CSE 433S HW 1 - Comm with Stream Cipher

Zihan Chen

September 2024

Contents

1 Implementation

According to the resource in HW1 , I achieve the server and client with C programming language.

In c, there is a strcut called sockaddr_in, which is used to store the IP address and port number.

```
// Inialize the server address and port
struct sockaddr_in server_addr;
char server_ip[16] = "192.168.92.132";
int server_port = 10888;

// Create socket, Same in client and server
int sever_sock = socket(AF_INET, SOCK_STREAM, 0);

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(server_port);
server_addr.sin_addr.s_addr = inet_addr(server_ip);
```

The server will bind the socket to the server address and port, and listen to the port. And the client will connect to the server address and port.

```
// Bind to the set port and IP
if(bind(sever_sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
    perror("Binding failed with error!");
    return 1;
}

// Send connection request to server
// be sure to set port and IP the same as server-side
if(connect(sock, (struct sockaddr*) &server_addr, sizeof(server_addr)) < 0) {
    perror("Connection failed with error!");
    return 1;
}
```

After the connection is established, the server will listen to the port and wait for the client to send the message. The client will send the message to the server, and this message is fget from stdin.

```
// Receive client's message
varread = recv(client_sock, client_message, 1024, 0);
printf("Msg from client: %s\n", client_message);

// Get input from the user:
printf("Enter message sent to the server: ");
```

```
fgets(client_message, sizeof(client_message), stdin);

// Send the message to server:
send(sock, client_message, strlen(client_message), 0);
```

The server will receive the message from the client, and send another message to the client. The client will receive the message from the server and print it out.

```
// Receive the server's response:
varread = read(sock, server_message, 1024);

printf("Server's response: %s\n",server_message);

// Respond to client
strcpy(server_message, "##Hello, Bob! This is Alice.##");
send(client_sock, server_message, strlen(server_message), 0);
```

In fact, in my Implementation, the server and client are all one-time program. But in the real world, the server can always listen to the port and wait for the client to connect. I can achieve this by using a while loop to keep the server running and also use memset to clear the message buffer.

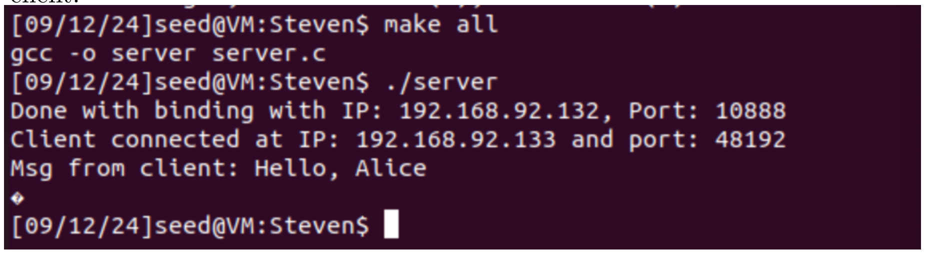
```
// while loop
while(1) {

    close(client_sock);
    memset(client_message, 0, sizeof(client_message));
}
```

2 Clinet and Server

I run the server and client in two terminals.

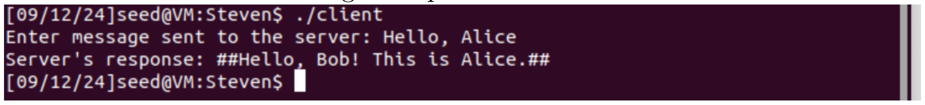
The server is listening on port 10888.

The server receives the message and sends another one from stdin back to the client.



```
[09/12/24]seed@VM:Steven$ make all
gcc -o server server.c
[09/12/24]seed@VM:Steven$ ./server
Done with binding with IP: 192.168.92.132, Port: 10888
Client connected at IP: 192.168.92.133 and port: 48192
Msg from client: Hello, Alice
�
[09/12/24]seed@VM:Steven$
```

The client sends a message to the server.
The client receives the message and prints it out.



