

Connection Game

COMP454: Computer Networks

Dr. Ahmad Al Hajj

Done by:

Name                                          ID                              Section

Khaled Sardouk                          201901490             COMP 454

Mohammad Al-Tayyeb Soubra 201901076           COMP 454

Hadi Salloum                              201900774               COMP 454

Salah El Deen Stouhi                 201901265               COMP 454

Karim Hamod                             201900057               COMP 454

Mohammad Kreidieh                 201900799               COMP 454

Abstract:

We were tasked to implement a game which contains an attacker and defender, taking turns trying to gain a higher score. The attacker sends the message “ATTACK” while the defender tries to intercept this message and prevent it from reaching the client as is.  This led to us searching the web to implement such a game, or program. We searched for methods to know more about socket programming and how to connect a server and client. We also found out that LogMeIn Hamachi can be used to make a VPN and to establish a connection between the attacker and defender to test the code using 2 separate and far computers. We also searched for ways to protect the message sent by the attacker from the interceptions of the defender. This includes methods such as binary shift and NRZ. Moreover, we searched for error control algorithms such as Hamming Code. We took all that we gathered and implemented it in a python program using PyCharm. What we came up with was a code that could have been more optimal, but nonetheless it was a fully functional code that met all the requirements.

Roles of each side:

* The Server:

Establishes a tcp connection between two clients which will take turn is attacking and defending.

* The Attacker side:

The attacker sends the message “ATTACK” to the receiver who receives the message. If the message is received as “ATTACK” then the attacker wins. If the defender can intercept the message and change it, then the attacker loses. Unfortunately, the latter was the case as the defender was always intercepting the message and changing it. To change this result, we decided to implement counter measures to ensure that the attacker can win against the defender. We decided to use two counter measures, the first was NRZ, and the second was Caesarian Shift. The NRZ flips the ‘0’s to ‘1’s and ‘1’s to ‘0’s so that if the defender decides to use NRZ, he would be flipping the message back to its original form and the word “ATTACK” would remain intact.

The Shift method shifts the characters in their ASCII representation by -10 which changes the entire message and the binary representation of the word “ATTACK” so that when the defender shifts by 10, the word “ATTACK” remains the same. However, this way of implantation seemed to cause trouble as the consistent conversion from binary to ASCII sometimes led to the loss of bits. So, we decided to use a shift method but this time we directly applied it to the binary representation .The attacker chooses randomly between the two counter measures using a simple IF ELSE statement.

* The Defender Side:

 The defender has to receive the “ATTACK” message from the attacker, and try to change it so that “ATTACK” does not decode at the defender’s side.

There are 4 functions in defender.py file:

* NRZ(Binary): it takes binary number and flips the 0s and 1s
* ByteToBinary(Byte): it takes ASCII as input and converts it to a binary number
* BinaryToWord(Binary): it takes a binary number and converts it to a word
* WordToBinary(Word): it takes a string of characters and converts it to a binary number.

The defender side receives the message in ascii, then turns it to binary using ByteToBinary(), then uses NRZ strategy to flip the bits using NRZ(), and then changes the bits back to word BinaryToWord() and reads it.

That way the word “ATTACK” won't be decoded at the defender’s side.

In addition, if we wish to read the word “ATTACK”, we can use the same NRZ strategy to flip the binary bits again so that they return back to the original “ATTACK” message binary bits. Then convert the binary number to word and read it.

**Initial Setup for each Run:**

1-      First run server.py application

2-      Then run client.py application.

3-      Then run client.py application again.

You will have 2 command windows, one for the 1st client, and the other one for the 2nd client.

Both clients will take attacker/defender rounds.

The client that is the attacker for this round will automatically send the message “ATTACK”.

The client that is the defender for this round will encrypt it. Then decrypt it.

|  |  |
| --- | --- |
| Team | Members |
| Connection Team | ·         Khaled Sardouk  ·         Karim Hamod |
| Attacker Team | ·         Salah El Deen Stouhi  ·         Hadi Salloum |
| Defender Team | ·         Mohammad Kreidieh  ·         Mohammad Al Tayyeb Soubra |

1- Establishing the connection:

We used the socket library to create both the server and the client, in the server part we created a server socket and bind to the local host for testing first then to the ip address to the Hamachi room and to the port 1234, then we called the listen function of the socket which allows the server to accept new connections. After we finished creating the server socket, we moved to the game function. In this function the server will accept two connections, and in this function, we applied a primitive way to make a turn-based game which is calling the function while changing the parameters and we thought to make it a game which includes best of 3 rounds. In this part it wall call the clients connection function which will assign roles to the two players and enable communication between them after the 3 turn we will exit this functions and call the check score function which will tell the two clients to calculate the score and send it to the server to see who won and reply to the clients and close the connection to the player and a choice will prompt in the server side if we want to continue and it will reset the process.

