Lab 1&2: Traffic Sniffing & Ip Spoofing

Feng Wei

February 3, 2020

1 Traffic Sniffing

1.1 Introduction

In this lab we are going to learn how to collect desired network traffic by Wireshark[7]. Below is the experiment enviorment for this lab.

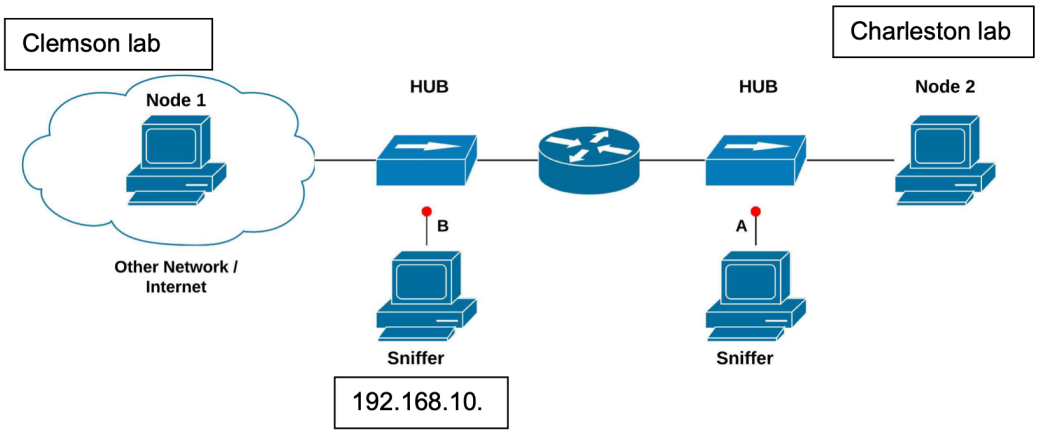

Figure 1: Lab Enviroment

The purpose of this lab is as below:

- Learn common tools used to collect network packets.
- Understand challenges in packet capturing and learn how to use capture and display filters.
- Be able to decide proper data collection points on a network.
- Have a better understanding of low level network communication and the structure of a network packet.

Necessary Equipment

- Two hubs
- Three host machines
- Wireshark / Tshark

1.2 Methods

I followed the instructions in the lab1 to finish the experiment step by step.

- The networks enviorment has already been set up.
- We used *ping* command to generate the traffic between Node1(Clemson) and Node2(Charleston). I used "Ping 192.168.20.11" to generate the background network traffic for sniffing.
- Collect and save the packets from the marked place (192.168.10.9) using Wireshark. I logged into this sniff machine with command "ssh -X wei8@192.168.10.9"
- Constrain your data collection for a specific type. After started Wireshark, first we need to choose the capture *Interface*. Then using *filter* such as *"ip.src == 192.168.10.8" or "ip.dst == 192.168.10.9"* to present the desired network traffic.
- To generate heavy traffic, I used command *"scp packts.pcap wei8@192.168.10.24:/home/wei8"* to transfer the captured packets file from the sniffing node *"192.168.10.9"* to my local node *"192.168.10.24"*.
- To plot the *time vs packets* graph, I used my own *Python* script.

1.3 Results

The experiment results are as below:

- Ping process *(I forgot to take screenshots, so I replayed those steps after the lab.)*

