Abstract

netcat_part very simply, runs a client that sends data to a listening server. The client may either send a message or a file to the server. Anything that the server receives, it means to store it in a file on its file system. When sending a file, the user at client-side has the option of specifying only a particular number of bytes to be sent to the server; the user may wish to read the file from a certain position from the beginning of the file (offset) along with the number of bytes from file that need to be read and sent to the server.

Another aspect of the program is to maintain message integrity. In order to check whether the same file that is being by client is in fact the file that is received by the server, an HMAC algorithm is used. A ***digest*** is generated at client-end using ***client data*** and a ***shared key*** that is known to (or resides) both client and server. The digest along with the client data is sent to server. At the server-end, the server checks the client-data received by running the HMAC algorithm on the **data** and ***server's own shared key***. The digest generated is compared with the digest received from client, if the two match, **message integrity** is maintained.

Files submitted

- *netcat_part.c*

  This file is mainly responsible to parse the user's arguments at command line along with the executable. So, for example, if the user types a command,

  $. /netcat_part –l localhost results.txt –p 20000

  The ***parse_args()*** function sifts through each argument, like those after the dashes (e.g. l, p, etc.). Accordingly the server's and the client's basic parameters like IP addresses, port numbers, etc. are parsed and stored. Another key function of the this function is to check for sanity i.e. to check that the user inputs the correct arguments.

  This file also hosts the ***main()*** function which constructs both server and client for our client-server environment.

- *server.c*

  This file helps populate the server's basic parameters like the port numbers that it needs to be listening on for incoming client connections, its IP address information, etc.

  This file helps supplement communication with client by performing the following tasks:
  - Create a listening socket
  - Bind a port number to the listening socket
  - Listen for incoming client communication
  - Accept client connection and create a new socket to exchange data with client
  - Close socket and thus connection once data communication is wrapped up.

- *client.c*

This file helps client creation by supplementing communication with server performing the following tasks:

  - Create a client socket
  - Initiate connection with server by connecting to server's welcoming-socket
  - Exchange data with server
  - Close socket and thus connection once data communication is wrapped up.

- *nc_args_t.h*

  This is a header file containing a structure *nc_args_t* that stores parameters entered by user on command line

- *prompt_error.h*

  This is a header file containing an error prompting function *promptError()* in cases where something wrong is sensed.

- *shared_key.h*

  A header file containing the shared key that is known to both client and server.

- *Makefile*

  Executes the program

- *README*

  Other helpful information about program execution and implementation.

Difficulties faced

- Could not successfully implement the security feature with HMAC
- Tried it for the part when user can send a message from client-side
  - used a delimiter to stick the message with a 40-byte hash value generated after using the HMAC function
  - the message part is separated from the client's digest at the server end; both parts are stored - message part is stored in a string and client's digest is stored in another; when the message is passed through HMAC to generate a server digest, the server's digest turned out different than the client's digest. Couldn't figure out why this was so
- I had to comment out the HMAC code as I was facing difficulties maintaining sanity of code to work at least for the message, file sending from client without the security feature
- I could demo the unsuccessful implementation to the Instructors if I may

Credits

1. Chapter 2, TCP/IP Sockets in C
2. http://www.linuxhowtos.org/C_C++/socket.htm
3. http://www.askyb.com/cpp/openssl-hmac-hasing-example-in-cpp/