CS 544/444: Introduction to Cybersecurity

Spring 2025

Afsah Anwar, afsah@unm.edu, Farris #2120.
TA: Hamim Md Adal, hmdadal@unm.edu, Farris #2150.

Network Security Lab

Guidelines

- Upload on Canvas under Network Security Lab.
- Late submissions will **NOT** be accepted.
- Students may collaborate among themselves to complete the homeworks/labs.
- Please follow the steps carefully and attach screenshots for each step as evidence.

Question 1: Getting Started with Network Mapping (10 pts)

Network mapping is the process of discovering all endpoints connected to a network. In penetration testing, it is used to identify devices and identify what ports and services a certain device is running. For this reason, it is important to be able to identify and map your network to understand what level of exposure you have. In this lab section, we will learn how to identify the devices running in a network using a popular network mapping tool, nmap.

To run the nmap (aka network mapper) tool, you will need a virtual machine running Kali Linux (See this link). After following the guidelines, just open a terminal, type "*nmap –help*" and you should see an output like Figure 1.

Question 2: Identifying devices on local network (40 pts)

We can use *nmap* to identify what devices are connected to our local network. To achieve this, perform the following steps:

```
┌──(kali㉿kali)-[~]
└─$ nmap --help
Nmap 7.93 ( https://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target specification}
TARGET SPECIFICATION:
  Can pass hostnames, IP addresses, networks, etc.
  Ex: scanme.nmap.org, microsoft.com/24, 192.168.0.1; 10.0.0-255.1-254
  -iL <inputfilename>: Input from list of hosts/networks
  -iR <num hosts>: Choose random targets
  --exclude <host1[,host2][,host3],...>: Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list from file
HOST DISCOVERY:
  -sL: List Scan - simply list targets to scan
  -sn: Ping Scan - disable port scan
  -Pn: Treat all hosts as online -- skip host discovery
  -PS/PA/PU/PY[portlist]: TCP SYN/ACK, UDP or SCTP discovery to given ports
  -PE/PP/PM: ICMP echo, timestamp, and netmask request discovery probes
  -PO[protocol list]: IP Protocol Ping
  -n/-R: Never do DNS resolution/Always resolve [default: sometimes]
  --dns-servers <serv1[,serv2],...>: Specify custom DNS servers
  --system-dns: Use OS's DNS resolver
  --traceroute: Trace hop path to each host
```

Figure 1: nmap help menu

1. Get your local network in CIDR format. (e.g. 192.168.1.1/24). You should get the network address from your original computer, not the VM. Hint: In Linux/Mac run *ifconfig* command, on Windows run *ipconfig* on the terminal.

2. On Kali, run the following command "*sudo nmap -sn 192.168.1.1/24*". Don't forget to replace the IP and subnet mask with yours. If your network supports many devices (e.g. /17 or /24) and is taking too long to scan, try setting up a mobile hotspot and connect one or two more devices to it. We can see an example result in Figure 2.