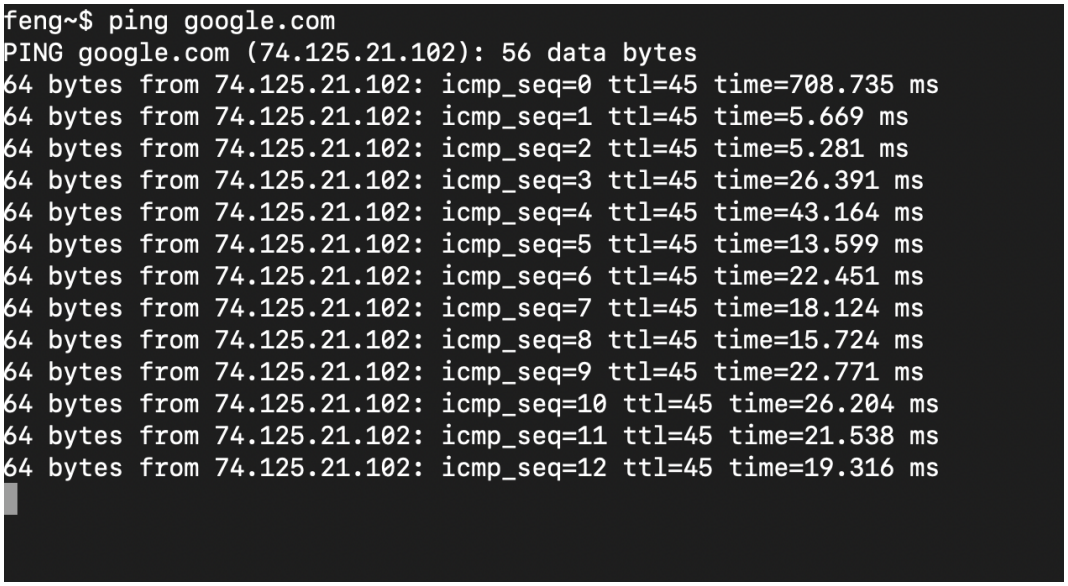

Figure 2: Ping

- Scp process

```
feng~/Downloads$ scp  -r wei8@access.computing.clemson.edu:/home/wei8/CpSc8860 .
/
jan15.pcapng                                  100% 2200MB   7.3MB/s   05:00
scpme.pcap                                     11%  293MB   7.2MB/s   05:09 ETA
```

Figure 3: Scp

- Tcpdump process

```
tions [nop,nop,TS val 581102201 ecr 521397042], length 1238
11:46:54.561332 IP access.computing.clemson.edu.ssh > 91.130.21.198.guest.wifi.d
yn.clemson.edu.52252: Flags [.], seq 105688489:105689727, ack 22652, win 323, op
tions [nop,nop,TS val 581102201 ecr 521397042], length 1238
11:46:54.561334 IP access.computing.clemson.edu.ssh > 91.130.21.198.guest.wifi.d
yn.clemson.edu.52252: Flags [.], seq 105689727:105690965, ack 22652, win 323, op
tions [nop,nop,TS val 581102201 ecr 521397042], length 1238
11:46:54.561335 IP access.computing.clemson.edu.ssh > 91.130.21.198.guest.wifi.d
yn.clemson.edu.52252: Flags [.], seq 105690965:105692203, ack 22652, win 323, op
tions [nop,nop,TS val 581102201 ecr 521397042], length 1238
11:46:54.561336 IP access.computing.clemson.edu.ssh > 91.130.21.198.guest.wifi.d
yn.clemson.edu.52252: Flags [.], seq 105692203:105693441, ack 22652, win 323, op
tions [nop,nop,TS val 581102201 ecr 521397042], length 1238
11:46:54.561337 IP access.computing.clemson.edu.ssh > 91.130.21.198.guest.wifi.d
yn.clemson.edu.52252: Flags [.], seq 105693441:105694679, ack 22652, win 323, op
tions [nop,nop,TS val 581102201 ecr 521397042], length 1238
11:46:54.561348 IP 91.130.21.198.guest.wifi.dyn.clemson.edu.52252 > access.compu
ting.clemson.edu.ssh: Flags [.], ack 105690965, win 19691, options [nop,nop,TS v
al 521397052 ecr 581102201], length 0
^C
101540 packets captured
107104 packets received by filter
5564 packets dropped by kernel
```

