Project 1

(一) Server

1. Connection process

   Socket -> bind -> listen -> accept -> authentication -> reply client message

2. Code (Authentication)

   a. Socket

      - Int socket(int domain , int type, int protocol)

        socket(AF_INET , SOCK_STREAM , 0);

        address family is set to be AF_INET (IP connection)。SOCK_STREAM connect with TCP。No socket is created if the return value is -1.

      - setsockopt(socket_desc, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

        To avoid the regular process is not closed properly (unable to bind the same port), use setsockopt() and SO_REUSEADDR to reuse the PORT.

      - server.sin_family = AF_INET;

        Set Address family to be TCP

      - server.sin_addr.s_addr = inet_addr(IP);

        Set IP to be local address 127.0.0.1

      - server.sin_port = htons(PORT)

        Set PORT 5566;

   b. Bind

      - bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0

        int bind(int sockfd, struct sockaddr *my_addr, int addrlen);

        Bind to the IP and Port set in socket. If return value <0 , fail to bind

   c. Listen

      - listen(socket_desc , 3);

        int listen(int sockfd, int backlog);

        Set the maximum waiting connection queue number to be 3

   d. Accept

      - client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);

        int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

        Accept the connection without returning value -1

   e. Authentication

      - Send

        Send message to Client

      - Receive

Receive PWD and username from client

- Strcmp

  Compare the string retrieve from client message with the username and pwd

f. Reply message

- Receive

  Receive message from client

- Send

  Reply with the same message that client enters

(二) Client

1. Connection process

   Socket -> connect ->authentication -> send message -> close socket

2. Code(Authentication)

   a. Socket : Similar to server setting IP , TCP and PORT

   b. Connect

   - connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0

     int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

     Check the connection with server, return zero if success

   c. Authentication

   - Send the username and pwd to server for authentication. Re-try if username or pwd is incorrect

   d. Send message

   - Send server any message beside ‘q’. ‘q’ for close connection
   - To receive the exact message sent by client

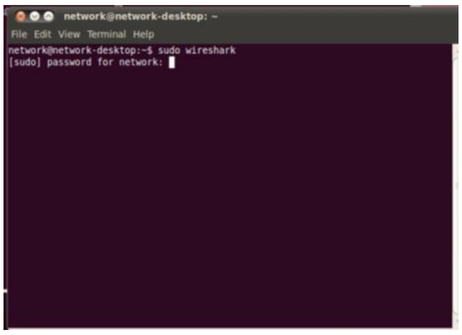
   e. close socket

   Terminate the connection with server

(三) WireShark analysis

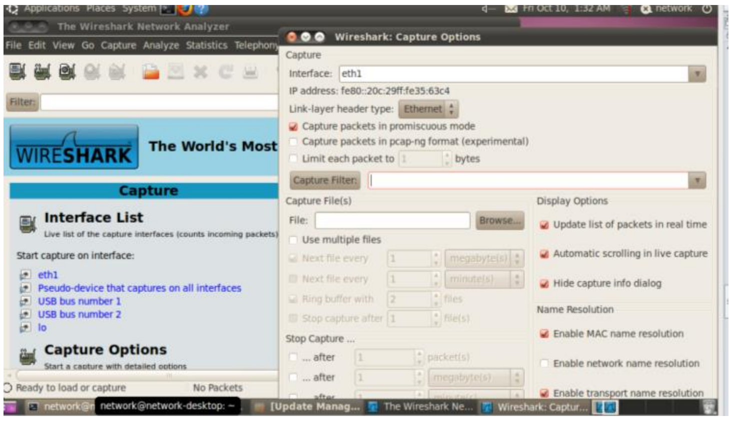
a. How to extract client credential

Step1 : open wireshark with sudo



```
network@network-desktop:~$ sudo wireshark
[sudo] password for network: █
```

Step2 : open wireshark and click “option”



Wireshark: Capture Options

Capture
Interface: eth1
IP address: fe80::20c:29ff:fe35:63c4
Link-layer header type: Ethernet
[✓] Capture packets in promiscuous mode
[ ] Capture packets in pcap-ng format (experimental)
[ ] Limit each packet to 1 bytes
Capture Filter:

Capture File(s)
File:
[ ] Use multiple files

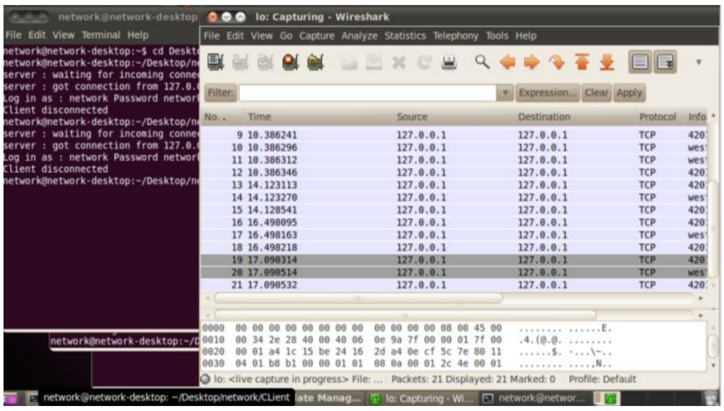
Display Options
[✓] Update list of packets in real time
[✓] Automatic scrolling in live capture
[✓] Hide capture info dialog

