CS 6500 – Network Security
Prof. Manikantan Srinivasan
Jan.-May 2021
Lab 4: Simple SSH Server and Client
Due Date:16th April 11:59pm . On-Line Submission via moodle
Extension: 15 % penalty for each 24-hr period; Max. of 48-hrs past the original deadline

1 Assignment Description

The objective of this task is to implement the functionality of an SSH client and an SSH server using the socket interface and needed crytographic routines. This is to help familiarize yourself with the working of SSH which is being used extensively. You may use C/C++, Python or Java.

There are two major components to the software: (i) the SSH server, and (ii) the SSH client. The communication will be based on TCP sockets.

1.1 SSH Server

The purpose of the SSH server is to respond to the commands sent by the SSH client. The server is expected to be up and running on the remote machine, listening to a specific port.

The server software contains these major components:

(i) NETWORK INTERFACE: This is responsible for processing the request from client, and passing it to the COMMAND PROCESSOR.

(ii) COMMAND PROCESSOR: This is responsible for processing the client's command, running appropriate system commands and then passing the generated output to the NETWORK INTERFACE.

When the server is started, it creates its pair of public and private keys and stores in a default directory called `serverkeys` as `serverpub.txt` and `serverpriv.txt`.

The server contains one more directory called `UserCredentials` which contains a file for every user. Each file is named with the respective user name such as `username.txt`. These files contain the username in first line and password in the second line for the respective user. When a user connects to the SSH server via SSH client, the client is authenticated using the corresponding username file. The username is an 8-character string (where, the characters are chosen from the set lowercase characters 'a'–'z' and '0' – '9').

The stored password contains a 64-bit salt and the encrypted output of the value **0..0** (16 bytes, i.e. 128 bits) using AES-128-CBC with the salt as the IV for CBC; this output serves as the 128-bit key derived from the passphrase. The 64-bit salt is extended to 128-bits by adding zeros in the least significant bits (i.e. left-shifting the salt value). These two values will be stored using Base64 encoding in the password file, along with the 8-character username.

For a given user, the salt is randomly generated at the time of password storage in the file. The passphrase is generated offline and the encrypted password is stored in the file, for each user. When you run the program, you are expected to know the passphrase for all users.

The server emulates some of the common Linux commands. The commands that the server will support are as follows. For each command received from the client, the server will print the command name and the parameters in the Server's Terminal window.

- **LS**: The server will send the list of all files present in the current directory to the client, which will then print the message.

- **PWD**: The server will send the current working directory to the client, which then print the message.
- **CD** `absolutepath`: The server will change from the current directory to the specified directory in the absolute path.
- **CP** `src dest`: The server will copy the file from the mentioned source to the destination with all it contents.
- **MV** `filename src dest`: The server will move the file from the mentioned source to the destination with all it contents.

1.2 SSH Client

An end user will be running the SSH client to communicate with the SSH server. The client software can be partitioned into these components:

(i) USER INPUT INTERFACE: The user-input interface accepts user commands, process them, and passes appropriate data to the network interface.

(ii) NETWORK INTERFACE: The network interface is responsible for establishing the required socket with the SSH server. It is responsible for accepting the data provided by the USER INPUT INTERFACE above and transmitting it over the socket. It is responsible for reading the responses transmitted by the server, and either storing the data on file locally, or displaying it on the screen.

The client program will wait at the prompt:
Main>

At the main prompt, the client types one of the following commands:

```
ssh <IPADDR> <PORT> <USER>
ssh <SYSNAME> <PORT> <USER>
ssh <PORT> <USER>
```

Here, the IP Address/System name and port number are provided if the SSH server is running on a different machine; and only the port number, if the SSH server is running on the same machine as the SSH Client. The USER parameter specifies the username.

As part of processing the command above at main prompt, the following tasks should take place, in the background, in a sequential manner.

