Lab 3 – Client-Server with Minimum Security (for TA version)

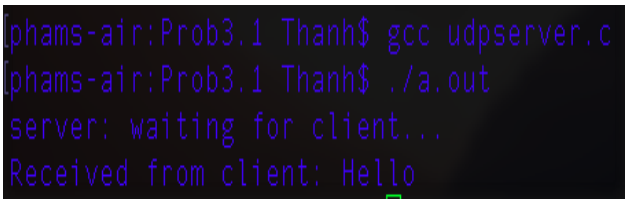
1. The purpose of the lab

In this laboratory, you will implement a simple UDP-based client-server system with limited security features (password verification and message integrity check – SHA1). This laboratory has two objectives:

(i) help you get familiarized with UNIX socket programming;
(ii) prepare you to tackle the next laboratory assignment which involves the design of a more secure client server system.

2. Assignment 1: Simple UDP socket programing (30 points)

In this assignment, you will be implementing a simple UDP client and server. The client sends a message to the client (the message can be whatever you want). The server after receiving the message from the client (which is "Hello" as in the example below) will sends a message "ACK" back to the client.



Client **Server**

A comprehensive tutorial on socket programming in C can be found at Beej's Guide to Network Programming Using Internet Sockets (a PDF file of the tutorial is available at the shared-folder). A simple implementation of UDP client and server can be found on the provided tutorial (Section 6.3 page 31-34) and at the folder **Sample_Code**. Basically, you can re-use most of the sample codes with some modifications according to the requirements of the assignment (**the server needs to send a message "Hello" back to the client**)

Note: **you need to run the two files for client and server on two different terminals**. When both client and server run on the same computer (host), the server's IP address is 127.0.0.1 (localhost).

Compile the sample codes (you need to run the server first, client later)

SunOS:

- Server: $ gcc udpserver.c -lsocket -lnsl
  $ ./a.out
- Client: $ gcc udpclient.c -lsocket -lnsl
  $ ./a.out 127.0.0.1 Hello

MacOS:

- Server: $ gcc udpserver.c
  $ ./a.out
- Client: $ gcc udpclient.c
  $ ./a.out 127.0.0.1 Hello

- Provided Code Skeleton: **Provided_Code/Prob3.1**
  - Client: udpclient/udpclient.c
  - Server: udpserver/udpserver.c

3. Assignment 2: Secured Client/Server Implementation (70 points)

In this assignment, we will be implementing a more complicated UPD client and server with simple security features.

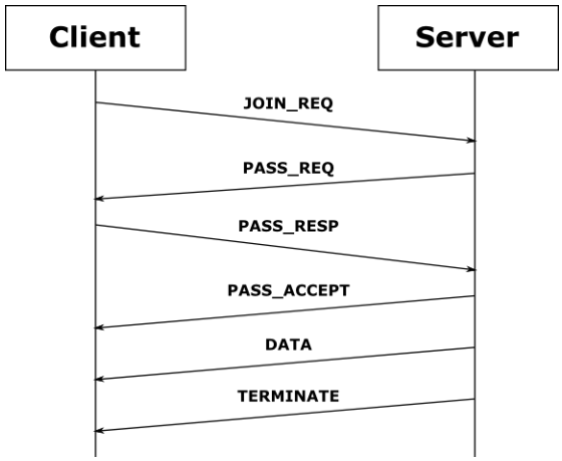


Figure 1: Interaction between Client-Server in the Simple Secured Protocol

3.1. Protocol specification

3.1.1 Requirements

There are two pieces of code you have to implement in two different files -- a client and a server. Below is the protocol specification which will give you the details you need to implement these two programs. Also, we are giving you the packet format, which specifies the content of the messages the client and server will exchange.

