Wireshark Technical Project Guide + Report

Beginner/Intermediate Cybersecurity Project

Conducted by: Burton Li

January 2024

1. Executive Summary

As a dedicated full-time cybersecurity student, it is crucial to establish a strong foundation across the breadth of this extensive field. The Wireshark technical project focused on enhancing understanding and proficiency in using Wireshark's basic functionalities. The project involved capturing and analyzing various types of network traffic, including RADIUS, HTTP, DNS, Telnet, SSH, and HTTPS. Specific tasks included decrypting encrypted passwords, analyzing authentication methods, and comparing different protocols. Through this, not only do we learn how to utilize a widely accepted tool across the field, but we also lay the groundwork for exploring more about the ever-changing world of cybersecurity.

In addition to honing practical skills in network traffic analysis, the Wireshark technical project also provided valuable insights into the intricate world of cybersecurity threats. By dissecting and scrutinizing different types of network traffic, the project facilitated a deeper understanding of potential vulnerabilities and attack vectors. This hands-on experience allowed for the identification of anomalous patterns and the development of effective strategies to mitigate potential risks. The exploration of authentication methods was a pivotal component of the project, offering a practical examination of how different protocols handle user verification. Analyzing authentication processes for protocols like RADIUS, Telnet, and SSH not only enhanced proficiency in Wireshark but also deepened comprehension of security mechanisms implemented across diverse network environments.

2. Project Overview

2.1 - Objectives

- Use Wireshark's basic functionalities to capture and analyze network traffic.
- Generate and capture RADIUS traffic, understand its architecture, and decrypt passwords.
- Generate, capture, and analyze HTTPS traffic - decrypting the captured data including basic and form-based authentication.
- Follow a TCP Stream
- Initiate and capture Telnet sessions, and analyze the traffic from a security perspective.
- Open SSH sessions, capture and compare SSH traffic with Telnet, and analyze host, packet, and protocol statistics.
- Extracting NTLM Hashes with Hashcat

2.2 - Scope

Hands-on tasks covering Wireshark functionalities and specific protocols.

Emphasis on security analysis and understanding encrypted protocols.

2.3 - Environment

The entirety of this project was performed either through a Windows virtual machine provided by Drexel University, or my desktop. The virtual machine ran on Windows 10 and contained Wireshark version 3.2.6. My desktop runs on Windows 11 and contains Wireshark version 4.2.0. All necessary permissions were granted before the start of this project.

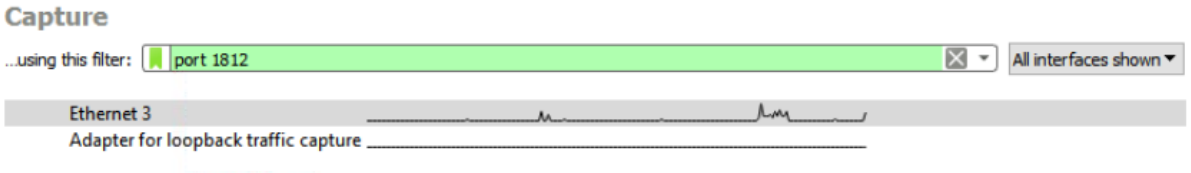
3. Methodology

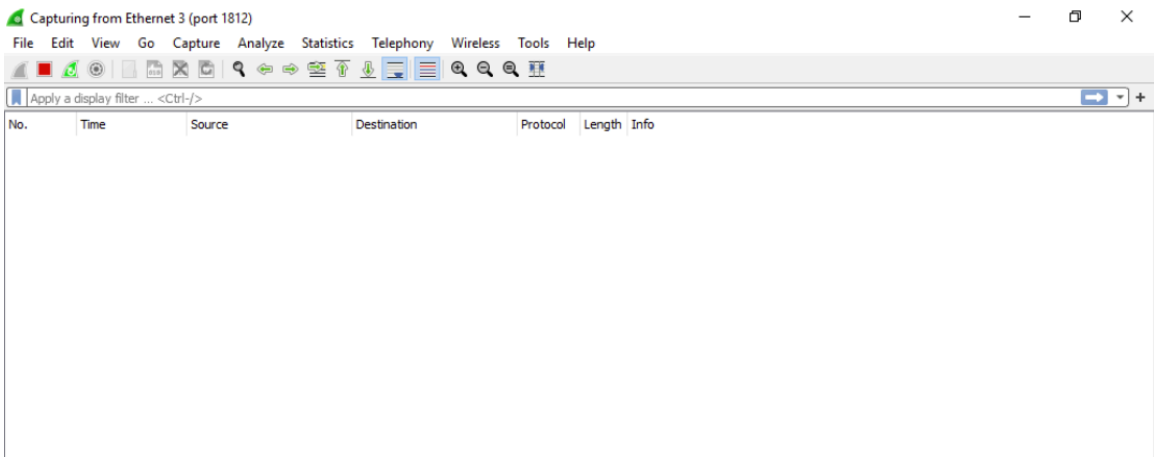
3.1 - Generating and capturing RADIUS traffic

- Step 1 - This page we are viewing is the GUI of the RADIUS Server from "idblender.com". Input a username and password and click Submit. The RADIUS server stores the usernames and passwords and authenticates them when the request comes.

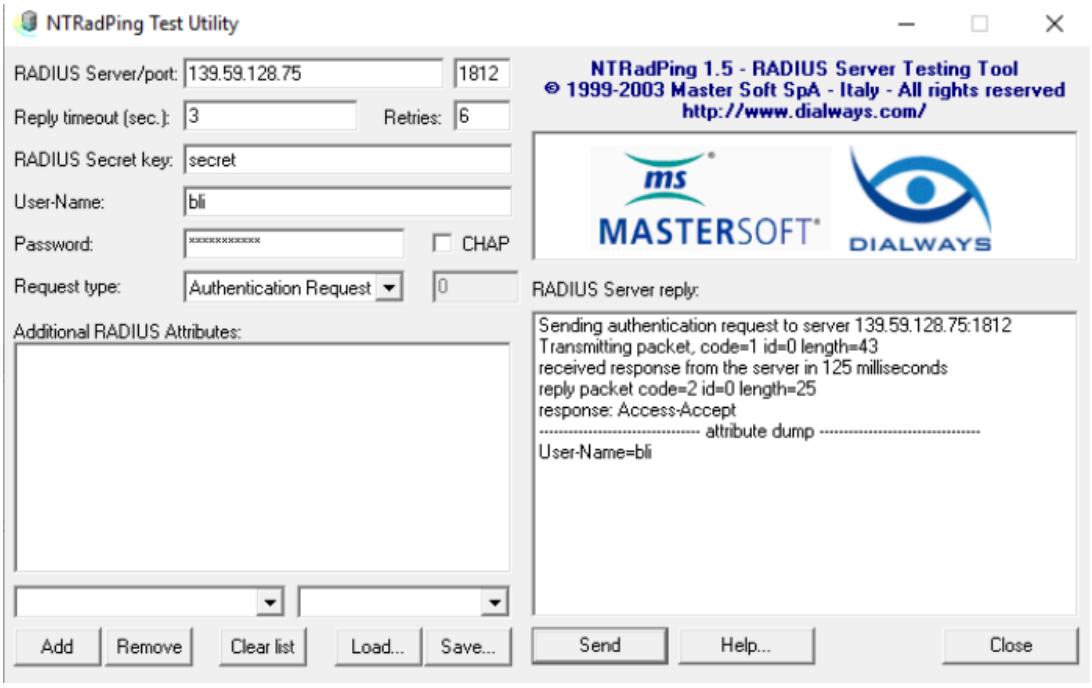
Take note of the IP "139.59.128.75" and authorization port 1812

- Step 2 - Open Wireshark and click on your network, type "`port 1812`" within the Capture filter, and hit Enter. This should automatically start a capture session for all traffic happening on port 1812. The capture screen should be blank.

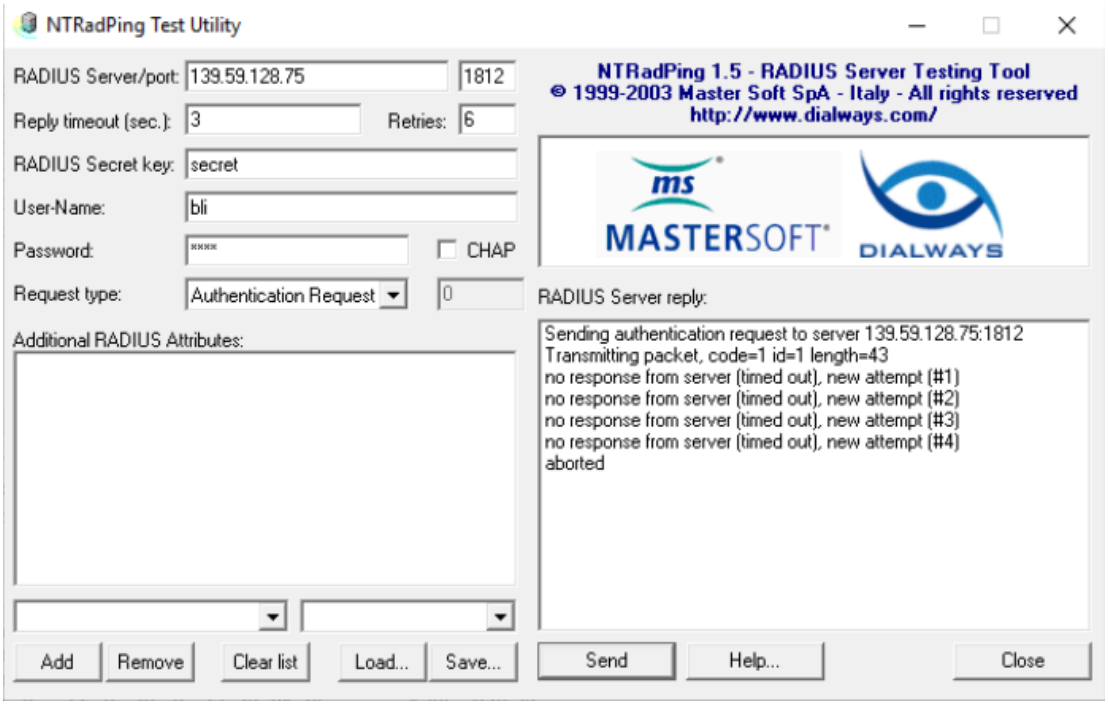




- Step 3 - To generate traffic, we will have to use a desktop application called NTRadPing. This will act as the client and will be used to send an authentication request to the public server. We will enter the IP and port as referenced from the first screenshot, as well as the credentials that we created. Everything else should be left default. Hit Send. As a test, we will also enter an incorrect password to analyze. After hitting send, abort after a few instances. For this example, I used "`1234`".

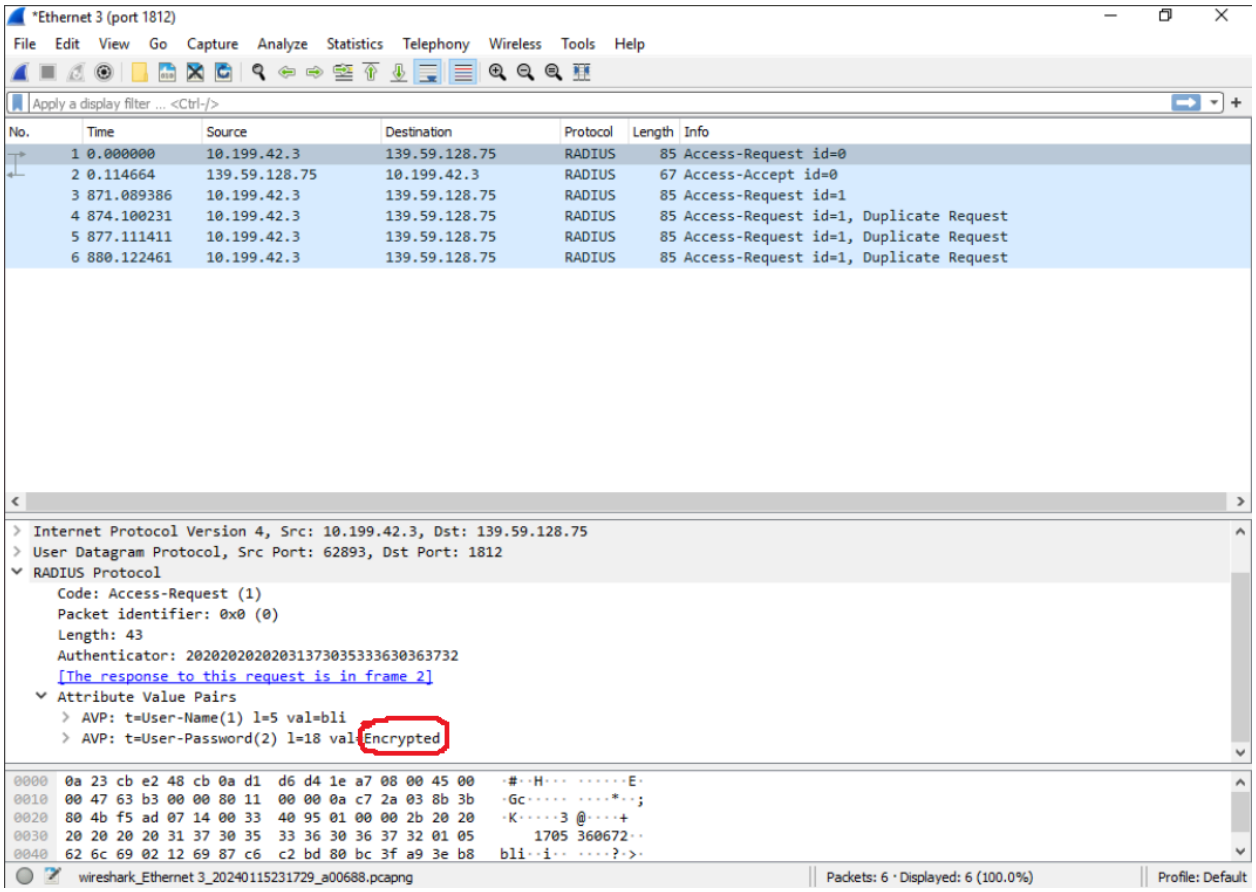


Take note of the "RADIUS Secret key".



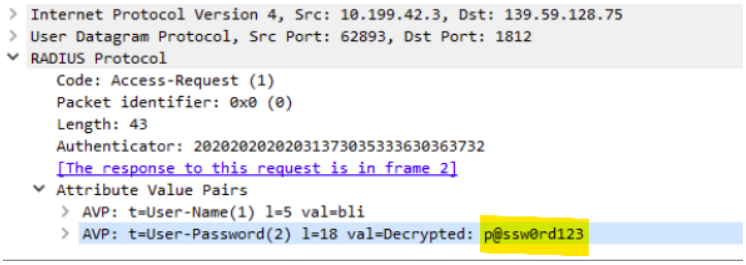
For this example, I used "1234" as my password.

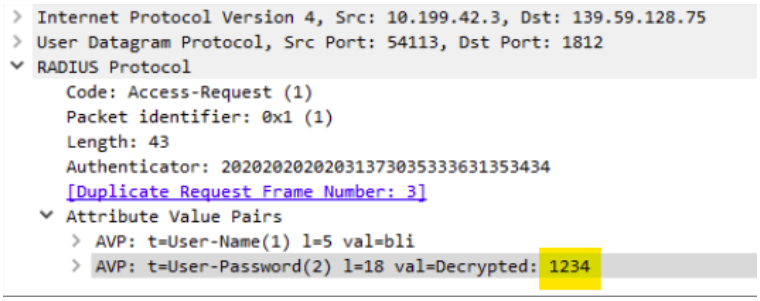
- Step 4 - After aborting the last ping, return to Wireshark and stop the capture. Observe that the RADIUS packets are unencrypted and not secure. It could be easily monitored by a third party that authentication is happening since the access request is visible. Click on the first packet and expand the “RADIUS Protocol” window. Notice that the password is Encrypted.



- Step 5 - We will now use Wireshark to decrypt these passwords. Remember that field called Secret Key? Also known as Shared secret, RADIUS uses this value and the MD5 hashing algorithm to hide the passwords between the server and the client, which is itself a weak protection of the user's Credentials. On the top left of the capture screen, select **Edit** and then **Preferences**. On the left side of the pop-up, select **Protocols** and look for

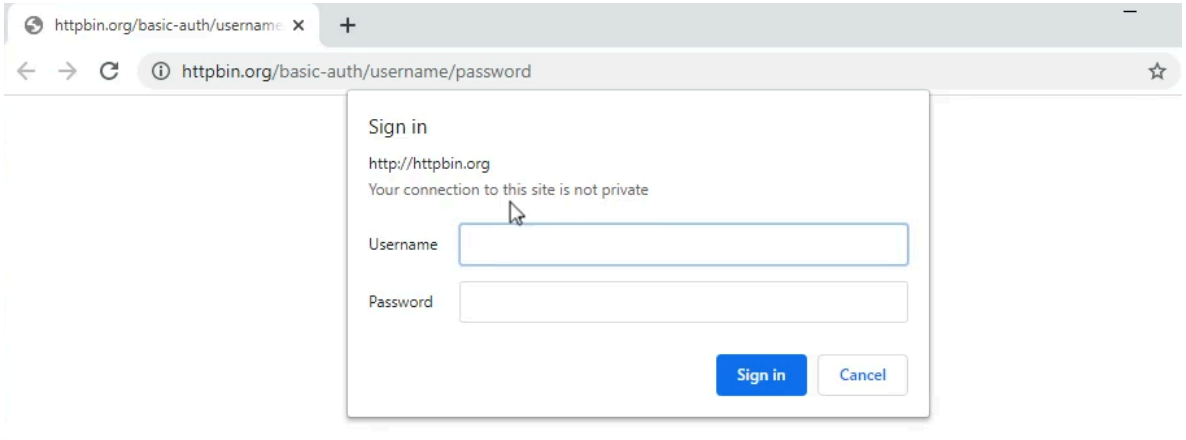
RADIUS. Within **Shared Secret**, type in the secret key we saw earlier, which is "`secret`". Click OK and you should now be able to see both passwords used.





