# Mini-project-report

# Web & Web Proxy Servers

**Team member:**

**Wenxiang He 301417521**

**Cole Thacker 301540568**

**Group: MP 21**

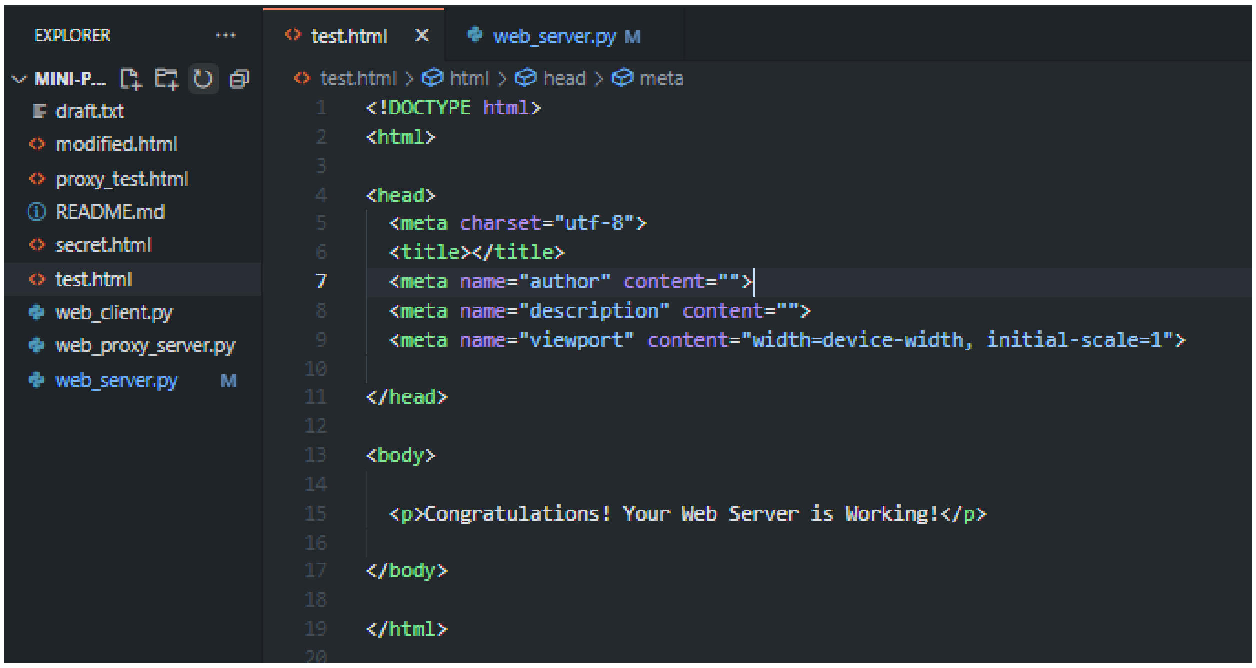
## Step Two Web Server test

**Test procedures**

For the server IP_ADDRESS and PORT we set 127.0.0.1(localhost) and 8080.

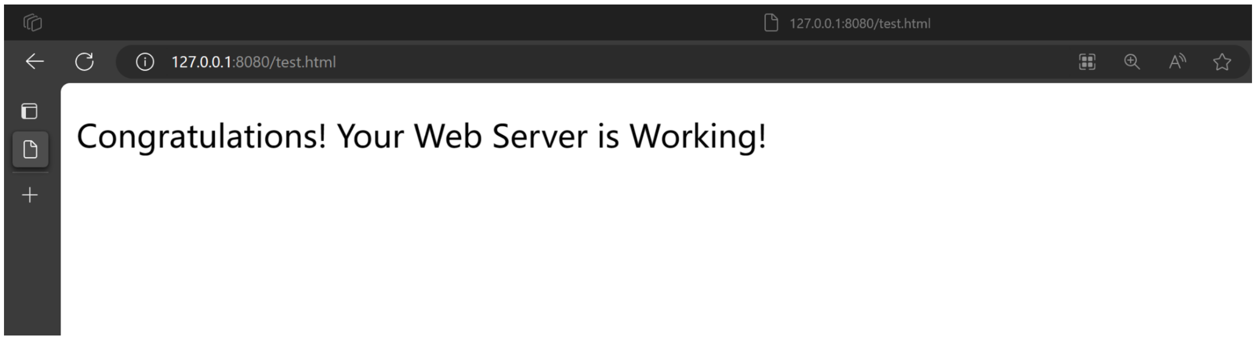
**Test for 200 OK:**

The server will respond with this message when successfully handling the request from the client. We initially have a test.html file in the server side



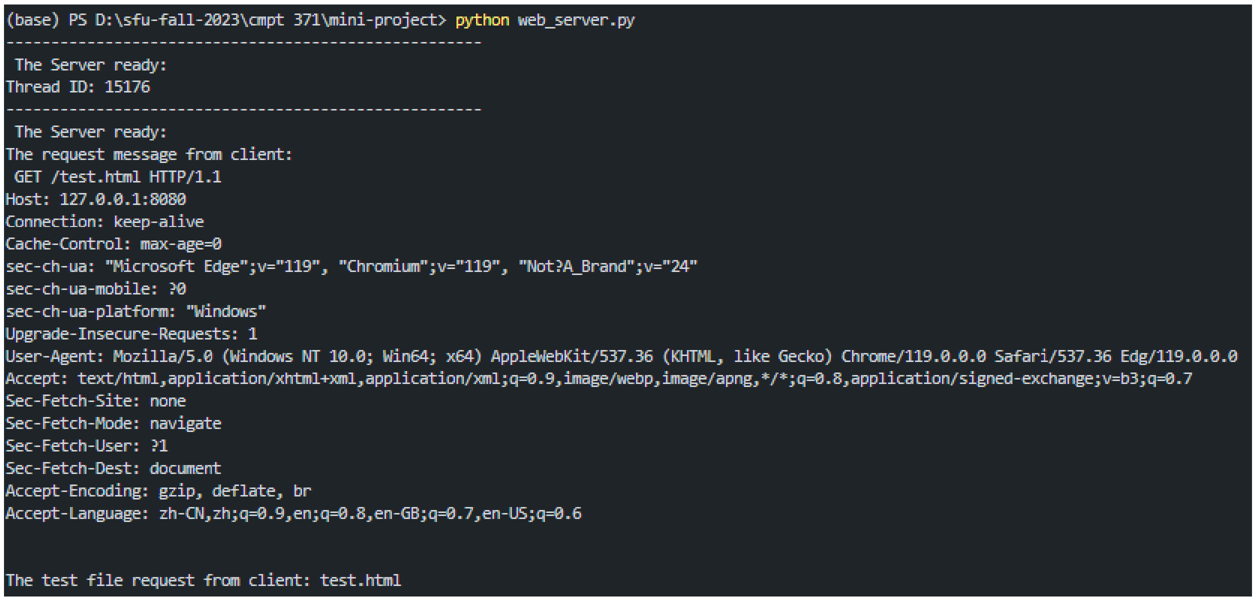
```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>Congratulations! Your Web Server is Working!</p>

</body>

</html>
```

We simply send a HTTP GET request to our server to access the test.html file, we visit the http://127.0.0.1:8080/test.html in our browser.



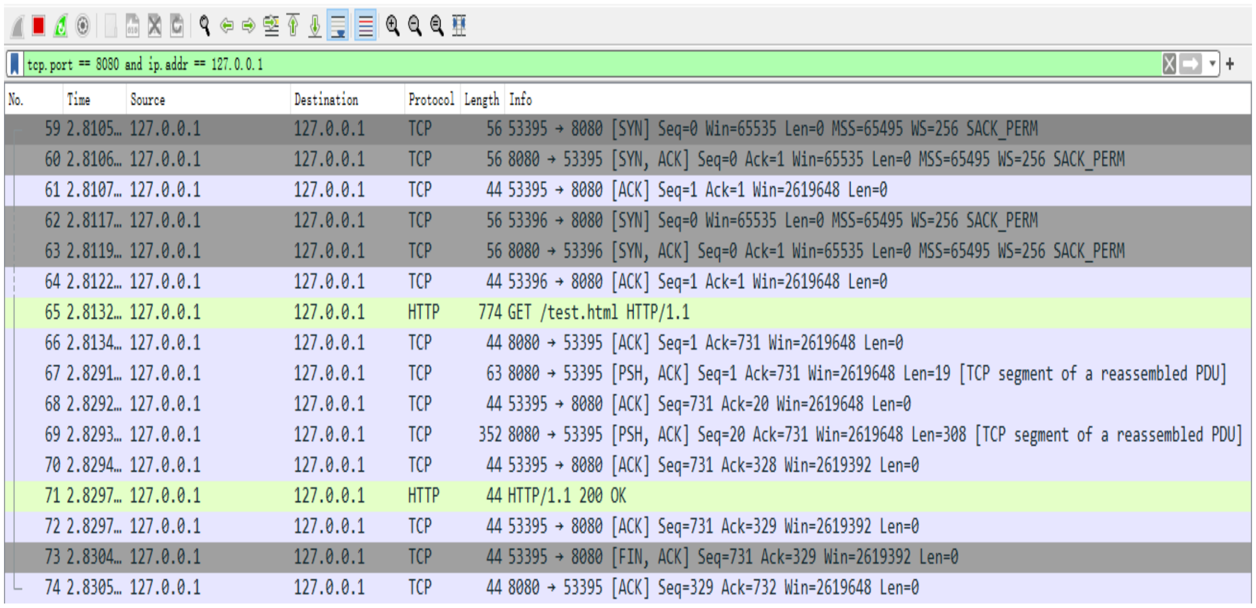
Congratulations! Your Web Server is Working!

Here is the response from the server, the client gets the test.html file and browses the web page.



```
(base) PS D:\sfu-fall-2023\cmpt 371\mini-project> python web_server.py
-----------------------------------------------------
 The Server ready:
Thread ID: 15176
-----------------------------------------------------
 The Server ready:
The request message from client:
 GET /test.html HTTP/1.1
Host: 127.0.0.1:8080
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Microsoft Edge";v="119", "Chromium";v="119", "Not?A_Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36 Edg/119.0.0.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6


The test file request from client: test.html
```

Here is the console output form the server, it successfully gets the request from the client and handles it.



tcp.port == 8080 and ip.addr == 127.0.0.1

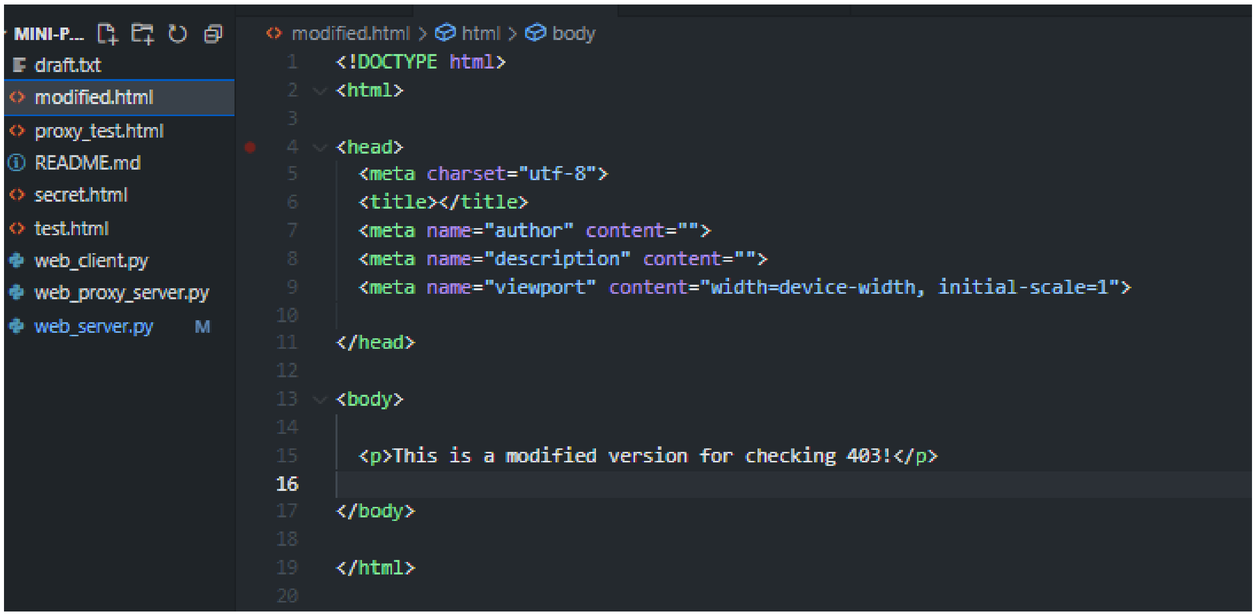
| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 59 | 2.8105… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 53395 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 60 | 2.8106… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 53395 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 61 | 2.8107… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 53395 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 62 | 2.8117… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 53396 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 63 | 2.8119… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 53396 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 64 | 2.8122… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 53396 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 65 | 2.8132… | 127.0.0.1 | 127.0.0.1 | HTTP | 774 | GET /test.html HTTP/1.1 |
| 66 | 2.8134… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 53395 [ACK] Seq=1 Ack=731 Win=2619648 Len=0 |
| 67 | 2.8291… | 127.0.0.1 | 127.0.0.1 | TCP | 63 | 8080 → 53395 [PSH, ACK] Seq=1 Ack=731 Win=2619648 Len=19 [TCP segment of a reassembled PDU] |
| 68 | 2.8292… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 53395 → 8080 [ACK] Seq=731 Ack=20 Win=2619648 Len=0 |
| 69 | 2.8293… | 127.0.0.1 | 127.0.0.1 | TCP | 352 | 8080 → 53395 [PSH, ACK] Seq=20 Ack=731 Win=2619648 Len=308 [TCP segment of a reassembled PDU] |
| 70 | 2.8294… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 53395 → 8080 [ACK] Seq=731 Ack=328 Win=2619392 Len=0 |
| 71 | 2.8297… | 127.0.0.1 | 127.0.0.1 | HTTP | 44 | HTTP/1.1 200 OK |
| 72 | 2.8297… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 53395 → 8080 [ACK] Seq=731 Ack=329 Win=2619392 Len=0 |
| 73 | 2.8304… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 53395 → 8080 [FIN, ACK] Seq=731 Ack=329 Win=2619392 Len=0 |
| 74 | 2.8305… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 53395 [ACK] Seq=329 Ack=732 Win=2619648 Len=0 |

Here we use wireshark to capture the packet in the certain tcp port 8080 and ip address 127.0.0.1. As shown above there is one GET request from the client to the server, and the server responds with the message **200 OK .**

