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

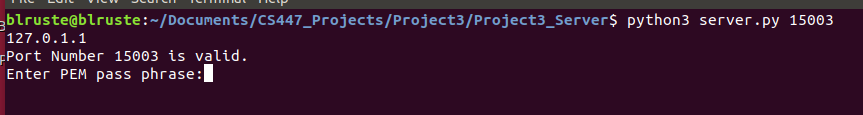
Design Choices

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:

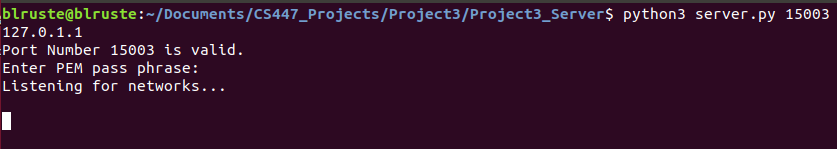
Establish TLS secure channel 🡪 SETUP command 🡪 User Authentication 🡪 PLAY || PAUSE || TEARDOWN.

Program Run-through

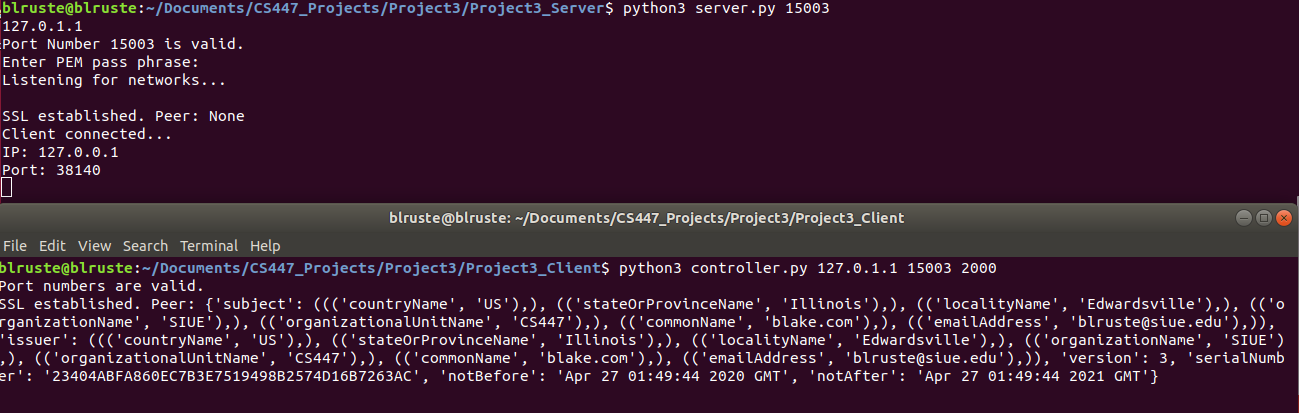
Client-Server secure channel



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

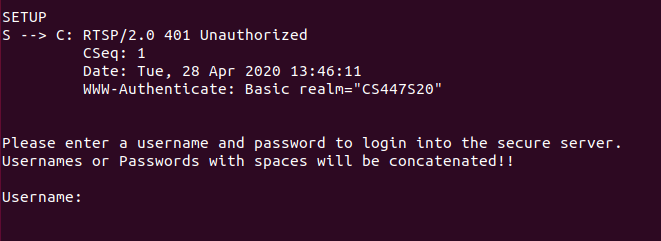


**Figure 1.2:** When the pass phrase is entered, the server will now listen for networks trying to connect to it

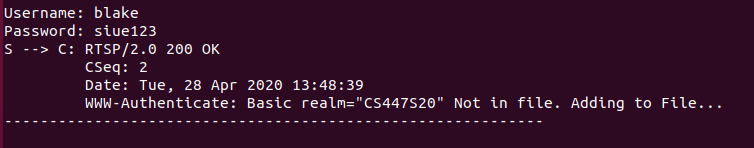


**Figure 1.3:** Now the controller will be able to connect to the server. When the controller connects it will return that cert that it used to connect to the server

