Keiran Berry

Networking

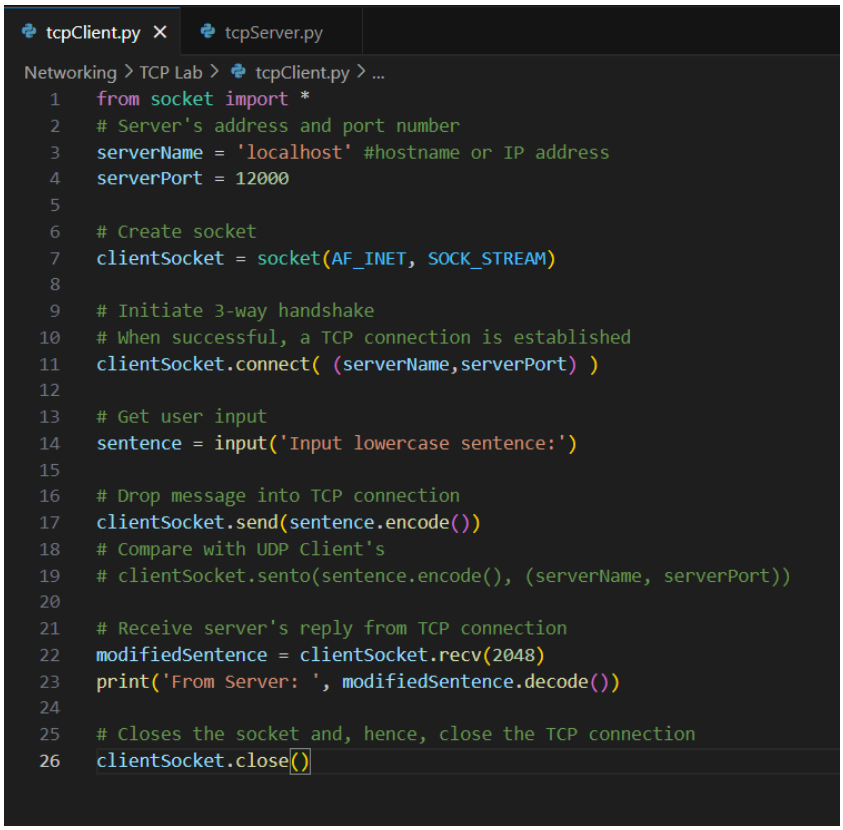
Lab 4: TCP/TLS Socket Programming

23 September 2024

Task 1

Writing client and server code from the teaching slides

Client Code:



Server Code:

