

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 to 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: Hey` 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.

\*Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port==465

No.	Time	Source	Destination	Protocol	Length	Flags	Info
1832	185.065285	172.16.225.225	142.250.142.108	TCP	54	0x010	23443 → 465 [ACK] Seq=691 Ack=5415 Win=1
1914	192.108646	172.16.225.225	142.250.142.108	TLSv1.3	96	0x018	Application Data
1915	192.128442	142.250.142.108	172.16.225.225	TCP	60	0x010	465 → 23443 [ACK] Seq=5415 Ack=733 Win=60
1930	197.326761	172.16.225.225	142.250.142.108	TLSv1.3	91	0x018	Application Data
1931	197.345841	142.250.142.108	172.16.225.225	TCP	60	0x010	465 → 23443 [ACK] Seq=5415 Ack=770 Win=60
1944	198.332142	172.16.225.225	142.250.142.108	TLSv1.3	78	0x018	Application Data
1945	198.350677	142.250.142.108	172.16.225.225	TCP	60	0x010	465 → 23443 [ACK] Seq=5415 Ack=794 Win=60
1974	199.344310	172.16.225.225	142.250.142.108	TLSv1.3	79	0x018	Application Data
1975	199.362979	142.250.142.108	172.16.225.225	TCP	60	0x010	465 → 23443 [ACK] Seq=5415 Ack=819 Win=60
1985	200.129701	142.250.142.108	172.16.225.225	TLSv1.3	163	0x018	Application Data
1986	200.184463	172.16.225.225	142.250.142.108	TCP	54	0x010	23443 → 465 [ACK] Seq=819 Ack=5524 Win=1
15335	554.844688	172.16.225.225	142.250.142.108	TLSv1.3	82	0x018	Application Data
15337	554.864330	142.250.142.108	172.16.225.225	TCP	60	0x010	465 → 23443 [ACK] Seq=5524 Ack=847 Win=60
15338	554.885987	142.250.142.108	172.16.225.225	TLSv1.3	167	0x018	Application Data
15339	554.885987	142.250.142.108	172.16.225.225	TCP	60	0x011	465 → 23443 [FIN, ACK] Seq=5637 Ack=847
15340	554.886122	172.16.225.225	142.250.142.108	TCP	54	0x010	23443 → 465 [ACK] Seq=847 Ack=5638 Win=1
15341	554.886364	172.16.225.225	142.250.142.108	TLSv1.3	78	0x018	Application Data
15342	554.886544	172.16.225.225	142.250.142.108	TCP	54	0x011	23443 → 465 [FIN, ACK] Seq=871 Ack=5638
15343	554.907062	142.250.142.108	172.16.225.225	TCP	60	0x004	465 → 23443 [RST] Seq=5638 Win=0 Len=0
15344	554.907062	142.250.142.108	172.16.225.225	TCP	60	0x004	465 → 23443 [RST] Seq=5638 Win=0 Len=0

< >

> Frame 15339: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF\_{A1329193-8EB1-4414-81F6-5F}^

> Ethernet II, Src: CiscoMer\_74:b7:f3 (98:18:88:74:b7:f3), Dst: IntelCor\_f7:6c:96 (64:5d:86:f7:6c:96)

> Internet Protocol Version 4, Src: 142.250.142.108, Dst: 172.16.225.225

▼ Transmission Control Protocol, Src Port: 465, Dst Port: 23443, Seq: 5637, Ack: 847, Len: 0

Source Port: 465

Destination Port: 23443

[Stream index: 3]

< >

0000 64 5d 86 f7 6c 96 98 18 88 74 b7 f3 08 00 45 00 d]..l...t....E-

0010 00 28 f5 b0 00 00 75 06 a4 c6 8e fa 8e 6c ac 10 .(....u. ....l..

0020 e1 e1 01 d1 5b 93 a0 6e 36 b9 86 28 07 fa 50 11 ....[.n 6..(..P-

0030 01 05 40 c7 00 00 00 00 00 00 00 00 ..@.....

