

Lab 2

Computer Networks Lab (CS302)



Team Members:

Alan Tony (191CS207)

Aashish Prateek (191CS225)

Question 1:

Code:

htdocs/index.html

```
<html>
<head>
  <title>Hello World</title>
  <link rel="shortcut icon" href="#" />
</head>
<body>
  <h1>Hello World!</h1>
</body>
</html>
```

P.T.O

HTTPServer.py

```
import socket

# Define socket host and port
SERVER_HOST = '0.0.0.0'
SERVER_PORT = 8000

# Create socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((SERVER_HOST, SERVER_PORT))
server_socket.listen(1)
print('Listening on port %s ...' % SERVER_PORT)

while True:
    # Wait for client connections
    client_connection, client_address = server_socket.accept()

    # Get the client request
    request = client_connection.recv(1024).decode()
    print(request)

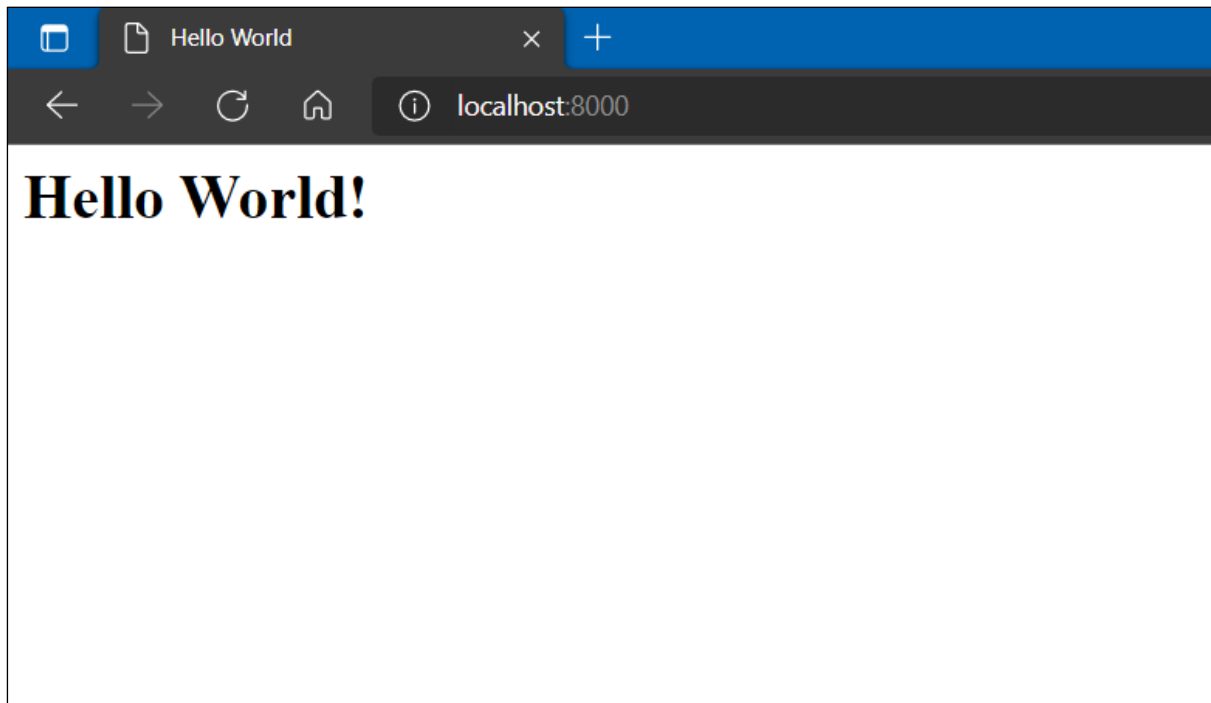
    try:
        # Get the content of webpage
        file_name = request.split()[1]
        if file_name == "/":
            file_name = "/index.html"
        fin = open('htdocs' + file_name)
        content = fin.read()
        fin.close()
        response = 'HTTP/1.0 200 OK\n\n' + content
    except:
        content = "<h1>Content not found</h1>"
        response = 'HTTP/1.0 404 Not Found\n\n' + content

    # Send HTTP response
    client_connection.sendall(response.encode())
    client_connection.close()
```

