Programming Assignment 03

Secure out of band socket protocol

Ben Hubler
Networking 447
December 3, 2018

Ben Hubler
Networking 447 F18
12/3/2018

Introduction:

Program 03 implements a make shift version of the RTSP out of band protocol to aid Captain Haddock monitor oxygen, temperature, and pressure in his quest to find the last unicorn. Unfortunately, Captain Haddock has fallen victim to man-in-the-middle attacks and unauthorized use of his sensor array. The primary objectives for program 3 is to create an application that utilizes multithreading to facilitate multiple client pairs in a secure manner and allow only authenticated users. Attempt to make the program stable utilizing error control with try/catch blocks. Exit client pairs cleanly with proper thread management. Provide an an easy to use interface, all while learning more about creating protocols using the sockets library and the OpenSSL library.

When successfully complied, there are three executables, server.app, controller.app, and receiver.app. The server and receiver should be running before the controller. The server is capable of accepting multiple clients that act independently of one another to receive the same sensor data.

Design:

Program 03 is written in c++, allowing for easy multithreading and utilization of the C sockets API. The application is simple and easy to use. Start the server, receiver, and then the controller. The controller and receiver must be on the same machine.

Once the controller is running, there is no need to interface with either the server or receiver.

To start receiving data, first run “setupauth”, which sends the following:

```
    Setup rtsp://<hostname> rtsp/2.0
    Cseq:0
    Sensor:*
    Authorization: Basic <Base64Encoded <username:password>>
```

To start receiving data, run “play”, which sends the following:

```
    Play rtsp://<hostname> rtsp/2.0
    Cseq:1
    Sensor:*
```

To pause data, run “pause”, which sends the following:

```
    Pause rtsp://<hostname> rtsp/2.0
    Cseq:2
```

To terminate both clients, run “teardown”, which sends the following:

```
    Teardown rtsp://<hostname> rtsp/2.0
    Cseq:3
```

All four commands can be typed out in long form as well.

The following are possible responses:

RTSP/2.0 200 OK
  Used to show valid input received and processed.
RTSP/2.0 400 Bad Request
  Used as a “catch all” for errors not caught be a more specific error
RTSP/2.0 401 Unauthorized

Used to let the user know they have not been authenticated

RTSP/2.0 403 Forbidden

Used to inform the user of a failed authentication attempt, after two attempts, the server will disconnect the client with a 410 response

RTSP/2.0 410 Gone

Sent to client after 2 failed authentication attempts, signifying connection has been closed

RTSP/2.0 456 Header Field Not Valid for Resource

Used to determine if the CSeq number is valid and present

RTSP/2.0 461 Unsupported Transport

Used to determine if Setup has a valid transport header

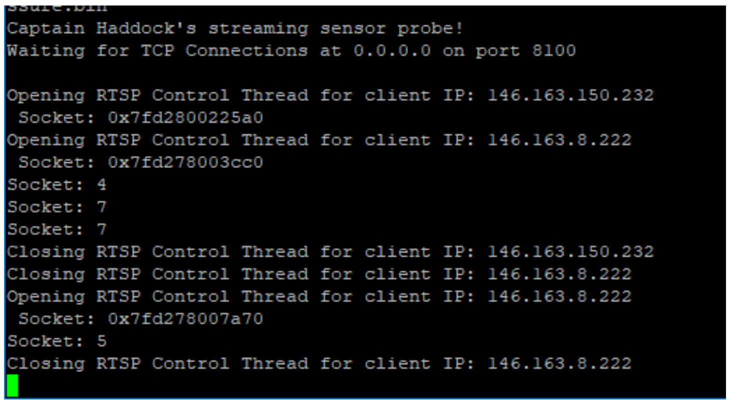
Extra Credit:

User friendliness has been improved by offering the use of quick commands and an advanced help information display on initial startup of the controller.

