

# **Wireshark for Basic Network Security Analysis**

This project will contain details on how to capture network traffic that is flowing through a machine now and analyzing already captured network traffic by opening a stored capture file in Wireshark. It will also contain the differences between Wireshark Capture and Display Filters. Generating, capturing, and analyzing RADIUS, HTTP, DNS and Telnet Traffic. It will contain Decrypting RADIUS passwords. Generating, capturing analyzing encrypted protocols such as SSH and HTTPS and decrypt HTTPS traffic

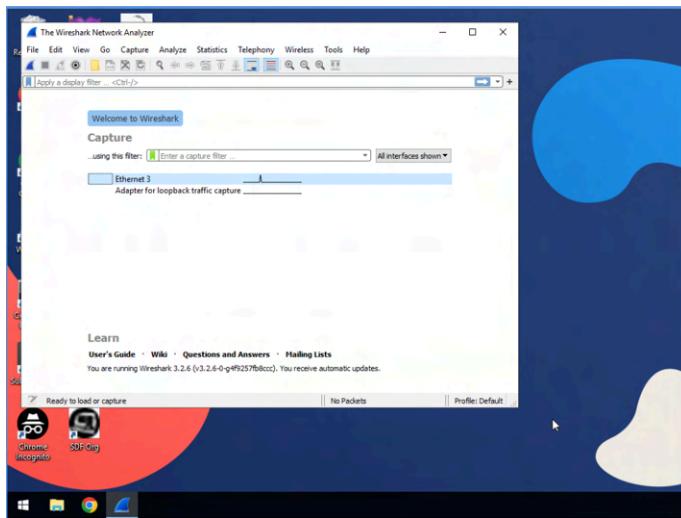
## **Scope and Objectives**

This project will include multiple tasks and will contain guided project pictures to explain my work.

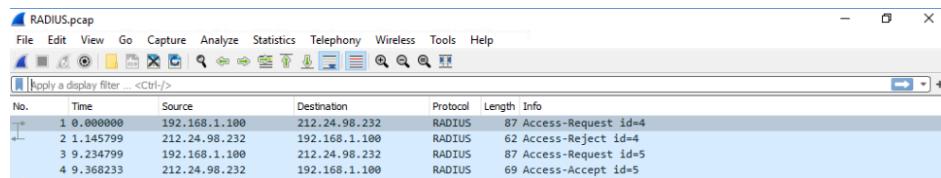
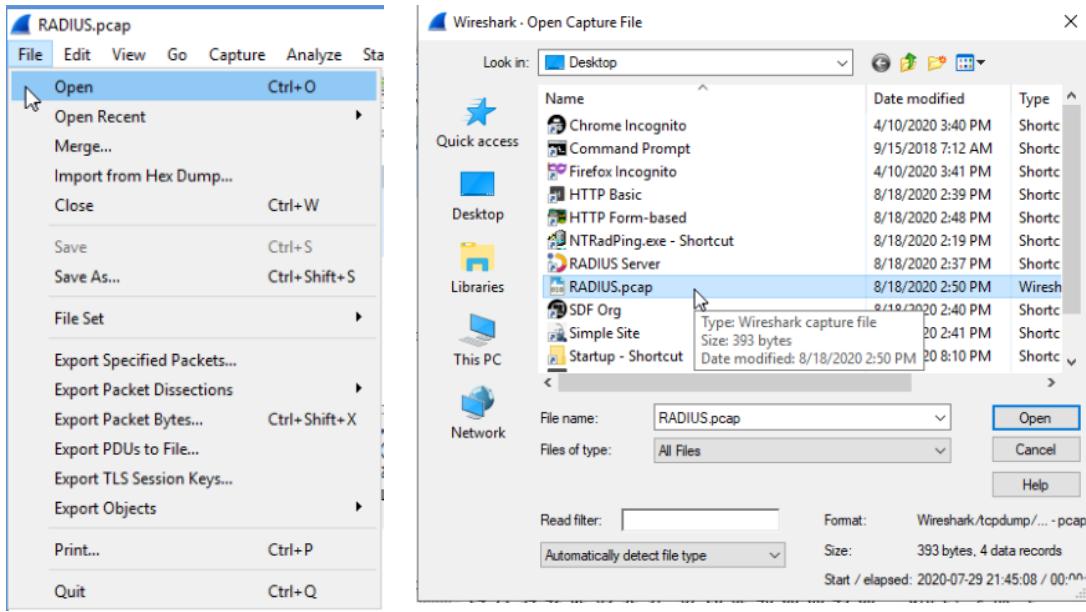
### **Tasks/Objectives:**

- Get to know Wireshark and its basic functions
- Generate and Capture RADIUS Traffic
- Analyze a HTTP Basic Authentication
- HTTP Form-Based Authentication and DNS
- Initiate, Capture and Analyze Telnet Sessions
- Capturing and Analyzing SSH Sessions
- Generate, Capture, analyze ten decrypt HTTPS Traffic

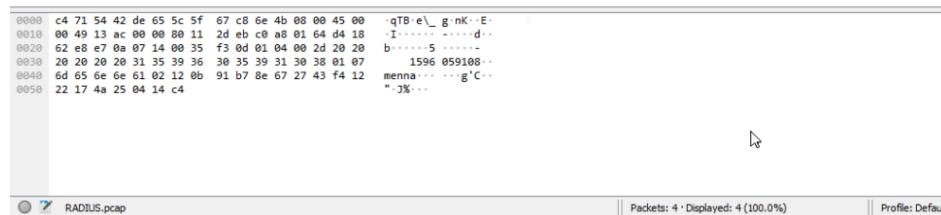
### **Task 1 - Get to know Wireshark and its basic functions**



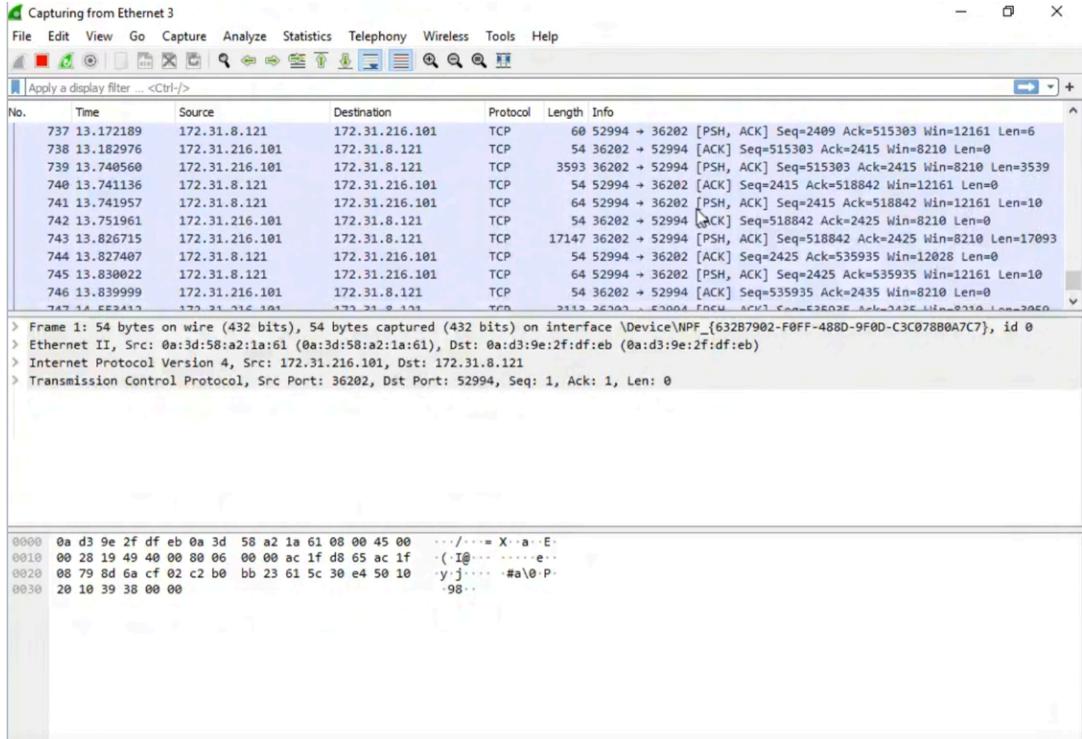
In this project, we will be using a Cloud workspace which is installed with a version of Windows 10. In here there are a couple programs already installed such as Wireshark, FireFox, Google Chrome, RADIUS server and a few others.



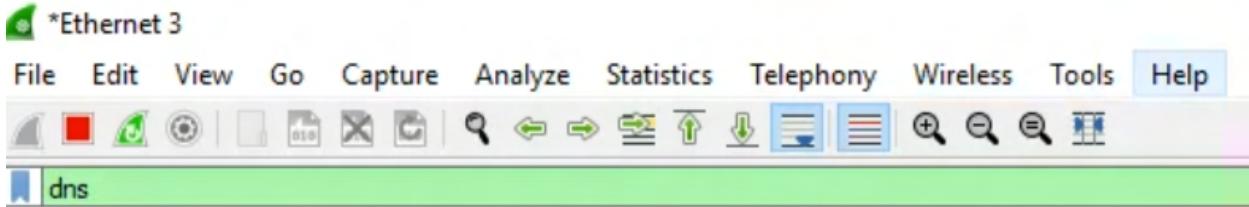
```
> Frame 1: 87 bytes on wire (696 bits), 87 bytes captured (696 bits)
> Ethernet II, Src: IntelCor_C8:6e:4b (5c:5f:67:c8:6e:4b), Dst: Tp-LinkT_42:de:65 (c4:71:54:42:de:65)
> Internet Protocol Version 4, Src: 192.168.1.100, Dst: 212.24.98.232
> User Datagram Protocol, Src Port: 59146, Dst Port: 1812
> RADIUS Protocol
```



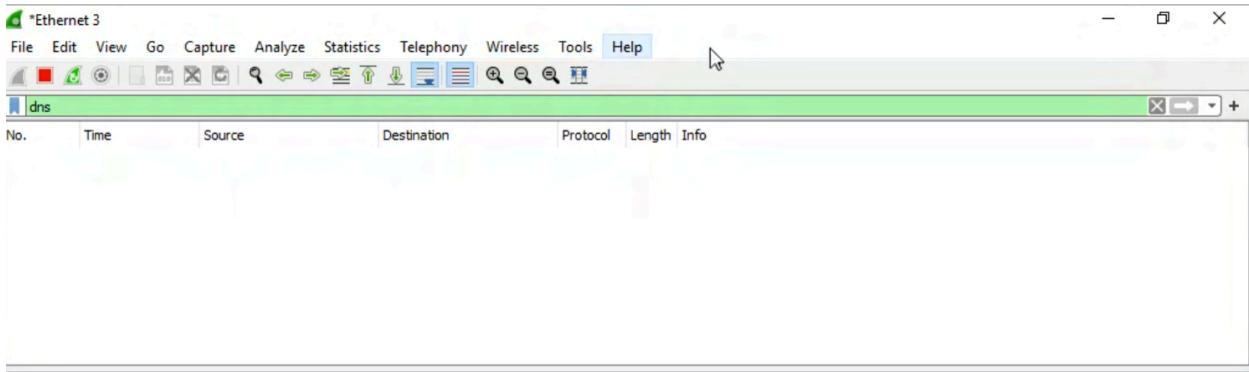
In the images above we are prompted to open a `.pcap` file from `RADIUS` on our desktop. To open this file on Wireshark we must first go into the file button on the top left hand corner. Once in “file” we should be able to navigate to open. Once the open application is available we must go into “desktop” and click on the `RADIUS.pcap` file which will open our desired file onto Wireshark.



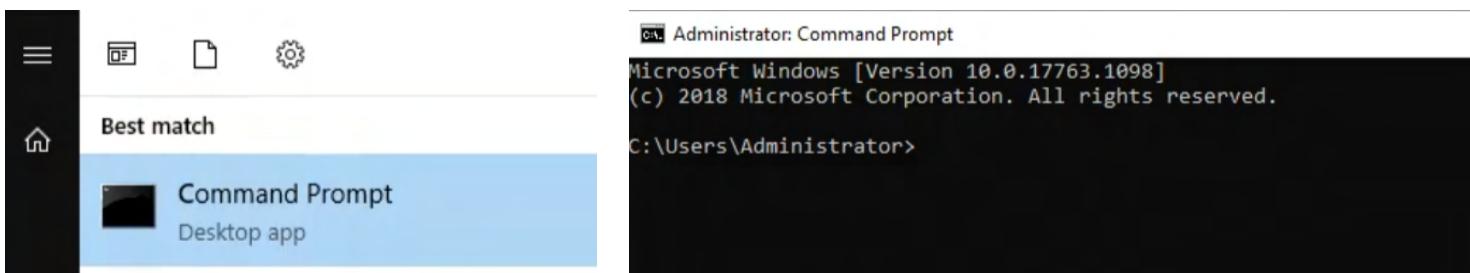
Next, we are prompted to open a live capture on Ethernet 3 network. We can double click on Ethernet 3 on the home page and it will bring up a live capture of internet traffic that is being captured at the moment.