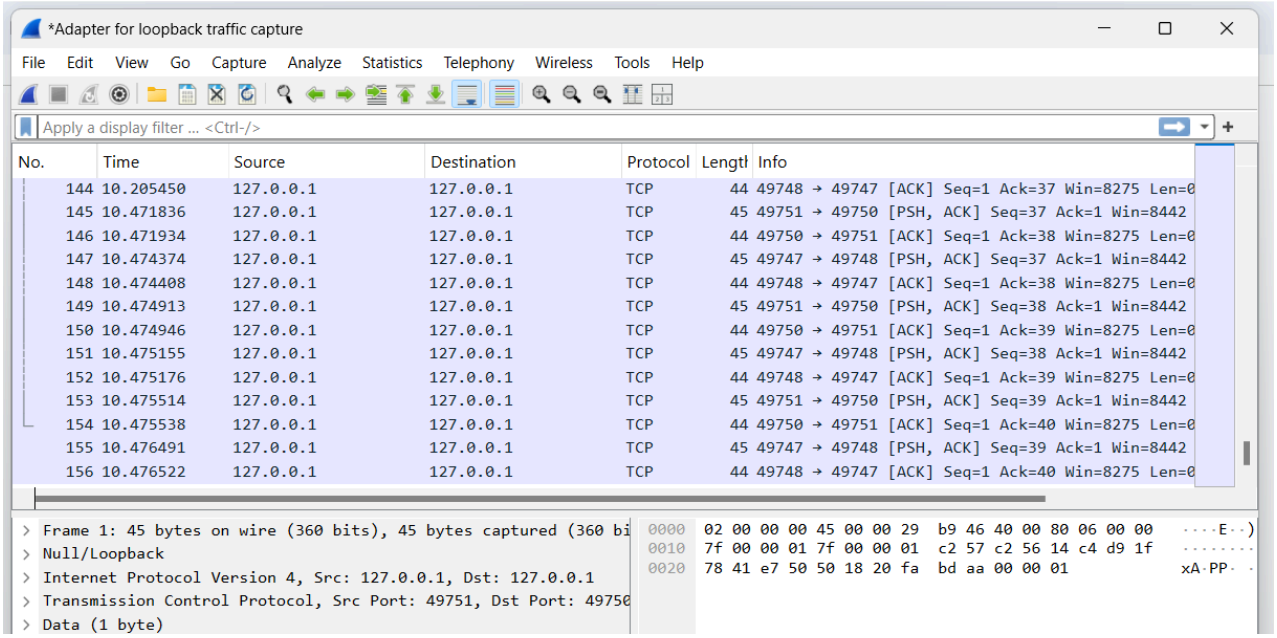
tcpClient.py tcpServer.py

Networking > TCP Lab > tcpServer.py > ...

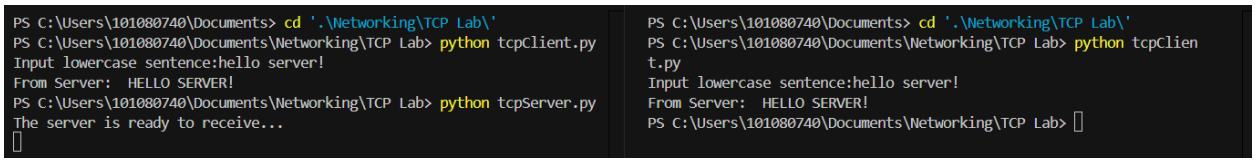
```python
from socket import *

# Set port number
serverPort = 12000

# Create "welcoming" socket
serverSocket = socket(AF_INET, SOCK_STREAM)

# Explicitly assign port number to socket
serverSocket.bind(('',serverPort))

# Wait and listen for some client to knock
serverSocket.listen(1)

print('The server is ready to receive...')

while True:
    # Welcome and create new dedicated socket
    connectionSocket, addr = serverSocket.accept()

    # Receive client's message from TCP connection
    sentence = connectionSocket.recv(2048)

    capitalizedSentence = sentence.decode().upper()

    # Send reply
    connectionSocket.send(capitalizedSentence.encode())

    connectionSocket.close()
```

