Writeup:

During this lab, I used Wireshark to analyze the packets flowing across my network. Wireshark is the most widely used application for network analyzer, it is free and open source. Wireshark allows users to capture and examine packets on a network, including headers, destinations, source information and much more.

Wireshark can be used for trouble shooting network issues, detecting intrusions and malicious devices, nefarious hacking, and as a teaching tool for students. The purpose of this lab is to give hands on experience with networks, specifically with identifying, analyzing, and capturing packets. I learned the difference between different protocols like ARP, TCP, UDP, DNS, HTTP, and HTTPS, and built skills learning how to monitor a network. I expected to see TCP handshakes, encrypted data when using HTTPS, and overall just how much goes on on a network during normal usage even when I am not currently searching anything.

This lab helped me understand how networks work at a low level, getting to see each packet move across the network was really interesting, as was seeing the devices and servers handshaking and making sure no packets get dropped in the process made it seem so much more in depth than I ever thought it could be.

The first step of this lab was to download and install Wireshark, this was a simple and straightforward process. I had to download Npcap also so I could capture the packets. The install took a few minutes so during that time I researched the history of Wireshark, like how it was originally called Ethereal until 2006, but had to be changed due to copyright issues.

Once Wireshark was downloaded, I launched it and was presented with a few different network interfaces. These interfaces included WiFi, Ethernet, adapter for loopback, and Wireguard for my VPN. I chose WiFi, and next I put Wireshark into capture mode and accessed a single website. The data I captured shows the SYN, SYN/ACK, and ACK process of TCP along with the encrypted HTTPS data. Even though the data was encrypted I could still see the source and destination of the packets, including my search going out and the website responding.

During this lab I filtered through the packets to find ARP packets, as shown in figure 1, these packets included the “Who has” and “is at” and “ARP Announcement". These ARP packets are used to find the MAC address associated with the given IP address.

The next protocol I looked for were TCP packets, these packets ensure a stable and reliable connection. As seen in figure 2, when combing through them I found the SYN, SYN/ACK, and ACK process, this three way handshake allows for a reliable connection.

The next step in the lab was to filter for the UDP packets. UDP packets prioritise speed over reliability of the data, thus no three way handshake like TCP. As seen in figure 3 UDP is much lighter and faster, used for DNS, online games, and video streaming for example. It is used when a lost packet is not detrimental to the program.

Next I used the HTTP filtering in wireshark, as shown in figure 4, I did have to go to a specific website that uses HTTP because most websites autodirect to HTTPS. I went to the forum website of my car because I know they only use HTTP, and captured the traffic in Wireshark. Wireshark would show in plain text the username and password if I put it in, but I did not want to show that. HTTP is much less secure than HTTPS because it is not encrypted and stores passwords, headers, and webpage content in plaintext.

After filtering for HTTP, I filtered for HTTPS. Filtering for HTTPS in Wireshark was easy, as all I had to do was filter for TLS. Since most of the data is encrypted I don't get to see as much as I did with HTTP, as seen in figure 5, I could see IP addresses, and length.

Next I filtered by FTP, FTP is the file transfer protocol, it is an older style transferring protocol for files from one host to another, since it is rather outdated I could not capture any FTP packets in the four hours I was packet sniffing.

Next I captured ICMP packets, these appear as Echo requests when a user pings a website, as seen in figure 6, networks use it to see if a server is reachable, the ones demonstrated were not.

Finally I filtered for DNS packets in wireshark. As seen in figure 7, anytime I attempted to access a website I saw a DNS request, I found it cool how I could see which servers my computer tried to connect to.

While I was collecting those, I also collected the one hour Wireshark log. I found that Wireshark shows a much more indepth view of what is happening on my network and it was a headache even finding the data I did find from my firewall. Figure 11 shows my 1 hour Wireshark run and Figure 12 shows my 4.5 hour Wireshark capture. This was one of the most interesting captures I did because it showed so much data and I could see when I did things online, when other people came home and when I figured out my VPN was on, thus corrupting the first thirty minutes of my capture for myself.

I did run into a problem because for the first 30 minutes of the 4.5 hour capture, I had a VPN running thus the protocol was only “Wireguard”, but I found that to be a fun learning experience in and of itself. During these captures I saw many different protocols that were talked about earlier from TCP and UDP, and of course DNS packets as well. Getting to see what sort of traffic occurs even when I am not online is a cool experience and it made me realize just how busy a network is.

In conclusion this lab was a really cool practice exercise in seeing a granular view of my network and I gained a better understanding of how my network works, from moving packets, to handshakes, to how my vpn works. I also learned the difference in the different protocols and how they keep the network flowing reliably and quickly.

Reflection:

Today networks are more complex than ever, everyone has at least one device that is connected to the internet and the application Wireshark can be used to capture, and analyze the data flowing through the network. Wireshark is the most popular and widely used network analyzer, it provides insight on what the packets contain, who they are coming from and going to.