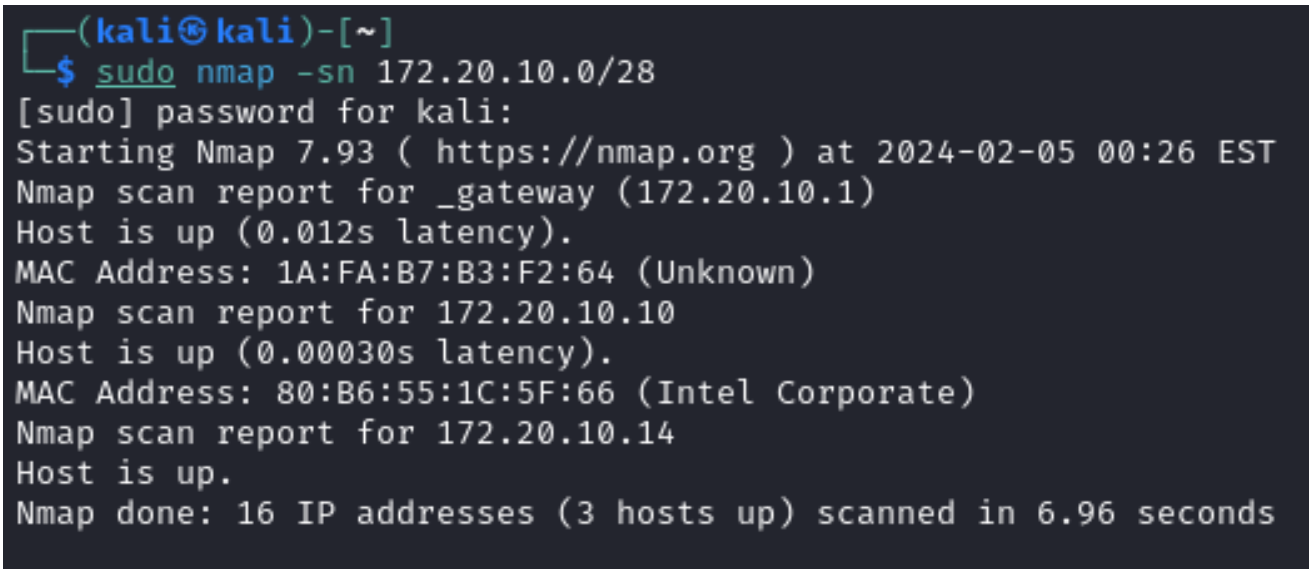

Figure 2: nmap scan for internal network

3. How many devices are up in your network? Can you identify any device?

4. Now target a specific IP, for this exercise use your original machine IP. See Figure 3, replicate, and answer: What is each flag doing? Can you see any open ports? If yes, what service is it running? Don't forget to replace the IP with your computer one.

5. Now, do some research and find a way to scan a specific port and get the service version of those ports. Hint: you can go to https://nmap.org/book/man.html or run *man nmap* to see the command's manual page.

6. Why is it important to not have any unwanted ports or services open?

7. Can we do port scanning to whatever device we want? What are the problems or legal issues about doing this?

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -p- -sS --open --min-rate 5000 -vvv 172.20.10.10
Starting Nmap 7.93 ( https://nmap.org ) at 2024-02-05 00:36 EST
Initiating ARP Ping Scan at 00:36
Scanning 172.20.10.10 [1 port]
Completed ARP Ping Scan at 00:36, 0.04s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 00:36
Completed Parallel DNS resolution of 1 host. at 00:36, 0.01s elapsed
DNS resolution of 1 IPs took 0.01s. Mode: Async [#: 3, OK: 1, NX: 0, DR: 0, SF: 0, TR: 1, CN: 0]
Initiating SYN Stealth Scan at 00:36
Scanning enrique-Zenbook (172.20.10.10) [65535 ports]
Discovered open port 8888/tcp on 172.20.10.10
Completed SYN Stealth Scan at 00:36, 1.71s elapsed (65535 total ports)
Nmap scan report for enrique-Zenbook (172.20.10.10)
Host is up, received arp-response (0.00010s latency).
Scanned at 2024-02-05 00:36:53 EST for 2s
Not shown: 65534 closed tcp ports (reset)
PORT     STATE SERVICE        REASON
8888/tcp open  sun-answerbook syn-ack ttl 64
MAC Address: 80:B6:55:1C:5F:66 (Intel Corporate)

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 1.92 seconds
           Raw packets sent: 65536 (2.884MB) | Rcvd: 65536 (2.621MB)
```

Figure 3: nmap scan for specific IP

Question 3: Packet sniffing and Spoofing (15 pts)

Packet sniffing and spoofing are two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security measures in networking. In this lab we will use a python module to sniff and spoof network packets.

To set up the packet sniffing and spoofing Lab do the following steps:

1. Install docker on the Kali Linux virtual machine by running the following commands:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install docker.io docker-compose
```

2. Download the lab zip file from:
   https://canvas.unm.edu/files/18780965/download?download_frd=1 or on canvas go to files → Labsetup.zip.

