

# **Computer Networks**

## **Lab**

### **Assignment – 1**

## **Report**

**Name: Farrukh Riaz**

**Roll: 22i-1669**

**Sec: CY-B**

## 1. Creating a TCP client-server connection using C code:

### Server Side:

```
server.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <unistd.h>
6  #include <netinet/in.h>
7
8  int main()
9  {
10     char buf[200];
11
12     // create the server socket
13     int server_socket;
14     server_socket = socket(AF_INET, SOCK_STREAM, 0);
15
16     // define the server address
17     struct sockaddr_in server_address;
18     server_address.sin_family = AF_INET;
19     server_address.sin_port = htons(3939);
20     server_address.sin_addr.s_addr = INADDR_ANY;
21
22     // bind the socket to our specified IP and port
23     bind(server_socket, (struct sockaddr*) &server_address, sizeof(server_address));
24
25     // listen for connections
26     listen(server_socket, 5);
27
28     // accept the connection
29     int client_socket;
30     client_socket = accept(server_socket, NULL, NULL);
31
32     // communication
33     recv(client_socket, &buf, sizeof(buf), 0);
34     printf("\n%s\n", buf);
35
36     // close the socket
37     close(server_socket);
38
39     return 0;
40 }
```

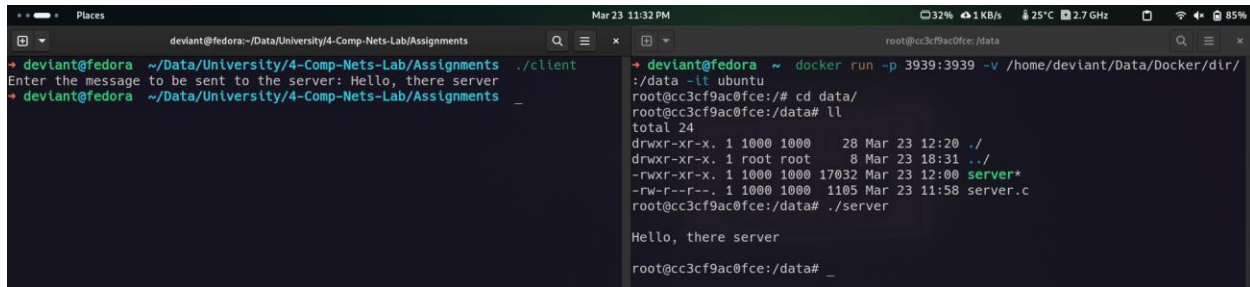
Creating a server agent that will listen and accept connection from clients. Created a **200-byte** buffer in which data will be received and stored from any client. Created a socket with IP info and connection type (TCP). Created a server address instance that shows entity with some identity info like IP, Port (**3939**), etc. In bind function we are passing socket and server address instance to bind our socket and server address, further we are listening for open connection with limit of **5**. Then used the accept function to accept any client request for connection. Now we are ready for communication by only accepting any data from the client side.

## Client Side:

```
C client.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <unistd.h>
6  #include <netinet/in.h>
7
8  int main()
9  {
10     char request[256];
11
12     // create the socket
13     int sock;
14     sock = socket(AF_INET, SOCK_STREAM, 0);
15
16     //setup an address
17     struct sockaddr_in server_address;
18     server_address.sin_family = AF_INET;
19     server_address.sin_addr.s_addr = INADDR_ANY;
20     server_address.sin_port = htons(3939);
21
22     // connect to the server
23     connect(sock, (struct sockaddr *) &server_address, sizeof(server_address));
24
25     printf("Enter the message to be sent to the server: ");
26     fgets(request, sizeof(request), stdin);
27     send(sock, request, sizeof(request), 0);
28
29     close(sock);
30
31     return 0;
32 }
```

Creating client agent that will be connected to server to access the server's services. Client side is also same until connect function in which we are passing our socket and server address to connect to the specific Port and IP. Using connect function to connect to the server by passing created socket. When the server accepts the client's connection request messages can be sent to the server, after sending data to server, connection is being closed on client side.