**Test for 304 Not Modified:**

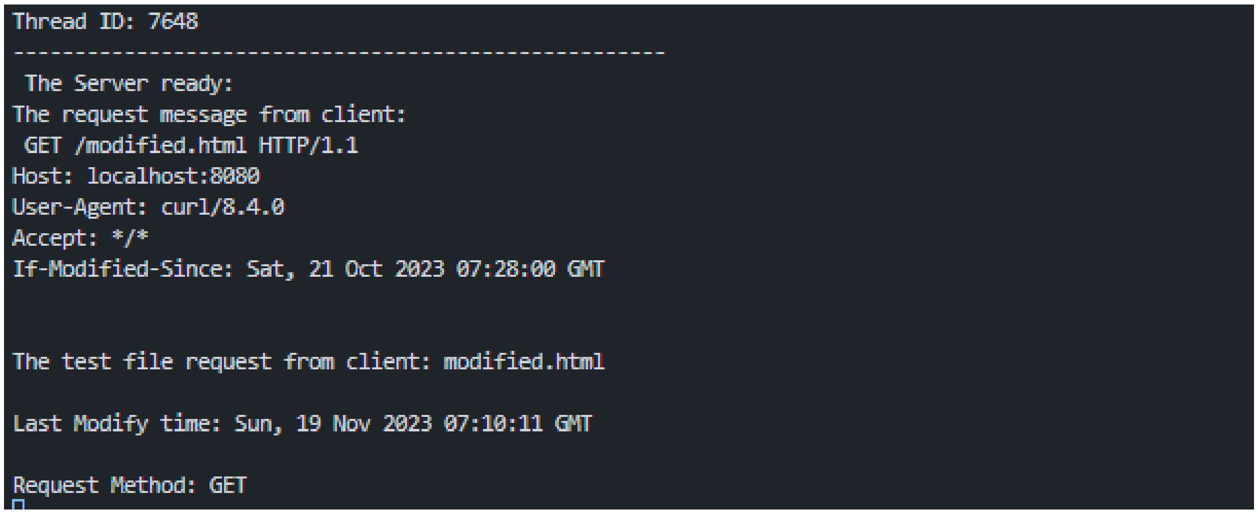
The server will respond with this message when the file that client requests has not been modified after a certain time. It does need to respond with the same not modified file, but only the message **304 Not Modified.** Otherwise, it responds with the modified file and message **200 OK.**

To test it we need to send a conditional HTTP GET request that includes an "If-Modified-Since" header to our server to request for one modified.html file.



```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>This is a modified version for checking 403!</p>

</body>

</html>
```

The modified.html file has a last modified time at Sun, 19 Nov 2023 07:10:11 GMT.

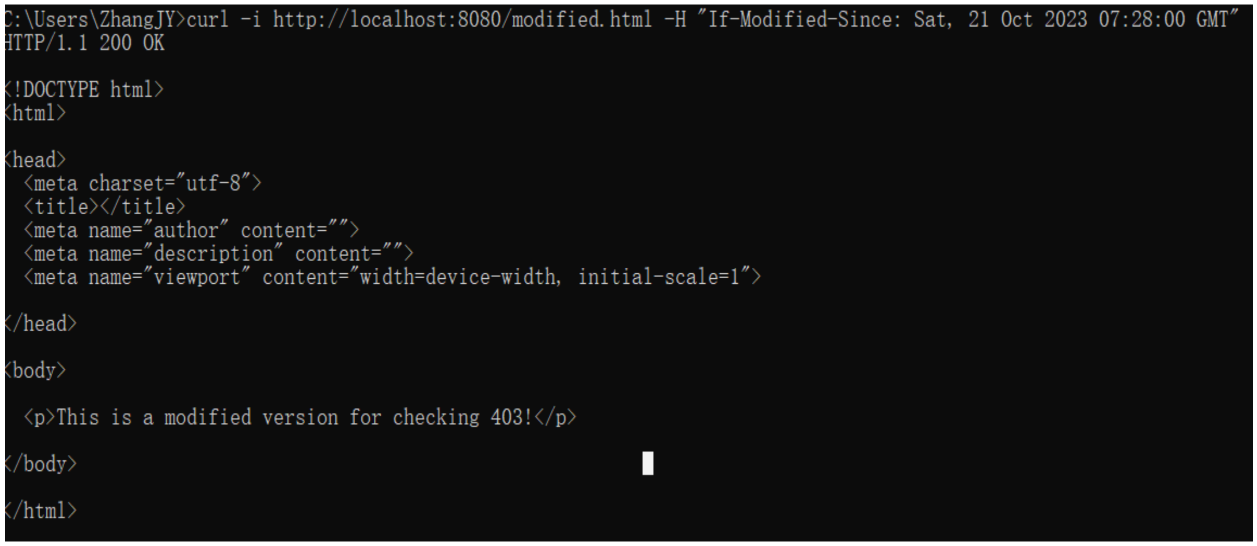


```
Thread ID: 7648
-----------------------------------------------------
 The Server ready:
The request message from client:
 GET /modified.html HTTP/1.1
Host: localhost:8080
User-Agent: curl/8.4.0
Accept: */*
If-Modified-Since: Sat, 21 Oct 2023 07:28:00 GMT


The test file request from client: modified.html

Last Modify time: Sun, 19 Nov 2023 07:10:11 GMT

Request Method: GET
```

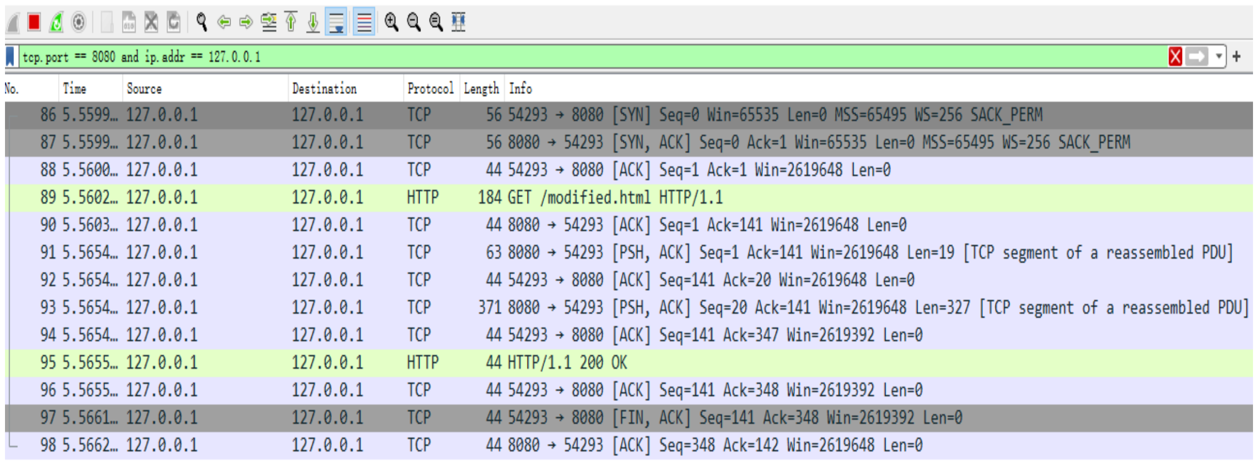
We use the “curl” command line tool to send two conditional HTTP GET requests that include different “If-Modified-Since” headers.

```
curl -i http://localhost:8080/modified.html -H "If-Modified-Since: Sat, 21 Oct 2023 07:28:00 GMT"
```

The first one we set the time in the “If-Modified-Since” header before the last modified time of modified.html, which means the file has been modified, we should get the message **200 OK** and the modified.html file.



```
C:\Users\ZhangJY>curl -i http://localhost:8080/modified.html -H "If-Modified-Since: Sat, 21 Oct 2023 07:28:00 GMT"
HTTP/1.1 200 OK

<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>This is a modified version for checking 403!</p>

</body>

</html>
```



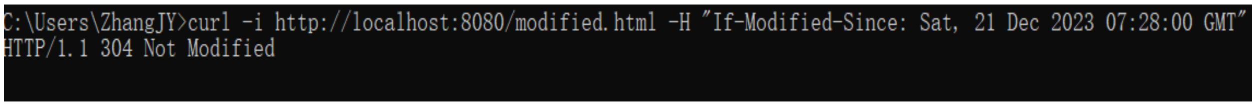
tcp.port == 8080 and ip.addr == 127.0.0.1

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 86 | 5.5599… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 54293 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 87 | 5.5599… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 54293 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 88 | 5.5600… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 54293 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 89 | 5.5602… | 127.0.0.1 | 127.0.0.1 | HTTP | 184 | GET /modified.html HTTP/1.1 |
| 90 | 5.5603… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 54293 [ACK] Seq=1 Ack=141 Win=2619648 Len=0 |
| 91 | 5.5654… | 127.0.0.1 | 127.0.0.1 | TCP | 63 | 8080 → 54293 [PSH, ACK] Seq=1 Ack=141 Win=2619648 Len=19 [TCP segment of a reassembled PDU] |
| 92 | 5.5654… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 54293 → 8080 [ACK] Seq=141 Ack=20 Win=2619648 Len=0 |
| 93 | 5.5654… | 127.0.0.1 | 127.0.0.1 | TCP | 371 | 8080 → 54293 [PSH, ACK] Seq=20 Ack=141 Win=2619648 Len=327 [TCP segment of a reassembled PDU] |
| 94 | 5.5654… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 54293 → 8080 [ACK] Seq=141 Ack=347 Win=2619392 Len=0 |
| 95 | 5.5655… | 127.0.0.1 | 127.0.0.1 | HTTP | 44 | HTTP/1.1 200 OK |
| 96 | 5.5655… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 54293 → 8080 [ACK] Seq=141 Ack=348 Win=2619392 Len=0 |
| 97 | 5.5661… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 54293 → 8080 [FIN, ACK] Seq=141 Ack=348 Win=2619392 Len=0 |
| 98 | 5.5662… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 54293 [ACK] Seq=348 Ack=142 Win=2619648 Len=0 |

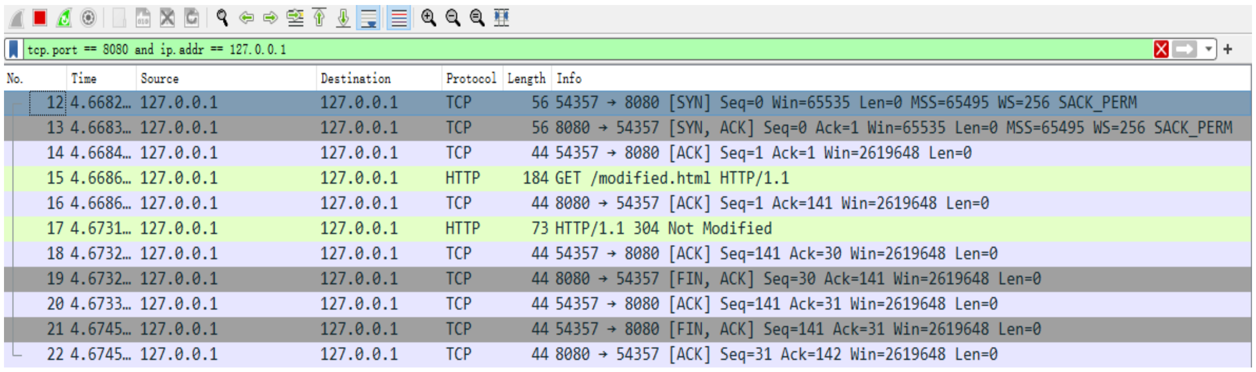
As seen from the command line output and wireshark capture the server responds with the modified.html file and the message **200 OK .**

```
curl -i http://localhost:8080/modified.html -H "If-Modified-Since: Sat, 21 Dec 2023 07:28:00 GMT"
```

The second one we set the date in the "If-Modified-Since" header after the last modified time, which means the file has not been modified, we should get the message **304 Not Modified.**



```
C:\Users\ZhangJY>curl -i http://localhost:8080/modified.html -H "If-Modified-Since: Sat, 21 Dec 2023 07:28:00 GMT"
HTTP/1.1 304 Not Modified
```



```
tcp.port == 8080 and ip.addr == 127.0.0.1
```

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 12 | 4.6682… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 54357 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 13 | 4.6683… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 54357 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 14 | 4.6684… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 54357 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 15 | 4.6686… | 127.0.0.1 | 127.0.0.1 | HTTP | 184 | GET /modified.html HTTP/1.1 |
| 16 | 4.6686… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 54357 [ACK] Seq=1 Ack=141 Win=2619648 Len=0 |
| 17 | 4.6731… | 127.0.0.1 | 127.0.0.1 | HTTP | 73 | HTTP/1.1 304 Not Modified |
| 18 | 4.6732… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 54357 → 8080 [ACK] Seq=141 Ack=30 Win=2619648 Len=0 |
| 19 | 4.6732… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 54357 [FIN, ACK] Seq=30 Ack=141 Win=2619648 Len=0 |
| 20 | 4.6733… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 54357 → 8080 [ACK] Seq=141 Ack=31 Win=2619648 Len=0 |
| 21 | 4.6745… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 54357 → 8080 [FIN, ACK] Seq=141 Ack=31 Win=2619648 Len=0 |
| 22 | 4.6745… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 54357 [ACK] Seq=31 Ack=142 Win=2619648 Len=0 |

As seen from the command line output and wireshark capture the server responds with the message **304 Not Modified.**

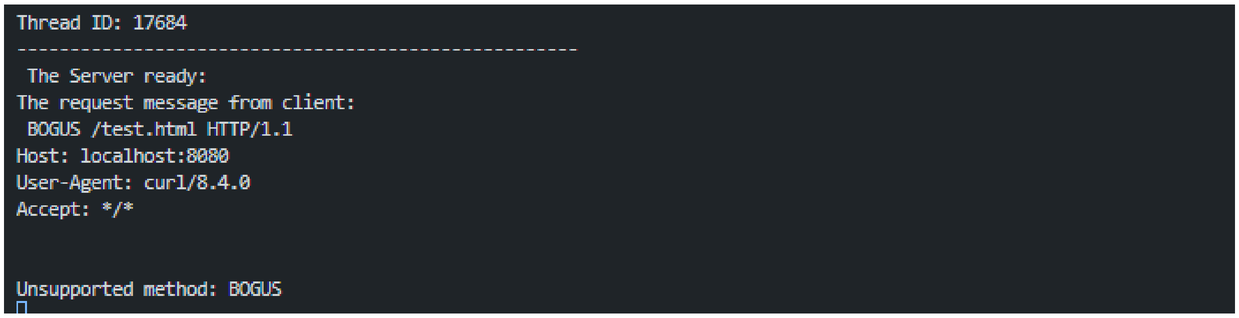
**Test for 400 Bad Request:**

For this message we can send a HTTP Get request that includes a non-standard or non-existent HTTP method "BOGUS" to our server to request for the test.html file.

We use the "curl" command line tool to send a HTTP Get request that includes a non-standard or non-existent HTTP method "BOGUS".

