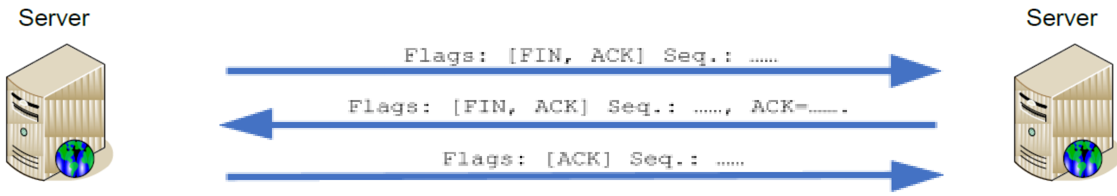


IP address:.50.135
Port: 21

IP address: .51.25
Port:21

For the FTP communication you previously tested, capture the connection termination and record the following.

Connection termination



IP address: …….
Port: …….

IP address: …….
Port: ………

Question 8 (5 points): List all of the protocols you have seen in your capture, and specify the operating TCP/IP layer for these protocols? Based on Internet lookups, which of these protocols try to provide **security/encryption**?

Telnet - (Layer 7)Not secure/Encrypted

FTP - (Layer 7) Provides passwords in plain text, not encrypted

TCP - (Layer 4) Susceptible to SYN style DDOS attacks, but encrypted and more secure

SSH - (layer 6 and 7)A secure shell protocol with encryption

SSHv2 - (layer 6 and 7) A secure shell protocol with encryption and SFTP (secure file transfer protocol)

ICMP - (layer 3) There are secure tunnels to be made with ICMP, but it is not secure by default

Question 9 (15 points): This part requires you to reverse engineer a custom protocol that is running on the network. First you need to find this protocol. The server is listening to port 8800. The protocol runs periodically. **What display filter did you use to isolate the protocol?**

tcp.port == 8800

Your goal for this question is to write a program that can connect to the server and eventually get a response that starts with: “Critical Inf.” There is a client running on the network that is responding properly to the server requests. You need to observe interactions between the running client and server to decide how the client is supposed to respond to each request. There are six stages in the interaction. The last four are randomized so don’t just copy paste from what you see the client on the network doing. Your job is to reverse engineer the protocol state machine that the server is running. This is representable using a simple finite automata: https://en.wikipedia.org/wiki/Finite-state_machine. For this question I encourage you to use the [Python Socket package](#). At some point you may need to change the source port that your client is using. This can be done with the command (for a socket named s before connecting): s.bind(('', <port_num>)).

We have provided a simple client server example in your home directory (echoServer.py and helloClient.py) in your project2 folder. At some point during this question you’ll need to compute a cryptographic hash. You can do this using the hashlib package (import hashlib) and the following code:

```
m = hashlib.sha256()
m.update(<thing to hash>)
hash = m.hexdigest()
```

a) What do you need to do to get the server to respond to your first message? Hello

```
▾ Data (5 bytes)
    Data: 48656c6c6f
    [Length: 5]
0000  82 2a fe 93 95 29 52 30  73 e9 9d a1 08 00 45 00   ·*···)R0 s·····E·
0010  00 39 74 36 40 00 40 06  08 ad ac 10 33 15 ac 10   ·9t6@·@· ····3···
0020  32 a6 96 4c 22 60 12 39  c4 fe e6 ec 28 c9 80 18   2··L"`·9 ····(···
0030  01 f6 be 07 00 00 01 01  08 0a e6 81 13 e3 38 16   ········ ······8·
0040  7d 82 48 65 6c 6c 6f                               }·Hello
```

a.

b) Second message? Hello, its Me

```
▾ Data (13 bytes)
    Data: 48656c6c6f2c20697473206d65
    [Length: 13]
0000  52 30 73 e9 9d a1 82 2a  fe 93 95 29 08 00 45 00   R0s····* ···)··E·
0010  00 41 d3 ee 40 00 40 06  a8 ec ac 10 32 a6 ac 10   ·A··@·@· ····2···
0020  33 15 22 60 96 4c e6 ec  28 c9 12 39 c5 03 80 18   3·"`·L·· (··9····
0030  01 fe 28 e9 00 00 01 01  08 0a 38 16 7d 82 e6 81   ··(····· ··8·}···
0040  13 e3 48 65 6c 6c 6f 2c  20 69 74 73 20 6d 65      ··Hello,  its me
```

a.