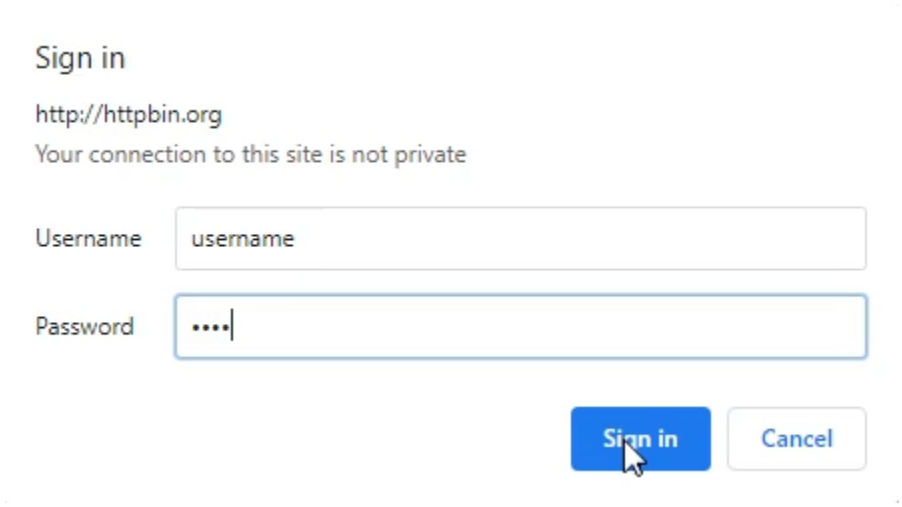
3.2 - Analyzing an HTTP Basic Authentication

- Step 1 - We will utilize a simple HTTP challenge and response mechanism from [httpbin.org](), where the server will request authentication from a client. The username and password can be seen in the URL - do not enter any credentials yet.

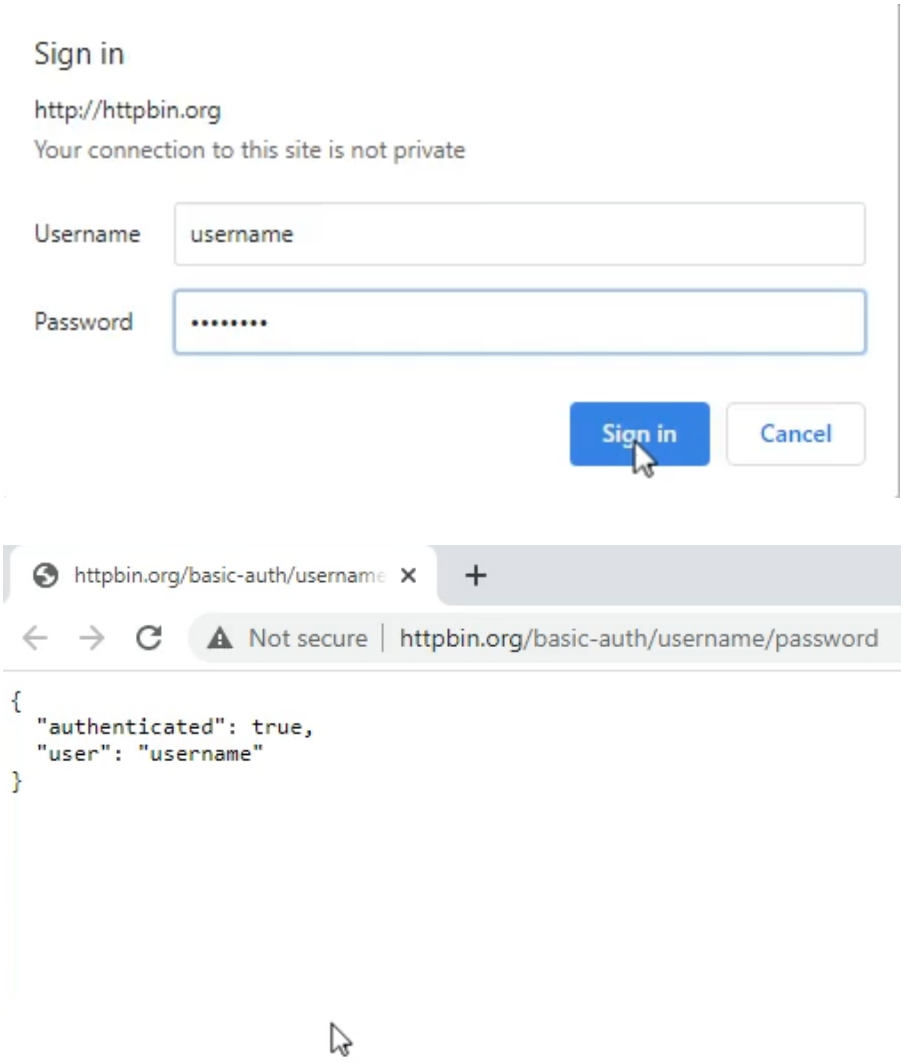


- Step 2 - Since HTTP operates on Port 80, we will type "`port 80`" into the capture filter and double-click on our network. This should begin capture.

- Step 3 - Going back to the web browser, we will perform our experiment through two authentication requests, one with correct and one with incorrect sign-on. First, enter "`username`" as username and then "`1234`" for the password. Next, when prompted again, enter the correct credentials as seen in the URL. This process should be completed when you see the authenticated page.



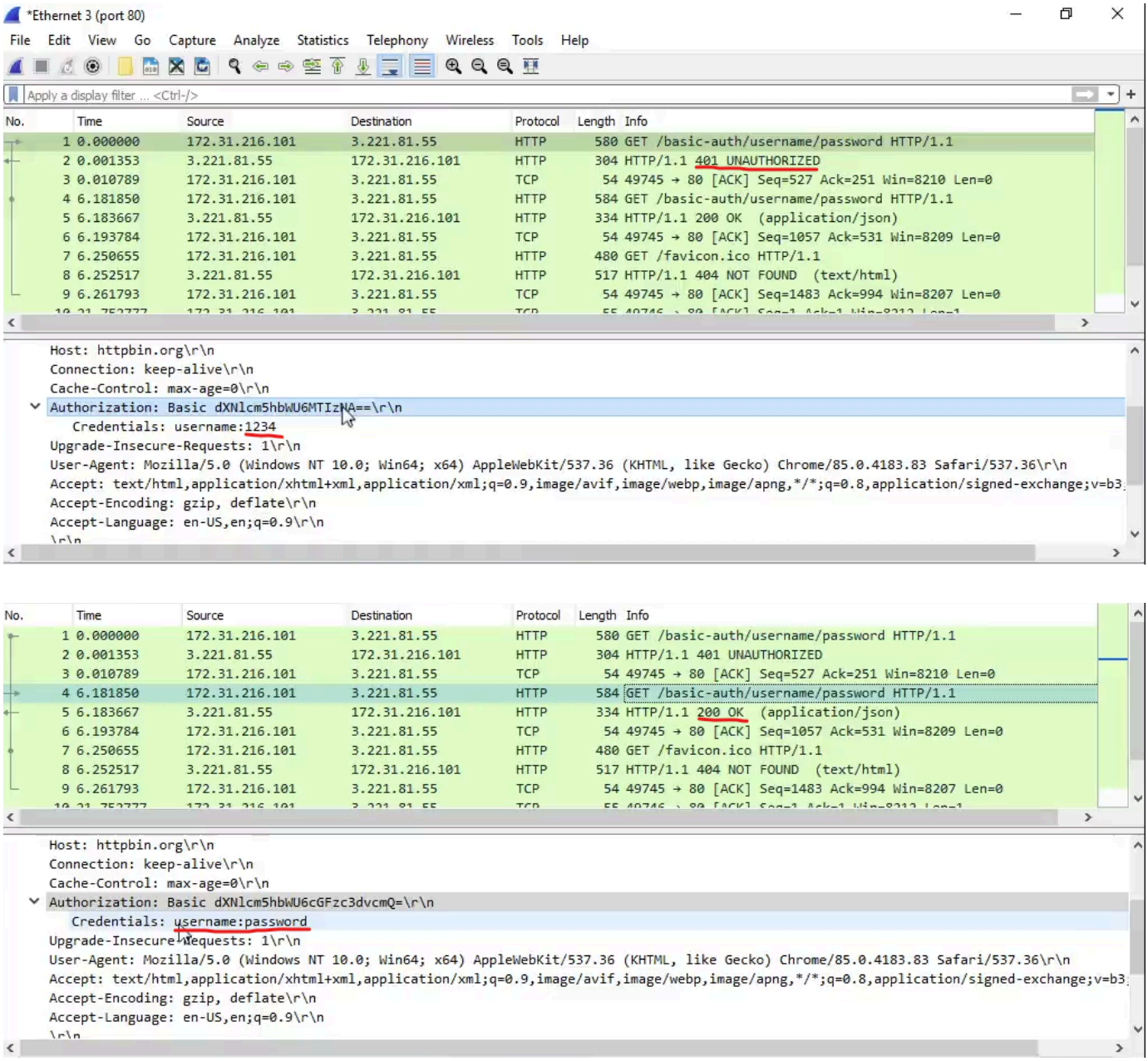
Notice how the site connection is not private



- Step 4 - Go back to Wireshark and stop the capture. Upon first glance, we can observe that everything is unencrypted, making this an insecure form of authentication. The first authentication request that was sent with an incorrect password has received a "401 UNAUTHORIZED" response whereas the second request that was sent with the correct password has received a code "200 OK".

- Step 5 - Click on the first GET packet and expand **Hyper Text Transfer Protocol**, then expand **Authorization**. We can observe the base 64 encoding of the incorrect username

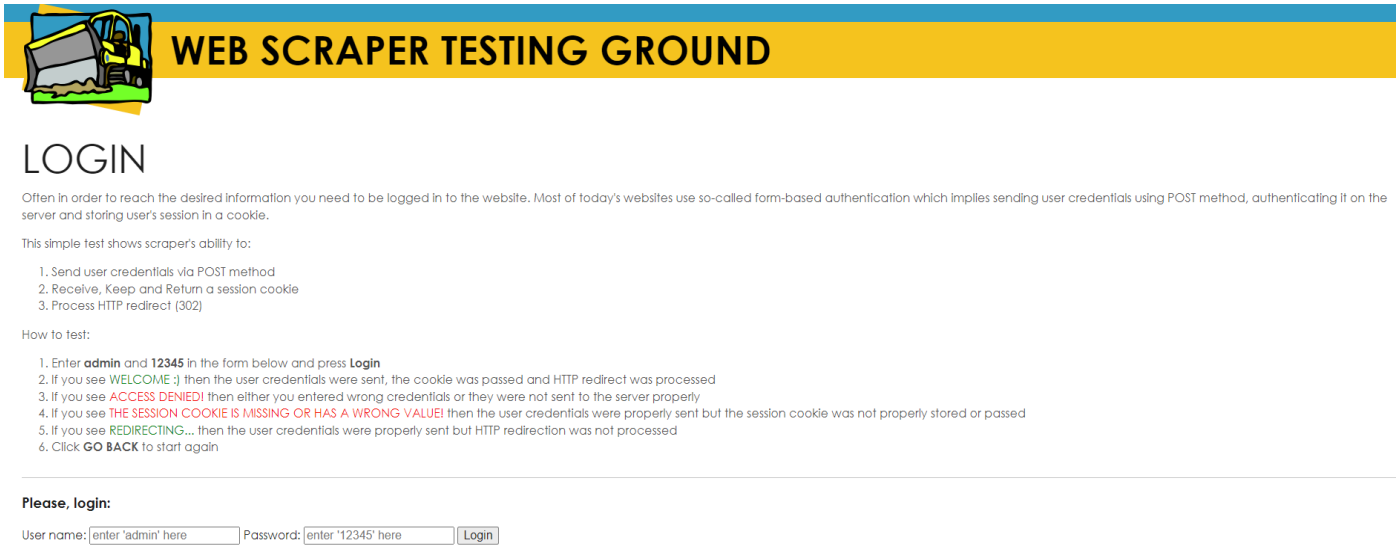
and password, which provides no encryption. Repeat this step for the second GET packet and you will see similar results.



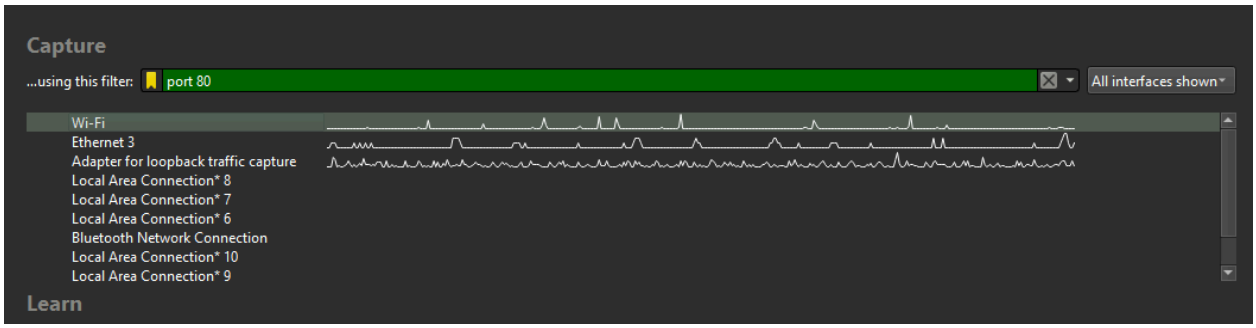
3.3 - HTTP Form-Based Authentication and DNS

- Step 1 - For this section, we will have to utilize a web scraper testing ground. Do not enter any credentials yet. This will be used to capture an HTTP form-based

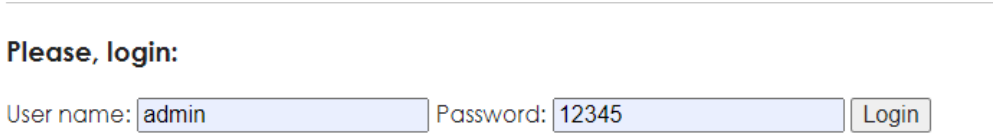
authentication, which was created to mitigate the downsides of the HTTP basic authentication.



- Step 2 - Open Wireshark. Since we are working with HTTP, we will type "`port 80`" in the capture filter and select our network. This should begin the capture.

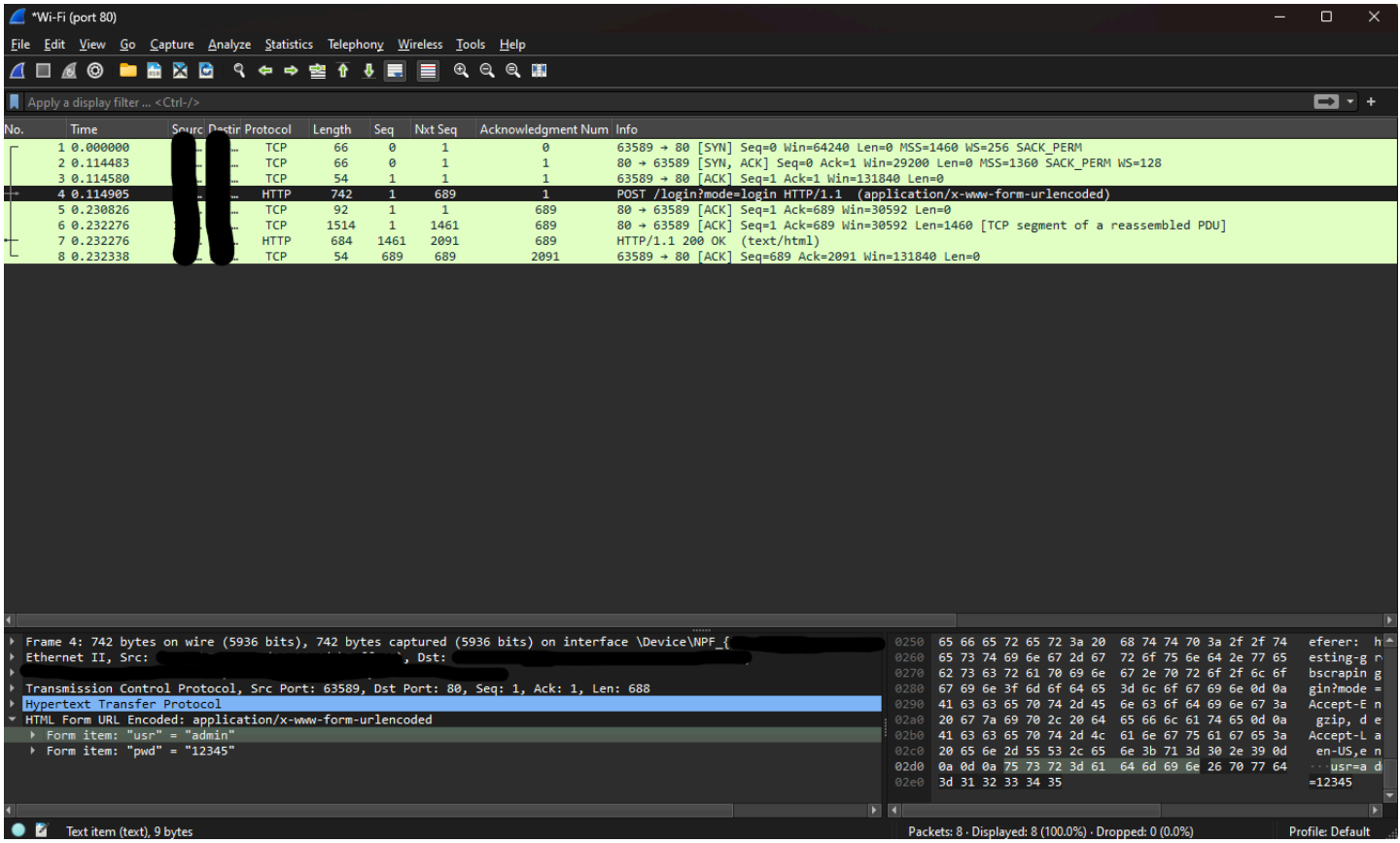


- Step 3 - Go back to the webpage and enter the credentials.

