Homework #3
CS468 Sscure Programming and Systems Fall 2021

This homework is due on 11/6/2021. This is an individual homework assignment to be done by each student individually. The submission of your homework is your acknowledgement of the honor code statement

I have not copied any solution from anyone and not provided any solution to anyone on this assignment. The solution has been entirely worked out by me and represents my individual effort.

Please

- Use a word processor (MS-Word, LaTex, Framemaker, …) for your solutions. No hand writing submission will be accepted.
- Include your name and G-number at the beginning of your homework submission.
- Pack all your files with zip or tar and gzip and name your packed file as CS468_HW3_<your last name>_<your G#>.zip (or tar.gz)
- Submit your packed solution to the blackboard

This homework requires accessing, using and programming at zeus.vse.gmu.edu. Please create a directory named "CS468_HW3_<your last name>_<your G#> and do everything of this HW there.

1. Answer the following short questions (20 points)

    1) What are the differences between little endian and big endian store of multi-byte data? (5 points)

    2) What is the network byte order representation (in hex format) of little endian number 0x87654321? (5 points)

    3) What is the IP address (in dotted decimal format) of www.gmu.edu? (4 points)

    4) What is the dotted decimal representation of IP address? What's the use of dotted decimal representation of IP address? (6 points)

2. Implementing a primitive authenticated remote shell client and server (60 points)

    The provided `SimpleRShellClient.c` and `SimpleRShellServer.c` are simple remote shell client and server based on TCP. It allows one to use `SimpleRShellClient` to execute shell

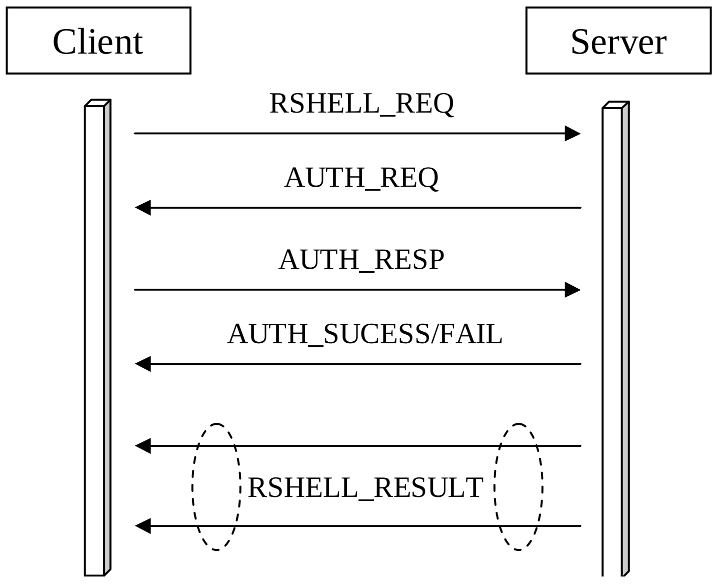

Figure 1: Message Interaction between RShellClient1 & RShellServer1

commands on a remote machine running `SimpleRShellServer` and get execution results back. However, they do not have any authentication whatsoever.

You are asked to improve the above client and server by implementing a primitive authentication based on user ID and password. Specifically, you need to develop two C programs: `RShellClient1.c` and `RShellServer1.c` (you can use `SimpleRShellClient.c` and `SimpleRShellServer.c` as starting point) such that

- `RShellServer1 <port number>  <password file>` will listen on the specified <port number> and authenticate the remote shell command by comparing the SHA1 hash of the password provided by the client with that in the <password file>. If the authentication is successful, execute the shell command and return the execution result back to client. The <password file> should contain one or more line: <ID string>; <hex of SHA1(PW)>
- `RShellClient1 <server IP> <server port number> <ID> <password>` will read shell command from the standard input (i.e., the keyboard) and send the shell command to the server listening at the <server port number> on <server IP>. It will provide the <ID> and the SHA1 hash of <password> to the server for authentication.

Protocol Specification

In order to have the functionalities described above, you need to implement the following protocol (illustrated in Figure 1) upon TCP between the client and server:

1. The client sends a `RSHELL_REQ` message that includes the client's ID and the shell command to the server.
2. The server responds with an `AUTH_REQ` message the client – asking the client to provide the password.
3. The client sends an `AUTH_RESP` message to the server – responding with the SHA1 hash of its password.

