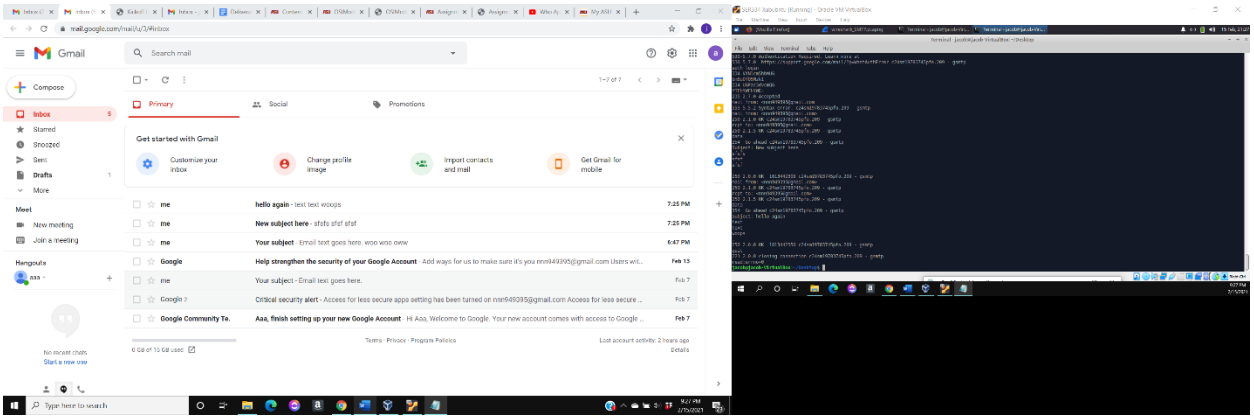
1. **SMTP**



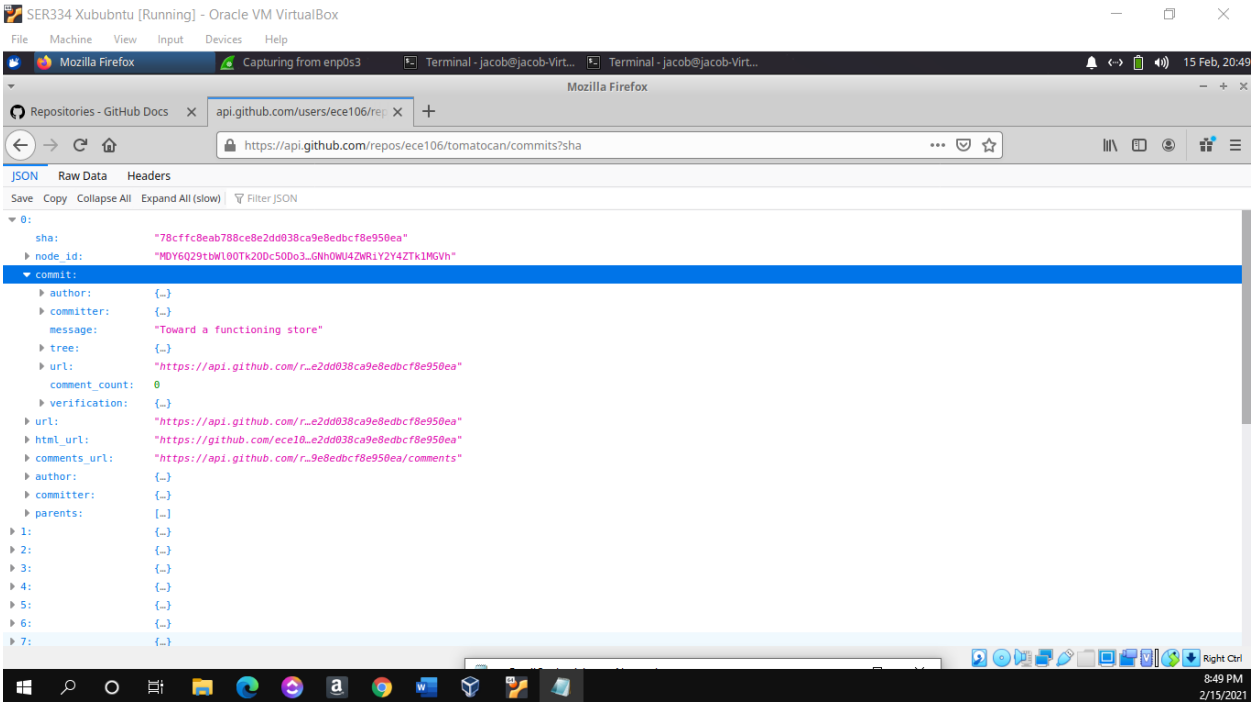
1. I used the filter tcp.port == 465 || tcp.flags.fin == 1. I filtered for TCP transactions since SMTP uses TCP as its transport protocol. I also specified port 465 since the command utilizes port 465 in sending the email over the command line. I checked for FIN flags as well to help in determining who sends the first FIN flag.
2. The three standard SMTP ports are ports 25, 587, and 465 with port 587 being the secure port. I imagine port 465 is being used since the email connection is less secure than regular email transports.
3. The echo commands piped with base64 are commands that are used to obtain the username and password of the email user in base64. These are needed for authenticating when sending the email. The openssl s_client command is used to establish an SSL/TLS client to be connected to the gmail server over port 465. The TLS client is needed in order to establish a secure connection between the client and email server. The helo command is used to establish a connection with the gmail server. The auth login command is used to allow the client to log in as a specific user on the gmail server. The echo command results are passed in as the certifications needed to login as that user. The mail from and rcpt to commands are used to establish who is sending the email and who is to receive the email. Subject is used to specify the subject line and the remaining lines up to the period fill the body of the email. The period is used to finalize the

