

Programming Assignment 03

Secure out of band socket protocol

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, create a server log, 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 compiled, 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.

*Note: The SSL certificate must be named bhubler_cs447.pem in the same directory as the executables. This file should also contain the public key, and private key. The file included was generated using openssl and is self signed.

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 by 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:

Server:

```

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$
```

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="146.163.150.2:8100
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$
```

Wireshark: SSL proof, see attached file: PR03-SSL-WireShark.pcap

Notice Protocol is TLSv1.2 and data is scrambled

The image shows a Wireshark capture of a network packet. The main window displays a list of captured packets. The selected packet is Frame 47, which is a TLSv1.2 Application Data packet. The packet details pane shows the following information:

- Source Port: 8100
- Destination Port: 35376
- [Stream index: 0]
- [TCP Segment Len: 98]
- Sequence number: 1649 (relative sequence number)
- [Next sequence number: 1747 (relative sequence number)]
- Acknowledgment number: 992 (relative ack number)
- 1000 = Header Length: 32 bytes (8)
- Flags: 0x018 (PSH, ACK)
- Window size value: 350

The packet bytes pane shows the raw data of the packet, which is scrambled (encrypted) TLS data. The status bar at the bottom indicates that 100 packets are displayed out of 100 captured packets.

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

PR03-WireShark.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
19	12.326023	127.0.0.1	127.0.0.1	TCP	66	8100
20	12.388782	127.0.0.1	127.0.0.1	TCP	76	55394
21	12.388799	127.0.0.1	127.0.0.1	TCP	66	8100
22	12.489464	127.0.0.1	127.0.0.1	TCP	74	55394
23	12.489494	127.0.0.1	127.0.0.1	TCP	66	8100
24	12.590261	127.0.0.1	127.0.0.1	TCP	68	55394
25	12.590298	127.0.0.1	127.0.0.1	TCP	66	8100
26	12.597604	127.0.0.1	127.0.0.1	TCP	135	8100
27	12.597670	127.0.0.1	127.0.0.1	TCP	66	55394
28	14.470872	127.0.0.1	127.0.0.1	UDP	77	49455
29	17.474001	127.0.0.1	127.0.0.1	UDP	77	49455
30	20.474675	127.0.0.1	127.0.0.1	UDP	77	49455
31	23.477981	127.0.0.1	127.0.0.1	UDP	77	49455

Frame 26: 135 bytes on wire (1080 bits), 135 bytes captured (1080 bits)

Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 8100, Dst Port: 55394, Seq: 204, Ack: 147, Len: 1000

Source Port: 8100

Destination Port: 55394

[Stream index: 0]

[TCP Segment Len: 69]

Sequence number: 204 (relative sequence number)

[Next sequence number: 273 (relative sequence number)]

Acknowledgment number: 147 (relative ack number)

1000 = Header Length: 32 bytes (8)

Flags: 0x018 (PSH, ACK)

Window size value: 342

0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.

0010 00 79 68 eb 40 00 40 06 d3 91 7f 00 00 01 7f 00 ..yh@@.....

0020 00 01 1f a4 d8 62 db 5d 59 bf b5 4d 1c e0 80 18b.]Y..M...

0030 01 56 fe 6d 00 00 01 01 08 0a af ce b8 c1 af ce ..V.m.....

0040 b8 ba 52 54 53 50 2f 32 2e 30 20 32 30 30 20 4f ..RTSP/2 .0 200 0

0050 4b 0d 0a 43 53 65 71 3a 20 31 0d 0a 44 61 74 65 K..CSeq: 1..Date

0060 3a 20 4d 6f 6e 20 4e 6f 76 20 20 35 20 31 34 3a : Mon No v 5 14:

0070 32 31 3a 32 34 20 32 30 31 38 0d 0a 53 65 6e 73 21:24 20 18..Sens

0080 6f 72 3a 20 2a 0d 0a ..or: *..

PR03-WireShark.pcap Packets: 54 · Displayed: 54 (100.0%) Profile: Default

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.