horizontal line

Heterogeneous Connection Configuration Steps

**July 29th September 2022**

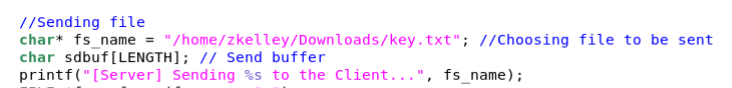
Zachary Schroeck

If you have any questions, please feel free to email me at ([zschroeck44@gmail.com](mailto:zschroeck44@gmail.com)).

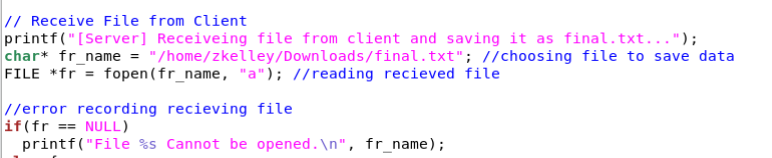
# Server

The server in this setup should have a file named key.txt with the key inside (ex: 123) it should also have a file named final.txt for the encrypted message to be saved too. These file directories are specified in the code server.c and server2.c. The file directories should be the directory you put them in, not the directory listed in the snips below.

**server.c**

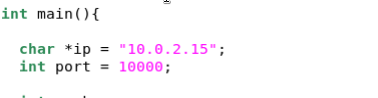


**server2.c**

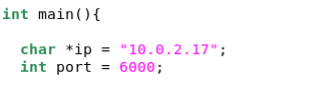


The server should also specify the IP of the socket it is connecting too and the port number. The IP address should match your client computer's IP addresses not the ones listed in the snips below.

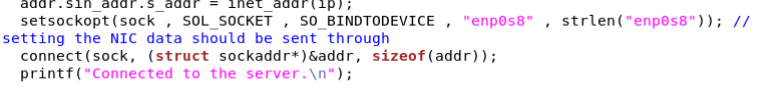
**server.c**



**server2.c**

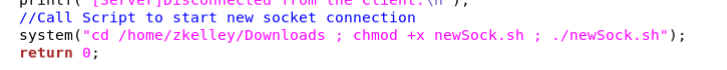


In the file server2.c the setsockopt command should specify the name of the NIC it is connecting to; this command should also be on client2.c. The NIC should match the second NIC on your client, not the ones listed in the snips below.



The server and client also have bash shell script files used to automate opening sockets and writing the files. The file locations of these scripts also need to be designated in the code. The file directories should be the directory you put them in, not the directory listed in the snips below.

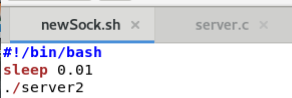
**server.c**



The only other thing on the server side that needs to be considered is the firewall, needing to punch a hole through it to allow the client TCP packets to transmit.

**newSock.sh**

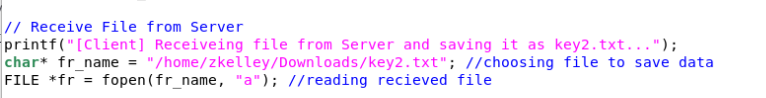
Both hosts will have a file called newSock.sh on them to begin their second socket creation but only the server side one has a specific time delay to allow for the client to start listening before it tries to connect. This time delay for testing was set to 1 second but can be reduced to .01 seconds.



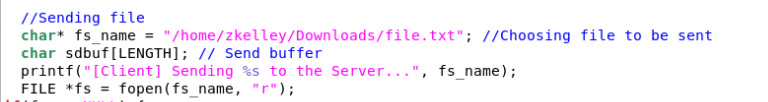
# Client

The client in this setup should have a file named key2.txt with no text inside it to receive the key, it should also have a file named file.txt with the message you want to transmit. These file directories are specified in the code client.c and client2.c.

**client.c**

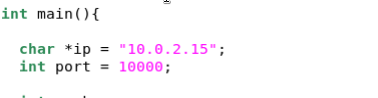


**client2.c**

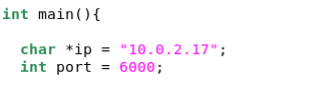


The client should also specify the IP of the socket it is binding to and the port number. The IP address should match your client computer's IP addresses not the ones listed in the snips below.