To continue, we are asked to see if we are able to use the display filter box. For this exercise we are prompted to use “dns” as an input into the display filter. (As shown in the image above) Funnily enough, when inputting dns, there is no live capture happening which is why there wireshark is left blank. (See image below)



We can try to bring in some captures by pinging (Coursera.com)  
First you must open cmd prompt



```

Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>

Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>ping www.coursera.com

Pinging vpc-redirector-2039330151.us-east-1.elb.amazonaws.com [34.238.10.10] with 32 bytes of data:
Request timed out.

Ping statistics for 34.238.10.10:
    Packets: Sent = 1, Received = 0, Lost = 1 (100% loss),
Control-C
^C
C:\Users\Administrator>

```

After inputting the command `ping www.coursera.com` we are able to see two new captures on Wireshark. (Image Below)

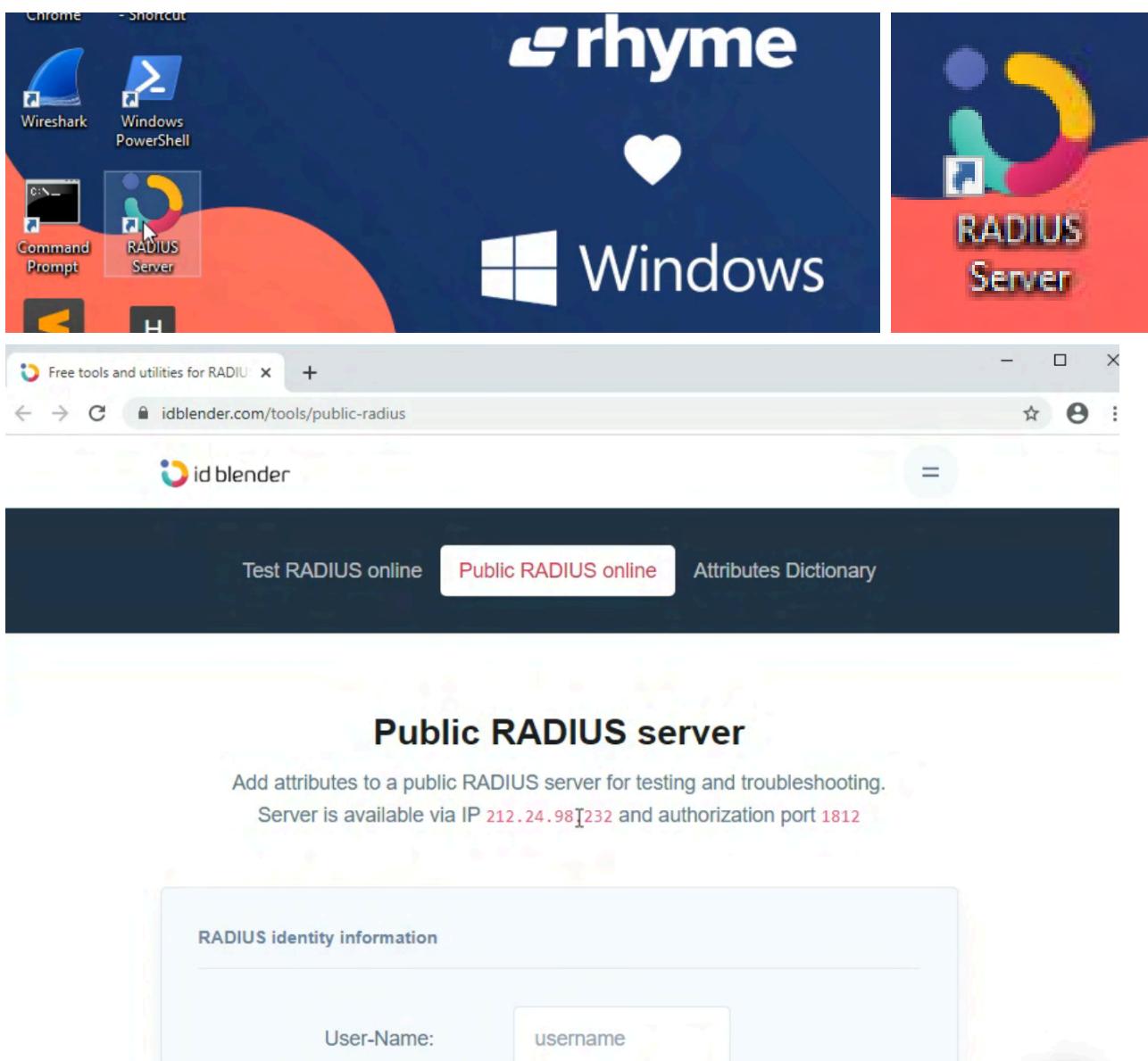
No.	Time	Source	Destination	Protocol	Length	Info
3197	107.135.81.10	172.31.216.101	172.31.0.2	DNS	76	Standard query 0x59a8 A www.coursera.com
3200	107.135.81.10	172.31.216.101	172.31.0.2	DNS	172	Standard query response 0x59a8 A www.coursera.com CNAME vpc-redi...

> Frame 3197: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface \Device\NPF\_{632B7902-F0FF-488D-9F0D-C3C078B0A7C7}, id 0  
> Ethernet II, Src: 0a:3d:58:a2:1a:61 (0a:3d:58:a2:1a:61), Dst: 0a:d3:9e:2f:df:eb (0a:d3:9e:2f:df:eb)  
> Internet Protocol Version 4, Src: 172.31.216.101, Dst: 172.31.0.2  
> User Datagram Protocol, Src Port: 58860, Dst Port: 53  
> Domain Name System (query)

0000 0a d3 9e 2f df eb 0a 3d 58 a2 1a 61 08 00 45 00 .../...= X-a-E-  
0010 00 3e 28 8e 00 00 80 11 00 00 ac 1f d8 65 ac 1f >(.....-e...  
0020 00 02 e5 ec 00 35 00 2a 30 e2 59 a8 01 00 00 01 ....5-\* 0-Y-....  
0030 00 00 00 00 00 00 03 77 77 77 08 63 6f 75 72 73 .....w ww cours  
0040 65 72 61 03 63 6f 6d 00 00 01 00 01 00 00 00 00 era.com .....

With this, we can conclude our first task, which is getting to know Wireshark and its basic features.

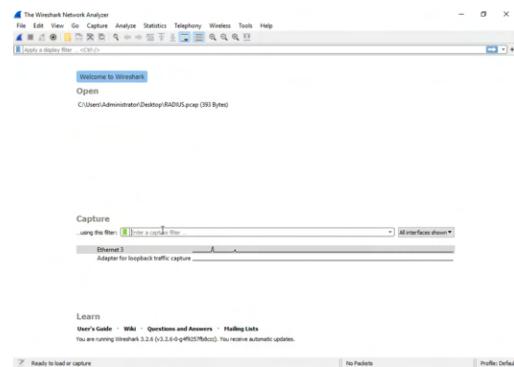
## Task 2 - Generate and Capture RADIUS Traffic



First we are going to open the RADIUS server. Doing so will open up a browser with RADIUS GUI Home page. We can note that the Server operates on the IP of 212.24.98.232 and Port of 1812.

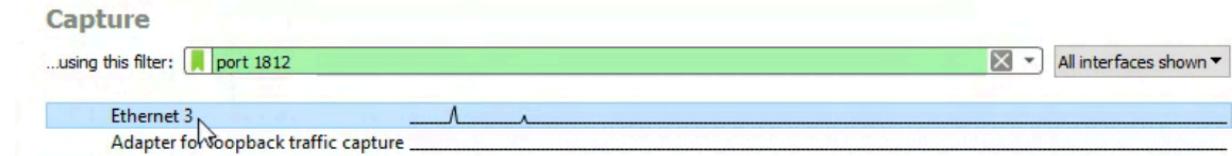
At the Bottom you are able to create your RADIUS server account - For this project, I chose to Use a generic username and password.

Before clicking the Submit button, we should first head back into Wireshark.

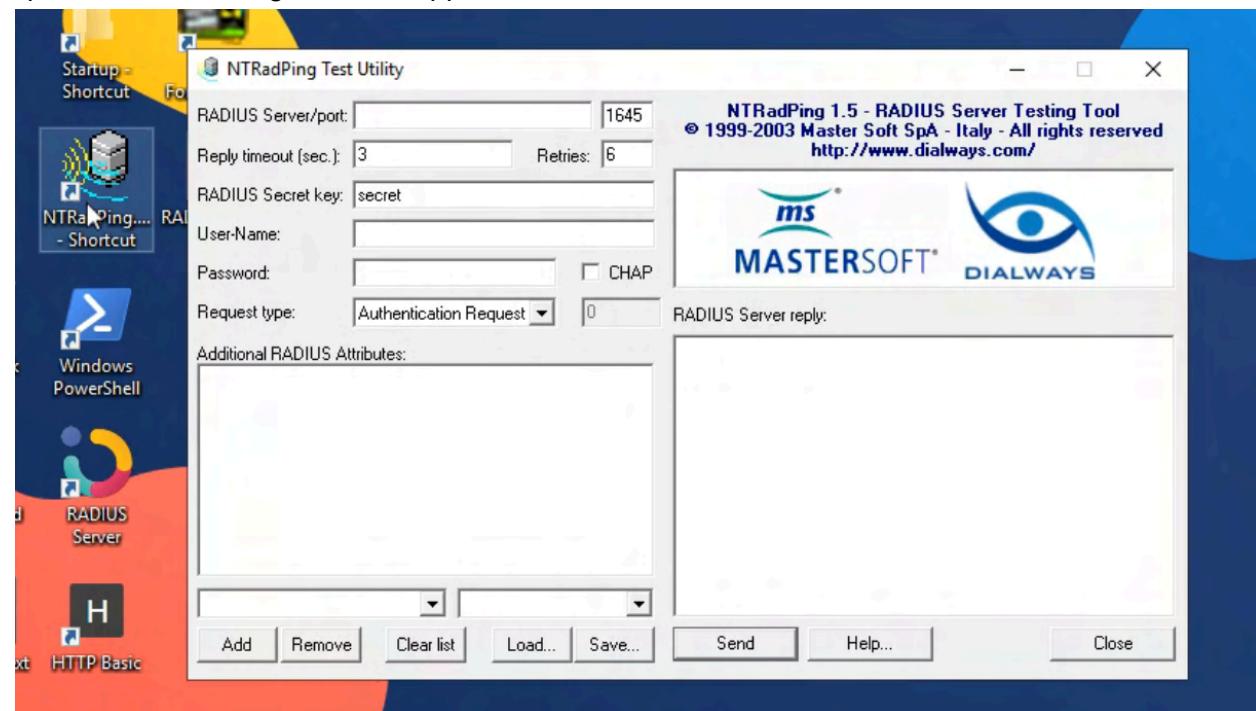


While in Wireshark we should be able to capture ONLY the RADIUS network traffic. How can we do this? We would need to use a filter before capturing network traffic. As stated previously we can see that the RADIUS server operates on port 1812.

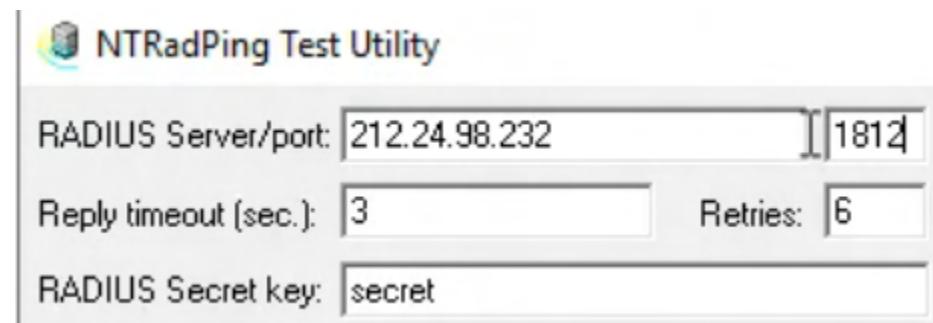
We can use this information to input the filter `port == 1812 or port 1812`



However, there is a second step you must follow before capturing RADIUS traffic. We must open the NTRadPing Shortcut/Application which would look like this:



This is what we will use to send out an authentication request to the public server we have created the user on.



Here at the top we have to input the RADIUS Server and the port number. We will leave the reply timeout and retries to default (3) and (6)

We will also leave RADIUS secret key to secret

User-Name:	<input type="text"/>
Password:	<input type="password"/> <input type="checkbox"/> CHAP
Request type:	Authentication Request <input type="button" value="0"/>

Moving forward - you may input the Username and Password that was created within the RADIUS home page GUI. For the project, I have chosen a generic username and password. For example - you could use "admin" for the username and "password" for the password.  
On the right hand side - we are able to see that the response for access has been accepted

#### RADIUS Server reply:

```
Sending authentication request to server 212.24.98.232:1812
Transmitting packet, code=1 id=0 length=45
received response from the server in 125 milliseconds
reply packet code=2 id=0 length=27
response: Access-Accept
----- attribute dump -----
```

If we were to put the incorrect password we will get an access rejected response shown here:

#### RADIUS Server reply:

```
Sending authentication request to server 212.24.98.232:1812
Transmitting packet, code=1 id=1 length=45
received response from the server in 1125 milliseconds
reply packet code=3 id=1 length=20
response: Access-Reject
----- attribute dump -----
```

With Wireshark running and activating RADIUS server ping we are now able to see entries within our network capture which looks like this:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.31.216.101	212.24.98.232	RADIUS	87	Access-Request id=0
2	0.122772	212.24.98.232	172.31.216.101	RADIUS	69	Access-Accept id=0
3	13.755753	172.31.216.101	212.24.98.232	RADIUS	87	Access-Request id=1
4	14.873483	212.24.98.232	172.31.216.101	RADIUS	62	Access-Reject id=1

\*Ethernet 3 (port 1812)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.31.216.101	212.24.98.232	RADIUS	87	Access-Request id=0
2	0.122772	212.24.98.232	172.31.216.101	RADIUS	69	Access-Accept id=0
3	13.755753	172.31.216.101	212.24.98.232	RADIUS	87	Access-Request id=1
4	14.873483	212.24.98.232	172.31.216.101	RADIUS	62	Access-Reject id=1

```
> Frame 1: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface \Device\NPF_{632B7902-F0FF-488D-9F0D-C3C078B0A7C7}, id 0
> Ethernet II, Src: 0a:3d:58:a2:1a:61 (0a:3d:58:a2:1a:61), Dst: 0a:d3:9e:2f:df:eb (0a:d3:9e:2f:df:eb)
> Internet Protocol Version 4, Src: 172.31.216.101, Dst: 212.24.98.232
> User Datagram Protocol, Src Port: 61767, Dst Port: 1812
> RADIUS Protocol
```

0000	0a d3 9e 2f df eb	0a 3d 58 a2 1a 61	08 00 45 00	.../.. = X..a..E..
0010	00 49 60 f9 00 00	80 11 00 00 ac 1f	d8 65 d4 18	.I.....e..
0020	62 e8 f1 47 07 14 00 35	bb cc 01 00 00 2d 20 20	b..G..5 .....	
0030	20 20 20 31 35 39 39	31 36 38 34 33 39 01 07	1599 168439..	

Notice how in the entries we are able to see two Access Requests. The first of which, which is highlighted, was accepted and the second request was rejected.

To dig even deeper, we can also view the authentication request username in the following field

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.31.216.101	212.24.98.232	RADIUS	87	Access-Request id=0
2	0.122772	212.24.98.232	172.31.216.101	RADIUS	69	Access-Accept id=0
3	13.755753	172.31.216.101	212.24.98.232	RADIUS	87	Access-Request id=1
4	14.873483	212.24.98.232	172.31.216.101	RADIUS	62	Access-Reject id=1

```
> Internet Protocol Version 4, Src: 172.31.216.101, Dst: 212.24.98.232
> User Datagram Protocol, Src Port: 61767, Dst Port: 1812
< RADIUS Protocol
  Code: Access-Request (1)
  Packet identifier: 0x0 (0)
  Length: 45
  Authenticator: 20202020202031353939313638343339
  [The response to this request is in frame 2]
< Attribute Value Pairs
```

Underneath the final entry, it would display the username we used to authenticate an access ping

For this demonstration I will not show the username used.

