NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

INFORMATION SECURITY LAB

NAME	Ayesha Imran
Class	CS-A
Lab	10
Course	Information Security
Date	12-December-25
Submitted To	Lec. Attiya Ashraf

LAB TASKS

Step 1: Open the http Trace File:

Browser behavior can be quite complex, using more HTTP features than the basic exchange, this trace will show us how much gets transferred.

Step 0:

1. Install Wireshark:

Update package list

sudo apt update

```
student@student-VMware-Virtual-Platform:~$ sudo apt update
[sudo] password for student:
Warning: The unit file, source configuration file or drop-ins of apt-news.servic
e changed on disk. Run 'systemctl daemon-reload' to reload units.
Warning: The unit file, source configuration file or drop-ins of esm-cache.servi
ce changed on disk. Run 'systemctl daemon-reload' to reload units.
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1,36
3 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1,664
kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [222
kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21
.5 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/main amd64 c-n-f Metadata
 [9,448 B]
Get:10 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Package
s [2,234 kB]
Get:11 http://archive.ubuntu.com/ubuntu noble-updates/main Translation-en [308 k
B]
```

Install Wireshark

sudo apt install wireshark

```
student@student-VMware-Virtual-Platform:~$ sudo apt install wireshark
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libgl1-amber-dri libglapi-mesa libllvm17t64
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  i965-va-driver intel-media-va-driver libaacs0 libavcodec60 libavformat60
  libavutil58 libb2-1 libbcg729-0 libbdplus0 libbluray2 libchromaprint1
  libcjson1 libcodec2-1.2 libdav1d7 libdouble-conversion3 libdrm-amdgpu1
  libdrm-common libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libdrm2
  libegl-mesa0 libgbm1 libgl1-mesa-dri libglx-mesa0 libgme0 libgsm1 libhwy1t64
  libigdgmm12 libjxl0.7 libllvm20 liblua5.2-0 libmbedcrypto7t64 libmd4c0
  libminizip1t64 libnghttp3-3 libnorm1t64 libopencore-amrnb0 libopenmpt0t64
  libpcre2-16-0 libpcre2-32-0 libpcre2-8-0 libpgm-5.3-0t64 libqt6core5compat6
  libqt6core6t64 libqt6dbus6t64 libqt6gui6t64 libqt6multimedia6
  libqt6network6t64 libqt6opengl6t64 libqt6printsupport6t64 libqt6qml6
  libqt6qmlmodels6 libqt6quick6 libqt6svg6 libqt6waylandclient6
  libqt6waylandcompositor6 libqt6waylandeglclienthwintegration6
  libqt6waylandeglcompositorhwintegration6 libqt6widgets6t64
```

You will need to log out and log back in for the change to take effect.

2. Download the Trace File:

wget https://kevincurran.org/com320/labs/wireshark/trace-http.pcap

```
student@student-VMware-Virtual-Platform:~$ wget https://kevincurran.org/com320/l
abs/wireshark/trace-http.pcap
--2025-12-11 14:03:14--  https://kevincurran.org/com320/labs/wireshark/trace-htt
p.pcap
Resolving kevincurran.org (kevincurran.org)... 188.114.97.6, 188.114.96.6, 2a06:
98c1:3120::6, ...
Connecting to kevincurran.org (kevincurran.org)|188.114.97.6|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1245922 (1.2M) [application/vnd.tcpdump.pcap]
Saving to: 'trace-http.pcap'

trace-http.pcap     100%[===================>]   1.19M   558KB/s    in 2.2s

2025-12-11 14:03:17 (558 KB/s) - 'trace-http.pcap' saved [1245922/1245922]

student@student-VMware-Virtual-Platform:~$
```

Step 1: Open the http Trace

Browser behavior can be quite complex, using more HTTP features than the basic exchange, this trace will show us how much gets transferred.

1. Open the trace file from here: https://kevincurran.org/com320/labs/wireshark/trace-http.pcap or https://learning.ulster.ac.uk/bbcswebdav/pid-8778207-dt-content-rid-47612278_1/xid47612278_1

You should then see a Wireshark screen like below.