Wireshark can be used both as a tool to monitor networks and also as a tool to detect anomalies. Wireshark is a powerful, open source tool used by security professionals, hackers, students and by people who are interested in their network.Wireshark was originally called Ethereal and released in 1998 as a network analysis tool.

Wireshark works by capturing live packets, or packets in transfer on a network. Wireshark can also be used for analyzing saved PCAP files of saved network captures. Wireshark It allows the user to breakdown the network traffic, letting you see each individual protocol from TCP, UDP, and HTTP.

Wireshark can also filter the traffic to make it easier for analysts to isolate what they are looking for. Wireshark can be filtered by port, IP, contents, protocol, and much more. This can be very useful when the network has heavy traffic flow, or after an incident.

Wireshark can be very useful during incident response because it can help reconstruct the network activity leading up to the incident. Security professionals can determine what systems are affected, how the attack occurred, and what method was used so they can prevent it in the future. This usage makes Wireshark an invaluable tool in network security. Wireshark can also be set up on a network to monitor for network scans, connection, and suspicious activity within the network.

Wireshark can be used to detect many cyber attacks, these include man in the middle attacks, port scanning, ARP poisoning, ect. According to the paper titled Wireshark as a Tool for Detection of Various LAN Attacks “wireshark can prove to be extremely beneficial in such scenarios and accentuates how various local area network attacks like ARP poisoning,DOS attack,MAC flooding and DNS spoofing can be detected using wireshark and also provides some mitigation techniques for these attacks.” (Iqbal and Naaz, 2019). Because wireshark is free and open source, it makes it very usable at any business, school, home or anywhere a wireless network is used.

Wireshark can also be used for training and education. Since it is a free and open source application available on a wide array of operating systems, it allows nearly anyone with a network connection to learn and understand network analysis. This is incredibly useful for students as they can see the topics they learn about in class, like ports, protocols, network topography, and the OSI model, working together on a screen. It also allows users to understand what normal traffic looks like, and to be able to differentiate between normal network behavior and deviations.

The use of Wireshark in classrooms and business for teaching gives the user a safe space for students to gain a better understanding of networks as well as allowing users to use a real application that actual companies use to protect their networks. The ability to actually poke around in an application that is used in the field is indispensable as a teaching tool.

According to the paper titled Evaluation of the Capabilities of WireShark as a tool for Intrusion Detection, they state, “With the appropriate driver support, Wireshark can capture traffic ”from the air” and decode it into a format that helps administrators track down issues

that are causing poor performance, intermittent connectivity, and other common problems”(Banerjee, Vashishtha, and Saxena, 2010). This shows how Wireshark can be used by corporations or individuals for diagnosing network issues.

Wireshark can not just be used during an incident, but it can help keep a network running safe and strong. A person can use Wireshark to detect network anomalies like latency, DNS delays. According to the paper Wireshark as a Tool for Detection of Various LAN Attacks, they state, “Consider a situation in which a network administrator receives a complaint that some people in the network are not able to access the network resources. The first step is to execute wireshark on a system to begin packet capture” (Iqbal and Naaz, 2019). This ability to be able to instantly view the flow of traffic, and to see where the problem is occurring shows how useful the proactive use of Wireshark can be.

Wireshark is not the most perfect tool though, it has its limitations. For instance, Wireshark cannot decrypt a lot of encrypted traffic using HTTPS. Because Wireshark does not have the private keys necessary for decrypting HTTPS traffic it cannot be used for finding passwords like it can be used for http traffic. Wireshark can also not replace a SIEM, but it can be used alongside one to gain a better understanding of the data and packets at a lower level.

