Chat Server Using Socket Programming

Venkat Narayan . G
Roll Number : 174266
Department of Electronics and Communication Engineering
National Institute of Technology Warangal

Rishabh Jain
Roll Number : 174245
Department of Electronics and Communication Engineering
National Institute of Technology Warangal

This paper provides a brief description of a basic chat server developed using python socket library. It provides insight on Caesar cipher and headers to provide security between chat and server devices.

Keywords—socket programming, chat server, chat client, cipher, encryption, decryption.

I. CHAT SERVER

Chat server is a standalone application that is made up the combination of two-application, server application (which runs on server side) and client application (which runs on client side). This application is using for chatting in LAN. To start chatting you must be connected with the server after that your message can broadcast to each and every client
Exchanging text messages in real time between two or more people logged into a particular instant messaging (IM) service. Instant messaging is more interactive than e-mail because messages are sent immediately, whereas e-mail can be queued up in a mail server for seconds or minutes. However, there are no elaborate page layout options in instant messaging as there are with e-mail. The basic operation is simple: type a brief message and press Enter.

.

SERVER:

Server-side application is used to get the message from any client and broadcast to each and every client. And this application is also used to maintain the list of users and broadcast this list to everyone. It is a process that performs some functions on request from a client.
Chat server is an application which does the following operations:

- Listens for incoming calls from clients. Client running in any PC can connect to the server if IP address of the server is known.
- Listens for messages from all the connected clients.
- Broadcasts the message from clients to all the clients connected to the server.
- You can also type-in messages in the server, which will be broadcasted to all the clients

CLIENT

The software that resides in the user's computer for handling instant messaging (IM) or chat rooms. The Client connects to the server using the IP of the server and the encryption. Once the client application has connected to the server it will be able to send or receive messages.

SOCKET PROGRAMMING

A socket is one endpoint of a two way communication link between two programs running on the network. The socket mechanism provides a means of inter-process communication (IPC) by establishing named contact points between which the communication take place.

Each socket has a specific address. This address is composed of an IP address and a port number.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection.

Server forms the listener socket while client reaches out to the server. They are the real backbones behind web browsing. In simpler terms there is a server and a client.
Socket programming is started by importing the socket library and making a simple socket.

Here we are have a server socket.

server_socket=socket.socket(socket.AF_INET,socket.SOCK_STREAM)

- AF_INET refers to the address family IPv4.
- The SOCK_STREAM means connection oriented TCP protocol.

server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

- SOL_ - socket option level
- SO_ - socket option
- Sets REUSEADDR (as a socket option) to 1 on socket. This permits reuse of address.

server_socket.bind((IP, PORT))

- A server has a bind() method which binds it to a specific ip and port so that it can listen to incoming requests on that IP and port.

server_socket.listen()

- A server has a listen() method which puts the server into listen mode. This allows the server to listen to incoming connections.

server_socket.recv(<buffer_size>) and server_socket.send(<buffer_size>)

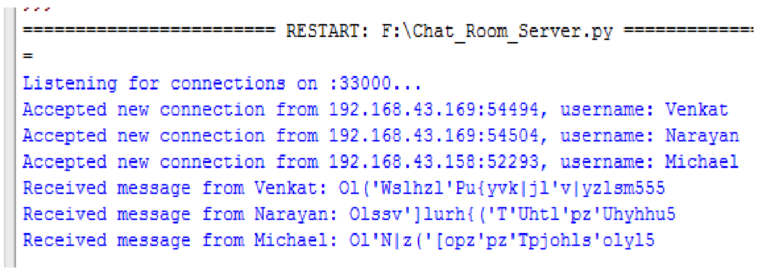
- Used to receive and send messages of the specified buffer size.

accept() and close()

- The accept method initiates a connection with the client and the close method closes the connection with the client.