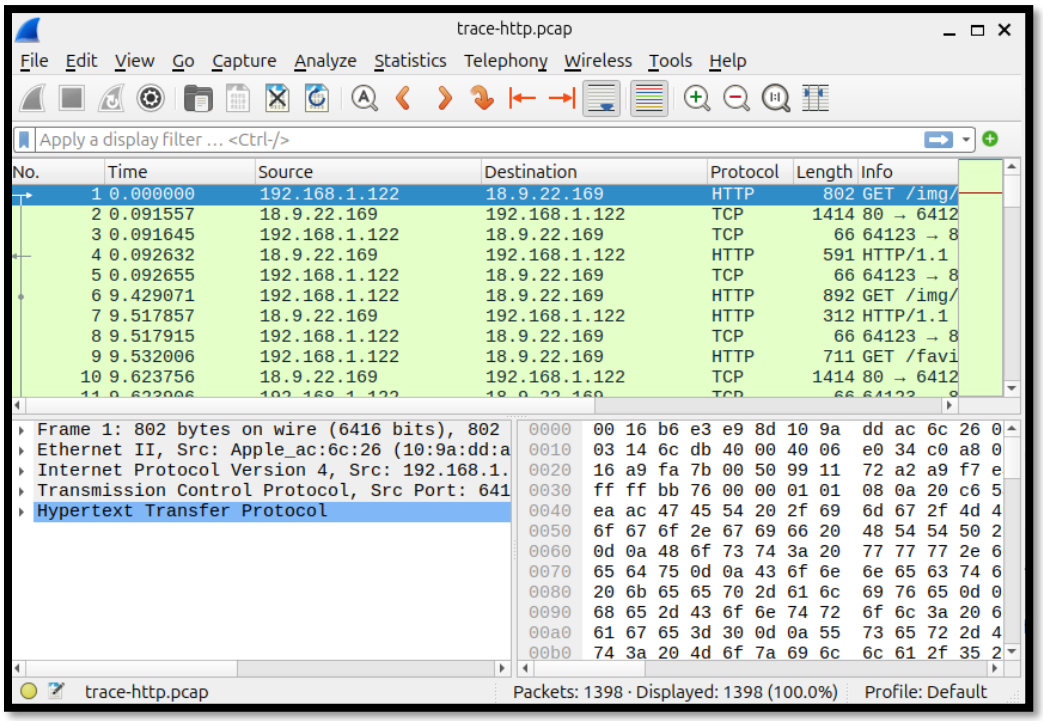


Figure 1: The Wireshark screen after opening the http trace file

Step 2: Inspect the Trace

2. To focus on HTTP traffic, *enter and apply a filter expression of* "http".

This filter will show HTTP requests and responses, but not the individual packets that are involved. You should see a screen like below.

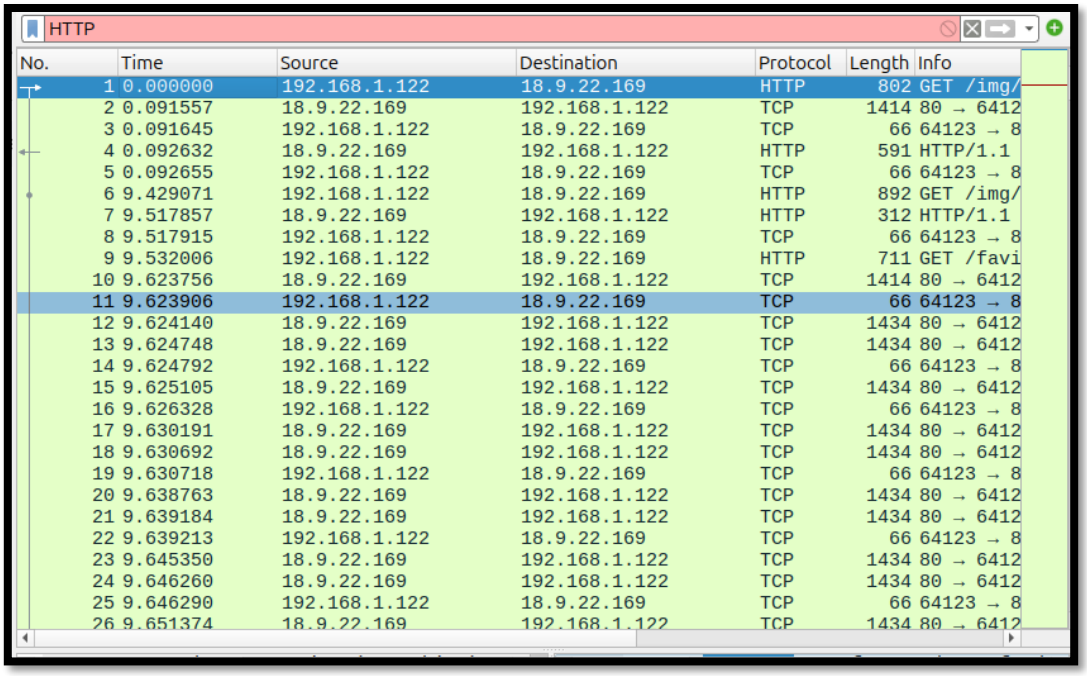


Figure 2: Trace of HTTP traffic

Recall that an HTTP response carrying content will normally be spread across multiple packets. When the last packet in the response arrives, Wireshark assembles the complete response and tags the packet with protocol HTTP. The earlier packets are simply TCP segments carrying data; the last packet tagged HTTP includes a list of all the earlier packets used to make the response. A similar process occurs for the request, but in this case it is common for a request to fit in a single packet. With the filter expression of "http" we will hide the intermediate TCP packets and see only the HTTP requests and responses.

3. Select the first GET in the trace and expand its HTTP block.

(This will be 802 GET /img/MIT_logo.gif HTTP/1.1 as shown below).

