Khanh Nguyen
CSCE465-500
UIN 525000335

Homework 1

Problem 1: Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not a detailed explanation like the one in the tutorial.

The sequence of library calls is as of below:

1. pcap_lookupdev: finds the capture device.
2. pcap_lookupnet: returns network number of the capture device.
3. pcap_open_live: starts sniffing.
4. pcap_datalink: returns the type of link-layer headers.
5. pcap_compile: compiles filter expression.
6. pcap_setfilter: sets the filter from pcap_compile.
7. pcap_next: captures a single packet at a time.
8. pcap_loop: continues sniffing by looping.
9. pcap_freecode: deallocates memory.
10. pcap_close: end sniffing.

Problem 2: Why do you need the root privilege to run sniffex? Where does the program fail if executed without the root privilege?

The root privilege is required because the code needs access to network device and performs sniffing through network socket.

Without the root privilege, it will fail at pcap_open_live (open access to capture device).

Problem 3: Please turn on and turn off the promiscuous mode in the sniffer program. Can you demonstrate the difference when this mode is on and off? (Note: even if the promiscuous mode does not work in TCR, you can still demonstrate your trial with the mode on and off in the sniffer program) Please describe the use of promiscuous mode (explained in the class) and explain your observations.

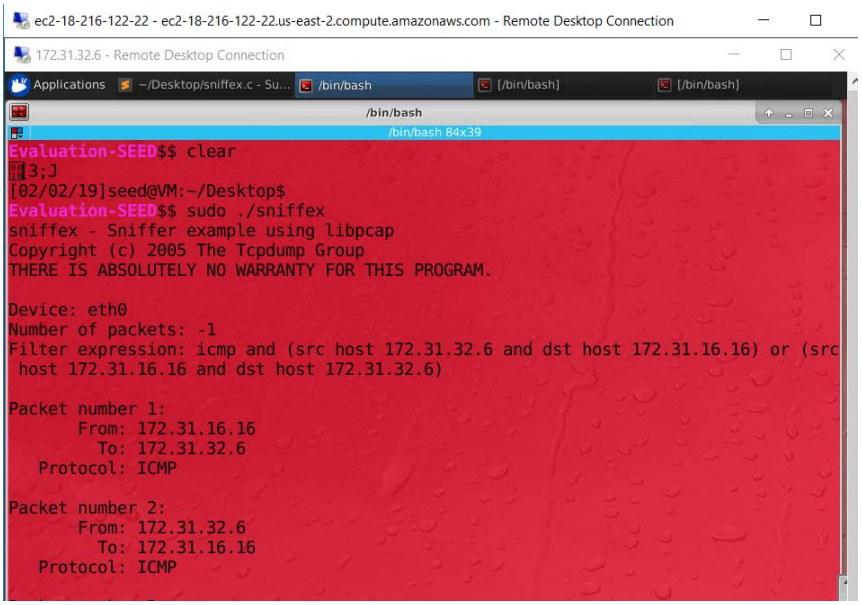
The promiscuous mode can be turn on and off by setting the third argument of pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf). 1 is on and 0 is off.

When the promiscuous mode is on, it shows all the contents of the package captured. Therefore the output is much more in promiscuous mode.

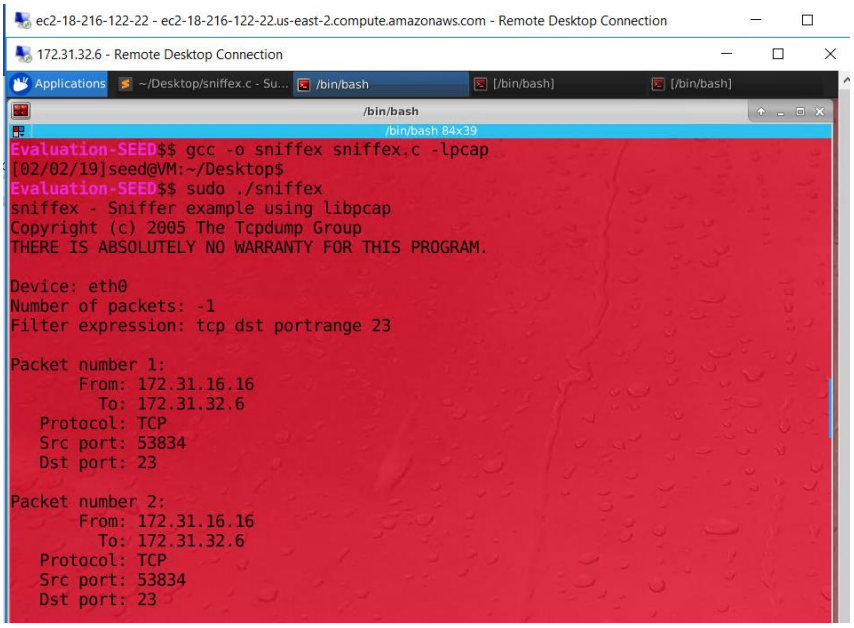
Problem 4: Please write filter expressions to capture each of the following. In your lab reports, you need to include screendumps to show the results of applying each of these filters.

