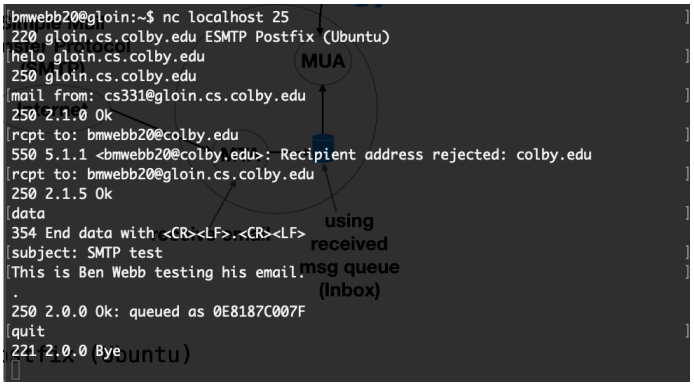
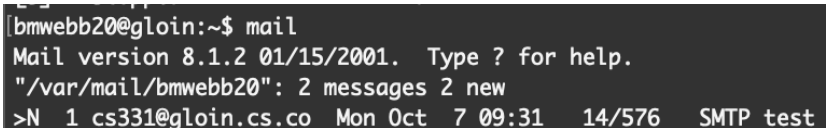
CS331 Project 1 Write Up – Ben Webb

The purpose of this project was to explore SMTP, particularly learning about how Mail Transfer Agents interact with one another.

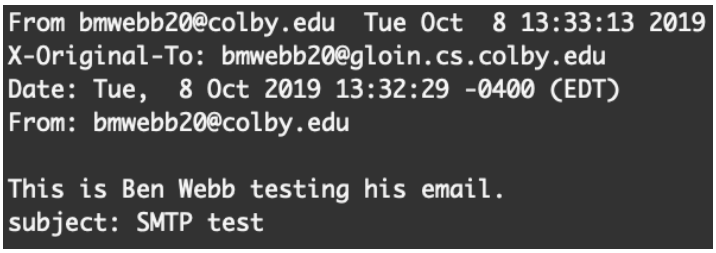
The first part of this project challenged us to explore SMTP server action. To do this I logged into gloin.cs.colby.edu and opened the local mail server with the command: `nc localhost 25` which is the command to open the Ubuntu postfix mailserver port, 25.



In this terminal interaction, I first type: `Helo gloin.cs.colby.edu` which lets the mail server know I am connecting to send mail. I then add the from address, cs331@gloin.cs.colby.edu. When I try to send an email to my email ending in @colby.edu, the address is rejected because Colby.edu is not local and . I am successful when I send an email to an address that ends in @gloin.cs.colby.edu because the message is delivered as seen below.



Then I tried to send an email from a non-local address.



This interaction was successful. This suggests that the From: field does not matter when you are running on the localhost, because all addresses will not be evaluated prior to sending. This had me a bit confused because this in combination with the first section goes against some of SMTP server action policy. Upon looking it up, it appears that the rcpt to: field is actually evaluated, and because this MTA is not connected to any external servers, the message will never be delivered.

The second part of this project was to write as short as possible of a program to have an SMTP conversation. I decided to use Python to do this. The entire code is 13 lines long.

```python
import socket
s, data = socket.socket(socket.AF_INET, socket.SOCK_STREAM), ''
s.connect(('localhost', 25))
while True:
    print(s.recv(1024).decode())
    if data.lower() == 'data':
        data = input('> ')
        s.send((data + '\r\n.\r\n').encode())
        continue
    elif data.lower() == 'quit': exit()
    data = input('> ')
    s.send((data + '\r\n').encode())
```