## 2. TCP connection and communication between client and server:



The screenshot shows two terminal windows. The left window is a client terminal with the prompt `deviant@fedora: ~/Data/University4-Comp-Nets-Lab/Assignments`. It shows the command `./client` being executed, which sends the message "Hello, there server" to the server. The right window is a server terminal with the prompt `root@cc3cf9ac0fce:/data`. It shows the command `docker run -p 3939:3939 -v /home/deviant/Data/Docker/dlr/:data -it ubuntu` being executed, which starts a container named `cc3cf9ac0fce`. Inside the container, the `ls` command is run, showing the directory structure. The server terminal also shows the message "Hello, there server" being received from the client.

Because we need to 2 devices on a network, so we created and docker container and mapped our container 3939 port to docker host

**Command:** `docker run -p 3939:3939 -v /volume_on_host/:/mount_on_container/ -it ubuntu`

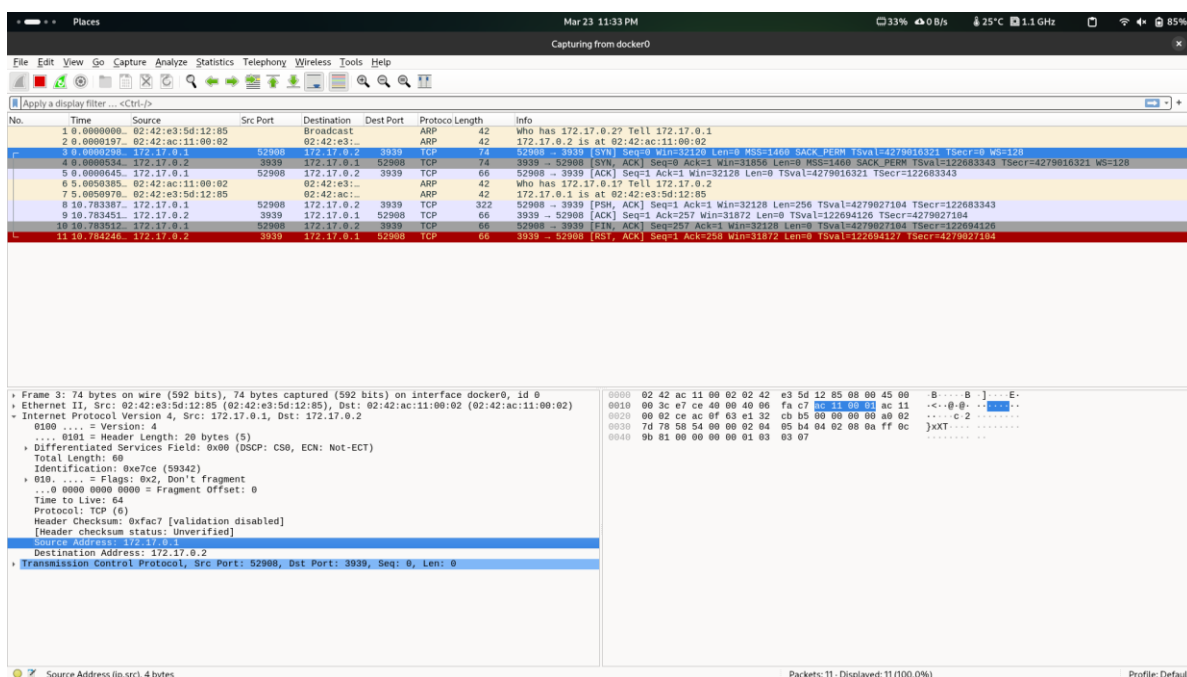
**-p** is mapping port between docker host and docker container.

**-v** is mounting volume between docker host and docker container.

**-it** is running container in interactive mode; means we are using container's stdin and stdout file descriptors

Our server is running on ubuntu machine on the same network and accepting connection and data. Client sent **"Hello, there server"** message to server and server received that message and printed on the interactive terminal.

## 3-5. Capturing network traffic via wireshark during communications:



## 6-7. Source and Destination IP:

```
Total Length: 60
Identification: 0xe7ce (59342)
  010. .... = Flags: 0x2, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header Checksum: 0xfac7 [validation disabled]
[Header checksum status: Unverified]
Source Address: 172.17.0.1
Destination Address: 172.17.0.2
  Transmission Control Protocol, Src Port: 52908, Dst Port: 3939, Seq: 0, Len: 0
```

Source IP is **172.17.0.1**, which is the IP of our **host client**.