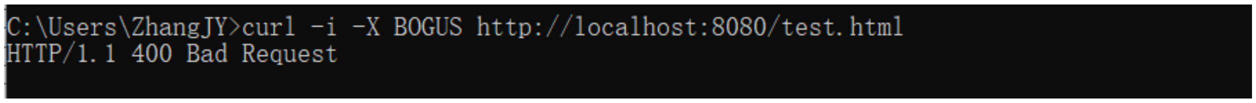
```
curl -i -X BOGUS http://localhost:8080/test.html
```

In such cases, the server does not support this non-existent HTTP method "BOGUS", and will respond with a **400 Bad Request message.**



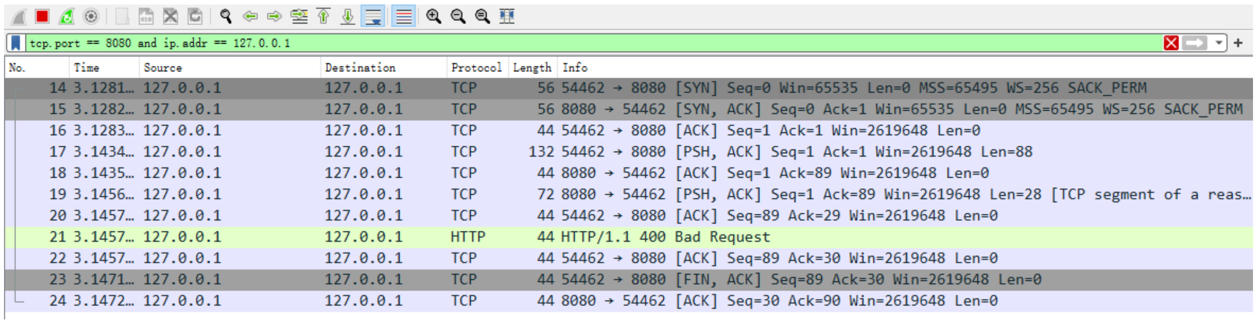
```
Thread ID: 17684
-----------------------------------------------------
 The Server ready:
The request message from client:
 BOGUS /test.html HTTP/1.1
Host: localhost:8080
User-Agent: curl/8.4.0
Accept: */*


Unsupported method: BOGUS
```

Here is the console output from the server, it shows an unsupported method.



```
C:\Users\ZhangJY>curl -i -X BOGUS http://localhost:8080/test.html
HTTP/1.1 400 Bad Request
```

From the client side we get the **400 Bad Request message.**



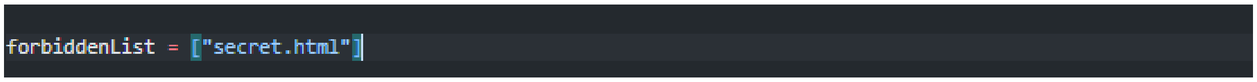
```
tcp.port == 8080 and ip.addr == 127.0.0.1

No.  Time       Source     Destination  Protocol Length Info
 14 3.1281…  127.0.0.1  127.0.0.1    TCP       56 54462 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 15 3.1282…  127.0.0.1  127.0.0.1    TCP       56 8080 → 54462 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 16 3.1283…  127.0.0.1  127.0.0.1    TCP       44 54462 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
 17 3.1434…  127.0.0.1  127.0.0.1    TCP      132 54462 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=88
 18 3.1435…  127.0.0.1  127.0.0.1    TCP       44 8080 → 54462 [ACK] Seq=1 Ack=89 Win=2619648 Len=0
 19 3.1456…  127.0.0.1  127.0.0.1    TCP       72 8080 → 54462 [PSH, ACK] Seq=1 Ack=89 Win=2619648 Len=28 [TCP segment of a reas…
 20 3.1457…  127.0.0.1  127.0.0.1    TCP       44 54462 → 8080 [ACK] Seq=89 Ack=29 Win=2619648 Len=0
 21 3.1457…  127.0.0.1  127.0.0.1    HTTP      44 HTTP/1.1 400 Bad Request
 22 3.1457…  127.0.0.1  127.0.0.1    TCP       44 54462 → 8080 [ACK] Seq=89 Ack=30 Win=2619648 Len=0
 23 3.1471…  127.0.0.1  127.0.0.1    TCP       44 54462 → 8080 [FIN, ACK] Seq=89 Ack=30 Win=2619648 Len=0
 24 3.1472…  127.0.0.1  127.0.0.1    TCP       44 8080 → 54462 [ACK] Seq=30 Ack=90 Win=2619648 Len=0
```

Using wireshark capture we see that the server responds with a **400 Bad Request** message.

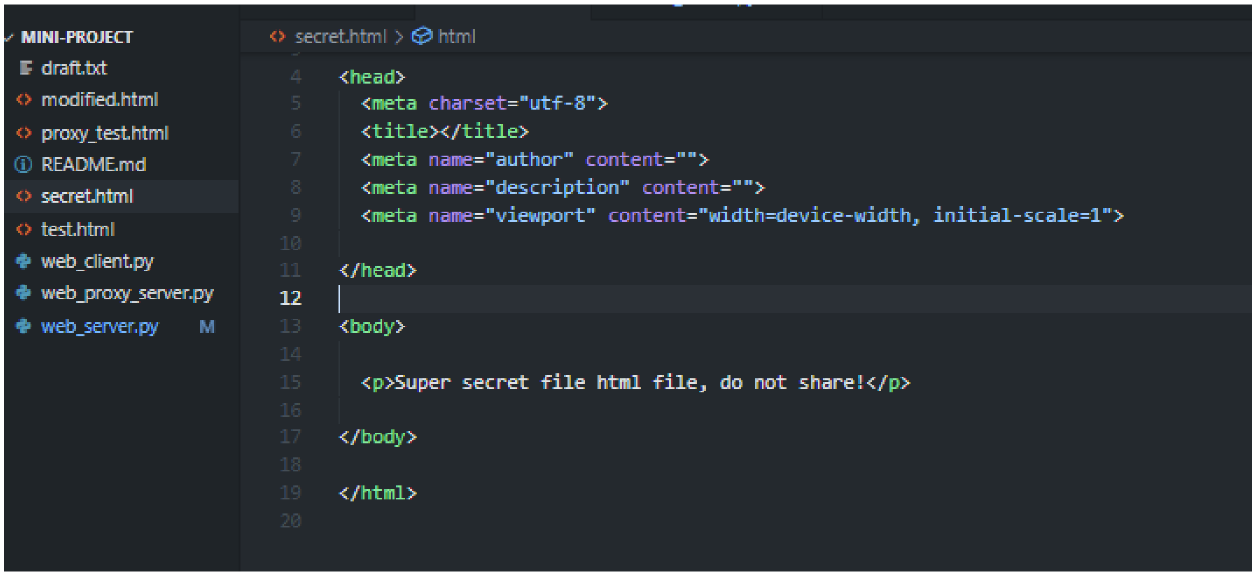
**Test for 403 Forbidden:**

The server will respond with this message whenever clients try to request files with no permission.

We created a forbiddenList in the server and it contains one secret.html file, whenever clients try to request for secret.html in the forbiddenList, the server will respond with the message **403 Forbidden.**



```
forbiddenList = ["secret.html"]
```

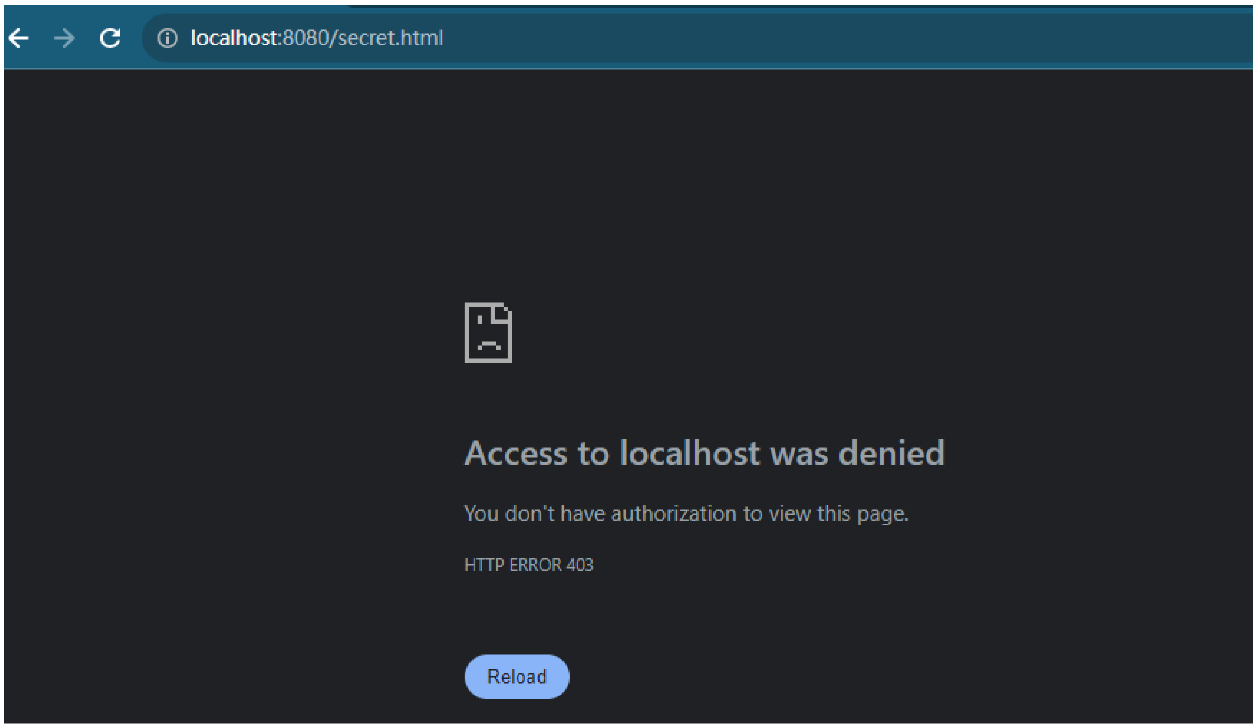


MINI-PROJECT
- draft.txt
- modified.html
- proxy_test.html
- README.md
- secret.html
- test.html
- web_client.py
- web_proxy_server.py
- web_server.py M

secret.html > html

```
<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>Super secret file html file, do not share!</p>

</body>

</html>
```

We can test this by visiting the http://127.0.0.1:8080/secret.html in our browser.

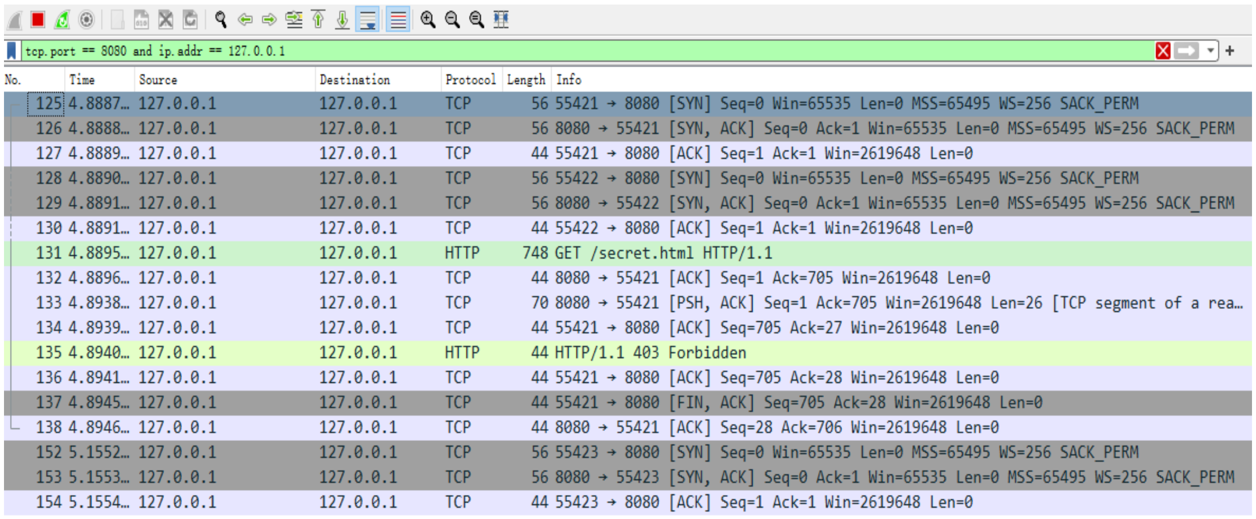

localhost:8080/secret.html

Access to localhost was denied

You don't have authorization to view this page.

HTTP ERROR 403

The client receives a **403 code**, and cannot access the page.


tcp.port == 8080 and ip.addr == 127.0.0.1

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 125 | 4.8887… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 55421 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 126 | 4.8888… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 55421 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 127 | 4.8889… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 55421 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 128 | 4.8890… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 55422 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 129 | 4.8891… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 55422 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 130 | 4.8891… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 55422 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 131 | 4.8895… | 127.0.0.1 | 127.0.0.1 | HTTP | 748 | GET /secret.html HTTP/1.1 |
| 132 | 4.8896… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 55421 [ACK] Seq=1 Ack=705 Win=2619648 Len=0 |
| 133 | 4.8938… | 127.0.0.1 | 127.0.0.1 | TCP | 70 | 8080 → 55421 [PSH, ACK] Seq=1 Ack=705 Win=2619648 Len=26 [TCP segment of a rea… |
| 134 | 4.8939… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 55421 → 8080 [ACK] Seq=705 Ack=27 Win=2619648 Len=0 |
| 135 | 4.8940… | 127.0.0.1 | 127.0.0.1 | HTTP | 44 | HTTP/1.1 403 Forbidden |
| 136 | 4.8941… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 55421 → 8080 [ACK] Seq=705 Ack=28 Win=2619648 Len=0 |
| 137 | 4.8945… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 55421 → 8080 [FIN, ACK] Seq=705 Ack=28 Win=2619648 Len=0 |
| 138 | 4.8946… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 55421 [ACK] Seq=28 Ack=706 Win=2619648 Len=0 |
| 152 | 5.1552… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 55423 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 153 | 5.1553… | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 55423 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 154 | 5.1554… | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 55423 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |

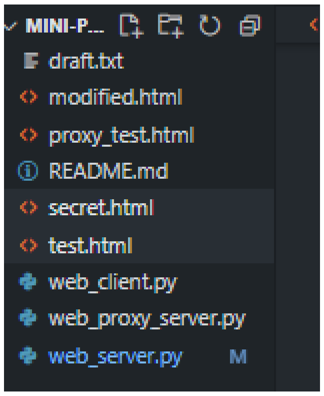
Using wireshark capture we see that when we try to visit the website in the forbidden list, the server only responds with the message **403 Forbidden.**