Name Resolution
[✓] Enable MAC name resolution
[ ] Enable network name resolution
[✓] Enable transport name resolution

Interface List
Live list of the capture interfaces (counts incoming packets)
Start capture on interface:
- eth1
- Pseudo-device that captures on all interfaces
- USB bus number 1
- USB bus number 2
- lo

Capture Options
Start a capture with detailed options

Step 3: choose IO and start the communication between server and client



```
network@network-desktop:~$ cd Deskt
network@network-desktop:~/Desktop/n
server : waiting for incoming conne
server : got connection from 127.0.
Log in as : network Password networ
Client disconnected
network@network-desktop:~/Desktop/n
server : waiting for incoming conne
server : got connection from 127.0.
Log in as : network Password networ
Client disconnected
network@network-desktop:~/Desktop/n
```

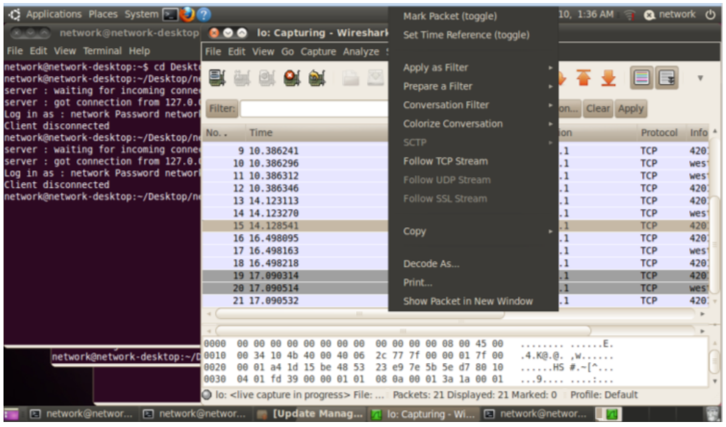
lo: Capturing - Wireshark

```
No. .  Time        Source      Destination  Protocol  Info
   9 10.386241   127.0.0.1   127.0.0.1    TCP       420
  10 10.386296   127.0.0.1   127.0.0.1    TCP       wes
  11 10.386312   127.0.0.1   127.0.0.1    TCP       wes
  12 10.386346   127.0.0.1   127.0.0.1    TCP       420
  13 14.123113   127.0.0.1   127.0.0.1    TCP       420
  14 14.123270   127.0.0.1   127.0.0.1    TCP       wes
  15 14.128541   127.0.0.1   127.0.0.1    TCP       420
  16 16.498095   127.0.0.1   127.0.0.1    TCP       420
  17 16.498163   127.0.0.1   127.0.0.1    TCP       wes
  18 16.498218   127.0.0.1   127.0.0.1    TCP       420
  19 17.090314   127.0.0.1   127.0.0.1    TCP       420
  20 17.090514   127.0.0.1   127.0.0.1    TCP       wes
  21 17.090532   127.0.0.1   127.0.0.1    TCP       420

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 34 2e 28 40 00 40 06  0e 9a 7f 00 00 01 7f 00   .4.(@.@. ........
0020  00 01 a4 1c 15 be 24 16  2d a4 0e cf 5c 7e 80 11   ......$. -...\~..
0030  04 01 b8 b1 00 00 01 01  08 0a 00 01 2c 4e 00 01   ........ ....,N..
```

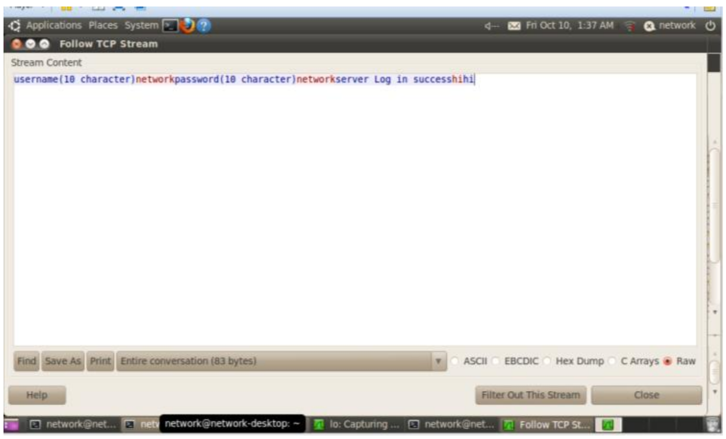
lo: <live capture in progress> File: ... Packets: 21 Displayed: 21 Marked: 0 Profile: Default

Step 4: observe some data transmitted between 127.0.0.1



```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 34 10 4b 40 00 40 06  2c 77 7f 00 00 01 7f 00   .4.K@.@. ,w......
0020  00 01 a4 1d 15 be 48 53  23 e9 7e 5b 5e d7 80 10   ......HS #.~[^...
0030  04 01 fd 39 00 00 01 01  08 0a 00 01 3a 1a 00 01   ...9.... ....:...
```

Step5 : click follow TCP stream and the transmitted data are shown



```
username(10 character)networkpassword(10 character)networkserver Log in successhihi
```

All the message transmitted are shown

b. How to solve the problem

- My solution : Simply encrypt the data, for example shift by 5 in client side, and shift the data back when decoding in server side
- Current Protocol solution:

(1) Encryption : both server and client constructs and exchanges cryptographic keys, and the whole process assures that only two endpoints know the key