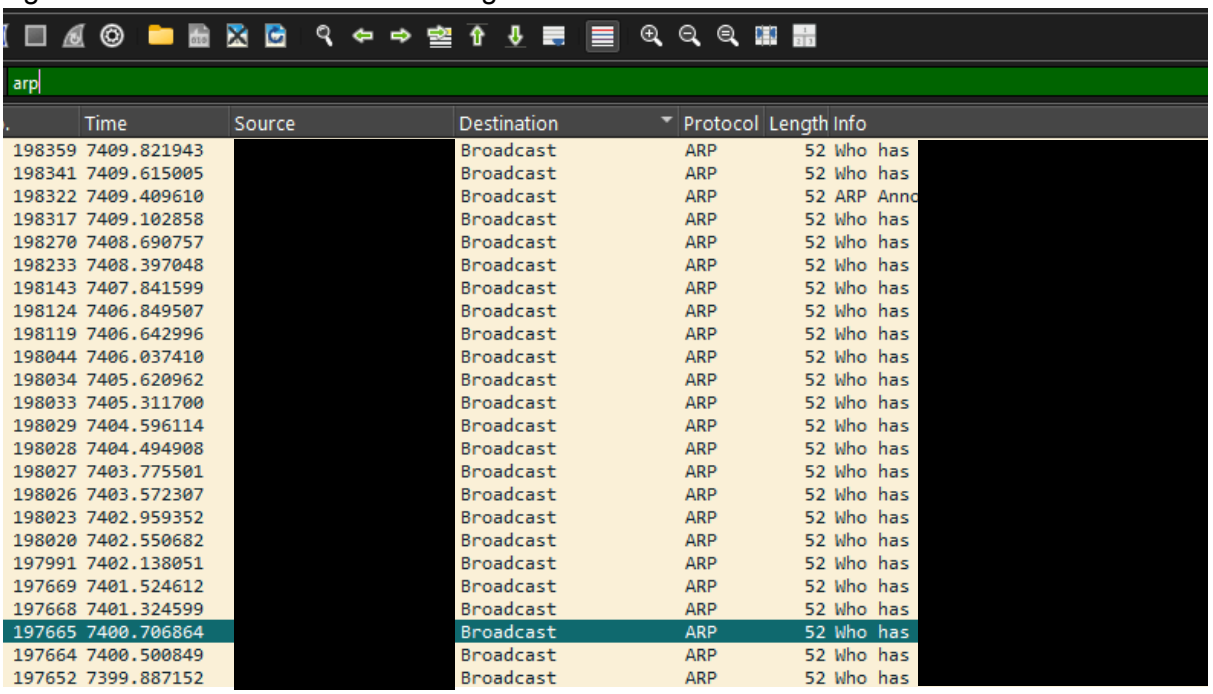
Wireshark may be the leading network analyzer today, but it is not the only one available. Other network analyzers include applications like Zeek and tcpdump as well as many others. These offer different interfaces, some are command line only, some are more user friendly, but none of them come to the prominence of Wireshark.

Another consideration to make is the ethical responsibility that comes with capturing packets. Since Wireshark can be used to view sensitive data flowing across the network, it can be seen as ethically immoral and legally irresponsible too as capturing packets without authorization can be illegal. Since passwords and other sensitive data is stored in cleartext when using the HTTP protocol, it can be risky to capture packets without authorization. As with all cybersecurity tools, it must be used within the approved scope allowed by the corporation.

In conclusion, Wireshark has become an indispensable tool for not just hackers and security professionals, but also students, and anyone who is interested in their network. It helps detect network intrusions, malicious behavior, performance issues, and it can be used as a training tool for students. As networks continue to become more complex, Wireshark will continue to be an essential tool for everyone in the cyber industry.

Appendices:

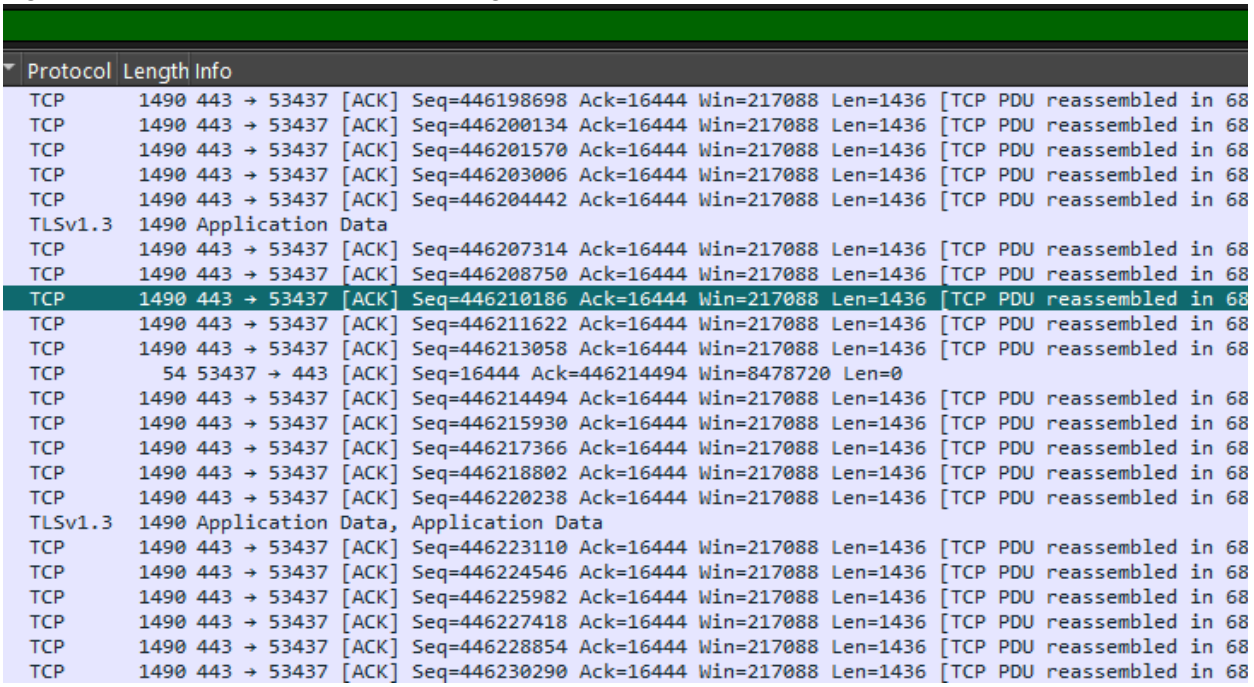
Figure 1: A screenshot of ARP filtering in Wireshark.