**Test for 404 Not Found:**

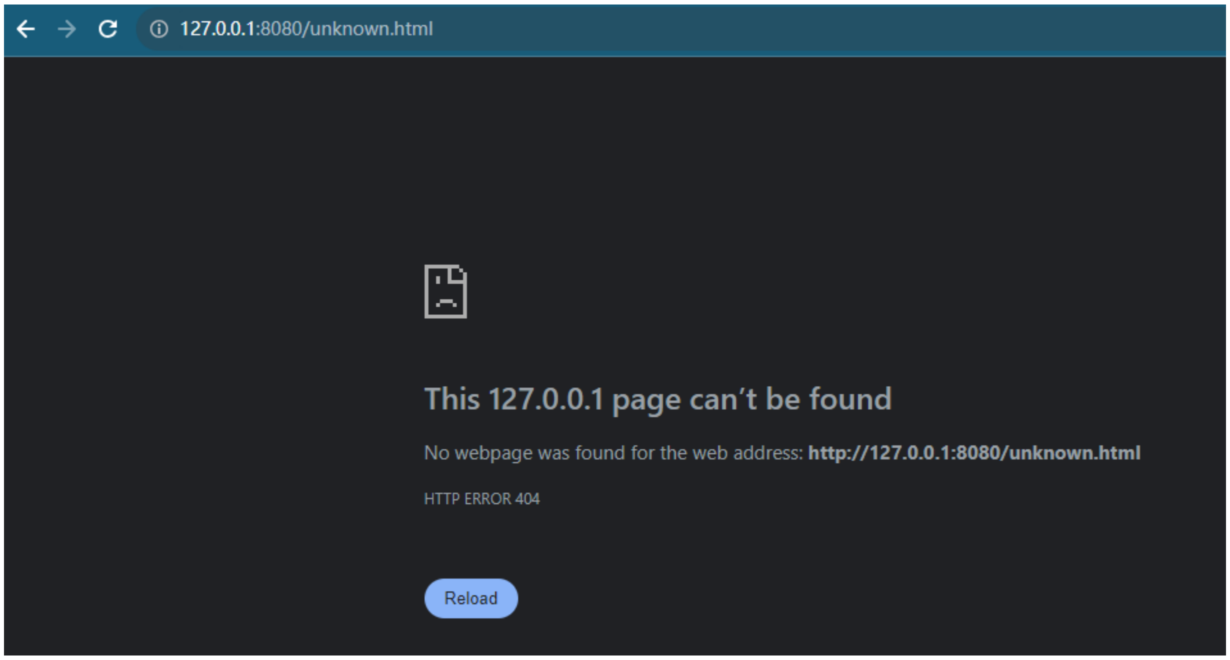
The server will respond with this message whenever clients try to request files not found on the server.

For this message we can test by requesting a file not stored in the server,  then the server will respond with the message **404 Not Found.**

We can test this by visiting the [http://127.0.0.1:8080/unknown.html](http://127.0.0.1:8080/unknown.html) in our browser.


```
MINI-P...
draft.txt
modified.html
proxy_test.html
README.md
secret.html
test.html
web_client.py
web_proxy_server.py
web_server.py        M
```

This unknown.html file is not stored on the server.

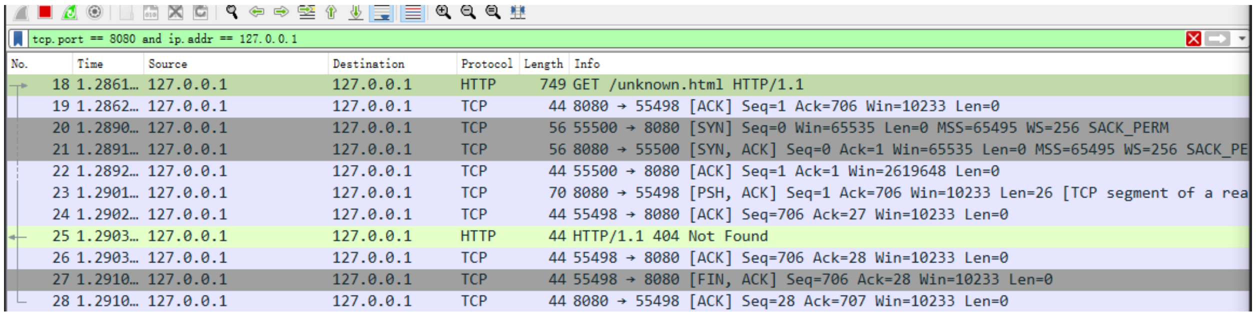

127.0.0.1:8080/unknown.html

This 127.0.0.1 page can't be found

No webpage was found for the web address: **http://127.0.0.1:8080/unknown.html**

HTTP ERROR 404

The client receives a **404 code.**



```
tcp.port == 8080 and ip.addr == 127.0.0.1
No.   Time      Source      Destination  Protocol Length Info
   18 1.2861…  127.0.0.1   127.0.0.1    HTTP      749 GET /unknown.html HTTP/1.1
   19 1.2862…  127.0.0.1   127.0.0.1    TCP        44 8080 → 55498 [ACK] Seq=1 Ack=706 Win=10233 Len=0
   20 1.2890…  127.0.0.1   127.0.0.1    TCP        56 55500 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   21 1.2891…  127.0.0.1   127.0.0.1    TCP        56 8080 → 55500 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PE
   22 1.2892…  127.0.0.1   127.0.0.1    TCP        44 55500 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
   23 1.2901…  127.0.0.1   127.0.0.1    TCP        70 8080 → 55498 [PSH, ACK] Seq=1 Ack=706 Win=10233 Len=26 [TCP segment of a rea
   24 1.2902…  127.0.0.1   127.0.0.1    TCP        44 55498 → 8080 [ACK] Seq=706 Ack=27 Win=10233 Len=0
   25 1.2903…  127.0.0.1   127.0.0.1    HTTP       44 HTTP/1.1 404 Not Found
   26 1.2903…  127.0.0.1   127.0.0.1    TCP        44 55498 → 8080 [ACK] Seq=706 Ack=28 Win=10233 Len=0
   27 1.2910…  127.0.0.1   127.0.0.1    TCP        44 55498 → 8080 [FIN, ACK] Seq=706 Ack=28 Win=10233 Len=0
   28 1.2910…  127.0.0.1   127.0.0.1    TCP        44 8080 → 55498 [ACK] Seq=28 Ack=707 Win=10233 Len=0
```

Using wireshark capture we see that when we try to request files not in the server, the server will only respond with the message **404 Not Found.**

**Test for 411 Length Required:**

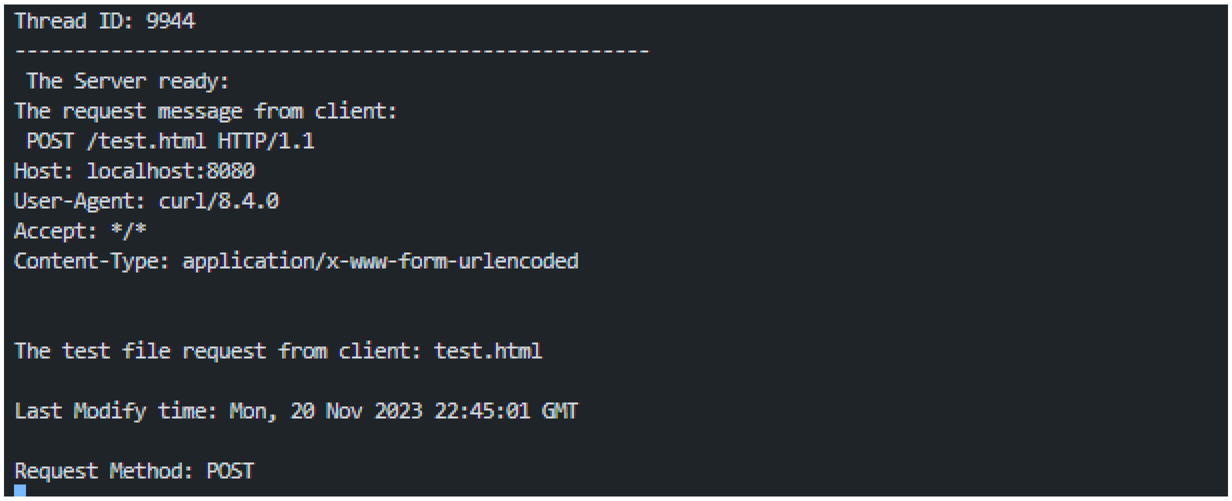
The server will respond with this message when it receives a request that needs a Content-Length section but not found.

The HTTP GET usually does not have a Content-Length section, we can test by POST or PUT request. The POST or PUT usually needs a Content-Length section, so a such request with Content-Length section will respond with message **200 OK.** And such a request without Content-Length section will respond with message **411 Length Required.**

We use the "curl" command line tool to send two conditional HTTP POST requests, one has a Content-Length section and the other does not.

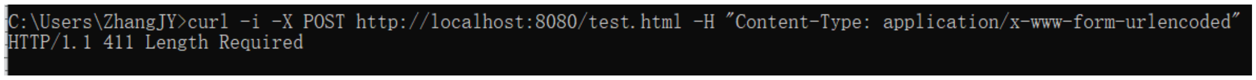
```
curl -i -X POST http://localhost:8080/test.html -H "Content-Type: application/x-www-form-urlencoded"
```

The first one we set without the Content-Length section , we should get the message **411 Length Required.**

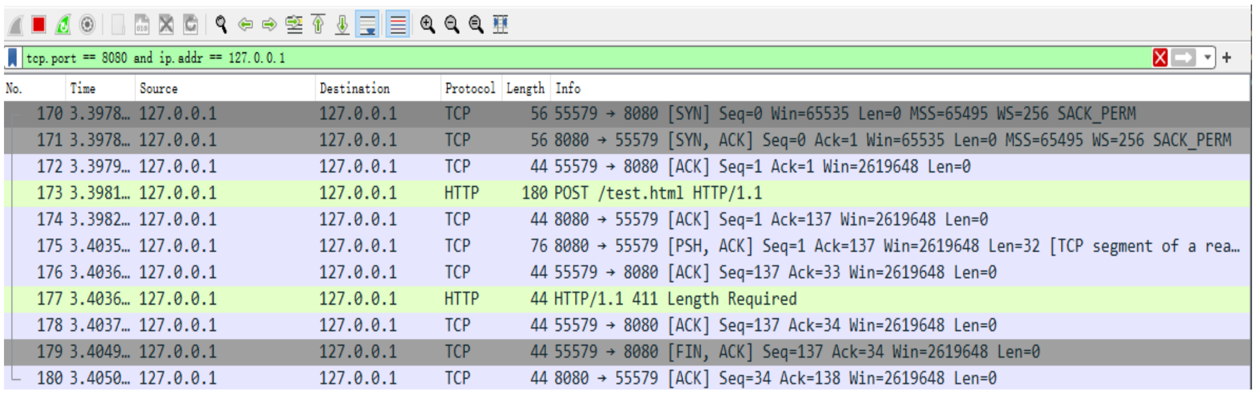


```
Thread ID: 9944
----------------------------------------------------
 The Server ready:
The request message from client:
 POST /test.html HTTP/1.1
Host: localhost:8080
User-Agent: curl/8.4.0
Accept: */*
Content-Type: application/x-www-form-urlencoded


The test file request from client: test.html

Last Modify time: Mon, 20 Nov 2023 22:45:01 GMT

Request Method: POST
```

The server receives the POST requests with no Content-Length section.



```
C:\Users\ZhangJY>curl -i -X POST http://localhost:8080/test.html -H "Content-Type: application/x-www-form-urlencoded"
HTTP/1.1 411 Length Required
```

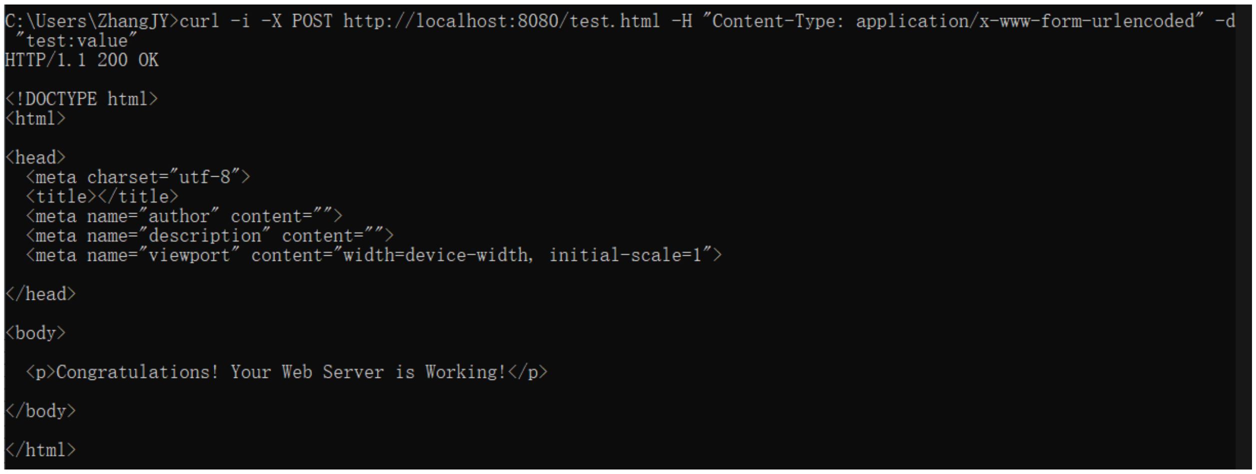


```
tcp.port == 8080 and ip.addr == 127.0.0.1

No.   Time       Source      Destination  Protocol Length Info
  170 3.3978…    127.0.0.1   127.0.0.1    TCP       56 55579 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  171 3.3978…    127.0.0.1   127.0.0.1    TCP       56 8080 → 55579 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  172 3.3979…    127.0.0.1   127.0.0.1    TCP       44 55579 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
  173 3.3981…    127.0.0.1   127.0.0.1    HTTP     180 POST /test.html HTTP/1.1
  174 3.3982…    127.0.0.1   127.0.0.1    TCP       44 8080 → 55579 [ACK] Seq=1 Ack=137 Win=2619648 Len=0
  175 3.4035…    127.0.0.1   127.0.0.1    TCP       76 8080 → 55579 [PSH, ACK] Seq=1 Ack=137 Win=2619648 Len=32 [TCP segment of a rea…
  176 3.4036…    127.0.0.1   127.0.0.1    TCP       44 55579 → 8080 [ACK] Seq=137 Ack=33 Win=2619648 Len=0
  177 3.4036…    127.0.0.1   127.0.0.1    HTTP      44 HTTP/1.1 411 Length Required
  178 3.4037…    127.0.0.1   127.0.0.1    TCP       44 55579 → 8080 [ACK] Seq=137 Ack=34 Win=2619648 Len=0
  179 3.4049…    127.0.0.1   127.0.0.1    TCP       44 55579 → 8080 [FIN, ACK] Seq=137 Ack=34 Win=2619648 Len=0
  180 3.4050…    127.0.0.1   127.0.0.1    TCP       44 8080 → 55579 [ACK] Seq=34 Ack=138 Win=2619648 Len=0
```