The specification of the protocol is shown in Figure 1 and the structure of the UDP-packets are shown in Figure 2. The communication between the client-program and the server-program are described with following scenario:

1. The client sends a JOIN_REQ packet to initiate communication with the server.
2. The server responds with a PASS_REQ packet, which is a password request to the user.
3. The client will send a PASS_RESP packet to the server which includes the password.
4. The server will verify the password and in case the password is correct, the server will send a PASS_ACCEPT packet to the client. After the third time the server sends a REJECT message to the client. The client closes the session, and the server exits as well.
5. Once the server transmits the PASS_ACCEPT packet to the client, the server begins transmitting the file using DATA packets. The file is broken into several segments (depending on the size of the file), and each segment is transmitted using a DATA packet.

6. When the server completes sending the file, it will transmit a TERMINATE packet which marks the end of the file download. This packet includes a file digest (SHA1 digest) that the client will use to verify the integrity of the received file.

Assumptions.

You may make the following assumptions to simplify the design:

(i) The server handles only one client at a time. There is no need to address issues associated with supporting multiple simultaneous clients. You will not need to use select/threads in this assignment.

(ii) Dealing with losses: We are using UDP-based data communication. While packet losses are possible with UDP, they are rare in a LAN environment such as the one where your code will be running on, and you will in all likelihood not encounter packet loss. There is no need for your code to implement any mechanism such as ACK/retransmission for reliable data delivery.

3.1.2 Packet formats

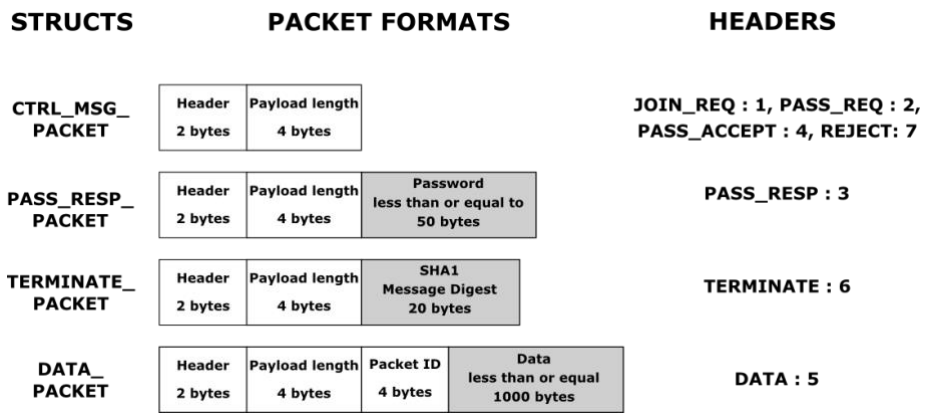


Figure 2: Specs of packets. Note that "payload length" refers to "Data" only (shaded part), if there is no data, its value is zero

3. 2 Sample codes and program for testing

We are providing the binary files of our implementation of the client and server (**Provided_Code/Prob3.2/Binary**). These files will be used as a reference to test your client and your server code. You need to ensure that your client and server programs work with our client and server implementations. Download the binary files for client and server according to your operating system. You may need to gain execute access permission before being able to execute the binary files by using **chmod +x** (See Lab1). In the following, binary files for MacOS are used for demonstration.

First, you need to run the UDP server by executing the command below.

```
$./udpserver_mac <server port> <password> <input file>
```

The server port is the port which the application will listen, and the password argument is the server password which you can choose by yourself. The input file represents the location of the data that should be transmitted when the connection is established with a correct password.

You execute the client with the command below. The first two arguments specify the location of the server (IP address) and what port to use.