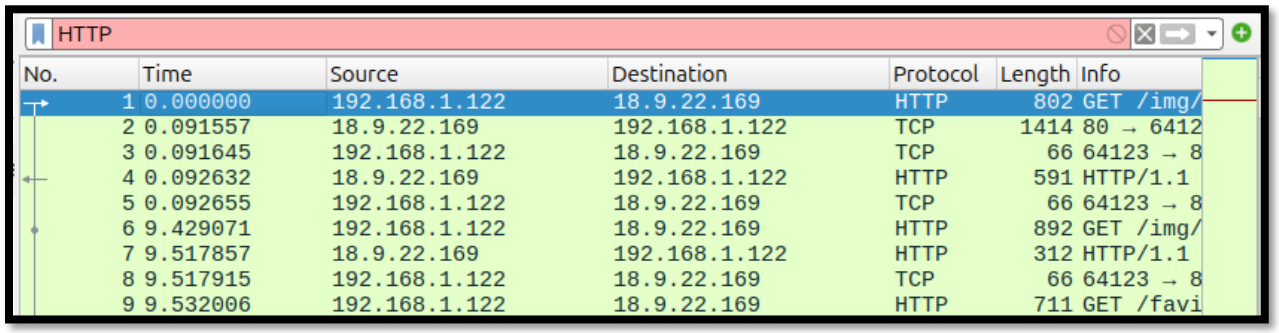


Figure 3: The first GET packet in trace: get /img/MIT_logo.gif HTTP/1.1

This will let us inspect the details of an HTTP request.

4. Inspect the HTTP header by expanding the down arrow beside "Hypertext Transfer Protocol" In the middle pane (as shown below).

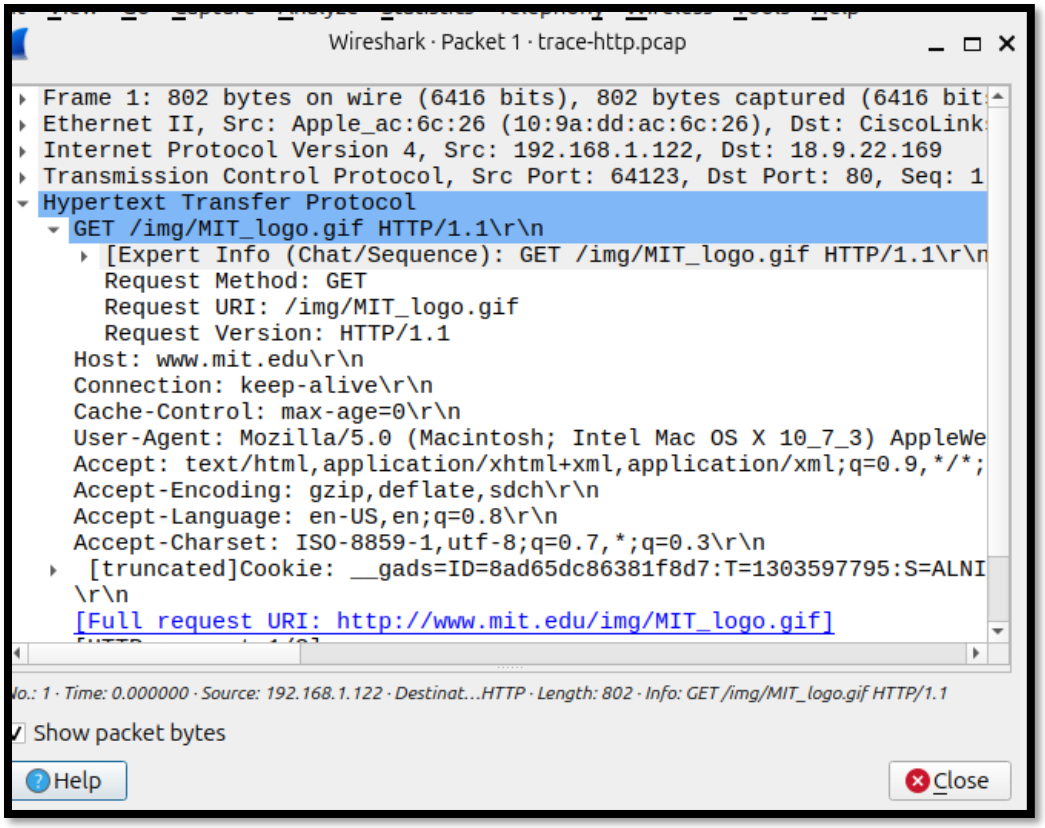


Figure 4: The Hypertext transfer protocol for the first http request

Observe that the HTTP header follows the TCP and IP headers, as HTTP is an application protocol that is transported using TCP/IP. To view it, select the packet, find the HTTP block in the middle panel, and ex pand it (by using the "v" expander or icon). This block is expanded in figure above.

Explore the headers that are sent along with the request. First, you will see the GET method at the start of the request, including details such as the path. Then you will see a series of headers in the form of tagged parameters. There may be many headers, and the choice of headers and their values vary from browser to browser. See if you have any of these common headers:

· Host. A mandatory header, it identifies the name (and port) of the server.
· User-Agent. The kind of browser and its capabilities.
· Accept, Accept-Encoding, Accept-Charset, Accept-Language. Descriptions of the formats that will be accepted in the response, e.g., text/html, including its encoding, e.g., gzip, and language. · Cookie. The name and value of cookies the browser holds for the website.
· Cache-Control. Information about how the response can be cached.

The request information is sent in a simple text and line-based format. If you look in the bottom panel you can read much of the request directly from the packet itself.

5. *Select the response that corresponds to the first GET in the trace and expand its HTTP block* (see below).