Figure 4: Tcpdump

- Wireshark Capture

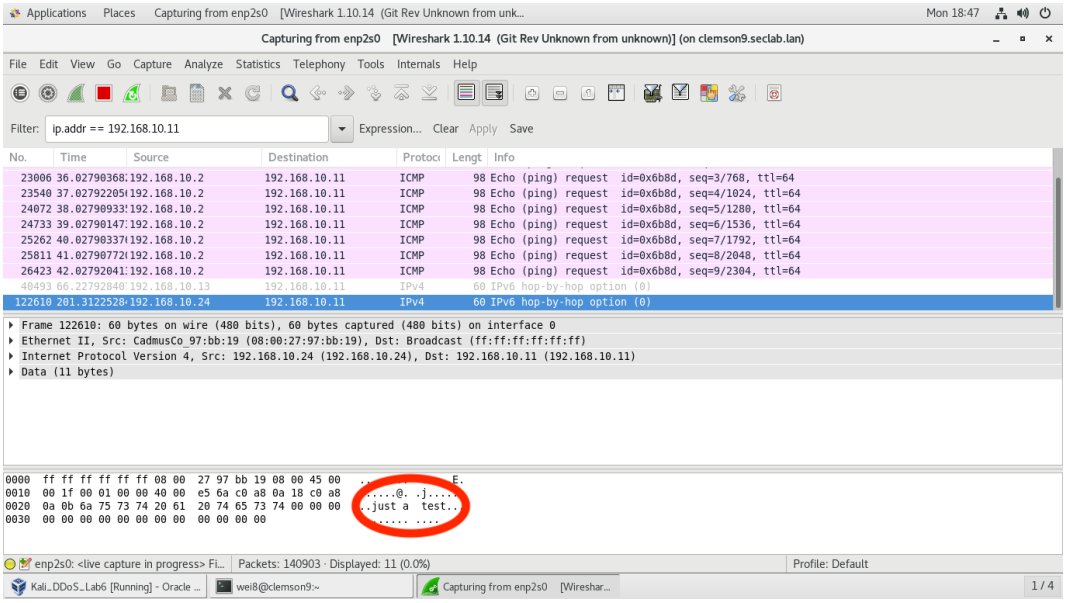

Figure 5: Wireshark Capture Packet

1.4 Question Answers

- Describe the layers and the fields of a packet captured from your network.
  Answer: The layers of the captured packets are as below.