c) Third message? You may want to look at multiple interactions to figure out what’s happening?
Em sti, olleh

```
0000  82 2a fe 93 95 29 52 30  73 e9 9d a1 08 00 45 00   ·*···)R0 s·····E·
0010  00 41 74 38 40 00 40 06  08 a3 ac 10 33 15 ac 10   ·At8@·@· ····3···
0020  32 a6 96 4c 22 60 12 39  c5 03 e6 ec 28 d6 80 18   2··L"`·9 ····(···
0030  01 f6 be 0f 00 00 01 01  08 0a e6 81 13 e4 38 16   ········ ······8·
0040  7d 82 65 6d 20 73 74 69  20 2c 6f 6c 6c 65 48      }·em sti  ,olleH
```

a.

d) Fourth message?

```
▾ Data (64 bytes)
    Data: 333561393462303034663033626164363739643961336432…
    [Length: 64]
0000  82 2a fe 93 95 29 52 30  73 e9 9d a1 08 00 45 00   ·*···)R0 s·····E·
0010  00 74 74 40 40 00 40 06  08 68 ac 10 33 15 ac 10   ·tt@@·@· ·h··3···
0020  32 a6 96 4c 22 60 12 39  c5 33 e6 ec 29 28 80 18   2··L"`·9 ·3··)(··
0030  01 f6 be 42 00 00 01 01  08 0a e6 81 13 e5 38 16   ···B···· ······8·
0040  7d 83 33 35 61 39 34 62  30 30 34 66 30 33 62 61   }·35a94b 004f03ba
0050  64 36 37 39 64 39 61 33  64 32 65 36 38 30 31 34   d679d9a3 d2e68014
0060  65 39 33 33 64 38 38 35  35 33 30 37 30 66 61 66   e933d885 53070faf
0070  66 35 62 31 35 35 32 36  39 32 36 33 36 65 38 32   f5b15526 92636e82
0080  34 38                                              48
```

a.

b. Hash: 3eb0ee6b3934697e6cd753192e232abaabb72e80c28f55c5145893c23d50ad34

e) Fifth message?

```
0000  52 30 73 e9 9d a1 82 2a  fe 93 95 29 08 00 45 00   R0s····* ···)··E·
0010  00 44 d3 f6 40 00 40 06  a8 e1 ac 10 32 a6 ac 10   ·D··@·@· ····2···
0020  33 15 22 60 96 4c e6 ec  29 18 12 39 c5 33 80 18   3·"`·L·· )··9·3··
0030  01 fe f9 29 00 00 01 01  08 0a 38 16 7d 83 e6 81   ···)···· ··8·}···
0040  13 e4 6d 76 73 7a 69 6a  6e 73 68 69 74 6c 72 71   ··mvszij nshitlrq
0050  65 6d                                              em
```

a.

f) Sixth message? Did you successfully receive the critical info when you sent this message? How do you have to change your program?

a. You need to connect us to port 23232

```
▾ Data (43 bytes)
    Data: 596f75206e65656420746f20636f6e6e656374207573696e…
    [Length: 43]
0000  52 30 73 e9 9d a1 82 2a  fe 93 95 29 08 00 45 00   R0s····* ···)··E·
0010  00 5f 41 6f 40 00 40 06  3b 4e ac 10 32 a6 ac 10   ·_Ao@·@· ;N··2···
0020  33 15 22 60 97 28 f1 94  e9 c2 52 cf 3d f5 80 18   3·"`·(·· ··R·=···
0030  01 fe b0 22 00 00 01 01  08 0a 38 1c 10 f3 e6 86   ···"···· ··8·····
0040  a7 4f 59 6f 75 20 6e 65  65 64 20 74 6f 20 63 6f   ·OYou ne ed to co
0050  6e 6e 65 63 74 20 75 73  69 6e 67 20 73 6f 75 72   nnect us ing sour
0060  63 65 20 70 6f 72 74 20  32 33 32 33 32            ce port  23232
```

b.

c. I'm not entirely sure if this is the right outcome, as I'm not sure I properly decrypted the final two messages, however I'm fairly confident the message "You need to connect using source port 23232" is in fact the final message. This is because the message before gave the Critical Inf, but as an encrypted string, and the message that came after was this bit of critical information.