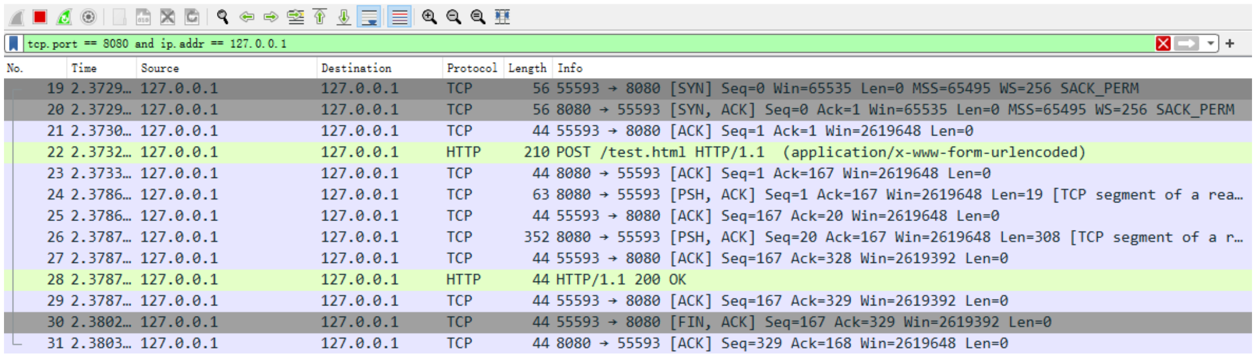
As seen from the client side command line output and wireshark capture the server responds with the message **411 Length Required.**

```
curl -i -X POST http://localhost:8080/test.html -H "Content-Type: application/x-www-form-urlencoded" -d
"test:value"
```

The second one we set with the Content-Length section , we should get the message **200 OK.**



```
C:\Users\ZhangJY>curl -i -X POST http://localhost:8080/test.html -H "Content-Type: application/x-www-form-urlencoded" -d
 "test:value"
HTTP/1.1 200 OK

<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>Congratulations! Your Web Server is Working!</p>

</body>

</html>
```

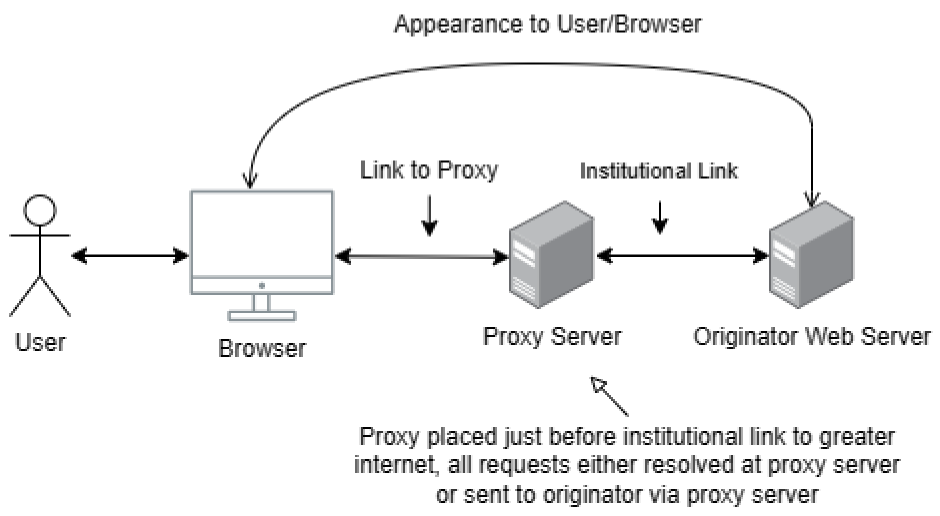


```
tcp.port == 8080 and ip.addr == 127.0.0.1

No.    Time      Source        Destination    Protocol Length Info
    19 2.3729… 127.0.0.1     127.0.0.1      TCP        56 55593 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
    20 2.3729… 127.0.0.1     127.0.0.1      TCP        56 8080 → 55593 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
    21 2.3730… 127.0.0.1     127.0.0.1      TCP        44 55593 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
    22 2.3732… 127.0.0.1     127.0.0.1      HTTP      210 POST /test.html HTTP/1.1  (application/x-www-form-urlencoded)
    23 2.3733… 127.0.0.1     127.0.0.1      TCP        44 8080 → 55593 [ACK] Seq=1 Ack=167 Win=2619648 Len=0
    24 2.3786… 127.0.0.1     127.0.0.1      TCP        63 8080 → 55593 [PSH, ACK] Seq=1 Ack=167 Win=2619648 Len=19 [TCP segment of a rea…
    25 2.3786… 127.0.0.1     127.0.0.1      TCP        44 55593 → 8080 [ACK] Seq=167 Ack=20 Win=2619648 Len=0
    26 2.3787… 127.0.0.1     127.0.0.1      TCP       352 8080 → 55593 [PSH, ACK] Seq=20 Ack=167 Win=2619648 Len=308 [TCP segment of a r…
    27 2.3787… 127.0.0.1     127.0.0.1      TCP        44 55593 → 8080 [ACK] Seq=167 Ack=328 Win=2619392 Len=0
    28 2.3787… 127.0.0.1     127.0.0.1      HTTP       44 HTTP/1.1 200 OK
    29 2.3787… 127.0.0.1     127.0.0.1      TCP        44 55593 → 8080 [ACK] Seq=167 Ack=329 Win=2619392 Len=0
    30 2.3802… 127.0.0.1     127.0.0.1      TCP        44 55593 → 8080 [FIN, ACK] Seq=167 Ack=329 Win=2619392 Len=0
    31 2.3803… 127.0.0.1     127.0.0.1      TCP        44 8080 → 55593 [ACK] Seq=329 Ack=168 Win=2619648 Len=0
```

As seen from the command line and wireshark capture the POST request is successful the client get the message **200 OK.**

## Step Three

a) **What is different in request handling in a proxy server and a web server that hosts your files? Write down the detailed specifications you come up with for a minimal proxy server only using the knowledge you have from module (2) slides 29-34, and implement them (2 points).**



Appearance to User/Browser
Link to Proxy
Institutional Link
User
Browser
Proxy Server
Originator Web Server
Proxy placed just before institutional link to greater internet, all requests either resolved at proxy server or sent to originator via proxy server

Web Proxy Basic Architecture

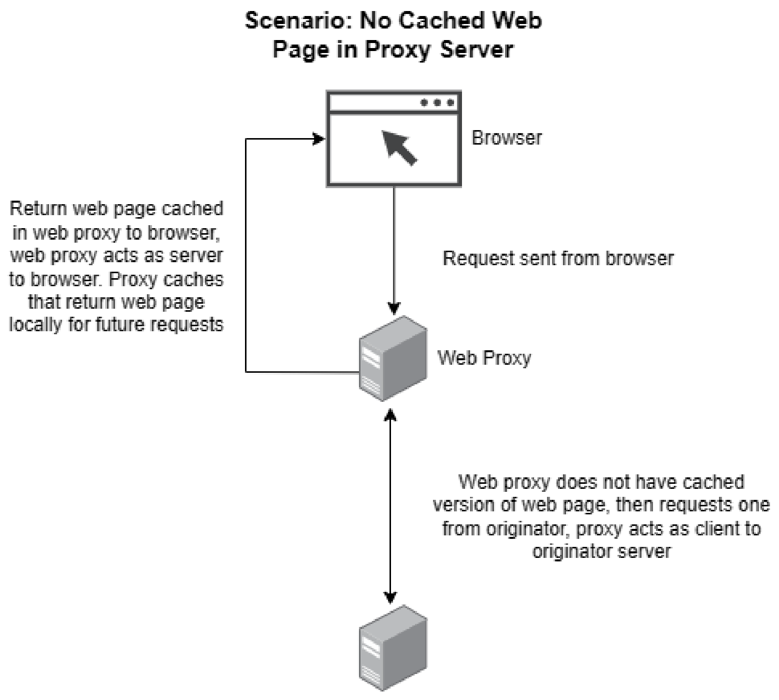
A proxy server acts as an intermediary between a client requesting a page and a server hosting that page. The server hosting the page is the 'originator' or 'creator' of that page and the proxy server has a copy of the page in it. This allows for a proxy server to return a web page to a client, if it has a copy of that page that is up to date. If the proxy server **does not** have a copy, then the proxy server then sends a request to the 'originator' server. If the server **does** have a copy, then we still have to check if the copy is up to date! However, in the scenario that the copy is up to date, then we now need to send back a HTTP 304 code saying that the stored web page is up to date, which is of lower cost than getting the entire web page back from the originator server for that web page. From the perspective of the user, it would appear that there is no proxy at all, and they are directly communicating with the originator web server.

In our own architecture for the proxy server, we'll use 127.0.0.1 for addresses for both the proxy server and webserver, using different ports to imitate being on different links. For the proxy server to web server link we'll use 8080, and from the client to the proxy server we'll use port 80.

b) **Decide the test procedure to show the working of your proxy server. Does this need possible changes at the client side? If your answer is yes, you are not required to implement them, but need to describe and find alternative ways to test your server side functionality.**

The client should not need a change to test the proxy server. The client should actually have minimal awareness of the server. To fully test the proxy server we'd have to consider the three scenarios described above and generate the situations

**No Cached Web Page in Proxy Server:**


Scenario: No Cached Web Page in Proxy Server
Browser
Return web page cached in web proxy to browser, web proxy acts as server to browser. Proxy caches that return web page locally for future requests
Request sent from browser
Web Proxy
Web proxy does not have cached version of web page, then requests one from originator, proxy acts as client to originator server

To test for this scenario we can generate an empty web page cache, as if the web cache in the proxy server had been flushed. We pass on the request to the originator server where we receive back the web page for the proxy server, storing it first in the web cache then sending it to the browser/client.

**Test procedure:**

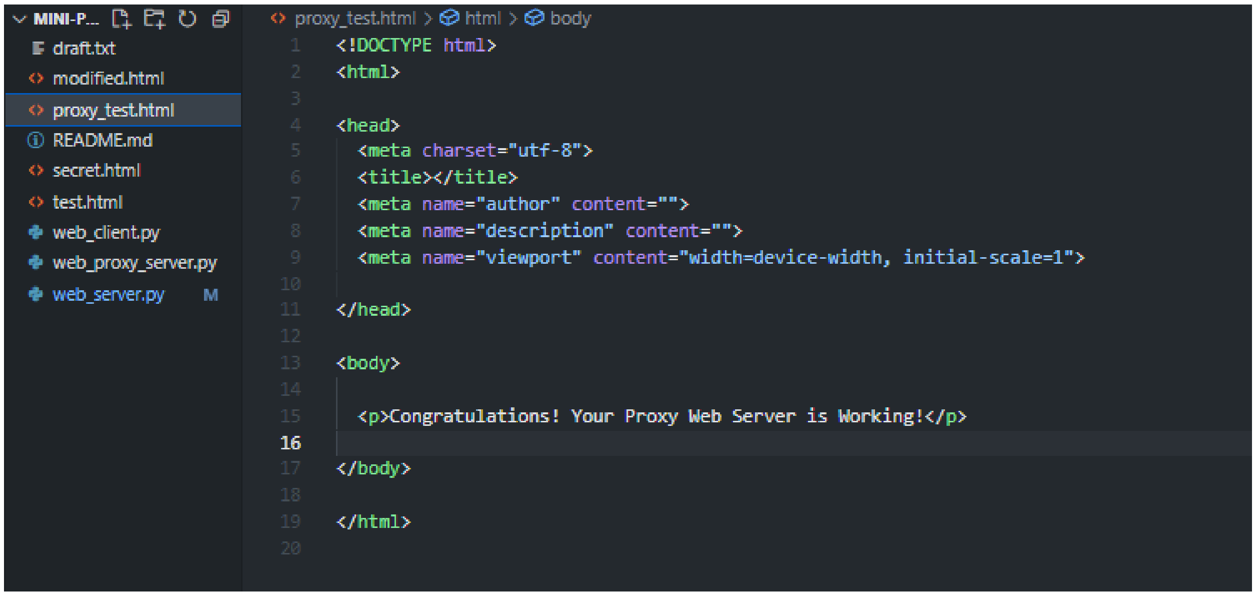
We have our server running and monitoring port 8080, and proxy server running and monitoring port 80.

In the proxy server we have the cache to store the files requested from the server, and these files have the time it stored as the last modified time.

We use the “curl” command line tool to send conditional HTTP Get requests that include different “If-Modified-Since” headers.

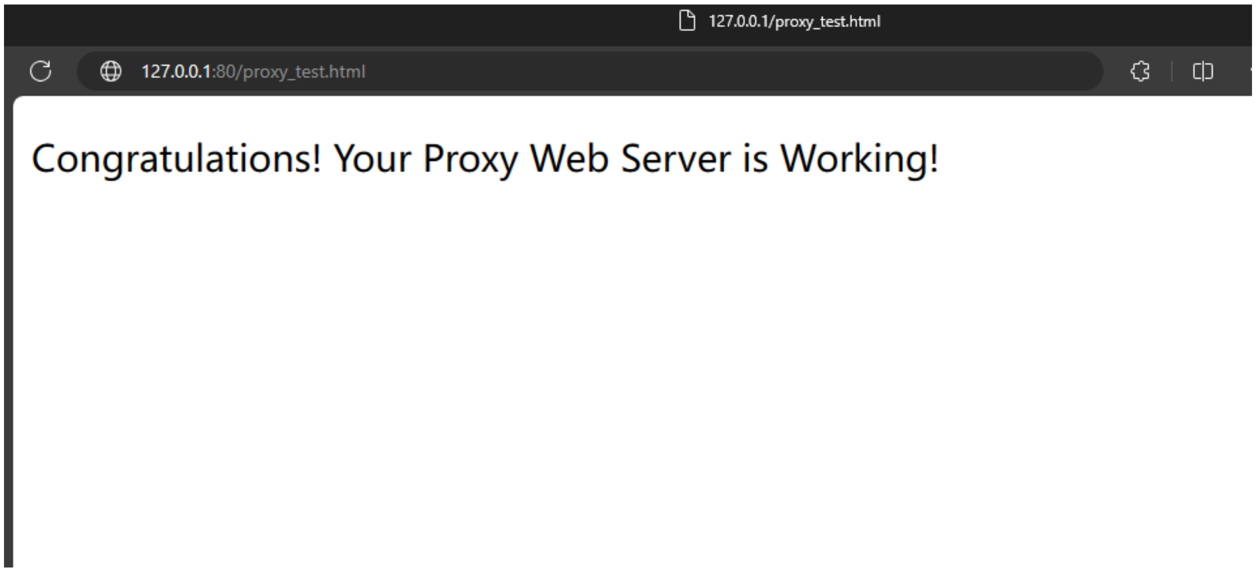
In the above scenario, we do not have a cached web page in the proxy server, so the client will send a request to the proxy server, then the proxy server will send a request to the server.