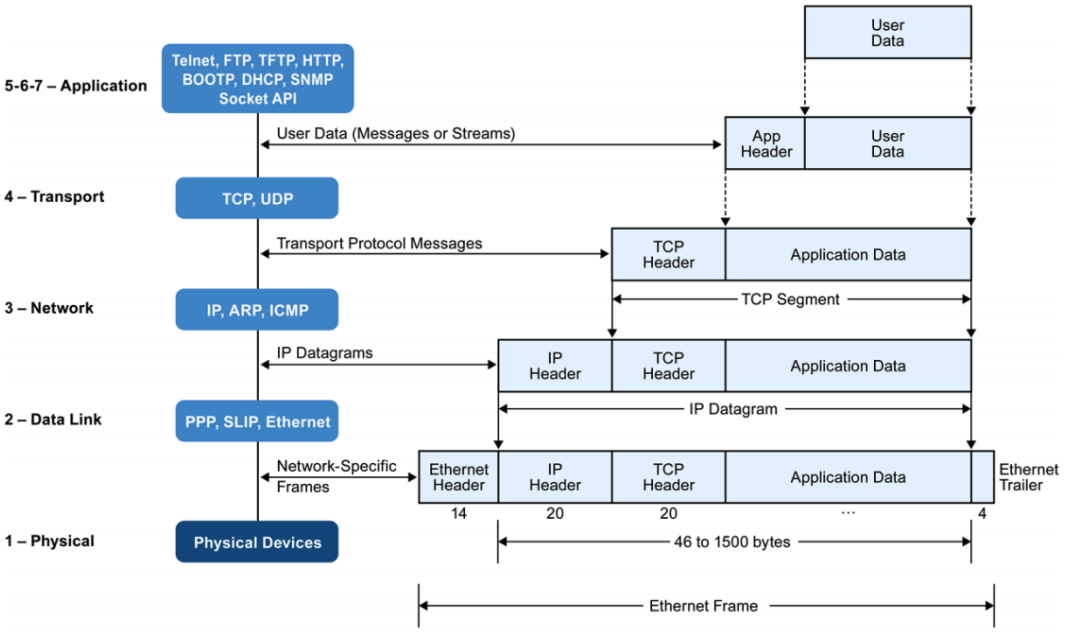

Figure 6: Layers

- Explain the characteristics and the functioning of a hub, switch and router in network science.
  Answer: Hubs, switches, and routers are all network devices that let us connect one or more network devices to other devices, or even other networks. Each of them has multiple connectors called ports, into which we plug the cables to make the connection.
- Explain the packet capture results obtained from the setups presented in Fig 1
  Answer: Using the set up experiment enviorment, we can capture all the traffic between Node 1 and Node 2.
- Write capture /display filters for Wireshark to collect packets coming from / going to a selected host on the network.
  Answer: Host *filter* such as *"ip.src == 192.168.10.8" or "ip.dst == 192.168.10.9"*
- Write capture / display filters to collect packets with specific protocol type on the network. For example collect only TCP packets.
  Answer: Just type *tcp* in the filter.
- Generate heavy traffic on the network. Did you drop any packets? If yes explain why.
  Answer: Yes, as shown in Fig 4. *"5564 packets dropped by the kernel"* That is because the kernel buffer space is limited, it cann't capture so many packets at the same time. To solve this problem we can use *"tcpdump -B ** "* to increase the kernel buffer size.
- Plot time-series graph for captured packets which is graph of number of packets vs time to demonstrate your data.
  Answer: Below is the *packets vs time* graph for scp scenario

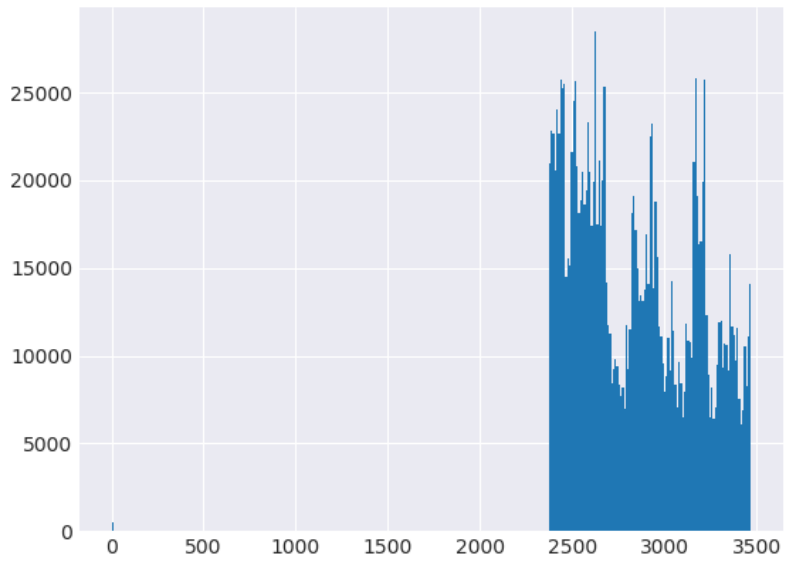

Figure 7: Heavy scenario

1.5 Discussion

- To generate the heavy traffic, maybe we can use *scp* command to transfer more files between Node 1 and Node 2.
- In order not to drop packets, we can increase the kernel buffer using command line *"tcpdump -B ** "*.

2 Ip Spoofing

2.1 Introduction

This is a Ip Spoofing exercise. We are going to spoof others' Ip address as well as detect whether our Ip address are spoofed or not.