## Attribute Value Pairs

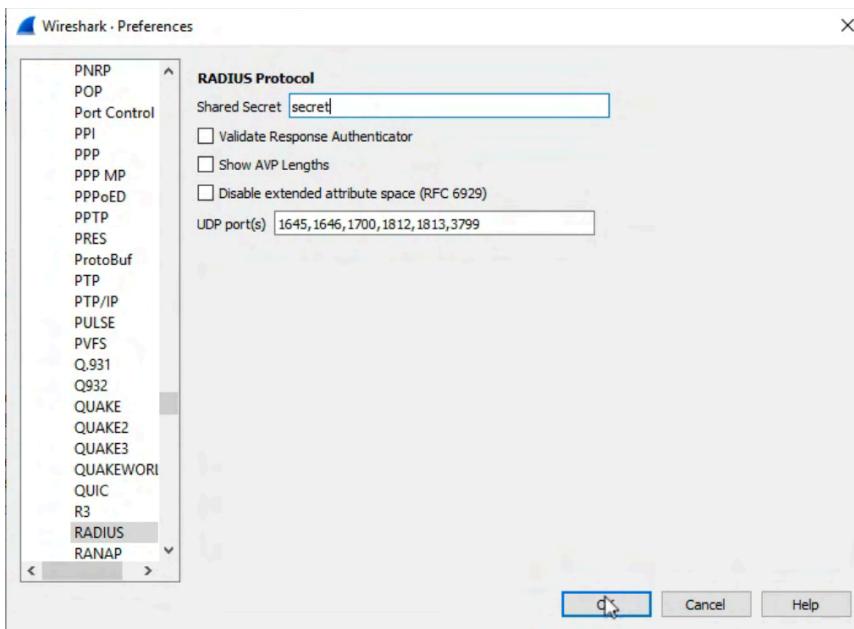
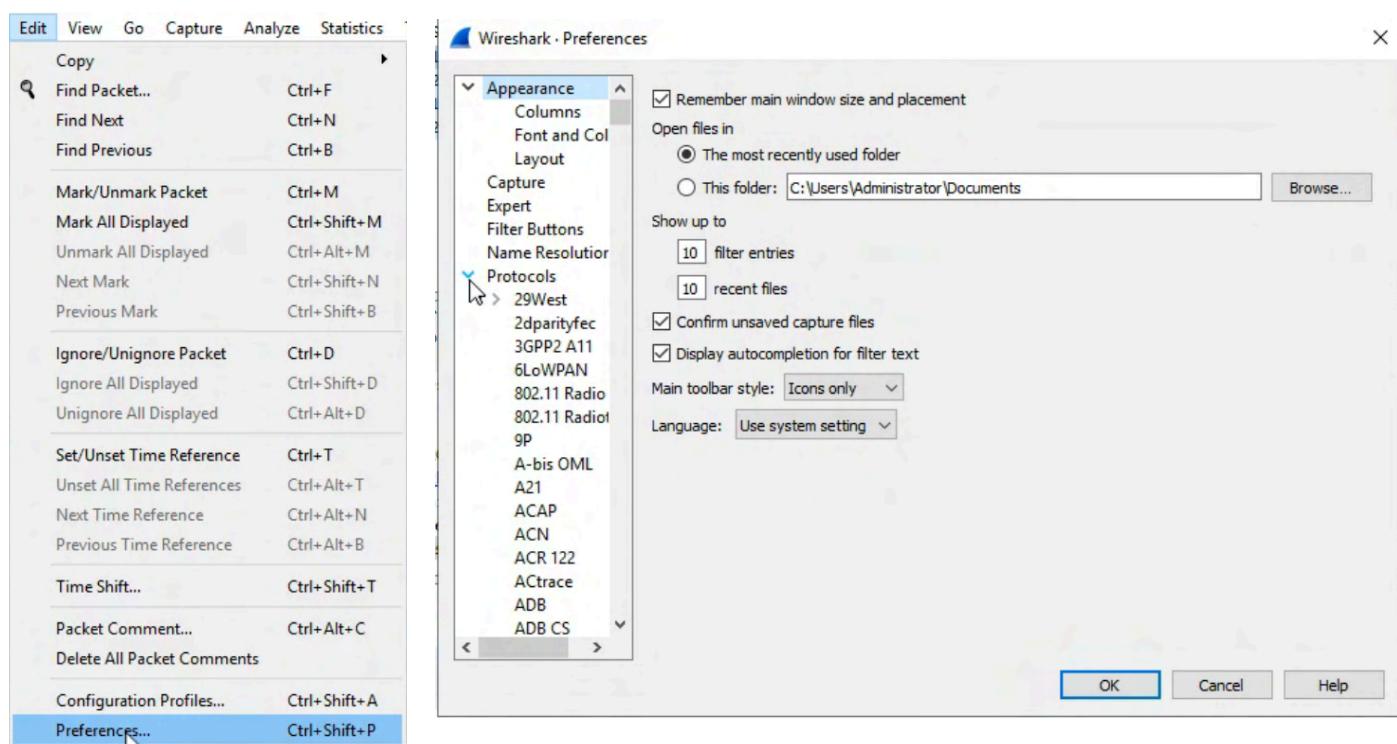
> AVP: t=User-Name(1) l=7 val=

Also along with this, there is an entry for password used however it is displayed as encrypted.

> AVP: t=User-Password(2) l=18 val=Encrypted

However, there is a way to display the password that was used. To do so: go into the edit button -> preferences -> protocols -> search for "radius"

Within shared secret, we will input the shared secret password. For this demonstration it was "secret". It will display the decrypted password after following these steps. Here is the photo demonstration:



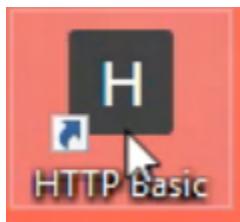
### **Task 3 - Analyze a HTTP Basic Authentication**

In this task, I am going to demonstrate Wireshark's capability to analyze HTTP traffic and dive into Basic Authentication

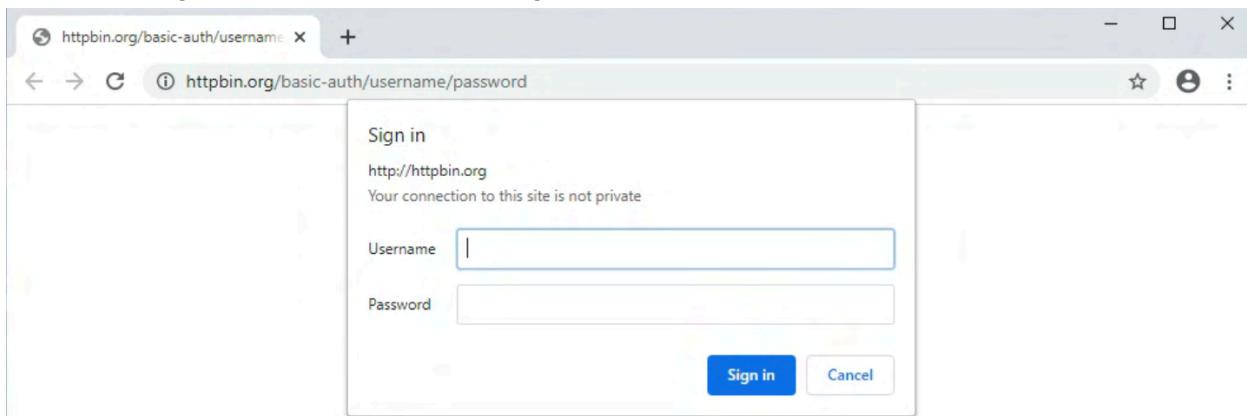
As we are informed, HTTP traffic is unencrypted. We may notice how sometimes when using a web browser, Microsoft's Windows Defender will warn their users when clicking into an HTTP link which may be insecure. For this task, we will be following a couple of steps to identify authentication traffic between a HOST to a HTTP server/website.

[Good things to know: HTTP = Port 80 - HTTPS = Port 443]

To begin, we can head back to our project workspace and click on the HTTP Basic Icon:



When opening this application, it will bring us to the website on our browser:



Note that for this demonstration our username and password is “username” and “password”  
(Very insecure **DO NOT DO THIS FOR ANY PERSONAL LOGINS**)

Before entering our credentials, we should start out Wireshark Capture.

Since we are looking for only HTTP traffic, we can apply a filter before capturing.

We may use either: `port == 80`, `port = 80`, or `port 80`. All will do the same exact thing.

Our first demonstration will begin with the us entering the incorrect password to our login credentials.

A screenshot of a browser window showing a failed sign-in attempt. The 'Sign in' form for 'http://httpbin.org' has 'username' entered in the Username field and '....' entered in the Password field. The 'Sign in' button is highlighted with a cursor, indicating it is about to be clicked.

Here, we have used the correct username however, we used the wrong password. We have inputted: 1234

Sign in

<http://httpbin.org>

Your connection to this site is not private

Username

Password

Sign in Cancel

http://httpbin.org/basic-auth/username

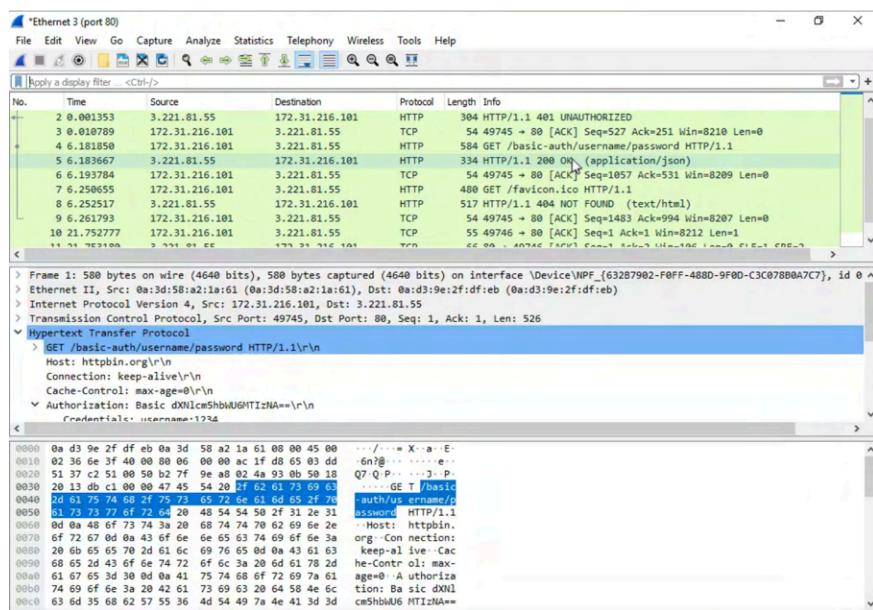
```
{ "authenticated": true, "user": "username" }
```

When entering the correct username and password, we are granted access to the webpage. Here in the final photo we can see that the authentication is displayed in the home page.

Lets head back to Wireshark.

As you can see on the right hand side, we have stopped the capture and is met with multiple captures.

Please notice how all the traffic is unencrypted. All credentials can be shown and everything we do can be shown and read.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.31.216.101	3.221.81.55	HTTP	580	GET /basic-auth/username/password HTTP/1.1
2	0.001353	3.221.81.55	172.31.216.101	HTTP	304	HTTP/1.1 401 UNAUTHORIZED

With our first entry we can see that we have sent out a authentication request and it was denied due to our failed attempt using the wrong password.

4	6.181850	172.31.216.101	3.221.81.55	HTTP	584	GET /basic-auth/username/password HTTP/1.1
5	6.183667	3.221.81.55	172.31.216.101	HTTP	334	HTTP/1.1 200 OK (application/json)

However, on our second attempt we can see that the authentication was accepted. Which is displayed with the `HTTP/1.1 200 OK`

This is very insecure. If we were to use this same procedure on an HTTPS server we would not have been able to see what has happened.

To further emphasize how insecure this is, we can click back onto the first entry and expand the Hypertext Transfer Protocol and the Authorization tab.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.31.216.101	3.221.81.55	HTTP	580	GET /basic-auth/username/password HTTP/1.1
2	0.001353	3.221.81.55	172.31.216.101	HTTP	304	HTTP/1.1 401 UNAUTHORIZED
3	0.010789	172.31.216.101	3.221.81.55	TCP	54	49745 → 80 [ACK] Seq=527 Ack=251 Win=8210 Len=0
4	6.181850	172.31.216.101	3.221.81.55	HTTP	584	584 GET /basic-auth/username/password HTTP/1.1
5	6.183667	3.221.81.55	172.31.216.101	HTTP	334	HTTP/1.1 200 OK (application/json)
6	6.193784	172.31.216.101	3.221.81.55	TCP	54	49745 → 80 [ACK] Seq=1057 Ack=531 Win=8209 Len=0
7	6.250655	172.31.216.101	3.221.81.55	HTTP	480	480 GET /favicon.ico HTTP/1.1
8	6.252517	3.221.81.55	172.31.216.101	HTTP	517	517 HTTP/1.1 404 NOT FOUND (text/html)
9	6.261793	172.31.216.101	3.221.81.55	TCP	54	49745 → 80 [ACK] Seq=1483 Ack=994 Win=8207 Len=0
10	6.262777	172.31.216.101	3.221.81.55	TCP	55	49746 → 80 [ACK] Seq=1 Ack=1 Win=8212 Len=1

```

> Frame 1: 580 bytes on wire (4640 bits), 580 bytes captured (4640 bits) on interface \Device\NPF_{632B7902-F0FF-488D-9F0D-C3C078B0A7C7}, id 0 
> Ethernet II, Src: 0a:3d:58:a2:1a:61 (0a:3d:58:a2:1a:61), Dst: 0a:d3:9e:2f:df:eb (0a:d3:9e:2f:df:eb)
> Internet Protocol Version 4, Src: 172.31.216.101, Dst: 3.221.81.55
> Transmission Control Protocol, Src Port: 49745, Dst Port: 80, Seq: 1, Ack: 1, Len: 526
└> Hypertext Transfer Protocol
   └> GET /basic-auth/username/password HTTP/1.1\r\n
      Host: httpbin.org\r\n
      Connection: keep-alive\r\n
      Cache-Control: max-age=0\r\n
      Authorization: Basic dXNlcjIyMjIwMTIzNA==\r\n
                     Credentials: username:1234
   └> Upgrade-Insecure-Requests: 1\r\n
   └> User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.83 Safari/537.36\r\n
   └> Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3
   └> Accept-Encoding: gzip, deflate\r\n
   └> Accept-Language: en-US,en;q=0.9\r\n
  
```

When doing so you can see the credentials and the password that was used for the first attempt.

Right next Authorization we can see the Basic encoding, which is not encrypted, of the username and password that is being sent to the server.