Task 2

1. Getting to the capturing packets portion of Wireshark:

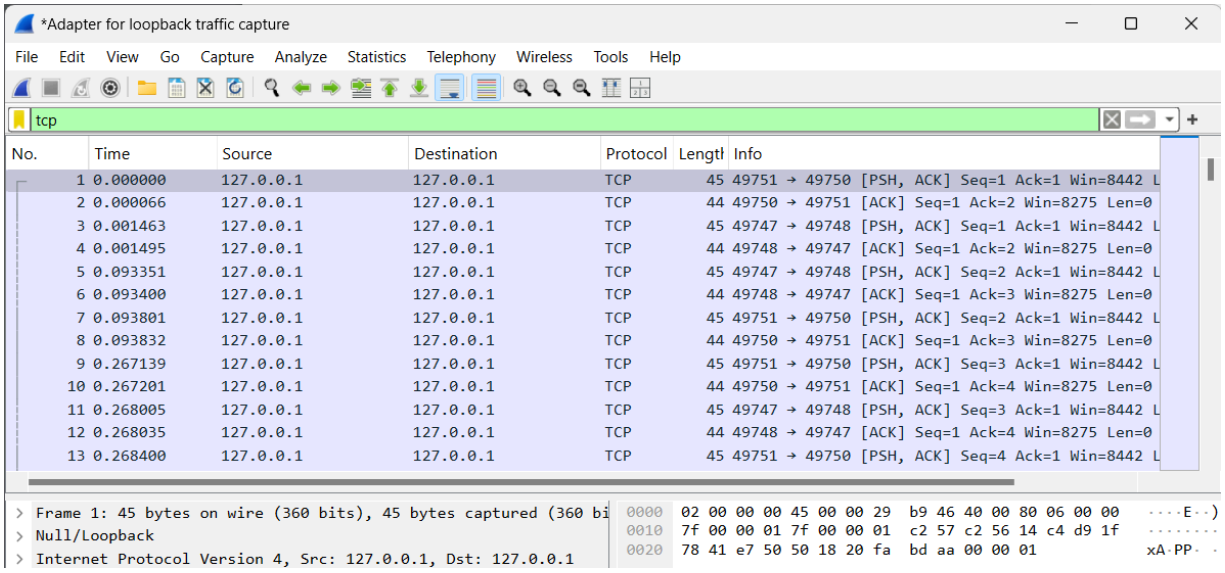


2. Running TCP server and client code (server first):

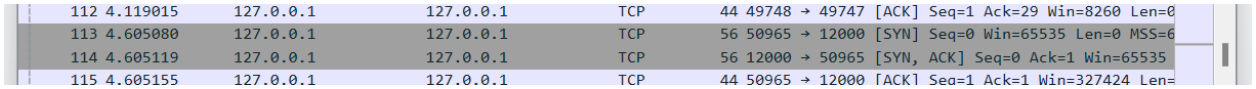


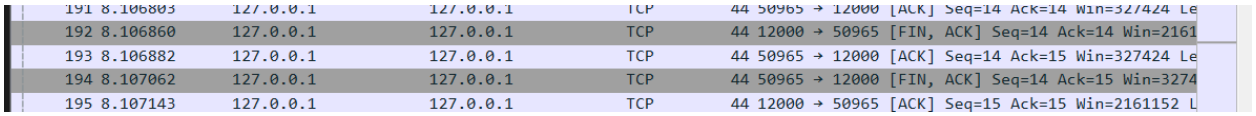
Here, I make sure to not only run the server and client but complete a run of the program from the client side so that we can see the full “TCP handshake”.

3. Applying filter to wireshark to only show TCP packets:

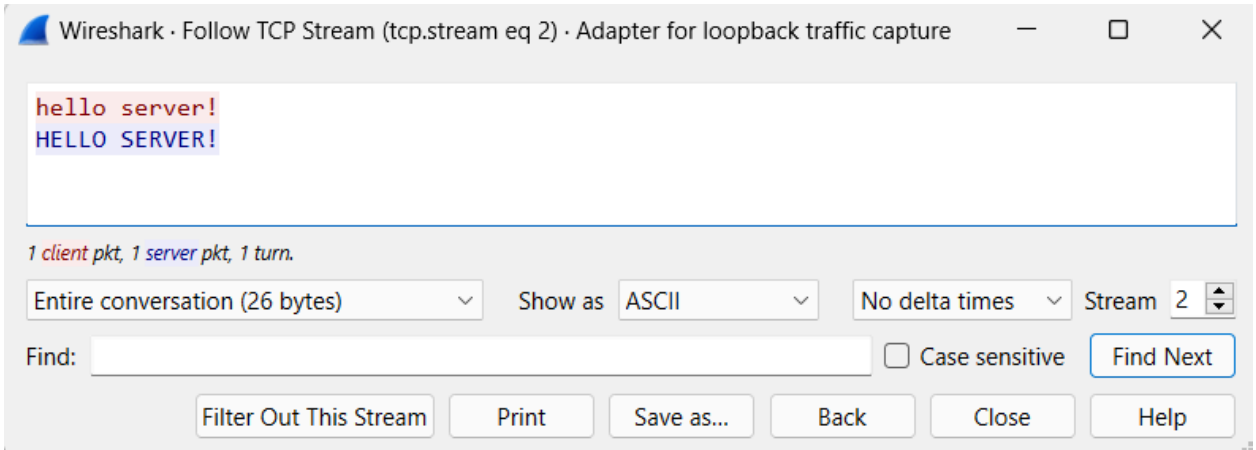


Here, I can observe the “TCP handshake” mentioned in class. Since there are lots of packets being picked up by Wireshark, they are not all clumped together. However, they are all here and can be seen a bit spread out.