2.2 Methods

- Start VM.
  Copy VM from TA's node to my node. *"scp wei8@192.168.10.10:tmpDDoS_Lab6.ova"*
  vboxmanage *"import Kali_DDoS_Lab6.ova"*
  log in with *"username: root password: private123"*
- Discover all the hosts in your subnet by using *nmap*[5].
- Start sniffing using Wireshark as Lab1.
- Spoof Ip address using Scapy [6] or Hping [4].
  send(IP(dst='192.168.10.2',src='192.168.10.11')/'just a test')
  sendp(Ether(src='12:34:56:78:9a:ab')/IP(dst= '192.168.10.11',src='8.8.8.8')/'jan 27')

2.3 Results

- Nmap

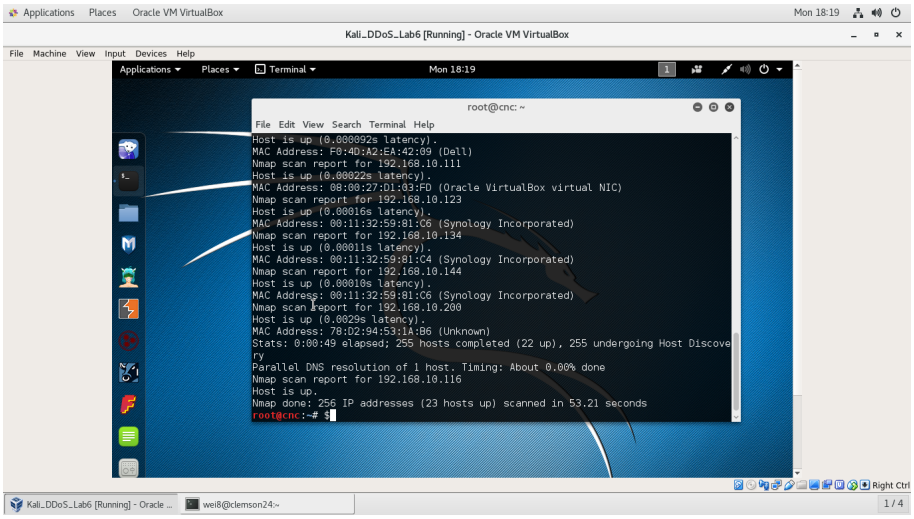

Figure 8: Nmap: 256 Ip address 23 hosts up

- Scapy spoof 1

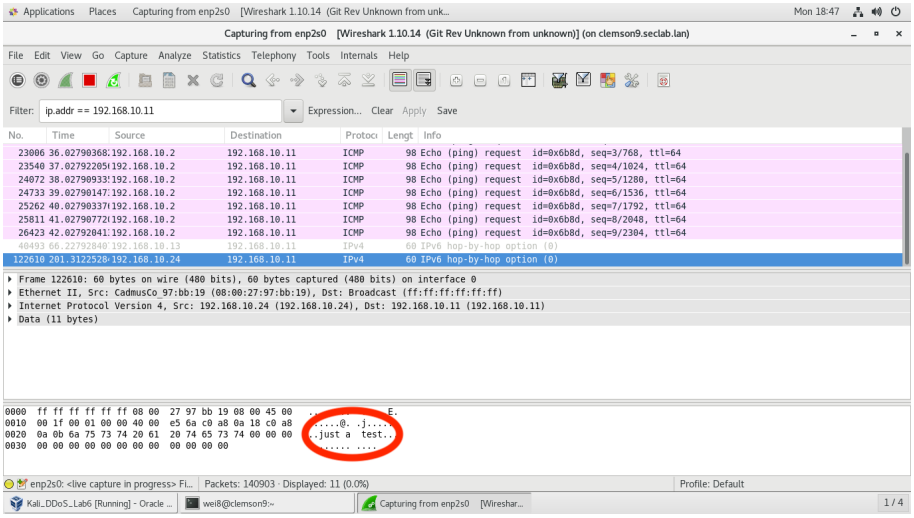

Figure 9: Spoof 1: just a test

- Scapy spoof 2

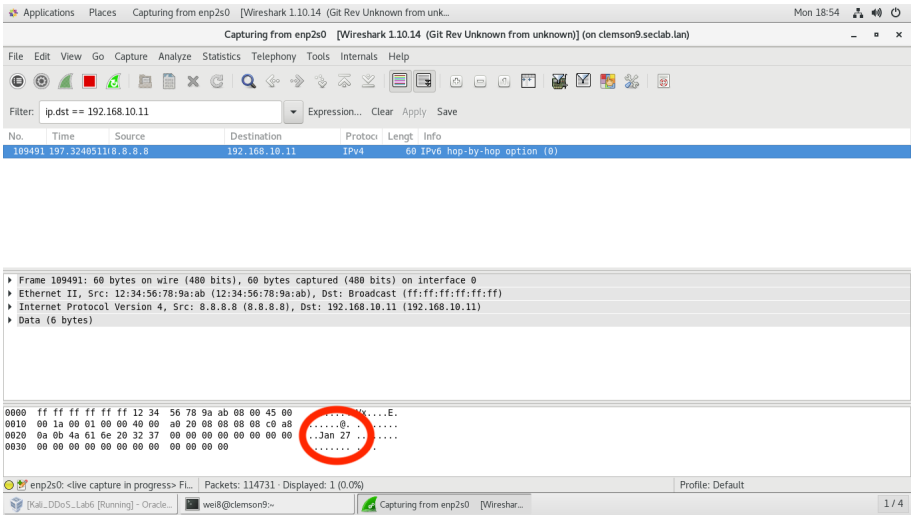

Figure 10: Spoof 1: Jan 27

2.4 Question Answers

- What filter did you use to detect the spoofing (show your results)?
  Answer: Using host filters *"ip.addr == 192.168.10.11"*.
- How can you evade detection? Is it possible to ascertain the identity of the sender?
  Answer: By carefully chose the *fake source Ip address*, attackers can successfully evade the detection.

To check the identity of the sender we need *Ip address or MAC address*, if all those address are fake then we can't find the identity of the senders.

- Read RFC 791 [2] then explain what IP address spoofing is, and what a host on the network must do to spoof its IP address.
  Answer: IP spoofing is the malicious creation of Internet Protocol (IP) packets which have a carefully altered source address in order to either hide the identity of the sender, to impersonate another computer system, or both. It is a technique often used by bad actors to invoke DDoS attacks against a target device or the surrounding infrastructure.