Flags (12 bits) (tcp.flags), 2 bytes

Packets: 21255 · Displayed: 61 (0.3%)

Profile: Default

## Github commits:

```
api.github.com/repos/amehlhase316/memoranda/commits
[
  {
    "sha": "b949872433ae07f723bebe13c916064d03ef8882",
    "node_id": "C_kwDOCC3J0aWQGI5NDk4bzI0bzNlZTA3ZjcyM2JlYmUxM2M5MTYwNjRkMDNlZjg4ODI",
    "commit": {
      "author": {
        "name": "amehlhase316",
        "email": "46384989+amehlhase316@users.noreply.github.com",
        "date": "2021-10-11T21:43:47Z"
      },
      "committer": {
        "name": "GitHub",
        "email": "noreply@github.com",
        "date": "2021-10-11T21:43:47Z"
      },
      "message": "Update DeliverableX.md\n\nSmall adjustment to Daily Scum notes",
      "tree": {
        "sha": "7398464890553a2b9641c8eb8d29e0a917f91115",
        "url": "https://api.github.com/repos/amehlhase316/memoranda/git/trees/7398464890553a2b9641c8eb8d29e0a917f91115"
      },
      "url": "https://api.github.com/repos/amehlhase316/memoranda/git/commits/b949872433ae07f723bebe13c916064d03ef8882",
      "comment_count": 0,
      "verification": {
        "verified": true,
        "reason": "valid",
        "signature": "-----BEGIN PGP SIGNATURE-----\n\nmvsBcBAARCAABQOJhZLAICRBRK7hJ40v3rIw4Ade0IACsmals5TThprIDpSue90suqNkKx/3acgRIsEccdydY3HQ0ptc0E8TyJMKr69YsITHMwZDc3Zb2tAo7jQf1vnhMAYE1c1645j3Kg1QM4f6I/Xi\n\npayload: "tree 7398464890553a2b9641c8eb8d29e0a917f91115\nparent 0f4044f534a5eaa1745c55a53f6531e8ae82c0ac\nauthor amehlhase316 46384989+amehlhase316@users.noreply.github.com 1633988627 -0700\ncommitter"
      }
    },
    "url": "https://api.github.com/repos/amehlhase316/memoranda/commit/b949872433ae07f723bebe13c916064d03ef8882",
    "html_url": "https://github.com/amehlhase316/memoranda/commit/b949872433ae07f723bebe13c916064d03ef8882",
    "comments_url": "https://api.github.com/repos/amehlhase316/memoranda/commits/b949872433ae07f723bebe13c916064d03ef8882/comments",
    "author": {
      "login": "amehlhase316",
      "id": 46384989,
      "node_id": "MDQ6VXNlcjQ2Mzg0Ojg5",
      "avatar_url": "https://avatars.githubusercontent.com/u/46384989?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/amehlhase316",
      "html_url": "https://github.com/amehlhase316",
      "followers_url": "https://api.github.com/users/amehlhase316/followers",
      "following_url": "https://api.github.com/users/amehlhase316/following{/other_user}",
      "gists_url": "https://api.github.com/users/amehlhase316/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/amehlhase316/starred{/owner}/{repo}",
      "subscriptions_url": "https://api.github.com/users/amehlhase316/subscriptions",
      "organizations_url": "https://api.github.com/users/amehlhase316/orgs"
    }
  }
]
```