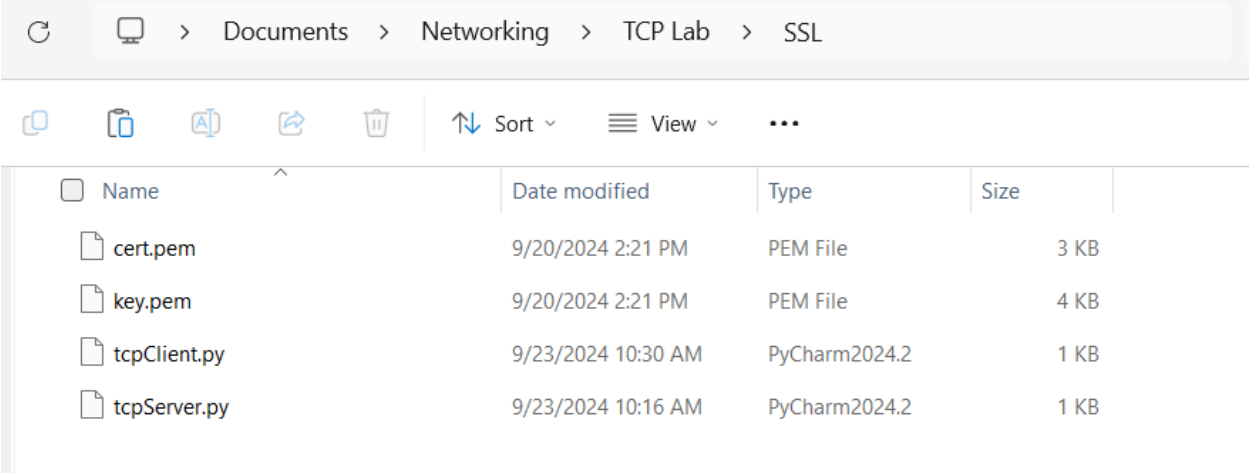
4. Following TCP stream from first packet of exchange:



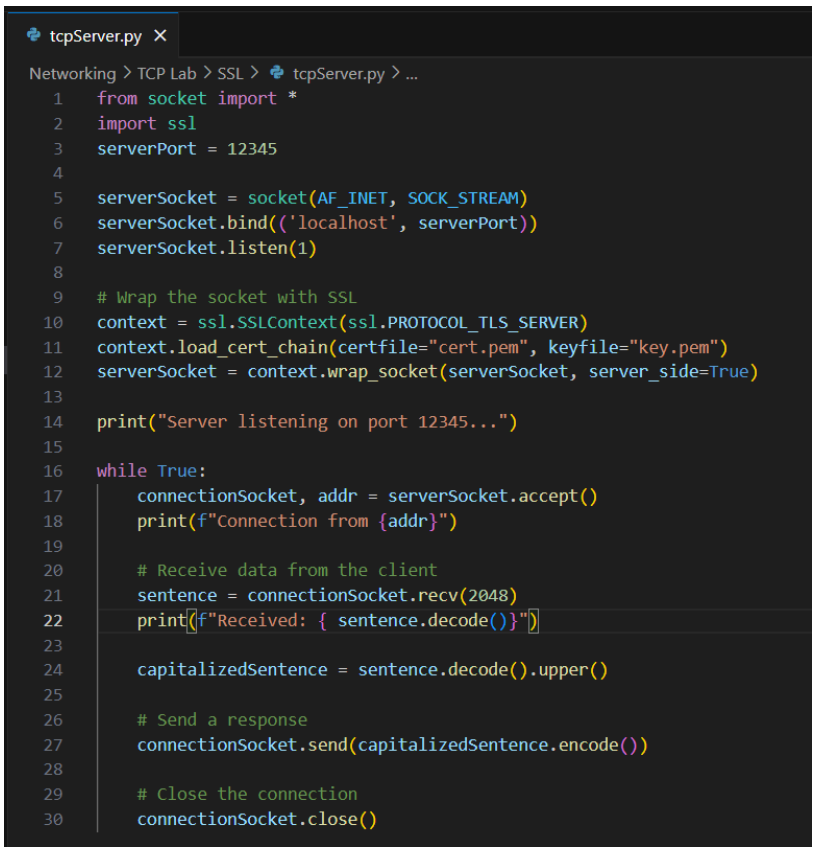
This is really interesting as Wireshark can not only see the packets, but also decode and display the encoded messages being sent back and forth. As we can see in the running server and client step, these messages are both the one that I sent to the server and the one which the server returned to me. This is because there is no security on these messages, so if this were my password being put in then it could easily be viewed by malicious parties.