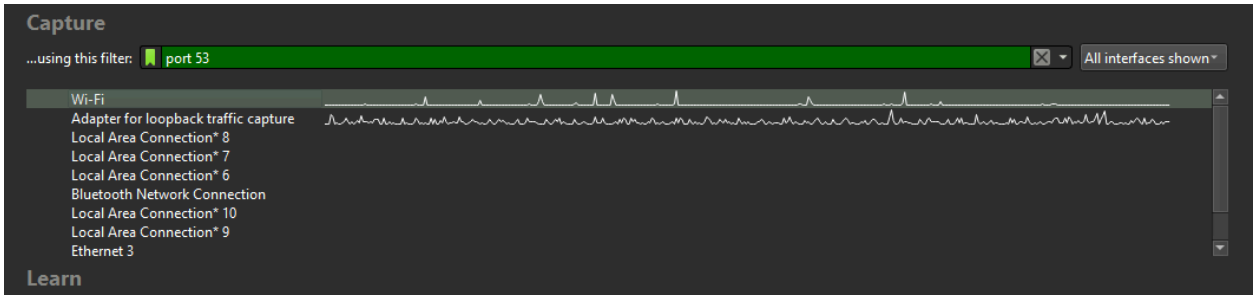


- Step 4 - Once you are logged in, go back to Wireshark and stop the capture. In the previous experiment, we were able to find the credentials in the basic authentication

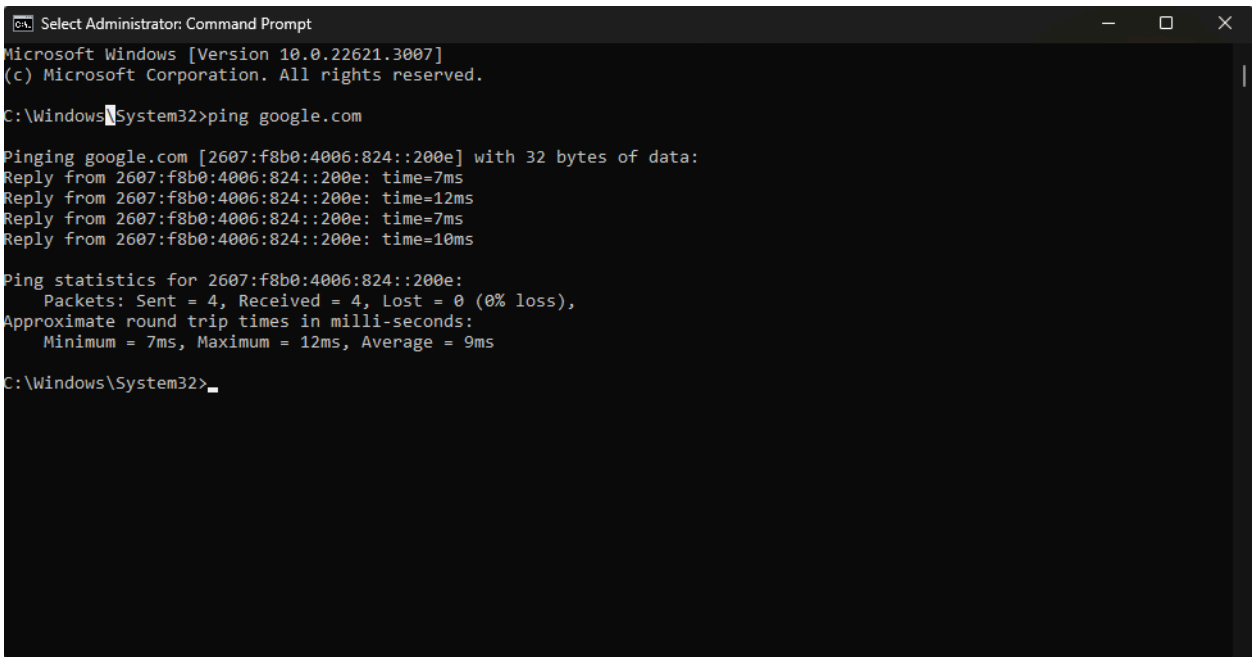
inside the HTTP. This time we can find them by expanding the **Hypertext Transfer Protocol**, without the base 64 encoding.



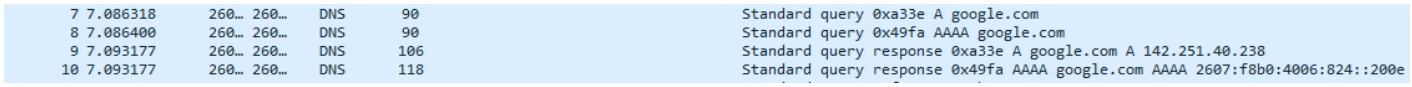
- Step 5 - Close out of the Wireshark capture and return to the front screen - we will now capture some DNS traffic. Since the standard port for DNS is 53, we will enter "`port 53`" into the capture filter. Begin capture



- Step 6 - We will now generate some DNS traffic. Open up the Window's command prompt and ping a website. For this example, we used Google. The command would be `ping google.com`

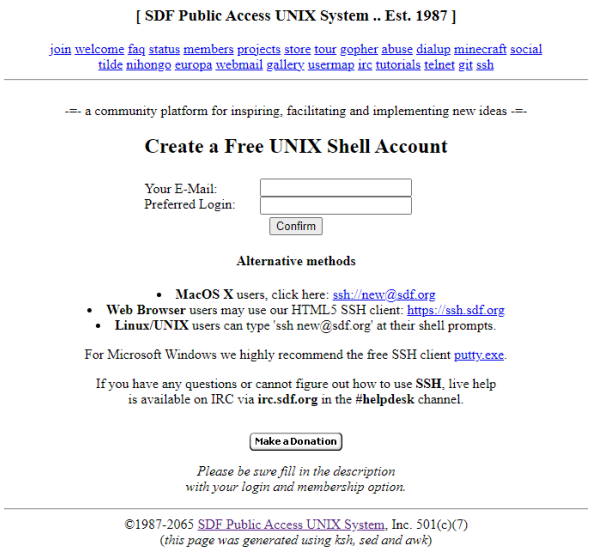


- Step 7 - Once the ping finishes, stop the Wireshark capture. Observing the traffic, we can see that the DNS is not encrypted whatsoever as the application data can be seen in clear text. We are even able to see which server our VM needs to know the IP for as well as the response.



3.4 - Initiate, Capture, and Analyze Telnet Sessions

- Step 1 - Create a free UNIX shell account through sdf.org. You should receive your credentials in the same email that was used to create the account.



[SDF Public Access UNIX System .. Est. 1987]

join welcome faq status members projects store tour gopher abuse dialup minecraft social
tilde nihongo europa webmail gallery usermap irc tutorials telnet git ssh

-=- a community platform for inspiring, facilitating and implementing new ideas -=-

Create a Free UNIX Shell Account

Your E-Mail:
Preferred Login:
Confirm

Alternative methods

- **MacOS X** users, click here: ssh://new@sdf.org
- **Web Browser** users may use our HTML5 SSH client: https://ssh.sdf.org
- **Linux/UNIX** users can type 'ssh new@sdf.org' at their shell prompts.

For Microsoft Windows we highly recommend the free SSH client putty.exe.

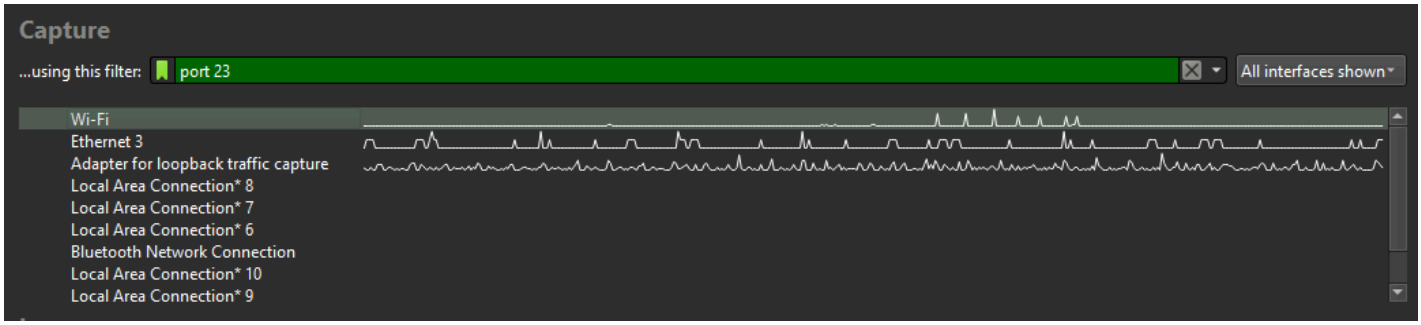
If you have any questions or cannot figure out how to use **SSH**, live help is available on IRC via **irc.sdf.org** in the **#helpdesk** channel.

Make a Donation

Please be sure fill in the description with your login and membership option.

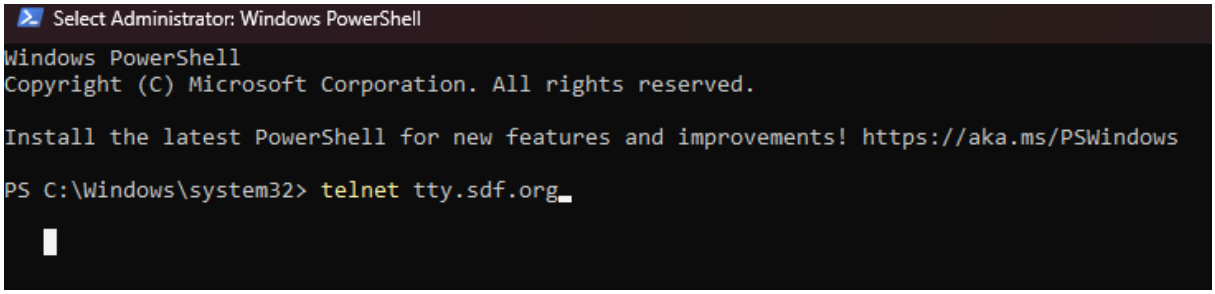
©1987-2065 SDF Public Access UNIX System, Inc. 501(c)(7)
(*this page was generated using ksh, sed and awk*)

- Step 2 - Open Wireshark. Since Telnet operates on port 23, we will enter "`port 23`" into the capture filter. Launch the capture.

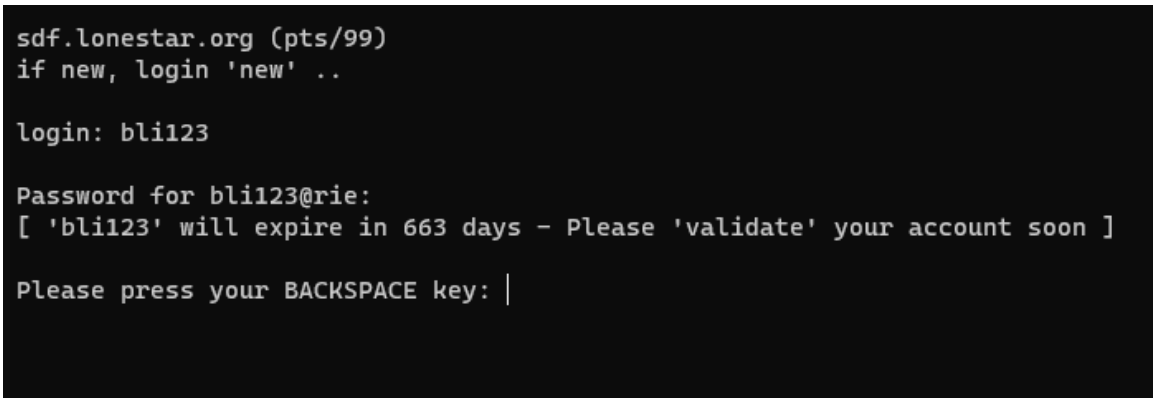


- Step 3 - We will now begin generating Telnet traffic. Open Windows Powershell and enter the command: "`telnet tty.sdf.org`"

  If you receive an error message stating that "telnet" is not recognized, you will have to activate it through ***Control Panel > Programs > Turn Windows features on or off****. Then, check "****Telnet Client****" and save the changes. It may take a few minutes for the changes to take effect.*



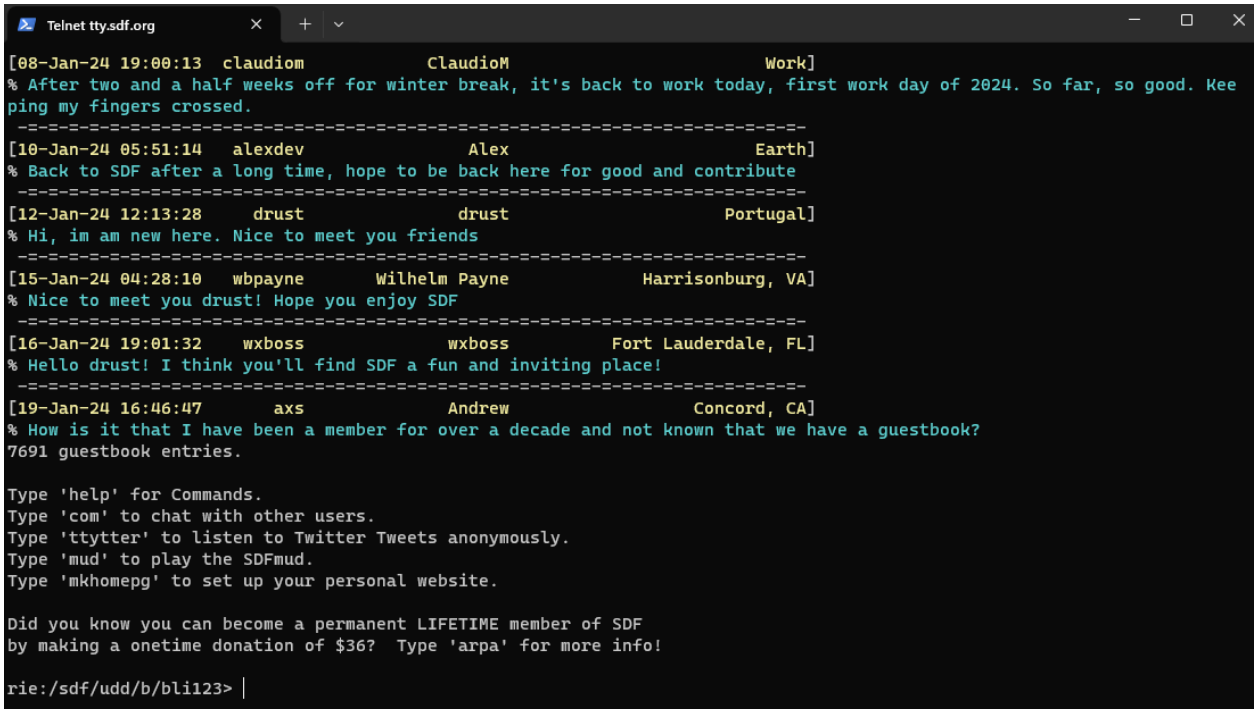
- Step 4 - After running the command, you should now be able to log in using the credentials provided via email. Be careful as the password is case-sensitive. Once logged in, hit backspace as instructed. After pressing backspace, hit Enter to continue.



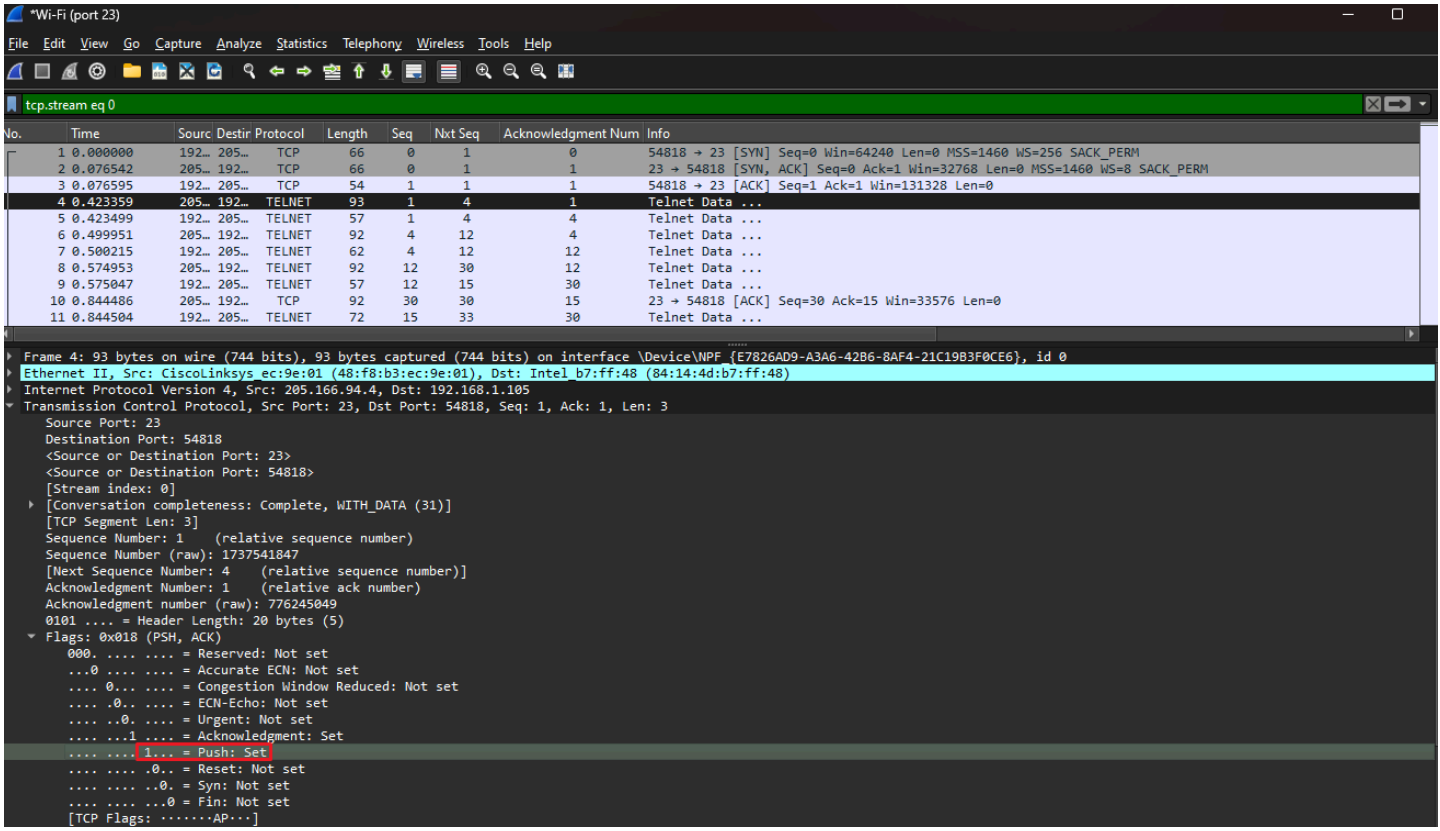
```
======================================================================
SDF host uptime report for Seattle WA, Dallas TX (USA) and Germany (EU)
            Please use 'tty.sdf.org' for general access
======================================================================

SERVER          DAYS+HOUR:MIN       USERS   MACHINE LOAD
----------------------------------------------------------------------
9p             up  34+08:25,   19 users,  load:   0.00,   0.00,   0.00
aNONradio      up 113+22:30,   67 users,  load:   0.03,   0.12,   0.17
faeroes        up 113+22:09,  109  user,  load:   2.58,   2.28,   1.75
iceland        up 113+22:30,   80 users,  load:   0.45,   0.58,   1.41
jitsi          up 113+22:25,  809 users,  load:   0.87,   1.47,   2.12
ma             up   35+7:33,  114 users,  load:   2.56,   2.49,   2.60
matrix         up 113+22:30,  452 users,  load:   1.92,   1.84,   1.83
mc             up  12+08:44,    1 users,  load:   3.40,   3.12,   3.41
miku           up 113+22:26,    3 users,  load:   1.29,   1.46,   1.56
mx             up 113+22:28,  364 users,  load:   8.24,   8.55,   9.55
norge          up 113+22:23,   91 users,  load:   1.96,   1.91,   1.88
otaku          up 113+22:28,   56 users,  load:   3.77,   3.75,   3.86
pixelfed       up 115+17:27, 2070 users,  load:   0.09,   0.14,   0.17
rie            up 113+22:25,   59 users,  load:   0.34,   0.19,   0.28
ryo            up  25+04:16,   49 users,  load:   0.70,   0.93,   1.07
sdf            up  36+08:43,  265 users,  load:   6.12,   5.18,   5.02
sdfeu          up 561+21:59,   24 users,  load:   1.31,   1.28,   1.30
sverige        up 113+22:26,   12  user,  load:   0.83,   1.13,   1.20
unix50         up 113+22:29,   94 users,  load:  41.45,  41.84,  41.93
vps3           up 113+22:21,   51 users,  load:   7.08,   5.56,   3.04
vps9           up  115+8:54,   16 users,  load:   2.00,   2.06,   2.42
                             4805 total
(continue)
```

- Step 5 - After hitting Enter, return to Wireshark and end the capture.