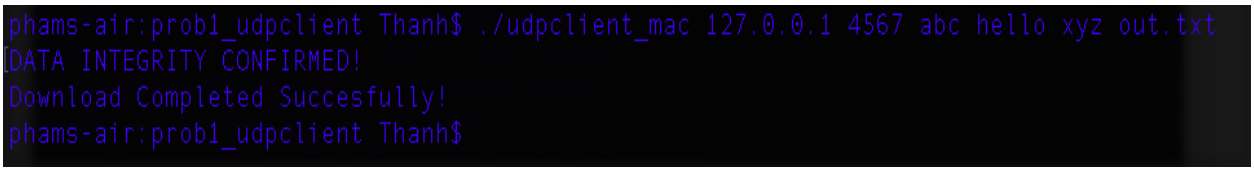
```
$./udpclient_mac <server IP address> <server port> <clientpwd1> <clientpwd2>
<clientpwd3> <output file>
```

The three passwords correspond to the passwords used in each of the three attempts the client uses to login. Note that once a correct password is transmitted, no further login attempts are needed, and the remaining password entries should be ignored. The output file argument is the location where the function should output its received data.

You should get the following results

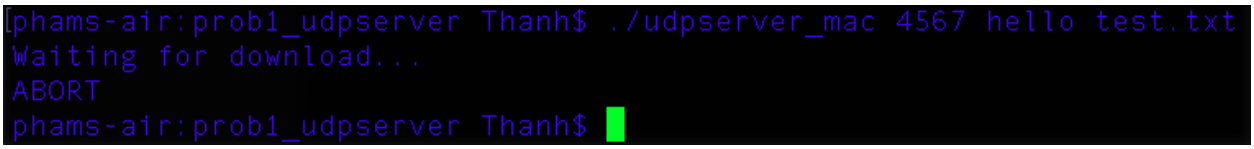
Case 1: One of the three passwords from the client is correct and the file (test.txt) is successfully downloaded from the server to the client

Server

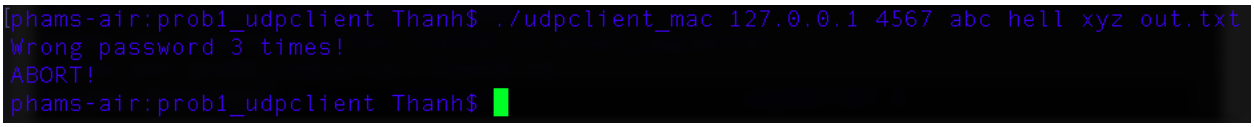


Client

Case 2: All of the three passwords are not correct.



Server



Client

3.3: Secured Client/Server Implementation

After you have comprehended the protocol, you should implement the client and server programs. To verify the functional procedures, both programs must printout the messages "OK" or "ABORT" depending on whether your application finishes correctly or terminates unexpectedly. In other words:

1. Print the message "OK" if your application finishes correctly. This is the case when the server is able to completely send the file to the client and the digest sent by the server matches the digest of the file received by the client.

2. Print the message "ABORT" if there occurs an error. Two examples of erroneous situations are:

(a) The digest of the file the client receives differs from the digest sent by the server.

(b) The client or server receives an unexpected packet.

Assignment

1. Implement the client program so it replicates the attached binary version. In the typed report elaborate on significant implementation details.
2. Implement the server program so it replicates the attached binary version. In the typed report elaborate on significant implementation details.

- Code skeleton: **Provided_Code/Prob3.2/Skeleton**
  - Client: udpclient/udpclient.c
  - Server: udpserver/udpserver.c
- Compile
  - MacOS: $ gcc -o udpclient udpclient.c -lssl -lcrypto -L/usr/local/opt/openssl/lib -I/usr/local/opt/openssl/include
  - SunOS: $ gcc -o udpclient udpclient.c -lsocket -lnsl -lssl -lcrypto

4. Report Preparation

The report should contain

(1) Report of assignment 1, copy and paste the screen to verify your work
(2) Report of assignment 2, copy and paste the screen to verify your work, make sure to have the elaboration on the implementation details, as required.
(3) The code files of the two assigments (submitted to TA via email)
  - Name the file as <Year.Month.Date><Student_ID><file_name>
  - For <file_name>: **udpclient.c** and **udpserver.c** for Assignment 1
    : **s_udpclient.c** and **s_updserver.c** for Assignment 2