Blake Rusteberg

Programming Assignment #03: TLS Protocol

CS 447: Networking and Data Communication

Introduction

This project was intended for us to get a better understanding of the TLS protocol over a TCP channel. I started this project in Python and learned the basic functionality of the *ssl* library. It took a while to understand the functionality of it because there were a lot of functions that did not work with others in the library. It was not clearly stated in the PR03 requirements that only 1 private key and public key (cert) were needed for this project. I later learned that we were just assuming the user already had the server's cert to begin with. This probably took the most amount of time because I did not know much TLS worked, and knew more about how user authentication worked.

This project was less technically heavy as the last project and there seemed like there was a clear direction to move in reading the Technical Requirements. The 3 main difficulties I had in this project were making a secure channel with the controller ←→ server, authenticating the user and error checking if a user already exists in the .user_pass file, and getting the Wireshark to listen to the TLS and UDP packets on the VM.

Objectives

- Learn how to use python's *ssl* protocol
- Create a cert and key using *openssl*
- Create a separate program that does the basic needs of the TLS protocol
- Establish a secure channel between the controller ←→ server
- Allow the user to type a username and password before playing the audio file
- Create a .user_pass file for the server to store credentials
- Authenticate the user
- Error checking the authentication of the user
- Create a log file to timestamp the server's actions
- Allow multiple clients to login
- Allow multiple clients to stream
- Switch from using base 64 encoding to a md5 hash function
- Learn how to run Wireshark on Linux
- Write report
- TEST code
- Clean up code, create make file, create readme, create tar file
- Turn in

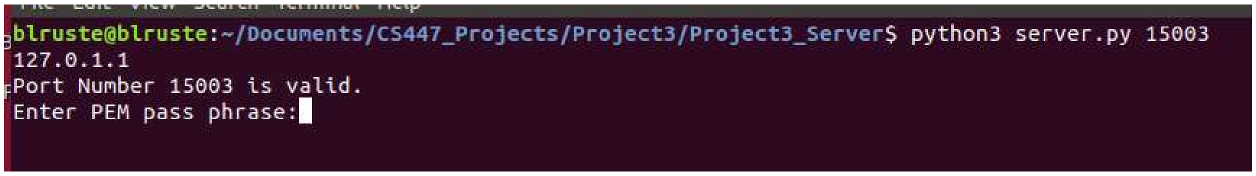
Design Choices

I stuck to a very simple design choice in this project because this was an addition to our Program 2. The user is only allowed to run SETUP at the beginning of the program because of user authentication. The flow of this project is simple:

Establish TLS secure channel → SETUP command → User Authentication → PLAY || PAUSE || TEARDOWN.

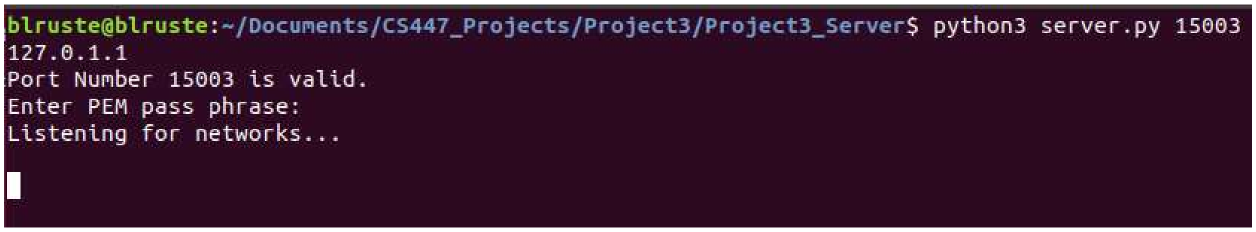
Program Run-through

Client-Server secure channel



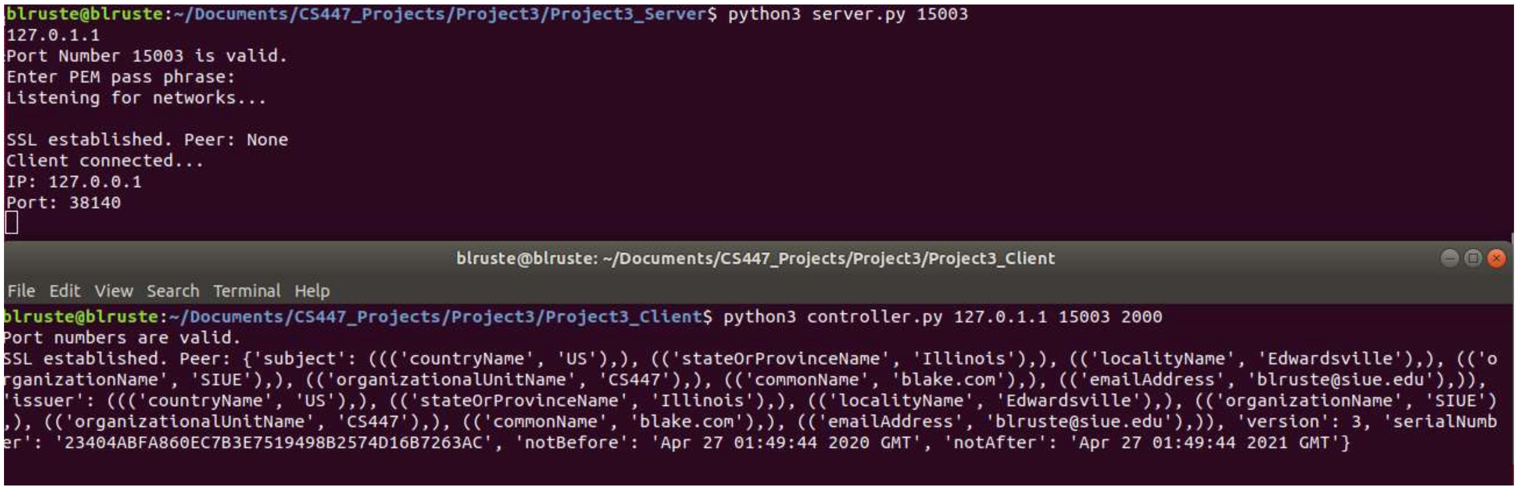
```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Server$ python3 server.py 15003
127.0.1.1
Port Number 15003 is valid.
Enter PEM pass phrase:
```