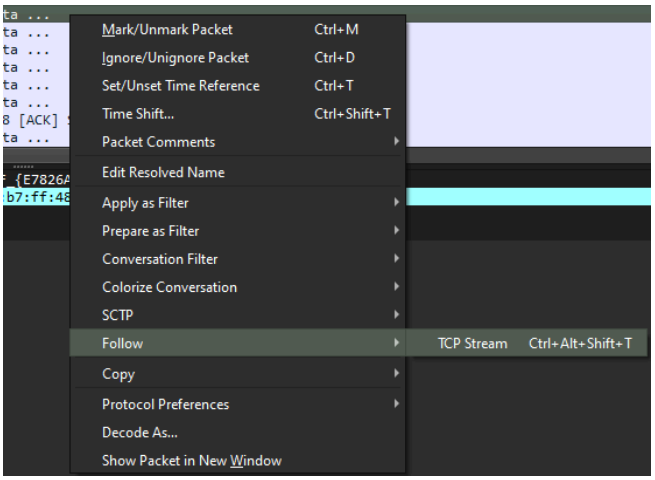


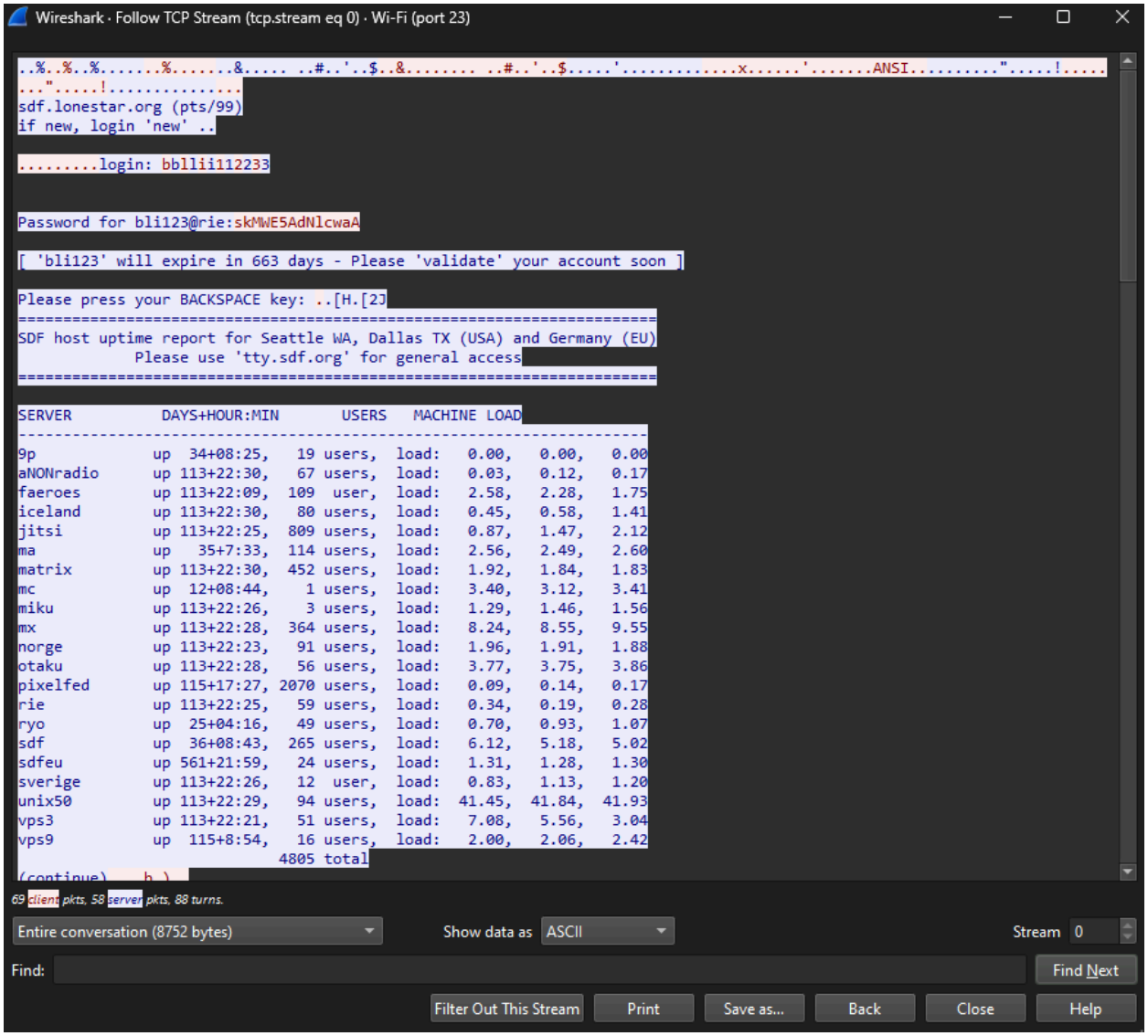
- Step 6 - After pausing the capture, click on the first TELNET packet. In the Packet Details pane, open up **Transmission Control Protocol**, and then open up **Flags**.



You will notice that the push flag is set to 1, which means that whenever a character comes from Telnet, it is immediately sent to the receiving application, rather than buffering it.

- Step 7 - Right-click that same packet and hover over **Follow.** Select **TCP Stream**



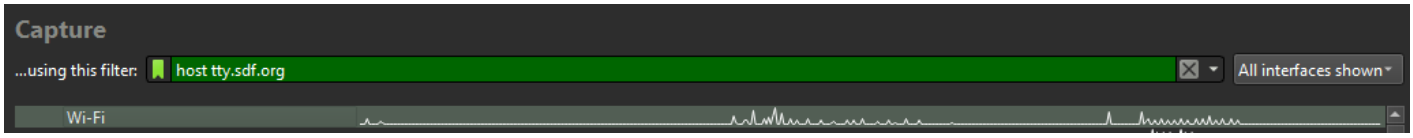


From the TCP Stream, we can observe that Telnet is not encrypted whatsoever as we can see everything that was exchanged. In Wireshark, letters highlighted in red represent what we sent to the server whilst letters highlighted in blue represent what the server sends back.

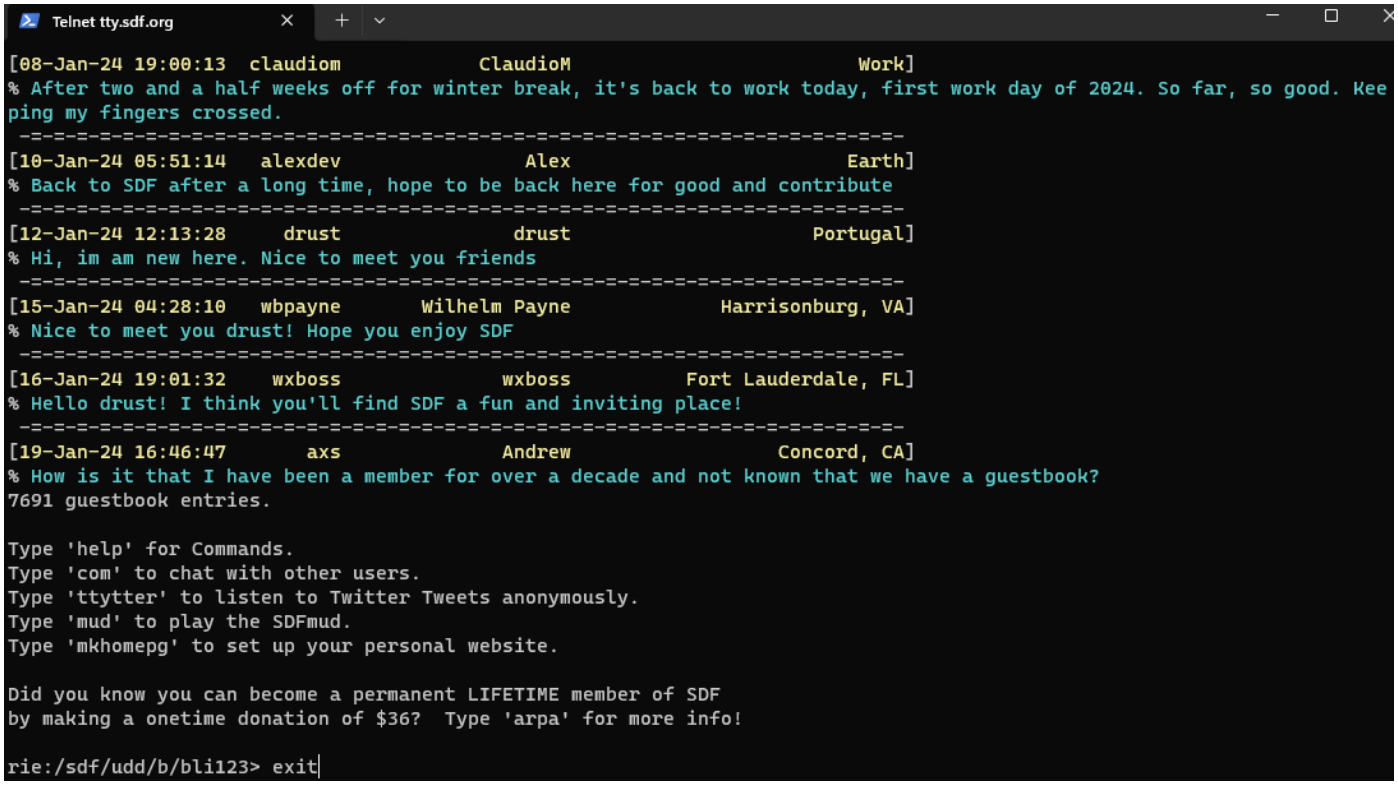
3.5 - Capturing and Analyzing SSH Sessions

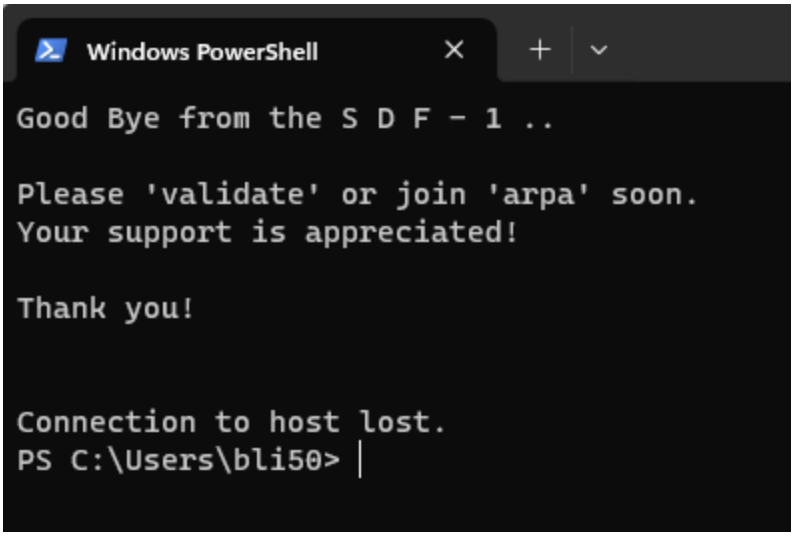
- Step 1 - For this task, we will be analyzing SSH, a more secure form of Telnet. As Telnet runs on port 23, SSH would run on port 22. However, we want to capture all of the traffic

that is happening between us and a remote server. Thus, instead of entering port 22 into the capture filter, we would enter "`host tty.sdf.org`".

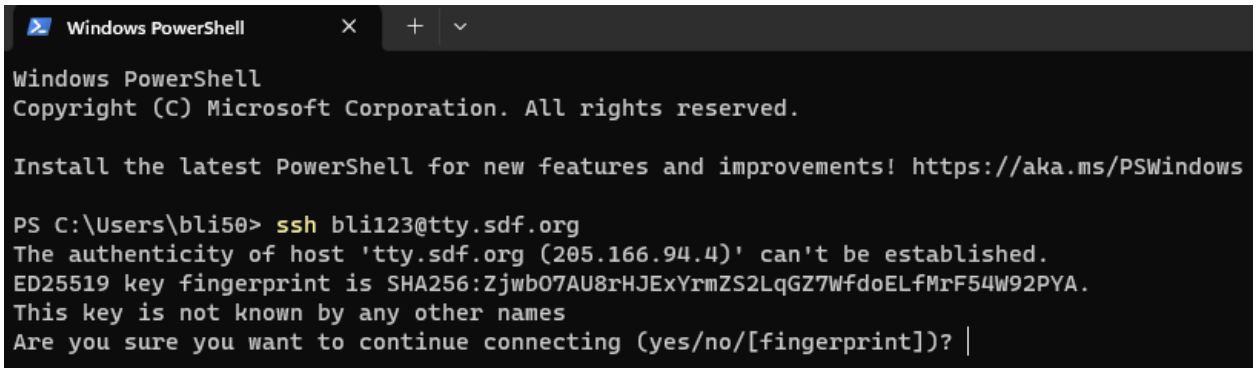


- Step 2 - Open up Powershell and enter "`telnet tty.sdf.org`". Login with the same credentials used in the previous experiment. Hit Backspace and then Enter. Upon reaching the final screen, type "`exit`" and press Enter. After the connection is terminated, close Powershell.



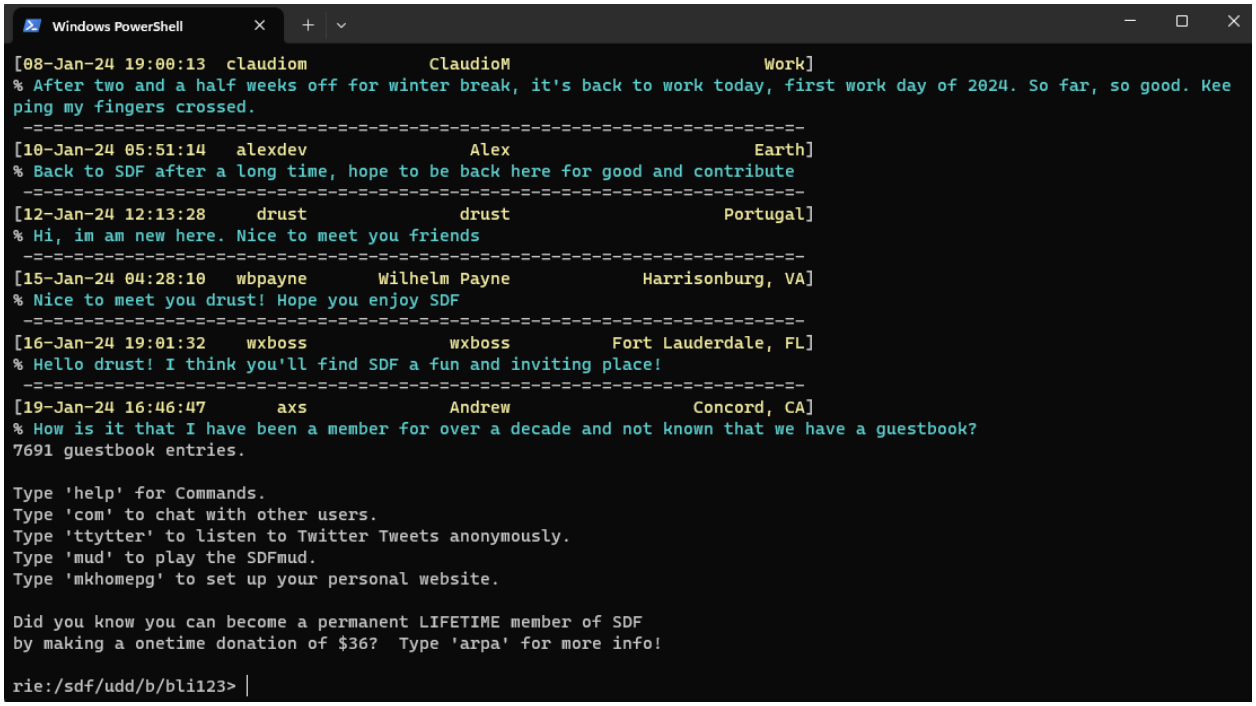


- Step 3 - Do not stop the Wireshark capture and open a new instance of Powershell. Type in "`ssh [USERNAME]@tty.sdf.org`", replacing the brackets with your login id. Respond "`yes`" to the next question.