Destination IP is **172.17.0.2**, which is **server container's** IP.

## 8. Sniffing packet data:

```

Frame 8: 322 bytes on wire (2576 bits), 322 bytes captured (2576 bits) on interface docker0, id 0
Ethernet II, Src: e92c:42:e3:5d:12:85 (02:42:e3:5d:12:85), Dst: 02:42:ac:11:08:02 (02:42:ac:11:08:02)
Internet Protocol Version 4, Src: 172.17.0.1, Dst: 172.17.0.2
0100 .... = Version: 4
ff 02 = Header Length: 20 bytes (16)
+ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 308
Identification: 0x0700 (59344)
0000 ... = Flags: 0x2, Don't Fragment
... 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header checksum: 0xfcd5 [validation disabled]
[Header checksum status: Unverified]
Source Address: 172.17.0.1
0070 ... = Destination Address: 172.17.0.2
+ Transmission Control Protocol, Src Port: 52908, Dst Port: 3939, Seq: 1, Ack: 1, Len: 256
Data (256 bytes)
Length: 48656c6cf2c267468657265297365727665720a09000000001bdf25c541ad0900000000...
[Data: 256]

```

As we can see on the bottom left side Data Length 256 because we created 256-byte buffer on client-side code that was sent to the server. On the right side we can see data of that buffer that was sent from client to server. In the beginning of buffer **“Hello, there server”** is written, it is the same data that we sent from client side to server.

**9-10. Source and Destination Port:**

```

Frame 8: 322 bytes on wire (2576 bits), 322 bytes captured (2576 bits) on interface docker0, id 0
Ethernet II, Src: Intel E81C8:8D:12:85 (02:42:e3:5d:12:85), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
Internet Protocol Version 4, Src: 172.17.0.1, Dst: 172.17.0.2
Transmission Control Protocol, Src Port: 52908, Dst Port: 3939, Seq: 1, Ack: 1, Len: 256
Source Port: 52908
Destination Port: 3939
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (63)]
[TCP Segment Len: 256]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 3778202550
[Next Sequence Number: 257 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 459225095
1000 .... = Header Length: 32 bytes (8)
Flags: 0x018 (PSH, ACK)
Window: 251
[Calculated window size: 32128]
[Window size scaling factor: 128]
Checksum: 0x594c [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]
TCP payload (256 bytes)

```

**Source Port 52908** which was random.

**Destination Port 3939**, because our server was **listening on port 3939** as we wrote in the code.

## 11. Modifying client and server code to make it harder to sniff the data:

### Client Side:

```
23     connect(sock, (struct sockaddr *) &server_address, sizeof(server_address));
24
25     printf("Enter the message to be sent to the server: ");
26     fgets(request, sizeof(request), stdin);
27     // modify the message to be sent to the server that it is not easy to sniff
28     for(int i = 0; i < sizeof(request); i++){
29         request[i] = request[i] + 1;
30     }
31     send(sock, request, sizeof(request), 0);
32
33     close(sock);
34
35     return 0;
36 }
```

Before sending data to the server, we applied simple encryption on data to make it harder to read it on the communication channel.

### Server Side:

```
30
31     // communication
32     recv(client_socket, &buf, sizeof(buf), 0);
33     for(int i=0; i<sizeof(buf); i++){
34         buf[i] = buf[i] - 1;
35     }
36     printf("\n%s\n", buf);
37
38     // close the socket
39     close(server_socket);
40
41     return 0;
42 }
```

Server side reads the data sent from client, and before printing it on the screen, it applied decryption on the data.