Output:

```
Anaconda Prompt (anaconda3) - python: HTTPServer.py
(base) C:\Users\alant>cd "C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab2\Q1"
(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab2\Q1>python HTTPServer.py
listening on port 8080 ...
GET / HTTP/1.1
Host: localhost:8080
Connection: keep-alive
sec-ch-ua: "Chromium";v="94", "Microsoft Edge";v="94", ";Not A Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36 Edg/94.0.992.31
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

GET / HTTP/1.1
Host: localhost:8080
Connection: keep-alive
sec-ch-ua: "Chromium";v="94", "Microsoft Edge";v="94", ";Not A Brand";v="99"
DNT: 1
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36 Edg/94.0.992.31
Accept: image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:8080/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```



Question 2:

Code:

HostToIP.py

```
import socket

hostname = input("Enter host name: ")
ip = socket.gethostbyname(hostname)
print("IP address: ", ip)
```

Output:

```
Anaconda Prompt (anaconda3)

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab2\Q2>python HostToIP.py
Enter host name: www.wikipedia.org
IP address: 103.102.166.224

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab2\Q2>
```

IPToHost.py

```
import socket

IP = input("Enter IP address: ")
hostname = socket.gethostbyaddr(IP)
print("Host name: ", hostname)
```

Output:

```
Anaconda Prompt (anaconda3)

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab2\Q2>python IPToHost.py
Enter IP address: 103.102.166.224
Host name: ('text-lb.eqs.in.wikimedia.org', [], ['103.102.166.224'])

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab2\Q2>_
```

P.T.O

Question 3:

Code:

ImdbTop50.py

```
import requests
import bs4
from datetime import datetime

curr_date = str(datetime.now()).split()[0]

#The release date is a parameter in the HTTP request with the following format
# :
#release_date=from_date,till_date
#Commas are required to separate multiple value for a single parameter
#Since I want movies of all time, from_date is left blank, till_date is put as
# current date
URL="http://www.imdb.com/search/title?release_date=" + curr_date

r = requests.get(URL)

soup = bs4.BeautifulSoup(r.content, 'html5lib')

table = soup.find_all('div', attrs = {'class' : 'lister-item-content'})
for i, content in enumerate(table):

    movie_name = content.h3.a.text
    rating = content.find('span', attrs = {'class' : {'rating-
rating'}}).span.text

    actors = content.find('p', attrs = {'class' : ''})
    actors = [name.text for name in actors.find_all('a')]

    print("{} . {} ({{}}) - Ft. ".format(i+1, movie_name, rating), end='')
    print(*actors, sep=', ')
```

Output:

```
Anaconda Prompt (anaconda3)