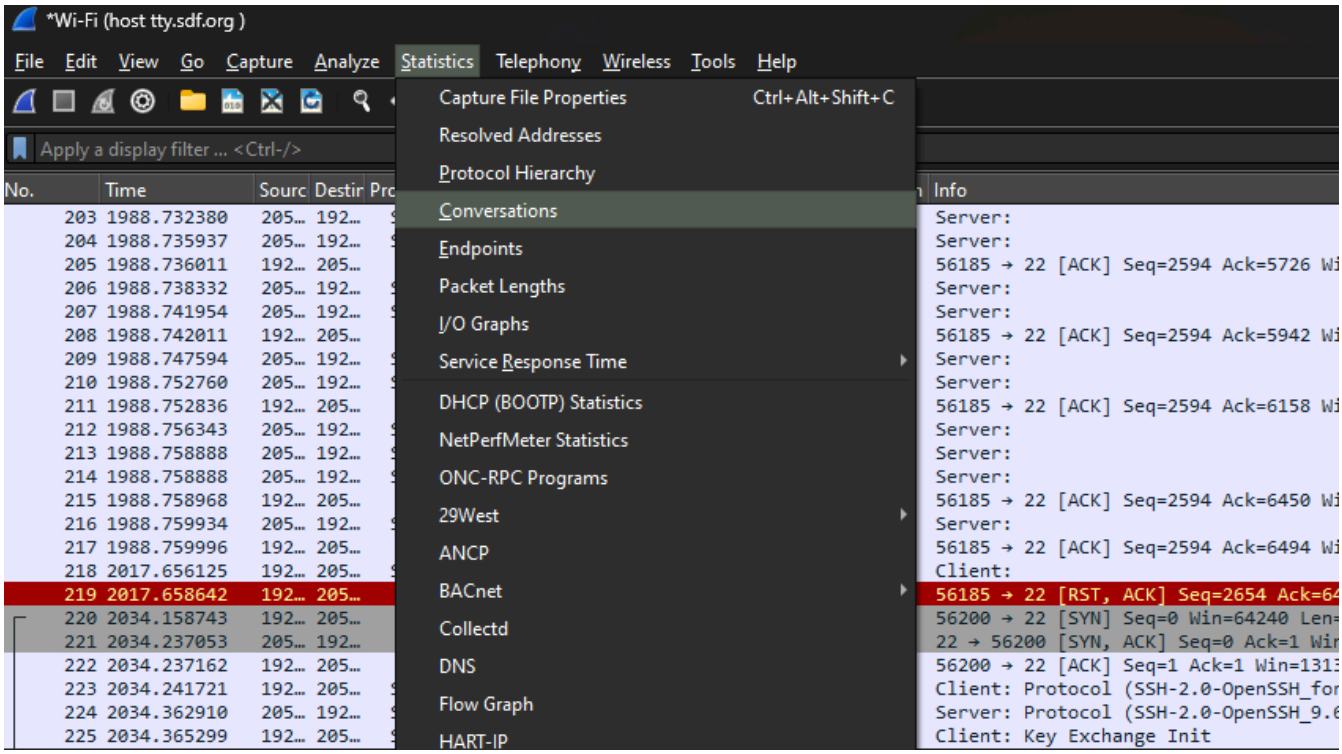


The authenticity question seen above is not an error, but rather just SSH being secure as it detects a new host. This ensures that it is not connecting with an imposter host. By typing in "yes", it will store your credentials and the message won't pop up again.

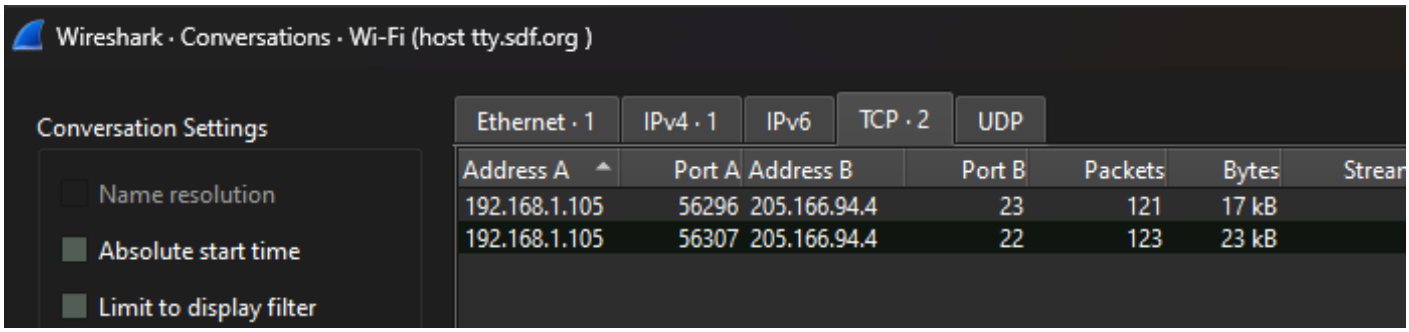
- Step 4 - After typing yes, you should be prompted to enter your password. Press Enter repeatedly until you see the same output from Telnet.



- Step 5 - On the final screen, type "`exit`" and close out of Powershell. Return to Wireshark and stop the capture. At the top, click on **Statistic** and then **Conversations**.

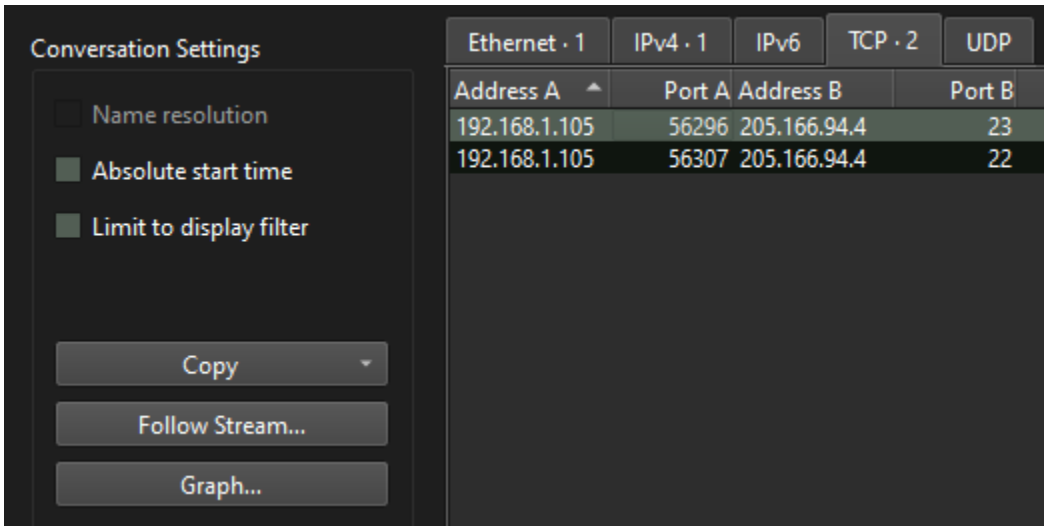


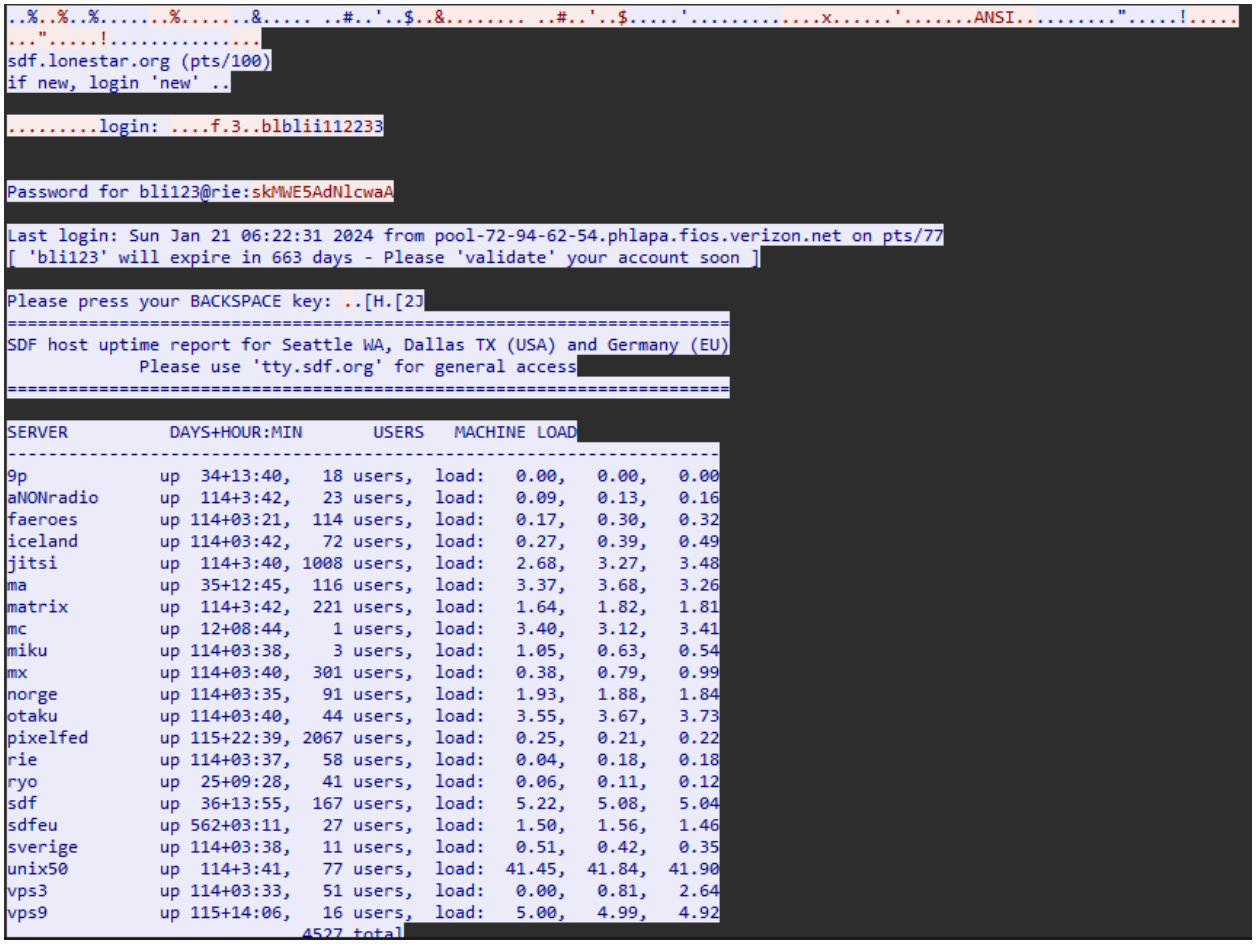
- Step 6 - Since both SSH and Telnet utilize TCP, click **TCP** at the top of the pop-up.



You can observe the two TCP conversations, the first being Telnet on port 23 and the second being SSH on port 22.

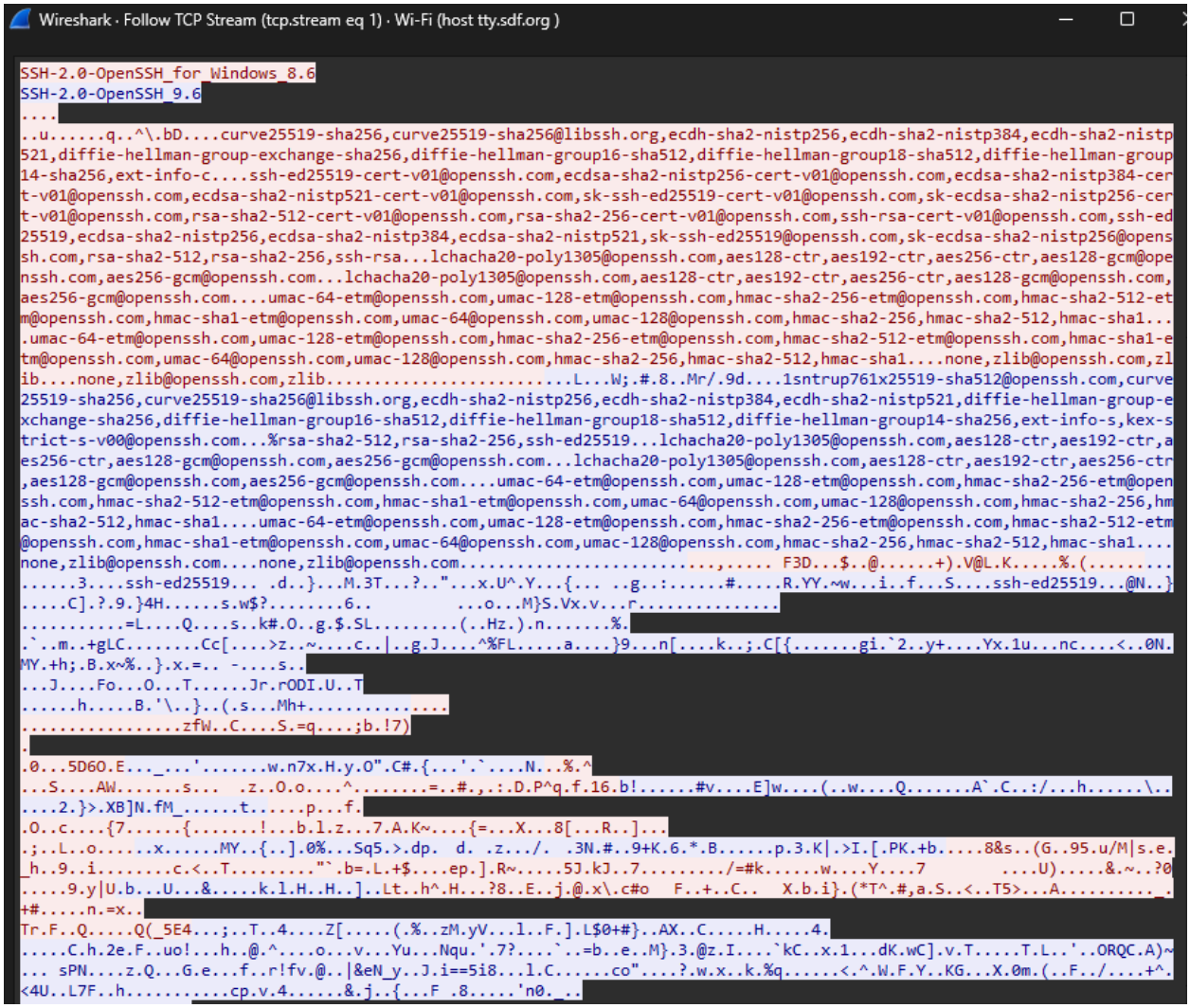
- Step 7 - Click on the first conversation, which should be Telnet. On the left side, click the **Follow Stream** button.





From the previous task, we know that Telnet is not secure as it does not encrypt any of the plain text information.

- Step 8 - Going back to the Conversations pop-up, select the second conversation on port 22 and click **Follow Stream**.

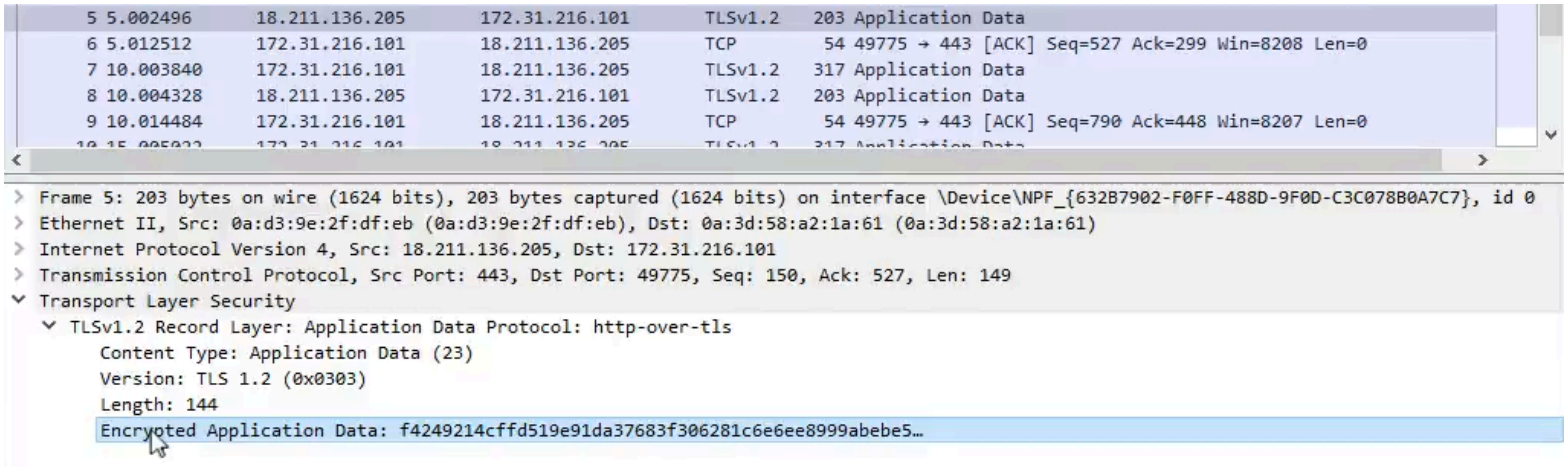


Over on SSH, we can see that everything is encrypted. Thus, if any middleman captures this traffic, they won't be able to decipher it.

3.6 - Generate, Capture, Analyze then Decrypt HTTPS Traffic

- Step 1 - For this task, we will experiment with HTTPS, an extension of HTTP that is used for secure communication. While HTTP operates on port 80, HTTPS operates on port 443. Thus, we will enter "`port 443`" into the filter and begin capture.

