Bradley Peradotto PR03 Report

Introduction

This program sends email to a server using SMTP over TCP with TLS v1 or better for the sender client -> server interaction. Unlike the previous projects, Pr01 and Pr02, this project (PR03) uses TCP instead of UDP and only covers the sender -> client interaction. It uses TLS v1 or better to encrypt data and provide security. In order for the TLS connection to work, both the server and the client need a public/private key and a certificate. Both, also, need a truststore that has each other's certificate stored in the truststore (since I am requiring both clientside and server side authentication). Once the server is up and running it will wait for an incoming connection. After the client launches and a connection is established, the TLS handshake will take place once data is sent to the server. Once handshake is established, regular socket transmissions can happen except now they are encrypted. The JRE (java) handles any re-connections, cipher changes, or re-established handshakes. Like the previous project (PR02) users are still required to login with username and password. These will still be Base64 encoded. Once the username and password have been verified, the user can create and send email using SMTP (using appropriate codes) but now the connection is encrypted.
For PR03 it is multithreaded allowing for multiple senders. Previous versions of the multithreading in PR01, and PR02 did not work very well. I have tried a different approach (see code) for PR03. PR03 also has hidden files for logs called ".server_log", and sender usernames and base64 encoded passwords files called ".user_pass."

– Design choices and protocol/reply codes used.

On the sender client → server interaction over SMTP I first had options for HELP and RSET (reset) which follow the SMTP rules for how those work. Then I used the reply following reply codes:

"250 OK" – For when a successful interaction in the sequence has happened. Used for a successful hello, mail from, rcpt to, and successful message send.

"334" – Response after an "auth" sequence is generated for Username then Password.

"330" - Response when a user enters their username for the first time responds with 330 and a new password.

"235" – A successful login has occurred finishing and "auth" sequence.

"354" - Response for when server is ready for an email to be written.

"500" – Response for when an unrecognized command is entered or syntax error

"501 - Response when correct command is used by wrong domain is entered

“503” – Response when incorrect sequence of commands happens.

“530” – Response when incorrect authentication command is entered or bad username and password entered.

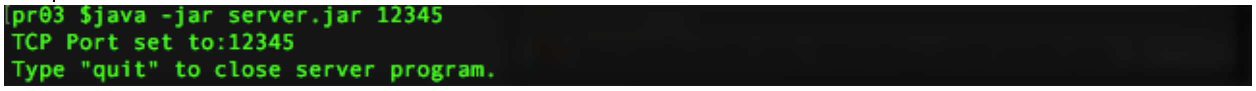
State Table for keeping track of order
Like the first projects PR01 and Pr02, I used an array list in PR03 of Objects called “State.” To store state data. Using the unique IP Address and port number associated with each sender client and receiver client, State data was stored in the array to determine where each connection was in the sequence of mail protocol. Once a connection was dropped, that connection was closed in the code and deleted from the server state table.

Keys, Certificates and Truststores
Each of the files for the keys, certificates, and truststores where created using java “keytool” and then saved into “db” folder. I decided to require both client sided and server side certificate authentication.

Make Keys Program
I also made a program that will generate a program that generates both the client keys and certificates and the server keys and certificates. They also generate truststores from each certificate. However, because the truststores need local certificate files, the server certificates is used to make the client truststore and vice versa. So the truststores have to be manually copied to each other’s folders. Client in server and server into client.

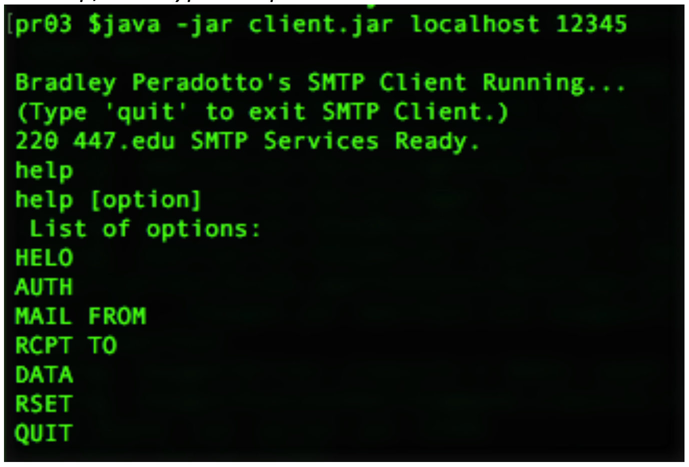
– The output of a sample run (including screenshots where applicable).