  To spoof an Ip address, we need to alter the network traffic packets header with an carefully chosen source Ip address.

- Read RFC 768 [1] and RFC 793 [3] then explain why an attacker cannot just grab any existing IP packet carrying UDP or TCP, change only the IP addresses in there, and expect the target host to accept the packet. Especially for TCP.
  Answer: UDP provides integrity verification by *Checksum* and Tcp using *Three ways handshakes*. So we can't just change the Ip address then expect the target host to accept the packet.
  To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:
  (1):SYN: The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a random value A.
  (2):SYN-ACK: In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number i.e. A+1, and the sequence number that the server chooses for the packet is another random number, B.
  (3):ACK: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A+1, and the acknowledgement number is set to one more than the received sequence number i.e. B+1.
  At this point, both the client and server have received an acknowledgment of the connection. The steps 1, 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. The steps 2, 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, a full-duplex communication is established.

2.5 Discussion

For this experiment, we can try to use multiple computers attack the same host at the same time.

References

[1] RFC 768. *https://www.ietf.org/rfc/rfc7.*
[2] RFC 791. *https://www.ietf.org/rfc/rfc791.txt.*
[3] RFC 793. *https://www.ietf.org/rfc/rfc793.*
[4] Hping. *http://www.hping.org.*
[5] Nmap. *https://nmap.org/.*
[6] Scapy. *https://scapy.readthedocs.io/en/latest/usage.html.*
[7] Wireshark. *https://www.wireshark.org/.*