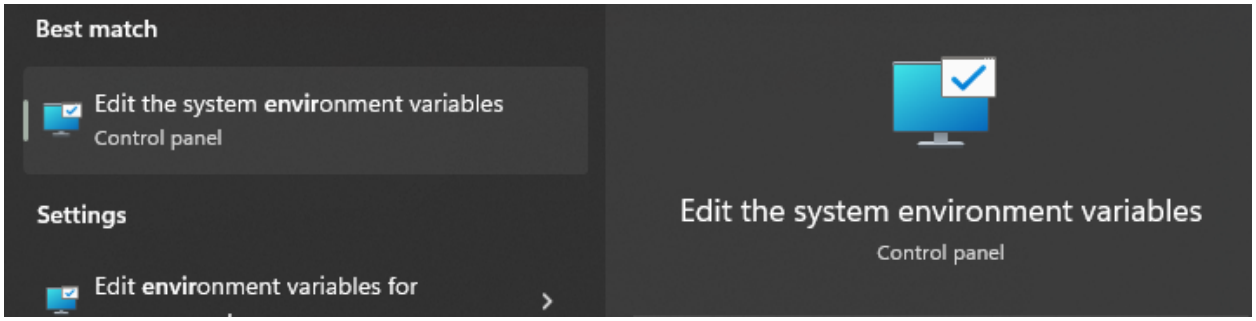
- Step 2 - Immediately, there should be some traffic being generated. However, if there are none, just simply open your web browser and visit a website such as Google or YouTube. Stop the capture.
- Step 3 - Select a packet that reads **Application Data** in the info column. In the bottom details window, expand **Transport Layer Security** and then expand the **TLS Record Layer**.



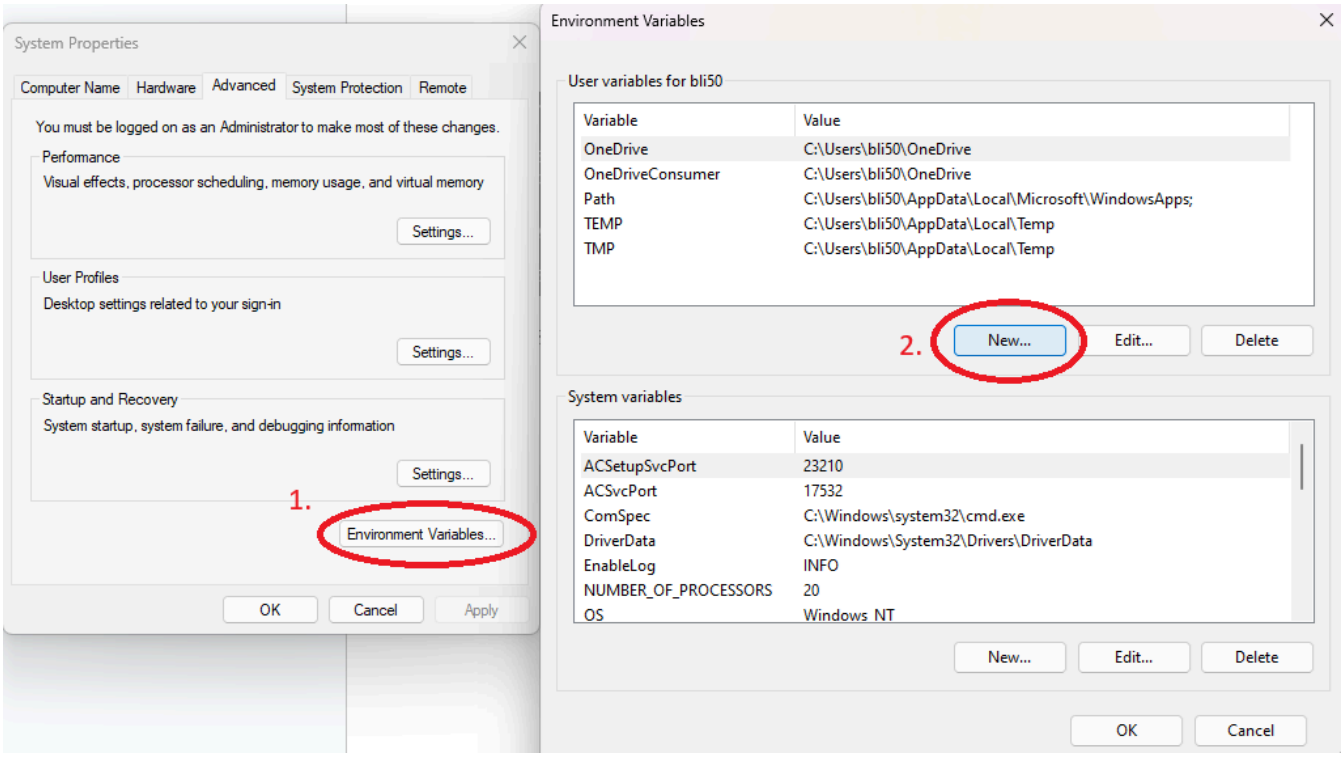
Observe that the Application Data is encrypted. This cannot be deciphered as we do not possess the private key that was used to encrypt. Remember for HTTP, we were able to see login credentials.

- Step 4 - We will now try to decrypt the SSL traffic using the pre-master secret key. Our browser will act as the client and will generate this secret key which will then be logged into an SSL key log file. The key is then used by the server (webpage) to generate a

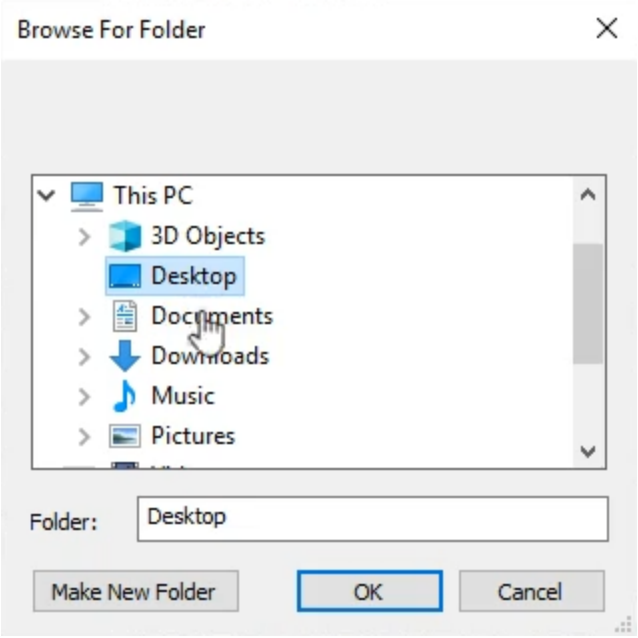
master secret key that encrypts all traffic. Click on the Windows Start button and type in "`environment variables`".

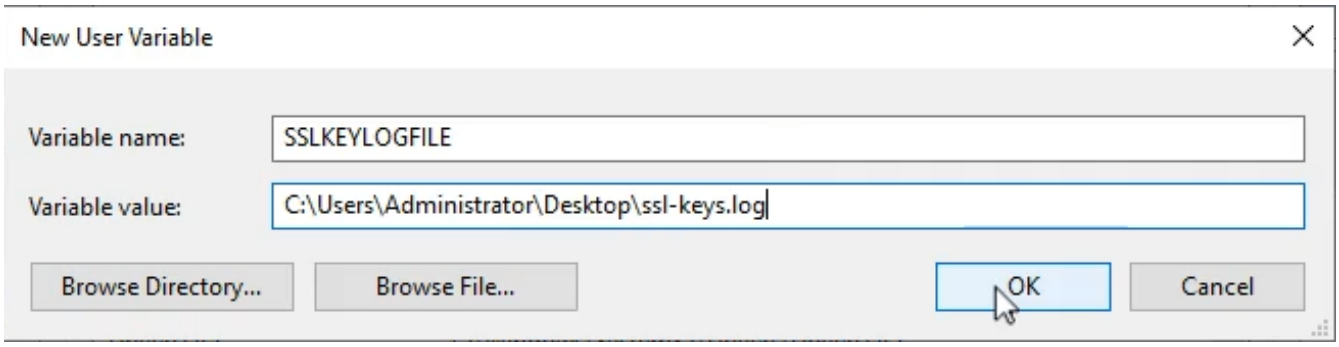


- Step 5 - In system properties, click on **Environment Variables**. In the pop-up, click "New".



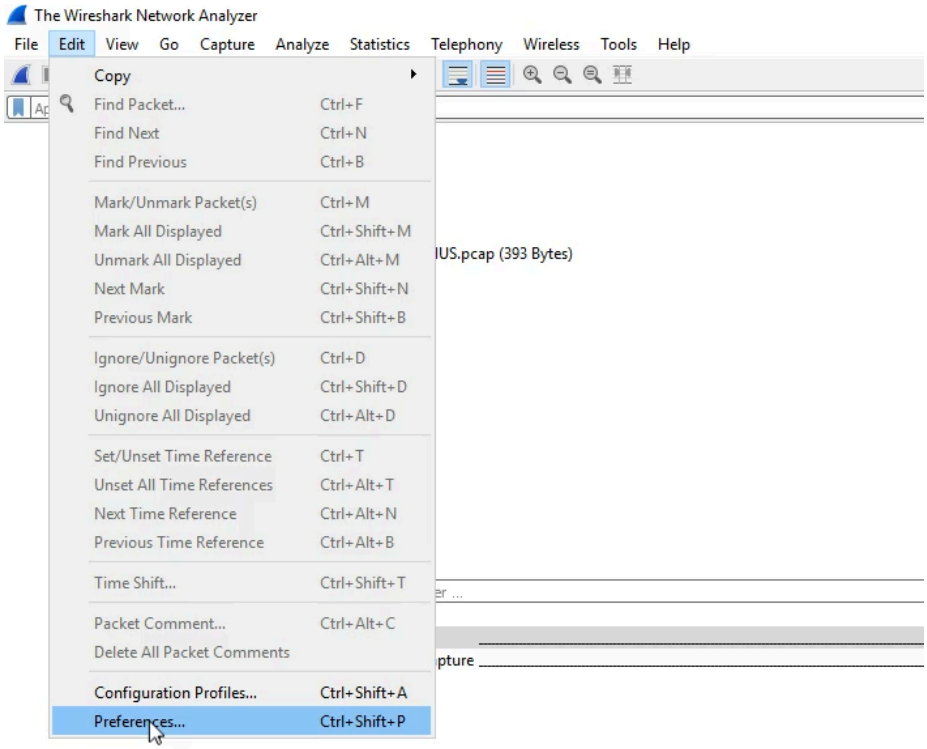
- Step 6 - For the New User Variable, type in "`SSLKEYLOGFILE`" next to the Variable name. For the Variable value, we should put the location where the key will be logged in. To do this, click on **Browse Directory** and select the path. At the end of the path, type in "`ssl-keys.log`". Click OK to close all windows.



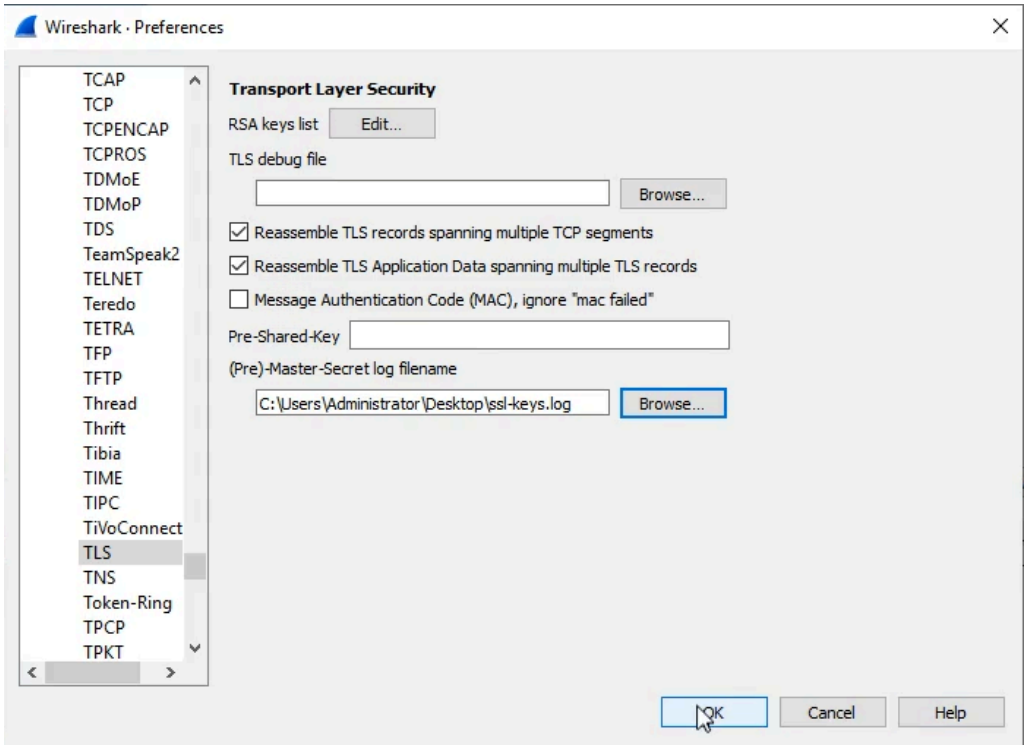


- Step 7 - Before we begin capturing traffic, let us make sure that the secret keys are being logged. Close any instances of your browser and open a new one. Go to any website. After reaching that website, there should now be a text document with the name "ssl-keys.log".

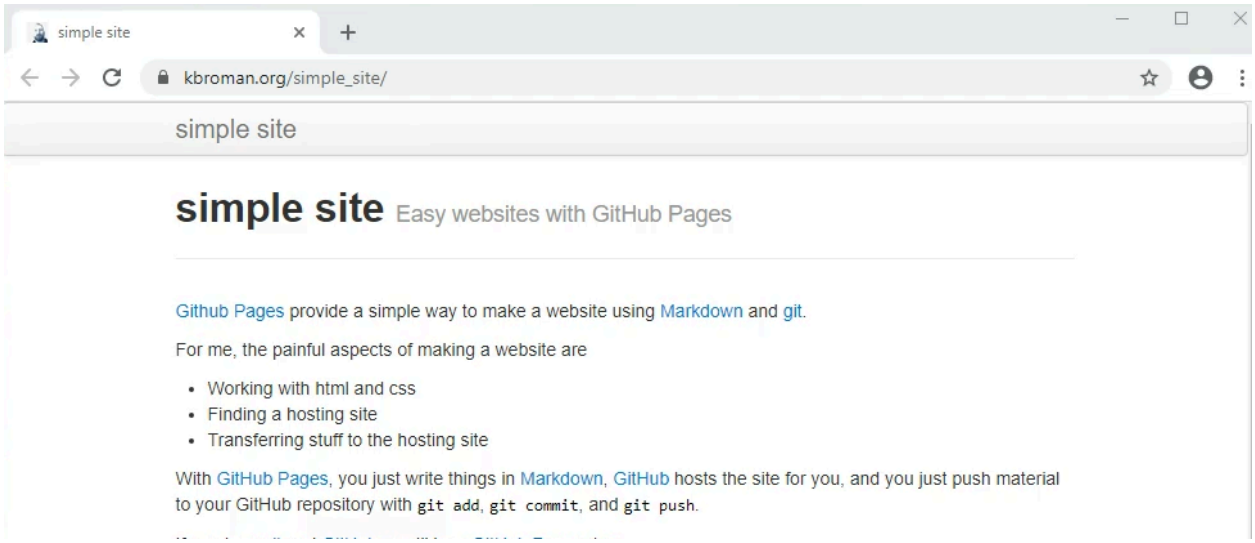
- Step 8 - Go back to the Wireshark home screen. Click **Edit** in the top left and then **Preferences**.



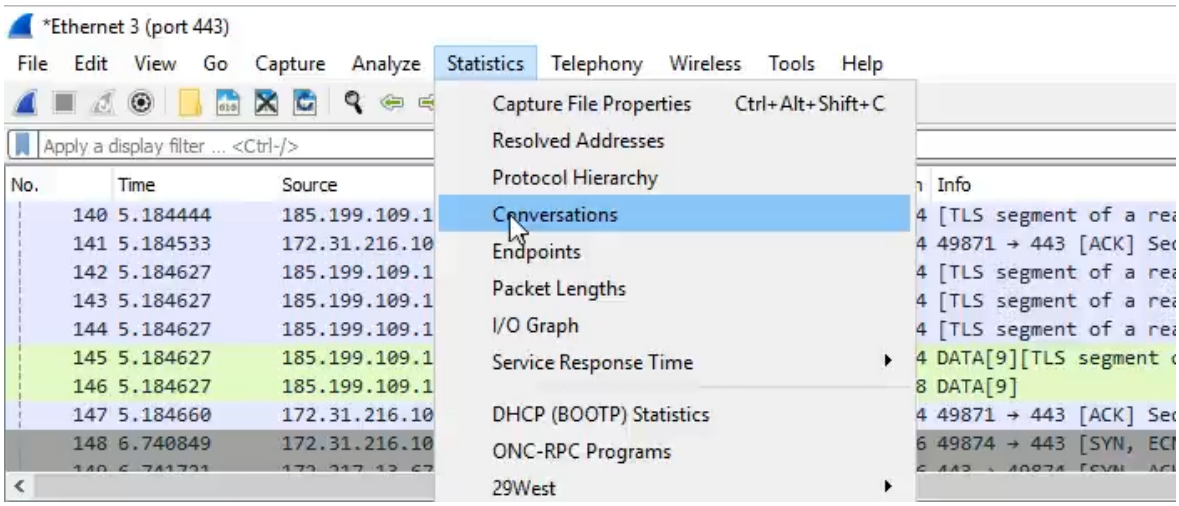
- Step 9 - Expand **Protocols** on the left side and look for **TLS**. Click **Browse** and search for the SSL key logs. Click **OK** once finished.