For the client side:(we have 2 clients)

We first established the connection with the server by choosing a specific port to connect to and providing the host address, where we used those 2 in a function called connect of socket library. Then we made a variable Response which is equal to the data sent from the server.

Then we made a loop where the client keeps receiving the data that is sent from the 2nd client to the server then to him.

Then we check the response:

-If it is “you are the defender” we receive the 2nd response and decode it into ascii form then process it using the error detection functions we implemented, and then we scramble the response either by using NRZ or by shifting. This is chosen randomly by a choice variable. Then if  the result of the response after processing was “ATTACK” we decremented the defence score by -10 else we increment the defence score by 10 and send defence successfully to the server .

-If the Response is “you are the attacker”:

We start by initializing the message to be sent and encoding it in ascii form then we scramble the message by NRZ or by shifting also by variable choice at random. Then we send it to the server which will send it to the 2nd client. We also receive the response from the server if the attack failed or was successful. At the same time when the connection was established, we got the time of it and the time when we received the response. As a result, we get the RTT and determine if the RTT is more than that of the threshold RTT. If that is the case, we decremented the attack score by 20 and incremented the defence score of the defender by 20 and if the response was that defence failed, we incremented the attack score by 10 else we decremented it by 10

-At last, if the response was none of the above, we print the total score of each client and send it to the server which compares who has higher score and send it to each one if he wins or loses.

3- Error Control algorithm:

 We used the Hamming Code to detect errors and correct them. The hamming code works by calculating the redundant bits in the packet sent, we use the following formula to calculate the redundant bits, 2^r ≥ m + r + 1 where, r = redundant bit, m = data bit.

We then use either even parity or odd parity to determine the values of our redundant bits. If we use odd parity, we count the number of 1s that correspond to that redundant bit, if the count was even, we set the bit to 1 else it is 0. And vice versa for the even parity method.

To determine the set of bits to use to calculate the redundant bits, we look at the position of the other bits. For example, if we were to calculate R1 (redundant bit 1), we need to look at all the bits whose position in binary end in 1, like 1,3,5,7,9, etc. As for R2 we need to look at all the bits whose position in binary include a 1 in the second position from the LSB (Least Significant Bit).

After calculating the amount, position, and value of redundant bits we send the packet, and the receiver will already know the amount, position, and value of redundant bits used, so he can compare that with the packet he received and if there is any error it should be corrected by itself by using the correction function that we implemented. The only downside of this error control algorithm is that it can only correct one bit.

Hamming Code source:

<https://www.geeksforgeeks.org/hamming-code-implementation-in-python/>

<https://www.geeksforgeeks.org/hamming-code-in-computer-network/>

**Additional Features:**

The game can also be played by 2 players on 2 separate computers.

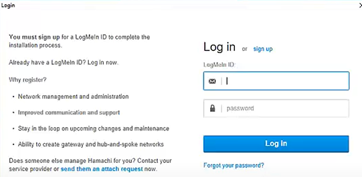
We used VPN Hamachi

**How to use Hamachi Login:**

1-     Turn it on



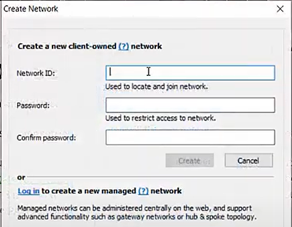
2-     Login with an email or signup.



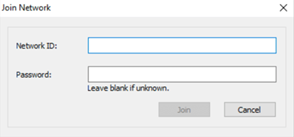
3-     Create a new network

****

4-     Name your network, give it a unique name because it will be your network ID.



5-     After creating the network, the other user should choose “Join an existing network” and fill in the network ID and password.



That process will put 2 people or more (up to 5) in a “room”. Hamachi room establishes a vpn connection between the people inside the room.

**The files setup:**

1- The user who will run the server should copy the IP address from Hamachi, and insert it in the server.py file and client.py.

2- The other player should take the same IP address and insert it in his client.py file.