## 12. Observing secure communication via wireshark:

```
deviant@fedora ~/Data/University/4-Comp-Nets-Lab/Assignments
+ deviant@fedora ~$ ./client
Enter the message to be sent to the server: Hello, there server
+ deviant@fedora ~$ ./client
Enter the message to be sent to the server: Hello, there server
+ deviant@fedora ~$ _

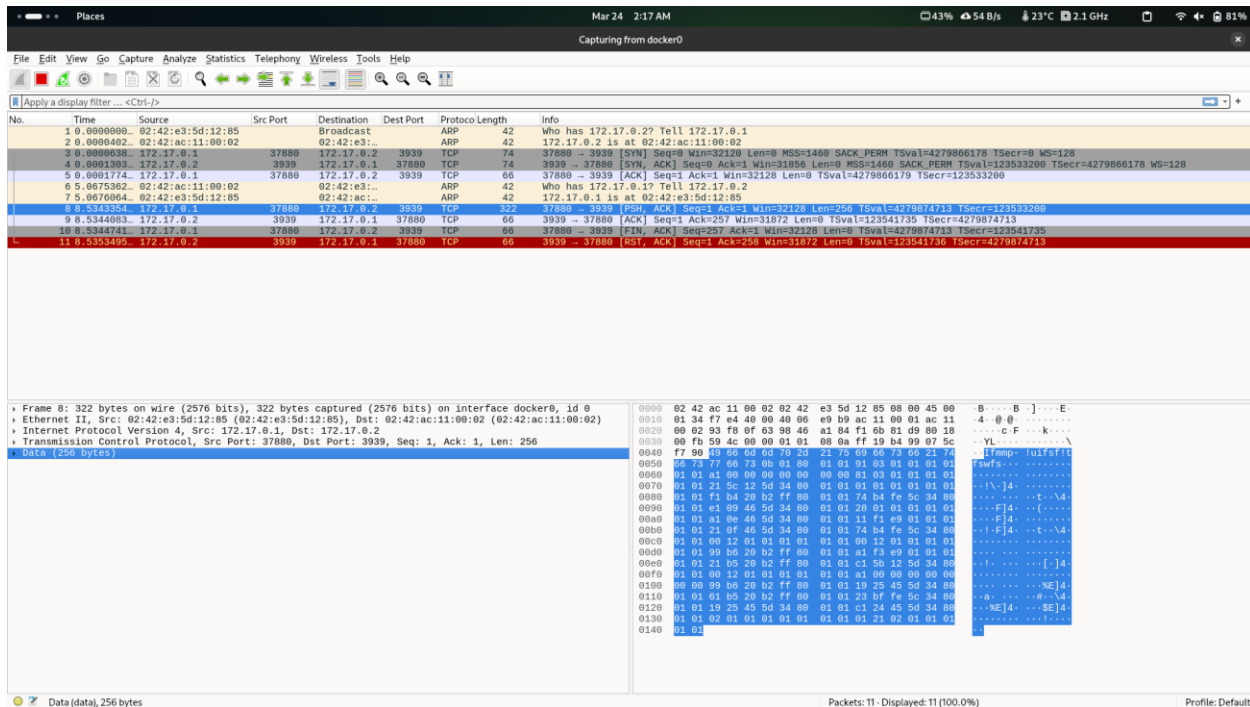
root@a258f63dcb0f:/data
+ deviant@fedora ~$ docker run -p 3939:3939 -v /home/deviant/Data/Docker/dir/
:/data -it ubuntu
root@cc3cf9ac0fce:/# cd data/
root@cc3cf9ac0fce:/data# ll
total 24
drwxr-xr-x. 1 1000 1000 28 Mar 23 12:20 ./
drwxr-xr-x. 1 root root 8 Mar 23 18:31 ../
-rwxr-xr-x. 1 1000 1000 17032 Mar 23 12:00 server*
-rw-r--r--. 1 1000 1000 1105 Mar 23 11:58 server.c
root@cc3cf9ac0fce:/data# ./server

Hello, there server

root@cc3cf9ac0fce:/data# exit
exit
+ deviant@fedora ~$ docker run -p 3939:3939 -v /home/deviant/Data/Docker/dir/
:/data -it ubuntu
root@a258f63dcb0f:/# cd data/
root@a258f63dcb0f:/data# ll
total 24
drwxr-xr-x. 1 1000 1000 28 Mar 23 21:16 ./
drwxr-xr-x. 1 root root 8 Mar 23 21:16 ../
-rwxr-xr-x. 1 1000 1000 17032 Mar 23 21:14 server*
-rw-r--r--. 1 1000 1000 1105 Mar 23 11:58 server.c
root@a258f63dcb0f:/data# ./server

Hello, there server

root@a258f63dcb0f:/data#
root@a258f63dcb0f:/data# _
```



We again captured data during TCP connection communication and tried to sniff that data which is “lfmmp- !uifsf!t fswff”, but this time our data is in random form, and it is not understandable.

In this way we had secure communication on the communication channel.