A screenshot of a cell phone

Description automatically generated

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, take 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 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 algorithm such as Hamming Code. We took all that we gathered and implemented it in 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 connection between the attacker and the defender.

* 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 .The attacker chooses randomly between the two counter measures using a simple IF ELSE statement.

* The Defender Side:

Initial Setup for each Run:

1. First run server.exe application
2. Then run client.exe application.
3. Then run client.exe application again.

You will have 2 command windows, one for 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 sends the message “ATTACK”.

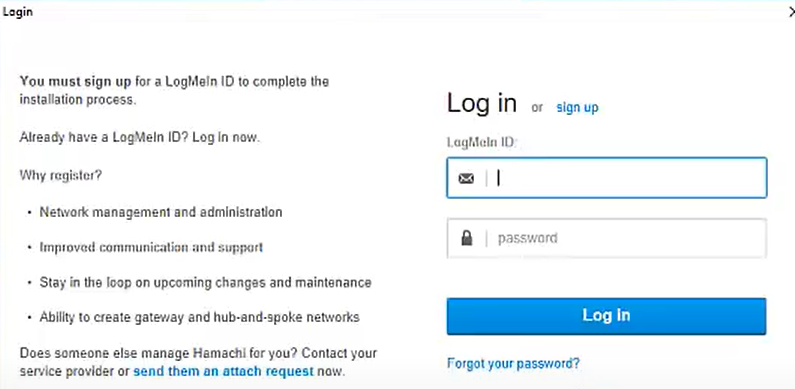
The client that is the defender for this round will encrypt it. Then decrypt it.

# How to use Hamachi:

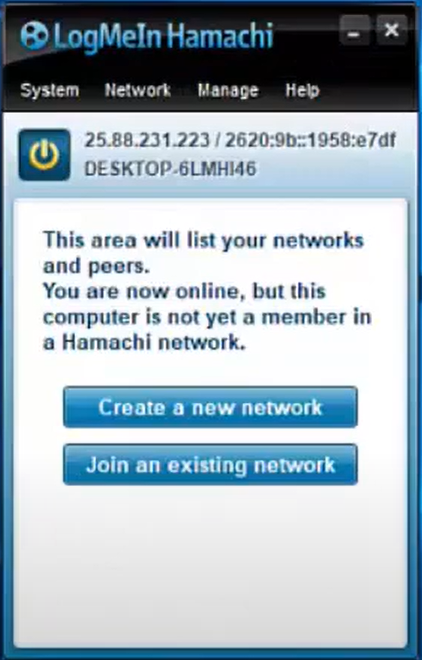
1. Turn it on



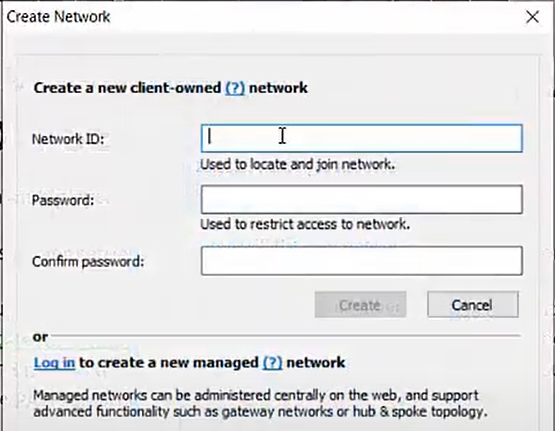
1. Login with an email or signup.



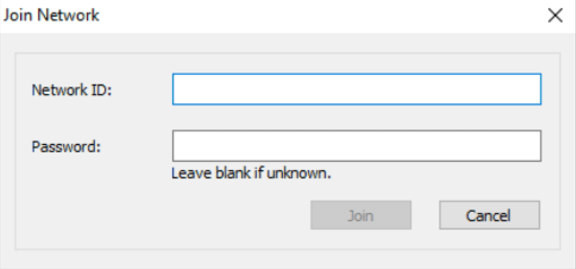
1. Create a new network



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



1. After creating the network, the other user should choose “Join an existing network” and fill with the network ID and password.



1. After that let the

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

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/>

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.