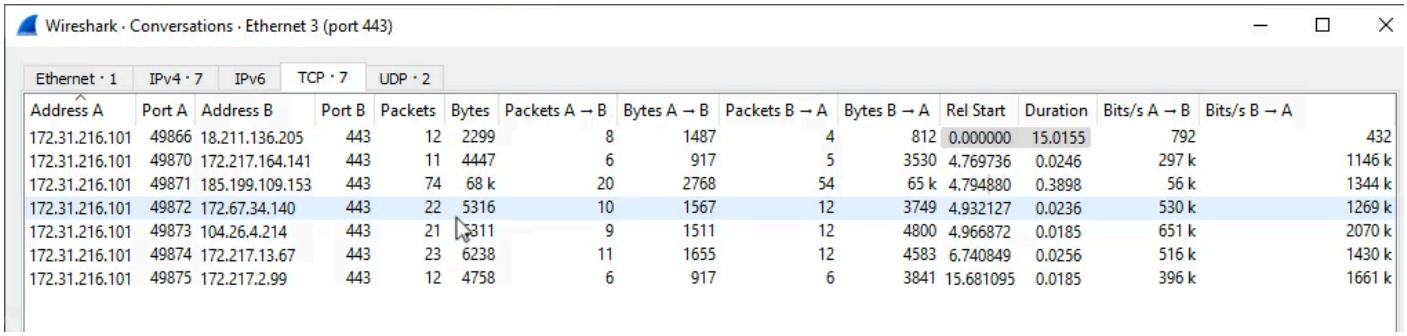
- Step 10 - Begin another Wireshark capture on `port 443`. Then, open this [simple website] on your browser.

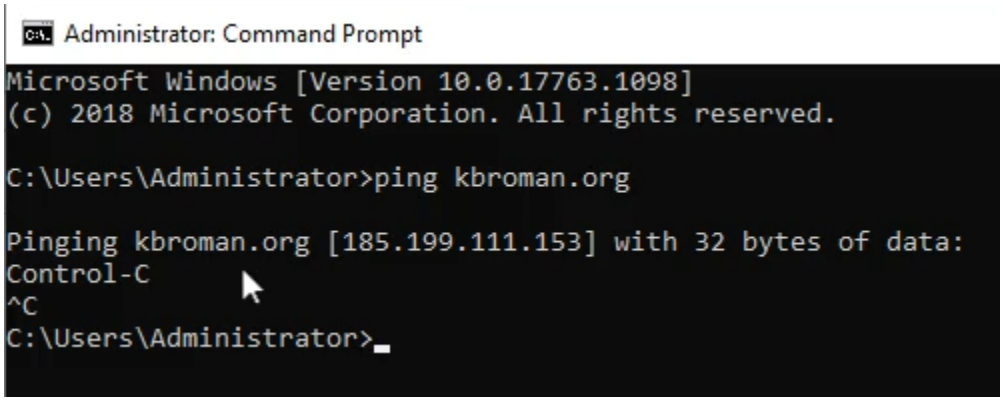


- Step 11 - Return to Wireshark and stop the capture. Above, click on **Statistics** and then click on **Conversations**.



- Step 12 - As you can see, there are several conversations in which we are connected. To figure out which conversation is the server, we must ping it. Retrieve the hostname of the website, which should be "`kbroman.org`", and ping it through the command prompt.

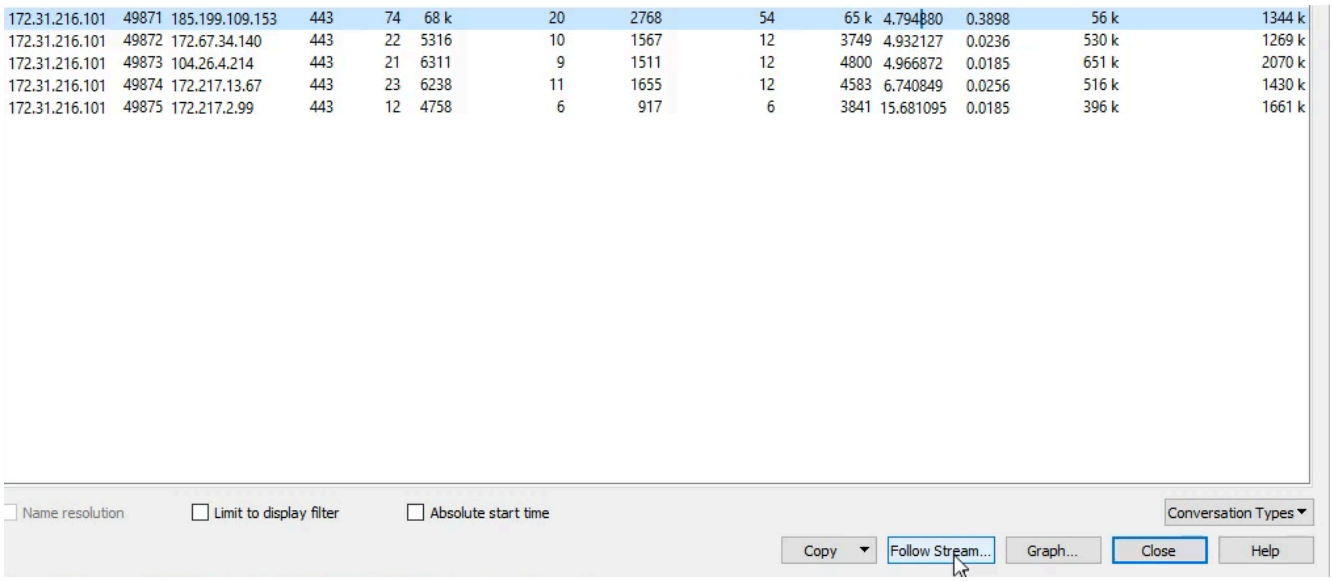




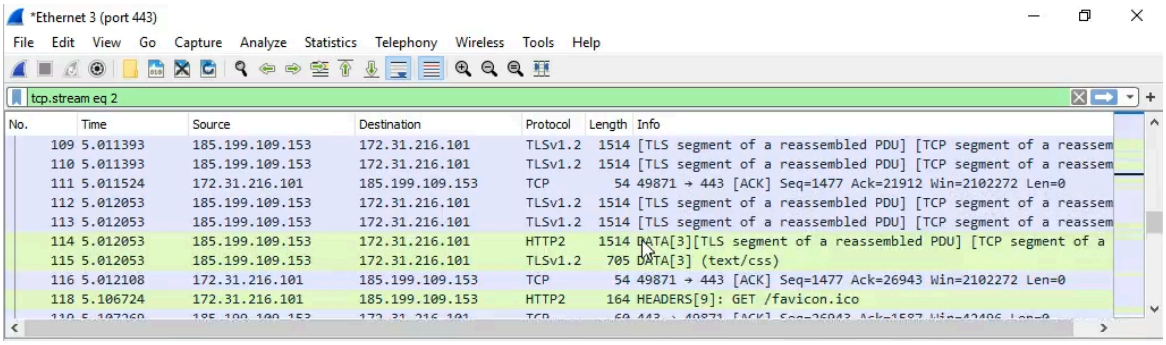
"ping kbroman.org"

From the ping, we can see that the IP is 185.199.111.153

- Step 13 - Once the IP address has been located in the Conversations window, click on it and then press **Follow Stream** at the bottom right.

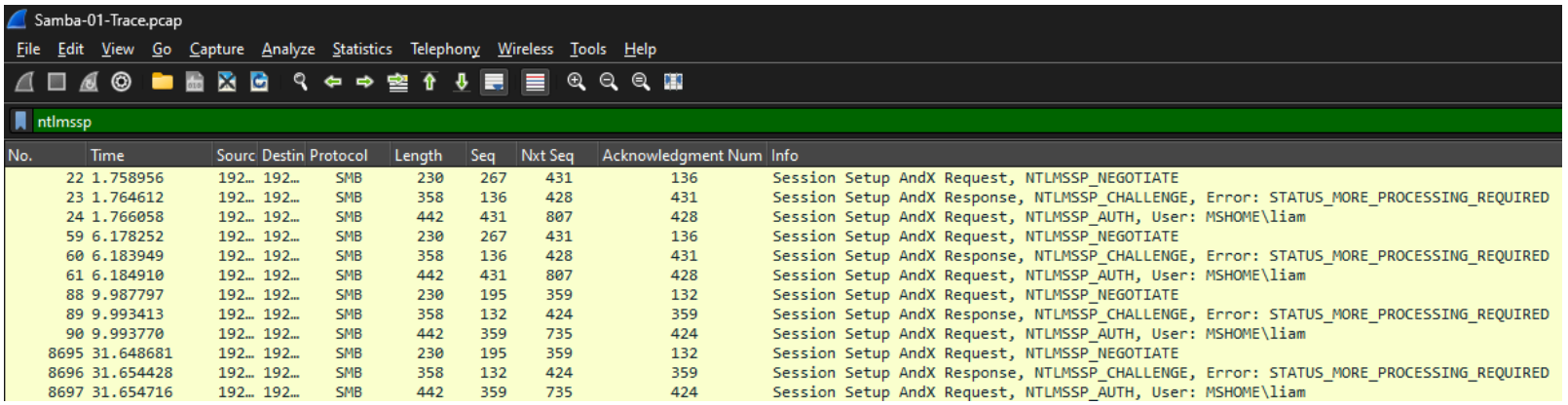


- Step 14 - Going back to the Wireshark capture, we can also see that all of the packet info has been decoded. Previously, it would say “Application Data” but now, we are even able to see the source code of websites.



3.7 - Extracting NTLMv2 Hashes

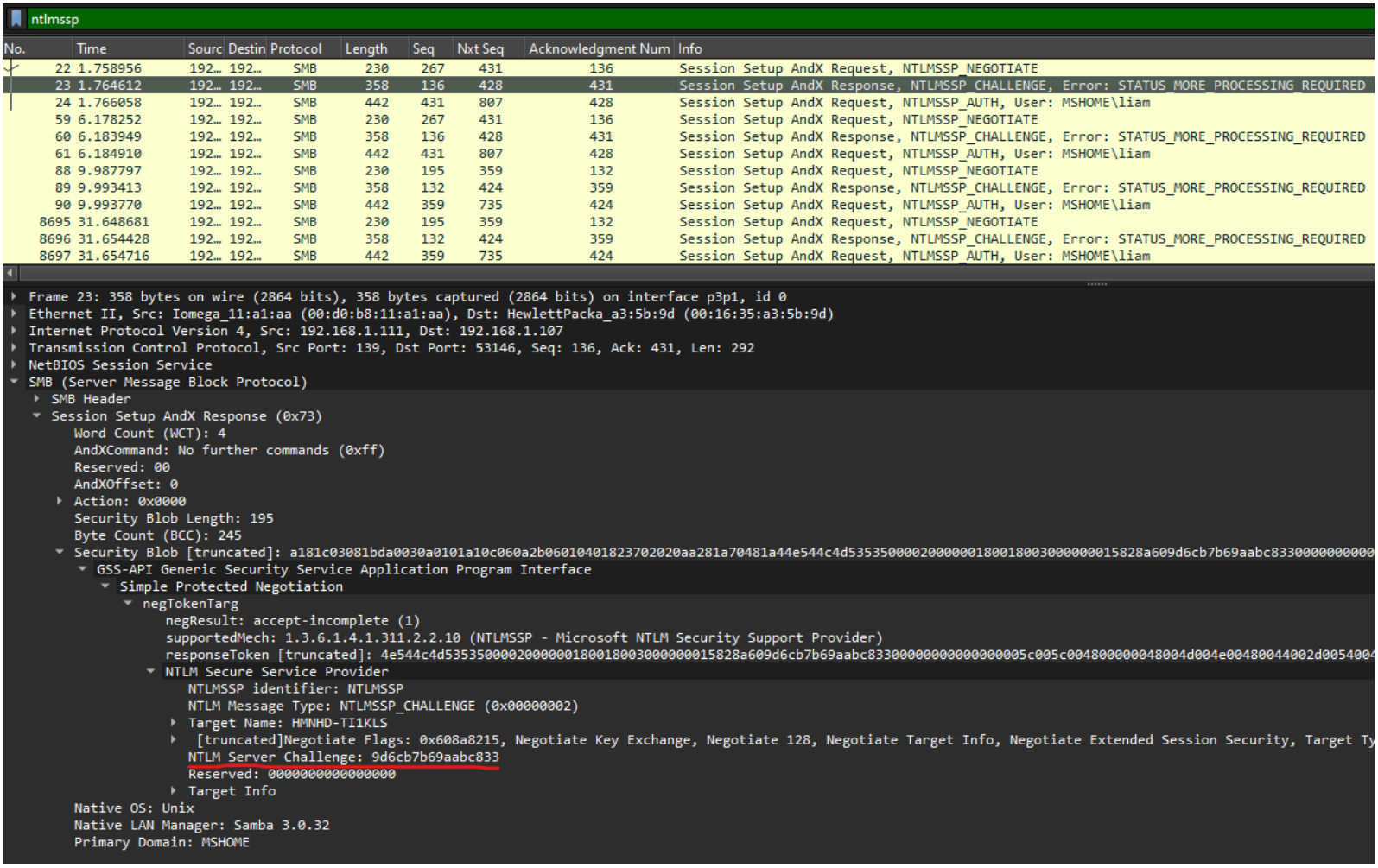
- Step 1 - Obtain a Wireshark capture that contains NTLM. If you are using an older system such as Windows 10 or older, this may still be possible through a normal capture. However, as NTLM was replaced by Kerberos for Windows 11, I found a pcap online that contained the necessary protocols. Filter "`ntlmssp`".



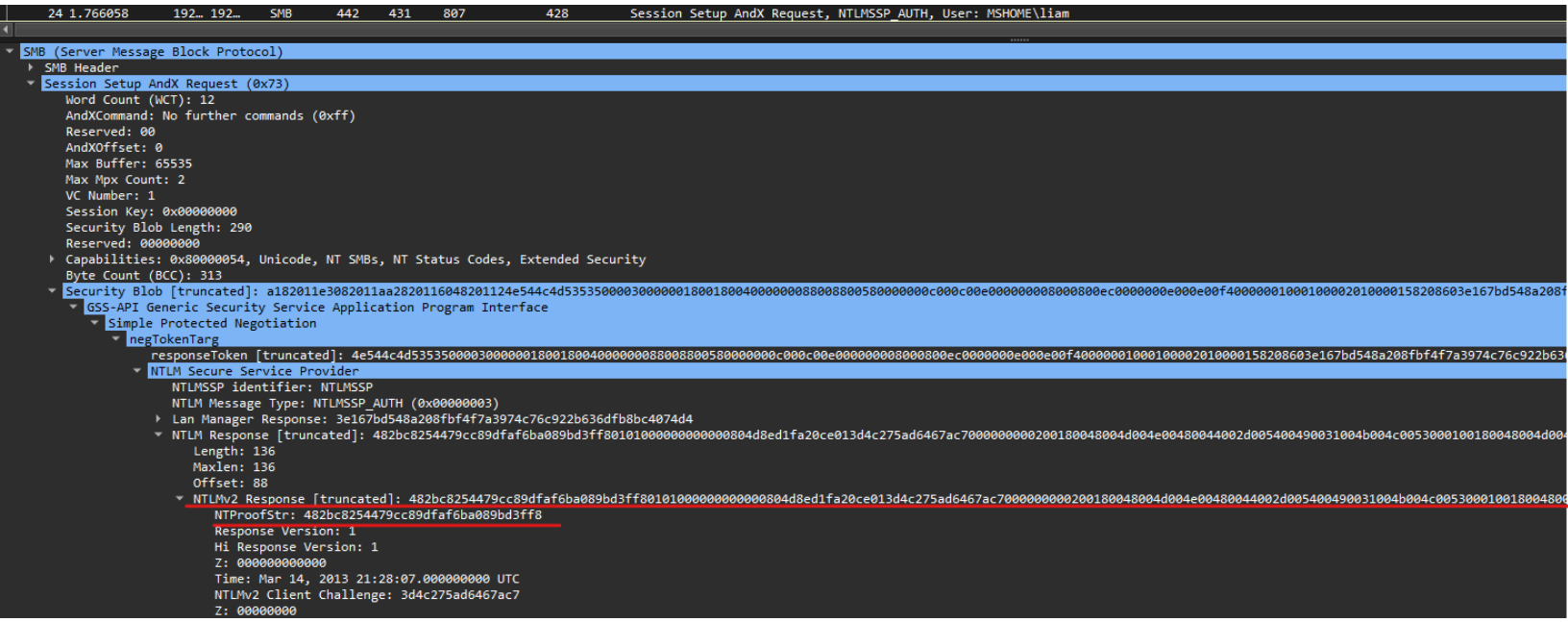
- Step 2 - Open Notepad and create a blank text document. Type out a list similar to the one in the screenshot below. This will be used to record the parts that make up our NTLMv2 hash.

```
User:
Domain:
Challenge:
HMAC-MD5:
NTLMv2Response:
```

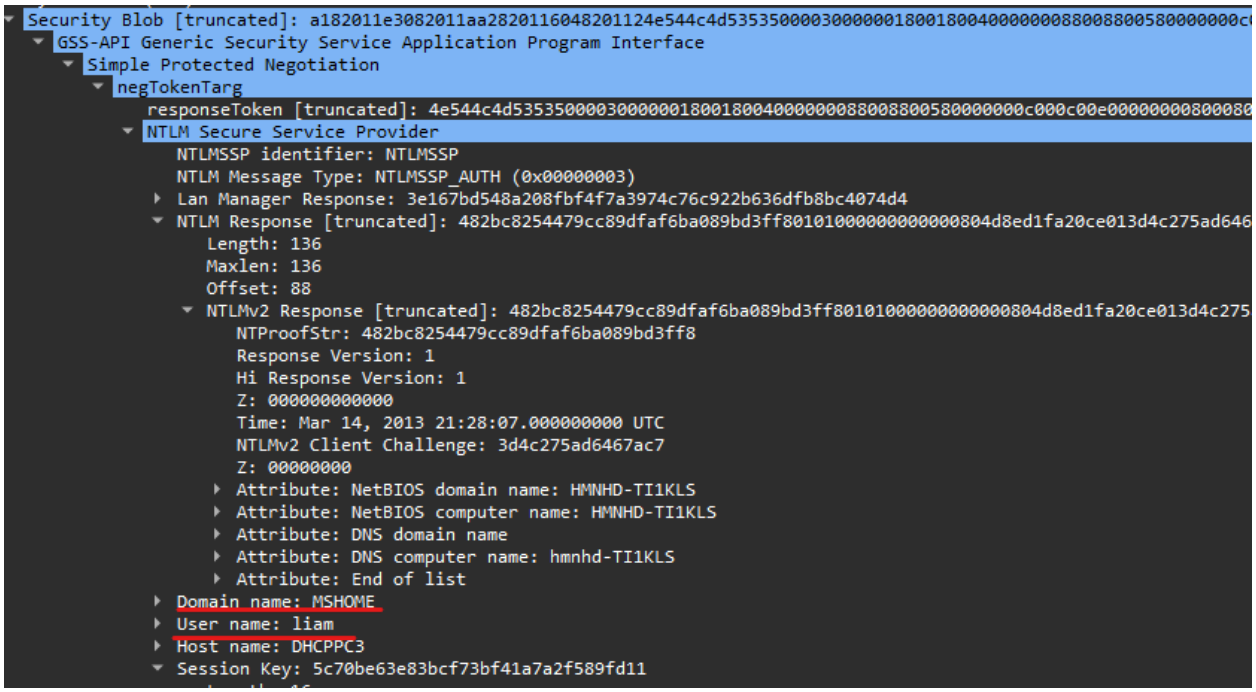
- Step 3 - Click on the packet that reads "NTLMSSP_CHALLENGE". Open each of the filters until you reach "NTLM Secure Service Provider". Under this filter, you should find **NTLM Server Challenge**. Enter the information into your list.



