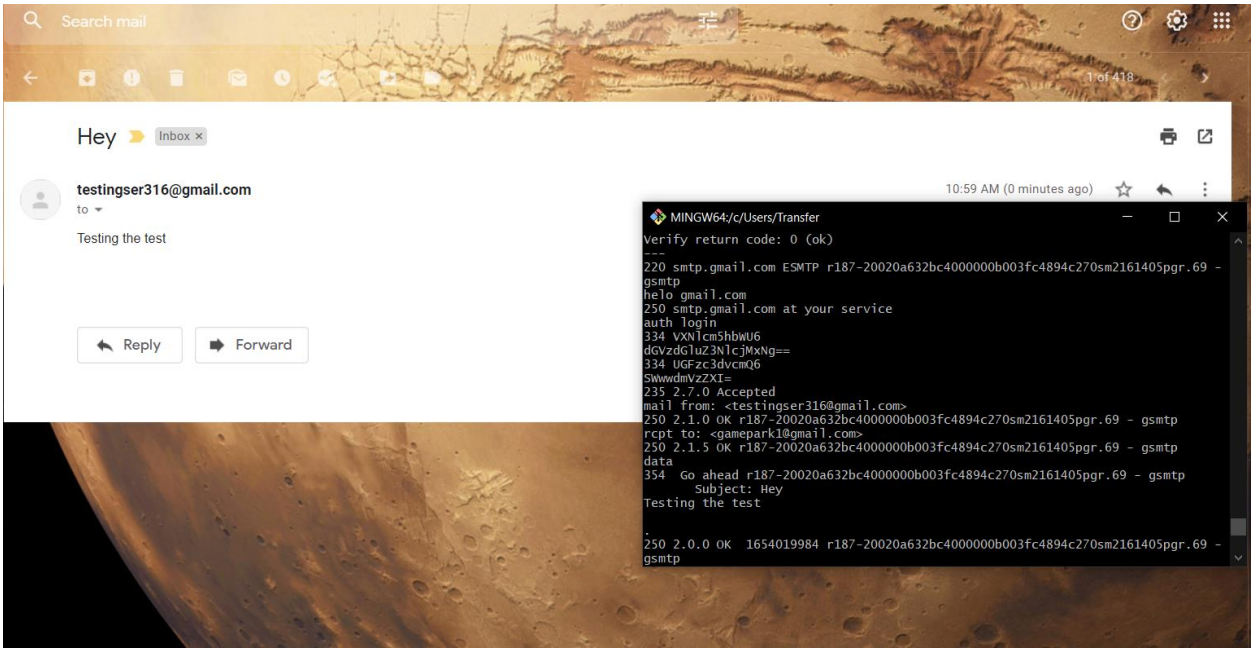
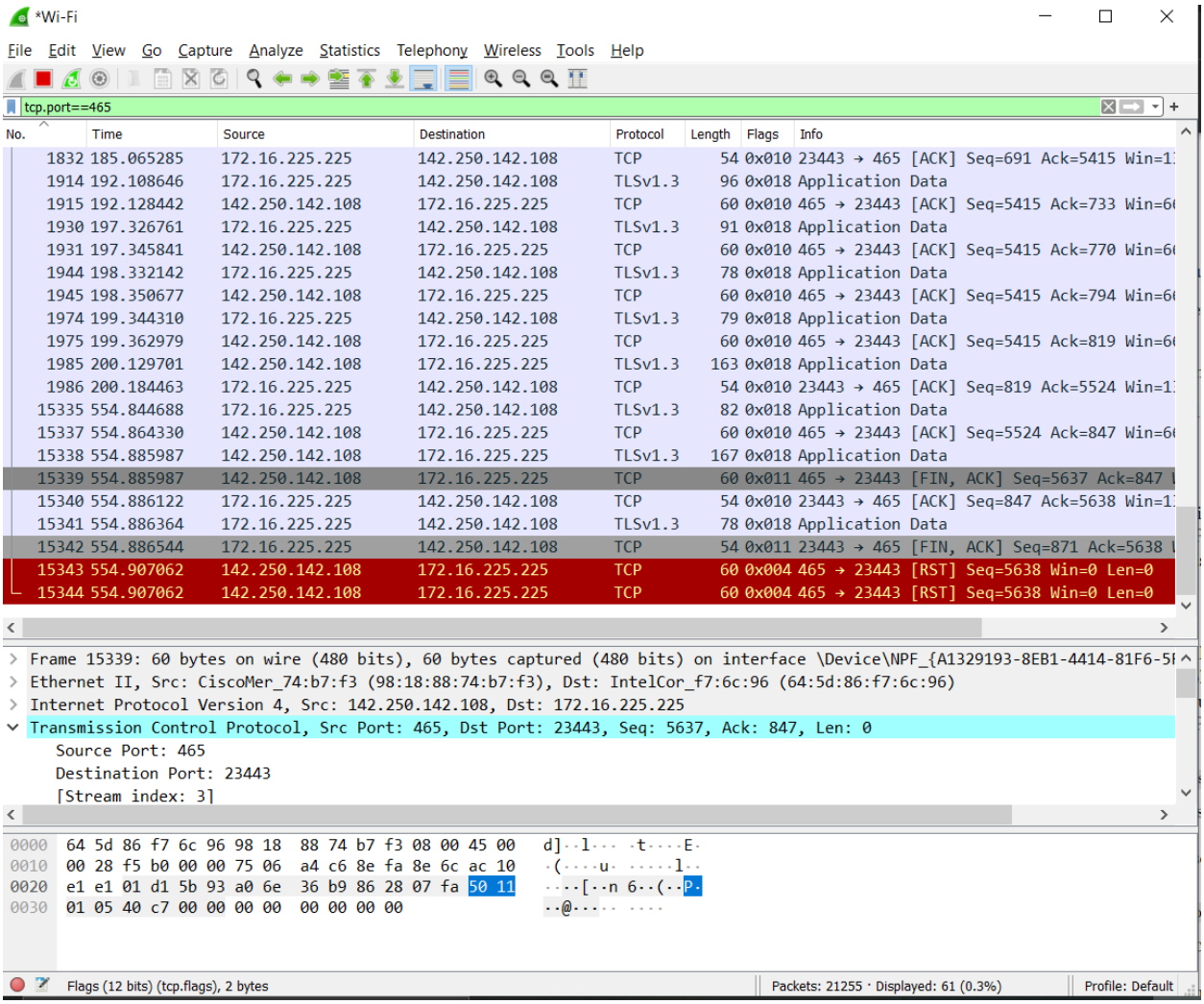