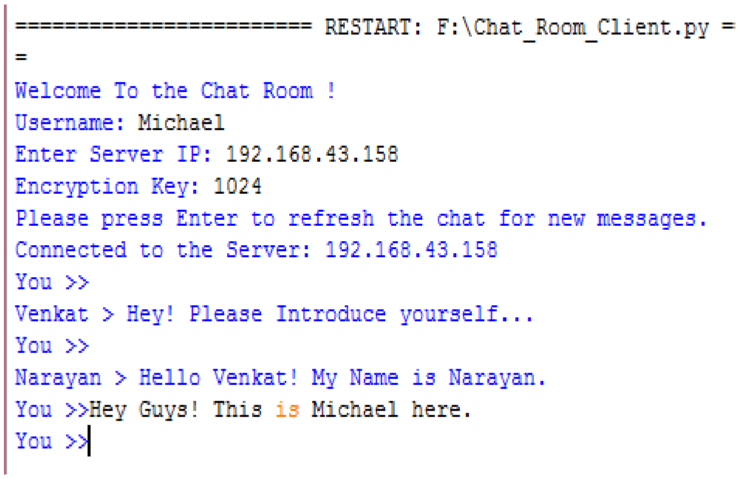
ENCRYPTION

The encryption used here is Caesar cipher. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.



```
>>>
======================== RESTART: F:\Chat_Room_Server.py ============
=
Listening for connections on :33000...
Accepted new connection from 192.168.43.169:54494, username: Venkat
Accepted new connection from 192.168.43.169:54504, username: Narayan
Accepted new connection from 192.168.43.158:52293, username: Michael
Received message from Venkat: Ol('Wslhzl'Pu{yvk|jl'v|yzlsm555
Received message from Narayan: Olssv']lurh{('T'Uhtl'pz'Uhyhhu5
Received message from Michael: Ol'N|z('[opz'pz'Tpjohls'olyl5
```

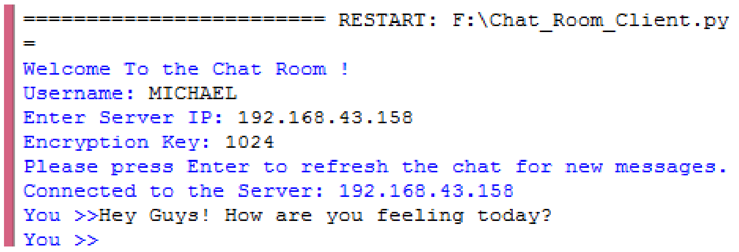
Fig1. Snip of the Encrypted messages received at the server.



```
======================== RESTART: F:\Chat_Room_Client.py =
=
Welcome To the Chat Room !
Username: Michael
Enter Server IP: 192.168.43.158
Encryption Key: 1024
Please press Enter to refresh the chat for new messages.
Connected to the Server: 192.168.43.158
You >>
Venkat > Hey! Please Introduce yourself...
You >>
Narayan > Hello Venkat! My Name is Narayan.
You >>Hey Guys! This is Michael here.
You >>
```

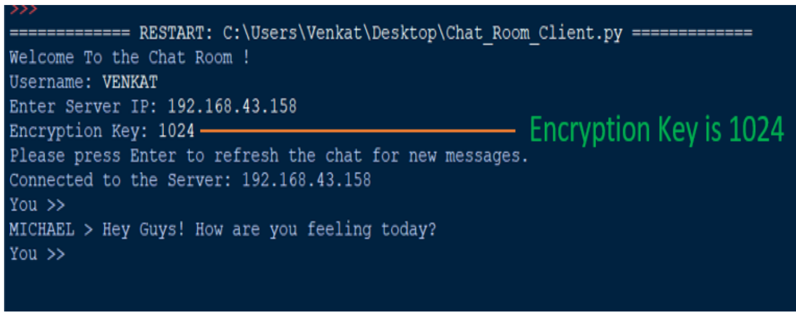
Fig2.Snip of The Message sent and received at the Client