The program runs by being logged into gloin and typing: python3 smtpclient.py. The first line imports socket. The second and third lines create the socket and message object and then connects to the server. It then enters a while loop. There are only two special cases, data, which needs a special end character and quit, which is how the program ends. Below is an example of the code in use. User entry is denoted by the '>'.

```
bmwebb20@gloin:~/Senior Year/CS331/coding things$ python3 smtpclient.py
220 gloin.cs.colby.edu ESMTP Postfix (Ubuntu)

> HELO gloin.cs.colby.edu
250 gloin.cs.colby.edu

> mail from: cs331@gloin.cs.colby.edu
250 2.1.0 Ok

> rcpt to: bmwebb20@gloin.cs.colby.edu
250 2.1.5 Ok

> data
354 End data with <CR><LF>.<CR><LF>

> I wanted to email myself, because I, Ben Webb am a narcissist
250 2.0.0 Ok: queued as B75937C007F

> quit
221 2.0.0 Bye

You have new mail in /var/mail/bmwebb20
bmwebb20@gloin:~/Senior Year/CS331/coding things$
```

The final part of this project was to implement another program that performed the same actions as the python script, but using the linux tool expect. Expect essentially allows for there to be a response. Expect works by pairing phrases entered in the command line with answers. The code looks like this:

```
#!/usr/bin/expect -f
set timeout -1
spawn nc localhost 25
expect "220 gloin.cs.colby.edu ESMTP Postfix (Ubuntu)\r"
send -- "HELO gloin.cs.colby.edu\r"
expect "250 gloin.cs.colby.edu\r"
send -- "mail from: cs331@gloin.cs.colby.edu\r"
expect "250 2.1.0 Ok\r"
send -- "rcpt to: bmwebb20@gloin.cs.colby.edu\r"
expect "250 2.1.5 Ok\r"
send -- "data\r"
expect "354 End data with <CR><LF>.<CR><LF>\r"
send -- "subject: eMailHeader\r"
send -- "body of the email\r"
send -- ".\r"
expect "250 2.0.0 Ok: queued as *\r"
send -- "quit\r"
expect eof

```

It first connects to the localhost, and then automatically has a conversation with the mailserver. Below is an example of it in use.

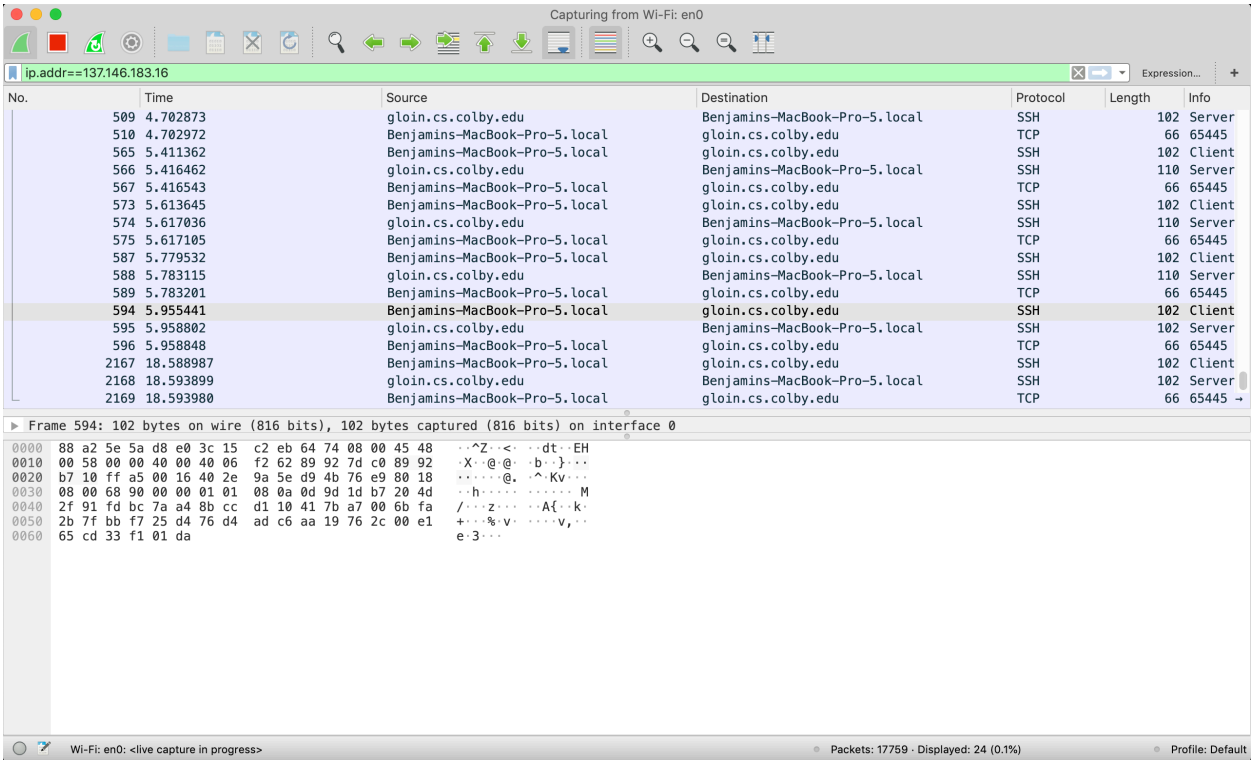
```
bmwebb20@gloin:~/Senior Year/CS331/coding things$ ./smtp.exp
spawn nc localhost 25
220 gloin.cs.colby.edu ESMTP Postfix (Ubuntu)
HELO gloin.cs.colby.edu
250 gloin.cs.colby.edu
mail from: cs331@gloin.cs.colby.edu
250 2.1.0 Ok
rcpt to: bmwebb20@gloin.cs.colby.edu
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
subject: eMailHeader
body of the email
.
250 2.0.0 Ok: queued as 315E37C007F
quit
221 2.0.0 Bye
```