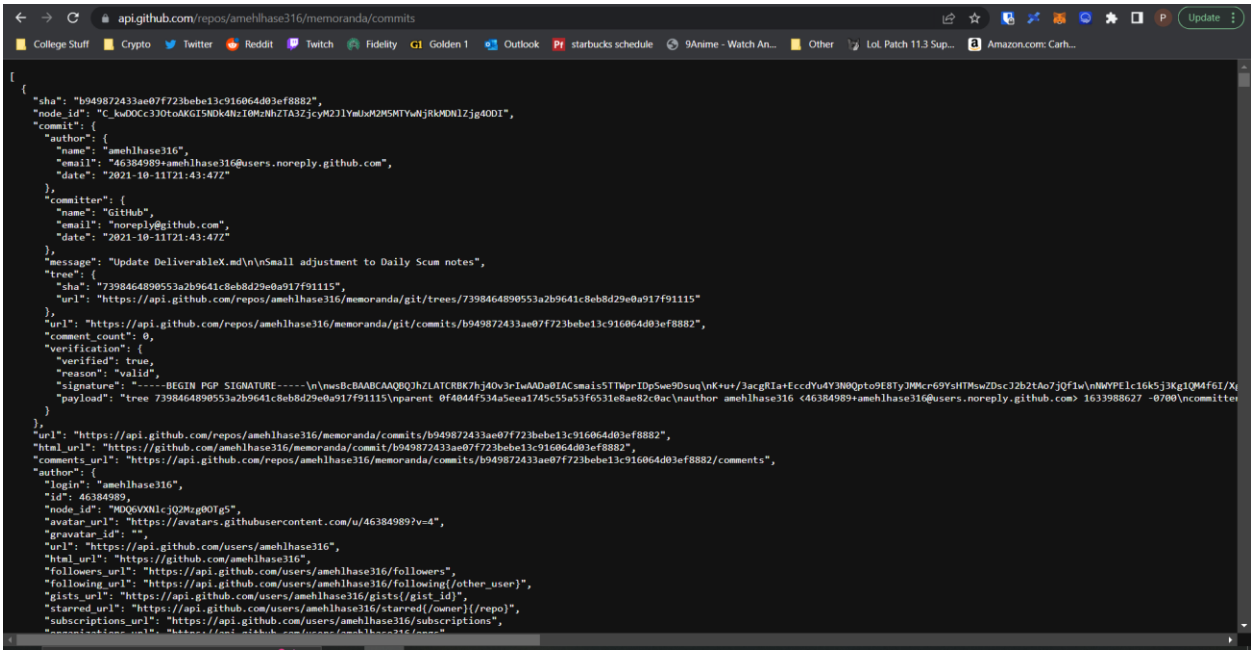