- Step 4 - Click on the packet that reads “NTLMSSP_AUTH”. Similar to the previous step, expand the packet until you see “NTLM Secure Service Provider”, and then expand once more on “NTLM Response” to reveal **NTLMv2 Response**. Copy and paste the contents into your list. The string below that, “NTProofStr:”, will also be entered into the list as **HMAC-MD5**.



- Step 5 - The **User** and **Domain** can also be found under the same packet and the same filters.



Once you have all of the components, your list should look like this:

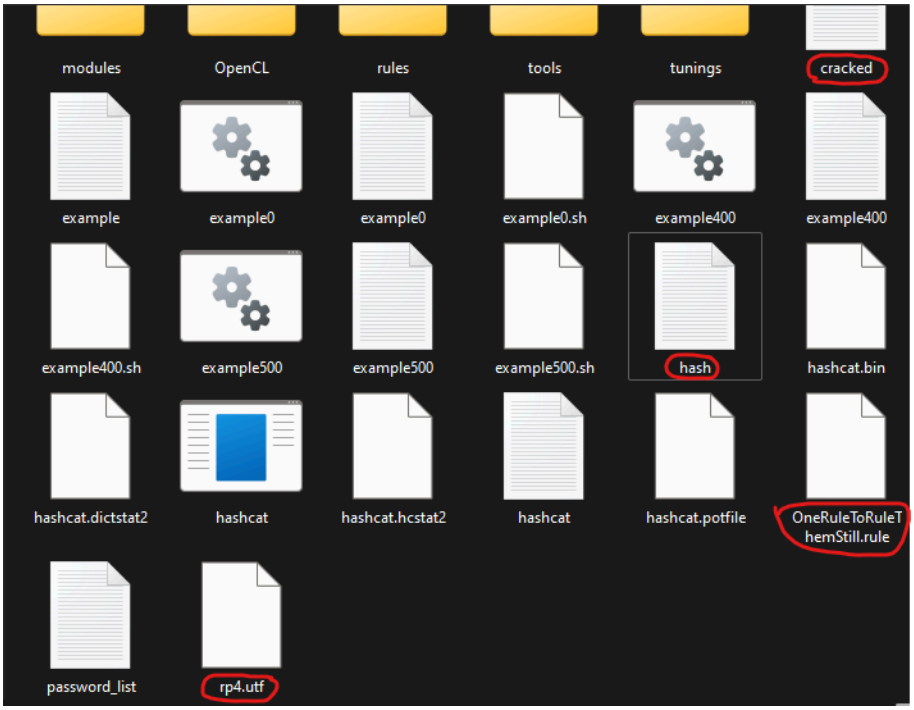
```
User: liam
Domain: MSHOME
Challenge: 9d6cb7b69aabc833
HMAC-MD5: 482bc8254479cc89dfaf6ba089bd3ff8
NTLMv2Response:
482bc8254479cc89dfaf6ba089bd3ff80101000000000000804d8ed1fa20ce01
005400490031004b004c00530000000000
```

- Step 6 - Combine the **User**, **Domain**, **Challenge**, **HMAC**, and **NTLMv2Response** to create one long NTLMv2 Hash.

User::Domain:Server Challenge:HMAC-MD5 (NTProofStr):NTLMv2Response (without HMAC)

"liam::MSHOME:9d6cb7b69aabc833:482bc8254479cc89dfaf6ba089bd3ff8:482bc8254479cc89dfaf6ba089bd3ff80101000000000000804d8ed1fa20ce013d4c275ad6467ac70000000002001800480004d004e00480044002d005400490031004b004c00530001001800480004d004e00480044002d005400490031004b004c00530004000000030018006800 6d006e00680064002d005400490031004b004c00530000000000"

- Step 7 - Download Hashcat. Within the folder where Hashcat is, create two text files. One will be named "**hash**" and the other "**cracked**". The **hash** text document will contain the NTLMv2 hash that you just created whereas the **cracked** document will contain the cracked password. You can also take this opportunity to download any dictionaries or rules that will help your search. I utilized the up4 wordlist as well as the OneRuleToRuleThemStill rules.



- Step 8 - Open up a command prompt and use the "`cd`" command to access the location of your Hashcat.

```
C:\Users\bli50>cd /d C:\Hashcat\hashcat-6.2.6

C:\Hashcat\hashcat-6.2.6>
```

- Step 8 - We will now attempt to crack the hash. Enter the command:

"`hashcat -a0 -m5600 hash.txt rp4.utf -r OneRuleToRuleThemStill.rule -o cracked.txt`"

(`-a0`) Specifies the attack mode. In this case, we are using a straight attack with a wordlist.

(`-m5600`) Identifies the hash as an NTLMV2 Hash

(`-r`) Specifies the rule file used.

(`-0`) Specifies the outfile.

- Step 9 - Depending on how powerful your system is, it may take a while for the cracking process to finish. For me, it took 27 minutes and this was the outcome:

```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 5600 (NetNTLMv2)
Hash.Target......: LIAM::MSHOME:9d6cb7b69aabc833:482bc8254479cc89dfaf6...000000
Time.Started.....: Mon Jan 29 21:41:54 2024 (26 mins, 58 secs)
Time.Estimated...: Mon Jan 29 22:08:52 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (rp4.utf)
Guess.Mod........: Rules (OneRuleToRuleThemStill.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1258.2 MH/s (1.38ms) @ Accel:64 Loops:64 Thr:64 Vec:1
Recovered........: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.........: 2308781549856/2308781549856 (100.00%)
Rejected.........: 0/2308781549856 (0.00%)
Restore.Point....: 47688304/47688304 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:48384-48414 Iteration:0-64
Candidate.Engine.: Device Generator
Candidates.#1....: $HEX[f2e8efeeebeee3e8ff36] -> $HEX[9953995332]
Hardware.Mon.#1..: Temp: 62c Fan: 74% Util: 93% Core:1980MHz Mem:6800MHz Bus:16

Started: Mon Jan 29 21:41:50 2024
Stopped: Mon Jan 29 22:08:53 2024
```

Unfortunately, Hashcat exhausted itself. This means that of every possible password combination within the given parameters, none yielded results. It does NOT mean that something went wrong, nor that something isn't working correctly. The next step would be to retry with different wordlists or rules.

```
Session..........: hashcat
Status...........: Cracked
Hash.Name........: SolarWinds Serv-U
Hash.Target......: e983672a03adcc9767b24584338eb378:00
Time.Started.....: Sun May 23 11:43:13 2021 (1 sec)
Time.Estimated...: Sun May 23 11:43:14 2021 (0 secs)
Guess.Mask.......: ?a?a?a?a?a?at [7]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 24620.9 MH/s (32.19ms) @ Accel:32 Loops:1024 Thr:1024 Vec:1
Recovered........: 1/1 (100.00%) Digests
Progress.........: 31606272000/735091890625 (4.30%)
Rejected.........: 0/31606272000 (0.00%)
Restore.Point....: 0/857375 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:35840-36864 Iteration:0-1024
Candidates.#1....: 4{,erat -> cyr ~}t
Hardware.Mon.#1..: Temp: 62c Fan: 31% Util:100% Core:1920MHz Mem:7000MHz Bus:16

Started: Sun May 23 11:43:12 2021
Stopped: Sun May 23 11:43:15 2021
```

This is what a successful cracked hash would look like.

4. Findings

4.1 Generating and capturing RADIUS traffic

For the first experiment, the focus was on analyzing the RADIUS (Remote Authentication Dial-In User Service) protocol, which operates on port 1812 and is commonly used for centralized authentication, authorization, and accounting management in network services. The demonstration illustrated that RADIUS traffic is transmitted in clear text, meaning the data, including usernames and authentication attempts, is unencrypted and vulnerable to interception by potential malicious actors. This lack of encryption poses a significant security risk, as demonstrated through the Wireshark captures, where authentication requests and responses were visible in plaintext. Additionally, the experiment showcased the use of a shared secret or secret key to hash and protect passwords during transmission. However, the encryption

provided by this method was shown to be weak, as Wireshark was used to decrypt and reveal the passwords easily. This experiment underscores the importance of securing sensitive information in network protocols, emphasizing the need for encryption mechanisms to ensure the privacy and integrity of user credentials during authentication processes.

4.2 Analyzing an HTTP Basic Authentication

For this task, the focus was on analyzing the security vulnerabilities associated with the HTTP (Hypertext Transfer Protocol) Basic authentication, which is an unencrypted protocol commonly used for user authentication on the web. The demonstration highlighted the inherent lack of security in HTTP, as it transmits sensitive information such as usernames and passwords in plaintext, making it susceptible to interception and unauthorized access by malicious entities. Wireshark captures revealed that even the basic authentication credentials were encoded in Base64, providing minimal obfuscation and no real encryption. The experiment emphasized the importance of transitioning to more secure protocols, such as HTTPS, which employs Transport Layer Security (TLS) to encrypt data during transmission, protecting user credentials from being easily visible to potential attackers. This analysis underscores the critical need for secure communication protocols, especially when handling sensitive information on the internet, and highlights the risks associated with relying on unencrypted protocols like HTTP for authentication.

4.3 HTTP Form-Based Authentication and DNS

Two HTTP authentication methods, Basic and Form-Based, were analyzed. It was noticed that while using form-based authentication, credentials were sent in the body of the POST request, making them visible in plaintext in HTML form fields. This demonstrated that to

encrypt sensitive data during transmission utilizing Form-Based authentication, HTTPS is required. Additionally, the experiment recorded and analyzed DNS traffic, which showed that sensitive information like IP addresses and website addresses is exposed due to unencrypted DNS communication. This underlines the significance of encryption protocols in securing various aspects of online communication, as sensitive information, including website addresses and corresponding IP addresses, is exposed without encryption in unencrypted protocols like DNS. Overall, the experiment underscored the critical need for adopting secure practices in online communication and the potential risks associated with unencrypted protocols.

4.4 Initiate, Capture, and Analyze Telnet Sessions

In this task, the Telnet protocol was explored to understand its communication characteristics and potential security vulnerabilities. The analysis revealed that Telnet, operating on TCP port 23, sends data in plaintext, making it susceptible to eavesdropping and unauthorized access. The experiment displayed a Telnet session to “tty.sdf.org” and showed via TCP Stream that the collected packets displayed the username and the keystrokes that were transmitted. The security of distant communication is jeopardized because confidential information is exposed by Telnet's lack of encryption. The results highlight the necessity of encrypted protocols and the significance of avoiding Telnet for the transmission of sensitive data, particularly when communicating with remote devices. The experiment concluded the study of unencrypted protocols and prepared the ground for the next tasks, which would investigate encrypted protocols.

4.5 Capturing and Analyzing SSH Sessions

The focus of this experiment was on comparing the encrypted protocol SSH (Secure Shell) with its unencrypted counterpart Telnet. The analysis revealed that Telnet, operating on port 23, sent data in plaintext, exposing sensitive information to potential eavesdropping. In contrast, SSH, utilizing port 22 and TLS encryption, secured the communication between the client and the server. The packet captures showed a significant contrast between the encrypted SSH connection, where the negotiation of encryption keys guaranteed the anonymity of the sent data, and the unencrypted Telnet session, where every keystroke and answer were visible. It was emphasized how crucial encryption is to the security of remote access protocols and how much more secure encrypted protocols—like SSH—offer than their unencrypted counterparts in terms of protection against unauthorized interception and improved remote communication security in general. This experiment serves as a valuable introduction to the crucial role of encryption in network protocols and protecting sensitive data during communication.

4.6 Generate, Capture, Analyze, and Decrypt HTTPS Traffic

In this task, the focus shifted towards the encryption capabilities of HTTPS, providing a stark contrast to the insecurities identified in unencrypted protocols like HTTP. By capturing and analyzing HTTPS traffic on port 443, the experiment demonstrated the effectiveness of TLS (Transport Layer Security) in securing data transmission. The captured data, encrypted and unreadable without the pre-master secret key, showcased the significance of encryption in protecting sensitive information during online communication. We then carried through the process of decrypting HTTPS traffic in Wireshark using a simple method involving logging pre-master secret keys. This final exercise emphasized the practical aspects of network security

troubleshooting, empowering learners to not only capture and analyze encrypted traffic but also decrypt it for comprehensive security analysis.

4.7 Extracting NTLMv2 Hashes

For the final experiment, the process of capturing NTLMv2 hashes using Wireshark and attempting to crack them with Hashcat was explored. NTLM (NT LAN Manager) is a widely used authentication protocol in Windows environments, and capturing and cracking its hashes can highlight potential security vulnerabilities. By analyzing a Wireshark capture containing NTLMSSP (NT LAN Manager Security Support Provider) traffic, the NTLMv2 hash was extracted from the "NTLMSSP_CHALLENGE" and "NTLMSSP_AUTH" packets. These hashes, along with related information like User, Domain, Challenge, HMAC, and NTLMv2 Response, were then used in an attempt to crack the password using Hashcat. The significance lies in the practical demonstration of the process security professionals might follow to identify weaknesses in NTLM authentication. Although the hash was unable to be cracked, this emphasizes the importance of strong password policies and the need for organizations to adopt more secure authentication mechanisms.

5. Ethical Considerations

When it comes to utilizing Wireshark and Hashcat, it is important to address security and ethical considerations. Participants engaging in network traffic analysis must adhere to ethical standards and legal guidelines to ensure responsible and respectful usage of this powerful tool. It is essential to emphasize the importance of obtaining proper authorization before capturing any network traffic, as unauthorized monitoring may violate privacy regulations and legal

boundaries. While Wireshark itself is a legal tool, it can easily become illegal if one were to monitor a network without the proper authorization. Similarly, Hashcat itself is not illegal; it is a legitimate and open-source password-cracking tool used for security testing, but unauthorized use, such as attempting to crack passwords without permission, is illegal and can lead to serious legal consequences.

Sensitivity to the confidentiality of information transmitted over the network is crucial, with a clear emphasis on avoiding the capture of sensitive or personally identifiable data unless explicitly permitted. Additionally, the project should underscore the significance of securing captured data, ensuring that any findings or reports are shared responsibly and only with authorized individuals. This ethical foundation promotes a culture of responsible cybersecurity practices, fostering a deeper understanding of the balance between analytical exploration and the ethical responsibilities associated with network traffic analysis.

6. Future Considerations

Future implementations or variations of this project should consider staying ahead of emerging cybersecurity trends by incorporating more advanced exercises on IoT security analysis, cloud security, and the implications of quantum computing on encryption. Integrating elements of machine learning for threat detection, exploring Zero Trust Architecture, and addressing containerization and microservices security will enhance participants' readiness for contemporary industry practices. Additionally, incorporating insights on threat intelligence analysis, compliance with data protection regulations, and privacy considerations will provide a holistic approach to cybersecurity challenges. By evolving to encompass these future considerations, the Wireshark project can ensure its relevance in preparing to navigate the dynamic and ever-evolving landscape of cybersecurity.

7. Conclusion

The Wireshark technical project provided valuable hands-on experience in network traffic analysis. By achieving the outlined learning objectives and completing tasks, a solid foundation in using Wireshark for basic network security analysis was established. The project as a whole provided insights into various network protocols, both unencrypted and encrypted, enhancing the participants' understanding of security challenges and solutions in real-world networking scenarios.

The project's findings underscored the crucial role of encryption in safeguarding sensitive information during network communication. Analyses of RADIUS, HTTP, Telnet, SSH, and HTTPS traffic emphasized the risks associated with unencrypted protocols and advocated for the adoption of secure alternatives. Additionally, the practical exercise of capturing and attempting to crack NTLMv2 hashes highlighted the importance of robust password policies. Overall, the project not only enhanced participants' proficiency in network traffic analysis but also equipped them to address real-world cybersecurity challenges through an understanding of encryption, authentication mechanisms, and secure communication protocols.

In conclusion, the Wireshark technical project served as a multifaceted learning experience, encompassing practical skills, theoretical knowledge, and a heightened awareness of cybersecurity challenges. By navigating through the intricacies of network traffic analysis, encryption, and authentication, participants mastered the use of Wireshark and laid the groundwork for addressing the dynamic and evolving landscape of cybersecurity. This comprehensive skill set positions individuals to tackle real-world security issues and adapt to the ever-changing nature of cyber threats.

8. Appendices

- Brathwaite, Shimon. “Hashcat Cheat Sheet.” *Medium*, Geek Culture, 1 Aug. 2022, medium.com/geekculture/hashcat-cheat-sheet-511ce5dd7857.
- Porup, J. (n.d.). *Hashcat explained: How this password cracker works*. CSO Online. https://www.csoonline.com/article/569355/hashcat-explained-why-you-might-need-this-password-cracker.html
- *How to perform a rule-based attack using Hashcat*. 4ARMED. (2016, September 12). https://www.4armed.com/blog/hashcat-rule-based-attack/
- Sharpe, R., Warnicke, E., & Lamping, U. (2024). Wireshark User’s Guide. https://www.wireshark.org/docs/wsug_html_chunked/