- **Capture the ICMP packets between two specific hosts.**
- **Capture the TCP packets that have a destination port range from to port 10 - 100.**

To capture ICMP packets between two specific hosts, filter expression can be changed to "**icmp and (src host 172.31.32.6 and dst host 172.31.16.16) or (src host 172.31.16.16 and dst host 172.31.32.6)**". Screenshot is below:

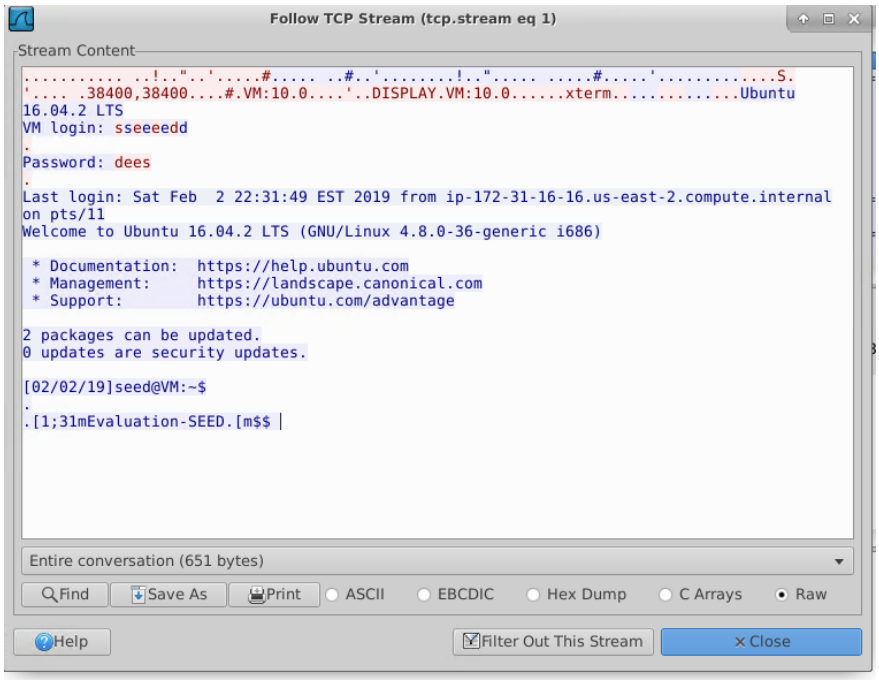


To capture TCP packets that have a destination port range from to port 10 – 100, filter expression can be changed to "**tcp dst portrange 10-100**". The Screenshot below:

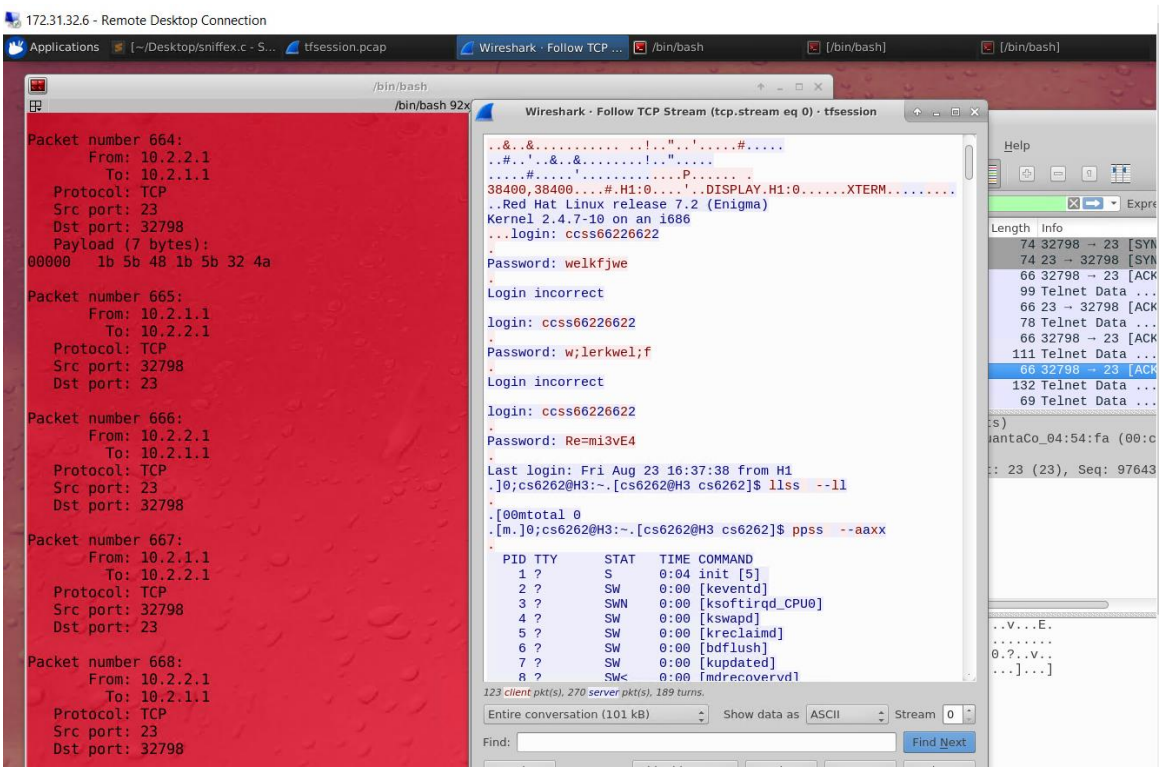


Problem 5: Please show how you can use sniffex to capture the password(s) when somebody is using telnet on the network that you are monitoring. You can start from modifying sniffex.c to implement the function. You also need to start the telnetd server on your VM. If you are using our pre-built VM, the telnetd server is already installed; just type the following command to start it.

Password can be capture by changing the filter expression to “tcp port 23” and telnet from another VM.



To capture password offline we can change the pcap_open_live(..) to pcap_open_offline(("tfsession.pcap",errbuf). The password captured from wireshark is below:



Problem 6: Please use your own words to describe the sequence of the library calls that are essential for packet spoofing. This is meant to be a succinct summary.

1. Socket(): creating a raw socket that help the program inject packets into the network.
2. Socketsetopt(): Set the socket option (we the ICMP protocol to send the package)
3. Construct the packet (swap the ip addresses of source and destination when duplicating the reply package)
4. Sendto():Send the packet through the raw socket made.

Problem 7: Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

This is because raw sockets can expose all information being transmitted over the network. The program will fail when a raw socket is attempted to be created.

Problem 8: Please combine your sniffing and the spoofing programs to implement a sniff-and-thenspoof program. This program monitors its local network; whenever it sees an ICMP echo request packet, it spoofs an ICMP echo reply packet with the information of another machine (other than the sender and receiver) in the network (in your AWS enclave).

This is the wireshark screenshot of the sniff-and-then-spoof program created. (The package with ip address of 172.31.1.1 is the spoofed package. Notice that there are two reply packages: one is real (172.31.32.6) and one is fake.). See attachment for the code.

ec2-18-216-122-22 - ec2-18-216-122-22.us-east-2.compute.amazonaws.com - Remote Desktop Connection

172.31.32.6 - Remote Desktop Connection

Applications | [~/Desktop/sniffexands... | Capturing from eth0 [Wi... | /bin/bash | [/bin/bash] | [/bin/bash]

Capturing from eth0 [Wireshark 2.0.2 (SVN Rev Unknown from unknown)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: icmp Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
63018	2019-02-03 0	172.31.16.16	172.31.32.6	ICMP	98	Echo (ping) request id=0x1
63019	2019-02-03 0	172.31.32.6	172.31.16.16	ICMP	98	Echo (ping) reply id=0x1
63022	2019-02-03 0	172.31.99.99	172.31.16.16	ICMP	74	Echo (ping) reply id=0xe
63241	2019-02-03 0	172.31.16.16	172.31.32.6	ICMP	98	Echo (ping) request id=0x1
63242	2019-02-03 0	172.31.32.6	172.31.16.16	ICMP	98	Echo (ping) reply id=0x1
63243	2019-02-03 0	172.31.99.99	172.31.16.16	ICMP	74	Echo (ping) reply id=0xe
64081	2019-02-03 0	172.31.16.16	172.31.32.6	ICMP	98	Echo (ping) request id=0x1
64082	2019-02-03 0	172.31.32.6	172.31.16.16	ICMP	98	Echo (ping) reply id=0x1
64083	2019-02-03 0	172.31.99.99	172.31.16.16	ICMP	74	Echo (ping) reply id=0xe
64651	2019-02-03 0	172.31.16.16	172.31.32.6	ICMP	98	Echo (ping) request id=0x1
64652	2019-02-03 0	172.31.32.6	172.31.16.16	ICMP	98	Echo (ping) reply id=0x1
64938	2019-02-03 0	172.31.99.99	172.31.16.16	ICMP	74	Echo (ping) reply id=0xe
65513	2019-02-03 0	172.31.16.16	172.31.32.6	ICMP	98	Echo (ping) request id=0x1

```
▸ Frame 63018: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▸ Ethernet II, Src: 0a:06:dd:b6:f6:98 (0a:06:dd:b6:f6:98), Dst: 0a:4d:c6:78:be:6e (0a:4d:c6:78:be:6e)
▸ Internet Protocol Version 4, Src: 172.31.16.16, Dst: 172.31.32.6
▸ Internet Control Message Protocol
```

```
0000  0a 4d c6 78 be 6e 0a 06  dd b6 f6 98 08 00 45 00   .M.x.n.. ......E.
0010  00 54 96 b9 40 00 40 01  1b 9b ac 1f 10 10 ac 1f   .T..@.@. ........
0020  20 06 08 00 f1 30 11 d3  00 01 c5 7e 56 5c df 1c    ....0.. ...~V\..
0030  0f 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

eth0: <live capture in progress... | Packets: 167328 · D... | Profile: Default