1. I chose the tcp.port==465 as the filter to catch the traffic. I chose this because in our code we specify that it is going to be on this port. When we use: openssl s_client −crlf −ign_eof −connect smtp.gmail.com: 465, the last part that has 465 is specifying that the data is going ot go through port 465.
2. Port 25 would be the standard port for SMTP. We chose a different port to make sure that the only data we are getting is the data we are trying to send from the command line. If we sued a standard port, we may get extra data from something else.
3. Line 1: openssl s_client −crlf −ign_eof −connect smtp.gmail.com: 465. This uses a SMTP connection. It is using our pc as the client. It also shuts down the connection when the end of the file is reached. –connect is specifying the port to connect to, which is port 465.
   Line 2: helo gmail.com is connecting to the gmail.com server.
   Line 3: auth login is prompting the user for the login information for gmail.com, which comes from using echo –ne username | base64 and echo –ne password | base64.
   Line 4: After the username and password is accepted, mail from: <> is choosing where to send the mail from.
   Line 5: rcpt to: <> is choosing what email to send the email to.
   Line 6: data accepts what data to send.
   Line 7: Subject specifies what is sent as the subject in the email.
   Line 8: Testing the test is the words sent in the body of the email.
   Line 9: “.” Means that the email is done and sending.
4. 14 packets are used to establish a connection
5. My local machine is using port 23443