We request a file proxy_test.html stored in the server initially.



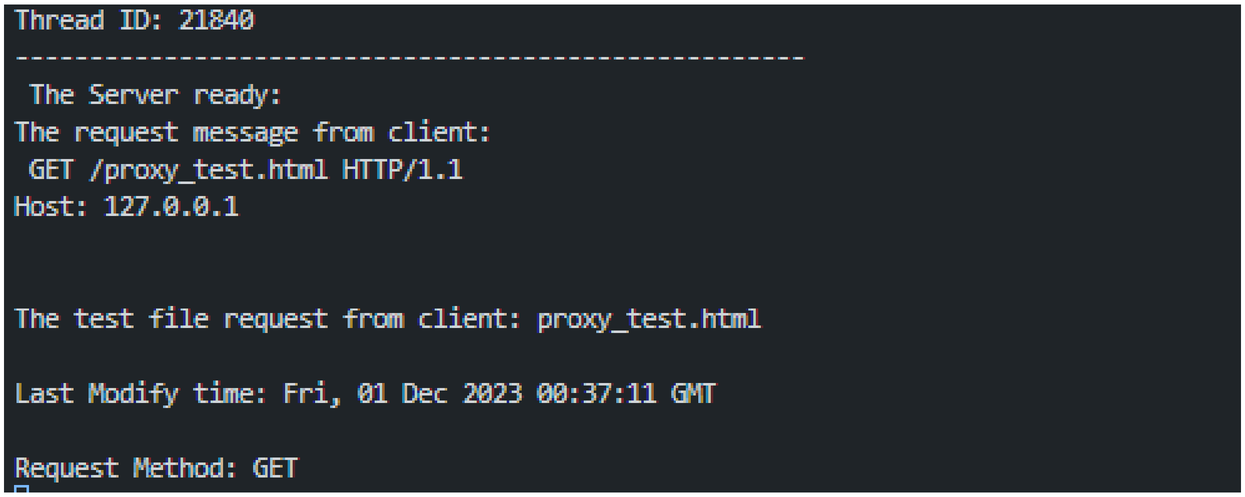
```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>Congratulations! Your Proxy Web Server is Working!</p>

</body>

</html>
```

We simply send a HTTP GET request to our proxy server to access the proxy_test.html file, we visit the http://127.0.0.1:80/proxy_test.html in our browser.



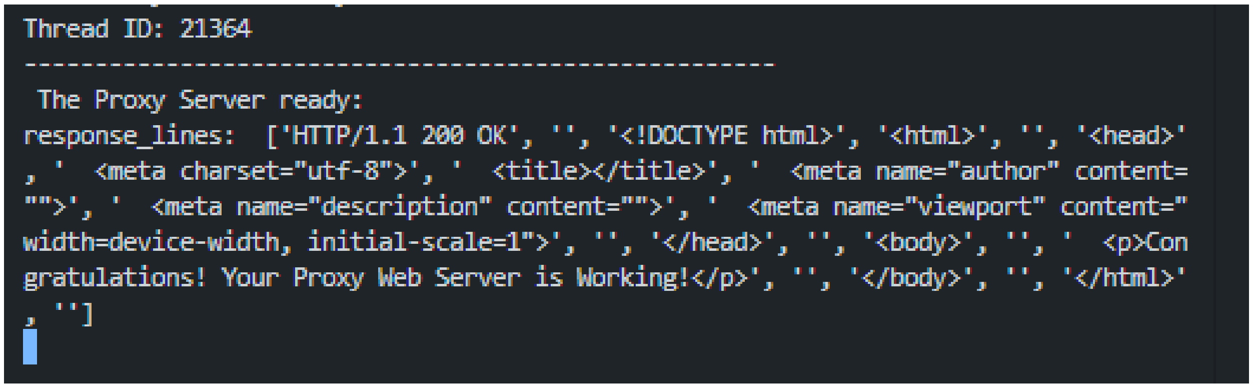
Congratulations! Your Proxy Web Server is Working!

The client client gets the proxy_test.html file and browses this web page.



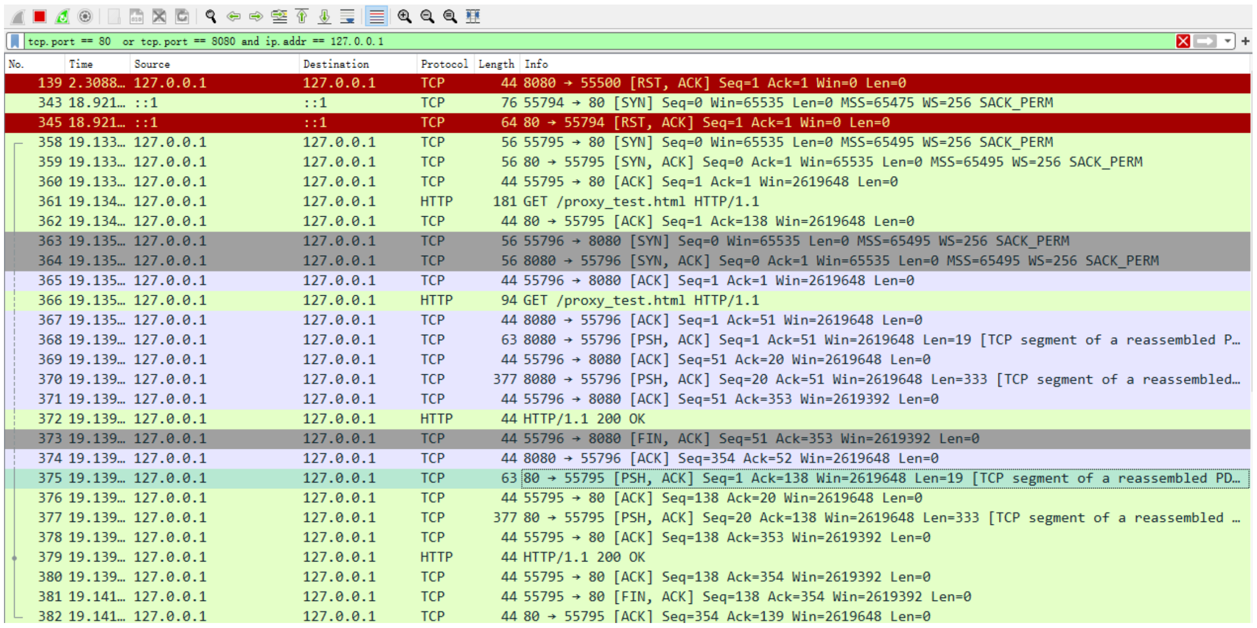
```
Thread ID: 21840
-----------------------------------------------------
 The Server ready:
The request message from client:
 GET /proxy_test.html HTTP/1.1
Host: 127.0.0.1


The test file request from client: proxy_test.html

Last Modify time: Fri, 01 Dec 2023 00:37:11 GMT

Request Method: GET
```

From the server side, it receives requests from the proxy server.



```
Thread ID: 21364
-----------------------------------------------------
 The Proxy Server ready:
response_lines:  ['HTTP/1.1 200 OK', '', '<!DOCTYPE html>', '<html>', '', '<head>'
, '  <meta charset="utf-8">', '  <title></title>', '  <meta name="author" content=
"">', '  <meta name="description" content="">', '  <meta name="viewport" content="
width=device-width, initial-scale=1">', '', '</head>', '', '<body>', '', '  <p>Con
gratulations! Your Proxy Web Server is Working!</p>', '', '</body>', '', '</html>'
, '']
```

From the proxy server side, it got a response from the server with message 200 OK and a request file.

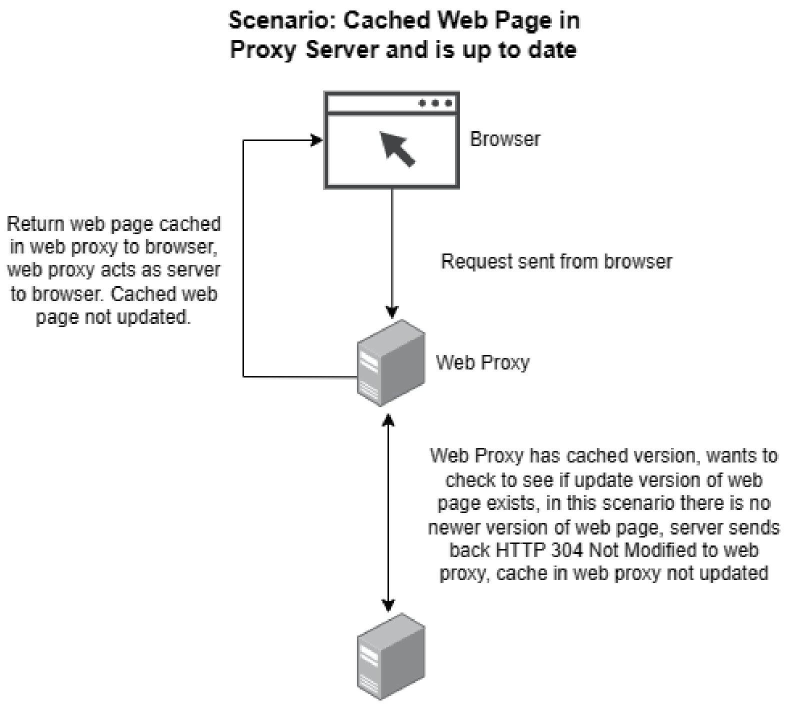


```
tcp.port == 80  or tcp.port == 8080 and ip.addr == 127.0.0.1
No.  Time       Source      Destination  Protocol Length Info
139 2.3088…  127.0.0.1     127.0.0.1     TCP      44 8080 → 55500 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
343 18.921…  ::1           ::1           TCP      76 55794 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
345 18.921…  ::1           ::1           TCP      64 80 → 55794 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
358 19.133…  127.0.0.1     127.0.0.1     TCP      56 55795 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
359 19.133…  127.0.0.1     127.0.0.1     TCP      56 80 → 55795 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
360 19.133…  127.0.0.1     127.0.0.1     TCP      44 55795 → 80 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
361 19.134…  127.0.0.1     127.0.0.1     HTTP    181 GET /proxy_test.html HTTP/1.1
362 19.134…  127.0.0.1     127.0.0.1     TCP      44 80 → 55795 [ACK] Seq=1 Ack=138 Win=2619648 Len=0
363 19.135…  127.0.0.1     127.0.0.1     TCP      56 55796 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
364 19.135…  127.0.0.1     127.0.0.1     TCP      56 8080 → 55796 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
365 19.135…  127.0.0.1     127.0.0.1     TCP      44 55796 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
366 19.135…  127.0.0.1     127.0.0.1     HTTP     94 GET /proxy_test.html HTTP/1.1
367 19.135…  127.0.0.1     127.0.0.1     TCP      44 8080 → 55796 [ACK] Seq=1 Ack=51 Win=2619648 Len=0
368 19.139…  127.0.0.1     127.0.0.1     TCP      63 8080 → 55796 [PSH, ACK] Seq=1 Ack=51 Win=2619648 Len=19 [TCP segment of a reassembled P…
369 19.139…  127.0.0.1     127.0.0.1     TCP      44 55796 → 8080 [ACK] Seq=51 Ack=20 Win=2619648 Len=0
370 19.139…  127.0.0.1     127.0.0.1     TCP     377 8080 → 55796 [PSH, ACK] Seq=20 Ack=51 Win=2619648 Len=333 [TCP segment of a reassembled…
371 19.139…  127.0.0.1     127.0.0.1     TCP      44 55796 → 8080 [ACK] Seq=51 Ack=353 Win=2619392 Len=0
372 19.139…  127.0.0.1     127.0.0.1     HTTP     44 HTTP/1.1 200 OK
373 19.139…  127.0.0.1     127.0.0.1     TCP      44 55796 → 8080 [FIN, ACK] Seq=51 Ack=353 Win=2619392 Len=0
374 19.139…  127.0.0.1     127.0.0.1     TCP      44 8080 → 55796 [ACK] Seq=354 Ack=52 Win=2619648 Len=0
375 19.139…  127.0.0.1     127.0.0.1     TCP      63 80 → 55795 [PSH, ACK] Seq=1 Ack=138 Win=2619648 Len=19 [TCP segment of a reassembled PD…
376 19.139…  127.0.0.1     127.0.0.1     TCP      44 55795 → 80 [ACK] Seq=138 Ack=20 Win=2619648 Len=0
377 19.139…  127.0.0.1     127.0.0.1     TCP     377 80 → 55795 [PSH, ACK] Seq=20 Ack=138 Win=2619648 Len=333 [TCP segment of a reassembled …
378 19.139…  127.0.0.1     127.0.0.1     TCP      44 55795 → 80 [ACK] Seq=138 Ack=353 Win=2619392 Len=0
379 19.139…  127.0.0.1     127.0.0.1     HTTP     44 HTTP/1.1 200 OK
380 19.139…  127.0.0.1     127.0.0.1     TCP      44 55795 → 80 [ACK] Seq=138 Ack=354 Win=2619392 Len=0
381 19.141…  127.0.0.1     127.0.0.1     TCP      44 55795 → 80 [FIN, ACK] Seq=138 Ack=354 Win=2619392 Len=0
382 19.141…  127.0.0.1     127.0.0.1     TCP      44 80 → 55795 [ACK] Seq=354 Ack=139 Win=2619648 Len=0
```

We can see from the wireshark capture, there are 2 HTTP Get requests, first from client port 55795 to port 80 (the proxy server), and the second from random distributed proxy server 55796 to port 8080 (server). There are also following corresponding responses from server to proxy server, and from proxy server to client.

- The proxy server binds locally to port 80 and is used to listen for connections from clients. However, when the proxy server communicates with the server, the local port it uses is dynamically allocated. It connects to port 8080 of the server at 127.0.0.1.

**Cached Web Page in Proxy Server and is up to date:**


Scenario: Cached Web Page in Proxy Server and is up to date
Browser
Return web page cached in web proxy to browser, web proxy acts as server to browser. Cached web page not updated.
Request sent from browser
Web Proxy
Web Proxy has cached version, wants to check to see if update version of web page exists, in this scenario there is no newer version of web page, server sends back HTTP 304 Not Modified to web proxy, cache in web proxy not updated

To test this scenario in the diagram above, we could generate an up to date cached web page in the proxy server, issue a Conditional-Get to the web server, receive back a 304 Not Modified code and pass on our cached page to the client/browser.

**Test procedure:**

We have our server running on port 8080, and proxy server running on port 80.