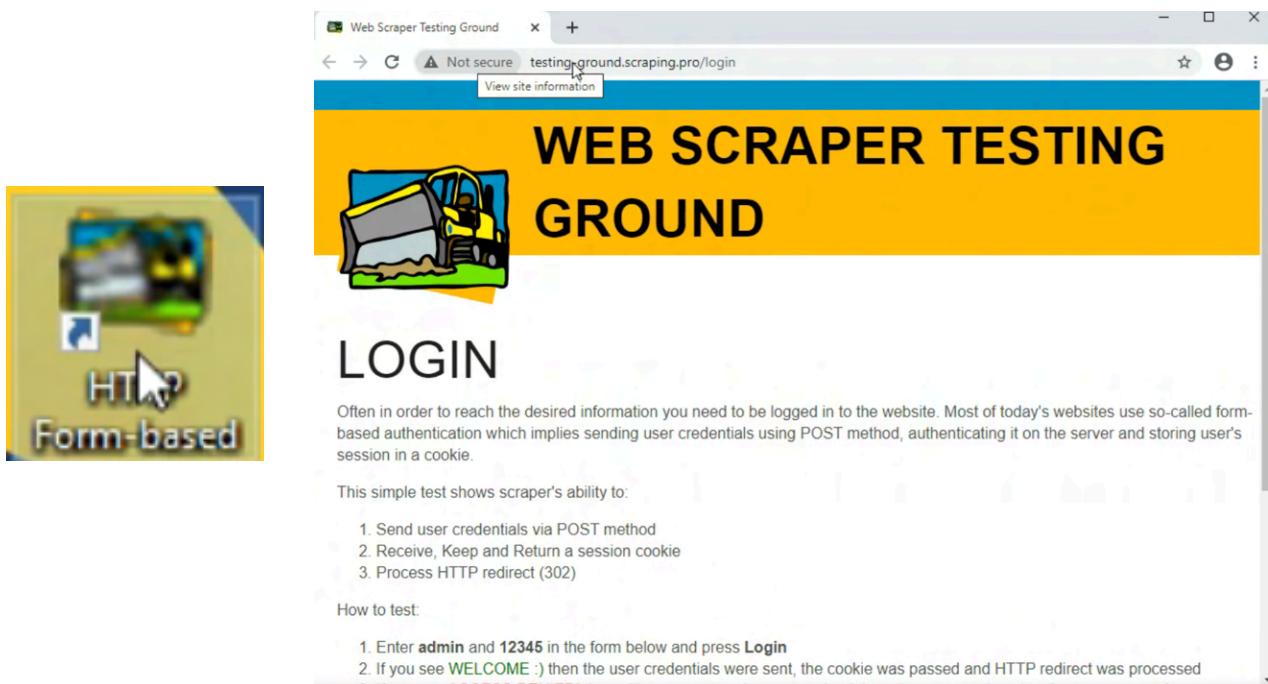
When clicking onto the entry with the correct credentials:

```
Host: httpbin.org\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=\r\n
    Credentials: username:password
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.83 Safari/537.36\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: en-US,en;q=0.9\r\n
```

We can see that correct username and password, along with the basic encoding that was sent to the server

As a threat actor, we can assume that HTTP is very easy to swipe someone's credentials and create a very dangerous situation to both personal/citizen or to an organization

## **Task 4 - HTTP Form-Based Authentication and DNS**



Form based authentication which was created to mitigate the downsides of basic authentication. Form based authentication uses the standard HTML form fields to pass the username and password values to the server. This does not send a get request with the credentials, instead it will send a post request and will pass the request in the post body.

In the images above, we are opening the HTTP Form-based shortcut which will open up a browser and a HTTP website. We can notice that the website is not secure as it uses HTTP. When scrolling down the page we can see instructions on how we can test this website and a credentials field. Shown here:

How to test:

1. Enter **admin** and **12345** in the form below and press **Login**
2. If you see **WELCOME** :) then the user credentials were sent, the cookie was passed and HTTP redirect was processed
3. If you see **ACCESS DENIED!** then either you entered wrong credentials or they were not sent to the server properly
4. If you see **THE SESSION COOKIE IS MISSING OR HAS A WRONG VALUE!** then the user credentials were properly sent but the session cookie was not properly stored or passed
5. If you see **REDIRECTING...** then the user credentials were properly sent but HTTP redirection was not processed
6. Click **GO BACK** to start again



### Please, login:

User name:  Password:

Before inputting the credentials, we should begin the Wireshark capture. Using the filters before capture we can capture only HTTP traffic using the query we used before. **Port == 80**, **Port = 80**, **Port 80**

### Capture



After enabling the capture, we can go back to our test website.

We can now input the credentials: Username = **admin** and Password = **12345**

### Please, login:

User name:  Password:

Once we have finished entering our credentials and clicking on the "login" button we can stop Wireshark capture.

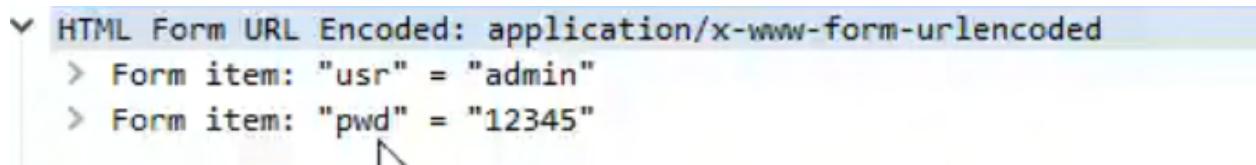
In our previous task, we can note that we were able to find the credentials within the basic authentication inside the hypertext transfer protocol. However, this time we are able to find this using the HTML Form Fields.

The screenshot shows the Wireshark interface with the following details:

- File Bar:** File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help
- Capture Bar:** Ethernet 3 (port 80)
- Toolbar:** Standard icons for opening files, saving, zooming, and analysis.
- Frame List:** Shows a list of 10 network frames. Frame 5 is selected, showing a POST request to the URL /login?mode=login.
- Frame Details:** Displays the raw bytes (hex and ASCII) of the selected frame, which corresponds to the POST request.
- Frame Bytes:** Displays the raw bytes of the selected frame.
- Frame Error:** Shows the error status of the selected frame.
- Frame Info:** Provides detailed information about the selected frame, including source and destination addresses, protocol (HTTP), and specific fields like 'Seq=0 Win=64240 Len=0 MSS=1460'.
- Status Bar:** Shows the status message: "Frame 5: 725 bytes on wire (5800 bits), 725 bytes captured (5800 bits) on interface \Device\NPF\_{632B7902-F0FF-488D-9F0D-C3C078B0A7C7}, id 0".
- Bottom Status:** Shows the protocol stack: Ethernet II, Src: 0a:3d:58:a2:1a:61 (0a:3d:58:a2:1a:61), Dst: 0a:d3:9e:2f:df:eb (0a:d3:9e:2f:df:eb).
- Bottom Bar:** Hypertext Transfer Protocol and HTML Form URL Encoded: application/x-www-form-urlencoded.

In the image above, it is shown that we have clicked on the HTTP protocol, length of 725. This is the capture entry of our authentication request. We can see that in the bottom there is the “Hypertext transfer Protocol Drop” down menu as well as the “HTML Form URL Encoded” drop down.

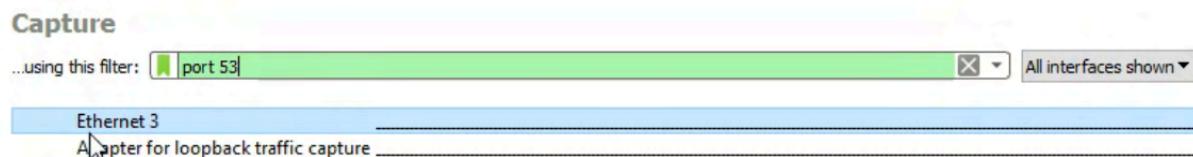
When expanding the HTML Form URL Encoded, we can see the form item of both the username and password:



While this procedure is even less secure than HTTP, HTML Form was developed with the case of HTTPS in mind. This is because the entire packet would already be encrypted anyway so there was no need for worrying about any encoding.

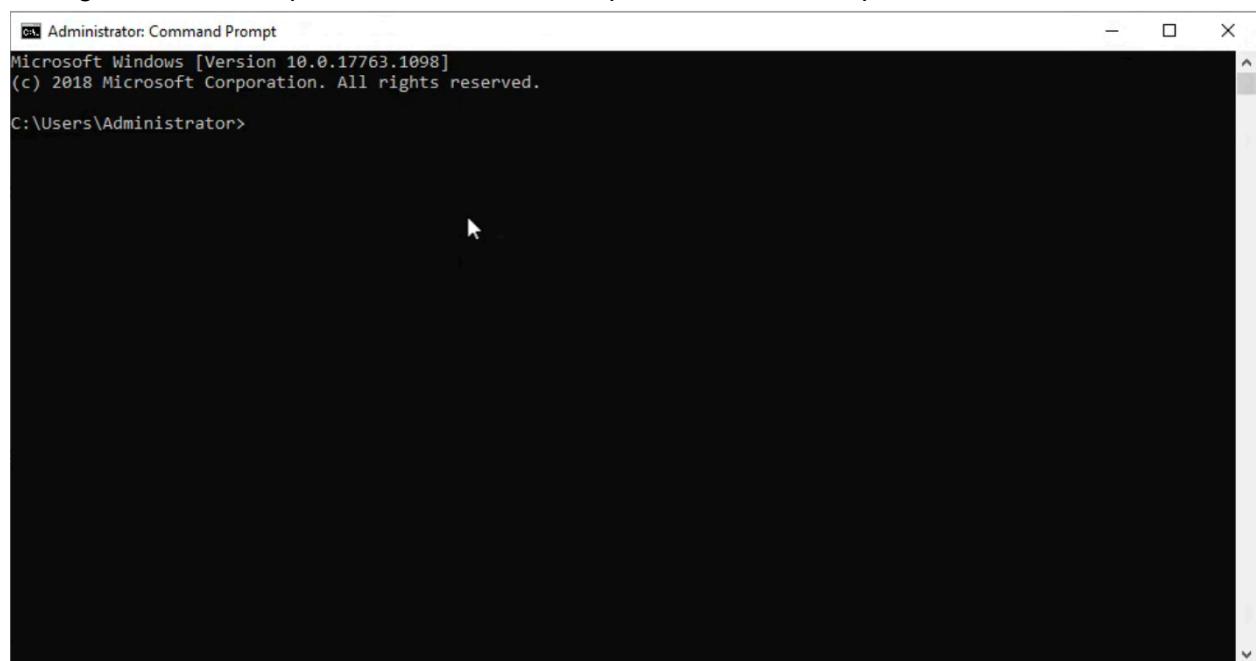
This is a great reminder to never input important/sensitive information online that is using HTTP. There are probably 3rd party games, pirating websites, even pirated streaming platforms that would require user credentials which can be insecure due to the nature of HTTP.

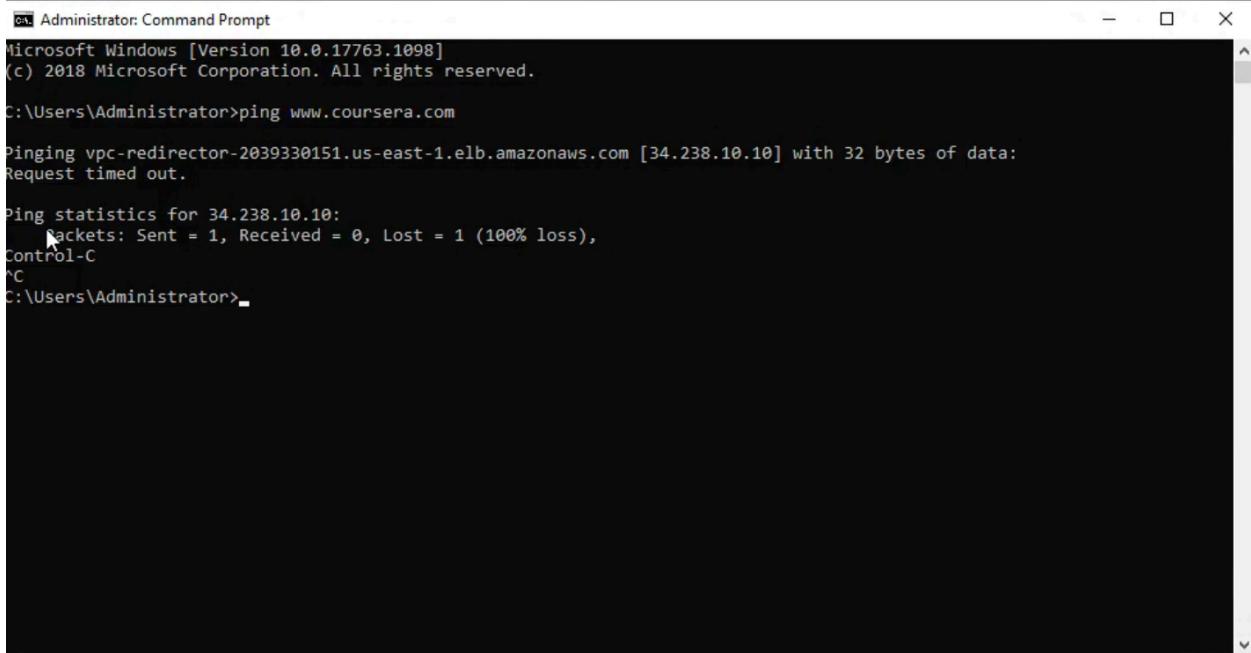
To continue this task, we can begin DNS traffic capture.



Notice how we are now jumping to port 53 for DNS traffic.

To begin, we should open the Command Prompt within our desktop.





```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

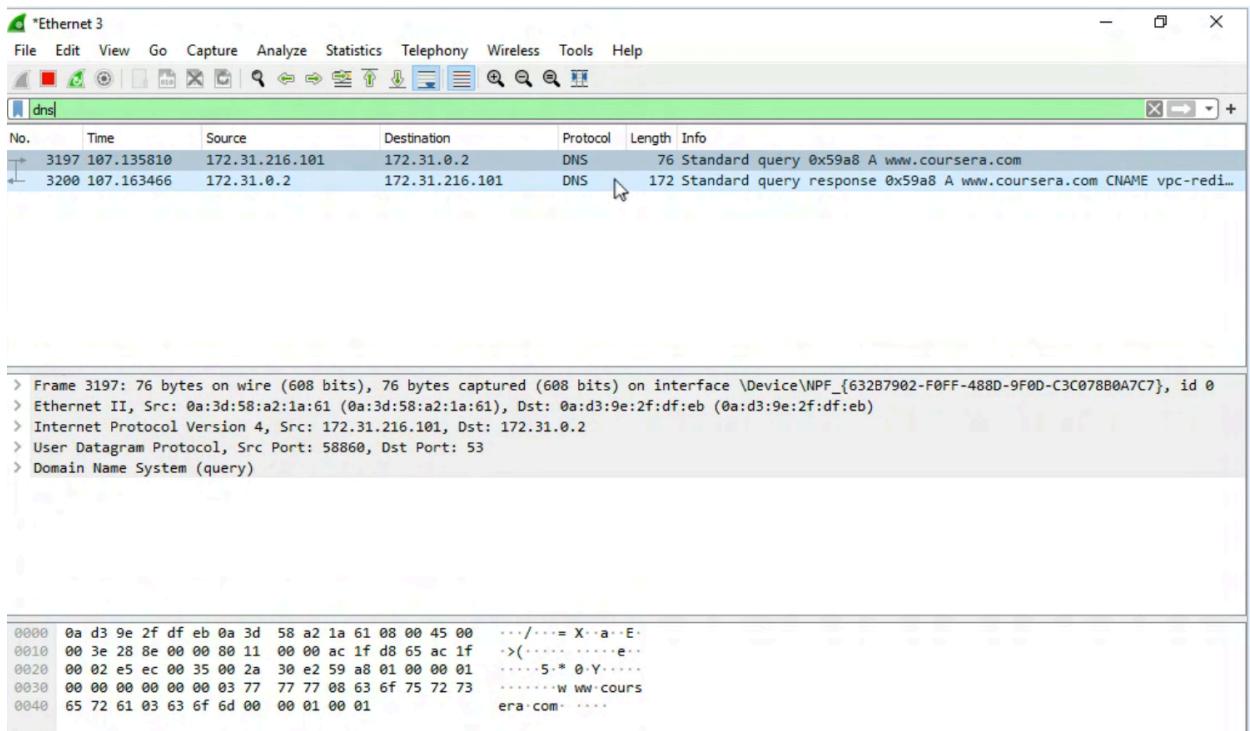
C:\Users\Administrator>ping www.coursera.com

Pinging vpc-redirector-2039330151.us-east-1.elb.amazonaws.com [34.238.10.10] with 32 bytes of data:
Request timed out.

Ping statistics for 34.238.10.10:
    Packets: Sent = 1, Received = 0, Lost = 1 (100% loss),
Control-C
^C
C:\Users\Administrator>
```

In here we can input the command `ping www.coursera.com`

This will allow us to capture DNS Traffic on Wireshark



As shown above, we have inputted the display filter `dns` onto our Wireshark application which should only display our DNS ping.