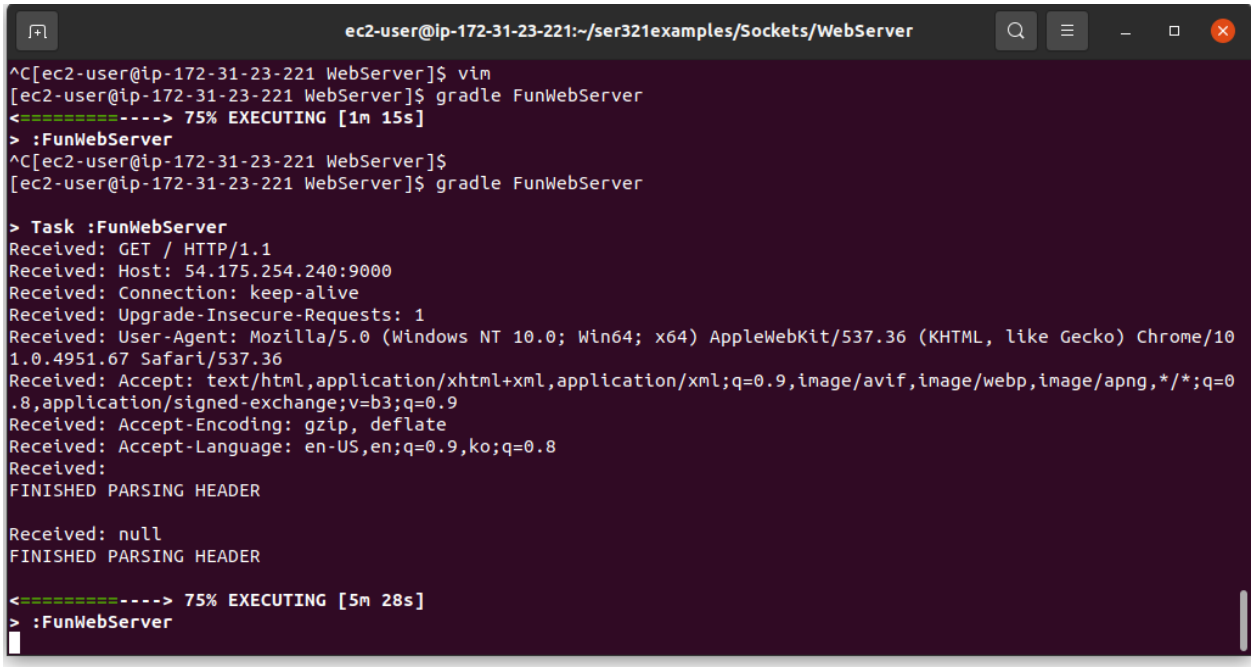
The computer sends the FIN flag to the server. Then the server acknowledges this. The server then sends the last bit of data to the computer and then acknowledges back to the computer that it is closing connection with FIN. Finally two more packets are sent that close the connection for both of them.