Message Format	Message Type 1 byte	Payload Length 2 bytes	Payload (variable length)		
RSHELL_REQ	0x01	Payload Length	ID	Shell Command	
AUTH_REQ	0x02	Payload Length	ID		
AUTH_RESP	0x03	Payload Length	ID	SHA1(PW)	
AUTH_SUCESS	0x04	Payload Length	ID		
AUTH_FAIL	0x05	Payload Length	ID		
RSHELL_RESULT	0x06	Payload Length	ID	MR	Execution Result

Figure 2: Message format of all the messages between RShellClient1 and RShellServer1

4. The server verifies the received SHA1 hash with the SHA1 hash of the password in the <password file>. If it does not match, the server sends an `AUTH_FAIL` message to the client. Otherwise, a) the server sends an `AUTH_SUCCESS` message to the client; b) the server executes the shell command from the client and sends the result back to the client via `RSHELL_RESULT` message(s).
5. If the client receives `AUTH_FAIL` message from the server, it should print out an error message "Authentication failed" and exit gracefully. If the client receives `AUTH_SUCESS` message, it should print out the shell command result received from `RSHELL_RESULT` message(s). Here the server may send one or multiple `RSHELL_RESULT` messages depending on the volume of the result. The client should be able to handle multiple `RSHELL_RESULT` messages.

Note, the client and server could repeat the above steps. The sever should authenticate the client for the first shell command request and remember the authentication status of the client ID for 60 seconds. In other words, the server should not ask the client for authentication if it has been successfully authenticated within past 60 seconds.

Message Format

Figure 2 illustrates the format of all the messages of the protocol. Each message starts with the 1-byte type field, then the 2-byte payload length field that contains the number of bytes of the payload that is right after. The payload is of variable length depending on the message type.

The ID field is of fixed length of 16 bytes. It contains a string of maximum length 15 bytes that denotes the user ID (e.g., Alice). The ID string is null terminated.

For message RSHELL_REQ, the “Shell Command” is simply the shell command the client is trying to execute at the remote host (e.g., “pwd”). It is null terminated string.

For message AUTH_RESP, the SHA1(PW) is the 20 byte (160 bits) SHA1 hash of the ID’s password. Note the SHA1(PW) hash is random binary value, you should treat it as raw byte stream.

For message RSHELL_RESULT, MR (more result) is a 1-byte field representing if there is any more execution result after the current RSHELL_RESULT message. If there is no more execution result, MR field should be zero. Otherwise, WR should be non-zero. The “Execution Result” is simply the execution result of the shell command at the remote host. Normally it should be printable ascii string. But it could contain unprintable char (e.g., if you trying to display a binary file). You want to make sure to print every byte in the “Execution Result” even if there is null in the middle.

Experiments:

1) create a text mode password file named `passwdfile.txt` that contains the following line:

```
Alice; <hex representation of SHA1(PW)>
```

   where “Alice” is a recognized ID, and the PW is “SecretPW”. Note you can obtain the SHA1 of “SecretPW” via appropriate OpenSSL command.

Take a screen shot or log (via script command) of the execution of the following commands:

2) run `./RShellServer1 <port num> passwdfile.txt` Here you want to choose some random port num between [1025, 65535]. If someone else is using the port num you picked, your server won’t be able to bind it. You can try some other port number.

3) run `./RShellClient1 localhost <port num> InvalidID SecretPW` The server should reject any shell command it sends as InvalidID is not defined in the passwdfile.txt

4) run `./RShellClient1 localhost <port num> Alice WrongPW` The server should reject any shell command it sends as the SHA1 hash of WrongPW does not match that in passwdfile.txt

5) run `./RShellClient1 localhost <port num> Alice SecretPW` and type the following shell commands:

```
1) id
2) date
3) pwd
4) ls -l
```

   the server should accept and execute any shell command it sends and return the execution result to the client. The client should print out the execution result on screen.

Name the screenshot or lof file “`CS468-HW3-1.*`” (here * depends on the format of your screenshot, e.g., jpg).

Submit:

- RShellClient1.c, RShellServer1.c, passwdfile.txt, corresponding makefiles and/or information about how to generate the executable from the source code. Note, you need to provide everything needed to generate the executable in zeus environment.
- `CS468-HW3-1.*`

3. List and explain 3 security vulnerabilities of the above authentication protocol (20 points)