### **Task 5 - Initiate, Capture and Analyze Telnet Sessions**

To further our understanding on unencrypted protocols, we will proceed with our 5th task which will revolve around Telnet. We will be experimenting with Telnet as well as comparing it with SSH in our next assignment. Telnet is a computer protocol that was built for interacting with remote devices to access and manage said device. In this use case, we will be using telnet to [tty.sdf.org](http://tty.sdf.org)

To begin we can open the shortcut on our desktop SDF Org

ssh://new@sdf.org', 'Web Browser users may use our HTML5 SSH client: <https://ssh.sdf.org>', and 'Linux/UNIX users can type 'ssh new@sdf.org' at their shell prompts.'. There is also a note for Microsoft Windows recommending Putty. A 'Make a Donation' button is located below the alternative methods section. At the bottom of the page, there is a copyright notice: '©1987-2065 SDF Public Access UNIX System, Inc. 501(c)(7)'." data-bbox="117 324 879 712"/>

We are presented with the credential login page for “Free UNIX shell Account”  
Telnet operates on TCP Port 23

#### **Capture**



To continue capture telnet packets we will be using windows Powershell

 Telnet tty.sdf.org

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> telnet tty.sdf.org
Connecting To tty.sdf.org...
```

When using, telnet in powershell we can also place the website url to the command which will connect our desktop with the corresponding URL.

You will then be greeted with:

```
sdf.lonestar.org (pts/0)
if new, login 'new' ..

login: ■
```

You will enter the Username and password for this session

Make sure that the password is case sensitive

When inputting the password there will not be a display of key inputs

```
Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005,
2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017,
2018, 2019 The NetBSD Foundation, Inc. All rights reserved.
Copyright (c) 1982, 1986, 1989, 1991, 1993
    The Regents of the University of California. All rights reserved.

[ 'menna' will expire in 628 days - Please 'validate' your account soon ]

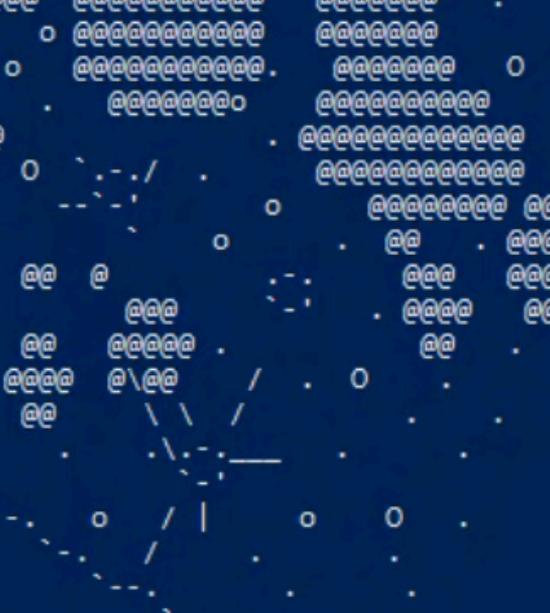
Please press your BACKSPACE key: ■
```

After inputting the correct credentials there will be a prompt to enter the backspace key

```
=====
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
-----
mx           up     3+18:07,    0 users, load 0.46, 0.46, 0.43
sdf          up     3+18:05,    6 users, load 0.11, 0.14, 0.15
rie          up     3+18:06,    3 users, load 0.00, 0.05, 0.06
vpn          up     87+15:16,   0 users, load 0.09, 0.09, 0.08
vps3         up     46+03:06,   0 users, load 2.00, 1.97, 1.65
miku         up     3+18:06,    0 users, load 0.31, 0.41, 0.34
otaku        up     3+18:05,    2 users, load 0.23, 0.29, 0.32
norge        up     3+18:05,    0 users, load 0.02, 0.01, 0.00
faeroes      up     3+18:07,    4 users, load 0.21, 0.20, 0.17
iceland       up     3+18:06,    2 users, load 0.16, 0.20, 0.23
anonradio    up     42+23:05,   0 users, load 0.20, 0.23, 0.20
(continue)■
```

The Moon is Waning Gibbous (84% of Full)



Full Moon +  
4 5:05:57  
Last Quarter -  
3 22:59:08

continue)  
ype 'help' for Commands.  
ype 'com' to chat with other users.  
ype 'ttytter' to listen to Twitter Tweets anonymously.  
ype 'mud' to play the SDFmud.  
ype 'mkhomepg' to set up your personal website.

If you know you can validate your account and gain weekend IRC access  
by making a donation of \$1 to \$3? Type 'validate' for more info!

With the images above, I have followed the prompt commands to continue with the telnet command. In the final image we are able to see a graphic of the moon. This will conclude our Wireshark capture.

\*Ethernet 3 (port 23)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
88	23.796973	205.166.94.8	172.31.216.101	TELNET	198	Telnet Data ...
89	23.836971	172.31.216.101	205.166.94.8	TCP	54	49751 → 23 [ACK] Seq=109 Ack=1874 Win=262656 Len=0
90	24.855210	172.31.216.101	205.166.94.8	TELNET	56	Telnet Data ...
91	24.933419	205.166.94.8	172.31.216.101	TELNET	60	Telnet Data ...
92	24.975030	172.31.216.101	205.166.94.8	TCP	54	49751 → 23 [ACK] Seq=111 Ack=1878 Win=262400 Len=0
93	25.053390	205.166.94.8	172.31.216.101	TELNET	1442	Telnet Data ...
94	25.093882	172.31.216.101	205.166.94.8	TCP	54	49751 → 23 [ACK] Seq=111 Ack=3266 Win=262656 Len=0
95	36.401045	172.31.216.101	205.166.94.8	TELNET	55	Telnet Data ...
96	36.479024	205.166.94.8	172.31.216.101	TELNET	60	Telnet Data ...
97	36.570027	172.31.216.101	205.166.94.8	TCP	54	49751 → 23 [ACK] Seq=112 Ack=3272 Win=262656 Len=0

Acknowledgment number: 0  
Acknowledgment number (raw): 0  
1000 .... = Header Length: 32 bytes (8)

Flags: 0x0c2 (SYN, ECN, CWR)

- 000. .... .... = Reserved: Not set
- ...0 .... .... = Nonce: Not set
- .... 1.... .... = Congestion Window Reduced (CWR): Set
- .... .1.... = ECN-Echo: Set
- .... ..0.... = Urgent: Not set
- .... ..0.... = Acknowledgment: Not set
- .... .... 0... = Push: Not set
- .... .... 0... = Reset: Not set
- .... .... 0... = Svn: Not set

Let's analyze the data. Familiarize yourself with the way that the network application chooses their transport layer protocol whether TCP or UDP.

Here we can see that Telnet needs to use TCP to send keystrokes to the remote server in real time.

Usually real time applications uses UDP, however Telnet requires traffic to be sent reliably using a trick that would send all traffic in a NEAR real time traffic.

4	0.177031	205.166.94.8	172.31.216.101	TELNET	60	Telnet Data ...
5	0.177212	172.31.216.101	205.166.94.8	TELNET	57	Telnet Data ...
6	0.255140	205.166.94.8	172.31.216.101	TELNET	62	Telnet Data ...
7	0.255289	172.31.216.101	205.166.94.8	TELNET	62	Telnet Data ...
8	0.333407	205.166.94.8	172.31.216.101	TELNET	72	Telnet Data ...
9	0.333771	172.31.216.101	205.166.94.8	TELNET	57	Telnet Data ...
10	0.507224	205.166.94.8	172.31.216.101	TCP	54	49751 → 23 [ACK] Seq=112 Ack=3272 Win=262656 Len=0

> Frame 4: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF\_{632B7902-F0FF-488D-9F0D-C3C078B0A7C7}, id 0

> Ethernet II, Src: 0:a:d3:9e:2f:df:eb (0:a:d3:9e:2f:df:eb), Dst: 0:a:3d:58:a2:1a:61 (0:a:3d:58:a2:1a:61)

> Internet Protocol Version 4, Src: 205.166.94.8, Dst: 172.31.216.101

Transmission Control Protocol, Src Port: 23, Dst Port: 49751, Seq: 1, Ack: 1, Len: 3

Source Port: 23  
Destination Port: 49751  
[Stream index: 0]  
[TCP Segment Len: 3]  
Sequence number: 1 (relative sequence number)  
Sequence number (raw): 1184891518  
[Next sequence number: 4 (relative sequence number)]  
Acknowledgement number: 1 (relative ack number)

Flags: 0x018 (PSH, ACK)

- 000. .... .... = Reserved: Not set
- ...0 .... .... = Nonce: Not set
- .... 0.... .... = Congestion Window Reduced (CWR): Not set
- .... .0.... .... = ECN-Echo: Not set
- .... ..0.... = Urgent: Not set
- .... .... 1.... = Acknowledgment: Set
- .... .... 1.... = Push: Set
- .... .... .0... = Reset: Not set
- .... .... 0... = Svn: Not set

We can view how this works by analyzing one of the capture entries. On Wireshark we can see entry number 4 has been selected. Expanding both the Transmission Control Protocol and the Flags drop down menu, we can see that the Push flag is set to 1. This means that whenever any character comes from Telnet it will just send it. It will not wait as TCP combines a whole PDU.

Acknowledgment number: 1 (relative ack number)  
 Acknowledgment number (raw): 1209052425  
 0101 .... = Header Length: 20 bytes (5)  
 ✓ Flags: 0x018 (PSH, ACK)  
 000. .... .... = Reserved: Not set  
 ...0 .... .... = Nonce: Not set  
 .... 0.... .... = Congestion Window Reduced (CWR): Not set  
 .... .0.... .... = ECN-Echo: Not set  
 .... ..0.... .... = Urgent: Not set  
 .... ...1.... .... = Acknowledgment  
 .... .... 1.... .... = Push: Set  
 .... .... .... 0.... .... = Reset: Not set

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.31.216.101	205.166.94.8	TCP	66	49751 → 23 [SYN, ECN, CWR] Seq=0 Win=26
2	0.078065	205.166.94.8	172.31.216.101	TCP	66	23 → 49751 [SYN, ACK] Seq=0 Ack=1 Win=26
3	0.078134	172.31.216.101	205.166.94.8	TCP	54	49751 → 23 [ACK] Seq=1 Ack=1 Win=26
4	0.177031	205.166.94.8	172.31.216.101	TELNET	1	Follow
5	0.177212	172.31.216.101	205.166.94.8	TELNET	1	Mark/Unmark Packet Ctrl+M
6	0.255140	205.166.94.8	172.31.216.101	TELNET	1	Ignore/Unignore Packet Ctrl+D
7	0.255289	172.31.216.101	205.166.94.8	TELNET	1	Set/Unset Time Reference Ctrl+T
8	0.333407	205.166.94.8	172.31.216.101	TELNET	1	Time Shift... Ctrl+Shift+T
9	0.333771	172.31.216.101	205.166.94.8	TELNET	1	Packet Comment... Ctrl+Alt+C
10	0.507224	205.166.94.8	172.31.216.101	TCP	1	Edit Resolved Name

0000 0a 3d 58 a2 1a 61 0a d3 9e 2f df  
 0010 00 2b 00 00 40 00 20 06 aa 89 cd  
 0020 d8 65 00 17 c2 57 46 a0 02 7e 48  
 0030 10 65 c9 8b 00 00 ff fd 25 1a f8

TCP Stream Ctrl+Alt+Shift+T  
 UDP Stream Ctrl+Alt+Shift+U  
 TLS Stream Ctrl+Alt+Shift+S  
 HTTP Stream Ctrl+Alt+Shift+H  
 HTTP/2 Stream  
 QUIC Stream

We could also right click on the same entry and click on Follow -> TCP Stream

This will show everything we have seen in the Telnet session, which would mean that this is fully unencrypted and it is not secure.

The Moon is Waning Gibbous (84% of Full)

faeroes up 3+18:16, 4 users, load 0.41, 0.39, 0.28  
 iceland up 3+18:16, 2 users, load 0.22, 0.19, 0.28  
 anonradio up 42+23:15, 0 users, load 0.14, 0.19, 0.17  
 (continue)

(continues)

Type 'help' for Commands.  
 Type 'com' to chat with other users.  
 Type 'twtter' to listen to Twitter Tweets anonymously.  
 Type 'mud' to play the SDFMUD.  
 Type 'mkhomepg' to set up your personal website.

Did you know you can become a permanent LIFETIME member of SDF  
 by making a one-time donation of \$36? Type 'arca' for more info!

Packet 84. 34 client pkts, 25 server pkts, 34 turns. Click to select.

Entire conversation (3377 bytes) Show and save data as ASCII Stream 0