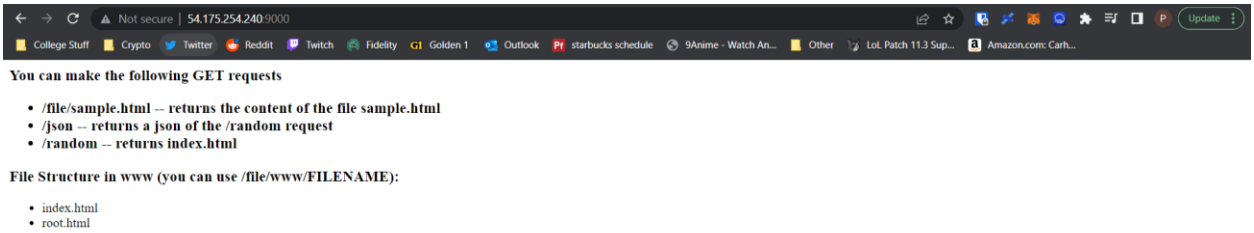


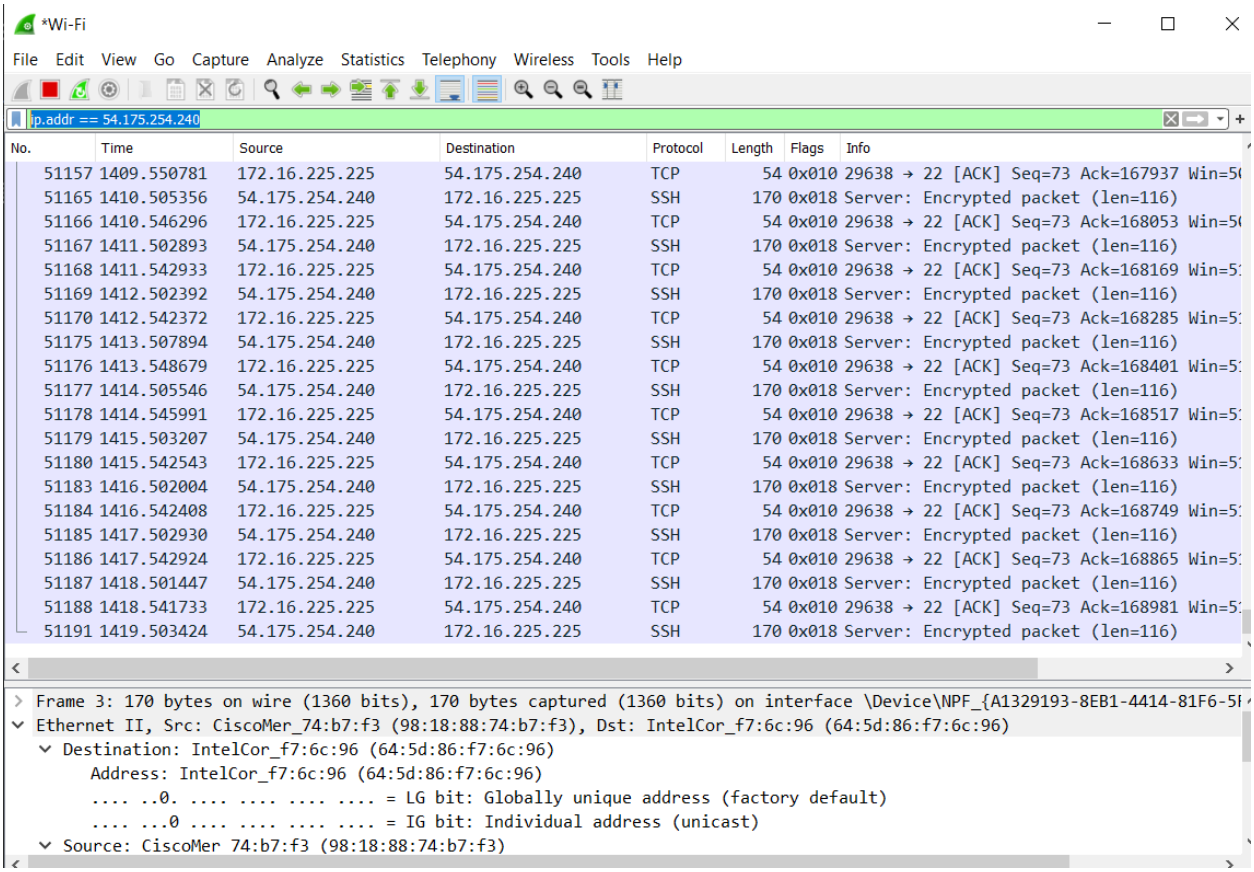
Github commits:



Stateful communications can react differently based on the history of the server. You can use the same input from the client and it could change the outcome based on how things change. Meanwhile a Stateless communication would focus on the client. The client would be used to maintain sessions, so the same input should have the same outcome.







3.3:

1. I chose the filter of ip.addr == 54.175.254.240, so that I can trace anything connecting to the server's ip address.

2. Hitting random returns a random object name and a corresponding picture with it.

3 and 4. I got the response code of XXX02 OK when I connected to the IP address to show that it connected successfully.

5. The server response I get shows the IP address, type of computer (Windows), that is it running on a website application (like Mozilla Firefox or Chrome), and the language that it is using. The data the server sends back to me shows all the text that is being sent (Ex. Type: text /html).

6. There is a lot of information being shared. IP address, type of computer, language, and type of application is a lot of data that I would not want being shared with everyone. The server is also sending all the information that it is trying to share with me without encrypting anything. This would make everything not secure at all, so it makes sense why it is not used for anything trying to be secure.