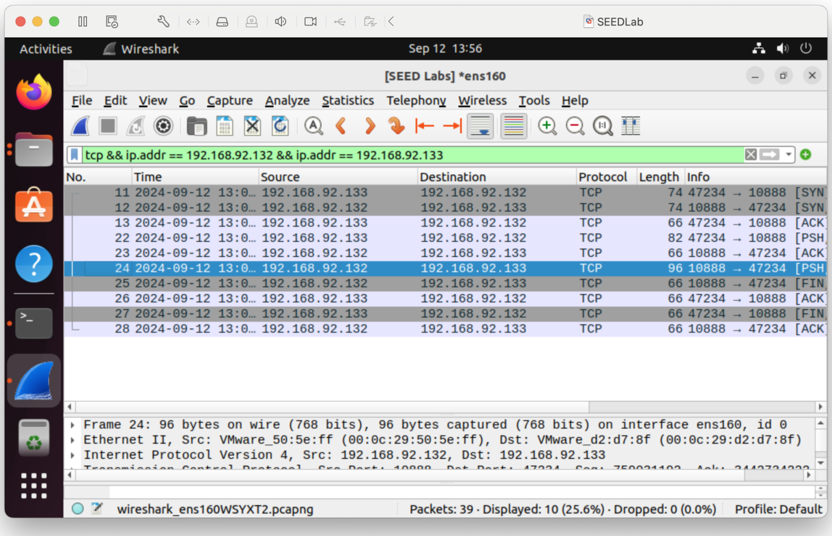
```
[09/12/24]seed@VM:Steven$ ./client
Enter message sent to the server: Hello, Alice
Server's response: ##Hello, Bob! This is Alice.##
[09/12/24]seed@VM:Steven$ 
```

3 Wireshark Analysis

Open wireshark, type `tcp && ip.addr == 192.168.92.132 && ip.addr == 192.168.92.133` to capture the target stream.

This can help me to capture the related packets about the server and client.

The first 3 packets is TCP handshake, we can easily find that the client will send a SYN packet to the server, and the server will send a SYN-ACK packet back to the client, and the client will send a ACK packet to the server. After all these packets is sent, the connection will be established.



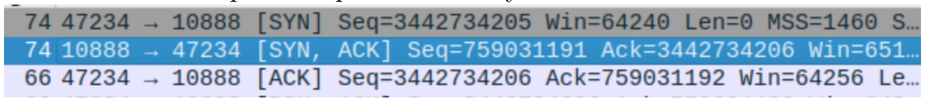
SEEDLab
Activities
Wireshark
Sep 12 13:56
[SEED Labs] *ens160
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp && ip.addr == 192.168.92.132 && ip.addr == 192.168.92.133
```

```
No.  Time              Source          Destination     Protocol Length Info
  11 2024-09-12 13:0… 192.168.92.133  192.168.92.132  TCP        74 47234 → 10888 [SYN
  12 2024-09-12 13:0… 192.168.92.132  192.168.92.133  TCP        74 10888 → 47234 [SYN
  13 2024-09-12 13:0… 192.168.92.133  192.168.92.132  TCP        66 47234 → 10888 [ACK
  22 2024-09-12 13:0… 192.168.92.133  192.168.92.132  TCP        82 47234 → 10888 [PSH
  23 2024-09-12 13:0… 192.168.92.132  192.168.92.133  TCP        66 10888 → 47234 [ACK
  24 2024-09-12 13:0… 192.168.92.132  192.168.92.133  TCP        96 10888 → 47234 [PSH
  25 2024-09-12 13:0… 192.168.92.132  192.168.92.133  TCP        66 10888 → 47234 [FIN
  26 2024-09-12 13:0… 192.168.92.133  192.168.92.132  TCP        66 47234 → 10888 [ACK
  27 2024-09-12 13:0… 192.168.92.133  192.168.92.132  TCP        66 47234 → 10888 [FIN
  28 2024-09-12 13:0… 192.168.92.132  192.168.92.133  TCP        66 10888 → 47234 [ACK

› Frame 24: 96 bytes on wire (768 bits), 96 bytes captured (768 bits) on interface ens160, id 0
› Ethernet II, Src: VMware_50:5e:ff (00:0c:29:50:5e:ff), Dst: VMware_d2:d7:8f (00:0c:29:d2:d7:8f)
› Internet Protocol Version 4, Src: 192.168.92.132, Dst: 192.168.92.133

wireshark_ens160WSYXT2.pcapng    Packets: 39 · Displayed: 10 (25.6%) · Dropped: 0 (0.0%)    Profile: Default
```

We can also find there is also a sequence number and ack number in the TCP header. In the first packet, the sequence number is 3442734205, and the ack number is 0. In the second packet, the sequence number is 759031191, and the ack number is 3442734206. In the third packet, the sequence number is 3442734206, and the ack number is 759031192.

All the ack number is the sequence number plus 1. The sequence number is the ack number of the previous packet and only related to the machine.

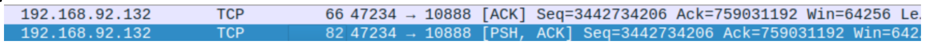


```
74 47234 → 10888 [SYN] Seq=3442734205 Win=64240 Len=0 MSS=1460 S…
74 10888 → 47234 [SYN, ACK] Seq=759031191 Ack=3442734206 Win=651…
66 47234 → 10888 [ACK] Seq=3442734206 Ack=759031192 Win=64256 Le…
```

The fourth and sixth packets are the data packets that the client and sever send to each other.

There is an interesting thing here, the sequence number and ack number of the fourth packet is the same as the third packet. This is because in ACK packet, there is no data transfer, so in the fourth packet, the seq and ack number will remain the same as the third packet.

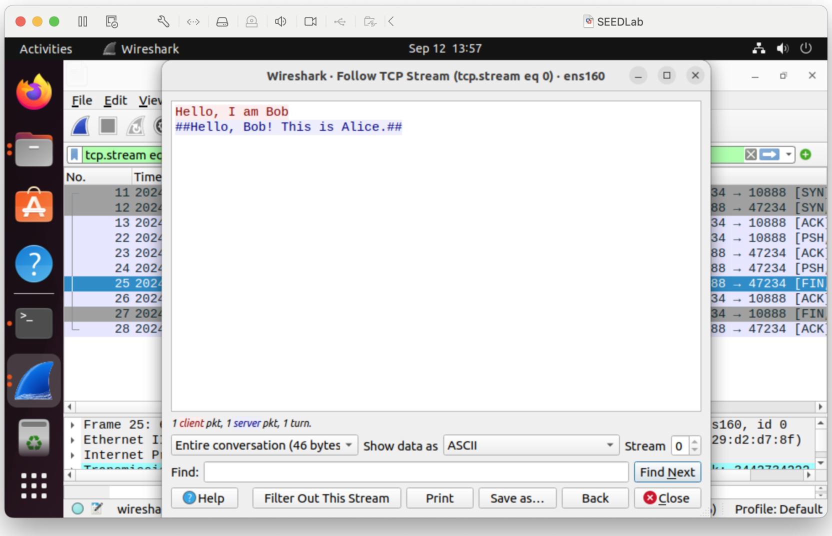
What's more, PSH is the flag that means the data is pushed to the application layer. So when the server and client send the data, the PSH flag will be set.