```
..%..%.....%.....&..... #...'$...$...#...'$...'$.....x.  
1.....ANSI.....".....!"....."  
sdf.lonestar.org (pts/0)  
if new, login 'new' ..
```

Here is a cool example of what Wireshark is capable of showing. Here we can see both a blue and red highlight. Each highlight means something. The red highlight represents what we send to the server. The blue highlight represents what the server spits back at us.

Telnet echoes back everything back on our screen. However when inputting the password the server does not spit the information back to us which is why the password is left blank when inputting a keystroke.

### **Task 6 - Capturing and Analyzing SSH Sessions**

In this task we will be investigating encrypted protocols and traffic. Encryption means that the data or plain text is encrypted with an encryption algorithm and encryption key

This process can be viewed in its original form only if it is encrypted with the correct key

Please note that this is the most simplest form of encryption so take it with a grain of salt

Previously we have analyzed data using Telnet, much like Telnet... SSH is used to remotely access and manage a device. The key difference is that SSH uses encryption which means that SSH encrypts all data transmitted over the network.

We should note that Telnet uses port 23 and SSH uses port 22

SSH also uses TLS which is the same protocol used by HTTPS to secure communication.

Please note that SSH is the predecessor of TLS which TLS being the newer standard and is more secure.

#### **Capture**



Notice how in the image above we are not capturing only SSH network packets. If we were to do so the capture filter would look something like this: port == 22

Instead we have inputted host tty.sdf.org to capture all packets regarding the host traffic of tty.sdf.org.

Before we move on, we should start capture and follow the same steps from Task 5 for telnet traffic. While finishing up telnet traffic, do not stop Wireshark Capture and move onto SSH Traffic

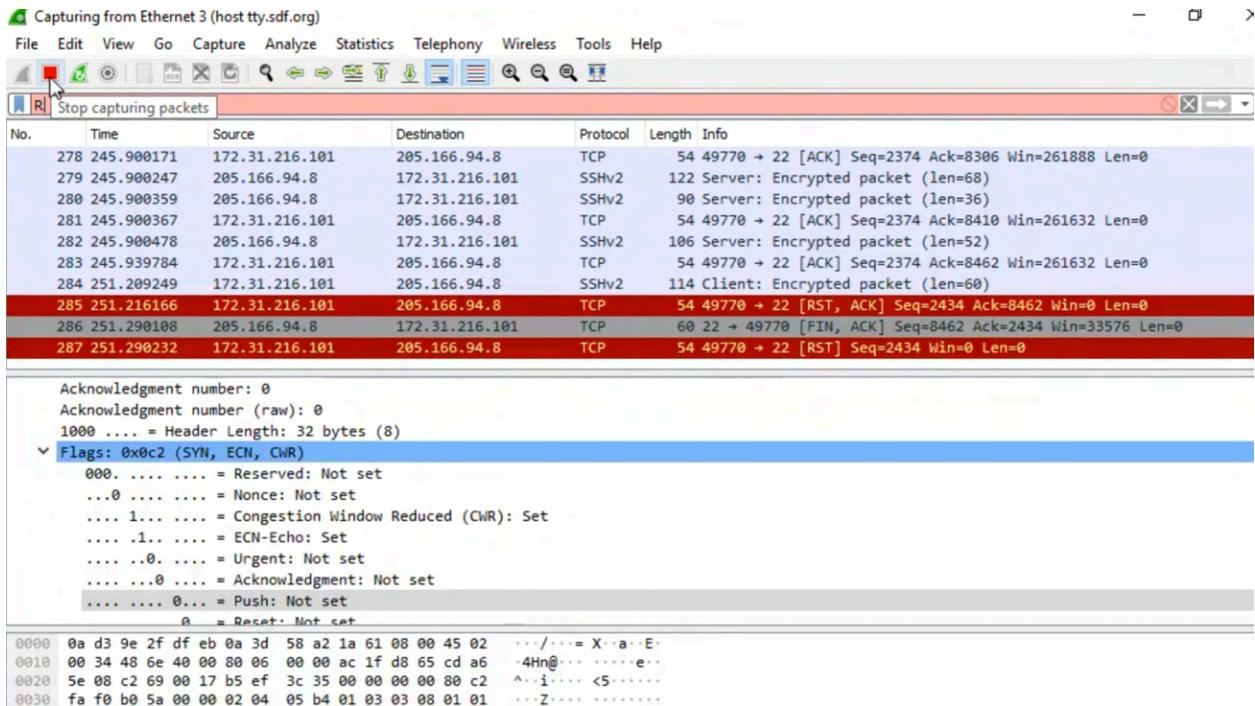
```
Administrator: Windows PowerShell  
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
PS C:\Users\Administrator> ssh [REDACTED]@tty.sdf.org  
The authenticity of host 'tty.sdf.org (205.166.94.8)' can't be established.  
ED25519 key fingerprint is SHA256:ZjwbO7AU8rHJExYrmZS2LqGZ7WfdoELfMrF54W92PYA.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'tty.sdf.org,205.166.94.8' (ED25519) to the list of known hosts.  
[REDACTED]@tty.sdf.org's password:
```

In the image above we have our first powershell command: ssh [username]@tty.sdf.org  
The blacked out box will be the username which you will input.

When inputting your password make sure to input it correctly - it is case sensitive

The same output will be present when using the SSH command.

We can now exit powershell and go back to Wireshark



When exiting powershell - there will be an error message on Wireshark that will be captured. This is not an actual error, rather it is SSH being cautious. Wireshark is sending out a message that when you are trying to connect to an unknown host, it will give you a warning. In this capture you are able to identify the ip address as well as its own fingerprint, in the real world you will be comparing the fingerprints with the fingerprints you were given in order to determine whether you are connecting to the correct host instead of an imposter.

Interestingly, Wireshark is also able to capture how many conversations were taken in place in the current entry.

Ethernet	IPv4	IPv6	TCP	UDP					
Address A 0e:3d:50:a2:1a:61	Address B 0xd3:9e:2f:df:eb	Packets 287	Bytes 30 k	Bytes A → B 139	Bytes B → A 10 k	Duration 148 ms	Rel Start 20 k	Bits/s A → B 0.000000	Bits/s B → A 251.2902

When going into statistics - Conversations, you are able to find multiple tabs with options Thernet, IPv4, IPv6, TCP and UDP

Since both SSH and Telnet uses TCP we can click on TCP

Wireshark - Conversations · Ethernet 3 (host tty.sdf.org)

Ethernet · 1	IPv4 · 1	IPv6	TCP · 2	UDP									
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
172.31.216.101	49769	205.166.94.8	23	119	10 k	67	3748	52	6391	0.000000	49.6825	603	1029
172.31.216.101	49770	205.166.94.8	22	168	20 k	72	6333	96	13 k 191.204636	60.0856	843	1820	

We can note that we are able to see 2 different entries, We can also note that although the IP address is the same they are using two different ports, 22 and 23

If we click on the first entry with the port number of 23 and hit follow stream

Follow Stream...

Wireshark · Follow TCP Stream (tcp.stream eq 0) · Ethernet 3 (host tty.sdf.org)

```
...%...%.....%.....&.... .#..'$..&..... .#..'$.....'.....x.
1.....'.....ANSI.....".....!.....".....!
sdf.lonestar.org (pts/0)
if new, login 'new' ...

.....login: [REDACTED]

Password for menna@otaku[REDACTED]

Last login: Sun Sep  6 11:24:46 2020 from ec2-54-242-174-76.compute-1.amazonaws.com on pts/0
Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005,
  2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017,
  2018, 2019 The NetBSD Foundation, Inc. All rights reserved.
Copyright (c) 1982, 1986, 1989, 1991, 1993
  The Regents of the University of California. All rights reserved.

[ 'menna' will expire in 628 days - Please 'validate' your account soon ]

Please press your BACKSPACE key: ..[H.[2]
=====
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          |
|---------|---------------|----------|-----------------------|
| mx      | up 3+19:05,   | 0 users, | load 0.23, 0.23, 0.31 |
| sdf     | up 3+19:04,   | 4 users, | load 0.25, 0.20, 0.12 |
| rie     | up 3+19:05,   | 2 users, | load 0.02, 0.02, 0.02 |
| vpn     | up 87+16:15,  | 0 users, | load 0.07, 0.08, 0.08 |
| vps3    | up 46+04:04,  | 0 users, | load 2.03, 2.01, 1.97 |
| miku    | up 3+19:05,   | 0 users, | load 0.15, 0.14, 0.15 |
| otaku   | up 3+19:04,   | 2 users, | load 0.14, 0.19, 0.24 |
| norge   | up 3+19:04,   | 0 users, | load 0.09, 0.03, 0.00 |
| faeroes | up 3+19:05,   | 3 users, | load 0.47, 0.26, 0.23 |
| iceland | up 3+19:05,   | 2 users, | load 0.39, 0.30, 0.25 |
| .       | .             | .        | .                     |


Packet 78. 39 client pkts, 31 server pkts, 44 turns. Click to select.
Entire conversation (3509 bytes) Show and save data as ASCII Stream 0
```

We can see that the whole conversation is visible, which is not secure. We can note that in the event of a Man in the Middle attack were to occur, they would be able to see everything that happened within this conversation.

However, lets view what happens when we enter into the entry with the port number of 22 instead.

Follow Stream...

Wireshark · Follow TCP Stream (tcp.stream eq 1) · Ethernet 3 (host tty.sdf.org)