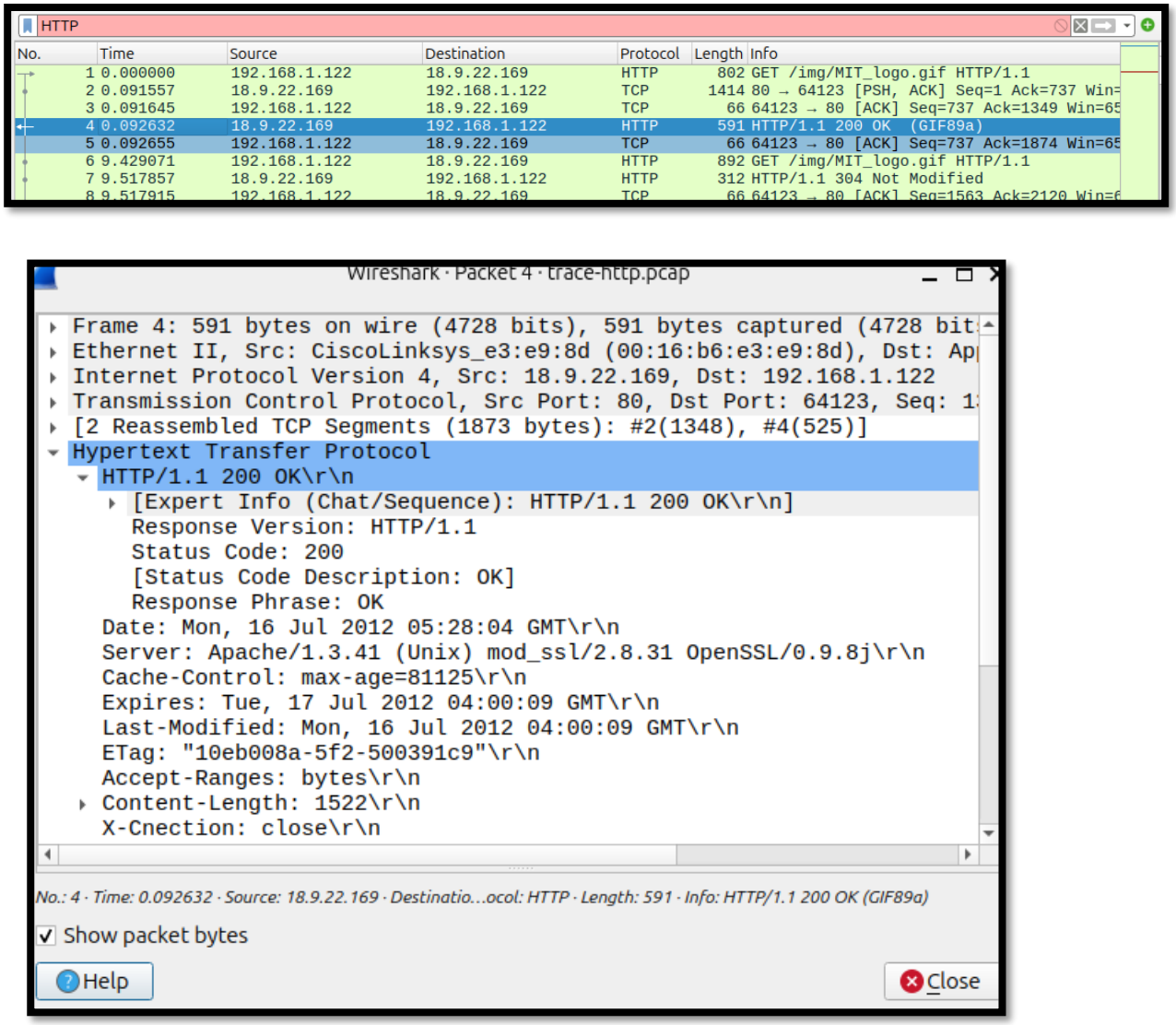


Figure 5: The response that corresponds to the first GET in the trace

The Info for this packet will indicate “200 OK” in the case of a normal, successful transfer. You will see that the response is similar to the request, with a series of headers that follow the “200 OK” status code. However, different headers will be used, and the headers will be followed by the requested content.

Examine the common headers such as:

· Server. The kind of server and its capabilities.

· Date, Last-Modified. The time of the response and the time the content last changed. · Cache-Control, Expires, Etag. Information about how the response can be cached.

Step 3: Content Caching

The second fetch in the trace is a re-fetch of the first URL. This fetch presents an opportunity for us to look at caching in action, since it is highly likely that the image or document has not changed and there fore does not need to be downloaded again. HTTP caching mechanisms should identify this opportunity. We will now see how they work.