arp

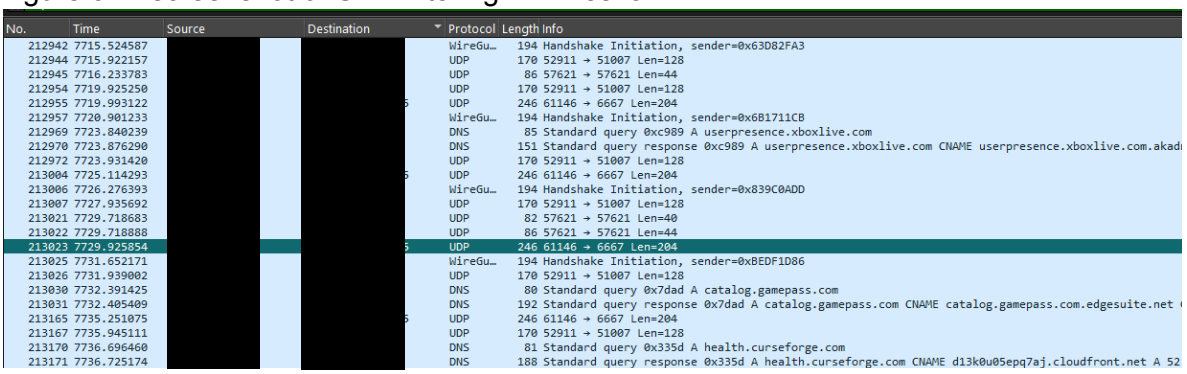
	Time	Source	Destination	Protocol	Length	Info
198359	7409.821943	[redacted]	Broadcast	ARP	52	Who has
198341	7409.615005		Broadcast	ARP	52	Who has
198322	7409.409610		Broadcast	ARP	52	ARP Anno
198317	7409.102858		Broadcast	ARP	52	Who has
198270	7408.690757		Broadcast	ARP	52	Who has
198233	7408.397048		Broadcast	ARP	52	Who has
198143	7407.841599		Broadcast	ARP	52	Who has
198124	7406.849507		Broadcast	ARP	52	Who has
198119	7406.642996		Broadcast	ARP	52	Who has
198044	7406.037410		Broadcast	ARP	52	Who has
198034	7405.620962		Broadcast	ARP	52	Who has
198033	7405.311700		Broadcast	ARP	52	Who has
198029	7404.596114		Broadcast	ARP	52	Who has
198028	7404.494908		Broadcast	ARP	52	Who has
198027	7403.775501		Broadcast	ARP	52	Who has
198026	7403.572307		Broadcast	ARP	52	Who has
198023	7402.959352		Broadcast	ARP	52	Who has
198020	7402.550682		Broadcast	ARP	52	Who has
197991	7402.138051		Broadcast	ARP	52	Who has
197669	7401.524612		Broadcast	ARP	52	Who has
197668	7401.324599		Broadcast	ARP	52	Who has
197665	7400.706864		Broadcast	ARP	52	Who has
197664	7400.500849		Broadcast	ARP	52	Who has
197652	7399.887152		Broadcast	ARP	52	Who has

Figure 2: A screenshot of TCP filtering in Wireshark.