Sample run Server:



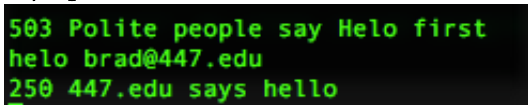
```
[pr03 $java -jar server.jar 12345
 TCP Port set to:12345
 Type "quit" to close server program.
```

Sample run Client:
Start up, then typed help.



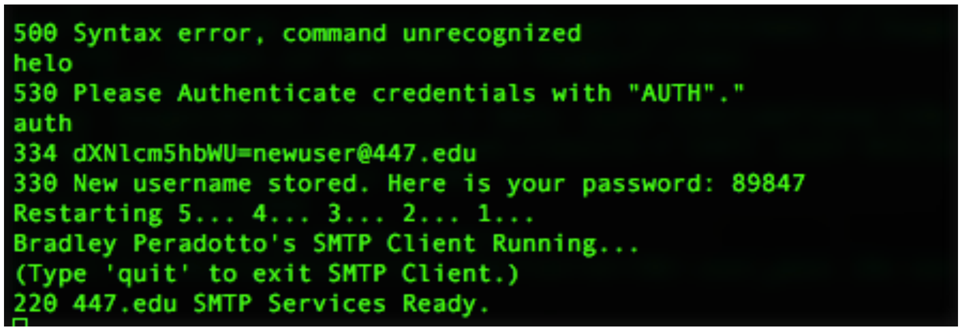
```
[pr03 $java -jar client.jar localhost 12345

 Bradley Peradotto's SMTP Client Running...
 (Type 'quit' to exit SMTP Client.)
 220 447.edu SMTP Services Ready.
 help
 help [option]
  List of options:
 HELO
 AUTH
 MAIL FROM
 RCPT TO
 DATA
 RSET
 QUIT
```

Saying hello:



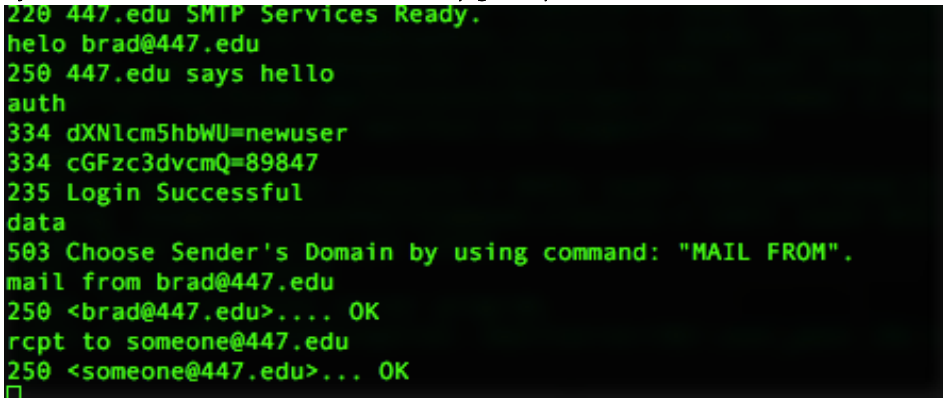
```
503 Polite people say Helo first
helo brad@447.edu
250 447.edu says hello
```

Showing authentication and reaction to incorrect input.
New username is entered and stored. Client restarts.