Figure 1.1: When the server is started it will first be asked to enter a passphrase: In this case it is 'blruste'



```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Server$ python3 server.py 15003
127.0.1.1
Port Number 15003 is valid.
Enter PEM pass phrase:
Listening for networks...
```

Figure 1.2: When the pass phrase is entered, the server will now listen for networks trying to connect to it



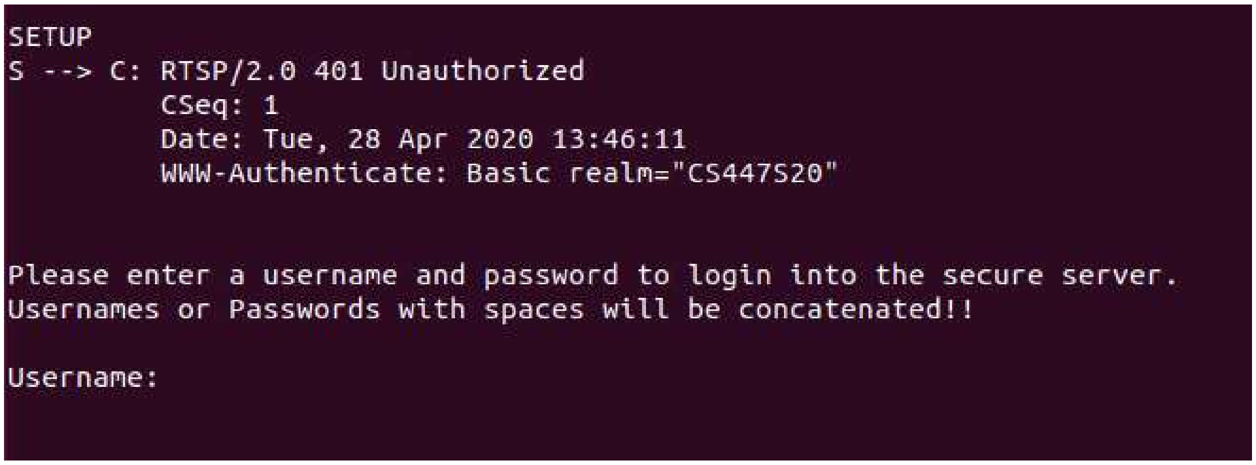
```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Server$ python3 server.py 15003
127.0.1.1
Port Number 15003 is valid.
Enter PEM pass phrase:
Listening for networks...

SSL established. Peer: None
Client connected...
IP: 127.0.0.1
Port: 38140
```

blruste@blruste: ~/Documents/CS447_Projects/Project3/Project3_Client
File Edit View Search Terminal Help

```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Client$ python3 controller.py 127.0.1.1 15003 2000
Port numbers are valid.
SSL established. Peer: {'subject': ((('countryName', 'US'),), (('stateOrProvinceName', 'Illinois'),), (('localityName', 'Edwardsville'),), (('o
rganizationName', 'SIUE'),), (('organizationalUnitName', 'CS447'),), (('commonName', 'blake.com'),), (('emailAddress', 'blruste@siue.edu'),)),
'issuer': ((('countryName', 'US'),), (('stateOrProvinceName', 'Illinois'),), (('localityName', 'Edwardsville'),), (('organizationName', 'SIUE')
,), (('organizationalUnitName', 'CS447'),), (('commonName', 'blake.com'),), (('emailAddress', 'blruste@siue.edu'),)), 'version': 3, 'serialNumb
er': '23404ABFA860EC7B3E7519498B2574D16B7263AC', 'notBefore': 'Apr 27 01:49:44 2020 GMT', 'notAfter': 'Apr 27 01:49:44 2021 GMT'}
```

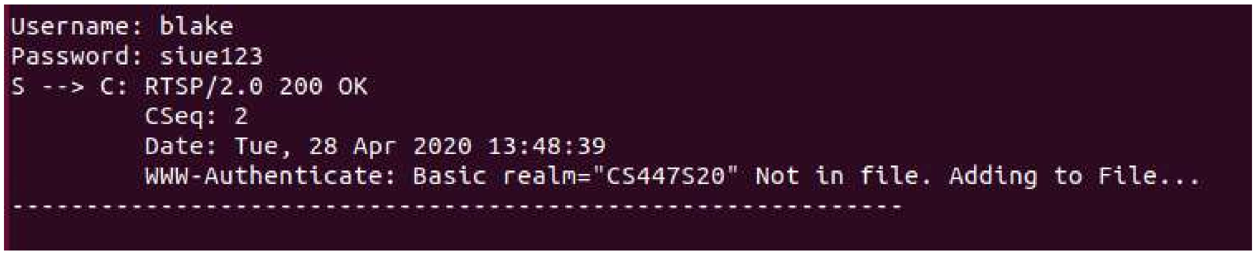
Figure 1.3: Now the controller will be able to connect to the server. When the controller connects it will return that cert that it used to connect to the server