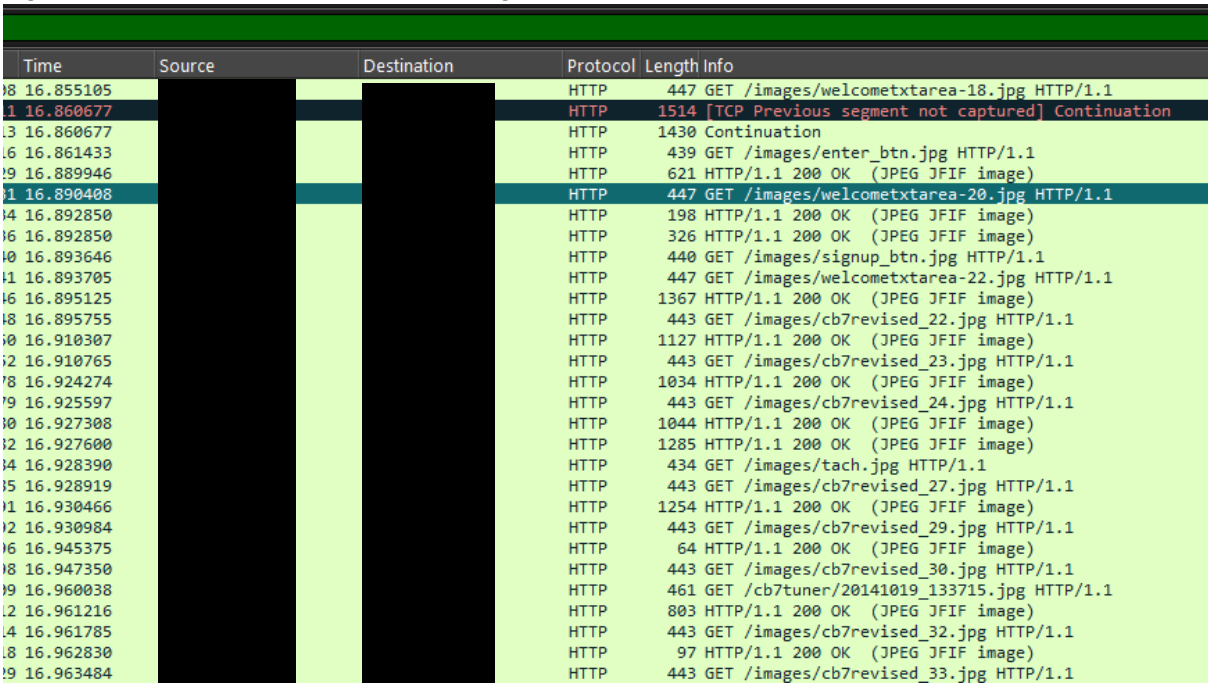
Protocol	Length	Info
TCP	1490	443 → 53437 [ACK] Seq=446198698 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446200134 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446201570 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446203006 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446204442 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TLSv1.3	1490	Application Data
TCP	1490	443 → 53437 [ACK] Seq=446207314 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446208750 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446210186 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446211622 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446213058 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	54	53437 → 443 [ACK] Seq=16444 Ack=446214494 Win=8478720 Len=0
TCP	1490	443 → 53437 [ACK] Seq=446214494 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446215930 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446217366 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446218802 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446220238 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TLSv1.3	1490	Application Data, Application Data
TCP	1490	443 → 53437 [ACK] Seq=446223110 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446224546 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446225982 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446227418 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446228854 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68
TCP	1490	443 → 53437 [ACK] Seq=446230290 Ack=16444 Win=217088 Len=1436 [TCP PDU reassembled in 68

Figure 3: A screenshot of UDP filtering in Wireshark.



No.	Time	Source	Destination	Protocol	Length	Info
212942	7715.524587	[redacted]	[redacted]	WireGu…	194	Handshake Initiation, sender=0x63D82FA3
212944	7715.922157			UDP	170	52911 → 51007 Len=128
212945	7716.233783			UDP	86	57621 → 57621 Len=44
212954	7719.925250			UDP	170	52911 → 51007 Len=128
212955	7719.993122			UDP	246	61146 → 6667 Len=204
212957	7720.901233			WireGu…	194	Handshake Initiation, sender=0x6B1711CB
212969	7723.840239			DNS	85	Standard query 0xc989 A userpresence.xboxlive.com
212970	7723.876290			DNS	151	Standard query response 0xc989 A userpresence.xboxlive.com CNAME userpresence.xboxlive.com.akad
212972	7723.931420			UDP	170	52911 → 51007 Len=128
213004	7725.114293			UDP	246	61146 → 6667 Len=204
213006	7726.276393			WireGu…	194	Handshake Initiation, sender=0x839C0ADD
213007	7727.935692			UDP	170	52911 → 51007 Len=128
213021	7729.718683			UDP	82	57621 → 57621 Len=40
213022	7729.718888			UDP	86	57621 → 57621 Len=44
213023	7729.925854			UDP	246	61146 → 6667 Len=204
213025	7731.652171			WireGu…	194	Handshake Initiation, sender=0xBEDF1D86
213026	7731.939002			UDP	170	52911 → 51007 Len=128
213030	7732.391425			DNS	80	Standard query 0x7dad A catalog.gamepass.com
213031	7732.405409			DNS	192	Standard query response 0x7dad A catalog.gamepass.com CNAME catalog.gamepass.com.edgesuite.net
213165	7735.251075			UDP	246	61146 → 6667 Len=204
213167	7735.945111			UDP	170	52911 → 51007 Len=128
213170	7736.696460			DNS	81	Standard query 0x335d A health.curseforge.com
213171	7736.725174			DNS	188	Standard query response 0x335d A health.curseforge.com CNAME d13k0u05epq7aj.cloudfront.net A 52

Figure 4: A screenshot of HTTP filtering in Wireshark.