6. *Click on the third line* in the sample trace i.e. 892 GET /img/MIT_logo.gif HTTP 1.1

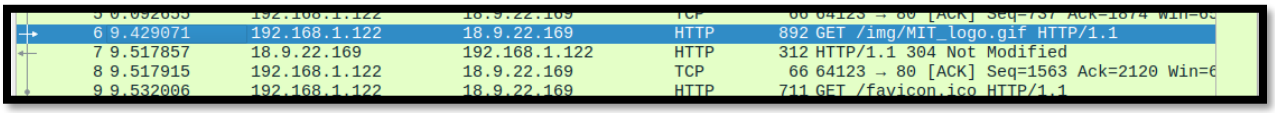


Figure 6: The second fetch - 892 GET /img/MIT_logo.gif HTTP 1.1

7. *Expand its HTTP block (see below)*

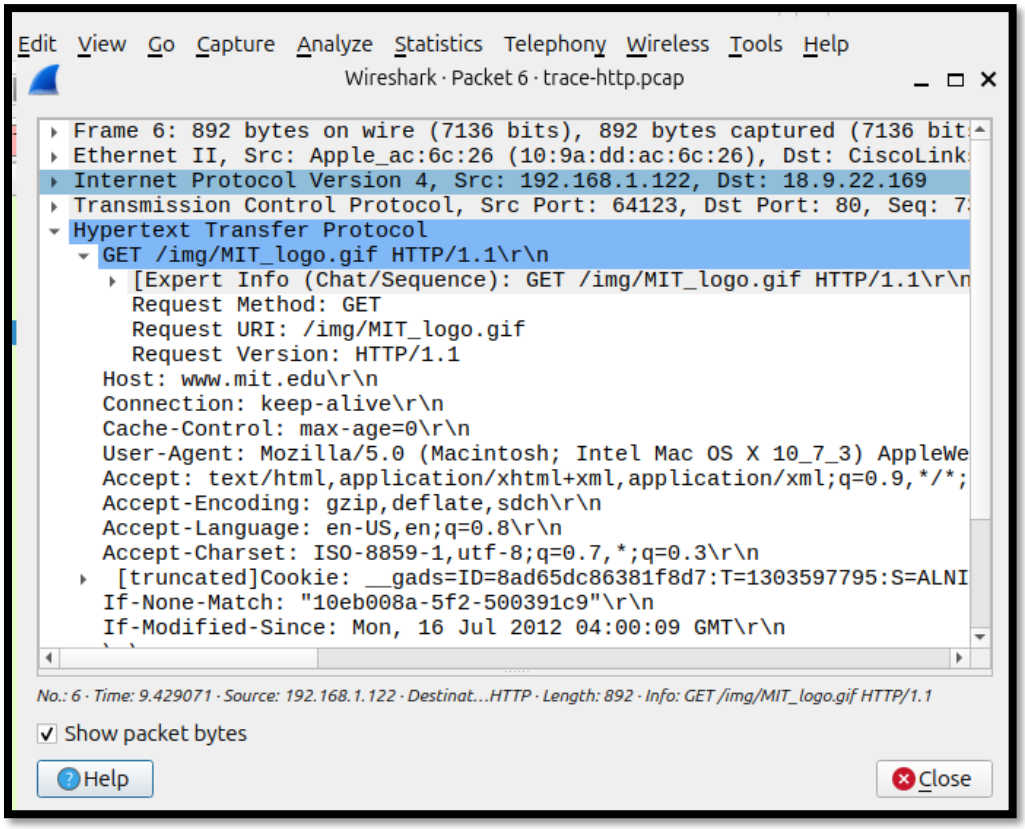


Figure 7: The HTTP block of 892 GET /img/MIT_logo.gif HTTP 1.1

Now find the header that will let the server work out whether it needs to send fresh content. The server will need to send fresh content only if the content has changed since the browser last downloaded it. To work this out, the browser includes a timestamp taken from the previous download for the content that it
has cached. This header was not present on the first GET since we cleared the browser cache so the browser had no previous download of the content that it could use. In most other respects, this request will be the same as the first-time request.

This header is called “If-Modified-Since”, i.e., it asks the server to send the content if it has been modi fied since a given time (see the header below).

8. *Now select the response to the re-fetch, and expand its HTTP block.*

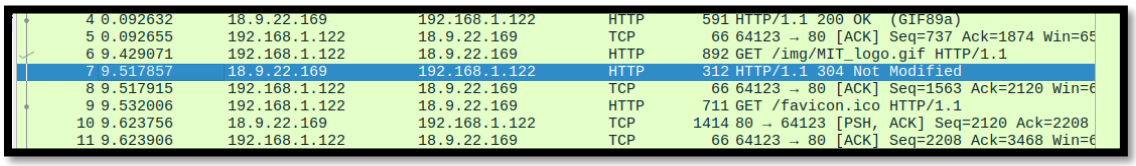


Figure 8: The response to cached image

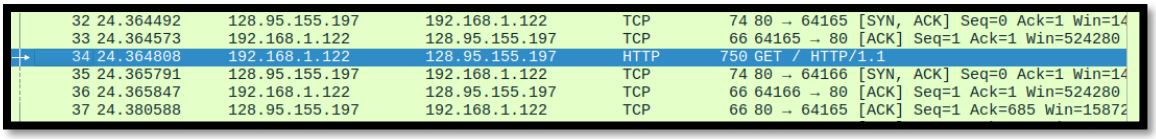
We can see that caching worked as expected, this response will not contain the content. Instead, the status code of the response is “304 Not Modified”. This tells the browser that the content is unchanged from its previous copy, and the cached content can then be displayed.