Task 3: Adding Encryption to the TCP Server and Client

1. Downloading and extracting the key and certificate, and copying the client and server files into the folder:



2. Updating tcpServer.py using the code from the lab document:



   a. In the code snippet from lines 10-12:

      i. An SSLContext object is created, specifically for a server and using TLS protocol.

      ii. The certificate and private key are loaded into the previously created SSLContext object, using the files from the same directory.

      iii. The socket is wrapped using the SSLContext object which was created earlier. We also specify once again that this is the server side here.

3. Test run of server:

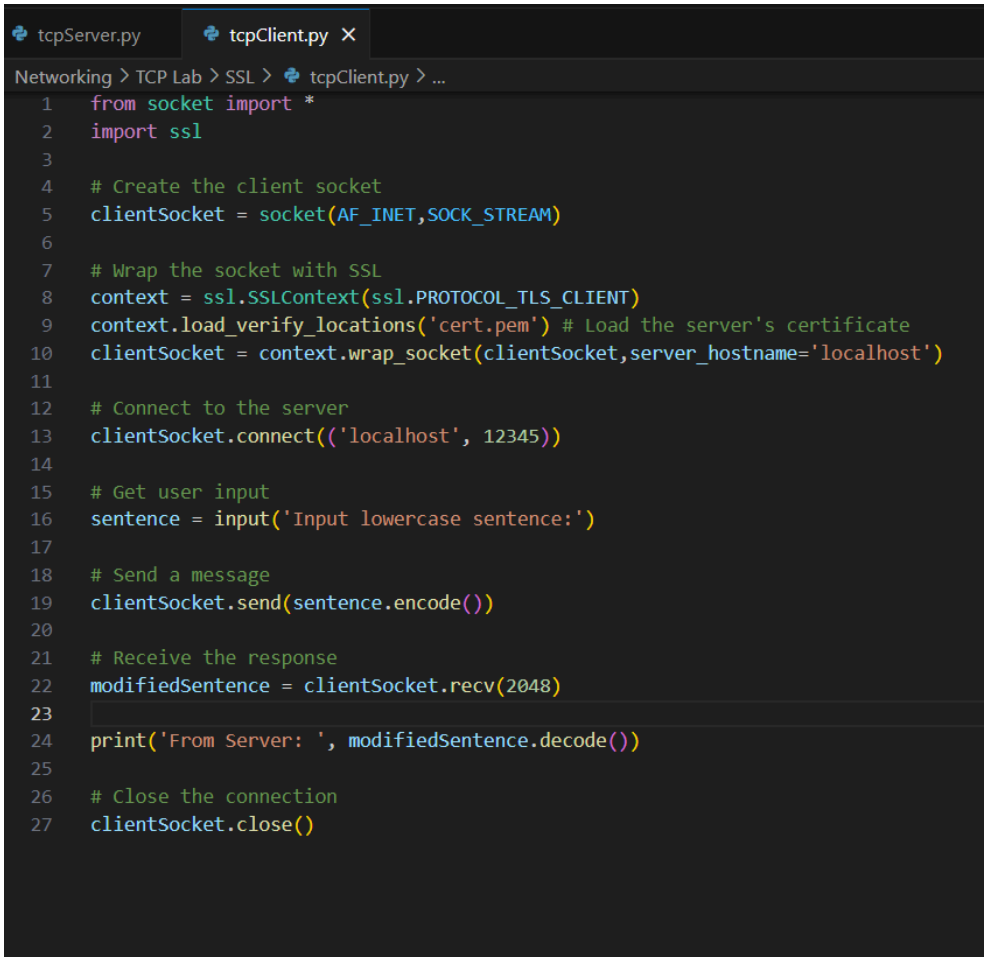
```
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL> python tcpServer.py
Server listening on port 12345...
```