Time	Source	Destination	Protocol	Length	Info
8 16.855105	[redacted]	[redacted]	HTTP	447	GET /images/welcometxtarea-18.jpg HTTP/1.1
1 16.860677			HTTP	1514	[TCP Previous segment not captured] Continuation
3 16.860677			HTTP	1430	Continuation
6 16.861433			HTTP	439	GET /images/enter_btn.jpg HTTP/1.1
9 16.889946			HTTP	621	HTTP/1.1 200 OK (JPEG JFIF image)
1 16.890408			HTTP	447	GET /images/welcometxtarea-20.jpg HTTP/1.1
4 16.892850			HTTP	198	HTTP/1.1 200 OK (JPEG JFIF image)
6 16.892850			HTTP	326	HTTP/1.1 200 OK (JPEG JFIF image)
0 16.893646			HTTP	440	GET /images/signup_btn.jpg HTTP/1.1
1 16.893705			HTTP	447	GET /images/welcometxtarea-22.jpg HTTP/1.1
6 16.895125			HTTP	1367	HTTP/1.1 200 OK (JPEG JFIF image)
8 16.895755			HTTP	443	GET /images/cb7revised_22.jpg HTTP/1.1
0 16.910307			HTTP	1127	HTTP/1.1 200 OK (JPEG JFIF image)
2 16.910765			HTTP	443	GET /images/cb7revised_23.jpg HTTP/1.1
8 16.924274			HTTP	1034	HTTP/1.1 200 OK (JPEG JFIF image)
9 16.925597			HTTP	443	GET /images/cb7revised_24.jpg HTTP/1.1
0 16.927308			HTTP	1044	HTTP/1.1 200 OK (JPEG JFIF image)
2 16.927600			HTTP	1285	HTTP/1.1 200 OK (JPEG JFIF image)
4 16.928390			HTTP	434	GET /images/tach.jpg HTTP/1.1
5 16.928919			HTTP	443	GET /images/cb7revised_27.jpg HTTP/1.1
1 16.930466			HTTP	1254	HTTP/1.1 200 OK (JPEG JFIF image)
2 16.930984			HTTP	443	GET /images/cb7revised_29.jpg HTTP/1.1
6 16.945375			HTTP	64	HTTP/1.1 200 OK (JPEG JFIF image)
8 16.947350			HTTP	443	GET /images/cb7revised_30.jpg HTTP/1.1
9 16.960038			HTTP	461	GET /cb7tuner/20141019_133715.jpg HTTP/1.1
2 16.961216			HTTP	803	HTTP/1.1 200 OK (JPEG JFIF image)
4 16.961785			HTTP	443	GET /images/cb7revised_32.jpg HTTP/1.1
8 16.962830			HTTP	97	HTTP/1.1 200 OK (JPEG JFIF image)
9 16.963484			HTTP	443	GET /images/cb7revised_33.jpg HTTP/1.1

Figure 5: A screenshot of HTTPS filtering in Wireshark.

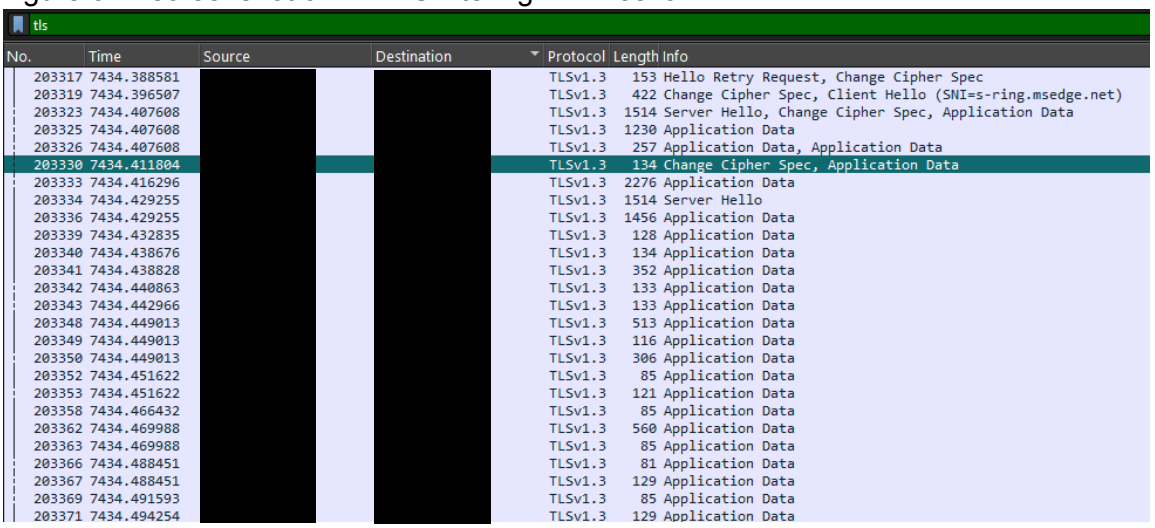


Figure 6: A screenshot of ICMP filtering in Wireshark.