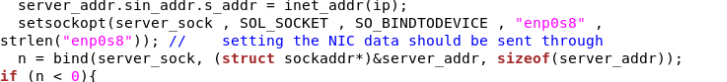
**client.c**



**client2.c**

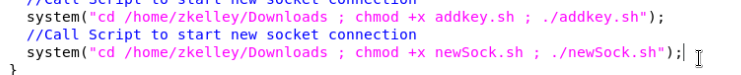


In the file client2.c the setsockopt command should specify the name of the NIC it is binding to; this command should also be on server2.c, this is listed above. The NIC should match the second NIC on your client, not the ones listed in the snips below.



The server and client also have bash shell script files used to automate opening sockets and writing the files. The file locations of these scripts also need to be designated in the code. The file directories should be the directory you put them in, not the directory listed in the snips below.

client.c



There are a few more things that might cause hiccups on the client side, make sure the firewall on both the client and the server is allowing TCP packets from each other. I also ran into security issues and sometimes had to run these three commands on the client to allow for the correct packets to be sent over their designated NICs.

sysctl -w net.ipv4.conf.all.accept\_local=1

sysctl -w net.ipv4.conf.all.rp\_filter=0

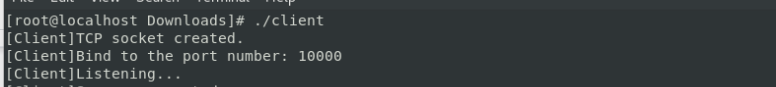
sysctl -w net.ipv4.conf.your\_nic.rp\_filter=0

**(your\_nic) being the NIC specified in client2.c and server2.c of the machine you are using.**

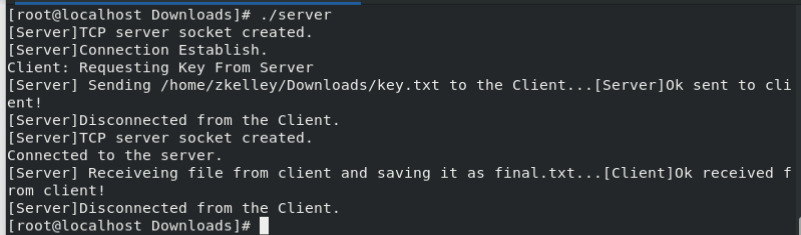
This should resolve an issue where packets are only directed over one NIC even if you are designating a specific interface.

# How to run code normally

1. Compile code using gcc compiler (gcc client.c -o client), make sure to do this for each program (server.c, server2.c, client.c, client2.c).
2. After compiling each code, on the client enter the command (./client) to start listening.



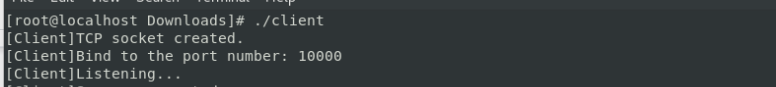
1. Then on the server enter the command (./server).



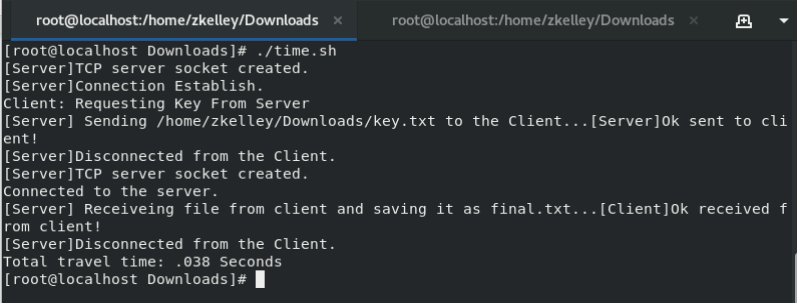
This is all that needs to be done; everything else is automated. If files and permissions are set up correctly you should now be able to see the key in the key2.txt file and it appended to the file.txt file on the client. You should also be able to see the full encrypted message in the final.txt file on the server. If you run into issues, make sure each host can ping one another and try entering commands listed above in the client section before running the programs.

# How to run code to get travel time for testing

1. Compile code using gcc compiler (gcc client.c -o client), make sure to do this for each program (server.c, server2.c, client.c, client2.c).
2. After compiling each code, on the client enter the command (./client) to start listening.



1. Then on the server enter the command (./time.sh) this will run the server-side code in the script but also keep track of the amount of time it takes for the code to run.



Make sure the time.sh file is in the correct directory with the rest of the files on the server.