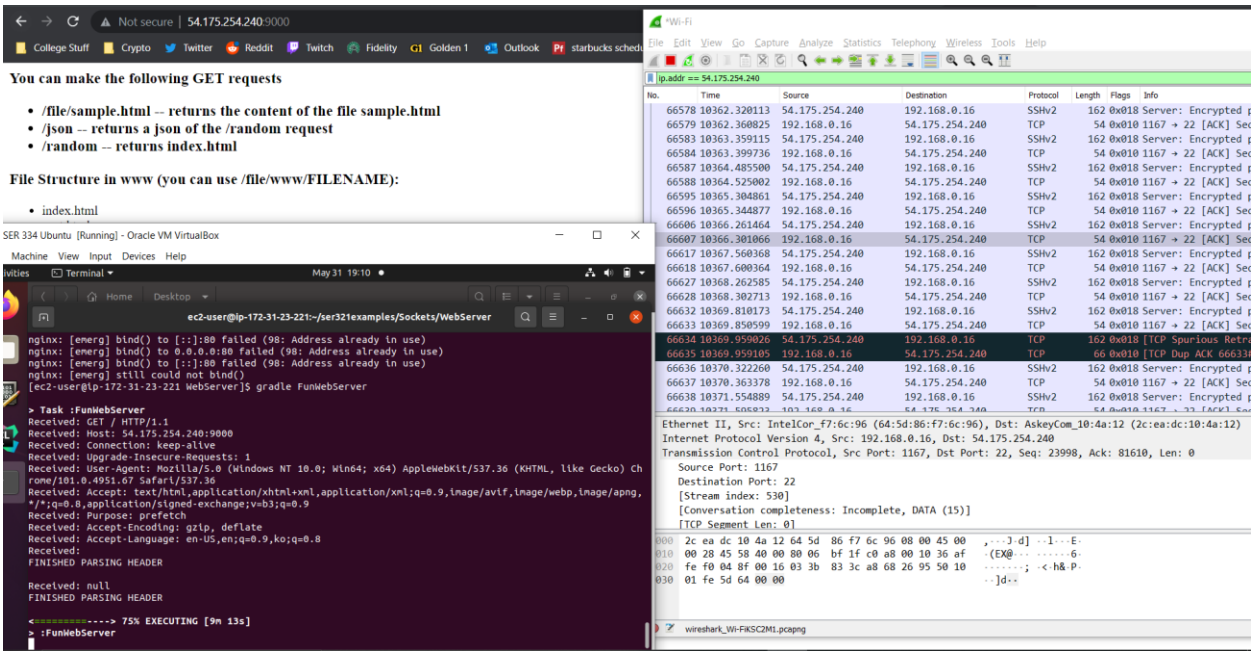
7. Port 9000. It is not the common port for HTTP, but we specified for it to run on this port when we ran the server.

8. Port 8127. This is a much smaller number than the port from the SMPT that we used, so it would most likely see more data.

3.3:

1. The source port is now 22. It is different now since it is more secure.

2. It is now HTTPS. You can tell because Wireshark says it is using SSHv2 instead of SSH, which is just a more secure version of SSH. Also, by looking at the data being sent, you can see that Wireshark is constantly sending encrypted packets that I can't understand by immediately looking at. This shows that it is more secure since it is encrypting data.



Multiply:

I decided to add builder.append() statements to determine if the input was provided properly. This makes sure there is the right amount of inputs and that the inputs are numbers. I decided to use error code 400 since the input was incorrect.

Github:

```
} else if (request.contains("github?")) {
  //builder.append("Hi");
  //return builder.toString().getBytes();
  // pulls the query from the request and runs it with GitHub's REST API
  // check out https://docs.github.com/rest/reference/
  //
  // HINT: REST is organized by nesting topics. Figure out the biggest one first,
  //     then drill down to what you care about
  // "Owner's repo is named RepoName. Example: find RepoName's contributors" translates to
  //     "/repos/OWNERNAME/REPONAME/contributors"
  Map<String, String> query_pairs = new LinkedHashMap<String, String>();
  query_pairs = splitQuery(request.replace("github?", ""));
  String json = fetchURL( aUrl: "https://api.github.com/" + query_pairs.get("query"));
  //System.out.println(json);
  while(query_pairs.get("full_name") != null) {
    String ownerName = query_pairs.get("login");
    String full_name = query_pairs.get("full_name");
    Integer id = Integer.parseInt(query_pairs.get("id"));
    builder.append("Owner: " + ownerName);
    builder.append(full_name);
    builder.append("id: " + id + "\n");
  }
  builder.append("Check the todos mentioned in the Java source file");
//1  JSONObject newObject = new JSONObject(json1);
  //JSONArray ownerName = new JSONArray(newObject.getJSONArray(""));
```