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab2\Q3>python ImdbTop50.py
1. Squid Game (8.3) - Ft. Lee Jung-jae, Greg Chun, Stephen Fu, Tom Choi
2. Midnight Mass (7.8) - Ft. Kate Siegel, Zach Gilford, Kristin Lehman, Samantha Sloyan
3. No Time to Die (7.6) - Ft. Cary Joji Fukunaga, Daniel Craig, Ana de Armas, Rami Malek, Léa Seydoux
4. Dune (8.4) - Ft. Denis Villeneuve, Timothée Chalamet, Rebecca Ferguson, Zendaya, Oscar Isaac
5. Sex Education (8.3) - Ft. Asa Butterfield, Gillian Anderson, Emma Mackey, Ncuti Gatwa
6. Free Guy (7.3) - Ft. Shawn Levy, Ryan Reynolds, Jodie Comer, Taika Waititi, Lil Rel Howery
7. The Many Saints of Newark (6.5) - Ft. Alan Taylor, Alessandro Nivola, Leslie Odom Jr., Jon Bernthal, Vera Farmiga
8. Venom: Let There Be Carnage (6.6) - Ft. Andy Serkis, Tom Hardy, Woody Harrelson, Michelle Williams, Naomie Harris
9. Ted Lasso (8.8) - Ft. Jason Sudeikis, Brett Goldstein, Brendan Hunt, Nick Mohammed
10. Foundation (7.5) - Ft. Lee Pace, Lou Llobell, Laura Birn, Terrence Mann
11. What If...? (7.6) - Ft. Jeffrey Wright, Matthew Wood, Terri Douglas, Robin Atkin Downes
12. The Walking Dead (8.2) - Ft. Andrew Lincoln, Norman Reedus, Melissa McBride, Lauren Cohan
13. The Sopranos (9.2) - Ft. James Gandolfini, Lorraine Bracco, Edie Falco, Michael Imperioli
14. American Horror Story (8) - Ft. Lady Gaga, Kathy Bates, Angela Bassett, Sarah Paulson
15. The Guilty (6.3) - Ft. Antoine Fuqua, Jake Gyllenhaal, Riley Keough, Peter Sarsgaard, Christina Vidal
16. Only Murders in the Building (8.3) - Ft. Steve Martin, Martin Short, Selena Gomez, Aaron Dominguez
17. Shang-Chi and the Legend of the Ten Rings (7.9) - Ft. Destin Daniel Cretton, Simu Liu, Awkwafina, Tony Chiu-Wai Leung, Ben Kingsley
18. Nine Perfect Strangers (7.1) - Ft. Nicole Kidman, Melissa McCarthy, Michael Shannon, Luke Evans
19. Lucifer (8.1) - Ft. Tom Ellis, Lauren German, Kevin Alejandro, D.B. Woodside
20. La Brea (5.1) - Ft. Natalie Zea, Foin Macken, Chiké Okonkwo, Karina Logue
21. Stranger Things (8.7) - Ft. Millie Bobby Brown, Finn Wolfhard, Winona Ryder, David Harbour
22. Grey's Anatomy (7.5) - Ft. Ellen Pompeo, Chandra Wilson, James Pickens Jr., Justin Chambers
23. The Starling (6.3) - Ft. Theodore Melfi, Melissa McCarthy, Chris O'Dowd, Kevin Kline, Timothy Olyphant
24. The Morning Show (8.4) - Ft. Jennifer Aniston, Reese Witherspoon, Billy Crudup, Nestor Carbonell
25. Game of Thrones (9.2) - Ft. Emilia Clarke, Peter Dinklage, Kit Harington, Lena Headey
26. Goliath (8.2) - Ft. Billy Bob Thornton, Nina Arianda, Tania Raymonde, Diana Hopper
27. Malignant (6.3) - Ft. James Wan, Annabelle Wallis, Maddie Hasson, George Young, Michole Briana White
28. The Chestnut Man (8.1) - Ft. Esben Dalgaard Andersen, Danica Curcic, David Dencik, Iben Dörner
29. La casa de papel (8.2) - Ft. Úrsula Corberó, Álvaro Morte, Itziar Ituño, Pedro Alonso
30. Vigil (7.4) - Ft. Suranne Jones, Rose Leslie, Shaun Evans, Paterson Joseph
31. Star Wars: Visions (7.3) - Ft. Michael Sinterniklaas, Neil Kaplan, Adam Sietz, JP Karliak
32. Y: The Last Man (6.3) - Ft. Diane Lane, Ashley Romans, Ben Schnetzer, Olivia Thirlby
33. Doom Patrol (7.9) - Ft. April Bowlby, Diane Guerrero, Matt Bomer, Brendan Fraser
34. Old (5.8) - Ft. M. Night Shyamalan, Gael García Bernal, Vicky Krieps, Rufus Sewell, Alex Wolff
35. Clickbait (7.3) - Ft. Zoe Kazan, Betty Gabriel, Phoenix Raei, Adrian Grenier
36. The Last Duel (7.4) - Ft. Ridley Scott, Jodie Comer, Matt Damon, Ben Affleck, Adam Driver
37. Dear Evan Hansen (6.1) - Ft. Stephen Chbosky, Ben Platt, Julianne Moore, Kaitlyn Dever, Amy Adams
38. Alice in Borderland (7.7) - Ft. Kento Yamazaki, Tao Tsuchiya, Nijirō Murakami, Keita Machida
39. Billions (8.4) - Ft. Paul Giamatti, Damian Lewis, Maggie Siff, David Costabile
40. Intrusion (5.2) - Ft. Adam Salky, Freida Pinto, Logan Marshall-Green, Robert John Burke, Megan Elisabeth Kelly
41. NCIS: Naval Criminal Investigative Service (7.7) - Ft. Mark Harmon, David McCallum, Sean Murray, Pauley Perrette
42. No One Gets Out Alive (5.3) - Ft. Santiago Menghini, Phil Robertson, Joana Bonja, Victoria Alcock, Cristina Rodlo
43. The Rookie (8) - Ft. Nathan Fillion, Alyssa Diaz, Richard T. Jones, Melissa O'Neil
44. Promising Young Woman (7.5) - Ft. Emerald Fennell, Carey Mulligan, Bo Burnham, Allison Brie, Clancy Brown
45. Wolf (4.2) - Ft. Nathalie Biancheri, Lily-Rose Depp, George MacKay, Paddy Considine, Terry Notary
46. Dune (6.4) - Ft. David Lynch, Kyle MacLachlan, Virginia Madsen, Francesca Annis, Leonardo Cimino
47. The Harder They Fall (6.2) - Ft. Jeymes Samuel, Zazie Beetz, Idris Elba, Regina King, Jonathan Majors
48. Titans (7.6) - Ft. Brenton Thwaites, Anna Diop, Ryan Potter, Teagan Croft
49. Cry Macho (5.8) - Ft. Clint Eastwood, Clint Eastwood, Dwight Yoakam, Daniel V. Graulau, Amber Lynn Ashley
50. American Crime Story (8.4) - Ft. Sarah Paulson, Annaleigh Ashford, Sterling K. Brown, Kenneth Choi
```

P.T.O

Question 4:

Code:

GETRequestDetails.py