Extensions

For my first extension I decided to play with using SSL Authentification to send an email on the gmail server. This was done in the file gmailclient.py and only imports socket and ssl for network programming. This process is very similar to the python file used for having an SMTP conversation but is much less compressed and there is an extra authentication step. The authentication is done using the command: `"AUTH PLAIN \0username\0password\r\n".` Below is an example of the code working:

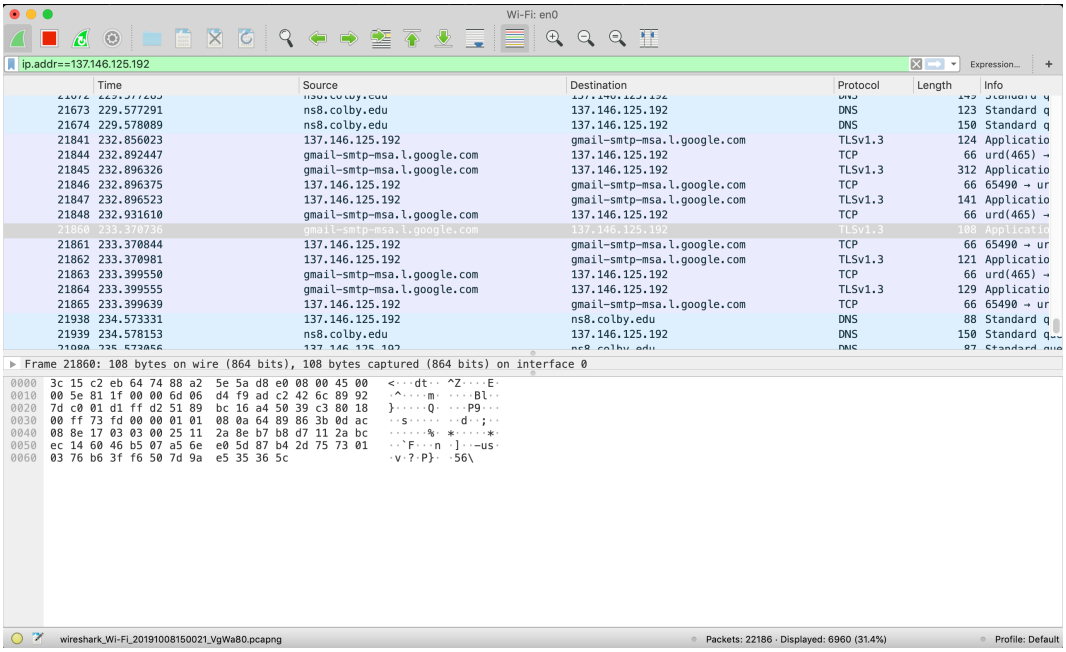
I used the package getpass to allow for the password to be typed without echoing in the terminal. I also print out all of the status messages to show how the google server responds. Below is the actual email as seen in the gmail webpage that was delivered to myself.

I also began to wonder if the authentication required would also produce an encrypted email, so I set wireshark to track gloins IP (137.146.183.16). What was interesting was the number of communication packets even to just type one character into the terminal. I paused for a couple seconds and began typing.

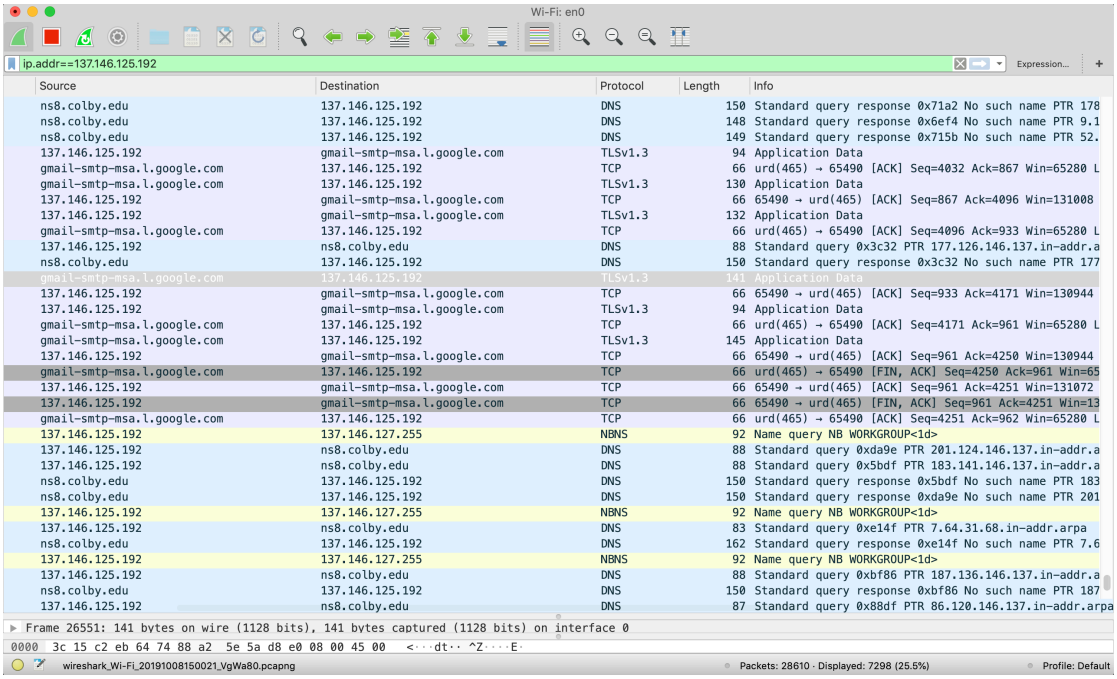


You can also see all of the SSH communications and the associated TCP level communication. But the TCP packet is never seen when coming from gloin to my computer. This is also really cool because you can see that a single key press is encapsulated to 66 bytes as TCP and 102 at SSH. Then I assume the TCP side on gloin would be decapsulated to 66 bytes.

I then wanted to look at an SMTP interaction but from my side instead of SSH so I could actually see the going ons of the message. Below is the activity captured as I was logging in via my python gmail client program.



I then wanted to better answer the problem set so I specifically captured what happen when I sent my message:



There is a continuous TLS, TCP rotation when sending the message, except the interaction ends with a set of TCP packets outside of TLSv1.3. TLS1.3 is an up to date version of TLS.