Figure 7: A screenshot of DNS filtering in Wireshark.

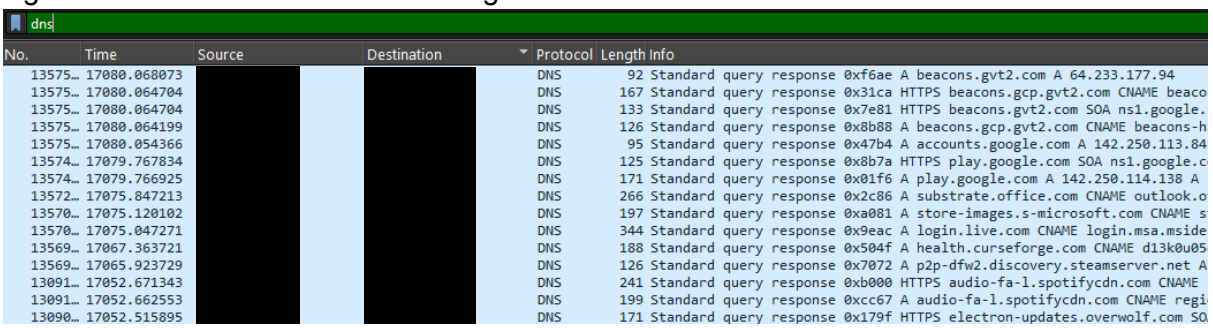
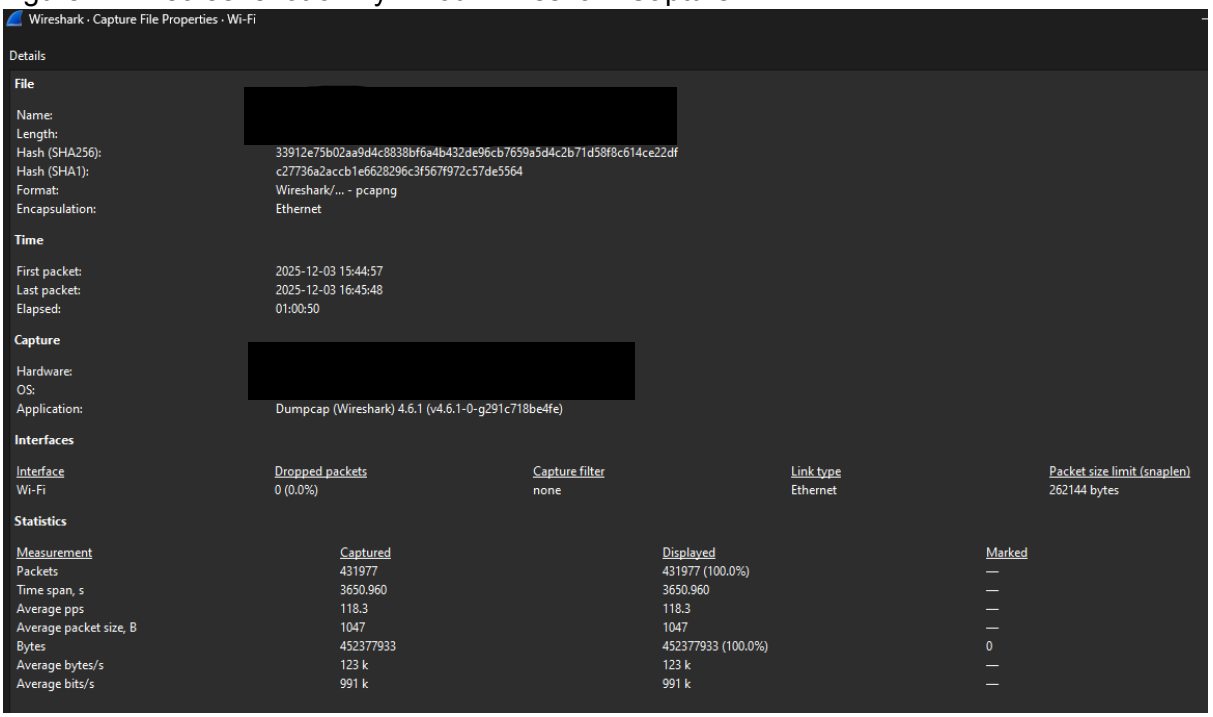


Figure 11: A screenshot of my 1 hour Wireshark Capture.



