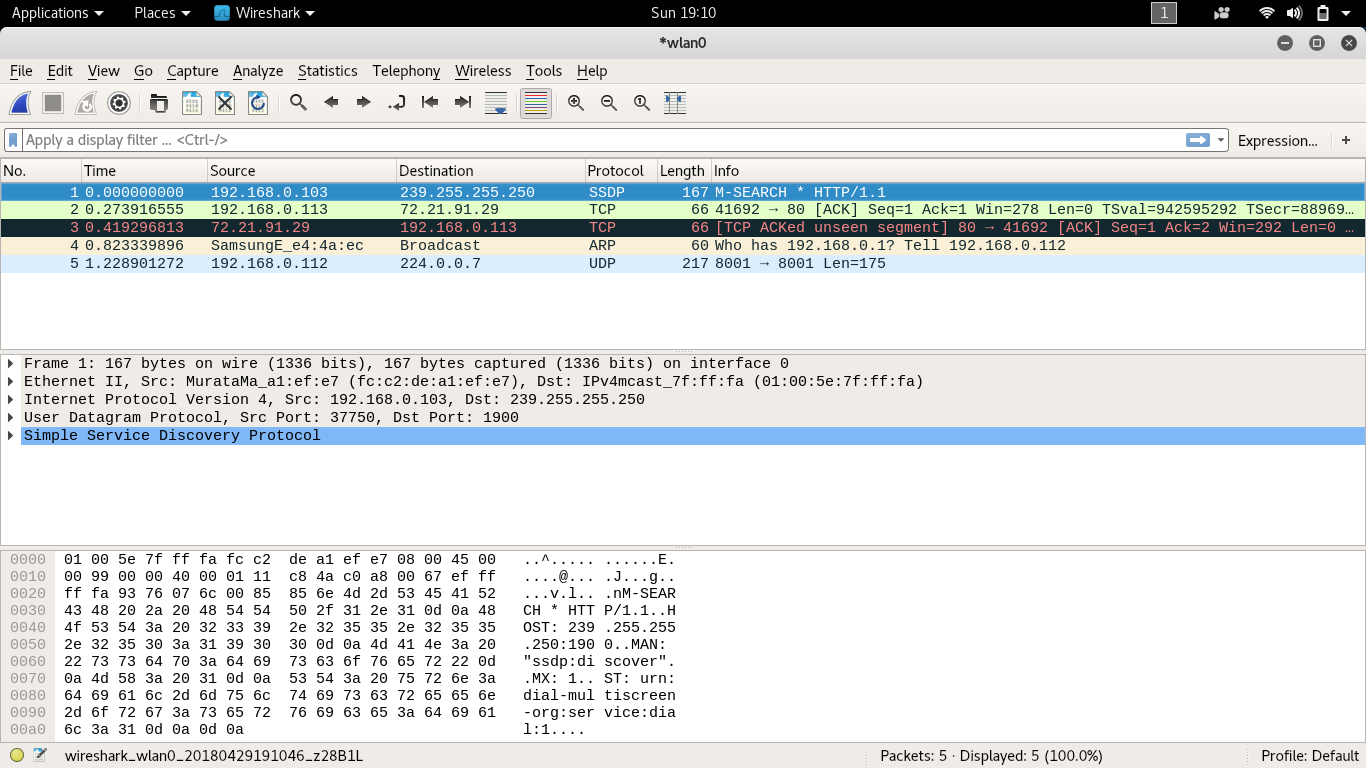
**Wireshark Packet Sniffing Project**

*Computer Details*

This report was done on a Lenovo Ideapad 320-15IAP with Kali Linux 4.14.0 at my home network. I have Spectrum as my ISP.

*Wireshark*

Wireshark is an application that captures packets that are sent on a wireless network. For example, if my computer wants to ping another computer, it will send an ICMP packet to the client and wait for a response. Wireshark can pick up this packet and display it in a table. The UI is very simple to use; first you must choose a device to start capturing packets from. I chose to use the wireless LAN port because there are some issues with trying to use Ethernet LAN. One of these issues is that if you have a switch and not a hub, you will only receive packets that belong to you or are addressed to you. In a Wi-Fi network, you can capture all packets that are being sent in the air, ones that belong to you and ones that don’t.



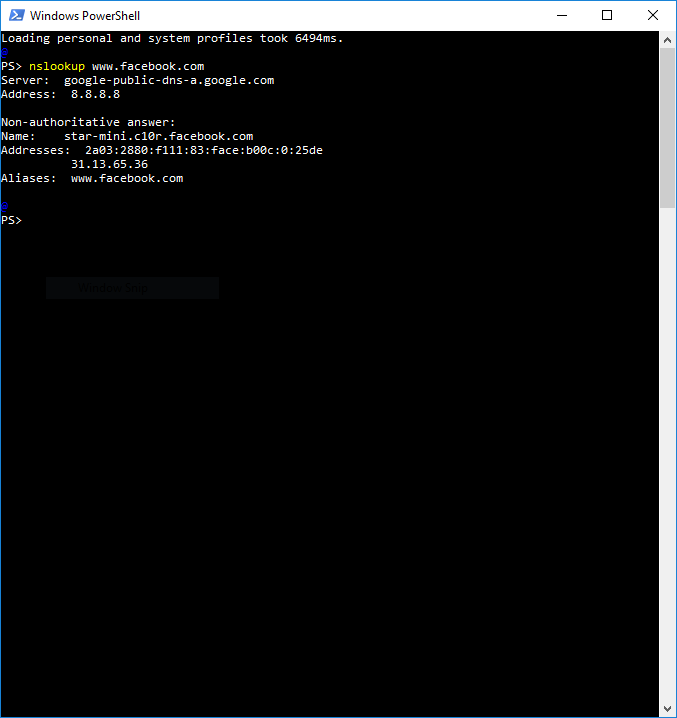
As you can see in this image, as soon as I start the capturing process I get many packets being sent over the air. One of these packets, packet number 4, belongs to my Dad’s Samsung phone.

*Facebook*

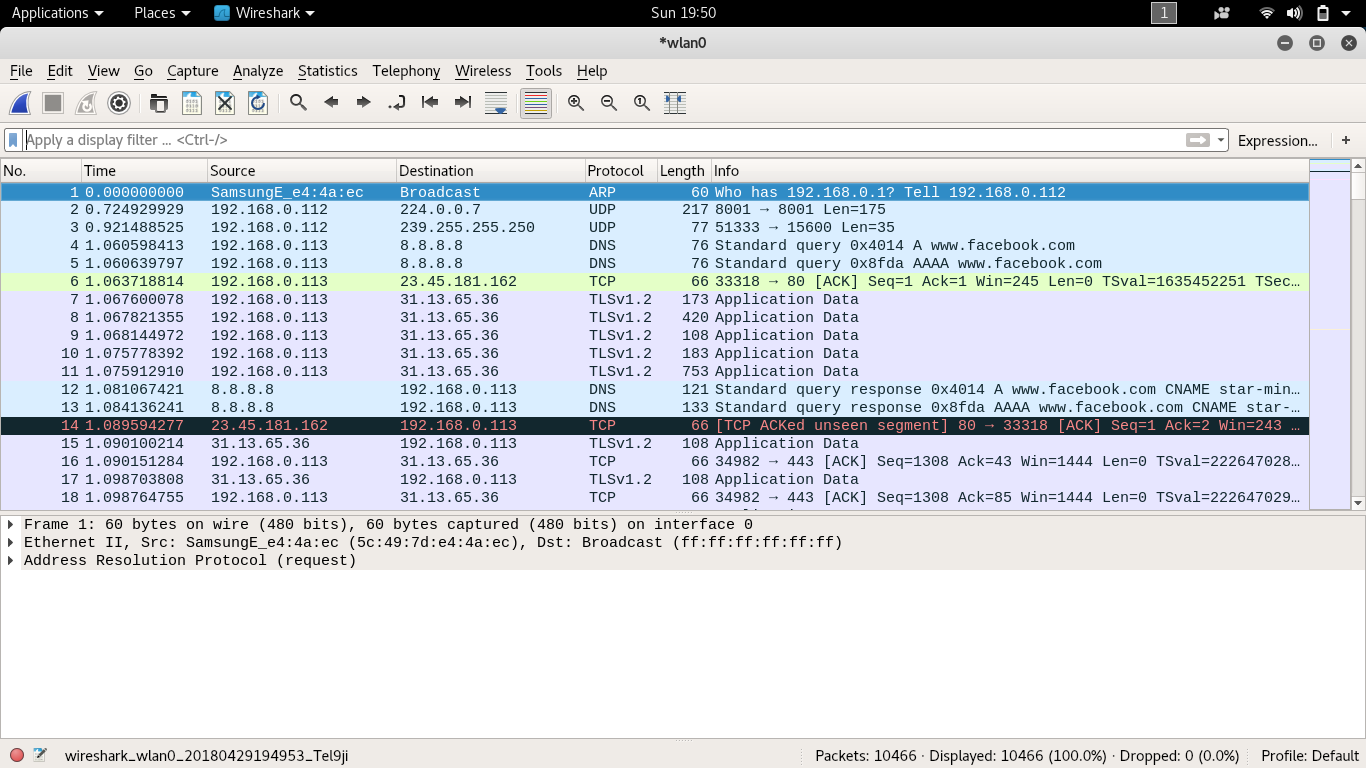
One of the first things I did was figure out my IP address. Because I have a DHCP server running, my IP will be dynamically decided by the server.