Note the timestamp value comes from the “Last-Modified” header of the most recent download of the content. It is a server timestamp for when the content last changed – it is not a timestamp according to the browser clock, and it is not a timestamp of the time of the download

Step 4: Complex Pages

Now we will examine the fourth fetch in the trace. This fetch was for a more complex web page that will likely have embedded resources. So the browser will download the initial HTML plus all of the embed ded resources needed to render the page, plus other resources that are requested during the execution of page scripts. As we will see, a single page can involve many GETs.

9. Click on no 34 which is 750 GET / HTTP/1.1 (as shown below).



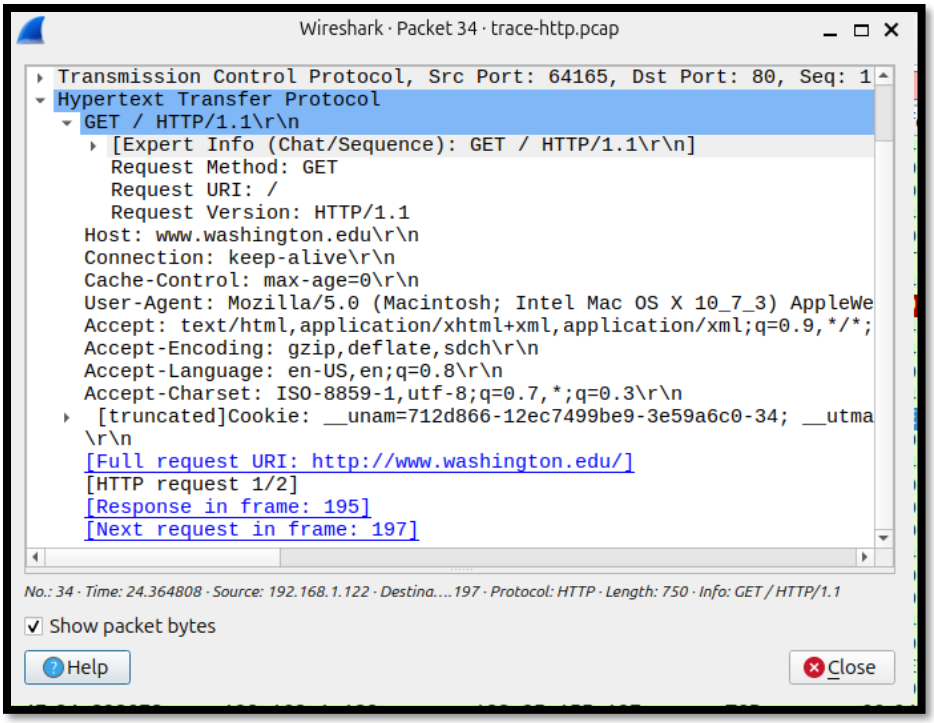


Figure 8: The large fourth fetch in http trace

10. *Summarize the GETs for this page by bringing up a HTTP Load Distribution panel.* You will find this panel under “Statistics” and “HTTP”

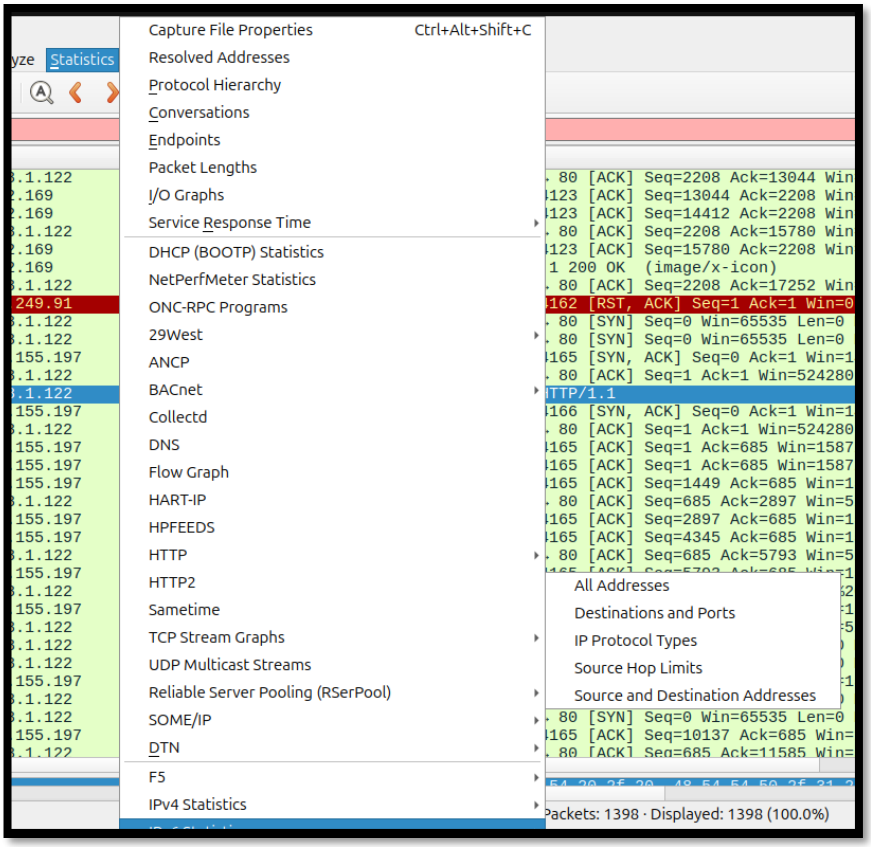


Figure 9: Statistics HTTP Load Distribution

Looking at this panel will tell you how many requests were made to which servers. This is typical, when surfing, chances are that your fetch will request content from other servers you might not have sus pected to build the page. These other servers may include third parties such as content distribution net works, ad networks, and analytics networks. Our panel is shown below – the page fetch involved 98 re quests to 5 different servers.