- The client will initiate the connection to the SSH server.
- The server will send its public key (base64 coded) to the client; This should then be stored in the local directory of the SSH client as *server_pub.txt*.
- The client will create a randomly chosen 256-bit AES session key.
- The client will encrypt {username, passphrase, sessionkey} with the server's public key and send it to the server.
- The server decrypts the above message with its private key and compares the username and passphrase in the message with the locally stored credentials of the client. The passphrase to 128-bit password conversion is as explained above.

Then, the SSH server sends the acknowledgment as **OK** if client is successfully authenticated; otherwise, it sends **NOK**.

If authentication is successful, further communication between client and server will be encrypted (AES in CCM mode) with the 256-bit session key as mentioned above; otherwise, the client program returns to the main prompt.

After successful authentication, the client software interface will accept any one of the following commands:

- listfiles: The client will send the message `LS` to the server, and print the server's response.
- cwd: The client will send the message `PWD` to the server, and print the server's response.
- chgdir absolutepath: The client will send the message `CD absolutepath` to the server, and print the server's response.
- cp filename src dest: The client will send the message `CP filename src dest` to the server, and print the server's response.
- mv filename src dest: The client will send the message `MV filename src dest` to the server, and print the server's response.

2 Sample Session

Assume that you have created the files sshclient.c and sshserver.c and the corresponding executables in your LAB4 directory. Please use a suitable unique port number for your server.

There will be two cases that you will test for: (i) server and client running on the same machine. You can invoke the client as: sshclient localhost $<$ *port* $>$; (ii) server and client running on different machines.

You can assume that all users have access to all files on the SSH server's file system (located under your home directory on that server).

```
Login to your account on a DCF machine (say dcf15) or a VM on your system
DCF15> cd LAB4
 ./sshserver 25678    -- Server is running
Login in to another DCF machine, say, dcf19 (or another VM)
DCF19> cd LAB4
DCF19 ./sshclient dcf15 25678 raman -- Client is running on another host
.... Server's Initial Response \hspace{0.5cm} -- OK (or) NOK
.... If OK is received by the Client, then the user-client program ...
Client-Prompt> listfiles

.... Server's Response

Client-Prompt> cwd
.... Server's Response
Client-Prompt> chgdir <absolutepath>
.... (DONE)
Client-Prompt> cp <filename> <source> <destination>
.... (DONE)
```

```
Client-Prompt> mv <filename> <source> <destination>
.... (DONE)
Client-Prompt> logout
....
DCF19>
```

3 What to Submit

Name your project directory as LAB4 (Note: ALL UPPERCASE). Once you are ready to submit, change directory to the directory above LAB4, and tar all files in the directory with the command:

```
tar czf Lab4-RollNo.tgz LAB4
```

The directory should contain the following files:

- Source Files
- Makefile
  Typing command 'make' at the UNIX command prompt, should generate all the required executables.
- a README file containing instructions to compile, run and test your program. The README should document known error cases and weaknesses with the program.
- a COMMENTS file which describes your experience with the project, suggestions for change, and anything else you may wish to say regarding this project. This is your opportunity for feedback, and will be very helpful.

4 Help

1. WARNING ABOUT ACADEMIC DISHONESTY: Do not share or discuss your work with anyone else. The work YOU submit SHOULD be the result of YOUR efforts. Any violation of this policy will result in an automatic ZERO on the project, a potential F in the course, and other academic action.
2. Ask questions EARLY. Do not wait until the week before. This project is quite time-consuming.
3. Implement the solutions, step by step. Trying to write the entire program in one shot, and compiling the program will lead to frustration, more than anything else.
4. Questions raised within 24 hours of the deadline will not be answered; you have to make your own assumptions and justify them.

5 Grading

- Server: 40 points
- Client: 40 points
- Demo using Wireshark (running on the server): 15 points
- Viva voce: 5 points

No README/COMMENTS: -5 points; Incomplete Compilation: -10 points