As of now, we do not yet have the client edited to match what the server is listening for.

However, the server does run as intended and listens for the incoming encrypted connections.

Part B: Modify the Client for Encryption

1. Modifying client code to use SSL with code from lab document:



   a. In the code from lines 8-10:

i. An instance of SSLContext is created for the client side, using TLS protocol.

ii. The server's certificate is loaded into the context, so that the client can confirm that it is connecting to the correct server and not a potentially harmful one.

iii. Finally, the socket is wrapped to enable the encrypted communication with the server. An argument to specify the name of the server which will be connected to is also passed in.

2. Test run of client:

```
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL> python tcpClient
.py
Input lowercase sentence:hello encrypted server!
From Server:  HELLO ENCRYPTED SERVER!
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL>
```

When running the client side of the updated code, we can see that we still get the same capitalized version of the text which we send to the server.

Now that the client can successfully connect to the server, here is the server's side of this interaction:

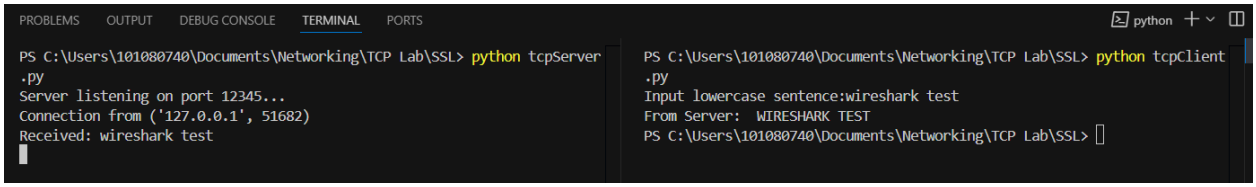
```
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL> python tcpServer
.py
Server listening on port 12345...
Connection from ('127.0.0.1', 51640)
Received: hello encrypted server!
```

Here, we can verify that the client is able to securely connect with the server and exchange messages.

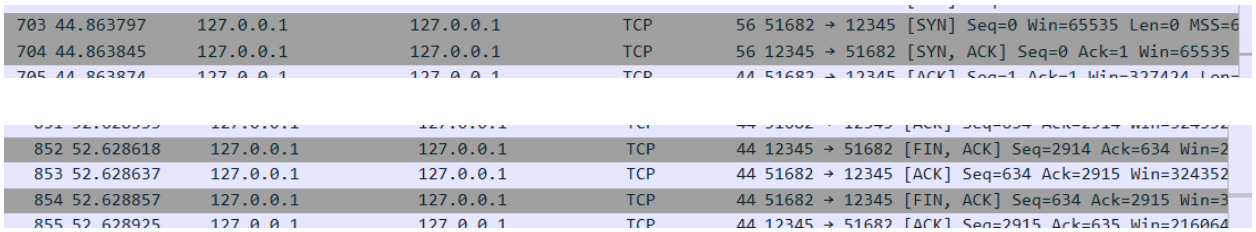
Part C: Validation

Verifying encryption using Wireshark:

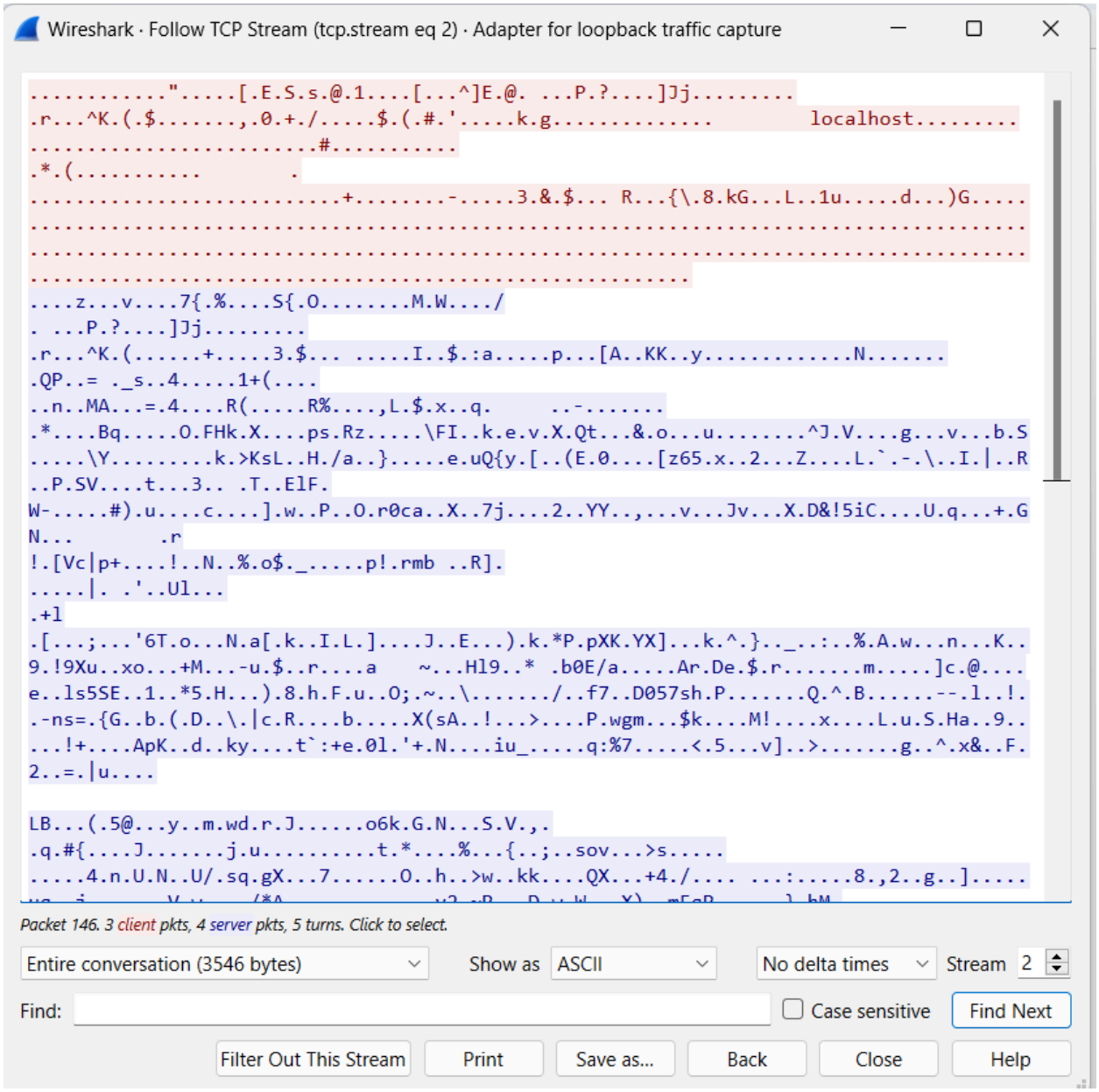
1. Sending a message to the server from the client