In the proxy server we have the cache to store the files requested from the server, and these files have the time it requested as the last modified time.

We use the “curl” command line tool to send conditional HTTP Get requests that include different “If-Modified-Since” headers.

In the above scenario, we already have a cached web page in the proxy server and it is up-to-date, so the client will send a request to the proxy server and directly get a response from it.

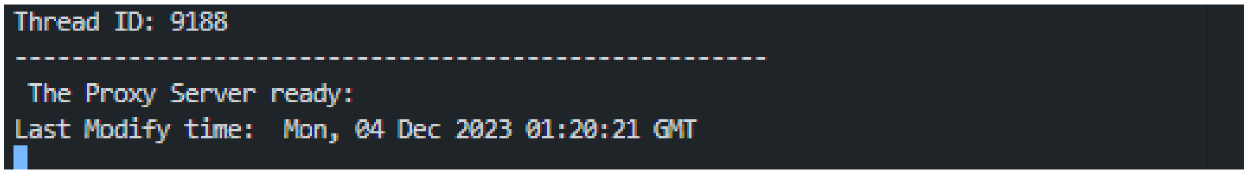
We send one conditional HTTP Get request that includes an “If-Modified-Since” header, such that the If-Modified-Since date is after the time last modified.

```
curl -i http://localhost/proxy_test.html -H "If-Modified-Since: Sat, 21 Dec 2023 07:28:00 GMT"
```

From the server side, there will be no response as the up-to-date file is already in the proxy server's cache.

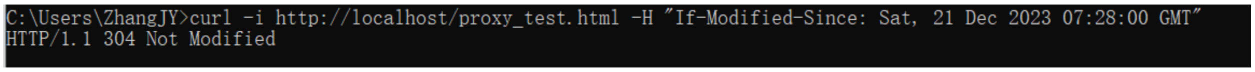
From proxy server side, it found the request file in its cache and is up to date

(Last Modify Time < If-Modified-Since):

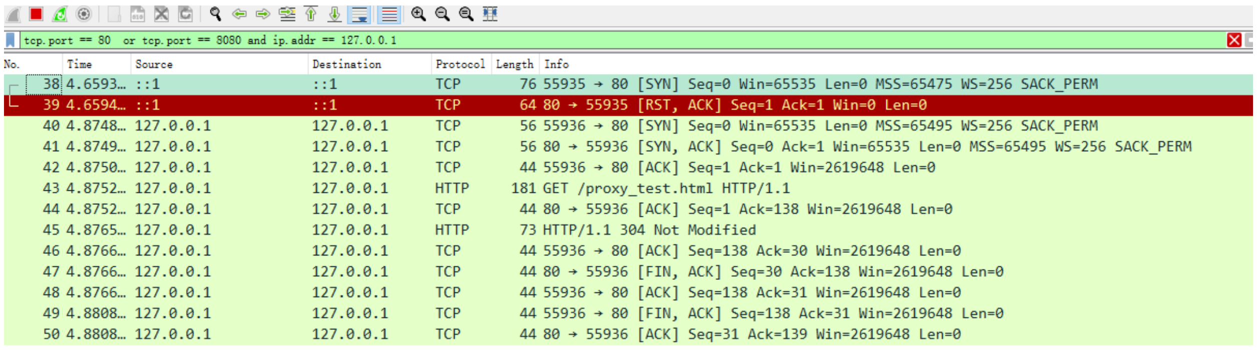


```
Thread ID: 9188
-----------------------------------------------------
 The Proxy Server ready:
Last Modify time:  Mon, 04 Dec 2023 01:20:21 GMT
```

And the client got the 304 Not Modified message



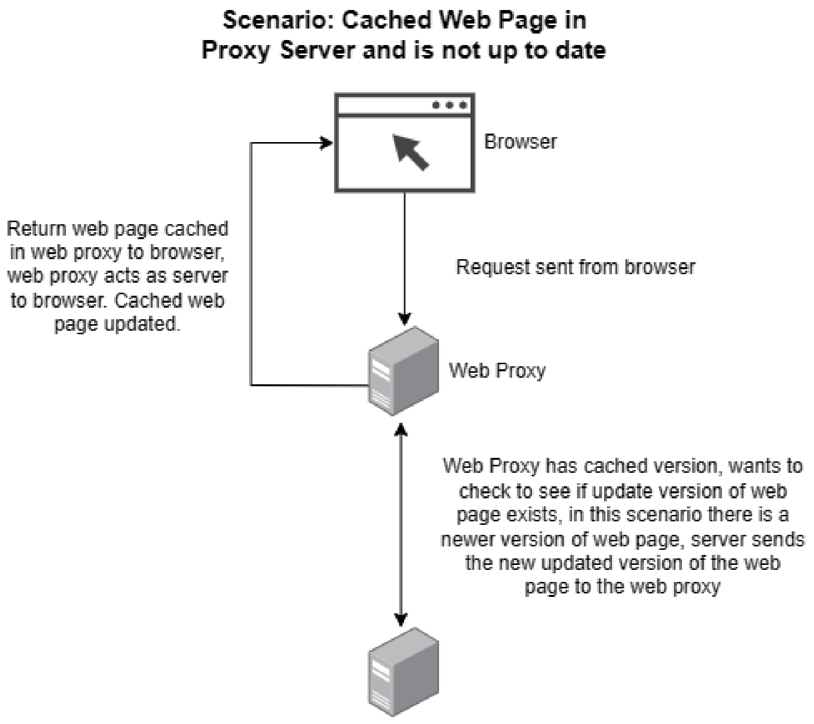
```
C:\Users\ZhangJY>curl -i http://localhost/proxy_test.html -H "If-Modified-Since: Sat, 21 Dec 2023 07:28:00 GMT"
HTTP/1.1 304 Not Modified
```



```
tcp.port == 80  or tcp.port == 8080 and ip.addr == 127.0.0.1

No.  Time      Source     Destination  Protocol Length Info
 38 4.6593… ::1        ::1          TCP       76 55935 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
 39 4.6594… ::1        ::1          TCP       64 80 → 55935 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 40 4.8748… 127.0.0.1  127.0.0.1    TCP       56 55936 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 41 4.8749… 127.0.0.1  127.0.0.1    TCP       56 80 → 55936 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 42 4.8750… 127.0.0.1  127.0.0.1    TCP       44 55936 → 80 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
 43 4.8752… 127.0.0.1  127.0.0.1    HTTP     181 GET /proxy_test.html HTTP/1.1
 44 4.8752… 127.0.0.1  127.0.0.1    TCP       44 80 → 55936 [ACK] Seq=1 Ack=138 Win=2619648 Len=0
 45 4.8765… 127.0.0.1  127.0.0.1    HTTP      73 HTTP/1.1 304 Not Modified
 46 4.8766… 127.0.0.1  127.0.0.1    TCP       44 55936 → 80 [ACK] Seq=138 Ack=30 Win=2619648 Len=0
 47 4.8766… 127.0.0.1  127.0.0.1    TCP       44 80 → 55936 [FIN, ACK] Seq=30 Ack=138 Win=2619648 Len=0
 48 4.8766… 127.0.0.1  127.0.0.1    TCP       44 55936 → 80 [ACK] Seq=138 Ack=31 Win=2619648 Len=0
 49 4.8808… 127.0.0.1  127.0.0.1    TCP       44 55936 → 80 [FIN, ACK] Seq=138 Ack=31 Win=2619648 Len=0
 50 4.8808… 127.0.0.1  127.0.0.1    TCP       44 80 → 55936 [ACK] Seq=31 Ack=139 Win=2619648 Len=0
```

We can see from the wireshark capture, there are only one HTTP Get requests to port 80 (the proxy server),  and as the file is already in the proxy server's cache, there is only one 304 Not Modified response message to the client.

**Cached Web Page in Proxy Server and is not up to date:**


Scenario: Cached Web Page in Proxy Server and is not up to date
Browser
Return web page cached in web proxy to browser, web proxy acts as server to browser. Cached web page updated.
Request sent from browser
Web Proxy
Web Proxy has cached version, wants to check to see if update version of web page exists, in this scenario there is a newer version of web page, server sends the new updated version of the web page to the web proxy

To test this scenario, we can generate a cached web page, however the If-modified-since: <date> would be of an earlier date in proxy servers cached web page, than the date the web page on the originator server, thus when the proxy server issues a Conditional-Get, we get back a web page and a HTTP 200 code instead of a 304 Not-Modified code. Thus we update the proxy servers' cached web page, and then send it to the client/browser.

**Test procedure:**

We have our server running on port 8080, and proxy server running on port 80.

In the proxy server we have the cache to store the files requested from the server, and these files have the time it requested as the last modified time.

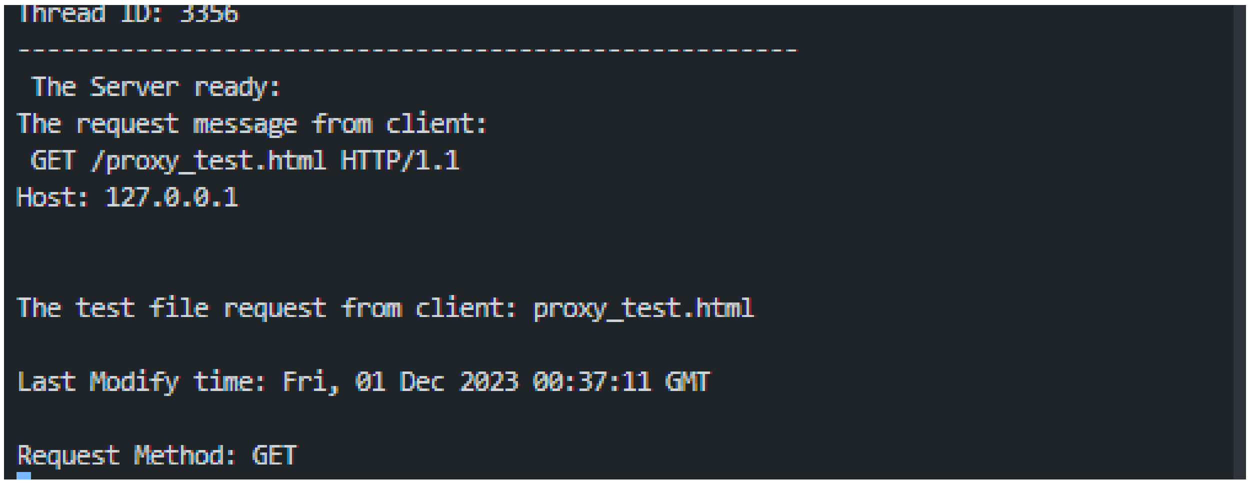
We use the “curl” command line tool to send conditional HTTP Get requests that include different “If-Modified-Since” headers.

In the above scenario, we already have a cached web page in the proxy server and it is not up-to-date, so the client will send a request to the proxy server and the proxy server will request the new version of the web page from the server.

We send one conditional HTTP Get request that includes an “If-Modified-Since” header, such that the If-Modified-Since date is before the time last modified.

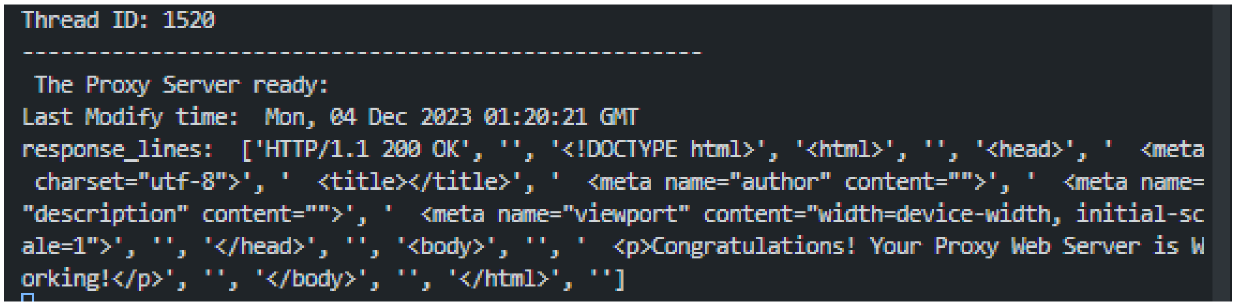
```
curl -i http://localhost/proxy_test.html -H "If-Modified-Since: Sat, 21 Oct 2023 07:28:00 GMT"
```

From the server side, it got a request from the proxy server.



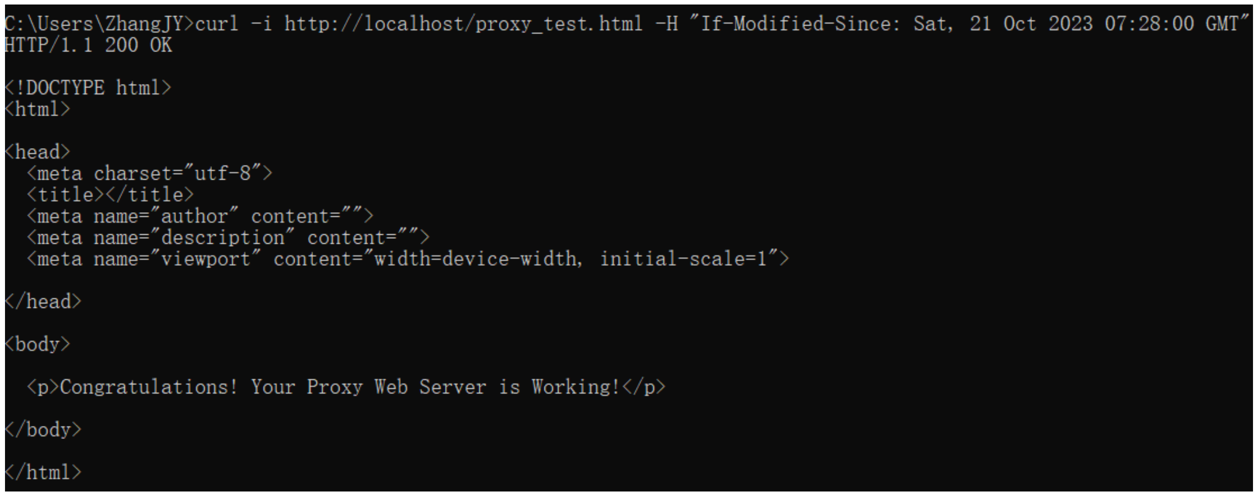
```
Thread ID: 3356
------------------------------------------------------
 The Server ready:
The request message from client:
 GET /proxy_test.html HTTP/1.1
Host: 127.0.0.1


The test file request from client: proxy_test.html

Last Modify time: Fri, 01 Dec 2023 00:37:11 GMT

Request Method: GET
```

From the proxy server side, it got a response from the server with message 200 OK and a request file, and it stored the file into its cache.