Output:

Wireshark: No SSL see attached file: PR03-WireShark.pcap
Wireshark: SSL proof, see attached file: PR03-SSL-WireShark.pcap

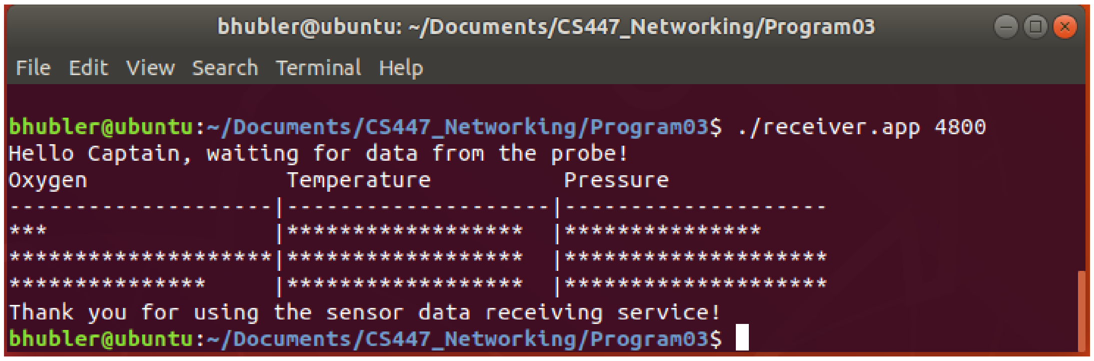
Server:



```
ssure.bin
Captain Haddock's streaming sensor probe!
Waiting for TCP Connections at 0.0.0.0 on port 8100

Opening RTSP Control Thread for client IP: 146.163.150.232
 Socket: 0x7fd2800225a0
Opening RTSP Control Thread for client IP: 146.163.8.222
 Socket: 0x7fd278003cc0
Socket: 4
Socket: 7
Socket: 7
Closing RTSP Control Thread for client IP: 146.163.150.232
Closing RTSP Control Thread for client IP: 146.163.8.222
Opening RTSP Control Thread for client IP: 146.163.8.222
 Socket: 0x7fd278007a70
Socket: 5
Closing RTSP Control Thread for client IP: 146.163.8.222
```

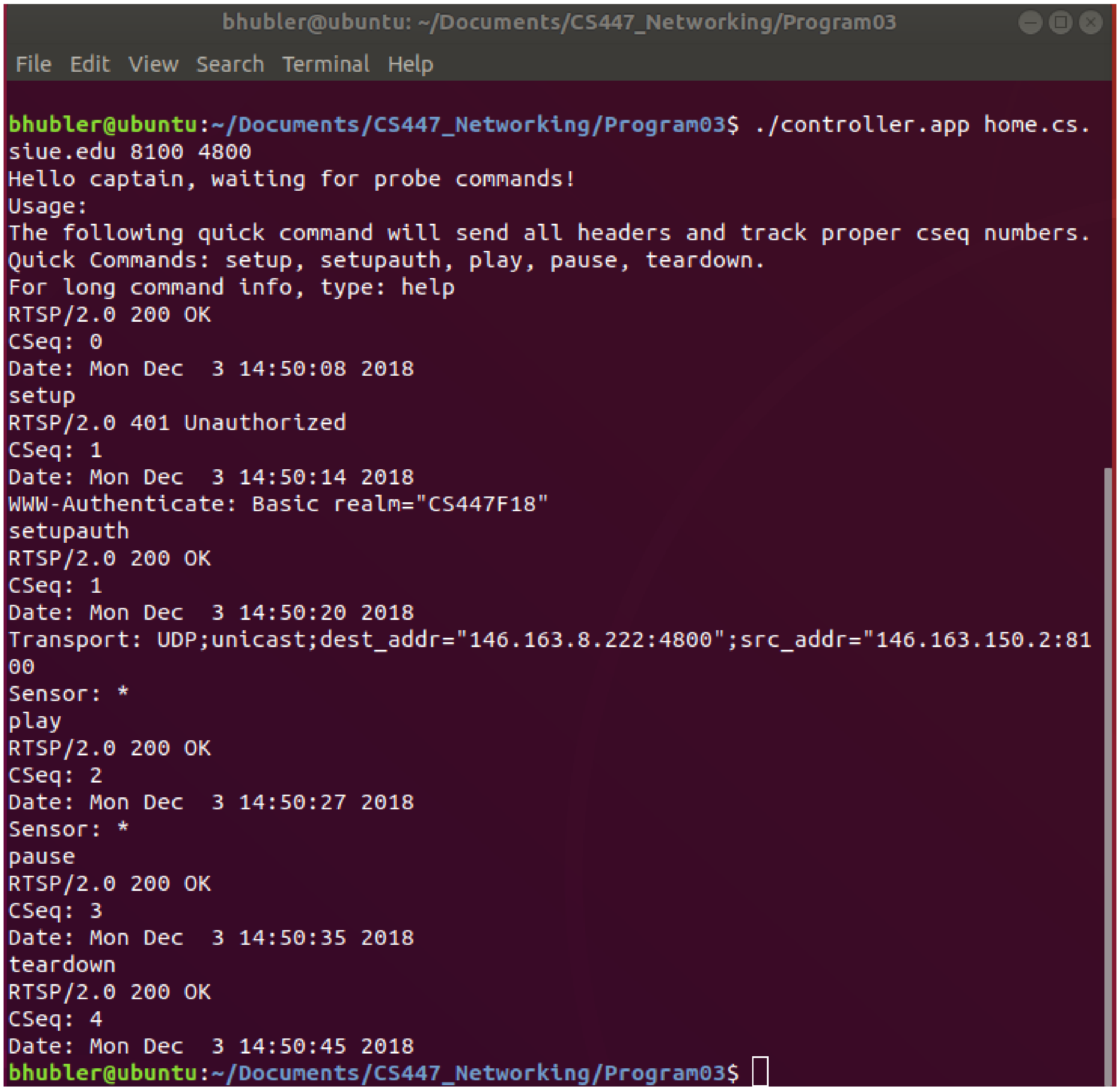
Receiver:


bhubler@ubuntu: ~/Documents/CS447_Networking/Program03
File Edit View Search Terminal Help

```
bhubler@ubuntu:~/Documents/CS447_Networking/Program03$ ./receiver.app 4800
Hello Captain, waiting for data from the probe!
Oxygen              Temperature          Pressure
--------------------|--------------------|--------------------
***                 |******************  |***************
********************|******************  |********************
***************     |******************  |********************
Thank you for using the sensor data receiving service!
bhubler@ubuntu:~/Documents/CS447_Networking/Program03$
```

Ben Hubler
Networking 447 F18
12/3/2018

Controller:


bhubler@ubuntu: ~/Documents/CS447_Networking/Program03
File Edit View Search Terminal Help

```
bhubler@ubuntu:~/Documents/CS447_Networking/Program03$ ./controller.app home.cs.
siue.edu 8100 4800
Hello captain, waiting for probe commands!
Usage:
The following quick command will send all headers and track proper cseq numbers.
Quick Commands: setup, setupauth, play, pause, teardown.
For long command info, type: help
RTSP/2.0 200 OK
CSeq: 0
Date: Mon Dec  3 14:50:08 2018
setup
RTSP/2.0 401 Unauthorized
CSeq: 1
Date: Mon Dec  3 14:50:14 2018
WWW-Authenticate: Basic realm="CS447F18"
setupauth
RTSP/2.0 200 OK
CSeq: 1
Date: Mon Dec  3 14:50:20 2018
Transport: UDP;unicast;dest_addr="146.163.8.222:4800";src_addr="146.163.150.2:81
00
Sensor: *
play
RTSP/2.0 200 OK
CSeq: 2
Date: Mon Dec  3 14:50:27 2018
Sensor: *
pause
RTSP/2.0 200 OK
CSeq: 3
Date: Mon Dec  3 14:50:35 2018
teardown
RTSP/2.0 200 OK
CSeq: 4
Date: Mon Dec  3 14:50:45 2018
bhubler@ubuntu:~/Documents/CS447_Networking/Program03$ 
```

Ben Hubler
Networking 447 F18
12/3/2018

Summary:

During the implementation of Program 03, I only ran into one major issue. During the implementation of OpenSSL, I ran into error after error. I first started troubleshooting the issues by connecting to the server using openssl and found that I was indeed only accepting TLS1.2. All functions seemed to work. Moving onto the client, I verified I was using TLS1.2, and still continued receiving errors. Eventually, I found that the way I implemented the SSL connection procedure, was at the core of all my issues. Essentially, I was attempting to create 2 SSL connections over one socket(I put the SSL connect in both threads, sending and receiving). The ultimate solution was to abstract out the SSL connect process into the function that calls each thread(send and receive).

Adding authentication was very straight forward. It also helped to resolve an issue from Program02, now play, pause, and teardown cannot be called unless setup has been called and properly authenticated. In addition to using basic authentication in the setup call, the server will close the socket of any client that has 2 failed authentication attempts.

Increased program stability is achieved by utilizing try/catch/throw controls and testing several different scenarios. Client pairs will exit independently of other pairs when the teardown command is sent. The improved interface was done by printing out instructions and adding a set of quick commands. All functionality in the program specification has been implemented and functions well. In addition, for the extra credit options, the user friendliness was improved greatly over a standard “telnet” type of interface. All in all, I feel this program assignment went very well and I have a solid understanding of multithreading, openssl, and socket programming.