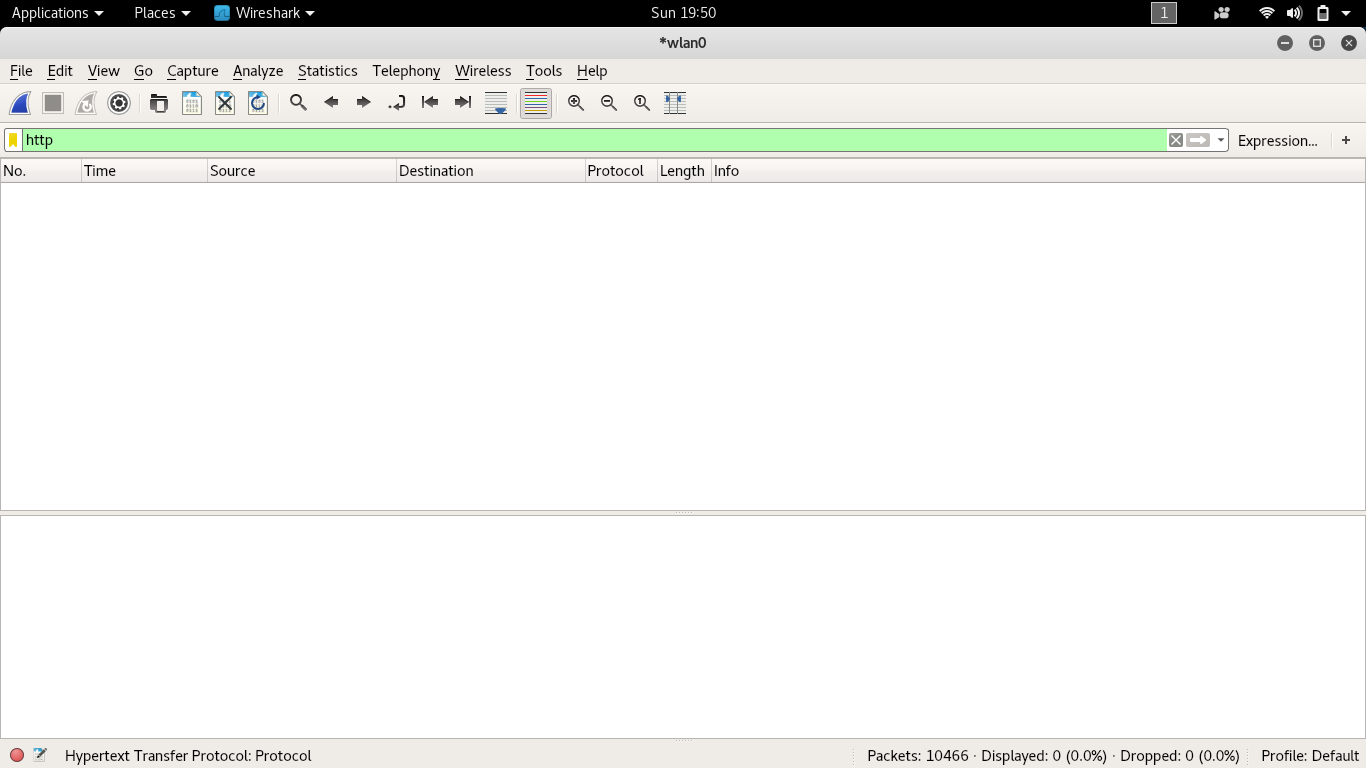
From this image you can see that I have the IP address: 192.168.0.113. Another thing I did was figure out the IP address of Facebook so we can filter out these 2 IP addresses.



From this image, you can see that the DNS resolves Facebook to IP: 31.13.65.36. (I was unable to find the screenshot; however, I took one my Windows partition)

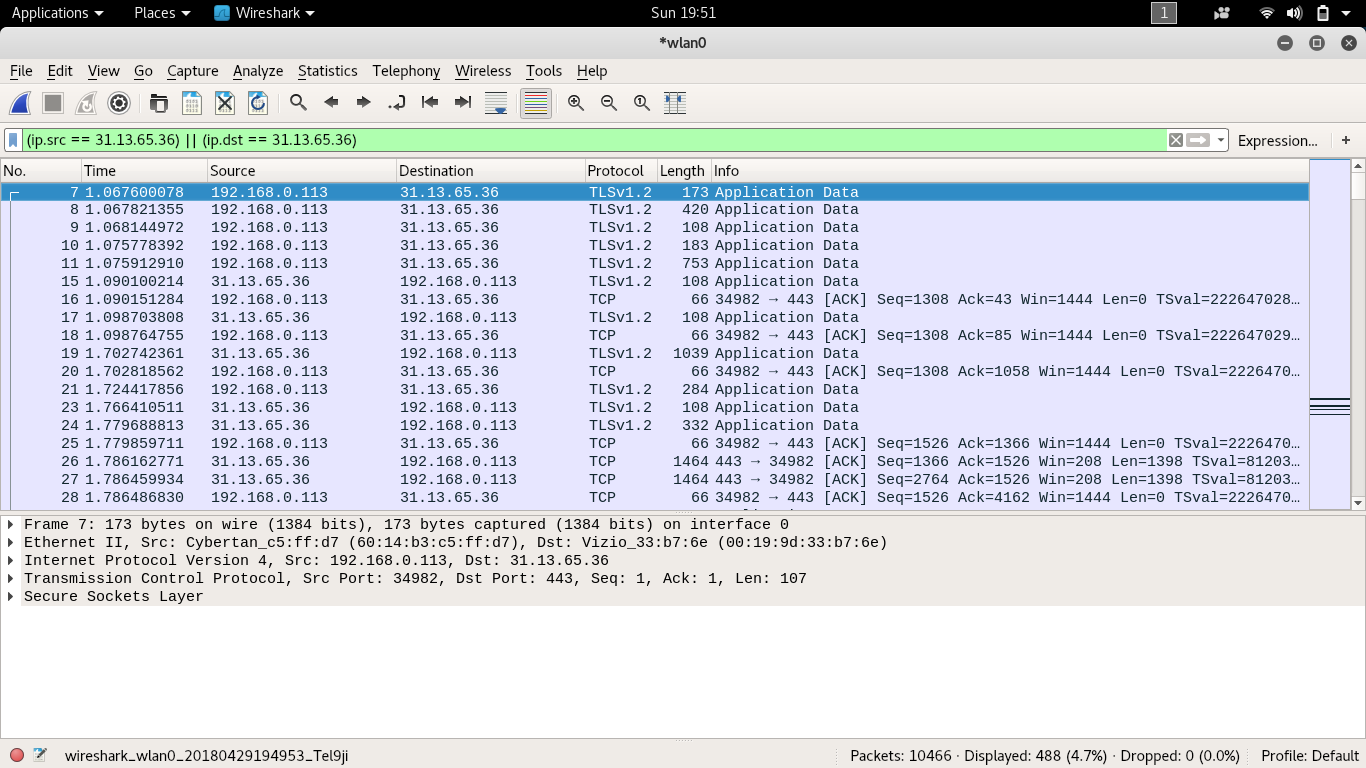


After starting restarting the capturing process and logging in to my Facebook account, we see a lot of traffic between my computer and Facebook’s server.



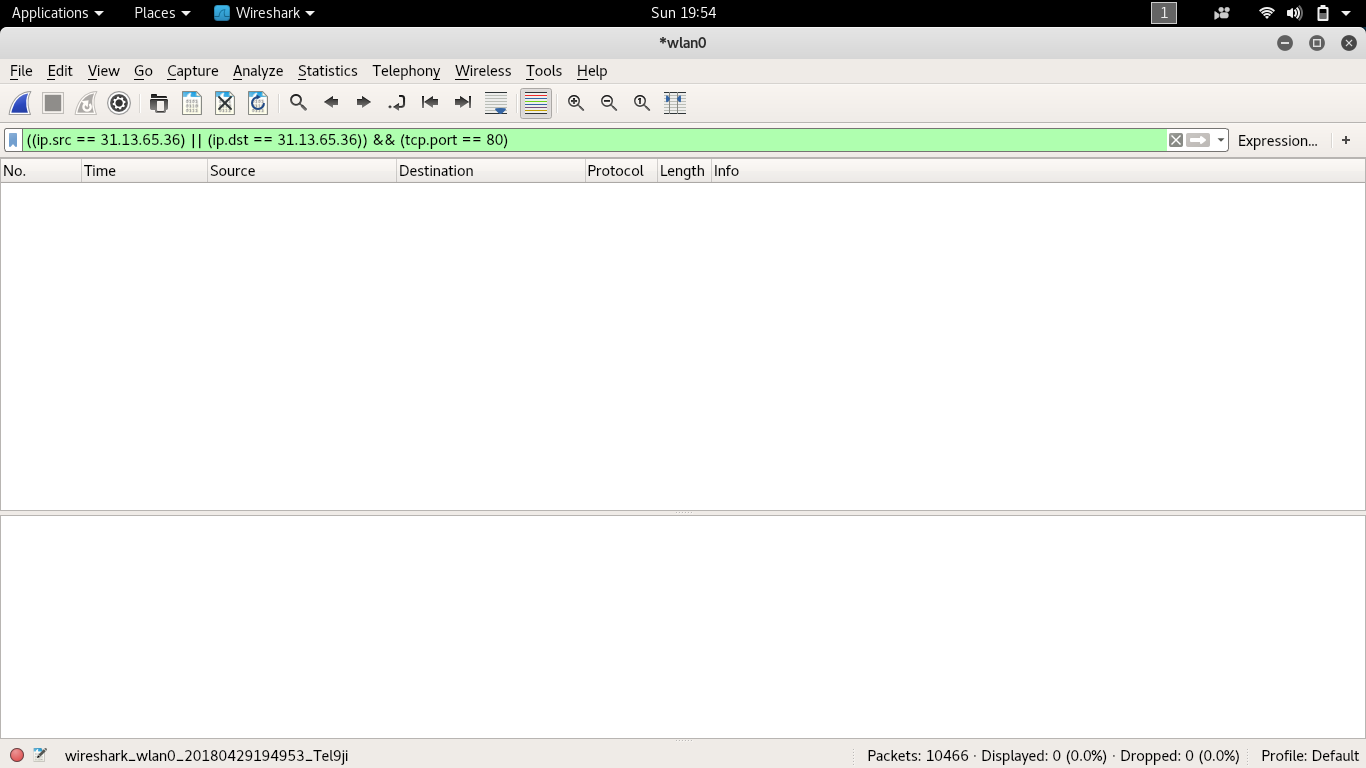
|  |  |
| --- | --- |
| Filter | Meaning |
| http | Only TCP packets that use the http protocol are displayed |

By filtering all packets so that only HTTP packets are shown we get a peculiar result: no packets. After some googling and rereading the prompt, I realized that because all the traffic between my computer and Facebook’s server is encrypted, no HTTP packets were exchanged. I googled many ways to give Wireshark my private RSA key so that it could unencrypt the packets and see the data, however my research shows that after Firefox version 48 this was disabled.

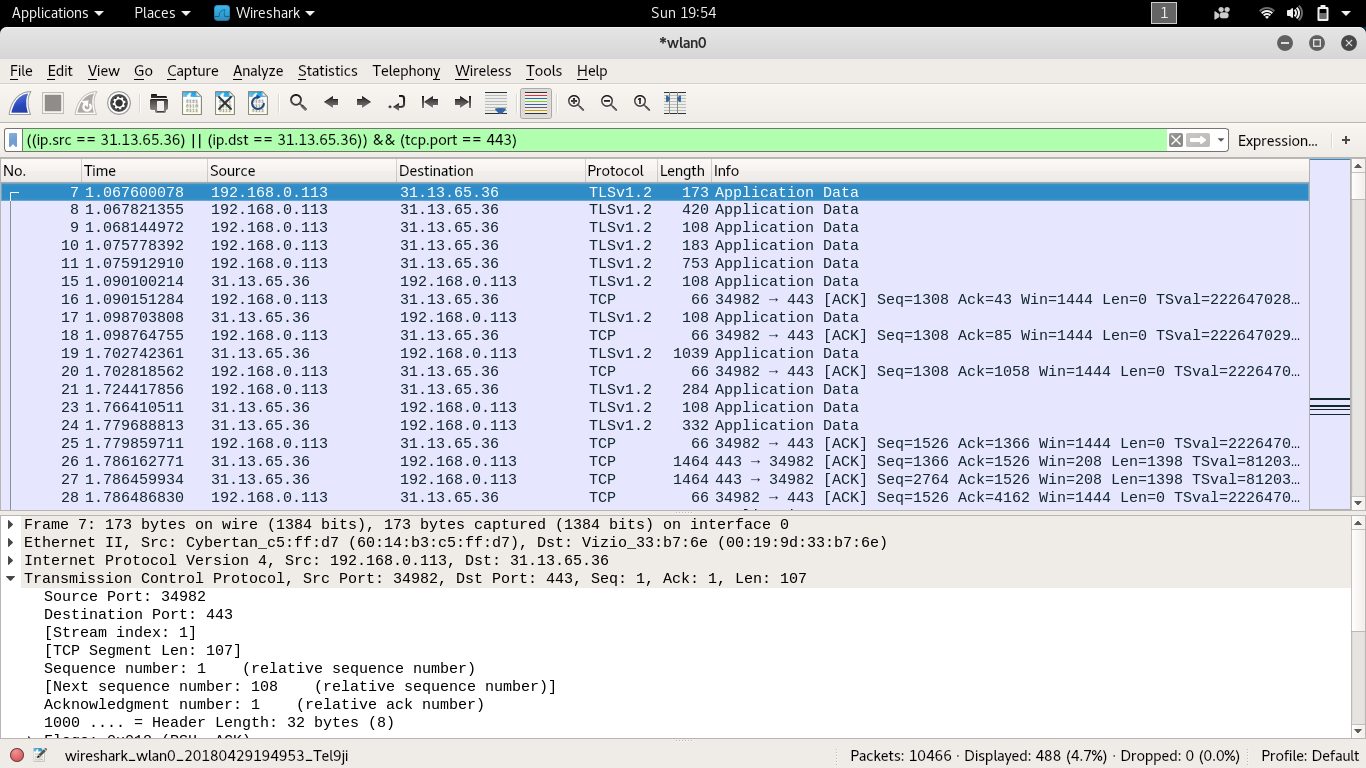


|  |  |
| --- | --- |
| Filter | Meaning |
| (ip.src == 31.13.65.36) || (ip.dst == 31.13.65.36) | Only packets that are sent from Facebook or sent to Facebook are displayed |

I then chose to filter by IP address rather than protocol and we see all the packets between my computer and Facebook. As you can see, there are many TLS packets being sent, this is the new encryption standard after SSL.

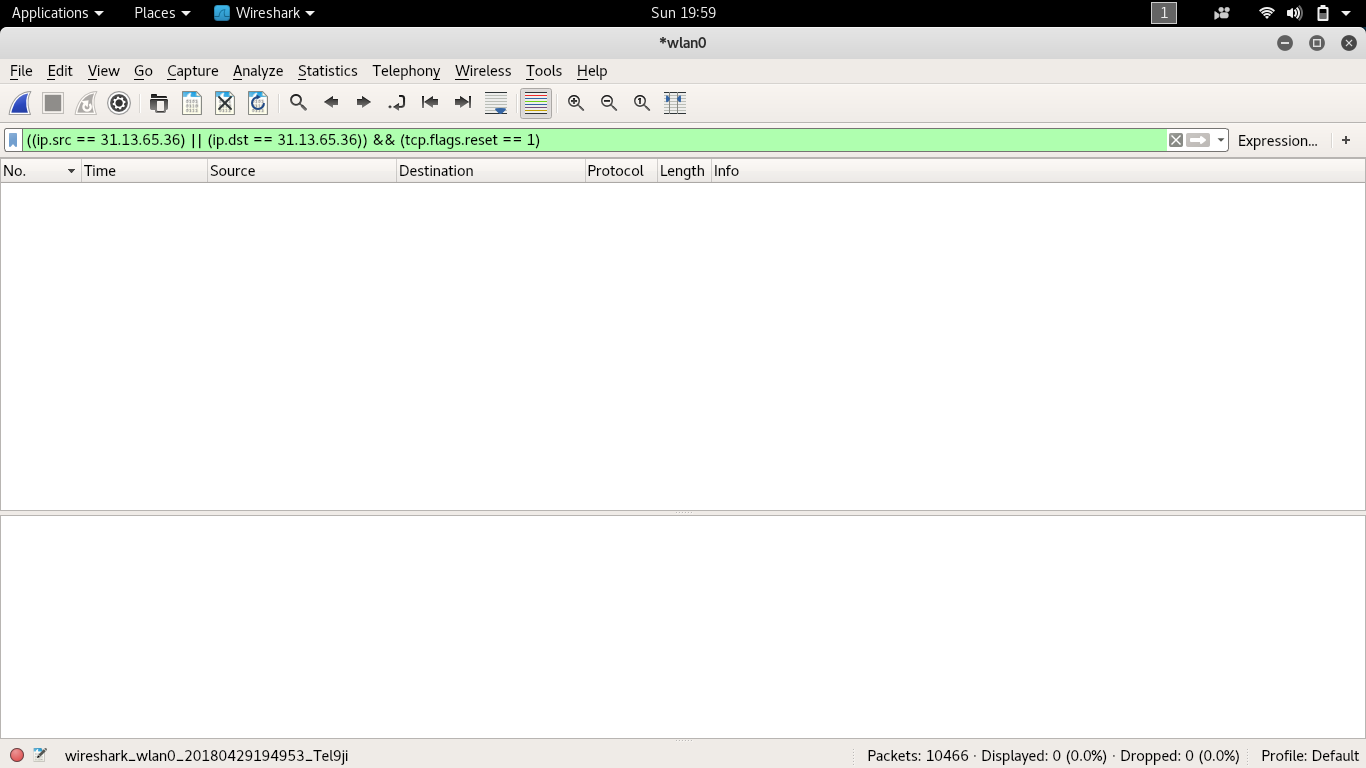


|  |  |
| --- | --- |
| Filter | Meaning |
| ((ip.src == 31.13.65.36) || (ip.dst == 31.13.65.36)) && (tcp.port == 80) | Only packets that are sent from Facebook or sent to Facebook and are addressed to port 80 are displayed |



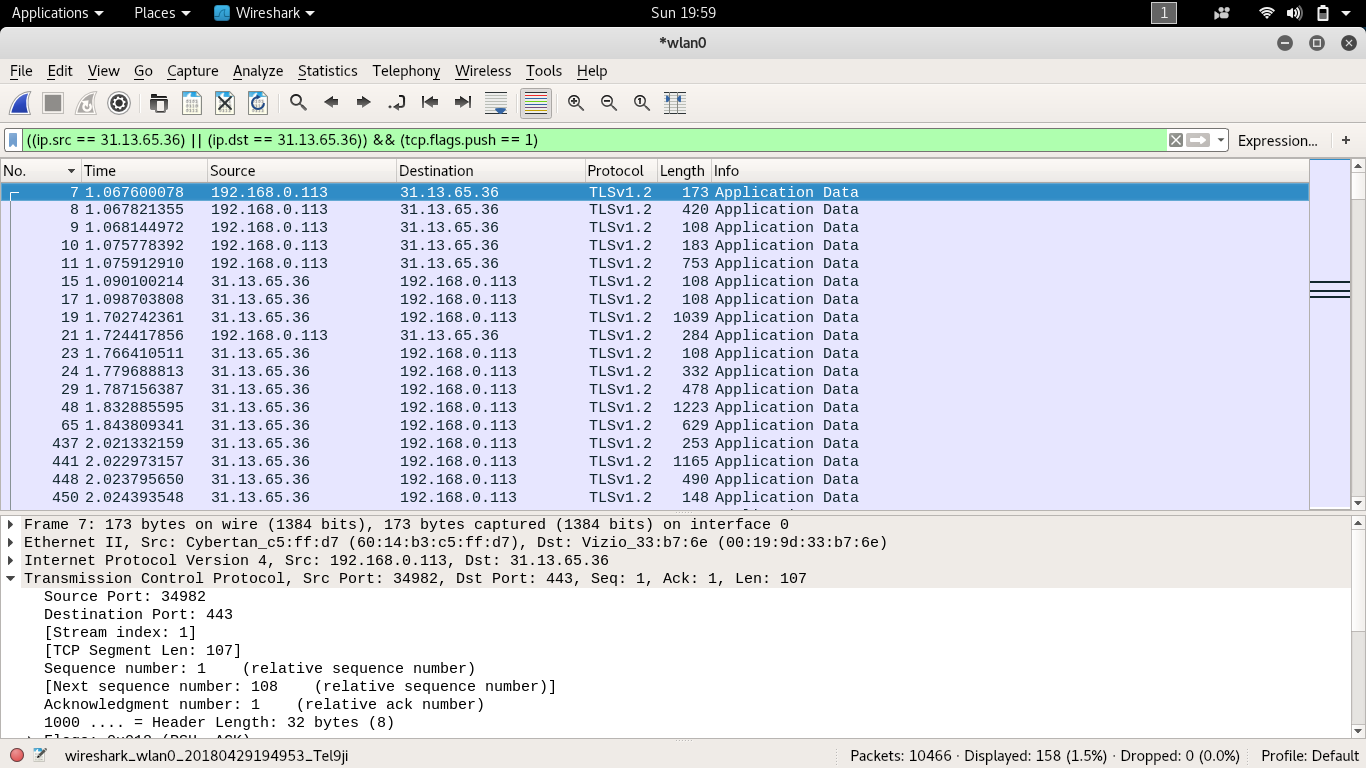
|  |  |
| --- | --- |
| Filter | Meaning |
| ((ip.src == 31.13.65.36) || (ip.dst == 31.13.65.36)) && (tcp.port == 443) | Only packets that are sent from Facebook or sent to Facebook and are addressed to port 443 are displayed |

As you can see, HTTPS does not use port 80 but port 443. Normal HTTP uses port 80.



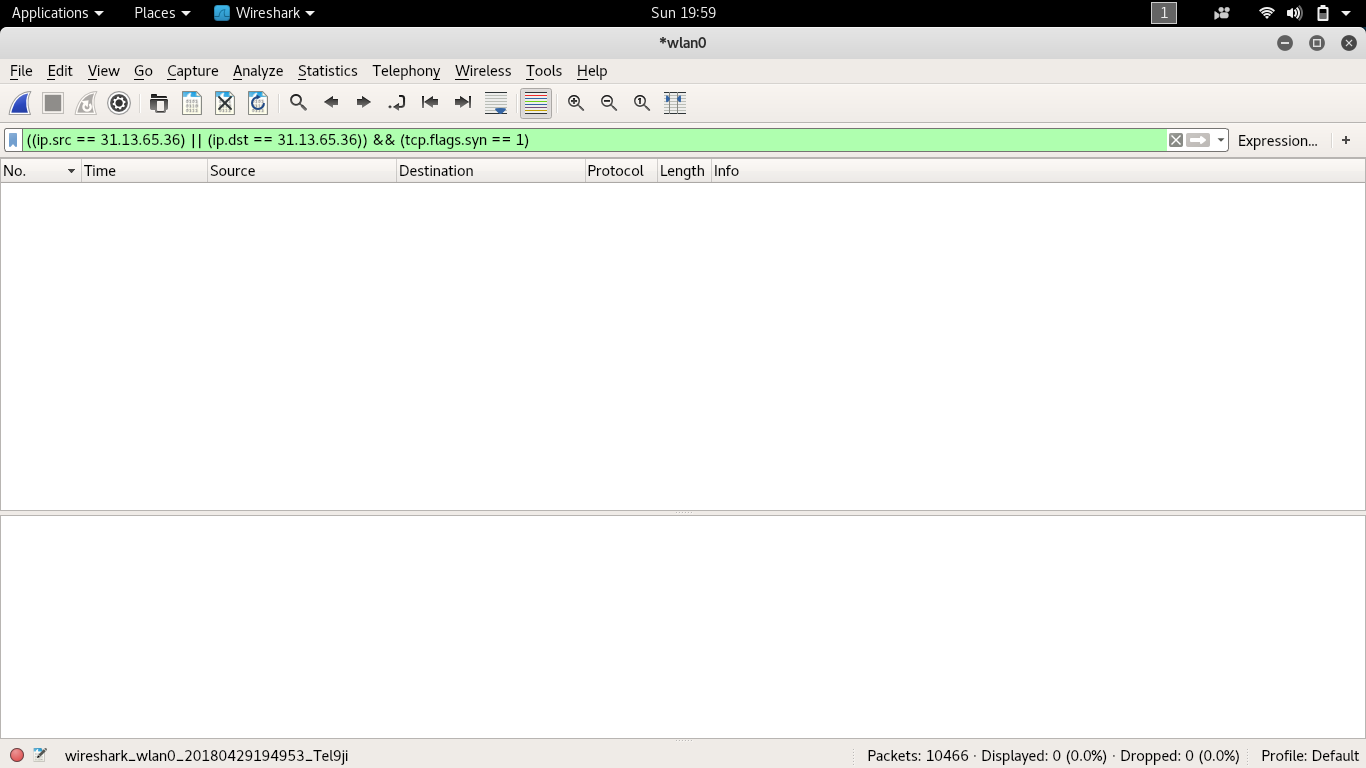
|  |  |
| --- | --- |
| Filter | Meaning |
| ((ip.src == 31.13.65.36) || (ip.dst == 31.13.65.36)) && (tcp.flags.reset == 1) | Only packets that are sent from Facebook or sent to Facebook and have the RST flag are displayed |

Each TCP packet has various flags set. The reset flag is set when we receive a packet that we weren’t expecting. As you can see, there were no packets with RST flag set.



|  |  |
| --- | --- |
| Filter | Meaning |
| ((ip.src == 31.13.65.36) || (ip.dst == 31.13.65.36)) && (tcp.flags.push == 1) | Only packets that are sent from Facebook or sent to Facebook and have the PSH flag are displayed |

Another common flag is the push flag. This flag is sent to the receiver to notify it to process the packets as they receive, rather than buffer them. There were many PSH packets.



|  |  |
| --- | --- |
| Filter | Meaning |
| ((ip.src == 31.13.65.36) || (ip.dst == 31.13.65.36)) && (tcp.flags.syn == 1) | Only packets that are sent from Facebook or sent to Facebook and have the SYN flag are displayed |

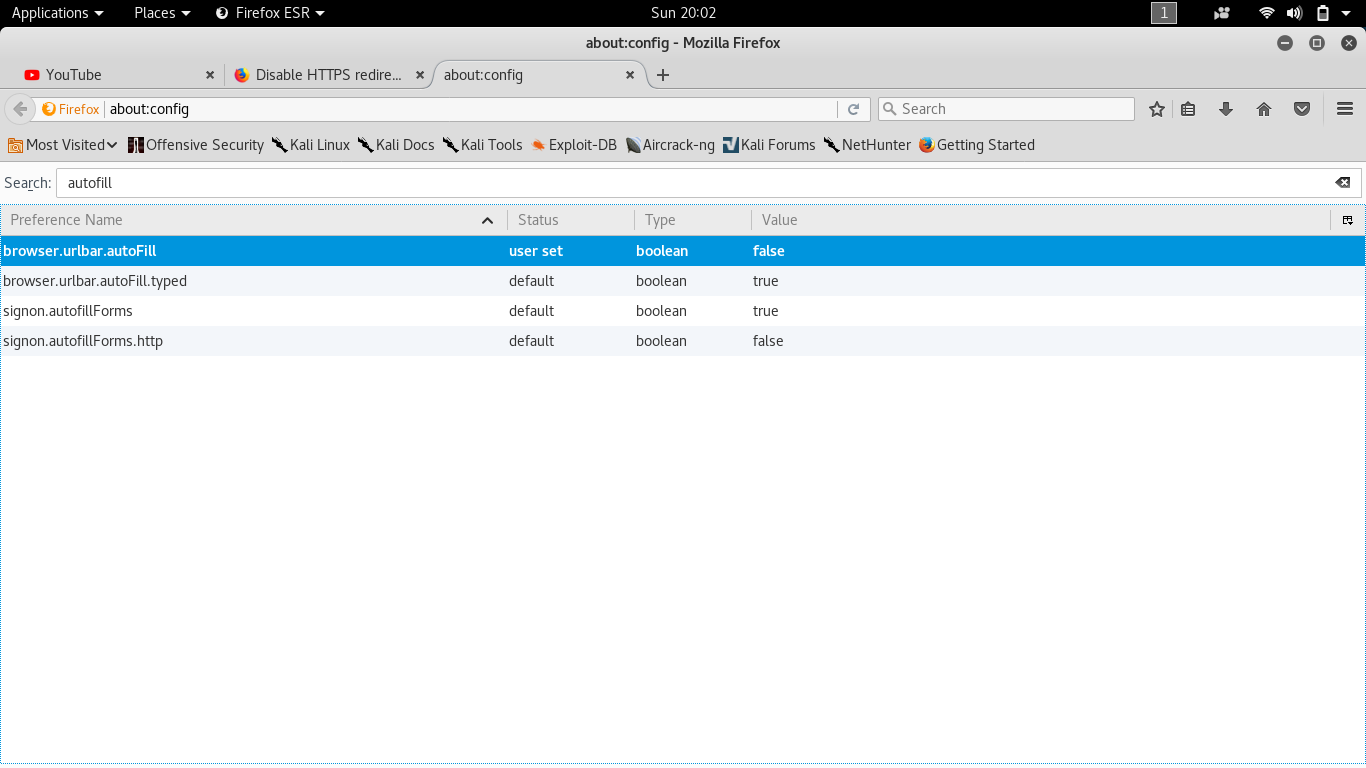
The last flag that we needed to analyze was the Synchronization flag. This flag is used to establish a 3-way handshake between 2 hosts. Surprisingly, I was unable to find any packets with the SYN flag set.

|  |  |  |  |
| --- | --- | --- | --- |
| Flag | SYN | PSH | RST |
| Facebook | 0 (0%) | 158 (32.4%) | 0 (0%) |

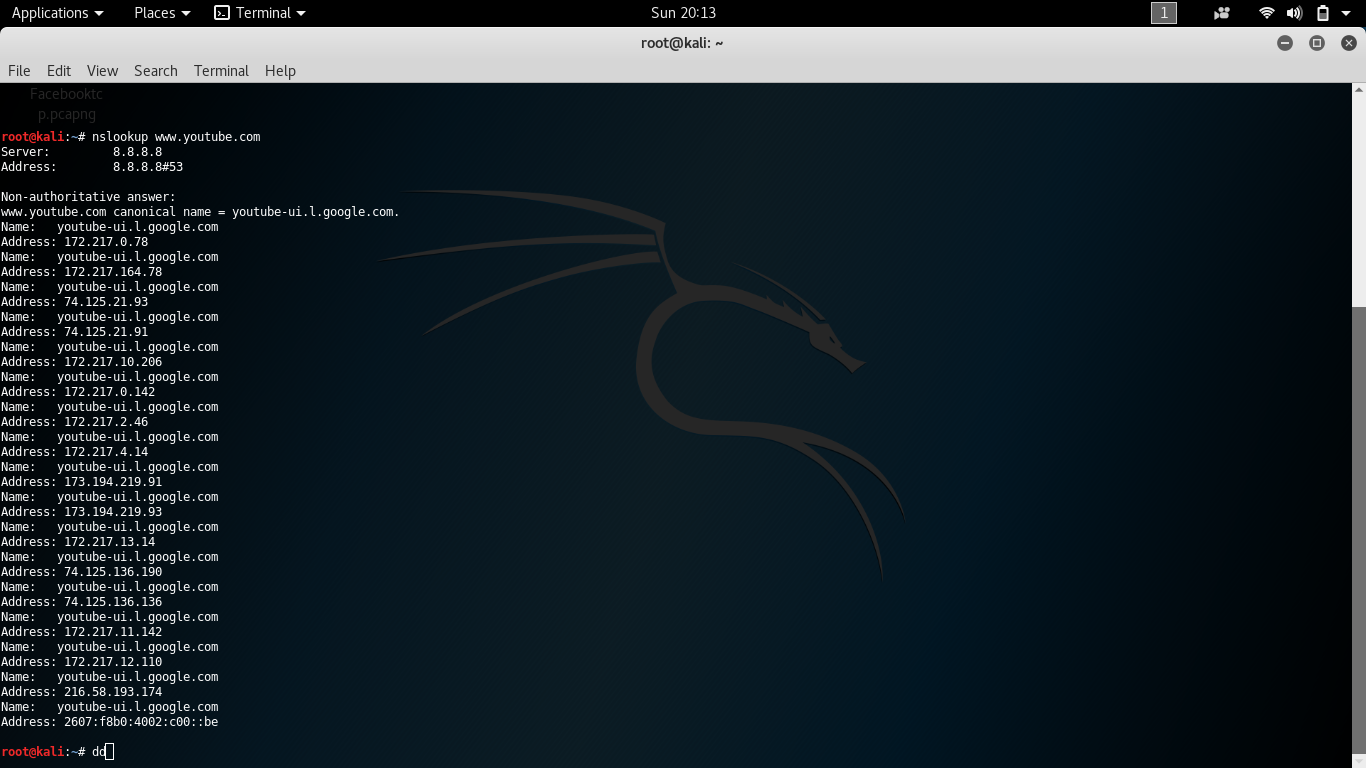
Above is a table of the 3 flags and the percent of packets that each flag contained. There was a total of 488 packets.

*YouTube*

The YouTube video I used is called “*Can Light be Black? Mind-Blowing Dark Light Experiments!”* <https://youtu.be/p-OCfiglZRQ> . With the Facebook connection, I did not attempt to turn off HTTPS because I didn’t want my Facebook password sent through the internet as plaintext, however with YouTube, there is no ‘security’ that I need to worry about. I Googled the internet to find a way to turn off HTTPS and use the HTTP version of the website.

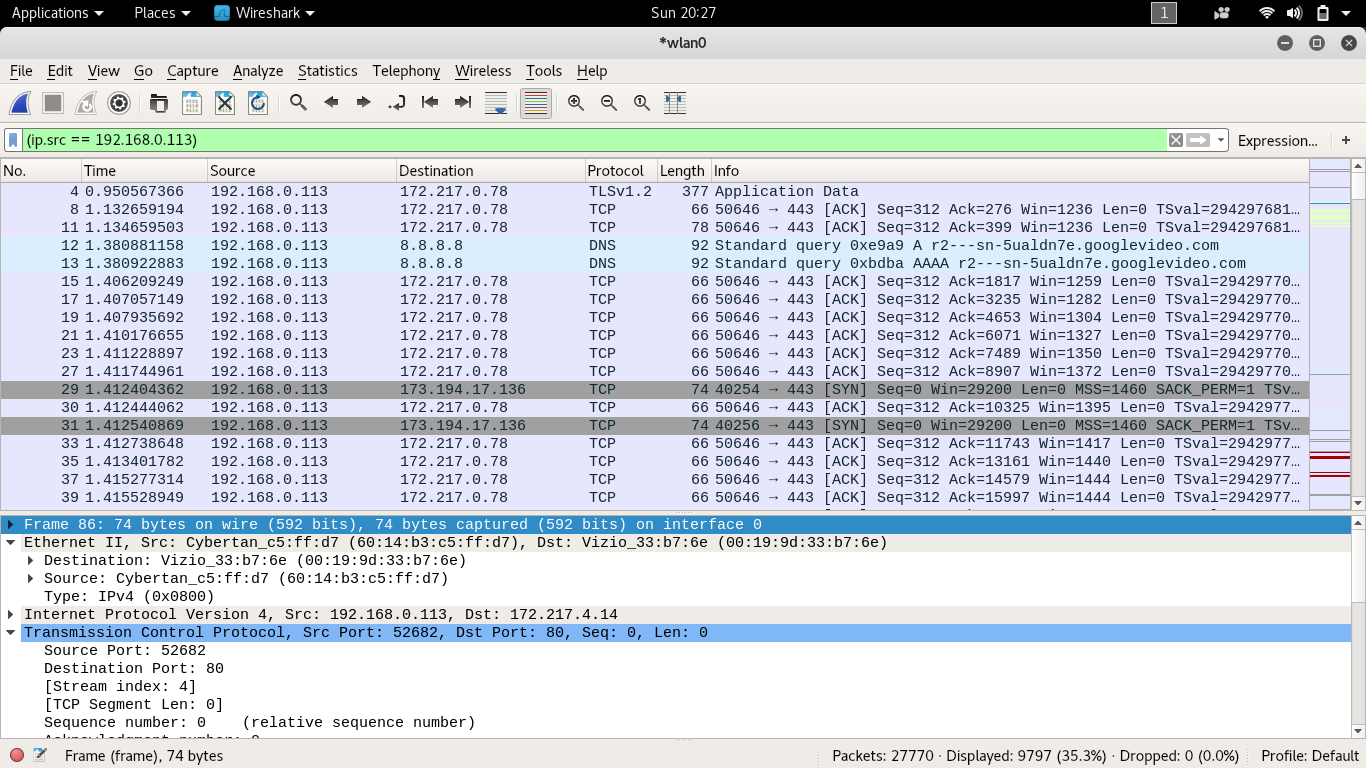


One way I found was to force Firefox to redirect me to the HTTPS website. However, this did not do anything and I was still being redirected to the HTTPS website. I decided it was best to continue with the project using HTTPS and maybe later come back to using HTTP only.



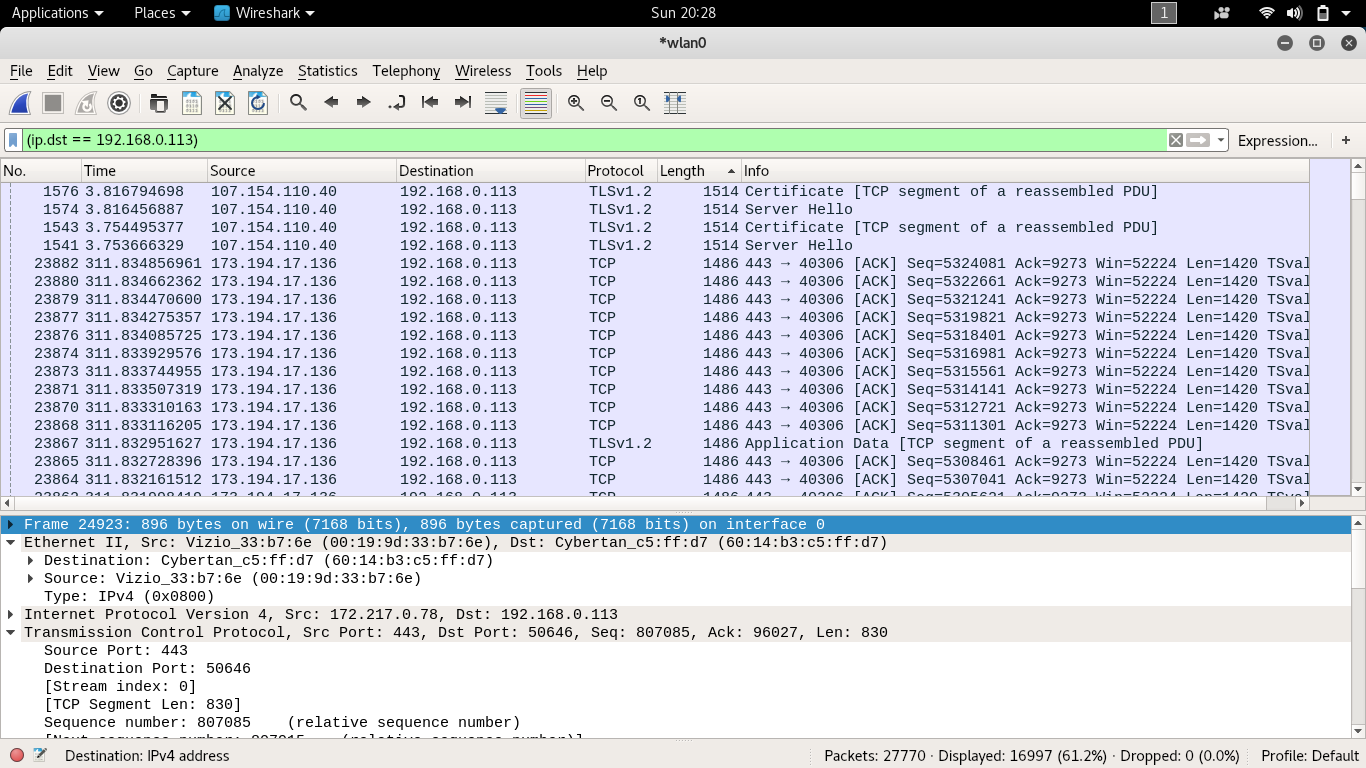
One of the first things I did was figure out the IP address of YouTube. As you can see YouTube resolves to many IP addresses, I’m assuming to load distributing. I was a bit worried, would I have to create a long filter, one for each IP address? I decided to just use my own IP address and create a filter with that.



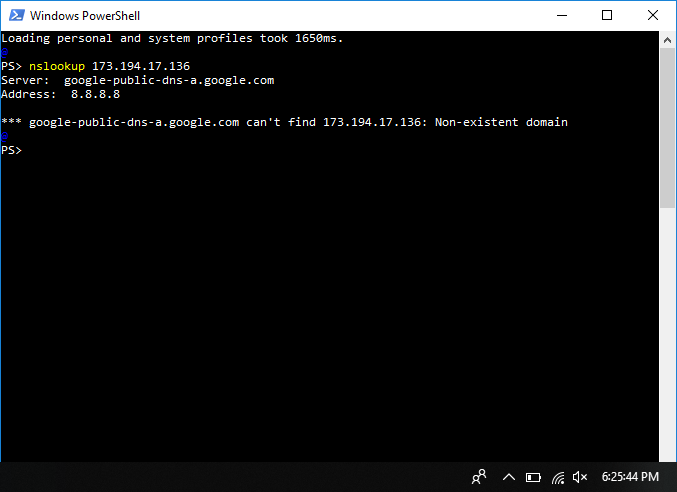


|  |  |
| --- | --- |
| Filter | Meaning |
| (ip.src == 192.168.0.113) | Only packets that are sent from my computer are displayed |

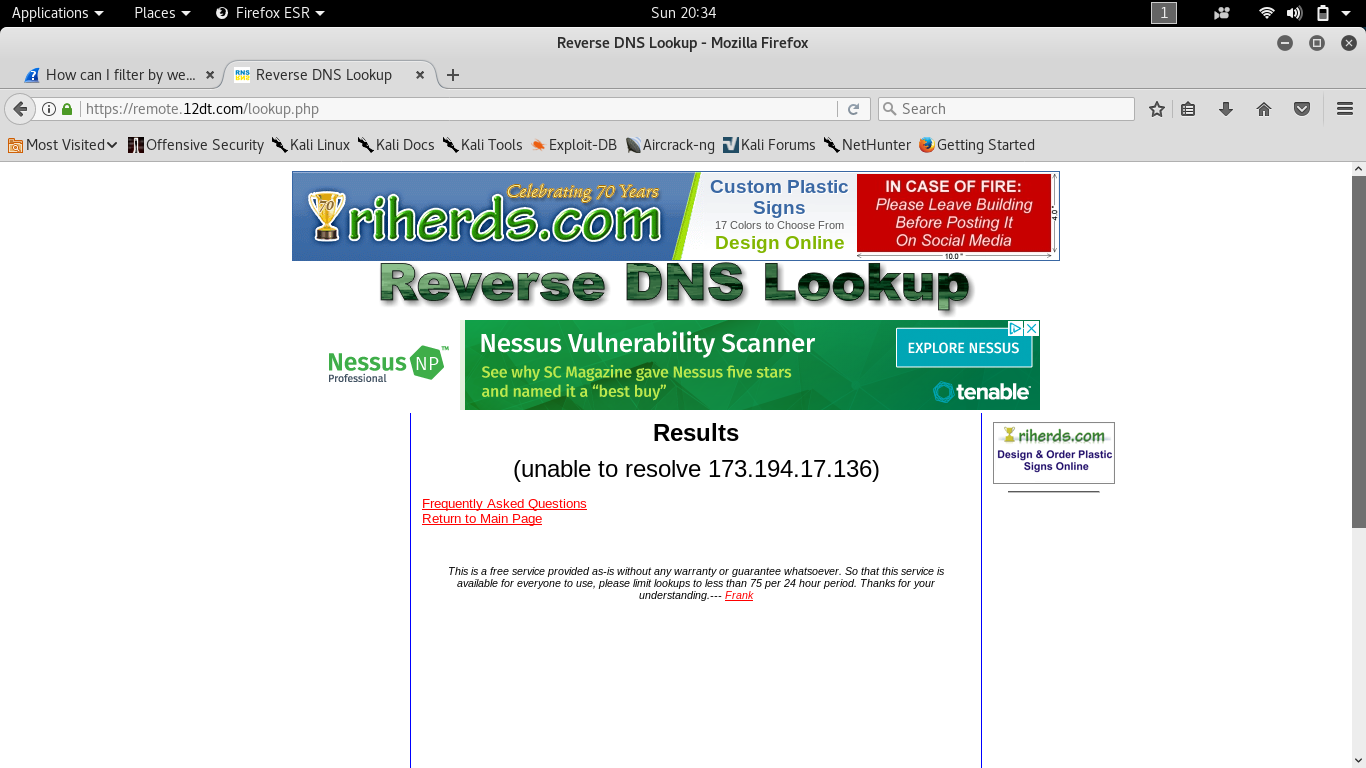
In this filter I got both YouTube packets and DNS packets, and may other IP addresses that didn’t show up in the YouTube DNS lookup.



I got a bit curious because I had a lot of packets with IP addresses that didn’t belong to YouTube. I then decided that I should figure out who’s IP address this is.

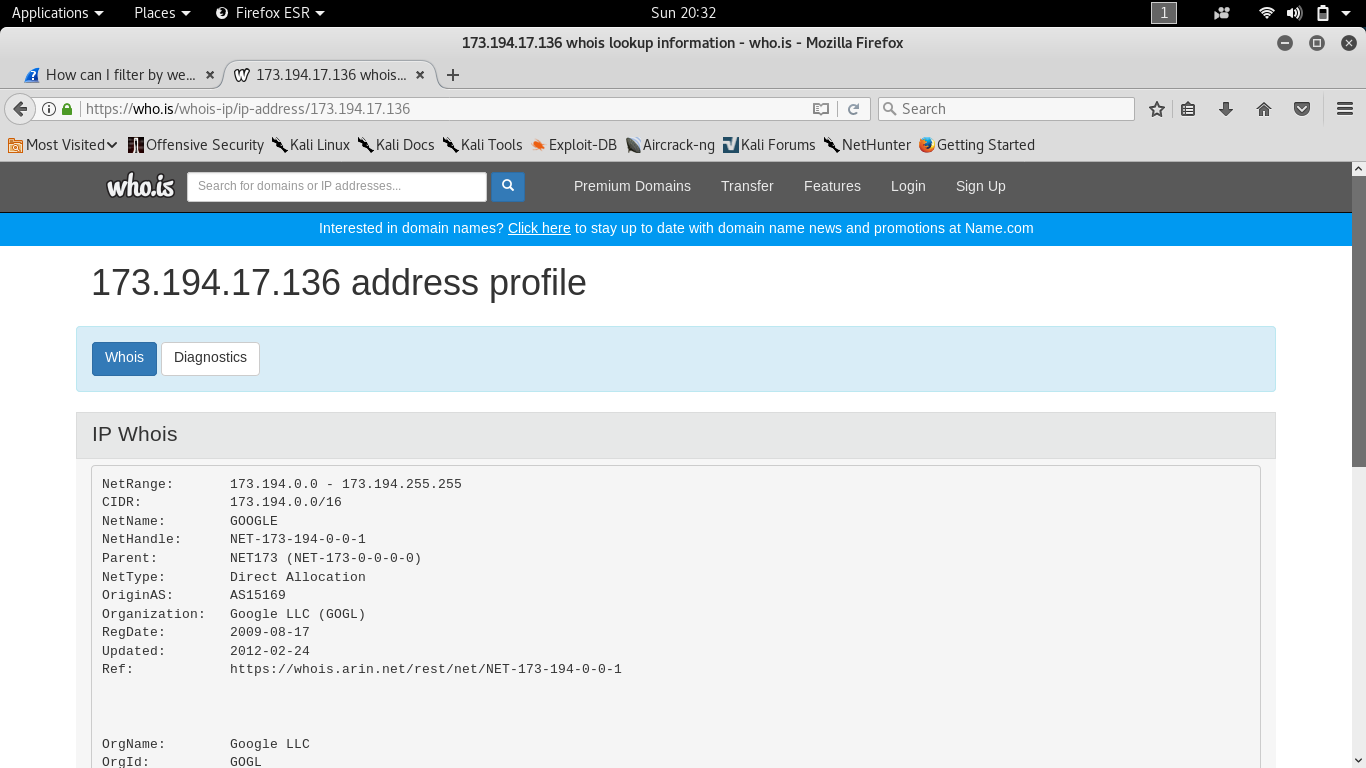


I did a reverse DNS lookup, but I couldn’t find a host or a server name.

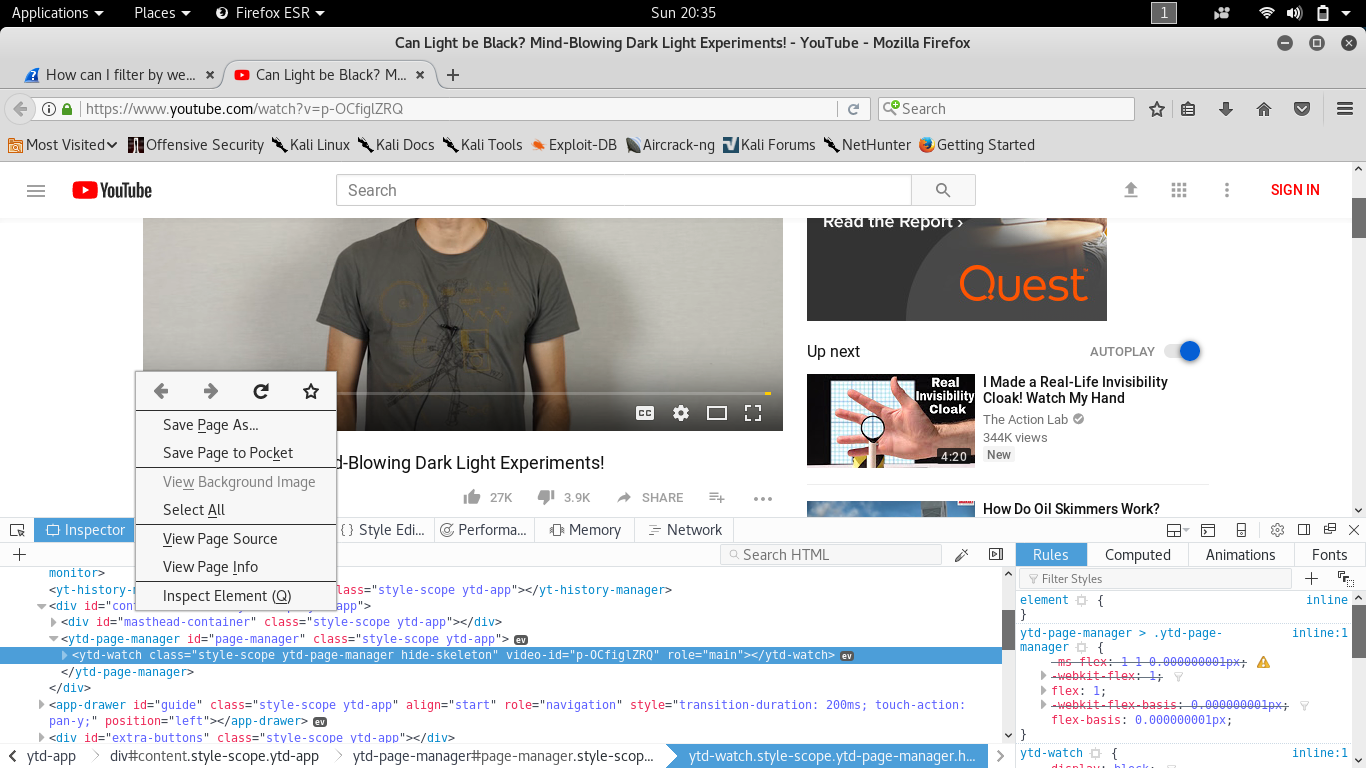




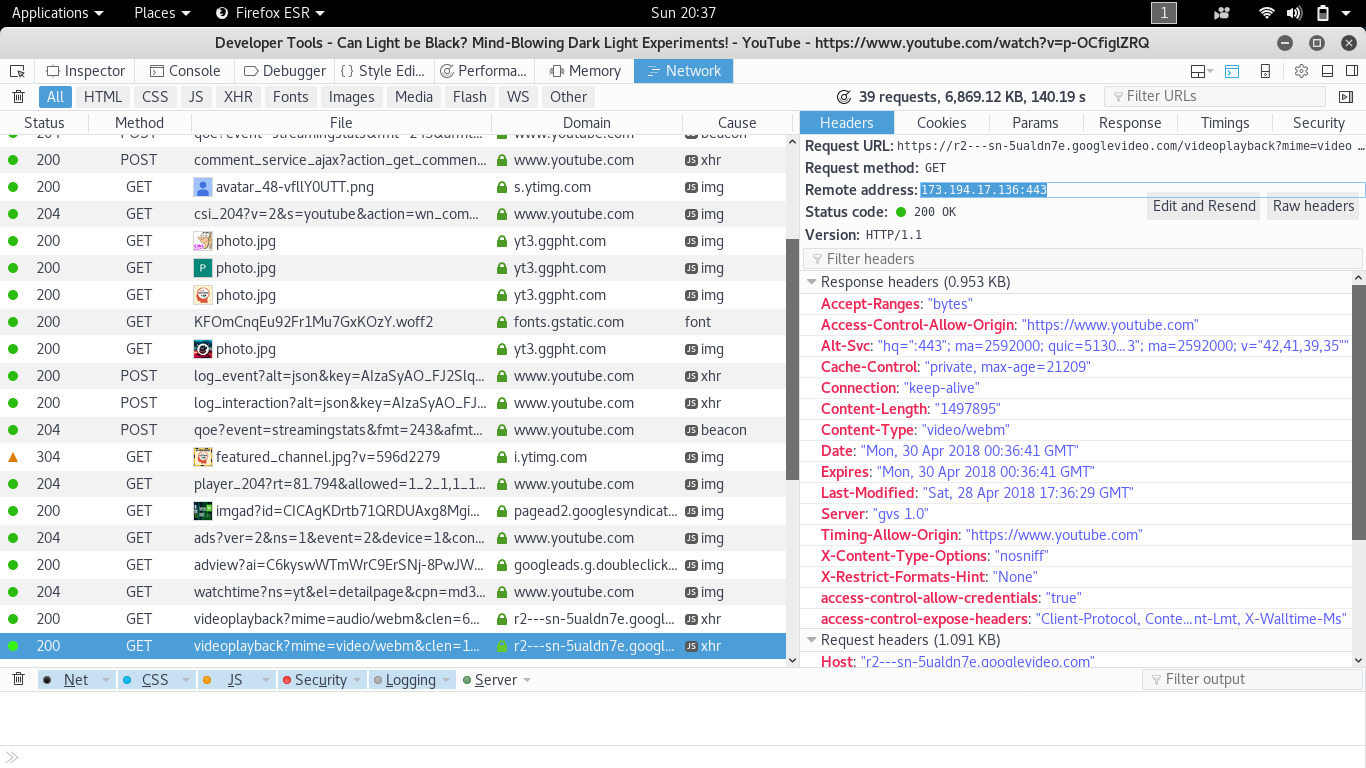
I decided to go to reverse DNS websites to see if I can get a domain name. However, these websites just confirmed that the command line told me. The IP address can’t be found. I then went to who.is to see if that can tell me what’s up with that IP address.



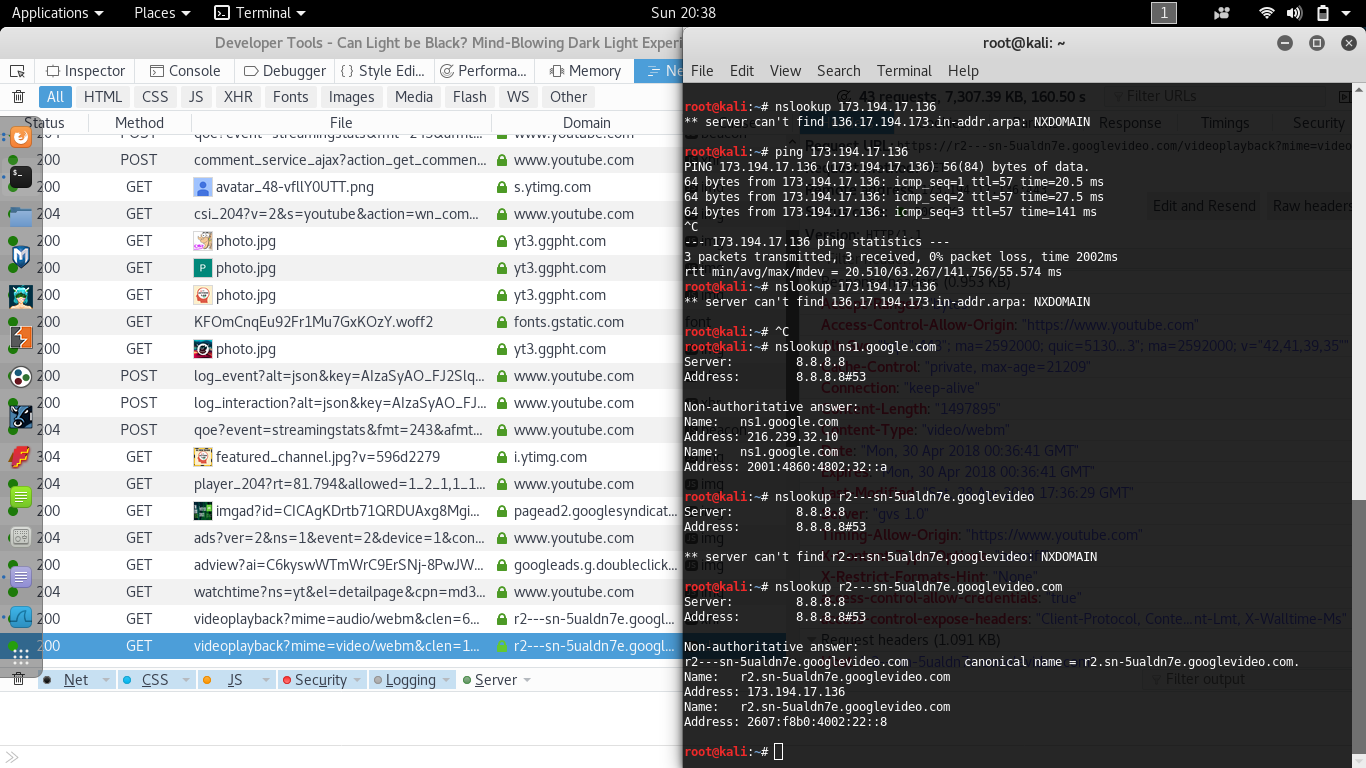
Who.is showed that that that block of IP address is owned by Google. I did this similarly to the other IP addresses and it showed me similar results. I was still intrigued, I wanted to figure out how I can get the domain of the server. So, I opened up Developer Options in Firefox.



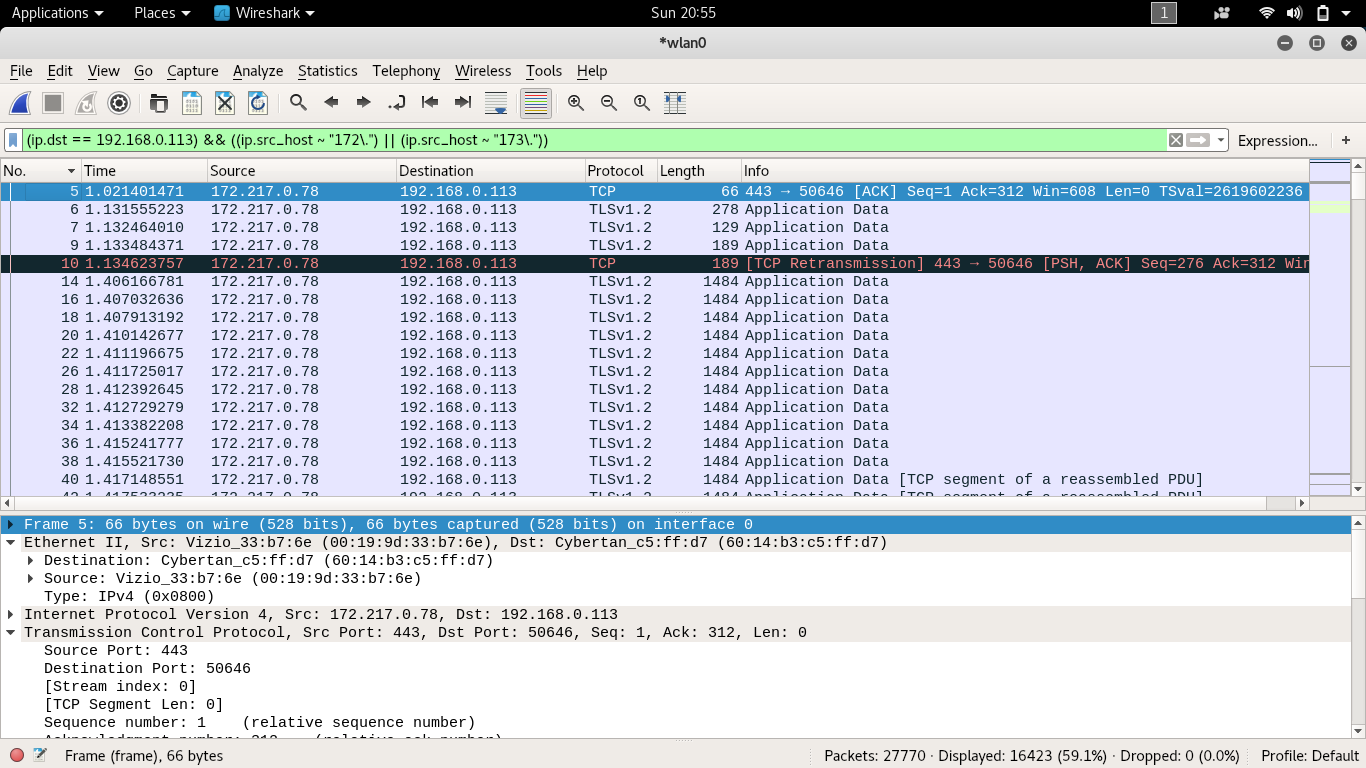
Going to the Network tab of the window, and then restarting the video we can see all the network traffic from Firefox, some of this traffic was of webm type. Exactly what we needed for figuring out the domain for that IP address.



Expanding that network request, we see on the right the IP address that we wanted and the URL of the network recourse.



Looking up that domain name we get what we wanted, the IP address.

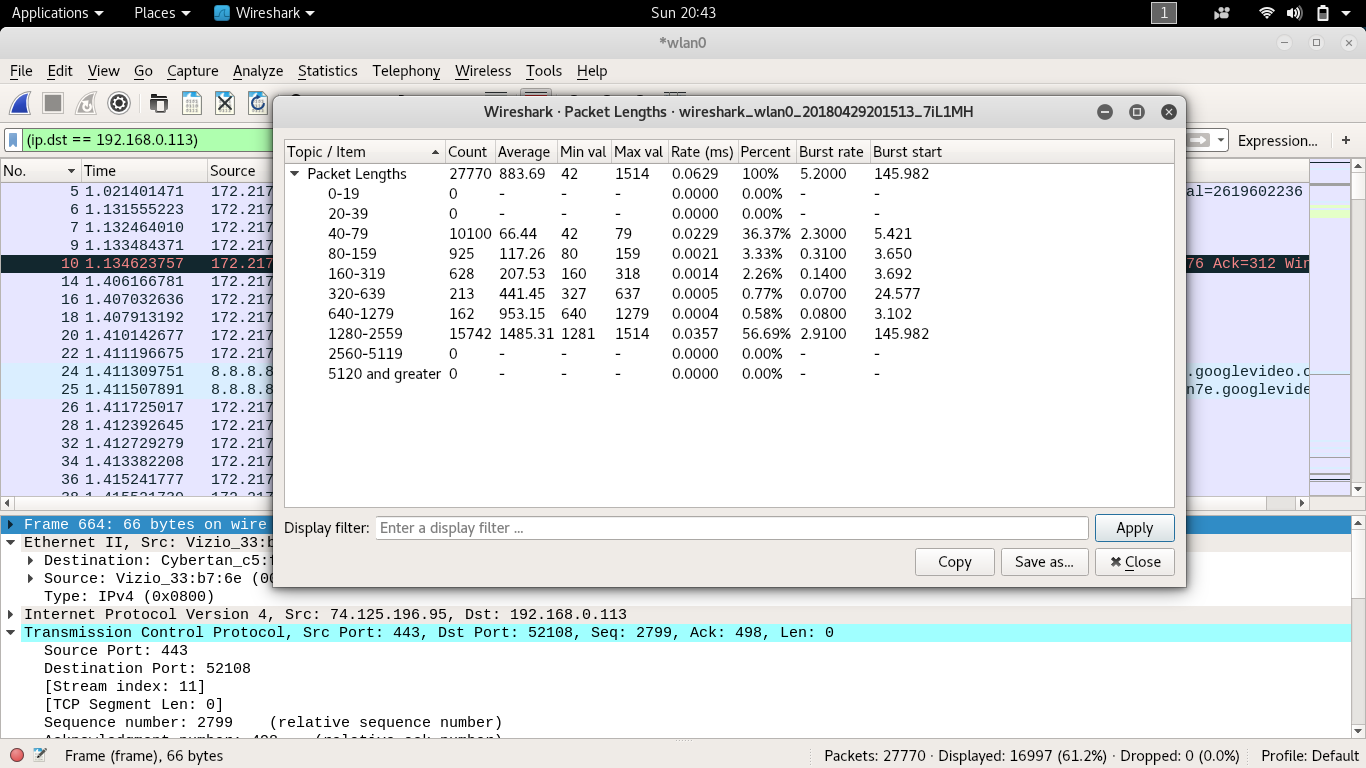


|  |  |
| --- | --- |
| Filter | Meaning |
| (ip.src == 192.168.0.113) && ((ip.src\_host ~ “172\.”) ||  (ip.src\_host ~ “173\.”)) | Only packets that are sent from my computer and that are received from IP addresses that match 172.\*.\*.\* or 173.\*.\*.\* are displayed |

I created a simple filter that filtered by the first byte of the IP address. This would filter out the DNS packets but keep all the YouTube packets.

|  |  |  |  |
| --- | --- | --- | --- |
| Flag | SYN | PSH | RST |
| YouTube | 21 (0.13%) | 539 (3.3%) | 12 (0.073%) |

Here is the table for the TCP flags. There was a total of 16,423 packets.



Wireshark has a cool feature that allows you to see the packet lengths as a table. This made it easy to create the histogram, shown below.

This is the histogram for the packet sizes during the YouTube part of this project.

I exported all the data to a comma separated value file (\*.csv), and uses excel to create a timeline for the data, shown below:

Below is a graph of the size of the packets versus the time on the video.