```
192.168.92.132    TCP    66 47234 → 10888 [ACK] Seq=3442734206 Ack=759031192 Win=64256 Le
192.168.92.132    TCP    82 47234 → 10888 [PSH, ACK] Seq=3442734206 Ack=759031192 Win=642
```

We can find out the data using TCP stream.

The Server send that: "Hello, I am Bob", and the client send that: "##Hello, Bob! This is Alice.##".



Wireshark · Follow TCP Stream (tcp.stream eq 0) · ens160

```
Hello, I am Bob
##Hello, Bob! This is Alice.##
```

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (46 bytes)   Show data as ASCII   Stream 0

```
11 2024    34 → 10888 [SYN
12 2024    88 → 47234 [SYN
13 2024    34 → 10888 [ACK
22 2024    34 → 10888 [PSH
23 2024    88 → 47234 [ACK
24 2024    88 → 47234 [PSH
25 2024    88 → 47234 [FIN
26 2024    34 → 10888 [ACK
27 2024    34 → 10888 [FIN
28 2024    88 → 47234 [ACK
```

4 Challenge & Solution

I face the problem when I try to achieve the server and client. I have no idea about how to achieve the server and client with C programming language. And also C is not a language that I am familiar with in designing the server and client.

But I find the resource mentioned in the HW1, and I follow the steps to achieve the server and client. C's pocket is not as easy as python. But after I read the sample code and understand the logic, I finally achieve the server and client.

5 Source Code

Server code:

```
// server.c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <errno.h>
#include <netinet/in.h>
#include <unistd.h>

int main() {
    // Declare variables
 ssize_t varread;
    char server_message[1024];
    char client_message[1024];

    struct sockaddr_in server_addr;
    char server_ip[16]= "192.168.92.132";
    int server_port = 10888;

    struct sockaddr_in client_addr;
    socklen_t client_addr_len = sizeof(client_addr);


    // Create socket
    int client_sock;
   int sever_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sever_sock < 0) {
        perror("Socket creation failed with error!");
        return 1;
    }
    if(setsockopt(sever_sock, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) < 0) {
        perror("Socket option failed with error!");
        return 1;
    }
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(server_port);
    server_addr.sin_addr.s_addr = inet_addr(server_ip);


    // Bind to the set port and IP
    if(bind(sever_sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("Binding failed with error!");
        return 1;
    }
    printf("Done with binding with IP: %s, Port: %d\n", server_ip, server_port);

    // Listen for clients:
    if(listen(sever_sock, 3) < 0) {
        perror("Listening failed with error!");
        return 1;
    }

    // Accept an incoming connection
    if((client_sock = accept(sever_sock, (struct sockaddr*)&client_addr, &client_addr_len))
        perror("Accepting failed with error!");
        return 1;
    }
    char * client_ip = inet_ntoa(client_addr.sin_addr);
    int client_port = ntohs(client_addr.sin_port);
    printf("Client connected at IP: %s and port: %i\n", client_ip, client_port);

    // Receive client's message
    varread = recv(client_sock, client_message, 1024, 0);
    printf("Msg from client: %s\n", client_message);


    // Respond to client
    strcpy(server_message, "##Hello, Bob! This is Alice.##");
    send(client_sock, server_message, strlen(server_message), 0);

    // Close the socket
    close(client_sock);
    close(sever_sock);


    return 0;
}
```

Client code:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <errno.h>
#include <unistd.h>

int main() {
    // Declare Variables
    struct sockaddr_in server_addr;

    char server_ip[16] = "192.168.92.132";
    int server_port = 10888;
    char server_message[1024];
    char client_message[1024];

    ssize_t varread;

    // Create socket:
    int sock = socket(AF_INET, SOCK_STREAM, 0);

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(server_port);
    server_addr.sin_addr.s_addr = inet_addr(server_ip);

    // Send connection request to server, be sure to set port and IP the same as server
    if(connect(sock, (struct sockaddr*) &server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed with error!");
        return 1;
    }

    // Get input from the user:
    printf("Enter message sent to the server: ");
    fgets(client_message, sizeof(client_message), stdin);

    // Send the message to server:
    send(sock, client_message, strlen(client_message), 0);

    // Receive the server's response:
    varread = read(sock, server_message, 1024);

    printf("Server's response: %s\n",server_message);

    // Close the socket:
    close(sock);

    return 0;

}
```