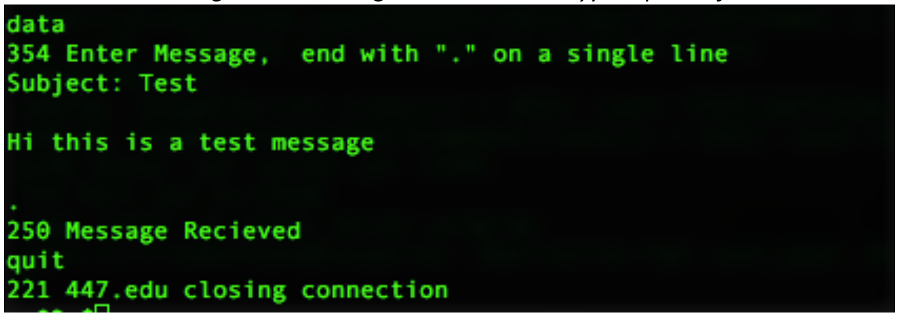
```
500 Syntax error, command unrecognized
helo
530 Please Authenticate credentials with "AUTH"."
auth
334 dXNlcm5hbWU=newuser@447.edu
330 New username stored. Here is your password: 89847
Restarting 5... 4... 3... 2... 1...
Bradley Peradotto's SMTP Client Running...
(Type 'quit' to exit SMTP Client.)
220 447.edu SMTP Services Ready.
```

After restart enter credentials with newly given password.



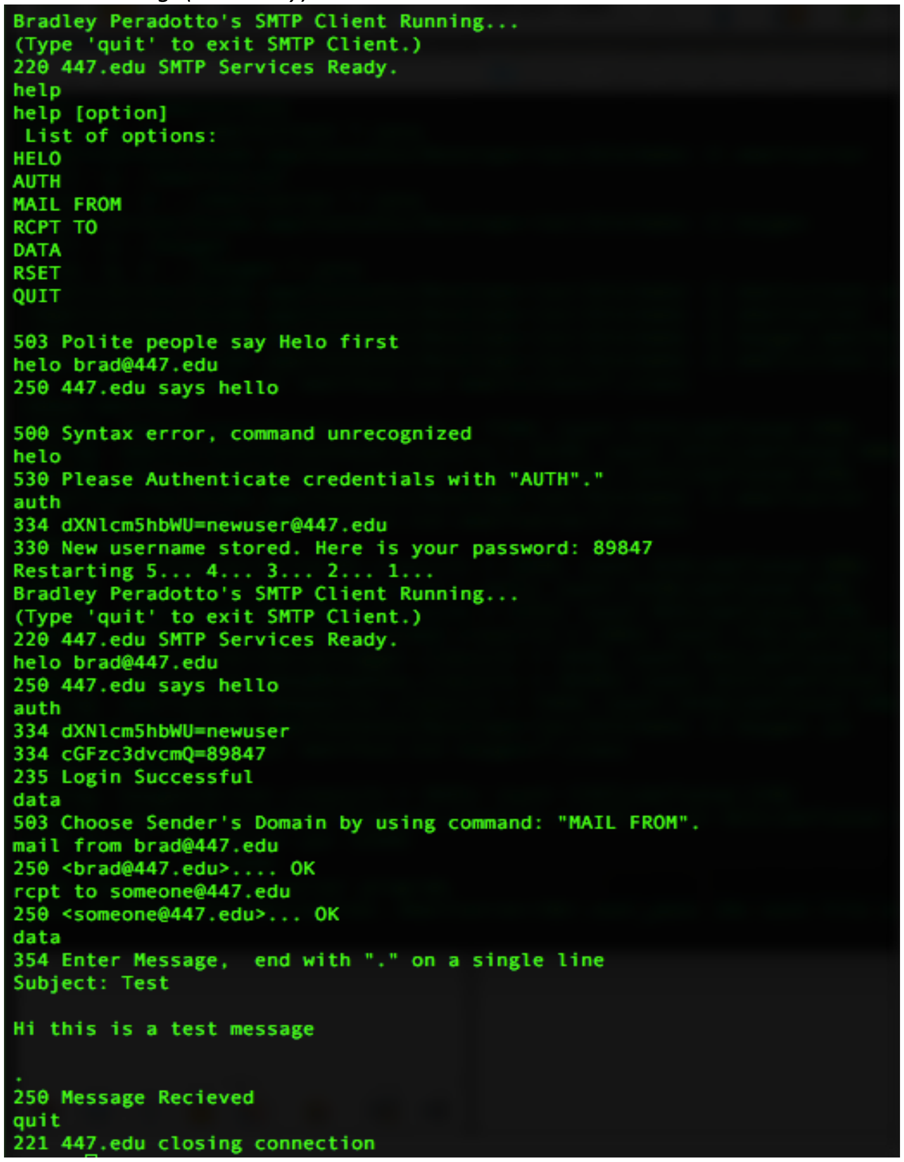
```
220 447.edu SMTP Services Ready.
helo brad@447.edu
250 447.edu says hello
auth
334 dXNlcm5hbWU=newuser
334 cGFzc3dvcmQ=89847
235 Login Successful
data
503 Choose Sender's Domain by using command: "MAIL FROM".
mail from brad@447.edu
250 <brad@447.edu>.... OK
rcpt to someone@447.edu
250 <someone@447.edu>... OK
```

Enter email message. End message with a dot "." Type "quit to finish.



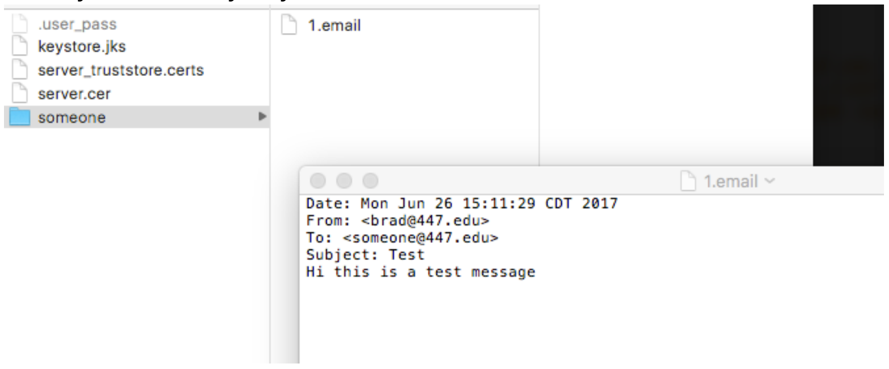
```
data
354 Enter Message,  end with "." on a single line
Subject: Test

Hi this is a test message


.
250 Message Recieved
quit
221 447.edu closing connection
```

Entire Exchange(Summary):



```
Bradley Peradotto's SMTP Client Running...
(Type 'quit' to exit SMTP Client.)
220 447.edu SMTP Services Ready.
help
help [option]
 List of options:
HELO
AUTH
MAIL FROM
RCPT TO
DATA
RSET
QUIT

503 Polite people say Helo first
helo brad@447.edu
250 447.edu says hello

500 Syntax error, command unrecognized
helo
530 Please Authenticate credentials with "AUTH"."
auth
334 dXNlcm5hbWU=newuser@447.edu
330 New username stored. Here is your password: 89847
Restarting 5... 4... 3... 2... 1...
Bradley Peradotto's SMTP Client Running...
(Type 'quit' to exit SMTP Client.)
220 447.edu SMTP Services Ready.
helo brad@447.edu
250 447.edu says hello
auth
334 dXNlcm5hbWU=newuser
334 cGFzc3dvcmQ=89847
235 Login Successful
data
503 Choose Sender's Domain by using command: "MAIL FROM".
mail from brad@447.edu
250 <brad@447.edu>.... OK
rcpt to someone@447.edu
250 <someone@447.edu>... OK
data
354 Enter Message,  end with "." on a single line
Subject: Test

Hi this is a test message


.
250 Message Recieved
quit
221 447.edu closing connection
```

View of the created files from above code.

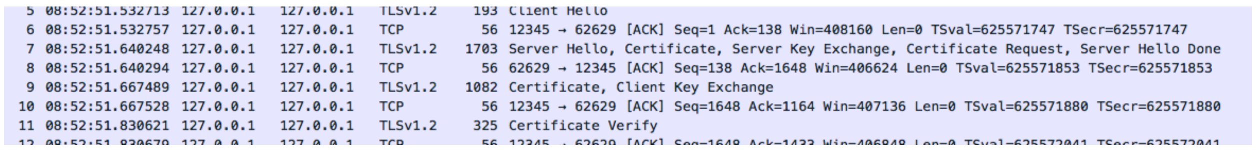


```
Date: Mon Jun 26 15:11:29 CDT 2017
From: <brad@447.edu>
To: <someone@447.edu>
Subject: Test
Hi this is a test message
```

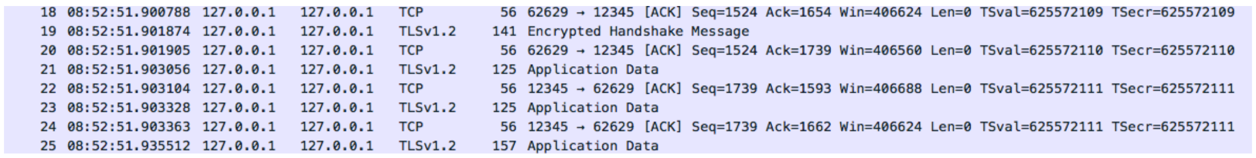
Wireshark Stuff (Extra Credit)

Secure Connection

I captured the following with Wireshark using the secure connection. Notice the handshake below. The “Server Hello, Certificate, Server key Exchange, Certificate Exchange ….” lines, the “Certificate, Client Key Exchange” lines, etc. (seen below) that show the handshake as well as the encrypted “application data” below.



```
 5 08:52:51.532713 127.0.0.1  127.0.0.1  TLSv1.2   193 Client Hello
 6 08:52:51.532757 127.0.0.1  127.0.0.1  TCP        56 12345 → 62629 [ACK] Seq=1 Ack=138 Win=408160 Len=0 TSval=625571747 TSecr=625571747
 7 08:52:51.640248 127.0.0.1  127.0.0.1  TLSv1.2  1703 Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
 8 08:52:51.640294 127.0.0.1  127.0.0.1  TCP        56 62629 → 12345 [ACK] Seq=138 Ack=1648 Win=406624 Len=0 TSval=625571853 TSecr=625571853
 9 08:52:51.667489 127.0.0.1  127.0.0.1  TLSv1.2  1082 Certificate, Client Key Exchange
10 08:52:51.667528 127.0.0.1  127.0.0.1  TCP        56 12345 → 62629 [ACK] Seq=1648 Ack=1164 Win=407136 Len=0 TSval=625571880 TSecr=625571880
11 08:52:51.830621 127.0.0.1  127.0.0.1  TLSv1.2   325 Certificate Verify
12 08:52:51.830679 127.0.0.1  127.0.0.1  TCP        56 12345 → 62629 [ACK] Seq=1648 Ack=1433 Win=406848 Len=0 TSval=625572041 TSecr=625572041
```



```
18 08:52:51.900788 127.0.0.1  127.0.0.1  TCP        56 62629 → 12345 [ACK] Seq=1524 Ack=1654 Win=406624 Len=0 TSval=625572109 TSecr=625572109
19 08:52:51.901874 127.0.0.1  127.0.0.1  TLSv1.2   141 Encrypted Handshake Message
20 08:52:51.901905 127.0.0.1  127.0.0.1  TCP        56 62629 → 12345 [ACK] Seq=1524 Ack=1739 Win=406560 Len=0 TSval=625572110 TSecr=625572110
21 08:52:51.903056 127.0.0.1  127.0.0.1  TLSv1.2   125 Application Data
22 08:52:51.903104 127.0.0.1  127.0.0.1  TCP        56 12345 → 62629 [ACK] Seq=1739 Ack=1593 Win=406688 Len=0 TSval=625572111 TSecr=625572111
23 08:52:51.903328 127.0.0.1  127.0.0.1  TLSv1.2   125 Application Data
24 08:52:51.903363 127.0.0.1  127.0.0.1  TCP        56 12345 → 62629 [ACK] Seq=1739 Ack=1662 Win=406624 Len=0 TSval=625572111 TSecr=625572111
25 08:52:51.935512 127.0.0.1  127.0.0.1  TLSv1.2   157 Application Data
```

/Users/augustgate/Desktop/brad school stuff/CS447/Pr03beans/Wireshark Files/capture1.pcapng 28 total packets, 28 shown

```
     21 08:52:51.903056    127.0.0.1             127.0.0.1             TLSv1.2  125    Application
Data
Frame 21: 125 bytes on wire (1000 bits), 125 bytes captured (1000 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 62629, Dst Port: 12345, Seq: 1524, Ack: 1739, Len: 69
    Source Port: 62629
    Destination Port: 12345
    [Stream index: 0]
    [TCP Segment Len: 69]
    Sequence number: 1524    (relative sequence number)
    [Next sequence number: 1593    (relative sequence number)]
    Acknowledgment number: 1739    (relative ack number)
    Header Length: 32 bytes
    Flags: 0x018 (PSH, ACK)
    Window size value: 12705
    [Calculated window size: 406560]
    [Window size scaling factor: 32]
    Checksum: 0xfe6d [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
        No-Operation (NOP)
        No-Operation (NOP)
        Timestamps: TSval 625572111, TSecr 625572110
            Kind: Time Stamp Option (8)
            Length: 10
            Timestamp value: 625572111
            Timestamp echo reply: 625572110
    [SEQ/ACK analysis]
Secure Sockets Layer
    TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
        Content Type: Application Data (23)
        Version: TLS 1.2 (0x0303)
        Length: 64
        Encrypted Application Data: a24978cb247d392f16a41798bf9e89eb3b22b90f9a1896ad...
```

Insecure Connection

For the insecure connection, there was no encryption hand shake. Plus, notice how you can read the SMTP commands below (denoted by arrows). Anyone could intercept this data. The Wireshark capture reads the plain messages.

/var/folders/vb/m07_44_x7dxdzcp57dp_xy080000gq/T//wireshark_lo0_20170626161104_mP3FJd.pcapng 60 total packets, 60 shown

```
      9 16:11:10.895056    127.0.0.1             127.0.0.1             TCP      88     12345 →
53806 [PSH, ACK] Seq=1 Ack=6 Win=408288 Len=32 TSval=754225657 TSecr=754225548
Frame 9: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 12345, Dst Port: 53806, Seq: 1, Ack: 6, Len: 32
Data (32 bytes)
0000  32 32 30 20 34 34 37 2e 65 64 75 20 53 4d 54 50   220 447.edu SMTP
0010  20 53 65 72 76 69 63 65 73 20 52 65 61 64 79 2e    Services Ready.
    Data: 323230203434372e65647520534d5450205365727669636...
    [Length: 32]
```

/var/folders/vb/m07_44_x7dxdzcp57dp_xy080000gq/T//wireshark_lo0_20170626161104_mP3FJd.pcapng 60 total packets, 60 shown

```
     21 16:11:25.686988    127.0.0.1             127.0.0.1             TCP      73     53806 →
12345 [PSH, ACK] Seq=11 Ack=78 Win=408192 Len=17 TSval=754240441 TSecr=754229402
Frame 21: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 53806, Dst Port: 12345, Seq: 11, Ack: 78, Len: 17
Data (17 bytes)
0000  68 65 6c 6f 20 62 72 61 64 40 34 34 37 2e 65 64   helo brad@447.ed
0010  75                                                u
    Data: 68656c6f2062726164403434372e656475
    [Length: 17]
```

/var/folders/vb/m07_44_x7dxdzcp57dp_xy080000gq/T//wireshark_lo0_20170626161104_mP3FJd.pcapng 60 total packets, 60 shown

```
     37 16:11:34.734568    127.0.0.1             127.0.0.1             TCP      60     53806 →
12345 [PSH, ACK] Seq=34 Ack=118 Win=408160 Len=4 TSval=754249481 TSecr=754242296
Frame 37: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 53806, Dst Port: 12345, Seq: 34, Ack: 118, Len: 4
Data (4 bytes)
0000  62 72 61 64                                       brad
    Data: 62726164
    [Length: 4]
```

– Summary and Issues encountered (if applicable).

There were three main issues that I encountered with this assignment. The first was creating the certificates and keystores correctly for the TLS handshake. I originally tried to create the certificates and keystores programmatically in the code so that the program could just run and have all that automatically work. But soon discovered this was difficult in getting the handshake to work. So then I used java's "keytool" which is very similar to "open_ssl." This produced the keystores and certificates I needed. The next issue I encountered was still related to the TLS handshake. I kept receiving errors. I then discovered through research online, that the certificate that was created for the server needed to be physically copied to the client's truststore file. Once I discovered this, the handshake worked. The final issue came in trying to get Wireshark to capture TLS. I soon discovered that Wireshark by default only captures TLS on port 443. So I added port "12345" to that list and ran my client -> server interaction over port "12345" and then it captured the TLS interactions.