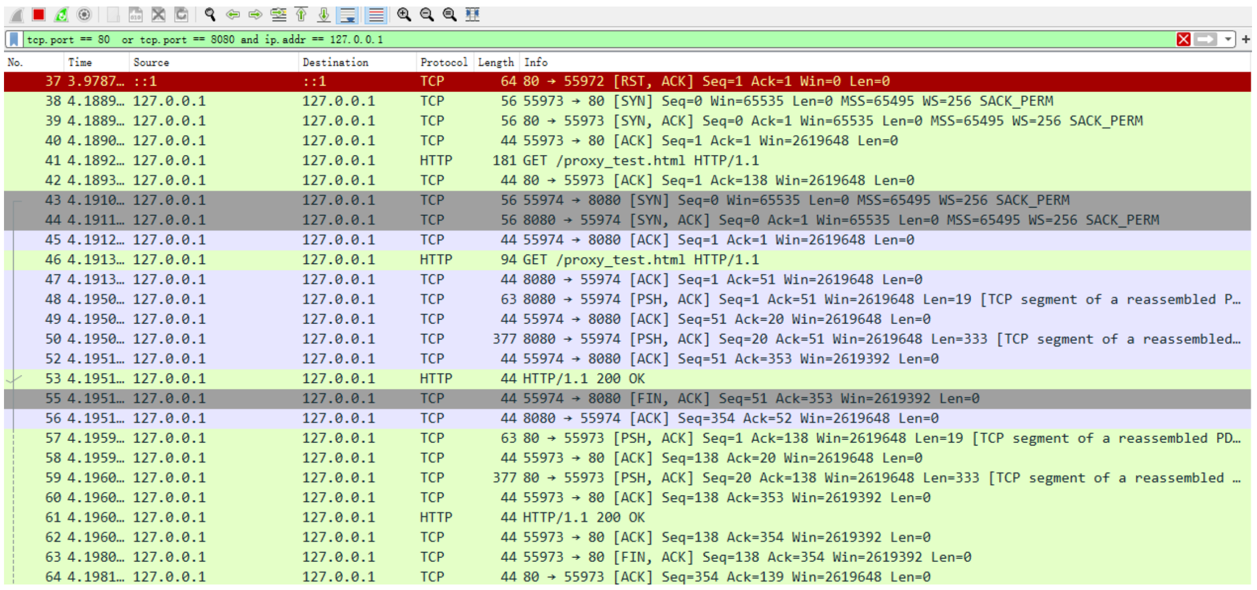


```
Thread ID: 1520
----------------------------------------------------
 The Proxy Server ready:
Last Modify time:  Mon, 04 Dec 2023 01:20:21 GMT
response_lines:  ['HTTP/1.1 200 OK', '', '<!DOCTYPE html>', '<html>', '', '<head>', '  <meta
 charset="utf-8">', '  <title></title>', '  <meta name="author" content="">', '  <meta name=
"description" content="">', '  <meta name="viewport" content="width=device-width, initial-sc
ale=1">', '', '</head>', '', '<body>', '', '  <p>Congratulations! Your Proxy Web Server is W
orking!</p>', '', '</body>', '', '</html>', '']
```

And finally the client got the 200 OK message and the request file.



```
C:\Users\ZhangJY>curl -i http://localhost/proxy_test.html -H "If-Modified-Since: Sat, 21 Oct 2023 07:28:00 GMT"
HTTP/1.1 200 OK

<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>Congratulations! Your Proxy Web Server is Working!</p>

</body>

</html>
```



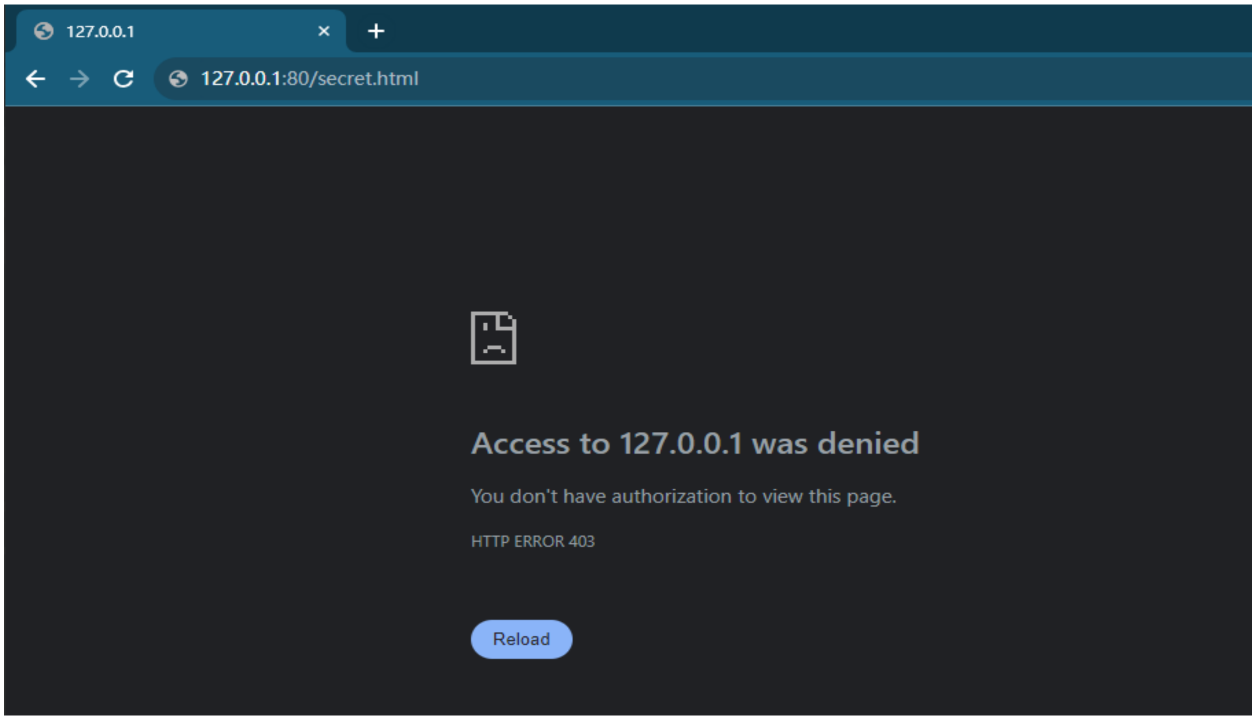
```
tcp.port == 80  or tcp.port == 8080 and ip.addr == 127.0.0.1
No.  Time      Source     Destination  Protocol Length Info
37 3.9787… ::1         ::1          TCP       64 80 → 55972 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
38 4.1889… 127.0.0.1   127.0.0.1    TCP       56 55973 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
39 4.1889… 127.0.0.1   127.0.0.1    TCP       56 80 → 55973 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
40 4.1890… 127.0.0.1   127.0.0.1    TCP       44 55973 → 80 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
41 4.1892… 127.0.0.1   127.0.0.1    HTTP     181 GET /proxy_test.html HTTP/1.1
42 4.1893… 127.0.0.1   127.0.0.1    TCP       44 80 → 55973 [ACK] Seq=1 Ack=138 Win=2619648 Len=0
43 4.1910… 127.0.0.1   127.0.0.1    TCP       56 55974 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
44 4.1911… 127.0.0.1   127.0.0.1    TCP       56 8080 → 55974 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
45 4.1912… 127.0.0.1   127.0.0.1    TCP       44 55974 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
46 4.1913… 127.0.0.1   127.0.0.1    HTTP      94 GET /proxy_test.html HTTP/1.1
47 4.1913… 127.0.0.1   127.0.0.1    TCP       44 8080 → 55974 [ACK] Seq=1 Ack=51 Win=2619648 Len=0
48 4.1950… 127.0.0.1   127.0.0.1    TCP       63 8080 → 55974 [PSH, ACK] Seq=1 Ack=51 Win=2619648 Len=19 [TCP segment of a reassembled P…
49 4.1950… 127.0.0.1   127.0.0.1    TCP       44 55974 → 8080 [ACK] Seq=51 Ack=20 Win=2619648 Len=0
50 4.1950… 127.0.0.1   127.0.0.1    TCP      377 8080 → 55974 [PSH, ACK] Seq=20 Ack=51 Win=2619648 Len=333 [TCP segment of a reassembled…
52 4.1951… 127.0.0.1   127.0.0.1    TCP       44 55974 → 8080 [ACK] Seq=51 Ack=353 Win=2619392 Len=0
53 4.1951… 127.0.0.1   127.0.0.1    HTTP      44 HTTP/1.1 200 OK
55 4.1951… 127.0.0.1   127.0.0.1    TCP       44 55974 → 8080 [FIN, ACK] Seq=51 Ack=353 Win=2619392 Len=0
56 4.1951… 127.0.0.1   127.0.0.1    TCP       44 8080 → 55974 [ACK] Seq=354 Ack=52 Win=2619648 Len=0
57 4.1959… 127.0.0.1   127.0.0.1    TCP       63 80 → 55973 [PSH, ACK] Seq=1 Ack=138 Win=2619648 Len=19 [TCP segment of a reassembled PD…
58 4.1959… 127.0.0.1   127.0.0.1    TCP       44 55973 → 80 [ACK] Seq=138 Ack=20 Win=2619648 Len=0
59 4.1960… 127.0.0.1   127.0.0.1    TCP      377 80 → 55973 [PSH, ACK] Seq=20 Ack=138 Win=2619648 Len=333 [TCP segment of a reassembled …
60 4.1960… 127.0.0.1   127.0.0.1    TCP       44 55973 → 80 [ACK] Seq=138 Ack=353 Win=2619392 Len=0
61 4.1960… 127.0.0.1   127.0.0.1    HTTP      44 HTTP/1.1 200 OK
62 4.1960… 127.0.0.1   127.0.0.1    TCP       44 55973 → 80 [ACK] Seq=138 Ack=354 Win=2619392 Len=0
63 4.1980… 127.0.0.1   127.0.0.1    TCP       44 55973 → 80 [FIN, ACK] Seq=138 Ack=354 Win=2619392 Len=0
64 4.1981… 127.0.0.1   127.0.0.1    TCP       44 80 → 55973 [ACK] Seq=354 Ack=139 Win=2619648 Len=0
```

We can see from the wireshark capture, there are 2 HTTP Get requests, first from client port 55795 to port 80 (the proxy server), and the second from random distributed proxy server 55796 to port 8080 (server). There are also following corresponding responses from server to proxy server, and from proxy server to client.

- The proxy server binds locally to port 80 and is used to listen for connections from clients. However, when the proxy server communicates with the server, the local port it uses is dynamically allocated. It connects to port 8080 of the server at 127.0.0.1.

This would work similarly to requesting pages that result in 403, 404, and other messages we implement on the server.

Simple case for 403:

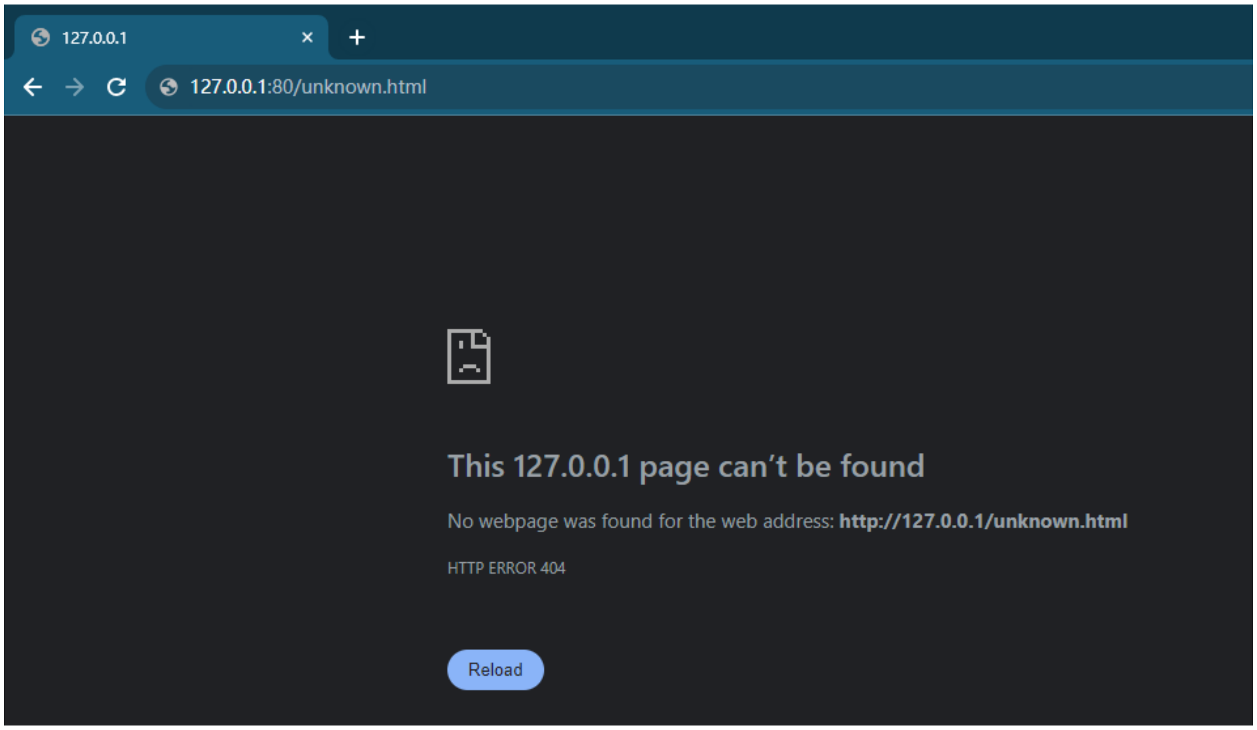


Access to 127.0.0.1 was denied

You don't have authorization to view this page.

HTTP ERROR 403

Simple case for 404:



This 127.0.0.1 page can't be found

No webpage was found for the web address: **http://127.0.0.1/unknown.html**

HTTP ERROR 404

## Step Four (Bonus Point):

**Does your server have an HOL problem? If yes, make changes in your server to avoid it and explain what you have done**. If not, explain why your server does not have this problem.

Our original server had a head-of-line blocking problem with a single connection occupying the server's attention until it was complete.

To avoid head-of-line blocking, we implemented in our servers a thread for each incoming connection. This allows us to take advantage of the operating systems scheduling/context switching to effectively divide up the sent objects into chunks. After a connection is accepted by the server we spin up a new thread each time to handle that connection and go back to listening for more incoming connections. After each thread when we close the connection and end the function call the thread will die off.

Our proxy server spins up a thread each connection then creates a connection, spawning a thread in our web server, only after the proxy has received all data will it finish its thread. Another change for our proxy server: we have created a lock when we write in the cache to prevent data races, since many threads can write into the cache at once, and many can read. This would allow many readers of the cache but only one writer, and no readers whilst writing is being done.