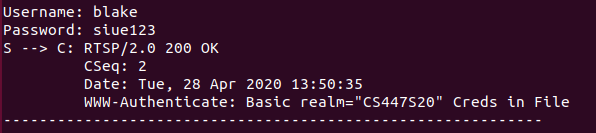
User Authentication



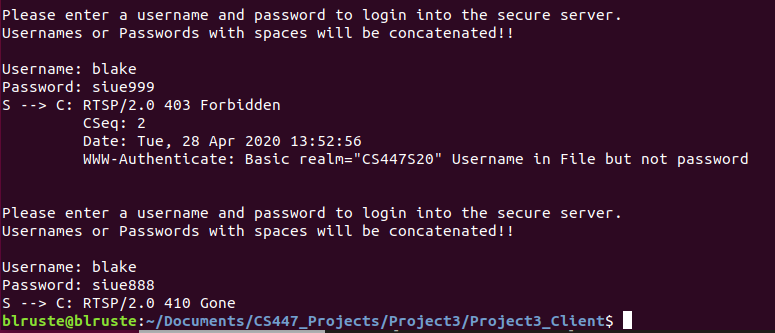
**Figure 2.1:** The controller will issue the SETUP command and the server will reply with a 401 Unauthorized code. The controller will now be able to enter a username and password



**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

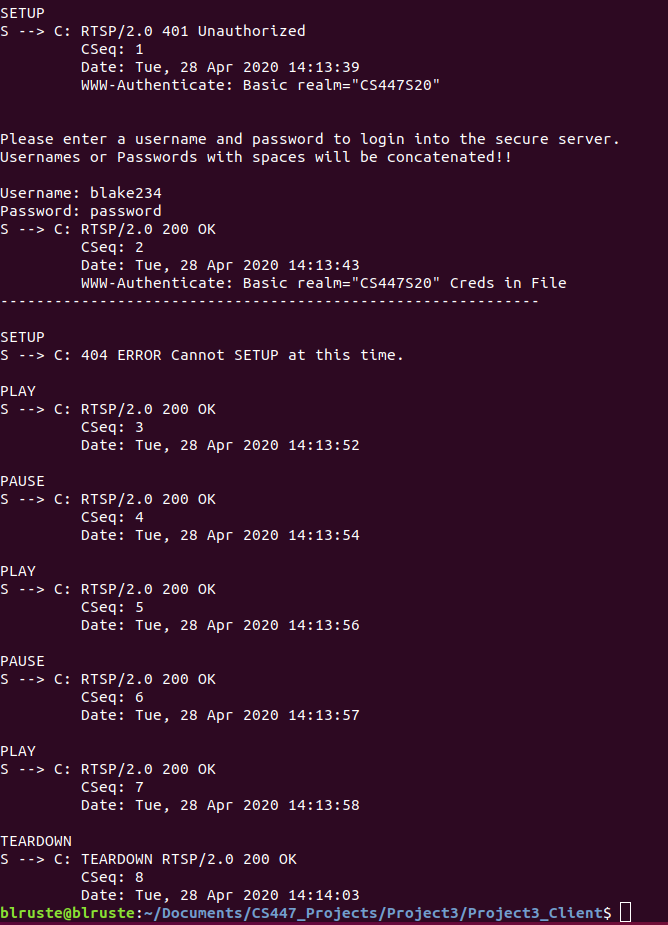


**Figure 2.3:** If the user enters a username and password that is correct, the server will reply with the OK 200 command and show that the Credentials are on File

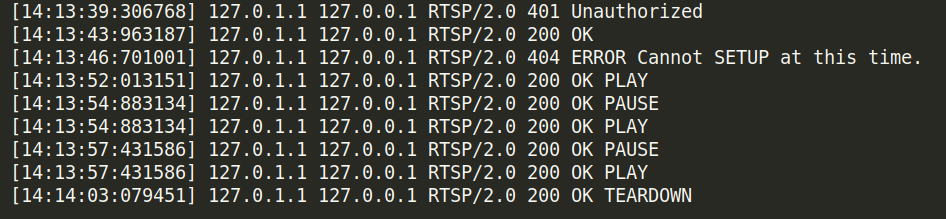


**Figure 2.4:** If the user enters a username that is correct but a password that is incorrect twice, the server will close.

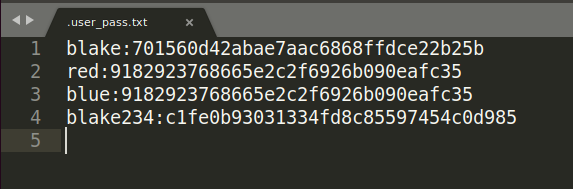
Sever Log File and Credentials File



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

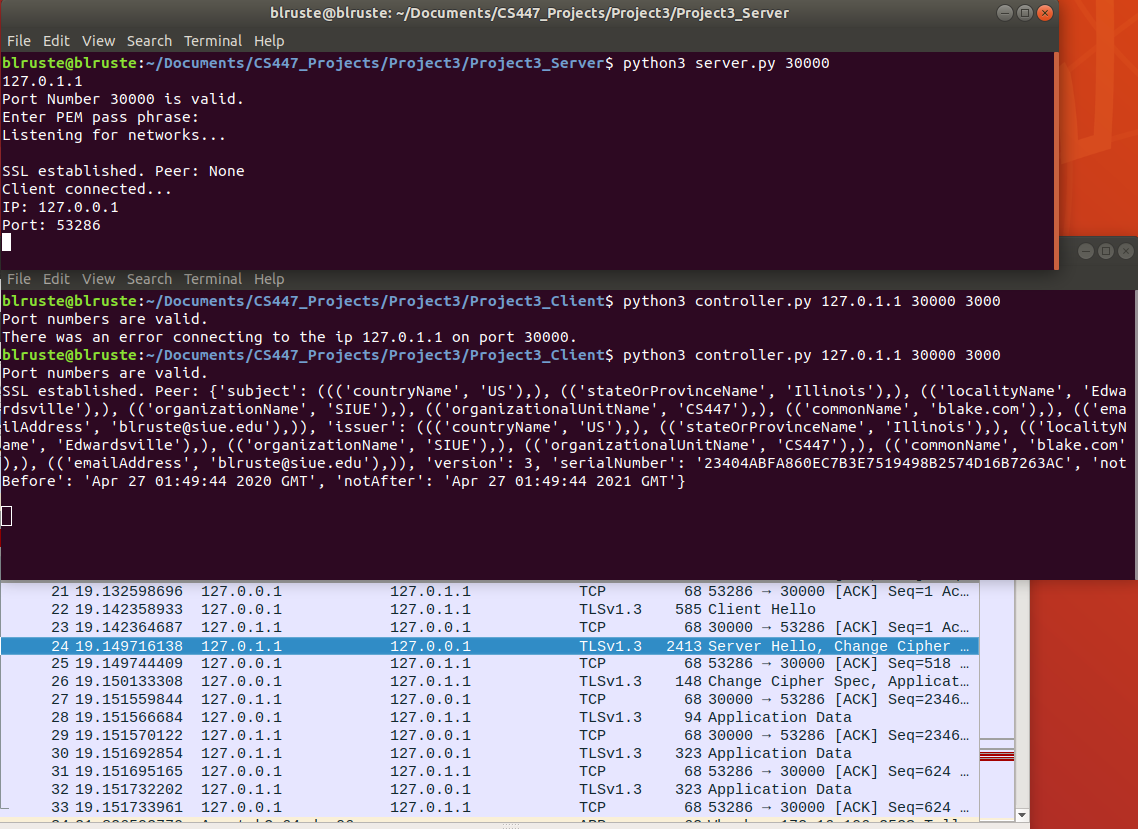


**Figure 3.2:** shows the logged commands the server sent to the client

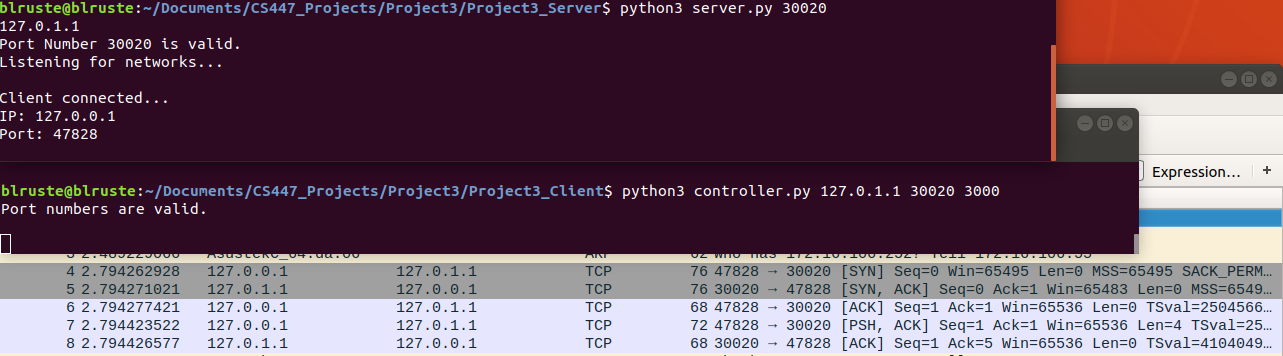


**Figure 3.2:** shows the credentials that are stored in the .user\_pass file. The passwords are encoded in a md5 hash function.

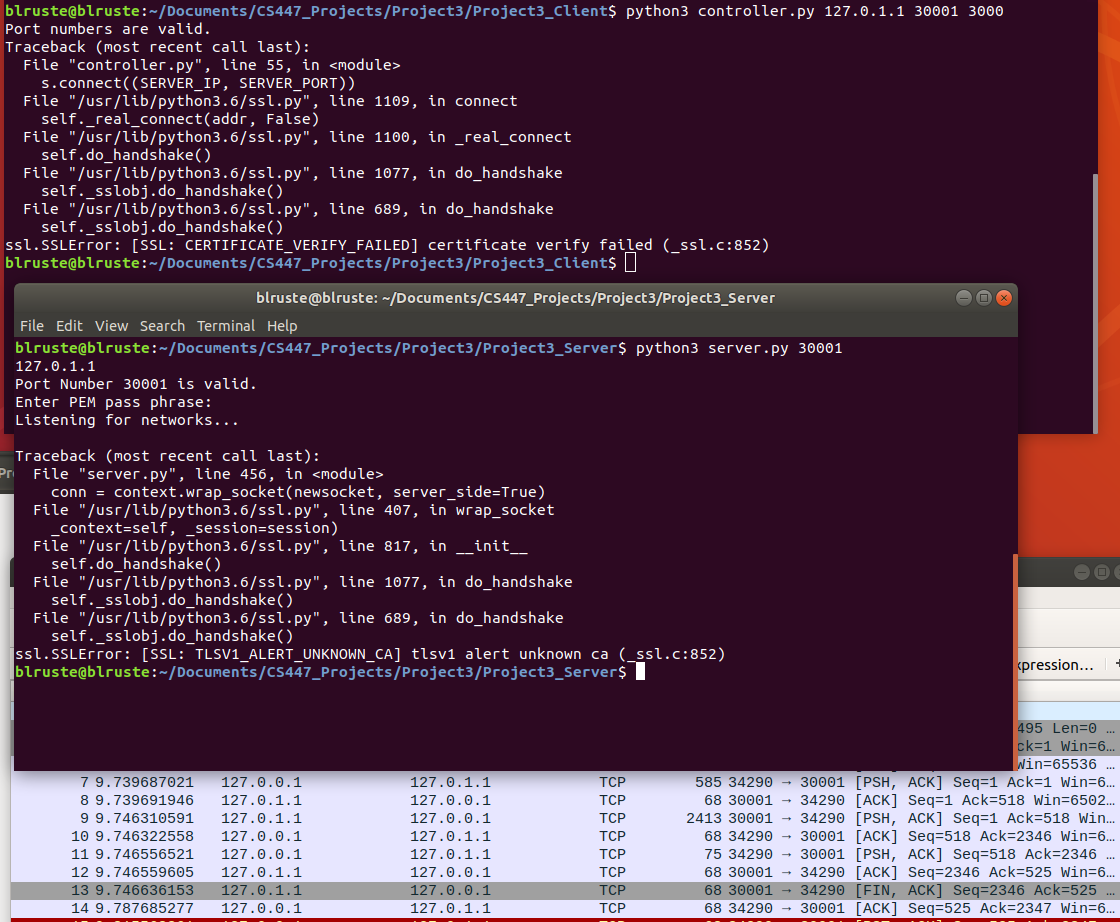
**Note:** I took the extra credit option to use a hash function instead of base 64 when storing passwords

Wireshark with TLSv1.3

**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

**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

**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.