Wireshark · Capture File Properties · Wi-Fi

Details

File

Name:	[redacted]
Length:	
Hash (SHA256):	33912e75b02aa9d4c8838bf6a4b432de96cb7659a5d4c2b71d58f8c614ce22df
Hash (SHA1):	c27736a2accb1e6628296c3f567f972c57de5564
Format:	Wireshark/... - pcapng
Encapsulation:	Ethernet

Time

First packet:	2025-12-03 15:44:57
Last packet:	2025-12-03 16:45:48
Elapsed:	01:00:50

Capture

Hardware:	[redacted]
OS:	
Application:	Dumpcap (Wireshark) 4.6.1 (v4.6.1-0-g291c718be4fe)

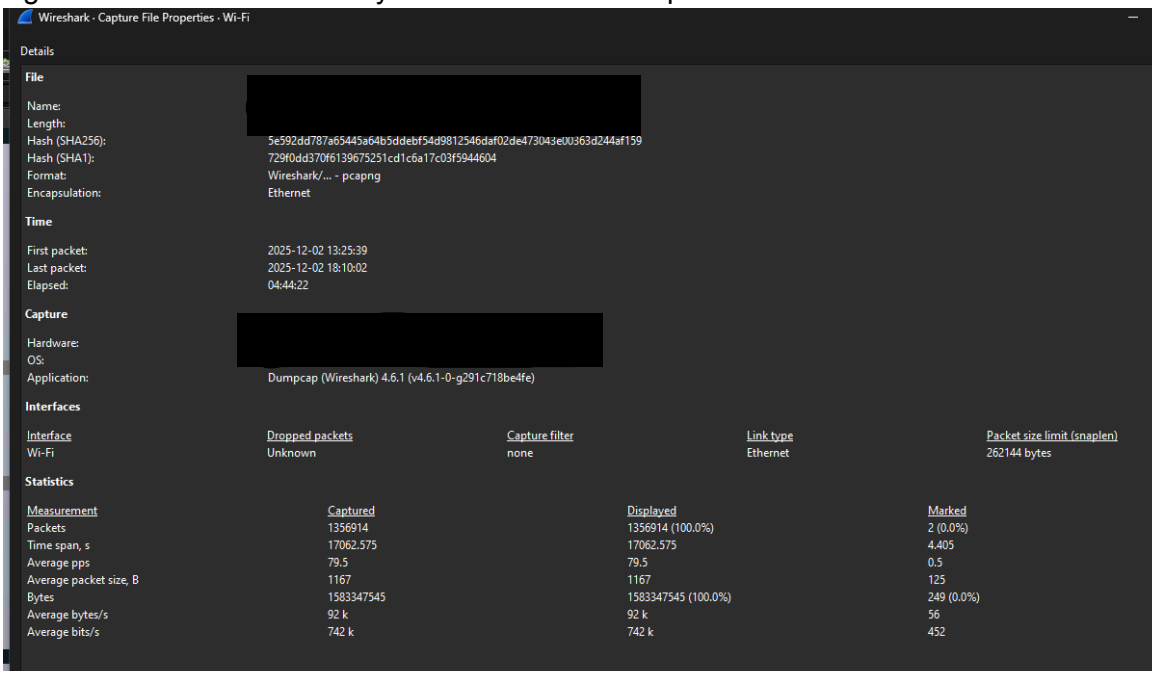
Interfaces

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
Wi-Fi	0 (0.0%)	none	Ethernet	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	431977	431977 (100.0%)	—
Time span, s	3650.960	3650.960	—
Average pps	118.3	118.3	—
Average packet size, B	1047	1047	—
Bytes	452377933	452377933 (100.0%)	0
Average bytes/s	123 k	123 k	—
Average bits/s	991 k	991 k	—

Figure 12: A screenshot of my 4 hour Wireshark Capture.



Wireshark · Capture File Properties · Wi-Fi

Details

File

Name:	[redacted]
Length:	
Hash (SHA256):	5e592dd787a65445a64b5ddebf54d9812546daf02de473043e00363d244af159
Hash (SHA1):	729f0dd370f6139675251cd1c6a17c03f5944604
Format:	Wireshark/... - pcapng
Encapsulation:	Ethernet

Time

First packet:	2025-12-02 13:25:39
Last packet:	2025-12-02 18:10:02
Elapsed:	04:44:22

Capture

Hardware:	[redacted]
OS:	
Application:	Dumpcap (Wireshark) 4.6.1 (v4.6.1-0-g291c718be4fe)

Interfaces

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
Wi-Fi	Unknown	none	Ethernet	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	1356914	1356914 (100.0%)	2 (0.0%)
Time span, s	17062.575	17062.575	4.405
Average pps	79.5	79.5	0.5
Average packet size, B	1167	1167	125
Bytes	1583347545	1583347545 (100.0%)	249 (0.0%)
Average bytes/s	92 k	92 k	56
Average bits/s	742 k	742 k	452

Bibliography:

Banerjee, U., Vashishtha, A., & Saxena, M. (2010). Evaluation of the Capabilities of WireShark as a tool for Intrusion Detection. International Journal of computer applications, 6(7), 1-5.

Iqbal, H., & Naaz, S. (2019). Wireshark as a tool for detection of various LAN attacks. Int. J. Comput. Sci. Eng, 7(5), 833-837.