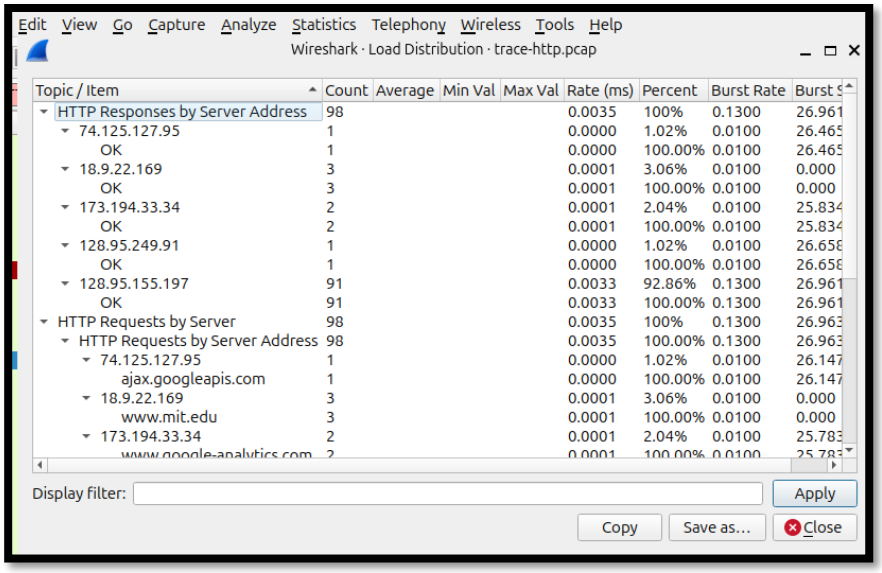


Figure 10: HTTP Load Distribution panel

11. For a different kind of summary of the GETs, *bring up a HTTP Packet Counter panel.*

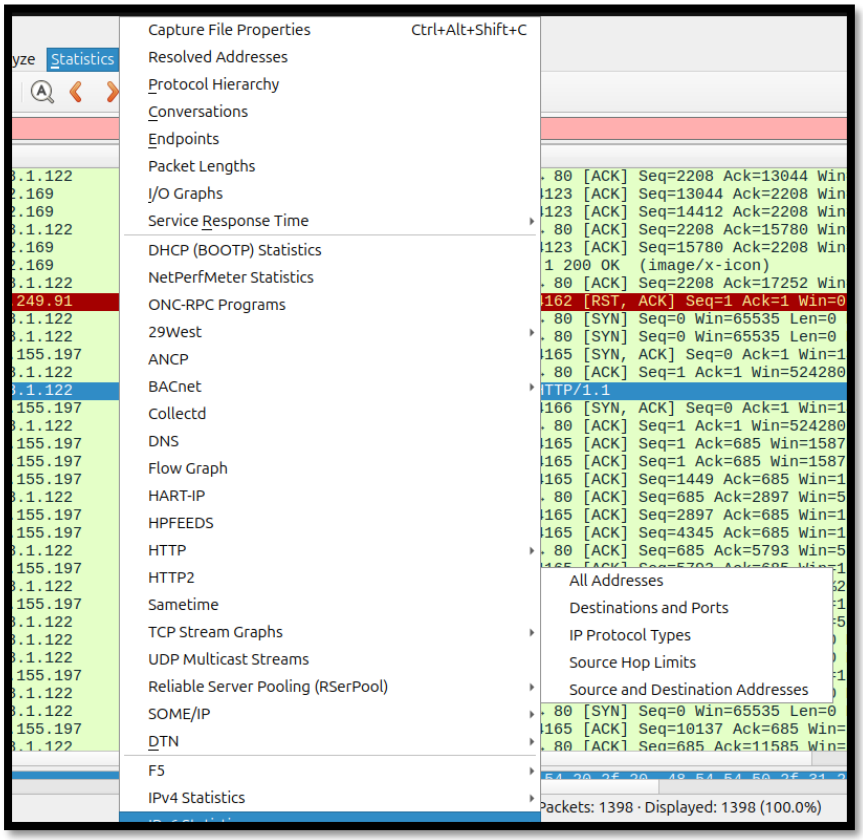


Figure 11: Statistics HTTP Packet Counter

You will also find this panel under “Statistics” and “HTTP”, and you should filter for the packets that are part of the fetch as before. This panel will tell you the kinds of request and responses. Our panel is shown in the figure below. You can see that it consists of GET requests that are matched by 200 OK re- sponses
and 304 Not Modified. However, there are a variety of other response codes that you might ob- serve in your trace, such as when the resource is already cached.