Case when Encryption Key is Wrong



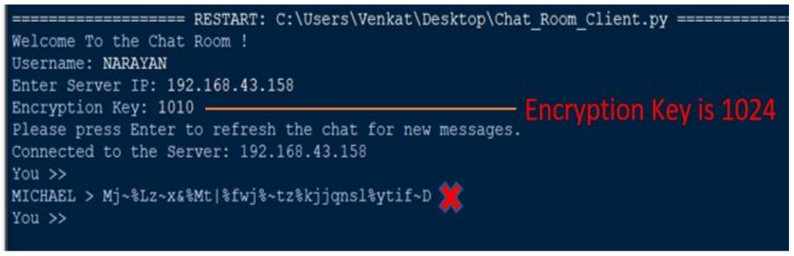
```
======================== RESTART: F:\Chat_Room_Client.py
=
Welcome To the Chat Room !
Username: MICHAEL
Enter Server IP: 192.168.43.158
Encryption Key: 1024
Please press Enter to refresh the chat for new messages.
Connected to the Server: 192.168.43.158
You >>Hey Guys! How are you feeling today?
You >>
```

Fig3. Message Sent with Key as 1024



```
>>>
============ RESTART: C:\Users\Venkat\Desktop\Chat_Room_Client.py ============
Welcome To the Chat Room !
Username: VENKAT
Enter Server IP: 192.168.43.158
Encryption Key: 1024
Please press Enter to refresh the chat for new messages.
Connected to the Server: 192.168.43.158
You >>
MICHAEL > Hey Guys! How are you feeling today?
You >>
```

Fig4.Message received with Key as 1024



```
================== RESTART: C:\Users\Venkat\Desktop\Chat_Room_Client.py ============
Welcome To the Chat Room !
Username: NARAYAN
Enter Server IP: 192.168.43.158
Encryption Key: 1010
Please press Enter to refresh the chat for new messages.
Connected to the Server: 192.168.43.158
You >>
MICHAEL > Mj~%Lz~x&%Mt|%fwj%~tz%kjjqnsl%ytif~D
You >>
```

Fig5.Message received with Key as 1010

When the encryption key is wrong that is 1010 instead of 1024 then the message is lost as seen in Fig5.

HEADER

```
HEADER_LENGTH = 10
message_header = client_socket.recv(HEADER_LENGTH)
message_length = int(message_header.decode('utf-8').strip())
return {'header': message_header, 'data': client_socket.recv(message_length)}
client_socket.send(user['header'] + user['data'] + message['header'] + message['data'])
```

The Header length has been set to size 10 and it carries the message length followed by trailing spaces.

Example:
If the message is: Hello World!

The Header would be: 12_ _ _ _ _ _ _ _
("12" followed by 8 trailing spaces)

The Packet sent would be: 12_ _ _ _ _ _ _ _ Hello World!

This is to ensure the Server client communication is not compromised. If the header length is different, for the server and client, the message would not be able to recover the message since:

- The message uses Caesar cipher.
- Message Length is unknown hence the buffer in the server will not function optimally.
- There are trailing spaces.

PYTHON OVER NPLANG

NPLang is restricted to Network programming and is very recent. The number of developers using this language would be relatively lesser than Python or Java. Python socket programming is efficient and powerful when compared to Java and hence further developments can be made using the project.

FURTHER DEVELOPMENTS

The client device needs to keep refreshing its command line and doesn't update itself automatically. Timed input needs to be implemented so that the client device can automatically display the received messages instead of command for updating. GUI can be used for better appearance and user interface.

CONCLUSION

The chat server developed has header, cipher and provides the necessary security as it also requires IP of the server to connect to the server.

REFERENCES

Python Socket Programming,

- Real Python Guide https://realpython.com/python-sockets/
- Tutorials Point https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm

Encryption and Cipher

- Practical Cryptography http://practicalcryptography.com/ciphers/
- Geeks for Geeks https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/