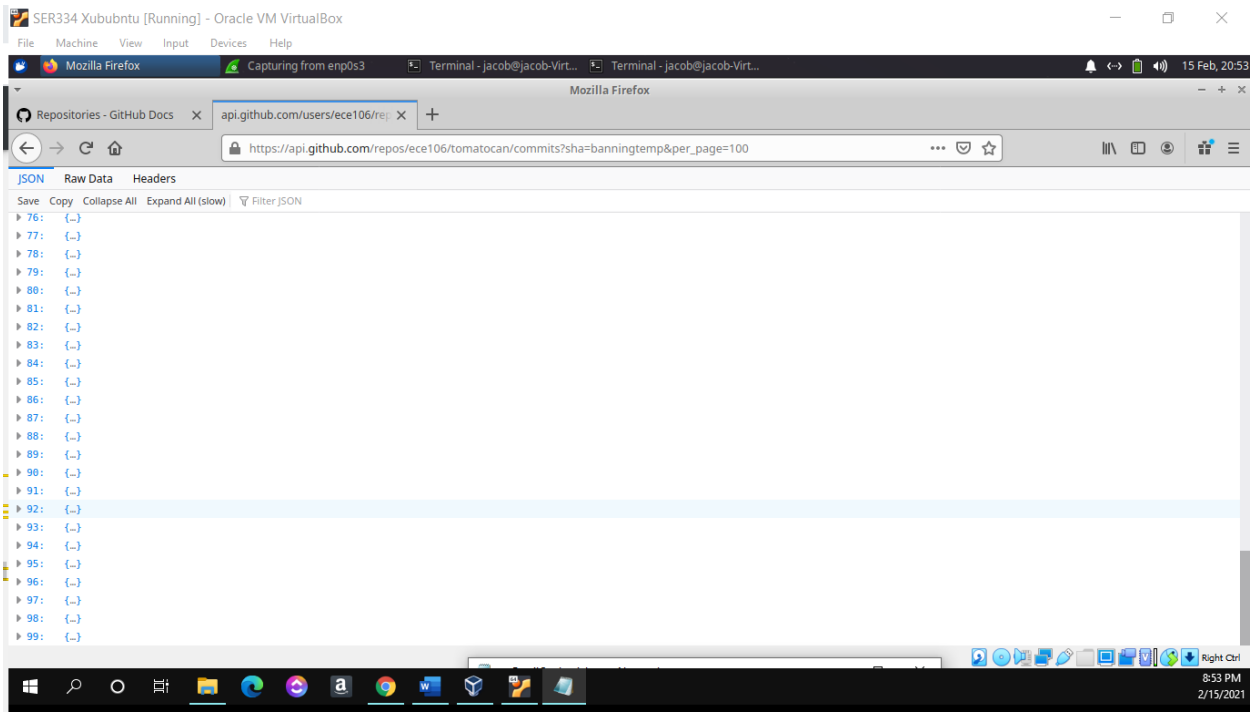
data of the email, which is then sent. The quit instruction would be used to end the TLS client communication with the gmail server.

4. For simply establishing the connection, I can see the three-way handshake of TCP as well as the TLS handshake. In short, there are 3 frames for the TCP handshake, a TLS Client Hello, and Acknowledgement, a Server Hello with a Change Cipher, another Acknowledgment, the sending of some Application Data, Another acknowledgement, a Change Cipher Spec, the sending of Application data twice in one frame, and one last acknowledgement establishing both the TCP and TLS connection.
5. The port of my local machine when sending the mail is 49888.
6. The first FIN flag is sent from port 465 to my local machine. What appears to be happening is that the quit instruction is sent over the line to the gmail server. The FIN flag is sent from the server and acknowledged on my local machine. My machine then sends a FIN flag, which I imagine is responsible for terminating the TLS client on my machine. After a few acknowledgements from port 465, the connection ceases.

Regarding the Wireshark data I collected, there is a noticeable connection loss around halfway down the sniff. I start another connection from there, send two emails and terminate.

2. HTTP



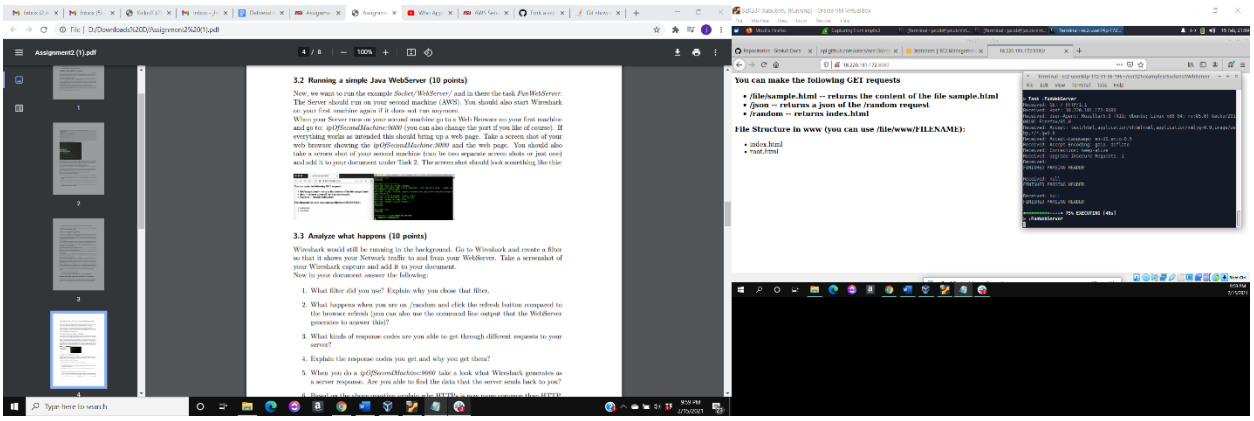


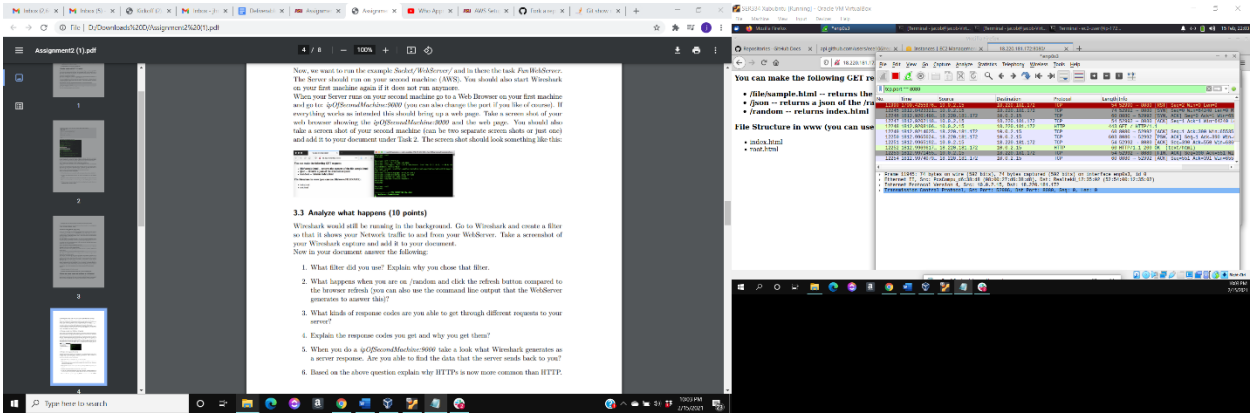
1. The two API calls used above showcase variants of the list commits API call. I first had to make a call to get the tomatocan repository belonging to ece106, which used /repos/ece101/tomatocan. I then used the list commits API call with the sha parameter to list

the 30 commits in the default master branch. I then added sha=banningtemp&per_page=100 to specify the banningtemp branch and list a maximum of 100 commits.

2. The difference between a stateless and stateful communication is that a stateless communication does not require retention of session information, like when packages are sent and received, while a stateful communication does require this retention.

3.2 Running Java Web Server





3.3 Analyze What Happens

1. I'm using the filter tcp.port == 8080 because the WebServer.java code specifies the use of port 8080. Additionally, the traffic I saw generated was using the TCP transport protocol instead of UDP, so I was able to narrow my filter further.

2. When clicking on the Random button, the TCP three-way handshake occurs followed by an HTTP GET /json, which returns a json of the /random request, which in turn displays a new random image. On the other hand, using the browser's refresh button causes the TCP three-way handshake followed by an HTTP GET /random request, which eventually causes another GET /json, causing the new image to be displayed that way. In short, the browser refresh actually has a bit more overhead in HTTP requests as opposed to pressing the random button.

3. The responses I'm able to generate are 200 OK, 400 Bad Request, 404 Not Found

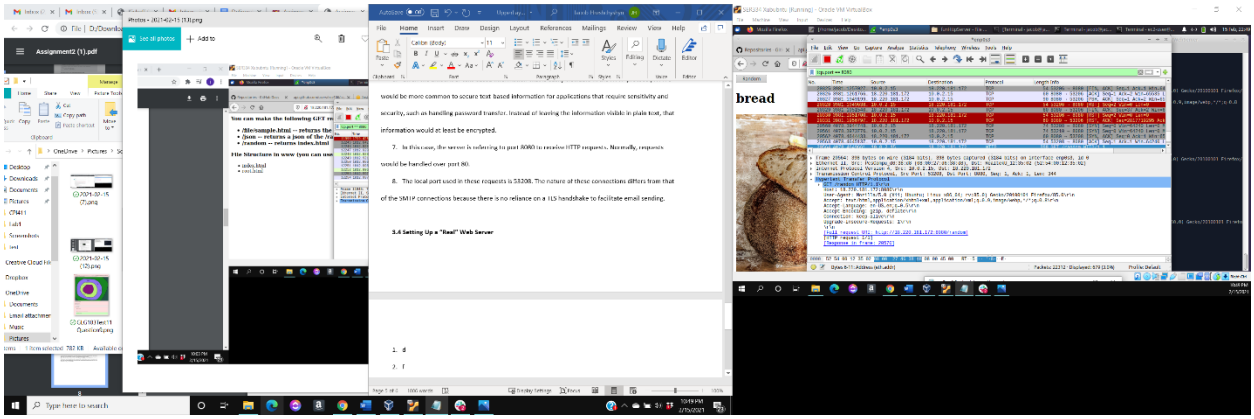
4. I get 200 for sending requests that the server can handle. I get 400 for sending in requests to things that the server cannot handle (like /random1, which is not supported). I get 404 for attempting to request access to pages or files that do not exist (e.g. file/sample/sample.html).

5. I am able to find data that the server returns through Wireshark. For example, when sending a bad request, I can click on the HTTP frame with the Bad Request message and find the Line-based text data "I am not sure what you want me to do". Similarly, the string values of image file can also be found on successful requests with pages containing images.

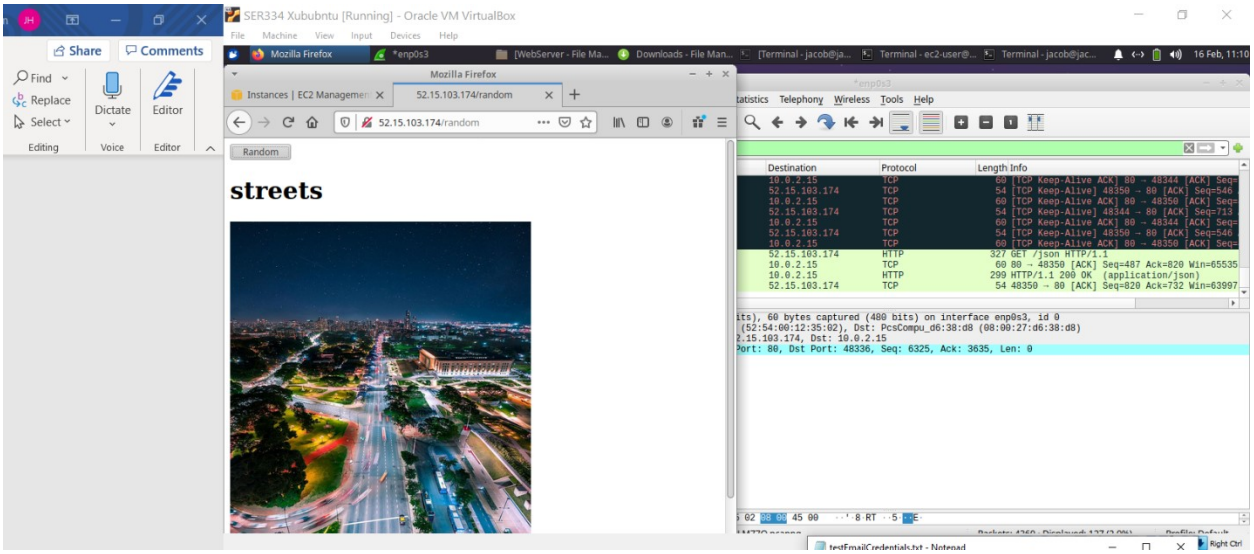
6. Given the amount of text-based information that is easily viewable in the HTTP protocol, HTTPs would be more common to secure text-based information for applications that require sensitivity and security, such as handling password transfer. Instead of leaving the information visible in plain text, that information would at least be encrypted.

7. In this case, the server is referring to port 8080 to receive HTTP requests. Normally, requests would be handled over port 80.

8. The local port used in these requests is 53208. The nature of these connections differs from that of the SMTP connections because there is no reliance on a TLS handshake to facilitate email sending.



3.4 Setting Up a “Real” Web Server



1. Now the traffic is being sent to port 80 instead of port 8080. This reflects the normal processing of an HTTP request.

2. This traffic is still HTTP since the port number is 80, which is the standard port for HTTP traffic. One can also tell that the traffic is still HTTP since the plain text can still be seen in Wireshark.

3.6.1 Some Programming on Your Web Server : Multiply

I used the code 200 OK to indicate that both numeric parameters were passed in correctly.

I used the code 204 No Content to indicate that neither required parameters were passed in correctly, whether that means that num1 or num2 is null or that they are not numbers.

I used the code 206 Partial Content to indicate that one and only one of the parameters was passed in correctly, meaning that the other parameter was either null or not a number.