```
SSH-2.0-OpenSSH_for_Windows_7.7
SSH-2.0-OpenSSH_8.0
...$      ...Pa.sh.....z...Y...0curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,ext-info-c..."ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,ssh-ed25519-cert-v01@openssh.com,ssh-rsa-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-ed25519,rsa-sha2-512,rsa-sha2-256,ssh-rsa...lchacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com...lchacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com....umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1....umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256- etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1- etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1....none....none.....].
...t..!e{.L.>....curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1...9ssh-ed25519,rsa-sha2-512,rsa-sha2-256,ssh-rsa,ssh-ed25519...lchacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com...lchacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com....umac-64-etm@openssh.com,umac-128- etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1- etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1....umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256- etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1- etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1....none,zlib@openssh.com....none,zlib@openssh.com.....X..
4.B..X..D..j..G..O....      ..@!.....3....ssh-ed25519....d..}...M.
3T....?.."...x.U^Y...{.... ...&....+I
....8tm.J.H.J..4(e..C...S....ssh-ed25519...@..x.u..w..h..
.K...Q..E.....\0Y.D.....eq0.....E,_..      L.
&%LS[.....F0..
```

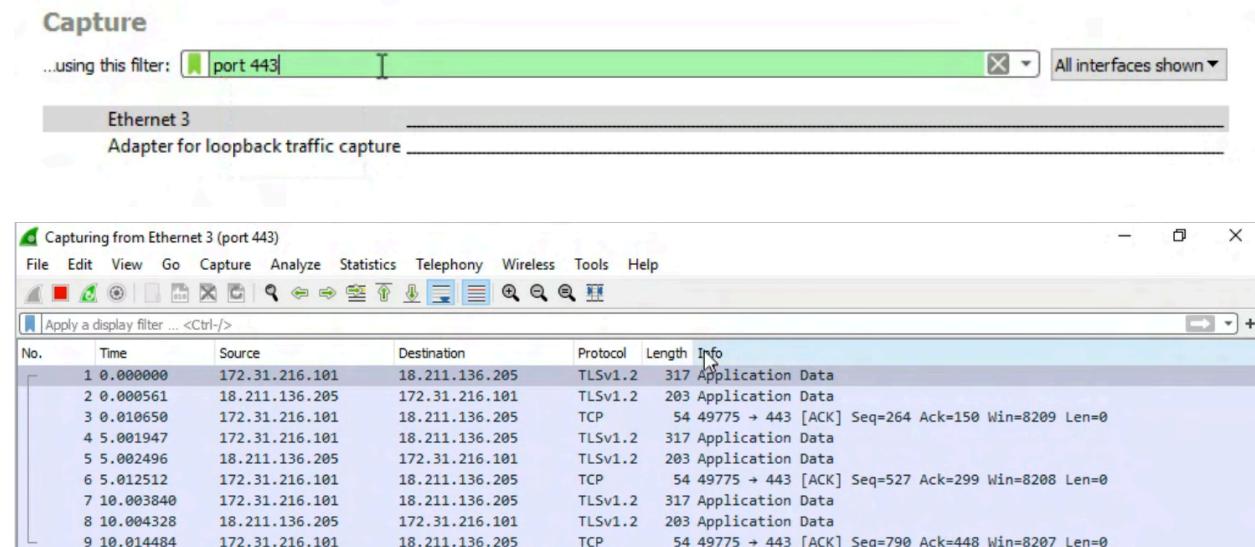
Notice how everything here is encrypted! This is the same entry as before... though this time a man in the middle attack will be quite difficult to decrypt this message.

### Task 7 - Generate, Capture, analyze ten decrypt HTTPS Traffic

In this task, we will be capturing packets from HTTPS Traffic. We can note that HTTPS is basically HTTP that operates with TLS.

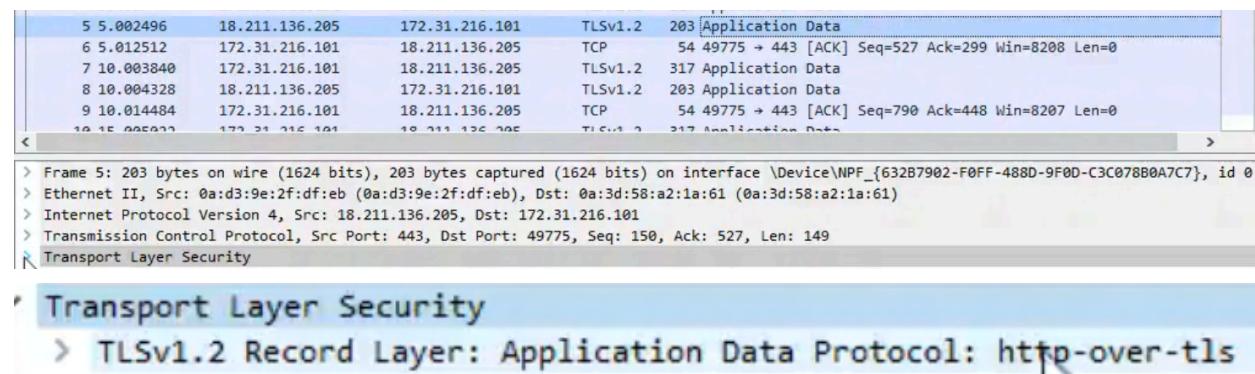
HTTP operates on port 80 while HTTPS operates on port 443.

We can begin Wireshark capture using capture filter of port 443



There is already traffic that is being captured using this filter. If there aren't any traffic being produced, which is odd, you can open google chrome and browse on the web for anything. There should be network traffic after following that step.

On entry 5, we can click on this application data and expand the transport layer security tab on the bottom



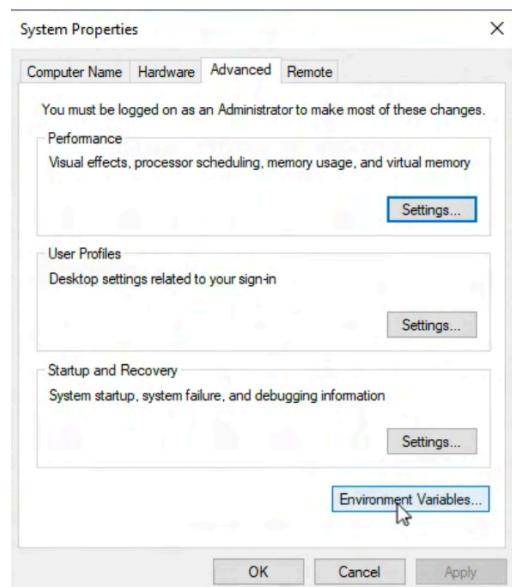
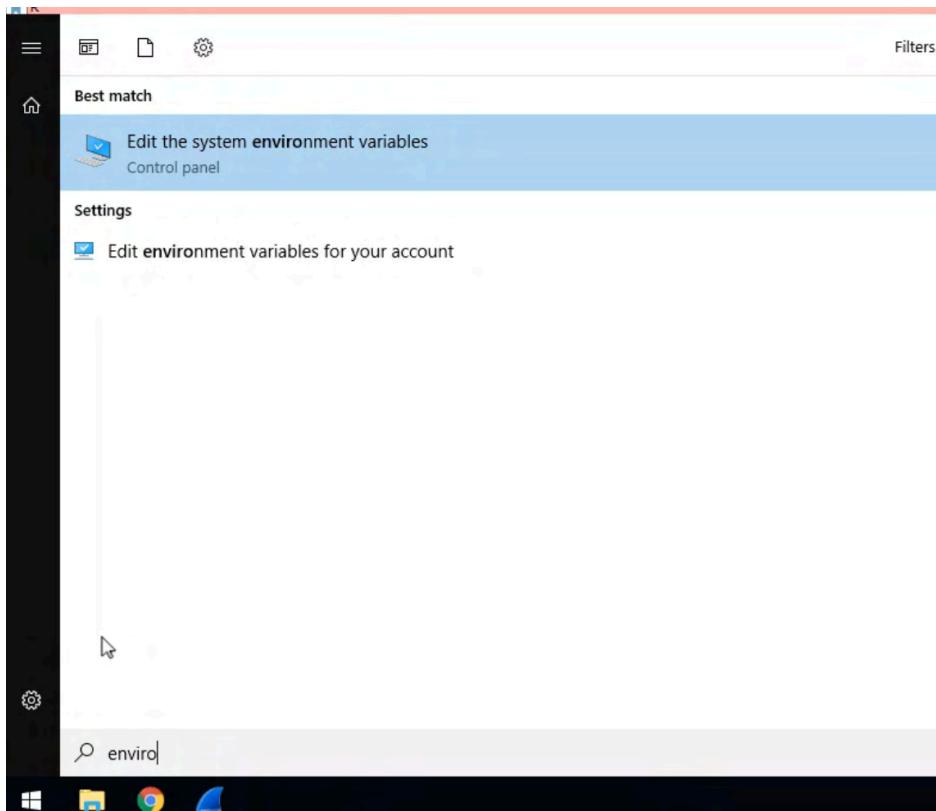
We can see that Wireshark is informing us that HTTPS is HTTP over TLS.

▼ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls  
Content Type: Application Data (23)  
Version: TLS 1.2 (0x0303)  
Length: 144  
Encrypted Application Data: f4249214cffd519e91da37683f306281c6e6ee8999abebe5...

When expanding even further, we can notice that Wireshark is informing us that this particular entry is being encrypted. We won't be able to decrypt this data due the fact that we do not possess the private key used for encryption.

However, say that you need to decrypt this data, perhaps for network administration or testing something in your home lab. We can decrypt SSL traffic using a pre-master secret key.

To set up this, we must first go in to the windows button and search for environmental variables



In here you will be able to click on the button Environmental Variables.  
Once in the environmental variables tab we should be able to add a new variable

User variables for Administrator	
Variable	Value
EXPORTER_MONITOR	us-west.monitor.rhyme.com
GOPATH	C:\Users\Administrator\go
Path	C:\Users\Administrator\AppData\Local\Microsoft\WindowsApps;C:...
TEMP	C:\Users\Administrator\AppData\Local\Temp
TMP	C:\Users\Administrator\AppData\Local\Temp

New...	Edit...	Delete
--------	---------	--------

System variables	
Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	2
OS	Windows_NT
Path	C:\Ruby26-x64\bin;C:\Program Files (x86)\Common Files\Oracle\Ja...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.RB;.RBW
PROCESSOR_ARCHITECTURE	AMD64

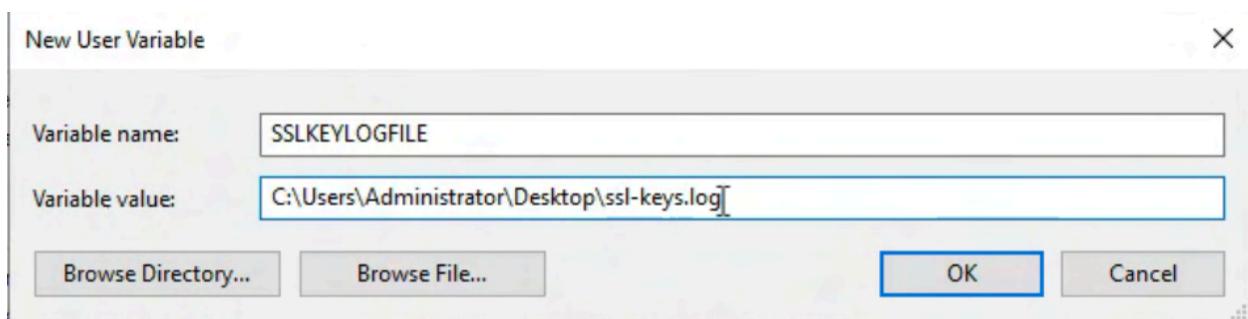
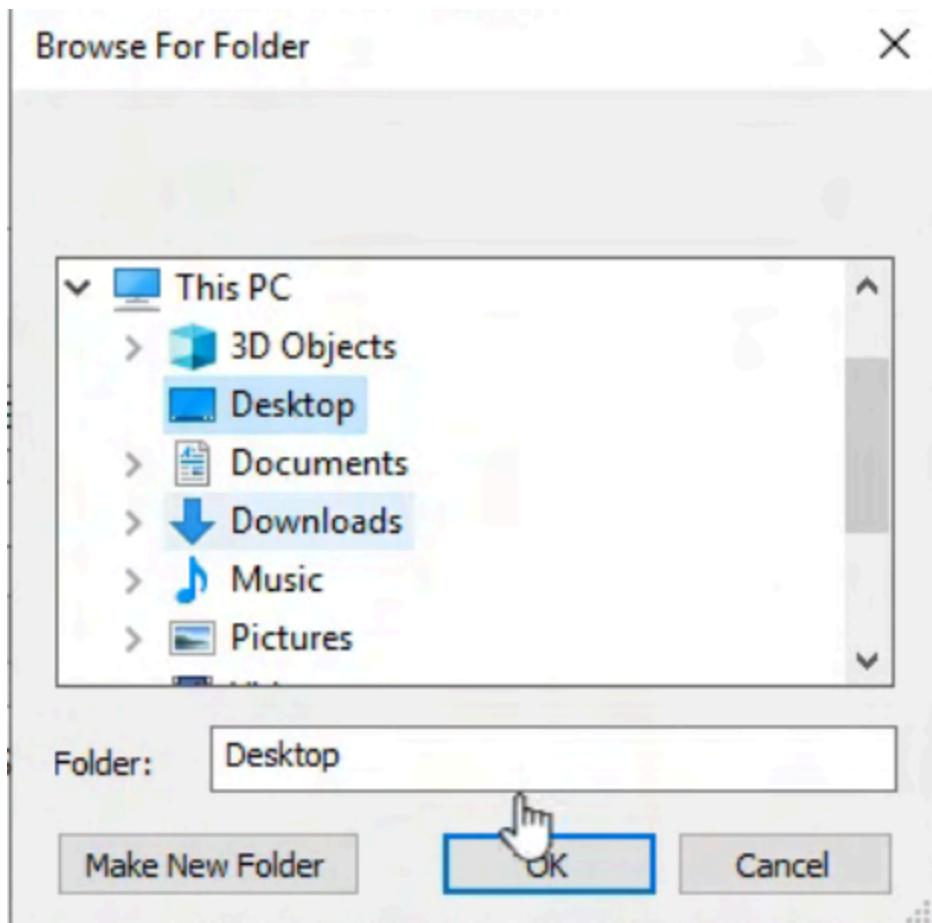
  

New...	Edit...	Delete
--------	---------	--------

OK	Cancel
----	--------

New User Variable			
Variable name:	SSLKEYLOGFILE		
Variable value:	[ ]		
<input type="button" value="Browse Directory..."/>	<input type="button" value="Browse File..."/>	<input type="button" value="OK"/>	<input type="button" value="Cancel"/>



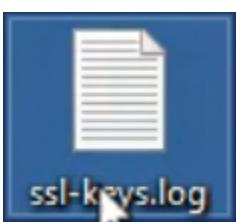
In the images above we are able to see that we creating a new variable named  
SSLKEYLOGFILE

This will be routed through the value C:\USers\Administrator\Desktop

We will be creating a new file using the back-slash key “\”

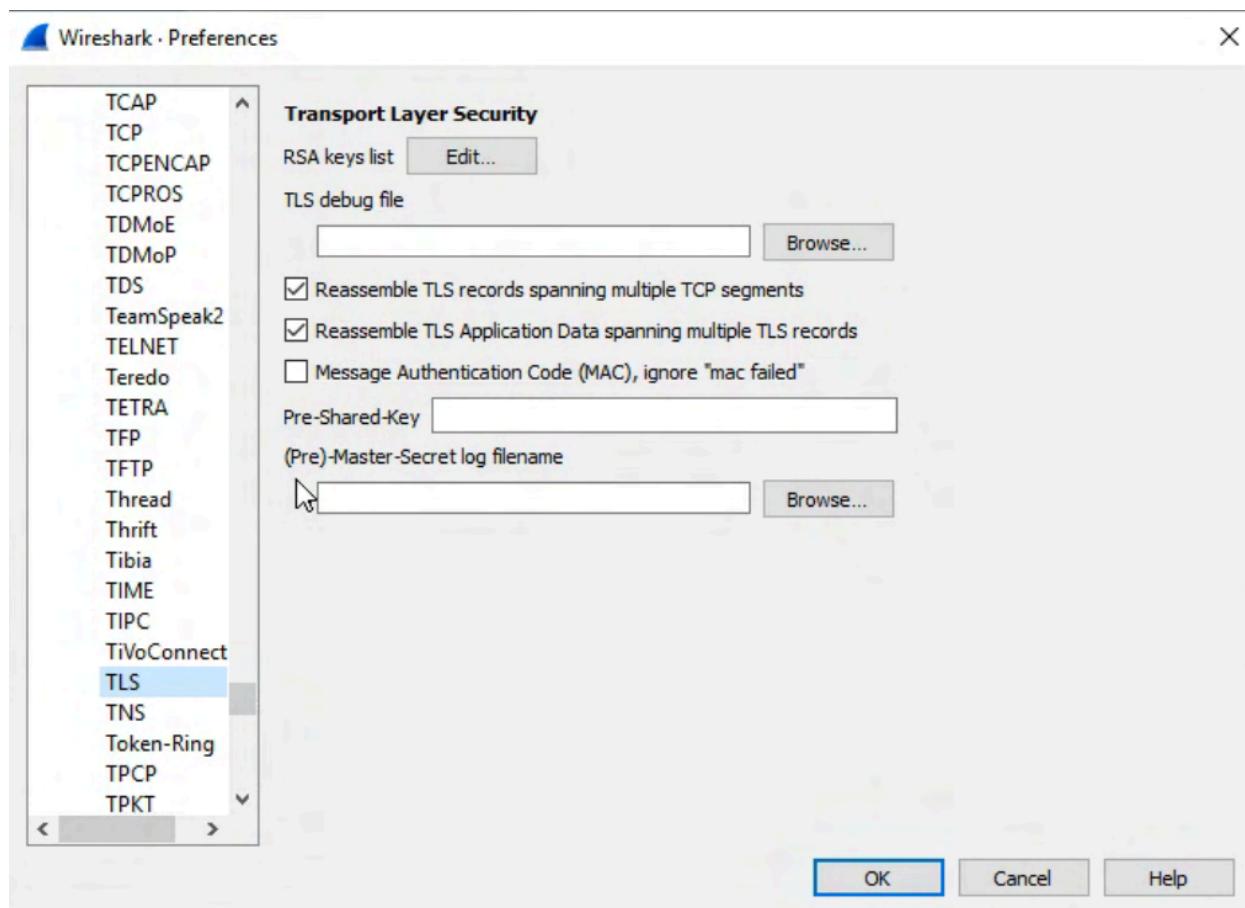
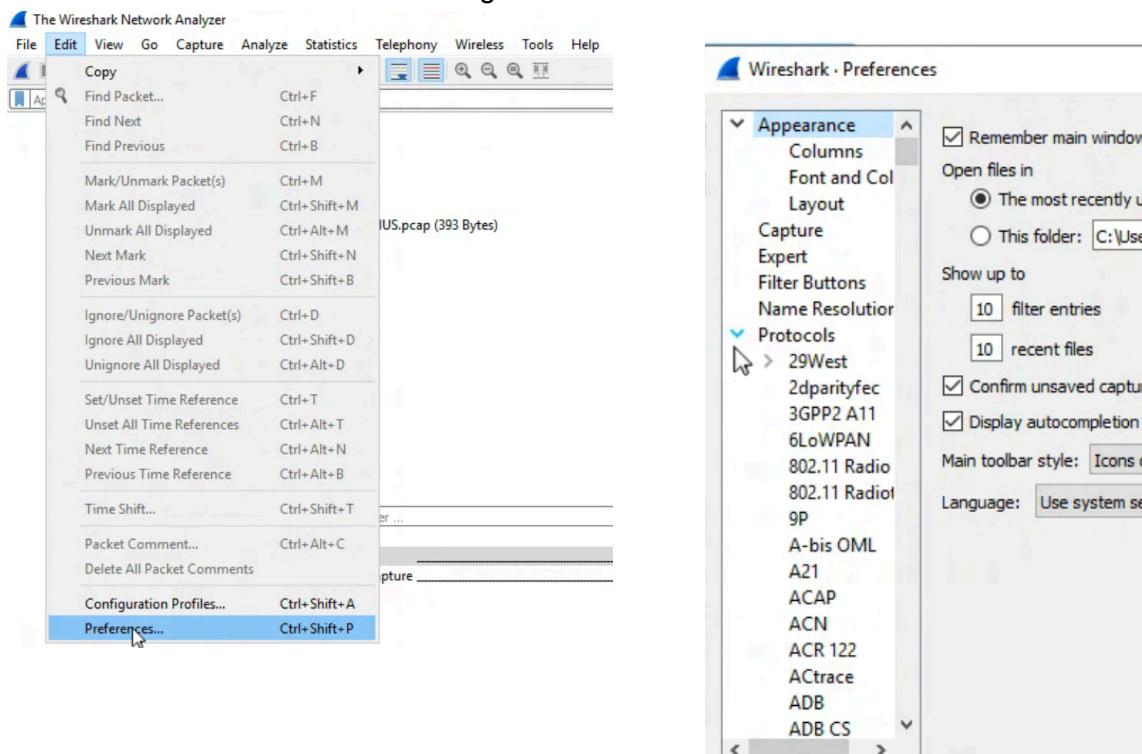
We are going to be creating a file ssl-keys.log

When finishing this, we should be able open google chrome and search for any random website such as amazon.com

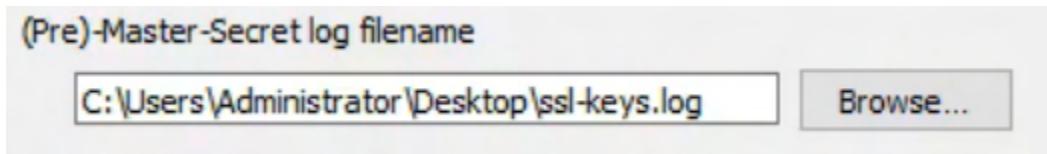


When completing these steps we will be able to see the new file created as a shortcut on our desktop home screen.

We will now return to Wireshark and go into Edit -> Preferences -> Protocols -> TLS



When completing the steps shown in the above image, the (Pre\_-Master-Secret log filename can be found. Here we will browse the desktop and find the file name we just made. It will look something like this



When completing these steps we should be able to capture network packets and view SSL in a decrypted format

**Capture**

...using this filter: port 443 All interfaces shown ▾

Ethernet 3 Adapter for loopback traffic capture

\*Ethernet 3 (port 443)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
140	5.184444	185.199.109.153	172.31.216.101	TLSv1.2	1514	[TLS segment of a reassembled PDU] [TCP segment of a reassembled PDU]
141	5.184533	172.31.216.101	185.199.109.153	TCP	54	49871 → 443 [ACK] Seq=1587 Ack=56143 Win=2102272 Len=0
142	5.184627	185.199.109.153	172.31.216.101	TLSv1.2	1514	[TLS segment of a reassembled PDU] [TCP segment of a reassembled PDU]
143	5.184627	185.199.109.153	172.31.216.101	TLSv1.2	1514	[TLS segment of a reassembled PDU] [TCP segment of a reassembled PDU]
144	5.184627	185.199.109.153	172.31.216.101	TLSv1.2	1514	[TLS segment of a reassembled PDU] [TCP segment of a reassembled PDU]
145	5.184627	185.199.109.153	172.31.216.101	TLSv1.2	1514	DATA[9][TLS segment of a reassembled PDU]
146	5.184627	185.199.109.153	172.31.216.101	HTTP2	618	DATA[9]
147	5.184660	172.31.216.101	185.199.109.153	TCP	54	49871 → 443 [ACK] Seq=1587 Ack=62547 Win=2102272 Len=0
148	6.740849	172.31.216.101	172.217.13.67	TCP	66	49874 → 443 [SYN, ECN, CWR] Seq=0 Win=64240 Len=0 MSS=1460 WScale=0 SackOK=1 Scale=0 Len=0 MSS=1460
149	6.741721	172.31.216.101	172.217.13.67	TCP	66	49874 → 443 [ECN ACK] Seq=0 Win=64240 Len=0 MSS=1460

> Frame 1: 317 bytes on wire (2536 bits), 317 bytes captured (2536 bits) on interface \Device\NPF\_{632B7902-F0FF-488D-9F0D-C3C078B0A7C7}, id 0  
> Ethernet II, Src: 0:a:3d:58:a2:1a:61 (0:a:3d:58:a2:1a:61), Dst: 0:a:d3:9e:2f:df:eb (0:a:d3:9e:2f:df:eb)  
> Internet Protocol Version 4, Src: 172.31.216.101, Dst: 18.211.136.205  
> Transmission Control Protocol, Src Port: 49866, Dst Port: 443, Seq: 1, Ack: 1, Len: 263  
▼ Transport Layer Security  
  ▼ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls  
    Content Type: Application Data (23)  
    Version: TLS 1.2 (0x0303)  
    Length: 258  
    Encrypted Application Data: 00000000000005c55ee75b4df9461cf9e6b64227634b017...

0030 20 13 21 47 00 00 00 17 03 03 01 02 00 00 00 00 00 00  
0040 00 00 5c 55 ee 75 b4 df 94 61 cf 9e 6b 64 22 76  
0050 34 b0 17 b6 2d b4 61 ce b6 33 9a 49 f6 b9 2d eb  
0060 1d 16 4e 60 03 8f b4 f9 00 e6 29 b7 64 2f 1d fe  
0070 88 c1 74 2d 7b bb 11 f3 a3 94 b3 d4 cb 03 37 1e  
0080 73 53 96 dc 02 70 58 78 fb 03 eb 88 79 58 76 28  
0090 28 45 fd 14 22 56 36 8d f6 bb ac 8d 4d 6e 57 d2  
00a0 be e4 07 9a 2a 7f 10 af 4f b4 87 81 9f 69 41 71

-!G-----  
..\\U-u... a-kd"v  
4.....a...-3...I...  
..N.....)d/..  
..t-{-.....7...  
s5...pXx...yXv.  
(E..."V6...MnL..  
....\*...0...jAn

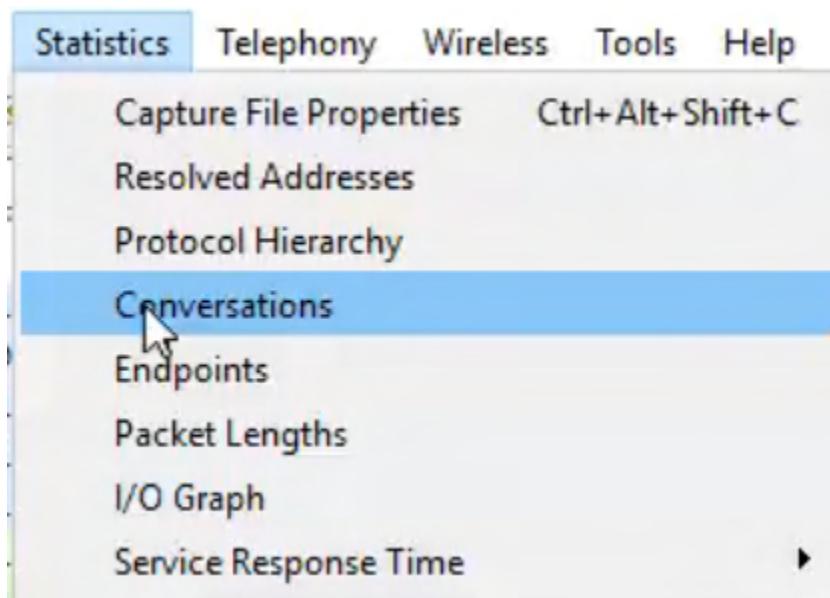
Notice how we have begun the network traffic using the capture filter port 443  
When entering these formats we can select an entry and inspect it further



We will be capturing network traffic from this website "Simple Site"

The screenshot shows a web browser window with the address bar containing 'kbroman.org/simple\_site/'. The page title is 'simple site'. Below the title, the text 'simple site Easy websites with GitHub Pages' is displayed. A note states: 'Github Pages provide a simple way to make a website using Markdown and git.' It lists the painful aspects of website creation: 'Working with html and css', 'Finding a hosting site', and 'Transferring stuff to the hosting site'. A note at the bottom says: 'With GitHub Pages, you just write things in Markdown, GitHub hosts the site for you, and you just push material'.

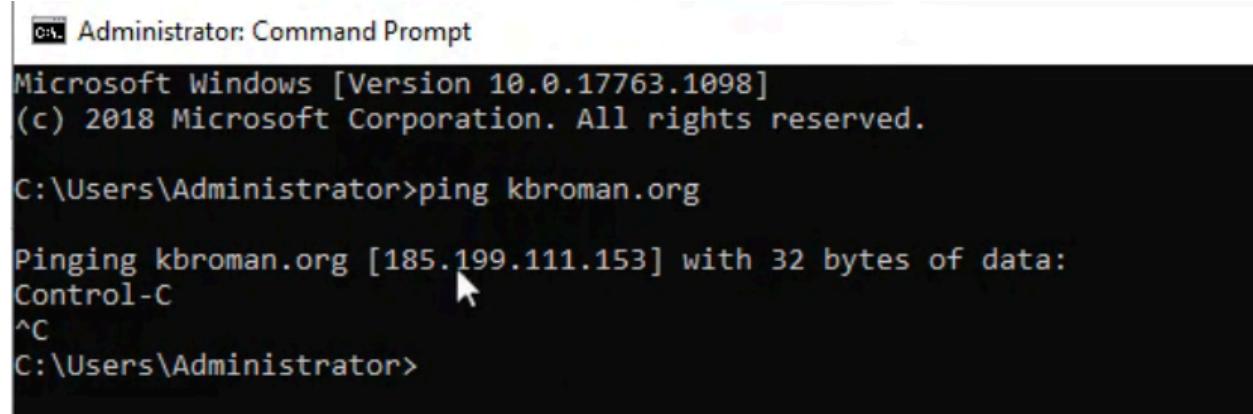
It should look something like this. With this we should be able to stop Wireshark capture. We will then head on over to statistics tab and click on conversations



The screenshot shows the 'Conversations' tab in Wireshark. The table displays network traffic statistics for Ethernet port 1. The columns include: Address A, Port A, Address B, Port B, Packets, Bytes, Packets A → B, Bytes A → B, Packets B → A, Bytes B → A, Rel Start, Duration, Bits/s A → B, and Bits/s B → A. The table shows several conversations between various IP addresses, with the last row being 172.31.216.101:49875 ↔ 172.217.2.99:443.

Ethernet · 1	IPv4 · 7	IPv6	TCP · 7	UDP · 2	Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
172.31.216.101	49866	18.211.136.205	443	12	2299		8	1487	4	812	0.000000	15.0155	792			432		
172.31.216.101	49870	172.217.164.141	443	11	4447		6	917	5	3530	4.769736	0.0246	297 k			1146 k		
172.31.216.101	49871	185.199.109.153	443	74	68 k		20	2768	54	65 k	4.794880	0.3898	56 k			1344 k		
172.31.216.101	49872	172.67.34.140	443	22	5316		10	1567	12	3749	4.932127	0.0236	530 k			1269 k		
172.31.216.101	49873	104.26.4.214	443	21	3311		9	1511	12	4800	4.966872	0.0185	651 k			2070 k		
172.31.216.101	49874	172.217.13.67	443	23	6238		11	1655	12	4583	6.740849	0.0256	516 k			1430 k		
172.31.216.101	49875	172.217.2.99	443	12	4758		6	917	6	3841	15.681095	0.0185	396 k			1661 k		

When on this page it will be a little difficult to find the correct entry to inspect further... We can mitigate this by pinging the server in cmd.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

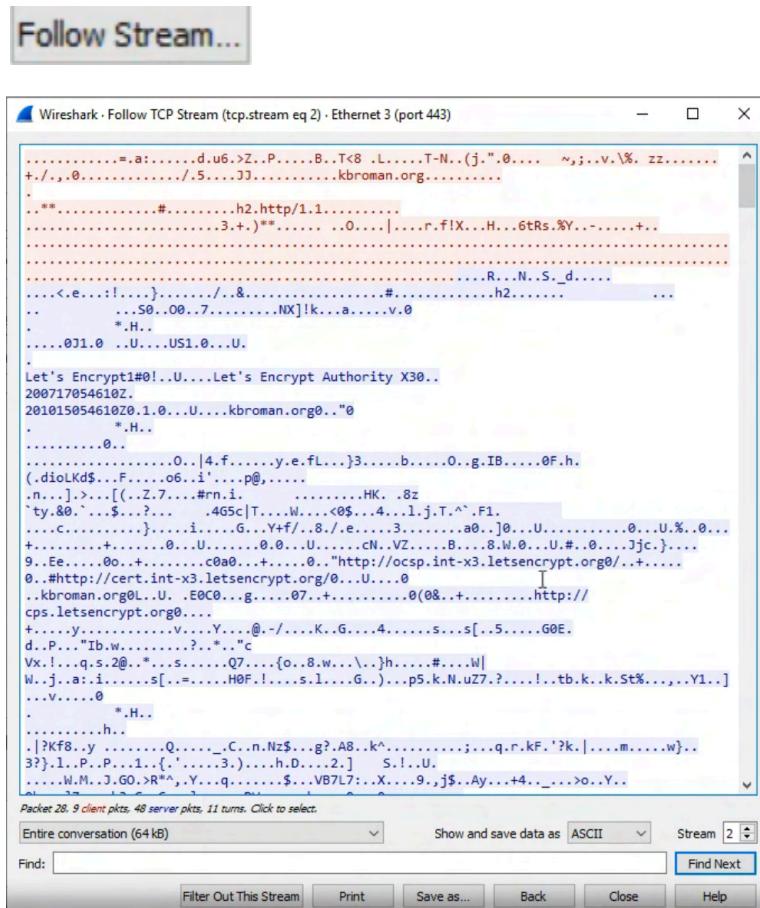
C:\Users\Administrator>ping kbroman.org

Pinging kbroman.org [185.199.111.153] with 32 bytes of data:
Control-C
^C
C:\Users\Administrator>
```

