NFS Lab 3: UDP and TCP

I Introduction

This lab session will introduce you to TCP and UDP communications and attacks using netcat and python programming.

I.1 Prerequisites

For this lab, you should make sure to be on your own device, or at least a linux VM, with root access.

You must install the following tools:

- wireshark
- netcat
- python3
- scapy (python3 package)
- docker (cf. https://docs.docker.com/engine/install/)

II UDP Protocol

Here are two command lines that will be useful for our UDP client/server implementations, as an example we use the port 9090:

```
# netcat or nc depending on your distro
# Each line is suppose to run in a different console
~$ netcat -u -l -k 9090
~$ netcat -u 127.0.0.1 9090
```

Question 1: Could you explain this setup, who is the client and the server?

Question 2: By looking at the *man netcat*, explain what are the options used here.

Now, we would like to recreate these client/server processes using python and sockets.

II.1 UDP Client

```
#!/usr/bin/python3

import socket

ip = "XXX.XXX.XXX.XXX"
port = XXX
msg = b"Hello, World!"

sock = socket.socket( socket.AF_INET, # Internet
                      socket.SOCK_DGRAM) # UDP
sock.sendto(...)
```

Question 3: This code is incomplete and need modification. Can you modify it to work like our netcat client?

To test your client, try to talk to your netcat server.

II.2 UDP Server

At this point, we should have a working UDP client receiving inputs from the standard input. Now, we would like to replace our server.

Question 4: Based on your client, and using the bind and recvfrom functions from *socket*, implement a UDP server.

II.3 UDP Ping-Pong Attack

Here is a squeleton of python code using scapy creating a packet pkt with IP, UDP, and data parts.

```
#!/usr/bin/python3

from scapy.all import *

ip = IP(src="XXX", dst="XXX")
udp = UDP(sport=XXX, dport=XXX)
data = "Hey\n"
pkt = ip/udp/data
send(pkt,verbose=0)
```

Question 5: As an attacker, complete this snippet to launch a ping-pong attack between two UDP servers. Don't forget, you should modify your servers' code to respond to incoming traffic for the attack to work.

Note: Depending on your scapy install, you might need to launch your script using:

```
~$ sudo python3 -E script.py
```

Checkpoint: Call a lab supervisor to show your progress.

III TCP Protocol

Here are two command lines that will be useful for our TCP client/server implementations, as an example we use the port 9090:

```
~$ netcat -lvn 9090
~$ netcat 127.0.0.1 9090
```

III.1 TCP client

As with our UDP client, we would like to replace the TCP netcat client with our own using python.

Question 6: Complete the following code snippet to act as our new TCP client.

```
#!/usr/bin/python3

import socket

ip = "XXX.XXX.XXX.XXX"
port = XXX
buffer_size = 1024
msg = "Hello, World!\n"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
s.connect(...)
s.send(...)
s.close()
```

III.2 TCP Server

Question 7: Using bind, listen, accept, and recv, complete this snippet to replace our netcat TCP server.

```
#!/usr/bin/python3

import socket

ip = "XXX.XXX.XXX.XXX"
port = XXX
buffer_size = 20
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# ...
```

III.3 TCP RST Attack

III.3.1 Docker Setup

For this lab, we will use docker containers to create virtual machines in a local network. The file is accessible from the NFS lecture page, or directly from https://avalonswanderer.github.io/assets/zip/nfs/tp3_docker.tar.gz.

Once downloaded, you should be able to unzip the folder, and then build and start the containers.

```
~$ docker compose build             # build the containers
~$ docker compose up -d             # start the containers in the background
~$ docker compose exec client bash  # launch a bash in the client container
~$ docker compose restart           # restart all containers
~$ docker compose down              # stop the containers
```

Our containers represent three devices: a client, a server, and an attacker. Here the attacker container can be ignored if you managed to install scapy and netcat on your host/VM. If so you can imagine the following figure with the Attacker being our regular host device:

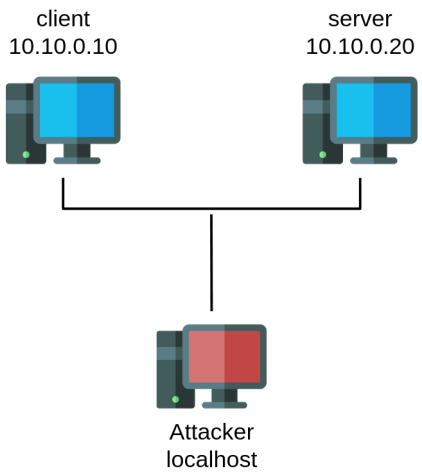

Figure 1: Network: 10.10.0.0/24

For our setup, we want to execute bash on our client (cf. docker commands above). Once on the machine, we want to open a TCP telnet connection on port 23 with the server.

```
client~$ telnet 10.10.0.20 23
```

For the telnet connection:
- login: victim
- pass: password

Question 8: Open wireshark on your host device. How can you filter the packets to only see the 10.10.0.0/24 network?

III.3.2 Mounting the attack

We want now to terminate the connection between the client and the server by sending a forged RST packet to one of them using this code:

```
#!/usr/bin/python3

from scapy.all import *

ip = IP(src="XXX", dst="XXX")
tcp = TCP(sport=XXX, dport=XXX, flags="X", seq=X)
pkt = ip/tcp
send(pkt,verbose=0)
```

Question 9: What does the TCP flags option represents? What is the one we need here?

Question 10: By looking at wireshark, complete this code snippet to send a RST packet to the server or the client.

Checkpoint Call a lab supervisor. Congratz you made it!

III.3.3 Bonus

Using sniff filter from scapy, try to sniff the packet between the client and the server to automatically retrieve the ports and seq#, to fill your forged RST packet.

```
#!/usr/bin/python3

import sys
import scapy.all import *

def attack(pkt):
  old_tcp_pkt = pkt[TCP] # TCP part of the sniffed packet.
  # ...
  send(pkt,verbose=0)

filter_str = 'tcp' # for now, matching all tcp packets.
sniff(filter=filter_str, prn=attack)
```

Acknowledgements

This work is inspired by previous materials from Mathieu Goessens, and the book *Internet Security: A Hands-on Approach (Computer & Internet Security) 3rd ed.* by Wenliang Du.