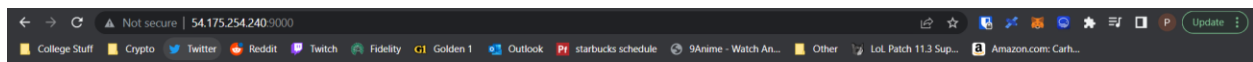
Stateful communications can react differently based on the history of the server. You can use the same input from the client and it could change the outcome based on how things change. Meanwhile a Stateless communication would focus on the client. The client would be used to maintain sessions, so the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

```
ec2-user@ip-172-31-23-221:~/ser321examples/Sockets/WebServer
^C[ec2-user@ip-172-31-23-221 WebServer]$ vim
[ec2-user@ip-172-31-23-221 WebServer]$ gradle FunWebServer
<=====--> 75% EXECUTING [1m 15s]
> :FunWebServer
^C[ec2-user@ip-172-31-23-221 WebServer]$
[ec2-user@ip-172-31-23-221 WebServer]$ gradle FunWebServer

> Task :FunWebServer
Received: GET / HTTP/1.1
Received: Host: 54.175.254.240:9000
Received: Connection: keep-alive
Received: Upgrade-Insecure-Requests: 1
Received: User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36
Received: Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Received: Accept-Encoding: gzip, deflate
Received: Accept-Language: en-US,en;q=0.9,ko;q=0.8
Received:
FINISHED PARSING HEADER

Received: null
FINISHED PARSING HEADER

<=====--> 75% EXECUTING [5m 28s]
> :FunWebServer
```



You can make the following GET requests

- /file/sample.html -- returns the content of the file sample.html
- /json -- returns a json of the /random request
- /random -- returns index.html

File Structure in www (you can use /file/www/FILENAME):

- index.html
- root.html

\*Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.addr == 54.175.254.240

No.	Time	Source	Destination	Protocol	Length	Flags	Info
51157	1409.550781	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=167937 Win=50
51165	1410.505356	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)
51166	1410.546296	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=168053 Win=50
51167	1411.502893	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)
51168	1411.542933	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=168169 Win=50
51169	1412.502392	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)
51170	1412.542372	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=168285 Win=50
51175	1413.507894	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)
51176	1413.548679	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=168401 Win=50
51177	1414.505546	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)
51178	1414.545991	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=168517 Win=50
51179	1415.503207	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)
51180	1415.542543	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=168633 Win=50
51183	1416.502004	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)
51184	1416.542408	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=168749 Win=50
51185	1417.502930	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)
51186	1417.542924	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=168865 Win=50
51187	1418.501447	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)
51188	1418.541733	172.16.225.225	54.175.254.240	TCP	54	0x010 29638 → 22	[ACK] Seq=73 Ack=168981 Win=50
51191	1419.503424	54.175.254.240	172.16.225.225	SSH	170	0x018	Server: Encrypted packet (len=116)

> Frame 3: 170 bytes on wire (1360 bits), 170 bytes captured (1360 bits) on interface \Device\NPF\_{A1329193-8EB1-4414-81F6-5F...}

▼ Ethernet II, Src: CiscoMer\_74:b7:f3 (98:18:88:74:b7:f3), Dst: IntelCor\_f7:6c:96 (64:5d:86:f7:6c:96)

- ▼ Destination: IntelCor\_f7:6c:96 (64:5d:86:f7:6c:96)  
Address: IntelCor\_f7:6c:96 (64:5d:86:f7:6c:96)  
.... ..0. .... = LG bit: Globally unique address (factory default)  
.... ..0. .... = IG bit: Individual address (unicast)
- ▼ Source: CiscoMer 74:b7:f3 (98:18:88:74:b7:f3)

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.

The screenshot displays three windows illustrating a web server setup and network traffic analysis:

- Web Browser (Top Left):** Shows a "Not secure" warning for the URL `54.175.254.240:9000`. The page content lists GET requests: `/file/sample.html` (returns file content), `/json` (returns a JSON object), and `/random` (returns random index.html). It also shows the file structure in `www` with `index.html`.
- Terminal (Bottom Left):** Shows the execution of `nghttp` and `nghttpx` commands. It displays the output of `nghttpx` connecting to `54.175.254.240:9000`, showing the request headers (Host, Connection, Upgrade-Insecure-Requests, User-Agent) and the response headers (Content-Type, Content-Length, Content-Disposition).
- Wireshark (Right):** Displays a packet capture of the network traffic. The top table shows the list of packets, including the initial connection attempt and subsequent data exchanges. The bottom pane shows the details of the selected packet, including the Ethernet II header, Internet Protocol Version 4 header, and Transmission Control Protocol header.

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(""));
}
```