It should look something like this. We can note that the IP address is shown so we can use this information to narrow down the search in our entry list on Wireshark.

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
172.31.216.101	49866	18.211.136.205	443	12	2299	8	1487	4	812	0.000000	15.0155	792	432
172.31.216.101	49870	172.217.164.141	443	11	4447	6	917	5	3530	4.769736	0.0246	297 k	1146 k
172.31.216.101	49871	185.199.109.153	443	74	68 k	20	2768	54	65 k	4.794880	0.3898	56 k	1344 k

We can see the IP address is listed here. We can click on it to have it highlighted and head on over to the follow stream button



We can see that on the image to our left, we have the encrypted... however before we were only able to see application data. Now we are able to see everything else like we normally would on HTTP traffic. We are even able to see the source code for this website.

## **Lessons Learned**

Using Wireshark and most of its basic functionalities: capturing some network traffic that is flowing through your machine now and analyzing already captured network traffic by opening a stored capture file.

Generating and capturing RADIUS Traffic, analyzing it and viewing it in Wireshark. Knowing what the RADIUS Architecture consists of and decrypting the encrypted password with the shared secret using Wireshark.

Knowing the basics about HTTP, and knowing the difference between Wireshark's Capture and Display Filters. Connecting to an HTTP Server and initiating Basic HTTP Authentication and capturing its Traffic on Wireshark, analyzing the captured packets and seeing the username and password being sent.

Initiating an HTTP Form-based authentication, capturing it in Wireshark and analyzing it so you can see the username and password clearly. Also Capturing DNS Traffic.

Knowing how Telnet works, starting a Telnet Session with a remote Device using Powershell, capturing its traffic in Wireshark and analyzing it from the Security perspective

Opening a SSH Session with the same Device as in Task 5, capturing the traffic and comparing it with the Telnet Packet Capture. Capturing traffic based on the host involved and how to see all the host, packet and protocol statistics as well as conversations that have happened in a certain Capture.