```
import requests
from requests.api import head

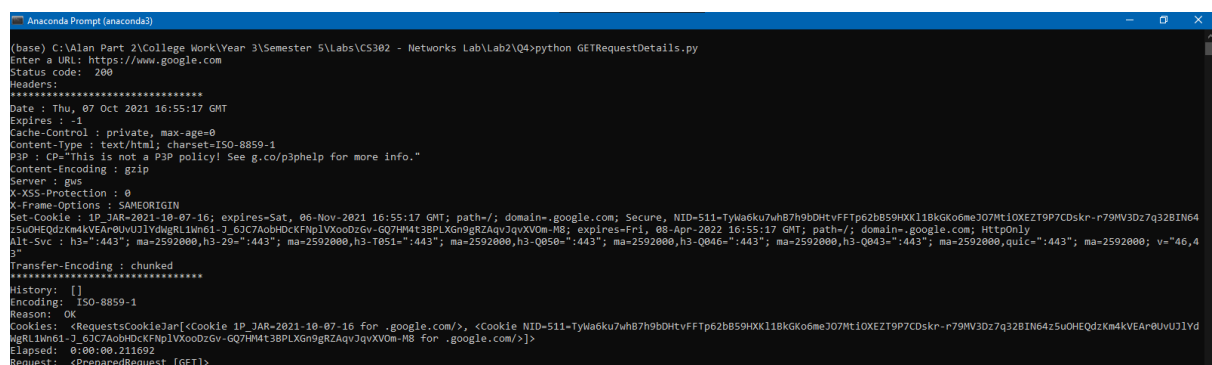
url = input('Enter a URL: ')
response = requests.get(url)

print("Status code: ", response.status_code)    #Returns a number that indicates the status (200 is OK, 404 is Not Found)

print("Headers: ")    #Returns a dictionary of response headers
print("*****")
headers = response.headers
for key, value in headers.items():
    print(f"{key} : {value}")
print("*****")

print("History: ", response.history)    #Returns a list of response objects holding the history of request (url)
print("Encoding: ", response.encoding)    #Returns the encoding used to decode response.text
print("Reason: ", response.reason)    #Returns a text corresponding to the status code
print("Cookies: ", response.cookies)    #Returns a CookieJar object with the cookies sent back from the server
print("Elapsed: ", response.elapsed)    #Returns a timedelta object with the time elapsed from sending the request to the arrival of the response
print("Request: ", response.request)    #Returns the request object that requested this response
```

Output:



```
Anaconda Prompt (anaconda3)
(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab2\Q4:python GETRequestDetails.py
Enter a URL: https://www.google.com
Status code: 200
Headers:
*****
Date: Thu, 07 Oct 2021 16:55:17 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="CP-This is not a P3P policy! See g.co/p3phelp for more info."
Content-Encoding: gzip
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2021-10-07-16; expires=Sat, 06-Nov-2021 16:55:17 GMT; path=/; domain=.google.com; Secure, NID=511-Tywa6ku7whB7h9bDHTvFFTP62bB59HXK11BkGKo6meJ07Mt10XEZT9P7CDskr~r79MV3Dz7q32BIN64z5uOHEQdzKm4KVEAr0UvUJLYdMgRL1m61-J_6JCTAobHdCKFnp1VXooDzGv-G07HMT3BPLXGn9gRZAqv3qvXVom-M8; expires=Fri, 08-Apr-2022 16:55:17 GMT; path=/; domain=.google.com; HttpOnly
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-T051=":443"; ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443"; ma=2592000; v="46,45"
Transfer-Encoding: chunked
*****
History: []
Encoding: ISO-8859-1
Reason: OK
Cookies: <RequestsCookieJar[<Cookie 1P_JAR=2021-10-07-16 for .google.com/>, <Cookie NID=511-Tywa6ku7whB7h9bDHTvFFTP62bB59HXK11BkGKo6meJ07Mt10XEZT9P7CDskr~r79MV3Dz7q32BIN64z5uOHEQdzKm4KVEAr0UvUJLYdMgRL1m61-J_6JCTAobHdCKFnp1VXooDzGv-G07HMT3BPLXGn9gRZAqv3qvXVom-M8 for .google.com/>]>
Elapsed: 0:00:00.211692
Request: <PreparedRequest [GET]>
```


Question 5:

Analyzing HTTP messages:

1. Webpage URL: <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html>

After requesting for the webpage on a private window (where no cookies are stored), this is the output I get:

No.	Time	Source	Destination	Protocol	Length	Info
517	76.969121	192.168.1.14	128.119.245.12	HTTP	542	GET /wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1
521	77.224588	128.119.245.12	192.168.1.14	HTTP	540	HTTP/1.1 200 OK (text/html)
523	77.296357	192.168.1.14	128.119.245.12	HTTP	488	GET /favicon.ico HTTP/1.1
525	77.549735	128.119.245.12	192.168.1.14	HTTP	538	HTTP/1.1 404 Not Found (text/html)

Observations:

- 1st GET request is sent to the server with IP 128.119.245.12 from our local private IP 192.168.1.14. We can confirm our local IP with the **ipconfig** command. Screenshot:

```
Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix  . : domain.name
Link-local IPv6 Address . . . . . : fe80::7839:34ec:b674:f1f1%19
IPv4 Address. . . . . : 192.168.1.14
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : fe80::bac1:acff:fe8f:b5bd%19
                             192.168.1.1
```

The following was the HTTP header:

```
GET /wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1\r\n
> [Expert Info (Chat/Sequence): GET /wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1\r\n]
Request Method: GET
Request URI: /wireshark-labs/HTTP-wireshark-file1.html
Request Version: HTTP/1.1
Host: gaia.cs.umass.edu\r\n
Connection: keep-alive\r\n
DNT: 1\r\n
```

- The response to the request was **200 OK**, which means “request has succeeded.”
- The **connection** header-line is set to **keep alive** to request a persistent connection.
- Upon reloading the page, our browser sent a GET request. But the server returned an **HTTP 304 Not Modified** response because the content on the server has not been modified, and the website contents are cached on our local machine.

The following header-line implements this:

If-Modified-Since: Thu, 07 Oct 2021 05:59:01 GMT\r\n

2. Webpage URL: <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html>

Screenshot of the webpage:



This little HTML file is being served by gaia.cs.umass.edu. It contains two embedded images. The image above, also served from the gaia.cs.umass.edu web site, is the logo of our publisher, Pearson. The image of our 8th edition book cover below is stored at, and served from, a WWW server kurose.cslash.net in France:



And while we have your attention, you might want to take time to check out the available open resources for this book at http://gaia.cs.umass.edu/kurose_ross.

The HTTP messages:

No.	Time	Source	Destination	Protocol	Length	Info
76	6.550233	192.168.1.14	128.119.245.12	HTTP	542	GET /wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
80	6.830469	128.119.245.12	192.168.1.14	HTTP	1355	HTTP/1.1 200 OK (text/html)
81	6.849304	192.168.1.14	128.119.245.12	HTTP	488	GET /pearson.png HTTP/1.1
86	7.127286	128.119.245.12	192.168.1.14	HTTP	761	HTTP/1.1 200 OK (PNG)
112	7.783211	192.168.1.14	178.79.137.164	HTTP	455	GET /8E_cover_small.jpg HTTP/1.1
116	7.952322	178.79.137.164	192.168.1.14	HTTP	225	HTTP/1.1 301 Moved Permanently

Exchanges:

- The first exchange was for the HTML file of the webpage. There are 2 images embedded in the HTML, which were delivered subsequently
- The 2nd exchange was for a .jpg file from the same host.
- The 3rd exchange was for a .png image file delivered from another host. So server returns a **301 Moved Permanently** message, which indicates that the requested content is present in another location:

Location: https://kurose.cslash.net/8E_cover_small.jpg\r\n

Analyzing HTTPS messages:

HTTPS is a more secure form of HTTP. TLS is used to encrypt requests and responses from a host. TLS uses a method called public-key encryption, where the host and the servers exchange public keys. Each host uses the public key to encrypt the outgoing data. Each host uses their private key to decrypt traffic from the other host. TLS is also used to verify the identity of the server. Once the identity is confirmed, the keys are generated and exchanged. This process is called the TLS handshake.

For this reason, Wireshark needs the session keys generated by the TLS handshake to decrypt the HTTP requests and responses made using the HTTP protocol. This is information can be logged by the web browser on setting an environment variable **SSLKEYLOGFILE**. We should also set the path of this file in Wireshark preferences.

Once the TLS handshake is decrypted, additional HTTP and TCP packets are visible, containing the decrypted exchange between the client and the server.

We can now filter the TLS messages with the keyword “gaia” since it is part of the domain name of the server:

No.	Time	Source	Destination	Protocol	Length	Info
43	9.992831	192.168.1.14	128.119.245.12	TLSv1.2		571 Client Hello
50	10.277637	192.168.1.14	128.119.245.12	TLSv1.2		571 Client Hello
52	10.280977	128.119.245.12	192.168.1.14	TLSv1.2		1506 Server Hello
56	10.289286	128.119.245.12	192.168.1.14	TLSv1.2		1345 Certificate, Server Key Exchange, Server Hello Done
60	10.606668	128.119.245.12	192.168.1.14	TLSv1.2		1506 Server Hello
64	10.616992	128.119.245.12	192.168.1.14	TLSv1.2		1345 Certificate, Server Key Exchange, Server Hello Done

On expanding the details section of the first Client Hello TLS message, we see an extension named “server-name (len=22). This should contain the server’s name:

```
▼ Server Name Indication extension
  Server Name list length: 20
  Server Name Type: host_name (0)
  Server Name length: 17
  Server Name: gaia.cs.umass.edu
```

This is Client Hello frame is a part of the TLS handshake. It doesn’t contain the HTTP layer above it:

```
> Frame 43: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits) on interface \Device\NPF_{BBC18A7A-F119-40E7-B5CE-42FE27E496C9}, id 0
> Ethernet II, Src: IntelCor_58:67:40 (44:af:28:58:67:40), Dst: b8:c1:ac:8f:b5:bd (b8:c1:ac:8f:b5:bd)
> Internet Protocol Version 4, Src: 192.168.1.14, Dst: 128.119.245.12
> Transmission Control Protocol, Src Port: 49693, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
> Transport Layer Security
```

To access the HTTP queries, we can filter out the frames exchanged which involves the public IP of the server (128.119.245.12), using `ip.addr == 128.119.245.12`. Now we can see all the other frames:

34	9.732056	192.168.1.14	128.119.245.12	TCP	66 49693 → 443 [SYN] Seq=0 Win=64240 Len=0
39	9.981910	192.168.1.14	128.119.245.12	TCP	66 56034 → 443 [SYN] Seq=0 Win=64240 Len=0
40	9.992422	128.119.245.12	192.168.1.14	TCP	66 443 → 49693 [SYN, ACK] Seq=0 Ack=1 Win=2
41	9.992534	192.168.1.14	128.119.245.12	TCP	54 49693 → 443 [ACK] Seq=1 Ack=1 Win=132896
43	9.992831	192.168.1.14	128.119.245.12	TLSv1.2	571 Client Hello
48	10.277295	128.119.245.12	192.168.1.14	TCP	66 443 → 56034 [SYN, ACK] Seq=0 Ack=1 Win=2
49	10.277413	192.168.1.14	128.119.245.12	TCP	54 56034 → 443 [ACK] Seq=1 Ack=1 Win=132896
50	10.277637	192.168.1.14	128.119.245.12	TLSv1.2	571 Client Hello
51	10.279651	128.119.245.12	192.168.1.14	TCP	54 443 → 49693 [ACK] Seq=1 Ack=518 Win=3833
52	10.280977	128.119.245.12	192.168.1.14	TLSv1.2	1506 Server Hello
53	10.280977	128.119.245.12	192.168.1.14	TCP	1506 443 → 49693 [ACK] Seq=1453 Ack=518 Win=3
54	10.280979	128.119.245.12	192.168.1.14	TCP	1246 443 → 49693 [PSH, ACK] Seq=2905 Ack=518
55	10.281028	192.168.1.14	128.119.245.12	TCP	54 49693 → 443 [ACK] Seq=518 Ack=4097 Win=1
56	10.289286	128.119.245.12	192.168.1.14	TLSv1.2	1345 Certificate, Server Key Exchange, Server
57	10.289315	192.168.1.14	128.119.245.12	TCP	54 49693 → 443 [ACK] Seq=518 Ack=5388 Win=1
58	10.300018	192.168.1.14	128.119.245.12	TLSv1.2	180 Client Key Exchange, Change Cipher Spec,
59	10.606668	128.119.245.12	192.168.1.14	TCP	54 443 → 56034 [ACK] Seq=1 Ack=518 Win=3833
60	10.606668	128.119.245.12	192.168.1.14	TLSv1.2	1506 Server Hello
61	10.609234	128.119.245.12	192.168.1.14	TCP	1506 443 → 56034 [ACK] Seq=1453 Ack=518 Win=3
62	10.609237	128.119.245.12	192.168.1.14	TCP	1246 443 → 56034 [PSH, ACK] Seq=2905 Ack=518
63	10.609277	192.168.1.14	128.119.245.12	TCP	54 56034 → 443 [ACK] Seq=518 Ack=4097 Win=1
64	10.616992	128.119.245.12	192.168.1.14	TLSv1.2	1345 Certificate, Server Key Exchange, Server
65	10.617081	192.168.1.14	128.119.245.12	TCP	54 56034 → 443 [ACK] Seq=518 Ack=5388 Win=1
66	10.617680	192.168.1.14	128.119.245.12	TLSv1.2	180 Client Key Exchange, Change Cipher Spec,
67	10.630335	128.119.245.12	192.168.1.14	TLSv1.2	328 New Session Ticket, Change Cipher Spec,
68	10.630758	192.168.1.14	128.119.245.12	HTTP	794 GET /wireshark-labs/HTTP-wireshark-file1
69	10.955308	128.119.245.12	192.168.1.14	TLSv1.2	328 New Session Ticket, Change Cipher Spec,
70	10.955309	128.119.245.12	192.168.1.14	HTTP	598 HTTP/1.1 200 OK (text/html)
71	11.001154	192.168.1.14	128.119.245.12	HTTP	724 GET /favicon.ico HTTP/1.1
72	11.006416	192.168.1.14	128.119.245.12	TCP	54 56034 → 443 [ACK] Seq=644 Ack=5662 Win=1
76	11.253729	128.119.245.12	192.168.1.14	HTTP	596 HTTP/1.1 404 Not Found (text/html)
77	11.296075	192.168.1.14	128.119.245.12	TCP	54 49693 → 443 [ACK] Seq=2054 Ack=6748 Win=
106	16.258567	128.119.245.12	192.168.1.14	TLSv1.2	85 Alert (Level: Warning, Description: Clos

These are our main HTTP GET requests and responses:

68	10.630758	192.168.1.14	128.119.245.12	HTTP	794 GET /wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1
69	10.955308	128.119.245.12	192.168.1.14	TLSv1.2	328 New Session Ticket, Change Cipher Spec, Finished
70	10.955309	128.119.245.12	192.168.1.14	HTTP	598 HTTP/1.1 200 OK (text/html)

These messages contain an extra layer called HTTP, which are now decrypted:

```
> Frame 68: 794 bytes on wire (6352 bits), 794 bytes captured (6352 bits) on interface \Device\NPF_{BBC18A7A-F119-40E7-B5CE-42FE27E496C9}, id 0
> Ethernet II, Src: IntelCor_58:67:40 (44:af:28:58:67:40), Dst: b8:c1:ac:8f:b5:bd (b8:c1:ac:8f:b5:bd)
> Internet Protocol Version 4, Src: 192.168.1.14, Dst: 128.119.245.12
> Transmission Control Protocol, Src Port: 49693, Dst Port: 443, Seq: 644, Ack: 5662, Len: 740
> Transport Layer Security
> Hypertext Transfer Protocol
```

The GET response contains the HTML data which is rendered by our browser:

✓ Line-based text data: text/html (4 lines)

```
<html>\n
Congratulations.  You've downloaded the file \n
http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html!\n
</html>\n
```

Webpage 2: <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html>

For the second webpage, the HTTP response was pretty much the same, but we got an **HTTP 200 OK** message instead of the **HTTP 304 Permanently Moved** message for the last image:

84	9.870208	192.168.1.14	128.119.245.12	HTTP	794 GET /wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
88	10.136547	128.119.245.12	192.168.1.14	HTTP	1413 HTTP/1.1 200 OK (text/html)
89	10.163712	192.168.1.14	128.119.245.12	HTTP	724 GET /pearson.png HTTP/1.1
99	10.429988	128.119.245.12	192.168.1.14	HTTP	819 HTTP/1.1 200 OK (PNG)
111	10.749959	192.168.1.14	178.79.137.164	HTTP	690 GET /8E_cover_small.jpg HTTP/1.1
576	11.796052	178.79.137.164	192.168.1.14	HTTP	949 HTTP/1.1 200 OK (JPEG JFIF image)

The difference between the HTTP protocol as experimented in the first section was that there is a TLS protocol layer beneath HTTP:

```
> Frame 89: 724 bytes on wire (5792 bits), 724 bytes captured (5792 bits) on interface \Device\NPF_{BBC18A7A-F119-40E7-B5CE-42FE27E496C9}, id 0
> Ethernet II, Src: IntelCor_58:67:40 (44:af:28:58:67:40), Dst: b8:c1:ac:8f:b5:bd (b8:c1:ac:8f:b5:bd)
> Internet Protocol Version 4, Src: 192.168.1.14, Dst: 128.119.245.12
> Transmission Control Protocol, Src Port: 59063, Dst Port: 443, Seq: 1384, Ack: 7021, Len: 670
> Transport Layer Security
> Hypertext Transfer Protocol
```