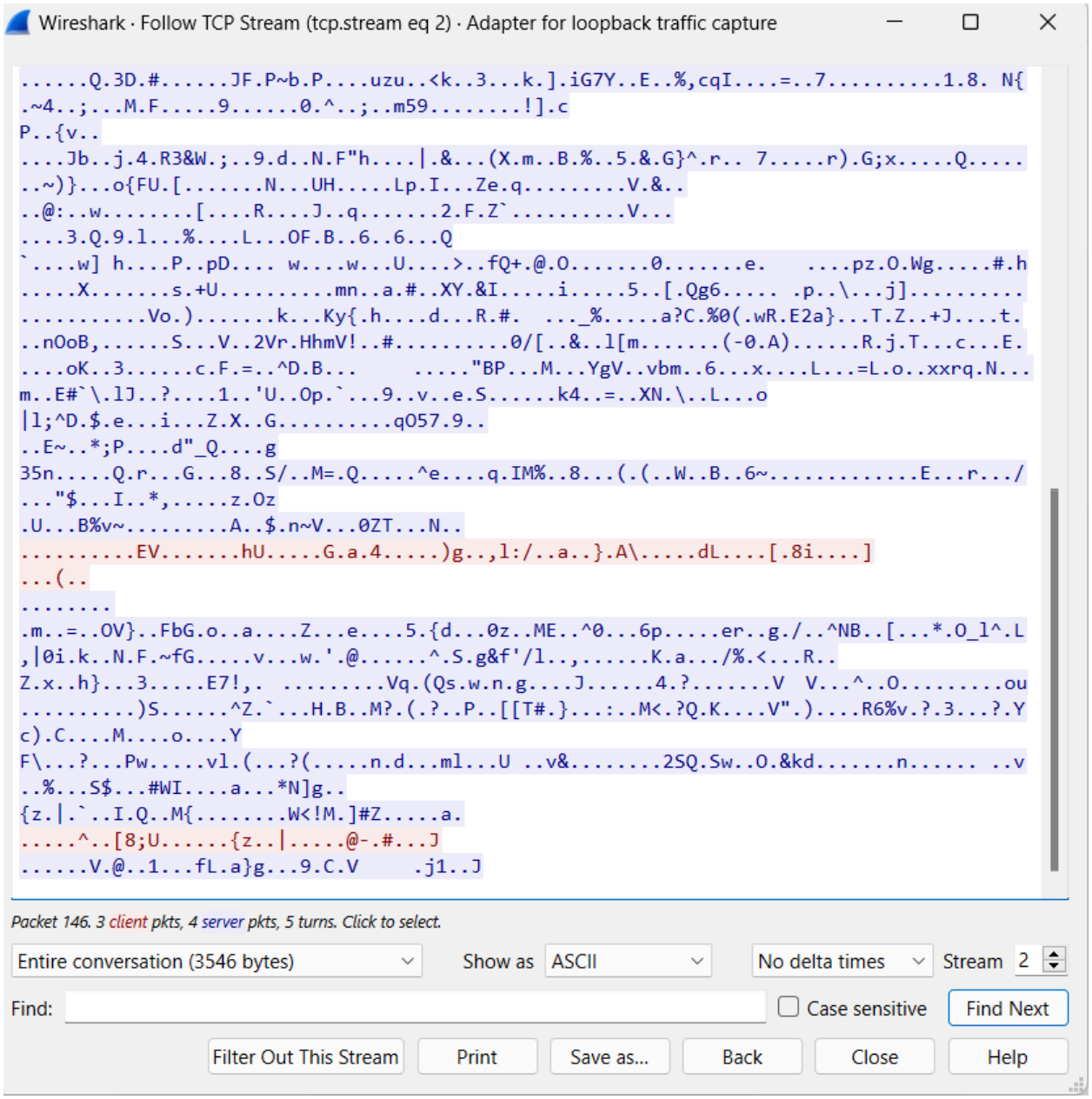


2. Finding the “TCP handshake” in Wireshark:



3. Following TCP stream:





Here, we can see that the traffic is definitely encrypted and is not readable in plaintext. This is a very stark contrast from the plaintext exchange we were able to see before. We can conclude that our encryption has succeeded, and the traffic is not so easily read anymore.