User Authentication



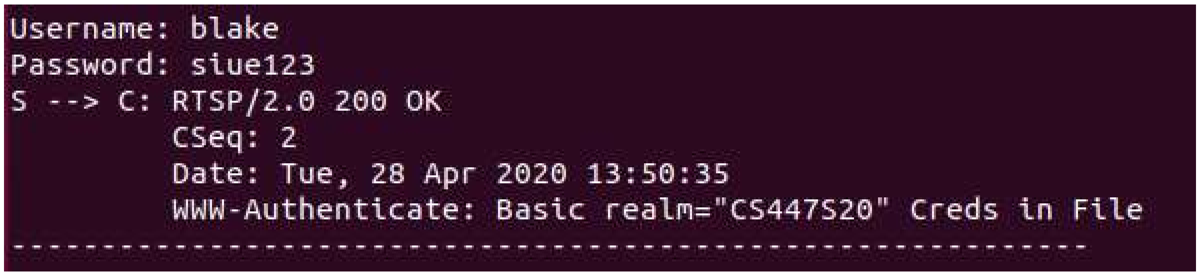
```
SETUP
S --> C: RTSP/2.0 401 Unauthorized
         CSeq: 1
         Date: Tue, 28 Apr 2020 13:46:11
         WWW-Authenticate: Basic realm="CS447S20"


Please enter a username and password to login into the secure server.
Usernames or Passwords with spaces will be concatenated!!

Username:
```

Figure 2.1: The controller will issue the SETUP command and the server will reply with a 401 Unauthorized code. The controller will now be able to enter a username and password



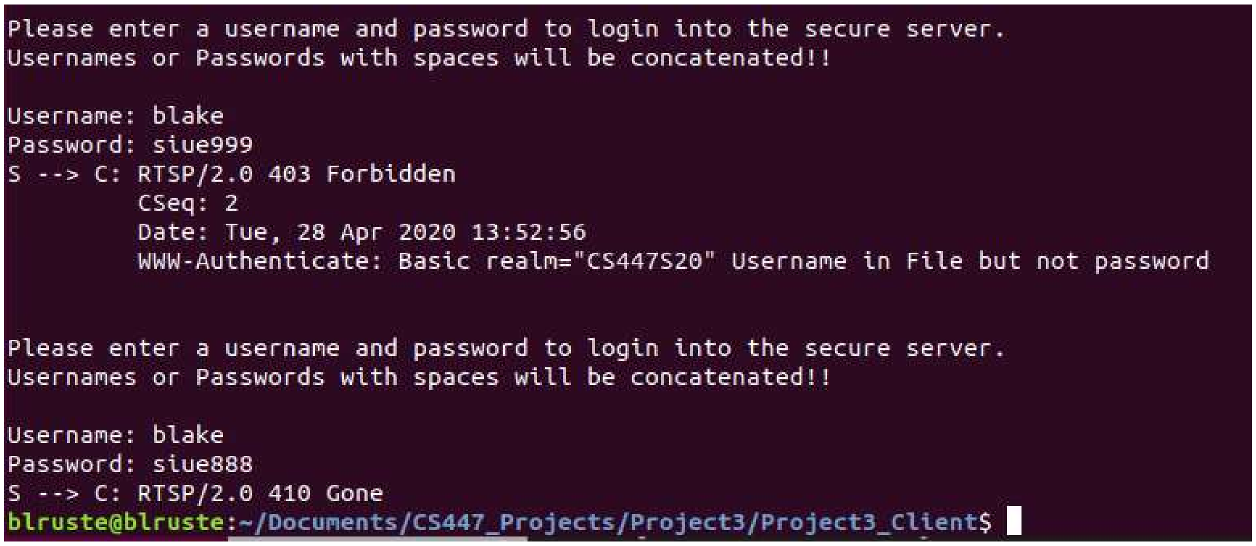
```
Username: blake
Password: siue123
S --> C: RTSP/2.0 200 OK
         CSeq: 2
         Date: Tue, 28 Apr 2020 13:48:39
         WWW-Authenticate: Basic realm="CS447S20" Not in file. Adding to File...
------------------------------------------------------------
```

Figure 2.2: If the user enters a username that does not exist it will create a new one and add it to the .user_pass file



```
Username: blake
Password: siue123
S --> C: RTSP/2.0 200 OK
         CSeq: 2
         Date: Tue, 28 Apr 2020 13:50:35
         WWW-Authenticate: Basic realm="CS447S20" Creds in File
------------------------------------------------------------
```