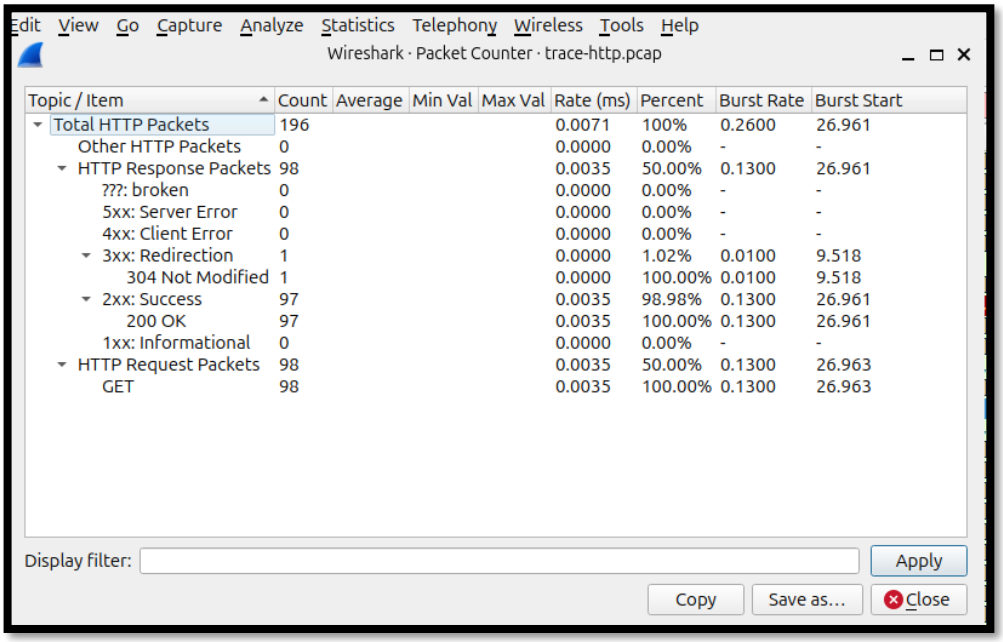


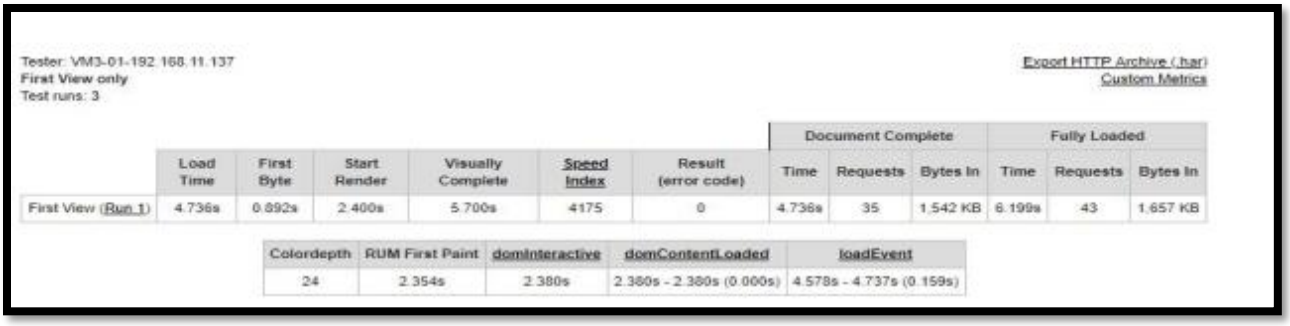
Figure 12: HTTP Packet Counter panel

You might be curious to know what content is being downloaded by all these requests. As well as seeing the URLs in the Info column, you can get a summary of the URLs in a HTTP Request panel under “Statis- tics” and “HTTP”. Each of the individual requests and responses has the same form we saw in an earlier step. Collectively, they are performed in the process of fetching a complete page with a given URL.

For a more detailed look at the overall page load process, use a site such as Google’s PageSpeed or https://www.webpagetest.org/. These sites will test a URL of your choice and generate a report of the page load activity, telling what requests were fetched at what times and giving tips for decreasing the overall page load time.

The beginning of the “waterfall” diagram for the page load corresponding to a trace on www.ul ster.ac.uk is shown in the next figure. After the initial HTML resource is fetched there are many subse quent quick fetches for embedded resources such as JavaScript scripts, CSS stylesheets, images, and more.

Figure 13: Start of waterfall graph for https://www.ulster.ac.uk/ (from pageloadtest.org)



Tester: VM3-01-192.168.11.137
First View only
Test runs: 3

Export HTTP Archive (.har)
Custom Metrics

	Load Time	First Byte	Start Render	Visually Complete	Speed Index	Result (error code)	Document Complete Time	Document Complete Requests	Document Complete Bytes In	Fully Loaded Time	Fully Loaded Requests	Fully Loaded Bytes In
First View (Run 1)	4.736s	0.892s	2.400s	5.700s	4175	0	4.736s	35	1,542 KB	6.199s	43	1,657 KB

Colordepth	RUM First Paint	domInteractive	domContentLoaded	loadEvent
24	2.354s	2.380s	2.380s - 2.380s (0.000s)	4.578s - 4.737s (0.159s)

Figure 14: Summary stats for https://www.ulster.ac.uk/ (from pageloadtest.org)

WaterFall: