Spring 2025

Cyber Security Fundamentals (INFT-3508 - 20332)

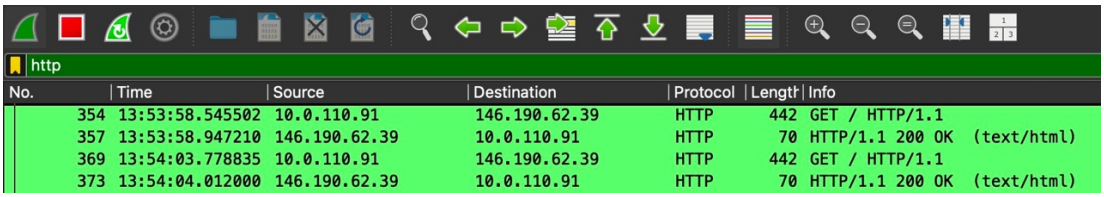
Aysel Panahova

Quiz 6

Task 1:

Firstly, I opened the WireShark and it started capturing traffic. As written in the instructions, we need to analyze HTTP requests, so I closed all my tabs and visited a single website that uses HTTP protocol (rather than HTTPS, which is encrypted and does not reveal readable data) in my browser. The website was " http://httpforever.com " This ensured that WireShark could capture HTTP requests for analysis.

I entered "HTTP" in the filter bar to display only HTTP traffic. This helped eliminate unrelated network protocols from the view, making it easier to focus on the HTTP request specifically.



Task 2:

From the screenshot, we can see that there are two pairs of requests and responses. We can calculate the time between the GET requests and the responses of these pairs.

First pair:

Frame 354 - GET request: 58.545502 sec

Frame 357 – HTTP OK: 58.947210 sec

Time difference: 0.401708 sec

Second Pair:

Frame 369 - GET request: 03.778835 sec

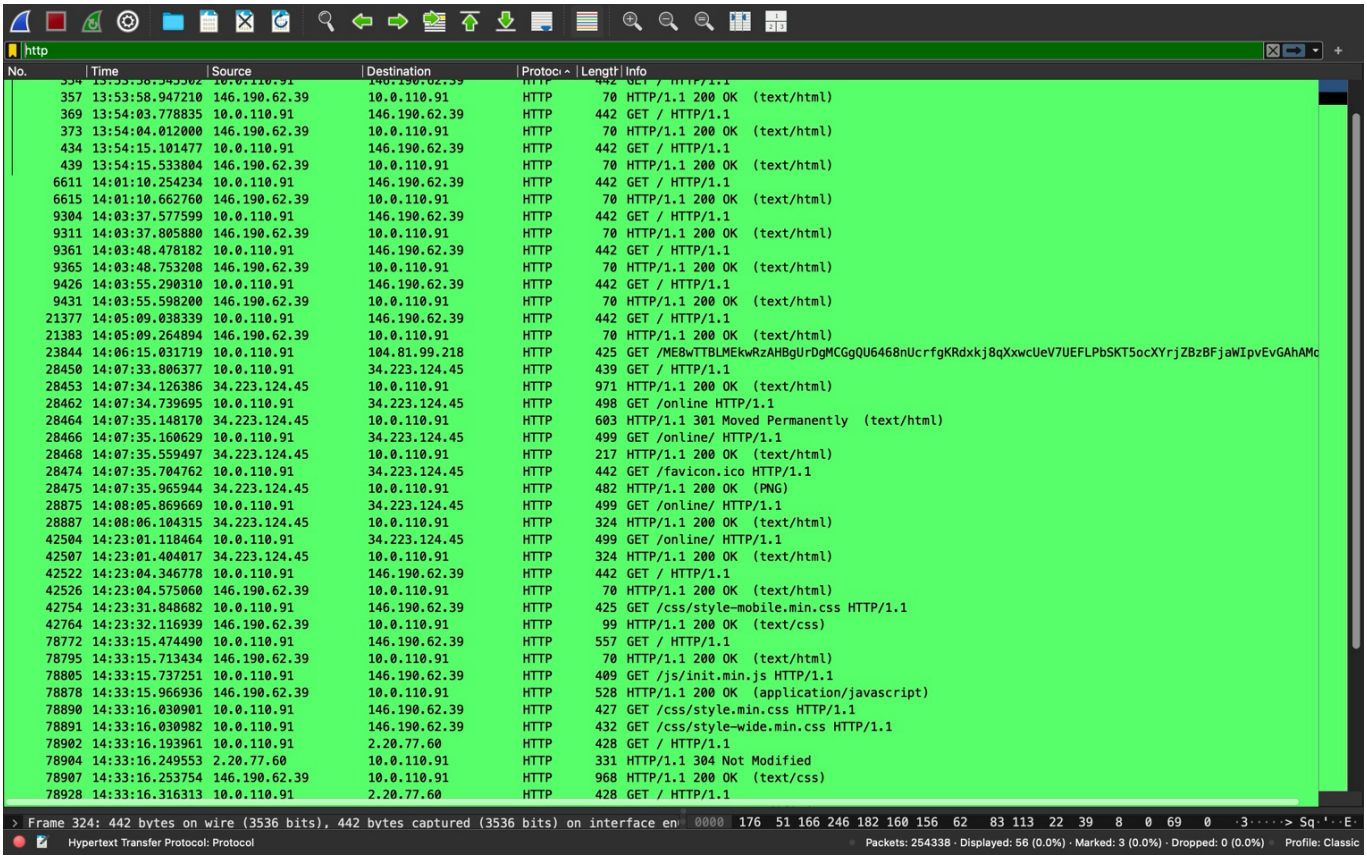
Frame 373 – HTTP OK: 04.012000 sec

Time difference: 0.233165 sec

The time between HTTP GET and response varies between approximately 0.40, 0.23 seconds.

Task: 3

Upon analyzing the packet capture, I identified 28 HTTP GET requests directed to a remote website, using the filter ip.dst == 146.190.62.39 && http. Additionally, the traffic reveals three distinct destination IP addresses: 34.223.124.45, 146.190.62.39, and 2.20.77.60, indicating that multiple external servers were contacted during the browsing session.



Task: 4

As shown in the picture, there are 4 HTTP GET requests that are images (frame 78996, 79000, 79154, 79160).

78996	14:33:16.811274	10.0.110.91	146.190.62.39	HTTP	494	GET /css/images/banner.svg HTTP/1.1
79000	14:33:16.813736	10.0.110.91	146.190.62.39	HTTP	509	GET /css/images/header-major-on-light.svg HTTP/1.1
79150	14:33:17.039826	146.190.62.39	10.0.110.91	HTTP/…	1377	HTTP/1.1 200 OK
79154	14:33:17.040967	10.0.110.91	146.190.62.39	HTTP	467	GET /favicon.ico HTTP/1.1
79160	14:33:17.043598	10.0.110.91	146.190.62.39	HTTP	508	GET /css/images/header-major-on-dark.svg HTTP/1.1
79161	14:33:17.045245	146.190.62.39	10.0.110.91	HTTP/…	1327	HTTP/1.1 200 OK
79201	14:33:17.271995	146.190.62.39	10.0.110.91	HTTP	1059	HTTP/1.1 200 OK (image/x-icon)
79203	14:33:17.271997	146.190.62.39	10.0.110.91	HTTP/…	1333	HTTP/1.1 200 OK

The images are downloaded in parallel (simultaneously). We can see this because the packet stamps show quick back-and-forth communication between my computer (10.0.110.91) and the website (146.198.62.39). For example, the server sends a packet, then my pc immediately sends another, and this keeps happening in an overlapping pattern. If the downloads were happening one at the same time (serially), we would see long pauses between requests for each image. Instead, the rapid switching between sending and receiving confirms that multiple images were being downloaded at the same time.

Task: 5

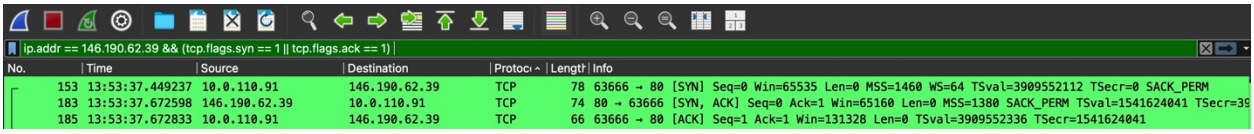
Frames 78996, 79000, 79154, and 79160 show packets sent from the local machine which is my pc (10.0.110.91) containing HTTP GET requests. Frames 79150, 79161, 79201, and 79203 show response packets from the remote server (146.190.62.39). This makes a total of eight packets exchanged, meaning four requests from the client matched by four responses from the server.

Task: 6

a) The TCP connection was established using the standard three-way handshake, which involves three packets:

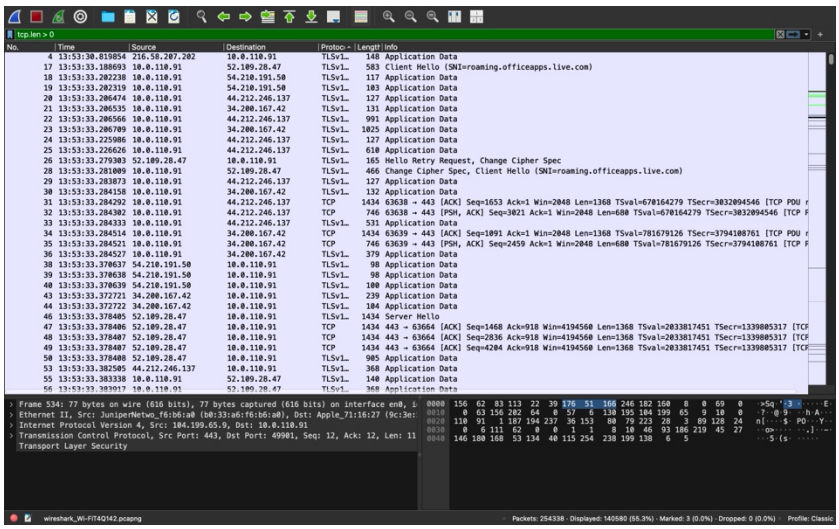
1. SYN (synchronize) → Initiated by the client (10.0.110.91) to request a connection.
2. SYN-ACK (synchronize-acknowledge) → Sent by the server (146.190.62.39) to acknowledge the request and propose its own connection parameters.
3. ACK (acknowledge) → Final confirmation from the client to establish the connection.

This exchange is clearly observed in frames 153, 183, and 185, confirming the connection setup was successful and adhered to TCP protocol standards.



b) To calculate the number of packets that were sent we can use the formula “tcp.len > 0”. This filter shows only the TCP packets that have a payload, meaning the packets that contain actual data, not just control information like ACK or handshakes.

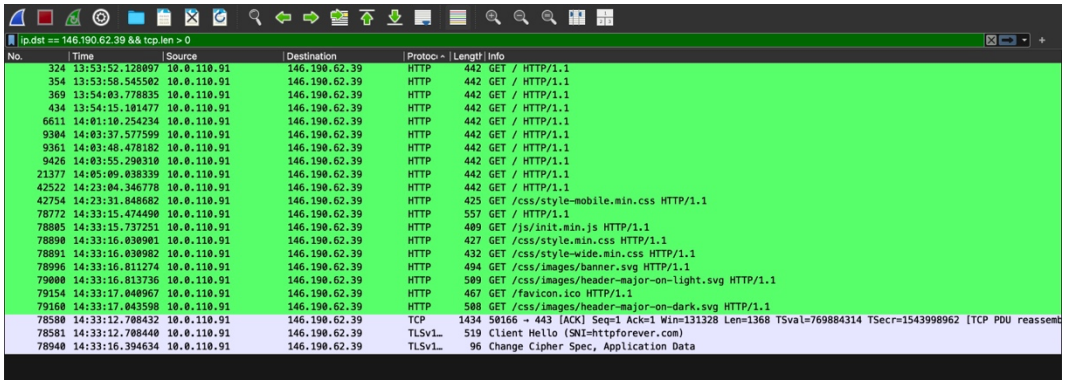
In this case, the result shows 140,580 packets.



c) To calculate the number of acknowledgment packets I used the formula “ip.dst == 146.190.62.39 && tcp.len > 0”.

- ip.dst == 146.190.62.39 means the packet is being sent to that IP address.
- tcp.len > 0 means the packet has a data payload (not just a control or acknowledgment packet).

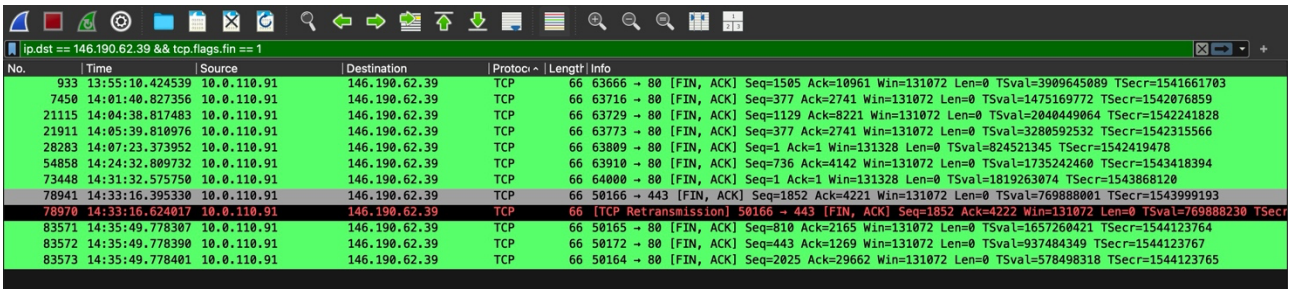
There are 22 packets.



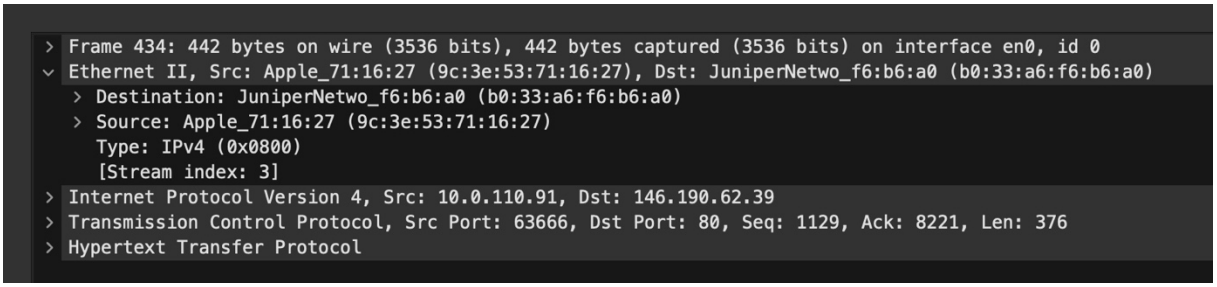
d) The TCP connection closed with these packets:

1. Client (10.0.110.91) sent FIN-ACK to signal it was done sending data.
2. Server (146.190.62.39) replied with FIN-ACK to confirm and close its side.

I used the formula “ip.dst == 146.190.62.39 && tcp.flags.fin == 1”. There appear 12 frames. This filter formula does two things: First, it looks for network packets going to the IP address 146.100.62.39 (the website). Second, it checks if those packets have the TCP FIN flag turned on. The FIN flag means the sender wants to close the connection.

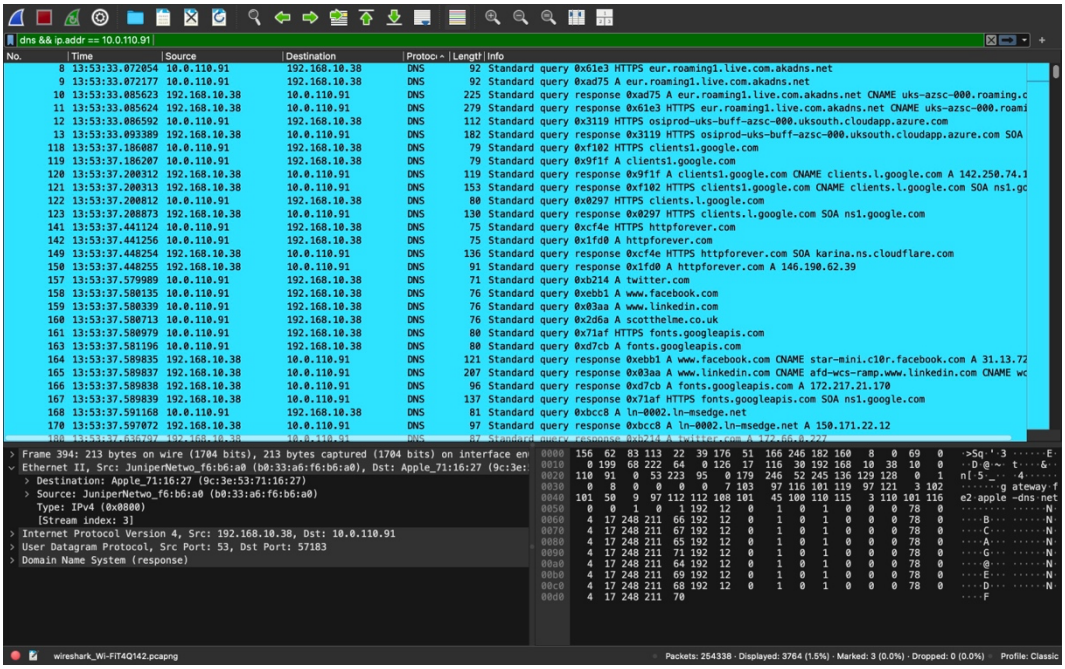


Task: 7



I clicked on any packet on the WireShark and it showed all the details about that packet. As we see from the screenshot, the source, which is my device is **Apple_71:16:27 (9c:3e:53:71:16:27)**. The destination, which is a remote device **is JuniperNetwo_f6:b6:a0 (b0:33:a6:f6:a0)**.

Task: 8



The filter formula “dns && ip.addr == <your IP address> is used in Wireshark analyze just the DNS activity of our own device. It helps us to see which domains our computer is trying to access, and what IP addresses it gets in return.

In this screenshot, the formula “dns && ip.addr == 10.0.110.91” filters the results to display only DNS traffic that involves the IP address 10.0.110.91 (my device).