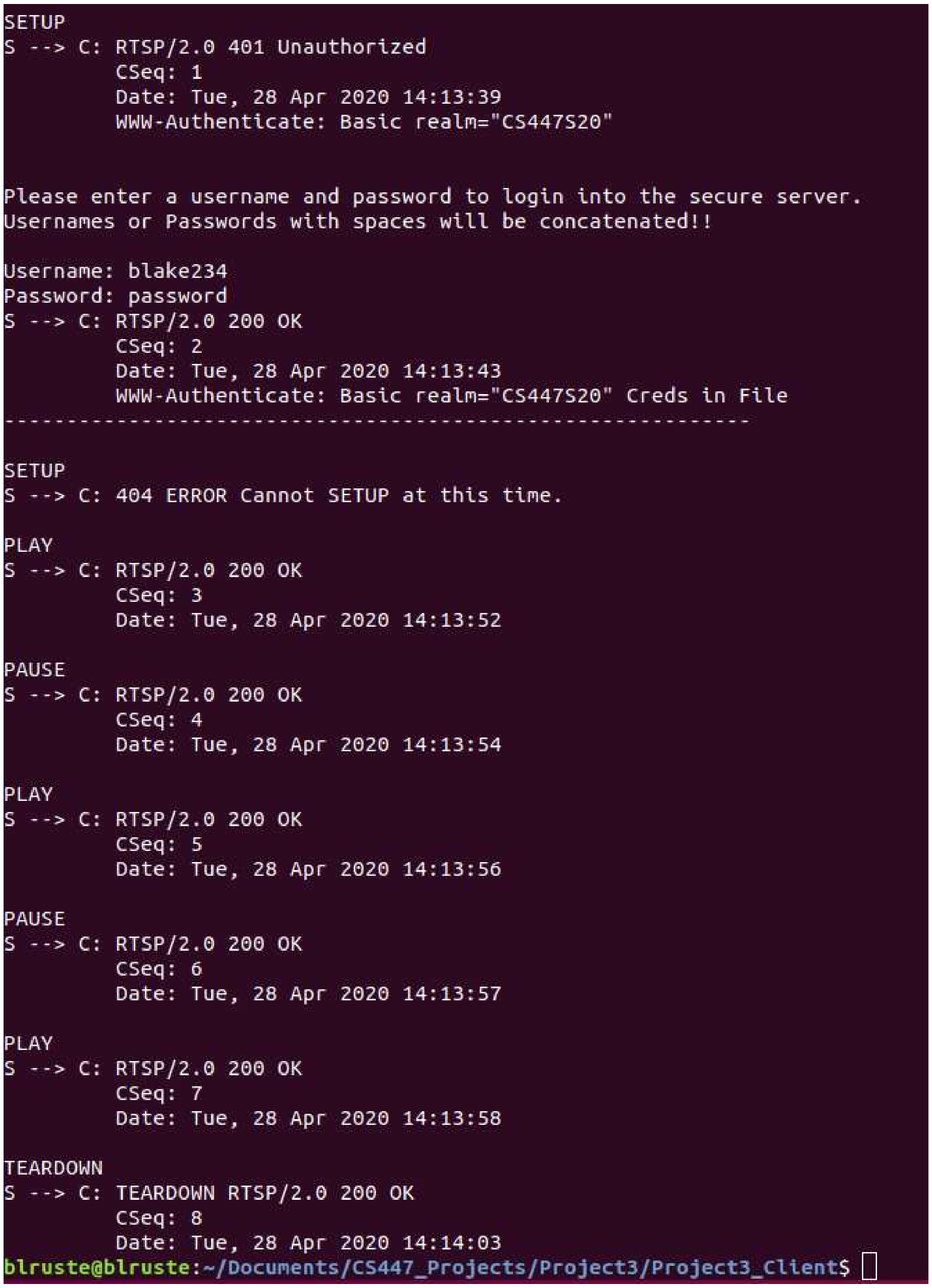
Figure 2.3: If the user enters a username and password that is correct, the server will reply with the OK 200 command and show that the Credentials are on File



```
Please enter a username and password to login into the secure server.
Usernames or Passwords with spaces will be concatenated!!

Username: blake
Password: siue999
S --> C: RTSP/2.0 403 Forbidden
         CSeq: 2
         Date: Tue, 28 Apr 2020 13:52:56
         WWW-Authenticate: Basic realm="CS447S20" Username in File but not password


Please enter a username and password to login into the secure server.
Usernames or Passwords with spaces will be concatenated!!

Username: blake
Password: siue888
S --> C: RTSP/2.0 410 Gone
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Client$
```

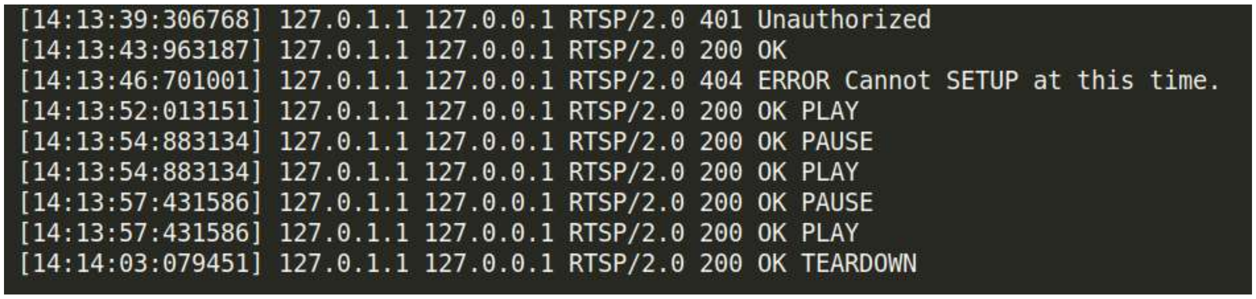
Figure 2.4: If the user enters a username that is correct but a password that is incorrect twice, the server will close.

Sever Log File and Credentials File



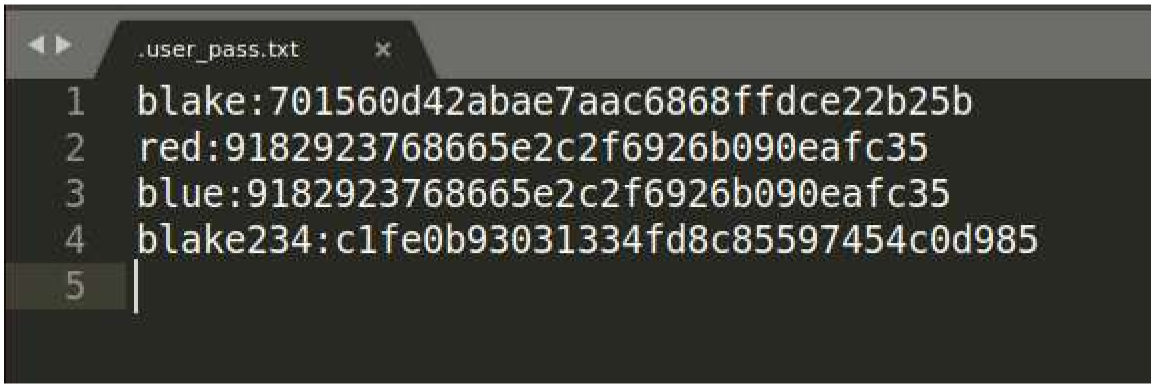
```
SETUP
S --> C: RTSP/2.0 401 Unauthorized
         CSeq: 1
         Date: Tue, 28 Apr 2020 14:13:39
         WWW-Authenticate: Basic realm="CS447S20"


Please enter a username and password to login into the secure server.
Usernames or Passwords with spaces will be concatenated!!

Username: blake234
Password: password
S --> C: RTSP/2.0 200 OK
         CSeq: 2
         Date: Tue, 28 Apr 2020 14:13:43
         WWW-Authenticate: Basic realm="CS447S20" Creds in File
------------------------------------------------------------

SETUP
S --> C: 404 ERROR Cannot SETUP at this time.

PLAY
S --> C: RTSP/2.0 200 OK
         CSeq: 3
         Date: Tue, 28 Apr 2020 14:13:52

PAUSE
S --> C: RTSP/2.0 200 OK
         CSeq: 4
         Date: Tue, 28 Apr 2020 14:13:54

PLAY
S --> C: RTSP/2.0 200 OK
         CSeq: 5
         Date: Tue, 28 Apr 2020 14:13:56

PAUSE
S --> C: RTSP/2.0 200 OK
         CSeq: 6
         Date: Tue, 28 Apr 2020 14:13:57

PLAY
S --> C: RTSP/2.0 200 OK
         CSeq: 7
         Date: Tue, 28 Apr 2020 14:13:58

TEARDOWN
S --> C: TEARDOWN RTSP/2.0 200 OK
         CSeq: 8
         Date: Tue, 28 Apr 2020 14:14:03
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Client$
```

Figure 3.1: shows the commands the server sends to the client.



```
[14:13:39:306768] 127.0.1.1 127.0.0.1 RTSP/2.0 401 Unauthorized
[14:13:43:963187] 127.0.1.1 127.0.0.1 RTSP/2.0 200 OK
[14:13:46:701001] 127.0.1.1 127.0.0.1 RTSP/2.0 404 ERROR Cannot SETUP at this time.
[14:13:52:013151] 127.0.1.1 127.0.0.1 RTSP/2.0 200 OK PLAY
[14:13:54:883134] 127.0.1.1 127.0.0.1 RTSP/2.0 200 OK PAUSE
[14:13:54:883134] 127.0.1.1 127.0.0.1 RTSP/2.0 200 OK PLAY
[14:13:57:431586] 127.0.1.1 127.0.0.1 RTSP/2.0 200 OK PAUSE
[14:13:57:431586] 127.0.1.1 127.0.0.1 RTSP/2.0 200 OK PLAY
[14:14:03:079451] 127.0.1.1 127.0.0.1 RTSP/2.0 200 OK TEARDOWN
```

Figure 3.2: shows the logged commands the server sent to the client

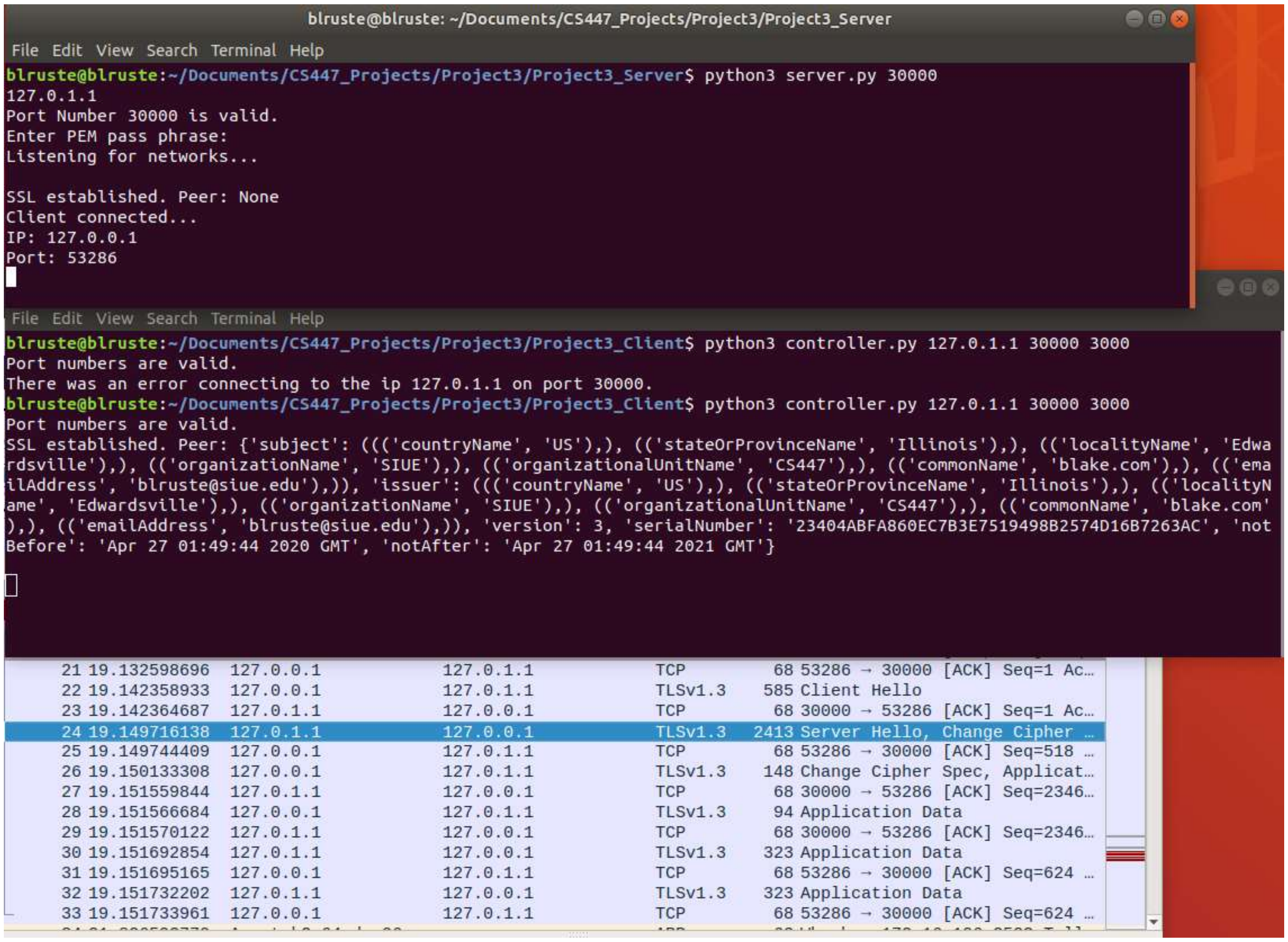


```
blake:701560d42abae7aac6868ffdce22b25b
red:9182923768665e2c2f6926b090eafc35
blue:9182923768665e2c2f6926b090eafc35
blake234:c1fe0b93031334fd8c85597454c0d985
```

Figure 3.2: shows the credentials that are stored in the .user_pass file. The passwords are encoded in a md5 hash function.

Note: I took the extra credit option to use a hash function instead of base 64 when storing passwords

Wireshark with TLSv1.3



blruste@blruste: ~/Documents/CS447_Projects/Project3/Project3_Server

File Edit View Search Terminal Help

```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Server$ python3 server.py 30000
127.0.1.1
Port Number 30000 is valid.
Enter PEM pass phrase:
Listening for networks...

SSL established. Peer: None
Client connected...
IP: 127.0.0.1
Port: 53286
```

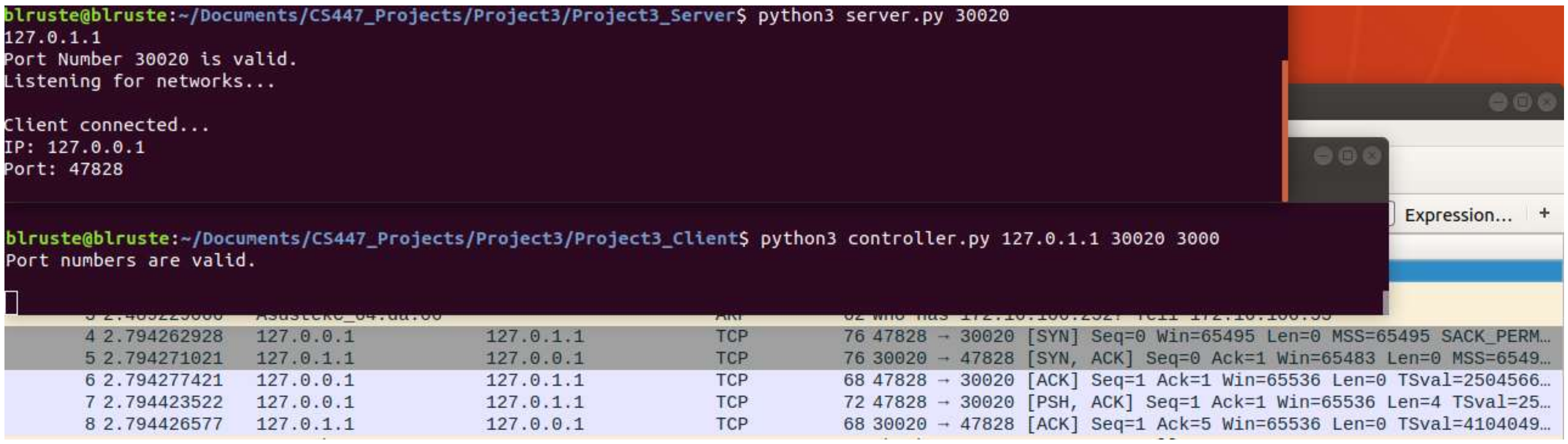
File Edit View Search Terminal Help

```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Client$ python3 controller.py 127.0.1.1 30000 3000
Port numbers are valid.
There was an error connecting to the ip 127.0.1.1 on port 30000.
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Client$ python3 controller.py 127.0.1.1 30000 3000
Port numbers are valid.
SSL established. Peer: {'subject': ((('countryName', 'US'),), (('stateOrProvinceName', 'Illinois'),), (('localityName', 'Edwa
rdsville'),), (('organizationName', 'SIUE'),), (('organizationalUnitName', 'CS447'),), (('commonName', 'blake.com'),), (('ema
ilAddress', 'blruste@siue.edu'),)), 'issuer': ((('countryName', 'US'),), (('stateOrProvinceName', 'Illinois'),), (('localityN
ame', 'Edwardsville'),), (('organizationName', 'SIUE'),), (('organizationalUnitName', 'CS447'),), (('commonName', 'blake.com'
),), (('emailAddress', 'blruste@siue.edu'),)), 'version': 3, 'serialNumber': '23404ABFA860EC7B3E7519498B2574D16B7263AC', 'not
Before': 'Apr 27 01:49:44 2020 GMT', 'notAfter': 'Apr 27 01:49:44 2021 GMT'}
```

```
21 19.132598696  127.0.0.1      127.0.1.1      TCP       68 53286 → 30000 [ACK] Seq=1 Ac…
22 19.142358933  127.0.0.1      127.0.1.1      TLSv1.3  585 Client Hello
23 19.142364687  127.0.1.1      127.0.0.1      TCP       68 30000 → 53286 [ACK] Seq=1 Ac…
24 19.149716138  127.0.1.1      127.0.0.1      TLSv1.3 2413 Server Hello, Change Cipher …
25 19.149744409  127.0.0.1      127.0.1.1      TCP       68 53286 → 30000 [ACK] Seq=518 …
26 19.150133308  127.0.0.1      127.0.1.1      TLSv1.3  148 Change Cipher Spec, Applicat…
27 19.151559844  127.0.1.1      127.0.0.1      TCP       68 30000 → 53286 [ACK] Seq=2346…
28 19.151566684  127.0.0.1      127.0.1.1      TLSv1.3   94 Application Data
29 19.151570122  127.0.1.1      127.0.0.1      TCP       68 30000 → 53286 [ACK] Seq=2346…
30 19.151692854  127.0.1.1      127.0.0.1      TLSv1.3  323 Application Data
31 19.151695165  127.0.0.1      127.0.1.1      TCP       68 53286 → 30000 [ACK] Seq=624 …
32 19.151732202  127.0.1.1      127.0.0.1      TLSv1.3  323 Application Data
33 19.151733961  127.0.0.1      127.0.1.1      TCP       68 53286 → 30000 [ACK] Seq=624 …
```

Figure 4.1: This shows the server accepting the controller's certification through Wireshark. The port numbers show that this was happening on the controller and server.

Wireshark with no TLSv1.3



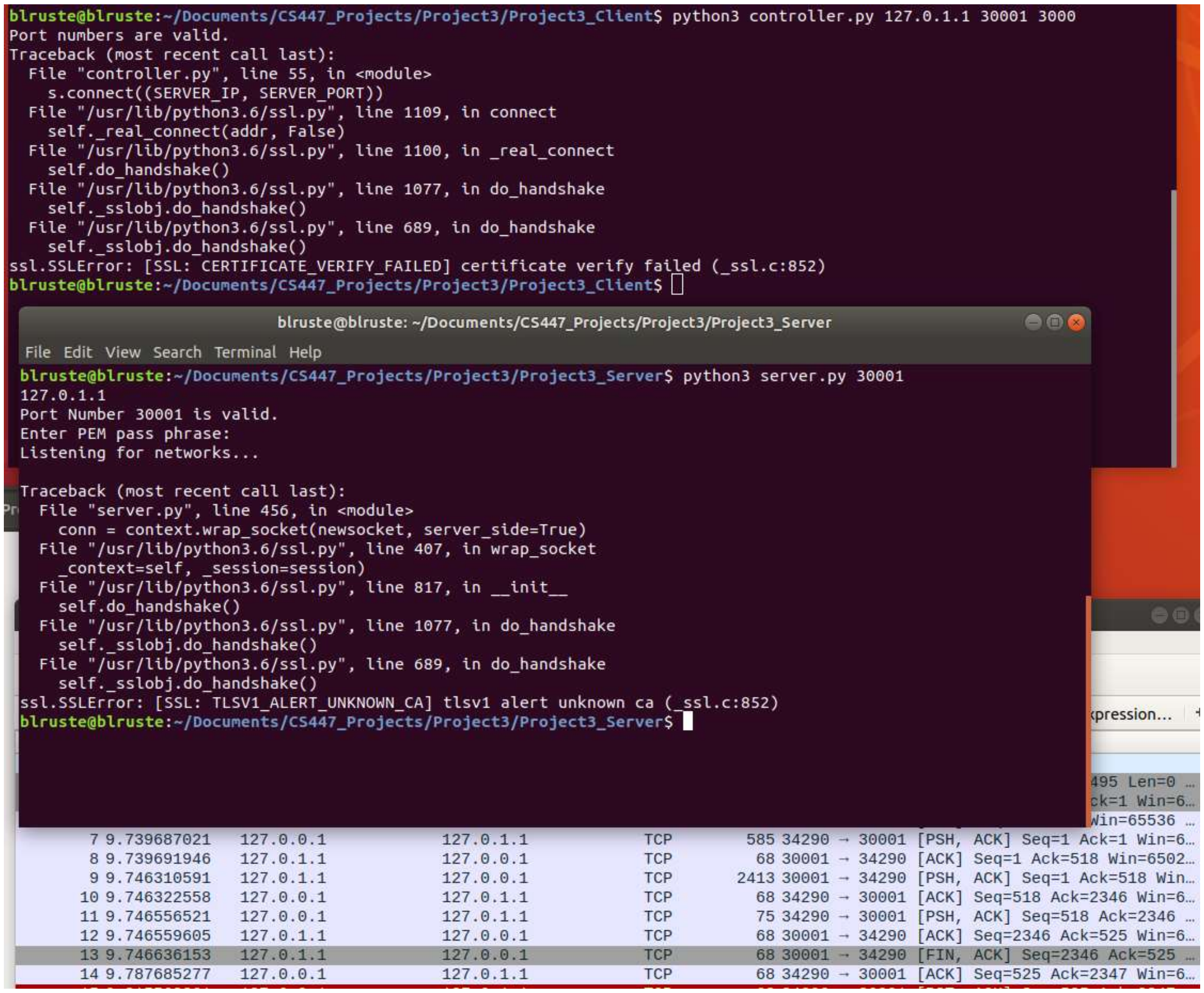
```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Server$ python3 server.py 30020
127.0.1.1
Port Number 30020 is valid.
Listening for networks...

Client connected...
IP: 127.0.0.1
Port: 47828
```

```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Client$ python3 controller.py 127.0.1.1 30020 3000
Port numbers are valid.
```

```
4 2.794262928   127.0.0.1   127.0.1.1   TCP   76 47828 → 30020 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM…
5 2.794271021   127.0.1.1   127.0.0.1   TCP   76 30020 → 47828 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549…
6 2.794277421   127.0.0.1   127.0.1.1   TCP   68 47828 → 30020 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2504566…
7 2.794423522   127.0.0.1   127.0.1.1   TCP   72 47828 → 30020 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=25…
8 2.794426577   127.0.1.1   127.0.0.1   TCP   68 30020 → 47828 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=4104049…
```

Figure 4.2: This shows that there is no TLS communication between the controller and client, only TCP

Wireshark With Failed Wrong Server Cert TLSv1.3



```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Client$ python3 controller.py 127.0.1.1 30001 3000
Port numbers are valid.
Traceback (most recent call last):
  File "controller.py", line 55, in <module>
    s.connect((SERVER_IP, SERVER_PORT))
  File "/usr/lib/python3.6/ssl.py", line 1109, in connect
    self._real_connect(addr, False)
  File "/usr/lib/python3.6/ssl.py", line 1100, in _real_connect
    self.do_handshake()
  File "/usr/lib/python3.6/ssl.py", line 1077, in do_handshake
    self._sslobj.do_handshake()
  File "/usr/lib/python3.6/ssl.py", line 689, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:852)
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Client$
```

```
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Server$ python3 server.py 30001
127.0.1.1
Port Number 30001 is valid.
Enter PEM pass phrase:
Listening for networks...

Traceback (most recent call last):
  File "server.py", line 456, in <module>
    conn = context.wrap_socket(newsocket, server_side=True)
  File "/usr/lib/python3.6/ssl.py", line 407, in wrap_socket
    _context=self, _session=session)
  File "/usr/lib/python3.6/ssl.py", line 817, in __init__
    self.do_handshake()
  File "/usr/lib/python3.6/ssl.py", line 1077, in do_handshake
    self._sslobj.do_handshake()
  File "/usr/lib/python3.6/ssl.py", line 689, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: TLSV1_ALERT_UNKNOWN_CA] tlsv1 alert unknown ca (_ssl.c:852)
blruste@blruste:~/Documents/CS447_Projects/Project3/Project3_Server$
```

```
     7 9.739687021    127.0.0.1     127.0.1.1     TCP     585 34290 → 30001 [PSH, ACK] Seq=1 Ack=1 Win=6…
     8 9.739691946    127.0.1.1     127.0.0.1     TCP      68 30001 → 34290 [ACK] Seq=1 Ack=518 Win=6502…
     9 9.746310591    127.0.1.1     127.0.0.1     TCP    2413 30001 → 34290 [PSH, ACK] Seq=1 Ack=518 Win…
    10 9.746322558    127.0.0.1     127.0.1.1     TCP      68 34290 → 30001 [ACK] Seq=518 Ack=2346 Win=6…
    11 9.746556521    127.0.0.1     127.0.1.1     TCP      75 34290 → 30001 [PSH, ACK] Seq=518 Ack=2346 …
    12 9.746559605    127.0.1.1     127.0.0.1     TCP      68 30001 → 34290 [ACK] Seq=2346 Ack=525 Win=6…
    13 9.746636153    127.0.1.1     127.0.0.1     TCP      68 30001 → 34290 [FIN, ACK] Seq=2346 Ack=525 …
    14 9.787685277    127.0.0.1     127.0.1.1     TCP      68 34290 → 30001 [ACK] Seq=525 Ack=2347 Win=6…
```

Figure 4.3: This shows the TLS connection between the server and controller has failed because the cert is invalid

Bugs

There may be a few bugs for special cases throughout the program, but I think I have error checked about 99% of them. Streaming still does not work perfectly from last project as well.

Conclusion

This project was meant to give a better understanding of how the TLS protocol works, and how a user authentication would work between a client and a server. I learned a lot from this project and was interesting to me because security is very important in everything, and we now know how it works over a network. This program in general was not as hard as the last project because it had a clearer direction and less technical steps required. This project took about half as long as the last one not including testing and documentation. I think overall this class gave me a better understanding of how protocols work, the flow of peer to peer communication works, and has enhanced my programming skillset in python and socket programming. The 3 projects that are assigned in this class are well defined and when once all complete gives an all around better understanding of network communication.