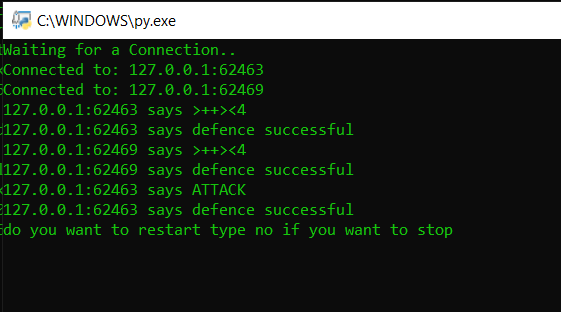
3- The host should run the server first, then both players may start client.py (the order of who starts client.py first does not matter.)

**We ran the game 3 times:**

First round:

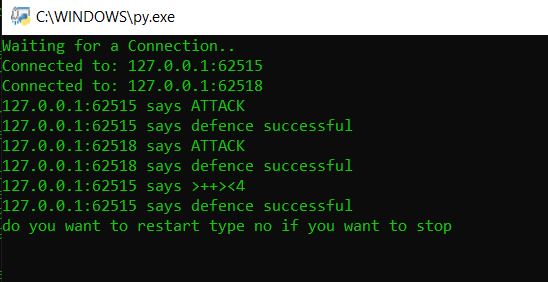
Client 1 & 2:



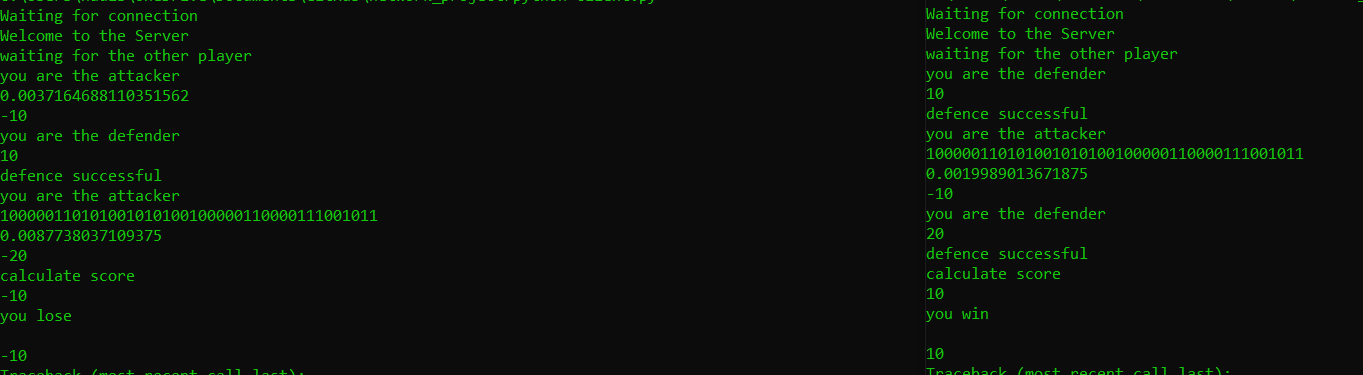
Serverside:

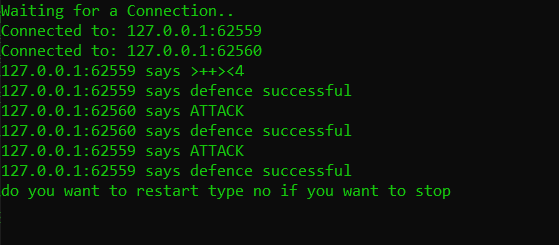
Second round:

Client 1 & 2:

Server side:

Third round:

Clients 1 & 2:

Server Side:

Conclusion:

The code that we implemented turned out to be fully functional. However, it could have been more optimized towards the attacker’s side as even when we tried to help the attacker gain an advantage on the defender, the latter more often than not ended up being the victor. This might have been optimized using reinforced learning and allowing the attacker to learn about the defender and how it works. However, we decided during implementation to prioritize the server connection and the defender interception as the message was already being sent from the attacker which meant that that aspect was operational. A good design requires a good compromise, and that is exactly the case in this project. Even though the attacker has very low chances of winning, we were still able to establish a connection between the client and server, the “ATTACK” message was as a result being sent from the attacker and received by the defender, the defender was able to  intercept the message and change its contents to achieve victory, error control was achieved using Hamming Code, and the ability to use LogMeIn Hamachi to run the code on two different and far devices is definitely a welcome addition.

And here is the GitHub repository in it detailed commit by each member and the code with the repport.

https://github.com/khaledAlsardouk/network\_project