3. Decompress the zip file and enter the directory on the terminal.

4. Run the following command:

```
sudo docker-compose up -d
```

5. The previous command will set up 3 docker containers. To understand better what a container is you can go to the following page: https://www.docker.com/resources/what-container/.

6. Now that you have the containers running, you can see each container name with its corresponding ID with the following command (see Figure 4):

Figure 4: Docker container IDs and names

```
sudo docker ps --format "{{.ID}}-{{.Names}}"
```

7. For this lab, we will need to execute commands within each container, to be able to interact with a shell within each container run the following command:

```
sudo docker exec -it ID /bin/bash
```

   Don't forget to replace *ID* with the corresponding container ID.

In this lab, we will use three containers that are connected to the same LAN. Figure 5 depicts the lab environment. We will do all the attacks on the attacker container while using the other containers as the user machines.

- Attach an image of your lab architecture successfully working.

Question 4: Using Scapy to Sniff and Spoof Packets (35 pts)

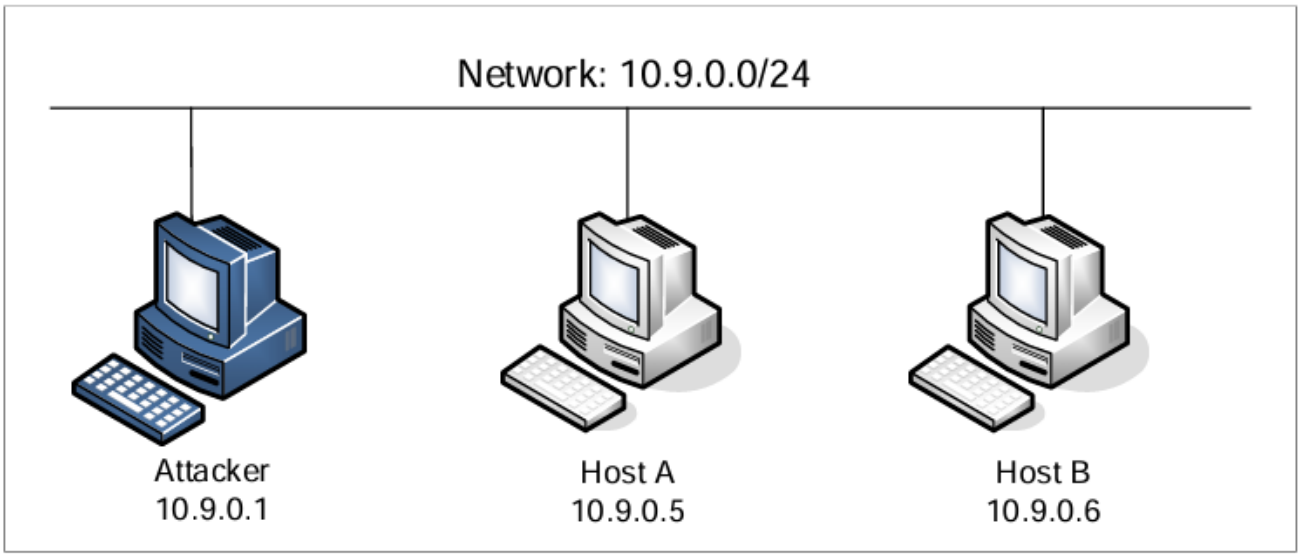

Figure 5: Lab architecture using three docker containers

Many tools can be used to do sniffing and spoofing, but most of them only provide fixed functionalities. Scapy is different: it can be used not only as a tool, but also as a building block to construct other sniffing and spoofing tools, i.e., we can integrate the Scapy functionalities into our own program. In this set of tasks, we will use Scapy for each task. To use Scapy, we can write a Python program, and then execute this program using Python. See the following example. We should run Python using the root privilege because the privilege is required for spoofing packets.

The following command is a basic example of a Python code using the Scapy library to get IP information.

```
#!/usr/bin/env python3
from scapy.all import *

a=IP()
a.show()
```

Question 4.1 Sniffing Packets

Wireshark is the most popular sniffing tool, and it is easy to use. However, it is difficult to use Wireshark as a building block to construct other tools. We will use Scapy for that purpose. The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs. A sample code is provided in the following:

```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()

pkt=sniff(iface='br-c93733e9f913', filter='icmp', prn=print_pkt)
```

The code above will sniff the packets on the br-c93733e9f913 interface. Change the interface name to the one corresponding to the network of the lab.

1. In the above program, for each captured packet, the callback function print_pkt() will be invoked; this function will print out some of the information about the packet. Run the program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations. There is a user called seed which you can switch to test the non-privileged.

2. Usually, when we sniff packets, we are only interested certain types of packets. We can do that by setting filters in sniffing. Scapy's filter uses the BPF(Berkeley Packet Filter) syntax; you can find the BPF manual on the Internet. Please set the following filters and demonstrate your sniffer program again (each filter should be set separately).

    - Capture only the ICMP packet.
    - Capture any TCP packet that comes from a particular IP and with a destination port number 23.
    - Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

Question 4.2 Spoofing ICMP packets

As a packet spoofing tool, Scapy allows us to set the fields of IP packets to arbitrary values. The objective of this task is to spoof IP packets with an arbitrary source IP address. We will spoof ICMP echo request packets, and send them to another VM on the same network. We will use Wireshark to observe whether our request will be accepted by the receiver. If it is accepted, an

echo reply packet will be sent to the spoofed IP address. The following code shows and example of how to spoof an ICMP packet.

```
#!/usr/bin/env python3
from scapy.all import *
a = IP()
a.dst = '10.0.2.3'
b = ICMP()
p = a/b
send(p)
```

In the code above, first we create an IP object from the IP class; a class attribute is defined for each IP header field. We can use ls(a) or ls(IP) to see all the attribute names/values. We can also use a.show() and IP.show() to do the same. Then we set the destination IP address field. If a field is not set the default value will be used.

```
>>> ls(a)
version    : BitField  (4 bits)          = 4                (4)
ihl        : BitField  (4 bits)          = None             (None)
tos        : XByteField                  = 0                (0)
len        : ShortField                  = None             (None)
id         : ShortField                  = 1                (1)
flags      : FlagsField  (3 bits)        = <Flag 0 ()>      (<Flag 0 ()>)
frag       : BitField  (13 bits)         = 0                (0)
ttl        : ByteField                   = 64               (64)
proto      : ByteEnumField               = 0                (0)
chksum     : XShortField                 = None             (None)
src        : SourceIPField               = '127.0.0.1'      (None)
dst        : DestIPField                 = '127.0.0.1'      (None)
options    : PacketListField             = []               ([])
```

Then we create an ICMP object, stacking a and b together to form a new object. Finally, the new object created is sent with the send() function.

- Use Wireshark and intercept the traffic going using the interface starting with "br-".
- In the attacker machine, create the script to spoof an ICMP packet using as src IP the Host A and target IP Host B.
- Demonstrate that the ICMP packet was successfully spoofed.

Question 4.3: Traceroute

The objective of this task is to use Scapy to estimate the distance, in terms of a number of routers, between your VM and a selected destination. This is basically what is implemented by the traceroute tool. In this task, we will write our own tool. The idea is quite straightforward: just send a packet (any type) to the destination, with its Time-To-Live (TTL) field set to 1 first. This packet will be dropped by the first router, which will send us an ICMP error message, telling us that the time-to-live has exceeded. That is how we get the IP address of the first router. We then increase our TTL to 2, send out another packet, and get the IP of the second router. We will repeat this procedure until our packet finally reaches its destination. It should be noted that this experiment only gets an estimated result, because in theory, not all these packets take the same route (but in practice, they may within a short period of time). The code in the following shows one round in the procedure.

```
a = IP()
a.dst = '1.2.3.4'
a.ttl = 3
b = ICMP()
send(a/b)
```

- Write a Python code to automate the procedure and get the traceroute automatically. Attach the code as a separate file from the submission.