Course: Cloud and Network Security
Name: Neville Ngothe Iregi
Student No.: CS-CNS10-25054
Date: Tuesday, 23 September 2025

Week 2 Assignment 2 HTB Academy: Introduction to Network Traffic Analysis

Introduction

Network Traffic Analysis (NTA) is a high-value and highly-transferable skill in cybersecurity that is used in monitoring network activity and looking for threats that could capitalize on security and operational issues. It is described as the act of examining network traffic to characterize common ports and protocols, establish a baseline for our environment, monitor and respond to threats, and ensure the greatest possible insight into our organization's network.

Offensive security practitioners use network traffic analysis to search for sensitive data such as credentials, hidden applications, reachable network segments, or other potentially sensitive information. For defenders, they use network traffic analysis to collect and analyze real-time and historical data of what is happening on the network. Network traffic analysis can also be used by both sides to search for vulnerable protocols and ciphers in use.

In this assignment, I explored the principles of network traffic analysis and practically demonstrated the use of traffic analysis tools such as **Wireshark** and **tcpdump** in Hack The Box. Wireshark is GUI-based while tcpdump is command line-based for Linux/UNIX. Tcpdump can be particularly useful if you have to analyze a non-GUI based system or a remote system where a GUI would be slow, inefficient, and not very stealthy.

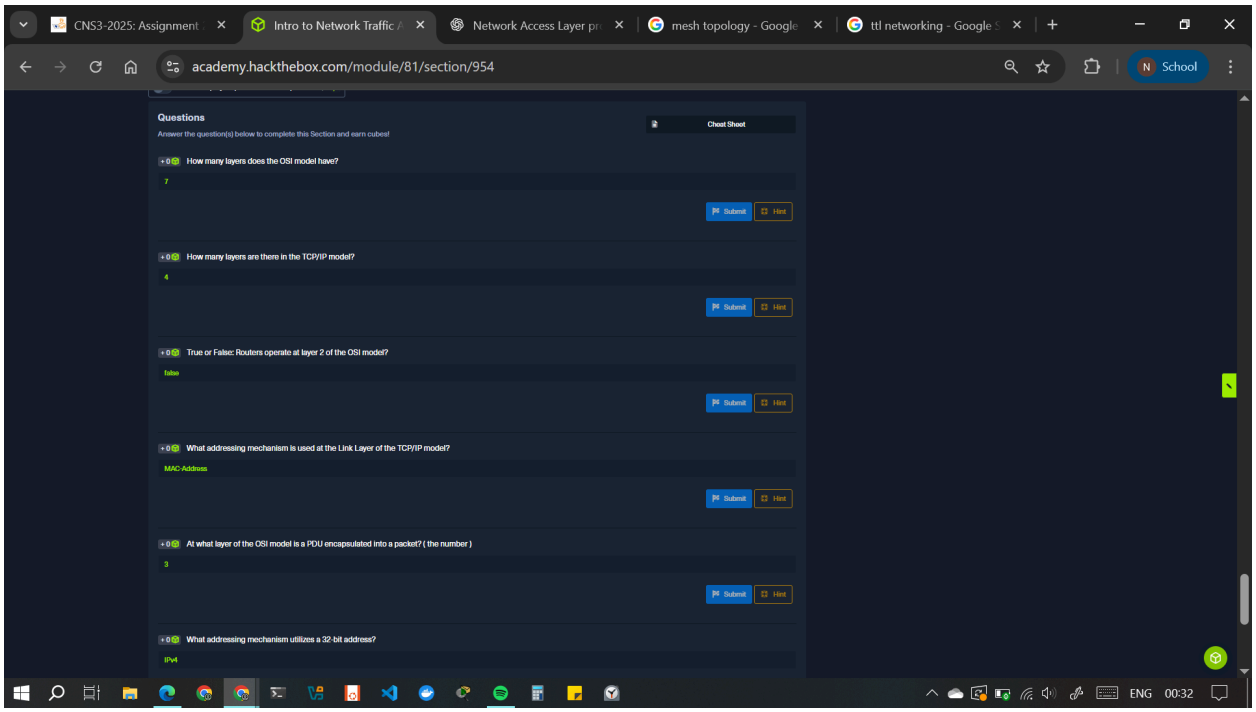
Some of the prerequisites for this assignment are:

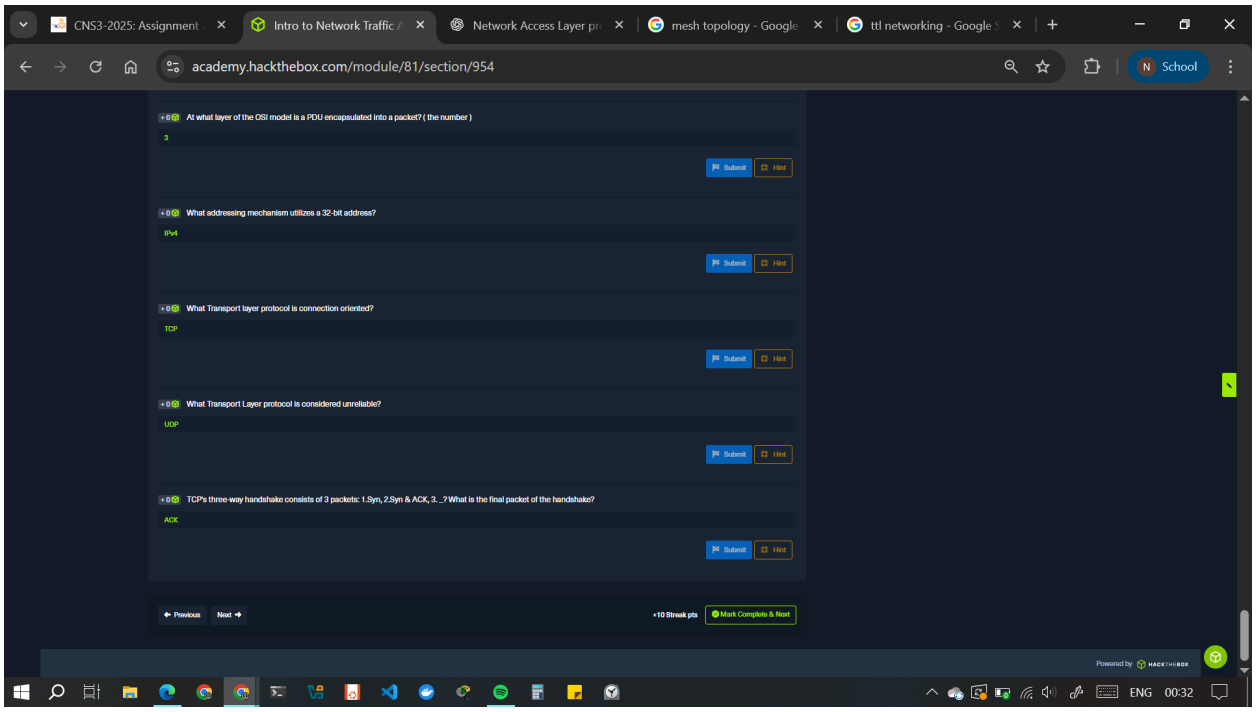
1. Linux fundamentals
2. Introduction to Networking
3. Web Requests

Use Cases of NTA include:

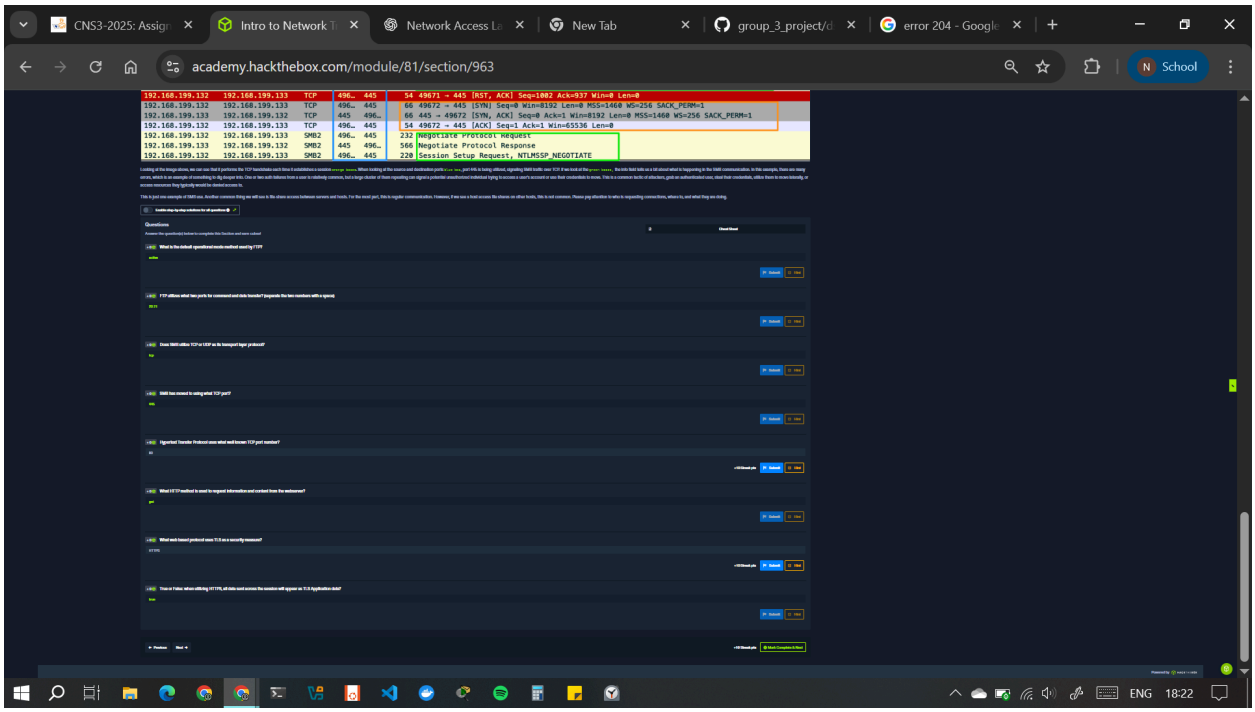
1. Collecting real-time traffic within the network to analyze upcoming threats.
2. Setting a baseline for day-to-day network communications.
3. Identifying and analyzing traffic from non-standard ports, suspicious hosts, and issues with networking protocols such as HTTP errors, problems with TCP, or other networking misconfigurations.
4. Detecting malware on the wire, such as ransomware, exploits, and non-standard interactions.
5. Useful when investigating past incidents and during threat hunting.

Networking Primer - Layers 1-4





Networking Primer - Layers 5-7



Tcpdump Fundamentals

It does not require a GUI and can be used through any terminal or remote connection, such as SSH.

To capture network traffic from "off the wire," it uses the libraries **pcap** and **libpcap**, paired with an interface in promiscuous mode to listen for data. This allows the program to see and capture packets sourcing from or destined for any device in the local area network, not just the packets destined for us. Due to the direct access to the hardware, we need the root or the administrator's privileges to run this tool, hence the need to utilize **sudo** to execute TCPdump.

Traffic Captures with Tcpdump

Switches modify how tcpdump runs or how it displays data.

Some Basic Capture Options(switches) for Tcpdump include:

	Switch Command	Result
1.	**D**	Will display any interfaces available to capture from.
2.	**i**	Selects an interface to capture from. ex. -i eth0
3.	**n**	Do not convert addresses (i.e., host addresses, port numbers, etc.) to names.
4.	**e**	Will grab the ethernet header along with upper-layer data.
5.	**X**	Show Contents of packets in hex and ASCII.

6. **XX** Same as X, but will also specify ethernet headers. (like using Xe)
7. **v, vv, vvv** Increase the verbosity of output shown and saved.
8. **c** Grab a specific number of packets, then quit the program.
9. **s** Defines how much of a packet to grab.
10. **S** change relative sequence numbers in the capture display to absolute sequence numbers. (13248765839 instead of 101)
11. **q** Print less protocol information.
12. **r file.pcap** Read from a file.
13. **w file.pcap** Write into a file
14. **l** Make stdout line buffered. Useful if you want to see the data while capturing it.e.g allowing us to pipe the contents of a pcap file out to another function such as 'grep'
15. **A** Shows only the ASCII text after the packet line, instead of both ASCII and hex.

Nb:

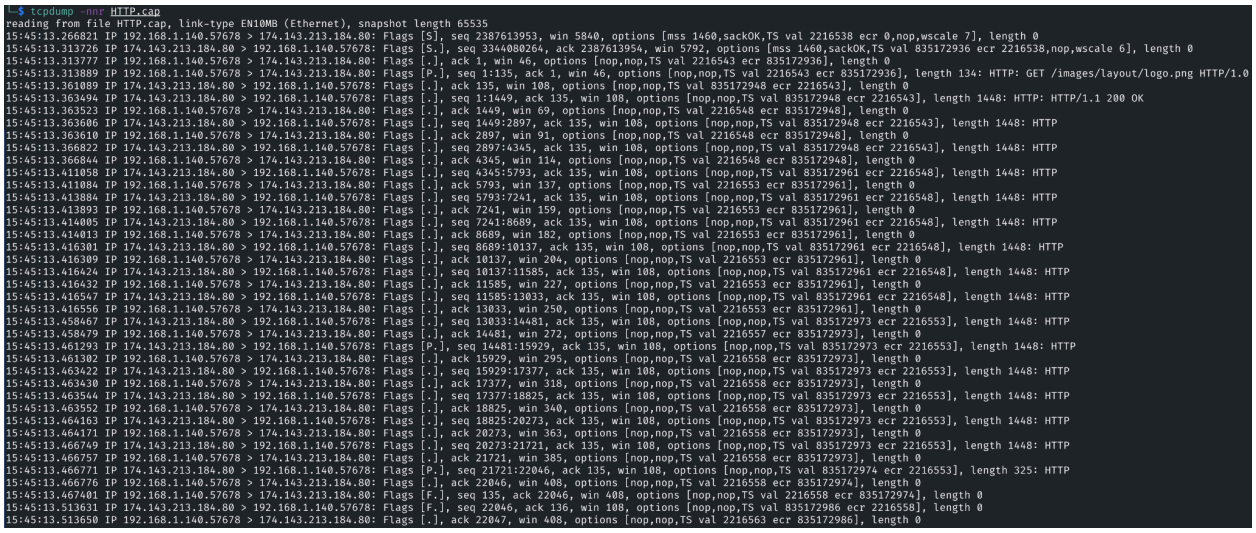
1. Optional best practice used with the switches is **Don't resolve hostnames** → -nn (it is faster, avoids delays in DNS querying).
2. Using modifiers and redirecting output can be a quick way to scrape websites for email addresses, naming standards, and much more. E.g `sudo tcpdump -Ar http.cap -l | grep 'mailto:*'`

More info on the complete list of switches is found in the man pages i.e **man tcpdump**

There is a great advantage in knowing how a network functions and how to use the filters that TCPDump provides. With them, we can view the network traffic, parse it for any issues, and identify suspicious network interactions quickly. Theoretically, we can use tcpdump to create an IDS/IPS system by having a Bash script analyze the intercepted packets according to a specific pattern. We can then set conditions to, for example, ban a particular IP address that has sent too many ICMP echo requests for a certain period.

Questions

1. Utilizing the output shown in question-1.png, who is the server in this communication? (IP Address) **Answer: 174.143.213.184**



```
└─$ tcpdump -nnr HTTP.cap
reading from file HTTP.cap, link-type EN10MB (Ethernet), snapshot length 65535
15:45:13.266821 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [S], seq 2387613953, win 5840, options [mss 1460,sackOK,TS val 2216538 ecr 0,nop,wscale 7], length 0
15:45:13.313726 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [S.], seq 3344098264, ack 2387613954, win 5792, options [mss 1460,sackOK,TS val 835172936 ecr 2216538,nop,wscale 6], length 0
15:45:13.313777 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 1, win 46, options [nop,nop,TS val 2216543 ecr 835172936], length 0
15:45:13.313889 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [P.], seq 1:135, ack 1, win 46, options [nop,nop,TS val 2216543 ecr 835172936], length 134: HTTP: GET /images/layout/logo.png HTTP/1.0
15:45:13.361089 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], ack 135, win 108, options [nop,nop,TS val 835172948 ecr 2216543], length 0
15:45:13.363494 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 1:1449, ack 135, win 108, options [nop,nop,TS val 835172948 ecr 2216543], length 1448: HTTP: HTTP/1.1 200 OK
15:45:13.363523 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 1449, win 69, options [nop,nop,TS val 2216548 ecr 835172948], length 0
15:45:13.363606 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 1449:2897, ack 135, win 108, options [nop,nop,TS val 835172948 ecr 2216543], length 1448: HTTP
15:45:13.363610 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 2897, win 91, options [nop,nop,TS val 2216548 ecr 835172948], length 0
15:45:13.366822 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 2897:4345, ack 135, win 108, options [nop,nop,TS val 835172948 ecr 2216543], length 1448: HTTP
15:45:13.366844 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 4345, win 114, options [nop,nop,TS val 2216548 ecr 835172948], length 0
15:45:13.411058 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 4345:5793, ack 135, win 108, options [nop,nop,TS val 835172961 ecr 2216548], length 1448: HTTP
15:45:13.411084 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 5793, win 137, options [nop,nop,TS val 2216553 ecr 835172961], length 0
15:45:13.413884 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 5793:7241, ack 135, win 108, options [nop,nop,TS val 835172961 ecr 2216548], length 1448: HTTP
15:45:13.413893 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 7241, win 159, options [nop,nop,TS val 2216553 ecr 835172961], length 0
15:45:13.414005 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 7241:8689, ack 135, win 108, options [nop,nop,TS val 835172961 ecr 2216548], length 1448: HTTP
15:45:13.414013 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 8689, win 182, options [nop,nop,TS val 2216553 ecr 835172961], length 0
15:45:13.416301 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 8689:10137, ack 135, win 108, options [nop,nop,TS val 835172961 ecr 2216548], length 1448: HTTP
15:45:13.416309 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 10137, win 204, options [nop,nop,TS val 2216553 ecr 835172961], length 0
15:45:13.416424 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 10137:11585, ack 135, win 108, options [nop,nop,TS val 835172961 ecr 2216548], length 1448: HTTP
15:45:13.416432 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 11585, win 227, options [nop,nop,TS val 2216553 ecr 835172961], length 0
15:45:13.416547 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 11585:13033, ack 135, win 108, options [nop,nop,TS val 835172961 ecr 2216548], length 1448: HTTP
15:45:13.416556 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 13033, win 250, options [nop,nop,TS val 2216553 ecr 835172961], length 0
15:45:13.458467 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 13033:14481, ack 135, win 108, options [nop,nop,TS val 835172973 ecr 2216553], length 1448: HTTP
15:45:13.458479 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 14481, win 272, options [nop,nop,TS val 2216557 ecr 835172973], length 0
15:45:13.461293 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [P.], seq 14481:15929, ack 135, win 108, options [nop,nop,TS val 835172973 ecr 2216553], length 1448: HTTP
15:45:13.461302 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 15929, win 295, options [nop,nop,TS val 2216558 ecr 835172973], length 0
15:45:13.463422 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 15929:17377, ack 135, win 108, options [nop,nop,TS val 835172973 ecr 2216553], length 1448: HTTP
15:45:13.463430 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 17377, win 318, options [nop,nop,TS val 2216558 ecr 835172973], length 0
15:45:13.463544 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 17377:18825, ack 135, win 108, options [nop,nop,TS val 835172973 ecr 2216553], length 1448: HTTP
15:45:13.463552 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 18825, win 340, options [nop,nop,TS val 2216558 ecr 835172973], length 0
15:45:13.464163 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 18825:20273, ack 135, win 108, options [nop,nop,TS val 835172973 ecr 2216553], length 1448: HTTP
15:45:13.464171 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 20273, win 363, options [nop,nop,TS val 2216558 ecr 835172973], length 0
15:45:13.466749 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [.], seq 20273:21721, ack 135, win 108, options [nop,nop,TS val 835172973 ecr 2216553], length 1448: HTTP
15:45:13.466757 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 21721, win 385, options [nop,nop,TS val 2216558 ecr 835172973], length 0
15:45:13.466771 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [P.], seq 21721:22046, ack 135, win 108, options [nop,nop,TS val 835172974 ecr 2216553], length 325: HTTP
15:45:13.466776 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 22046, win 408, options [nop,nop,TS val 2216558 ecr 835172974], length 0
15:45:13.467401 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [F.], seq 135, ack 22046, win 408, options [nop,nop,TS val 2216558 ecr 835172974], length 0
15:45:13.513631 IP 174.143.213.184.80 > 192.168.1.140.57678: Flags [F.], seq 22046, ack 136, win 108, options [nop,nop,TS val 835172986 ecr 2216558], length 0
15:45:13.513650 IP 192.168.1.140.57678 > 174.143.213.184.80: Flags [.], ack 22047, win 408, options [nop,nop,TS val 2216563 ecr 835172986], length 0
```

2. Were absolute or relative sequence numbers used during the capture? (see question-1.zip to answer) **Relative - These are a human-friendly view where the first sequence number observed in a flow is normalized to 0 (or 1). The first two packets have tcp headers with absolute sequence numbers.**

3. If I wish to start a capture without hostname resolution, verbose output, showing contents in ASCII and hex, and grab the first 100 packets; what are the switches used? please answer in the order the switches are asked for in the question.

   -nvXc 100

   Hostname resolution: n verbose output: v showing contents in ASCII and hex: X grab the first 100 packets: -c 100

4. Given the capture file at /tmp/capture.pcap, what tcpdump command will enable you to read from the capture and show the output contents in Hex and ASCII? (Please use best practices when using switches)

   sudo tcpdump -Xr /tmp/capture.pcap

   -X comes before r and sudo is used

5. What TCPDump switch will increase the verbosity of our output? (Include the - with the proper switch)

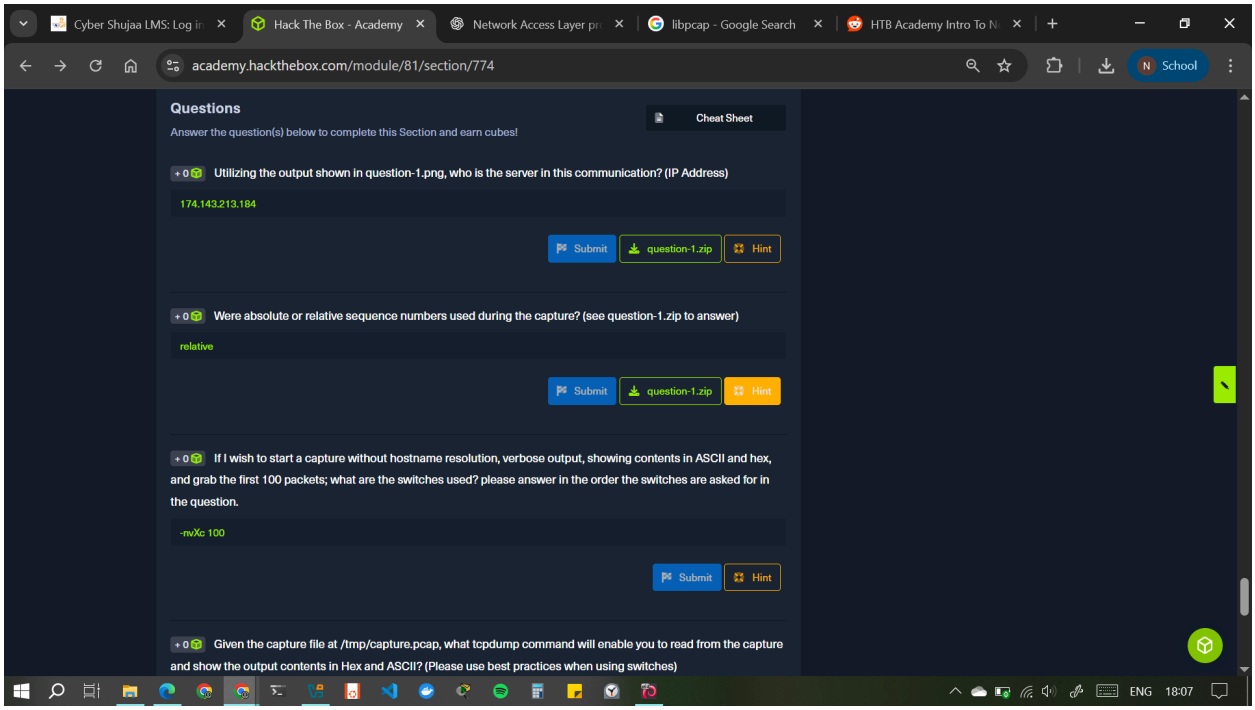
   -v

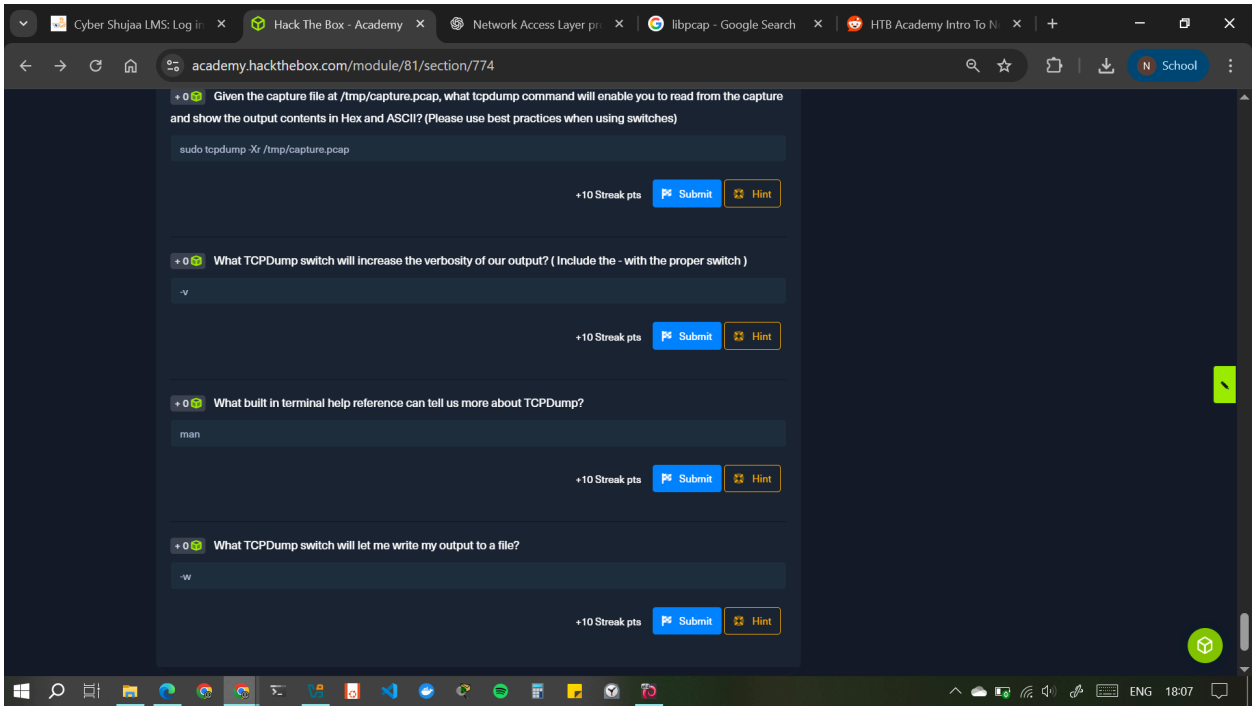
6. What built in terminal help reference can tell us more about TCPDump?

   man pages (man tcpdump)

7. What TCPDump switch will let me write my output to a file?

-w





Tcpdump: Fundamentals Lab

This lab exposed me to tcpdump through practice various tcpdump basics such as:

- reading from and writing to files
- utilizing basic switches
- locating files in the terminal

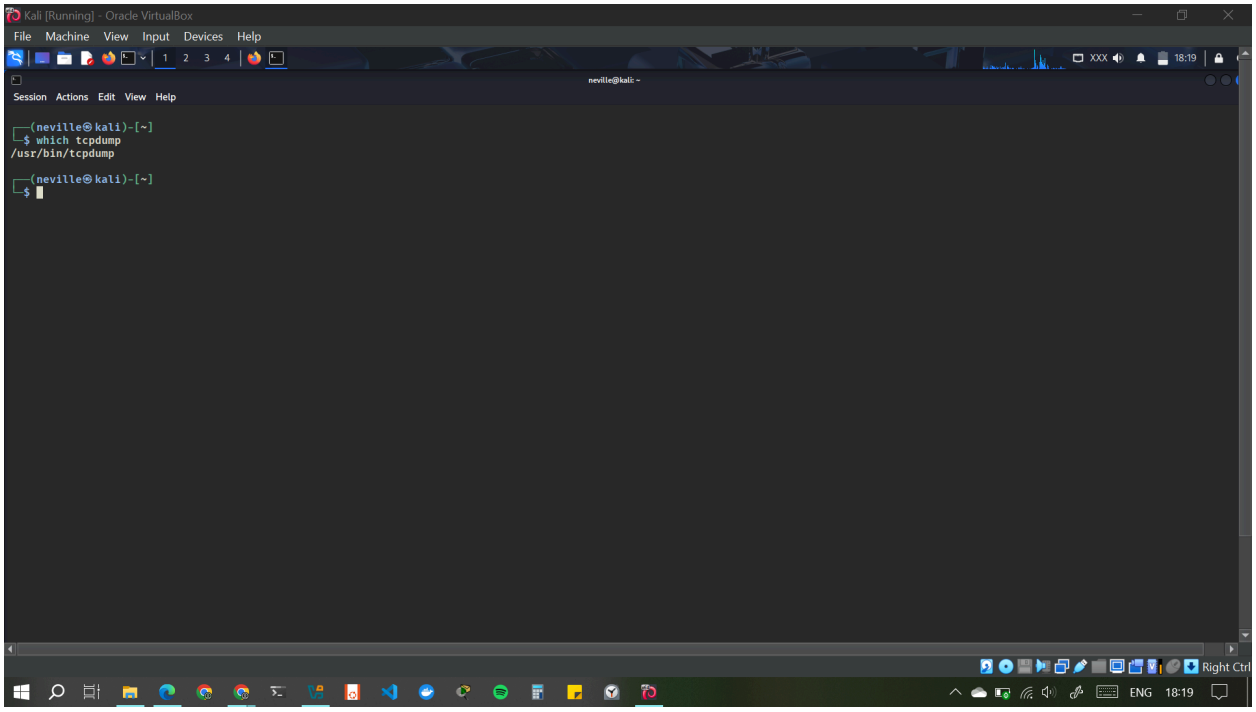
This type of work is often used to examine specific hosts and servers in more detail and find out who they all interact with. This procedure can also be used to identify so-called **backdoors** or other **potential breaches**. This could be used to monitor and log all communication from one server to analyze the packets sent and received.

Scenario:

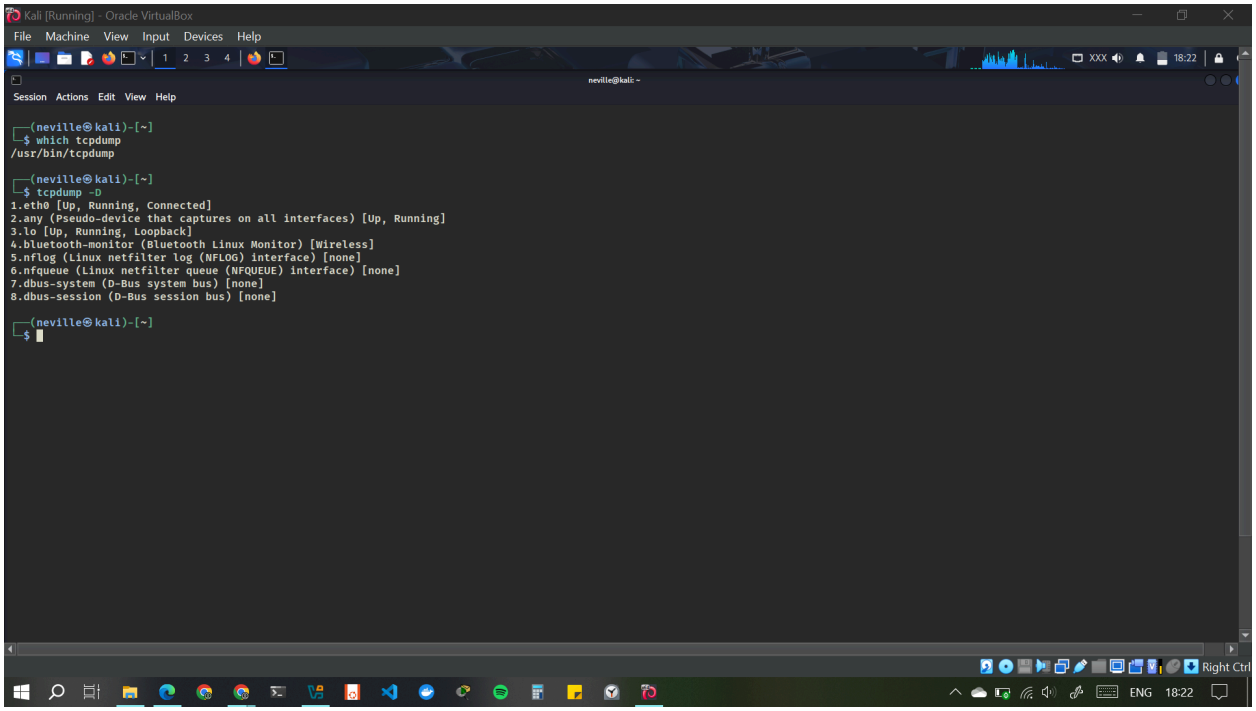
As the new network administrator for the Corporation, we have been tasked with capturing some network traffic to help baseline and validate the Corporation's network. As a test, we start utilizing tcpdump to get a small capture of our local broadcast domain traffic to ensure our capture device will work to accomplish this task. We need to ensure the tools and dependencies required are installed and test our ability to read traffic and capture it to a file.

Steps taken

1. Validate Tcpdump is installed on our machine using command: **which tcpdump**



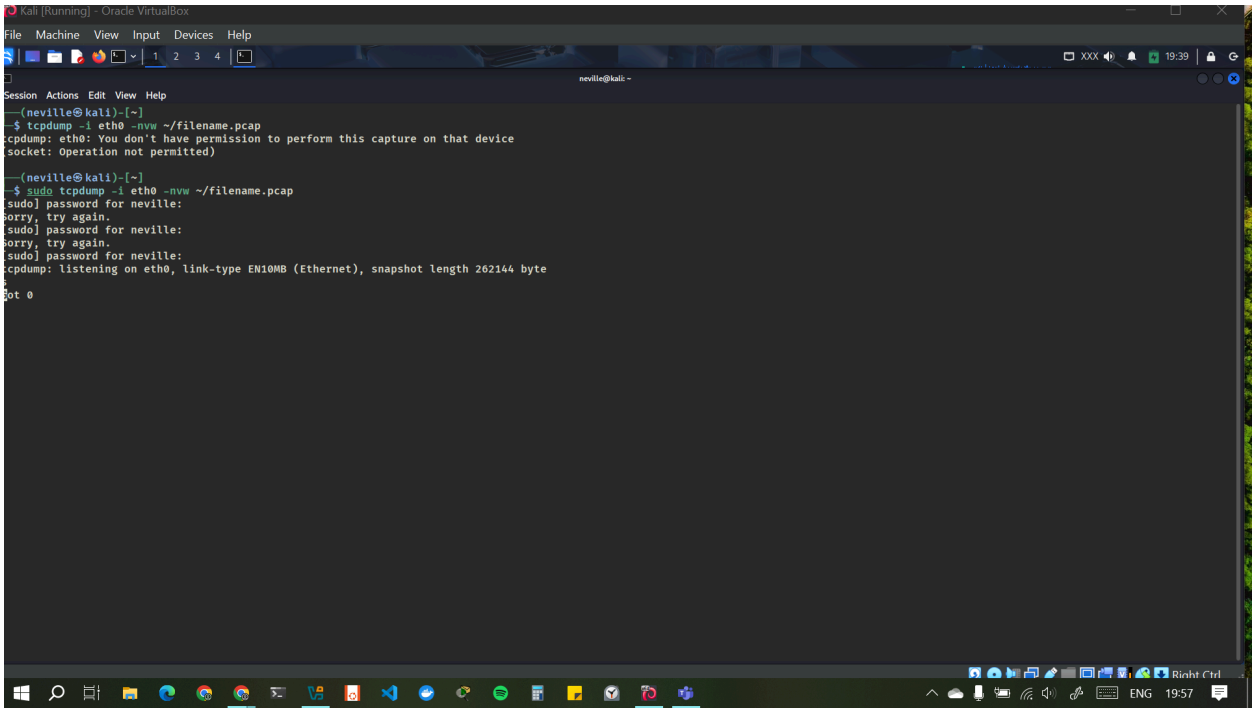
2. Start a capture by checking all possible interfaces we can listen to using switch **-D**



3. Utilize Basic Capture Filters such as adding verbosity to our output and displaying contents in ASCII and Hex using command: **tcpdump -i [interface name or #] -vX**
    - Disable name resolution and display relative sequence numbers for another challenge would use command: **tcpdump -i [interface name or #] -nn**

      (-S is not used since it is used to show absolute sequence numbers)

4. Save a Capture to a .PCAP file.
    - Capture filters modify what we get. We grab our first full capture from the wire and save it to a PCAP file. This becomes a sample to baseline the enterprise network.
    - Command used: **tcp -i [interface name or #] -nvw [/path/of/filename.pcap]**



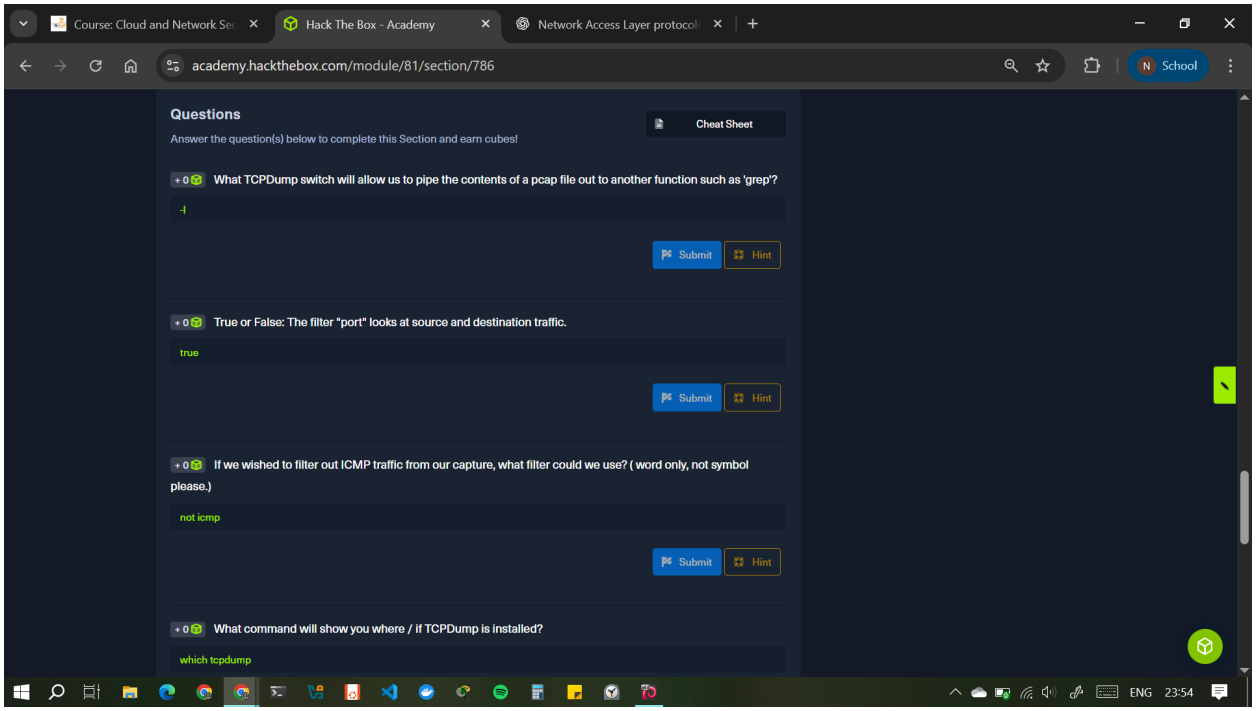
5. Read the Capture from a .PCAP file. - Our team members have given us a PCAP they captured while surveying another section of the enterprise, read the PCAP file into tcpdump, and modify our view of the PCAP to help us determine what is happening. We can disable hostname and port resolution for simplicity and ensure we see any TCP sequence and acknowledgment numbers in absolute values. For the sake of the lab, utilize the PCAP file we created in the previous step for this task.
    - Command used: **tcp -nnSXr [file/to/read.pcap]**
    - The switches used above will not resolve hostnames or port numbers, apply for absolute sequence numbers, and show contents in Hex and ASCII when reading from the PCAP file.

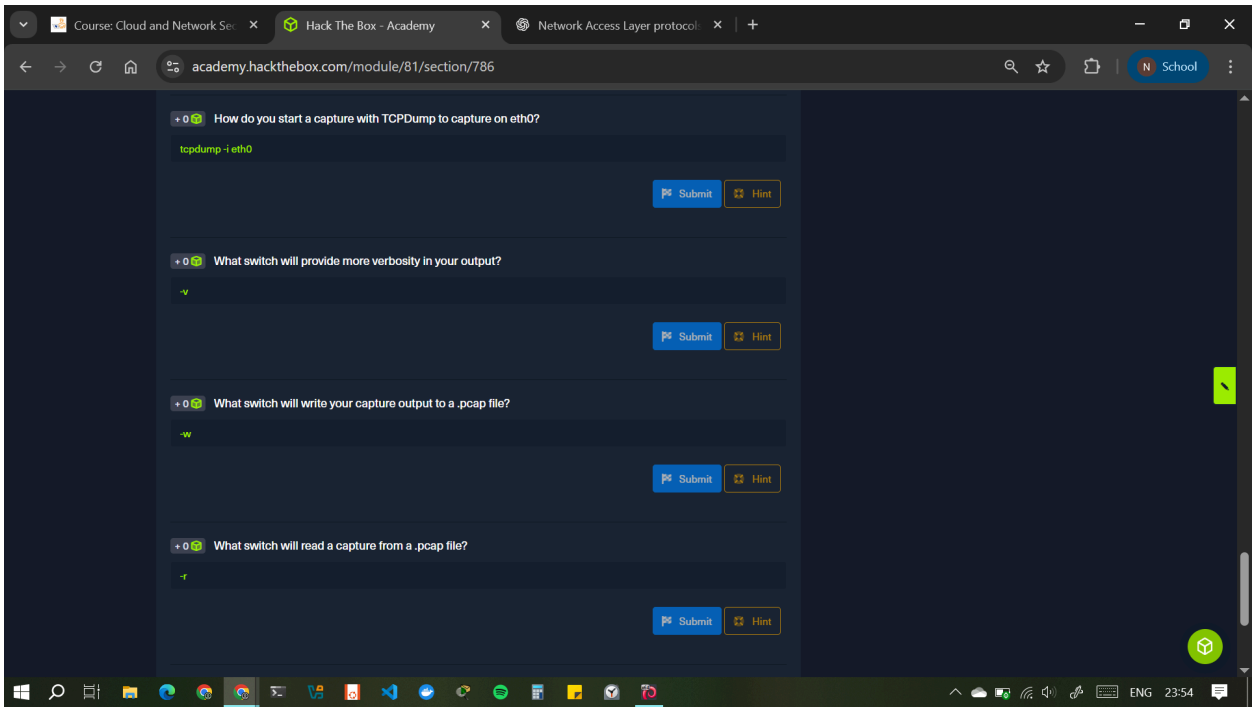
Questions

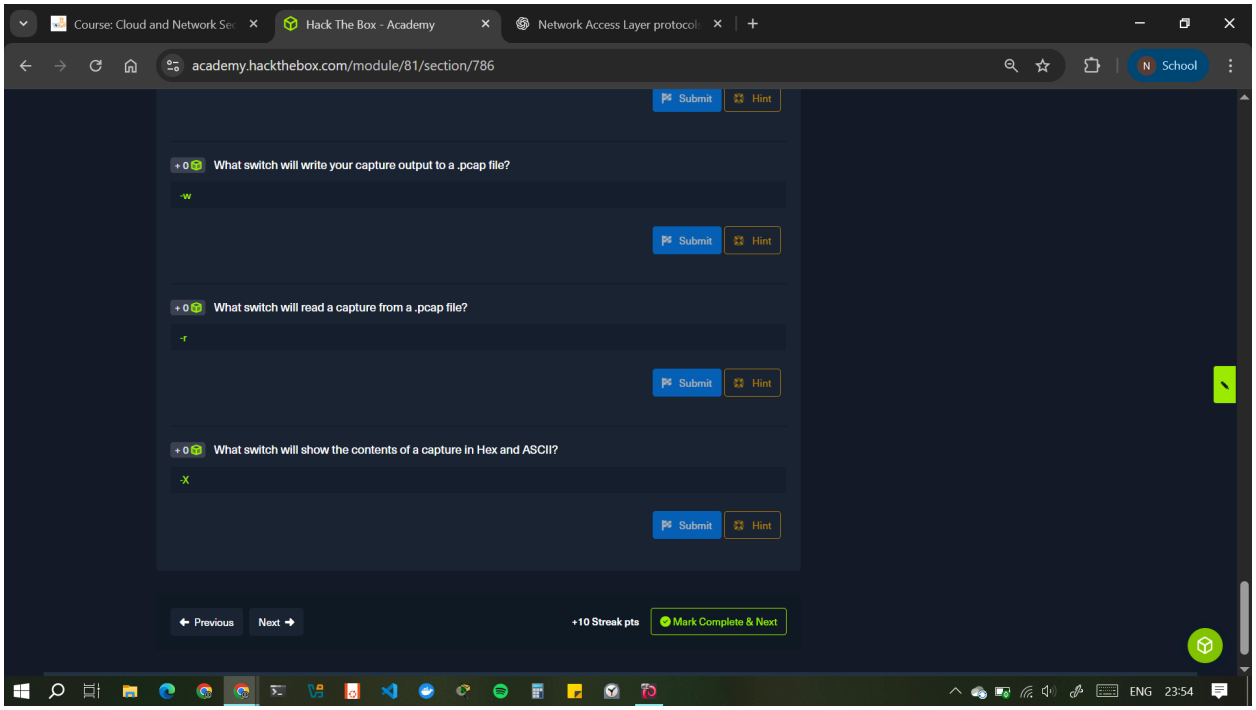
Questions

1. What TCPDump switch will allow us to pipe the contents of a pcap file out to another function such as 'grep'? **Answer: -l**
2. True or False: The filter "port" looks at source and destination traffic. **Answer: true**
3. If we wished to filter out ICMP traffic from our capture, what filter could we use? (word only, not symbol please.) **Answer: not icmp(filter out means remove from view)**
4. What command will show you where / if TCPDump is installed?**Answer: which tcpdump**

5. How do you start a capture with TCPDump to capture on eth0? **Answer: tcpdump -i eth0**
6. What switch will provide more verbosity in your output? **Answer: -v**
7. What switch will write your capture output to a .pcap file? **Answer: -w**
8. What switch will read a capture from a .pcap file? **Answer: -r**
9. What switch will show the contents of a capture in Hex and ASCII? **Answer: -X**







Tcpdump Packet filtering

Tcpdump provides a robust and efficient way to parse the data included in our captures via **packet filters**. They can be combined with standard tcpdump syntax options to capture as widely as we wish or be specific to capture packets from a particular host, or even with a particular bit in the TCP header set to on. They tell tcpdump which packets to capture/show.

Some advanced helpful TCPDump filters include:

Filter	Result
host	host will filter visible traffic to show anything involving the designated host. Bi-directional
src / dest	src and dest are **modifiers**. We can use them to designate a source or destination host or port.
net	net will show us any traffic sourcing from or destined to the network designated. It uses / notation.

proto	will filter for a specific protocol type. (ether, TCP, UDP, and ICMP as examples)
port	port is bi-directional. It will show any traffic with the specified port as the source or destination.
portrange	portrange allows us to specify a range of ports. (0-1024)
less / greater "< >"	less and greater can be used to look for a packet or protocol option of a specific size.
and / &&	and && can be used to concatenate two different filters together. for example, src host AND port.
or	or allows for a match on either of two conditions. It does not have to meet both. It can be tricky.
not	not is a modifier saying anything but x. For example, not UDP.

Pre-Capture Filters VS. Post-Capture Processing

Pre-capture filters: any traffic not matching the filter is dropped. The amount of data in the captures is reduced and potentially, so is traffic that may be needed later. Hence, they are best used when looking for something specific such as troubleshooting a network connectivity issue.

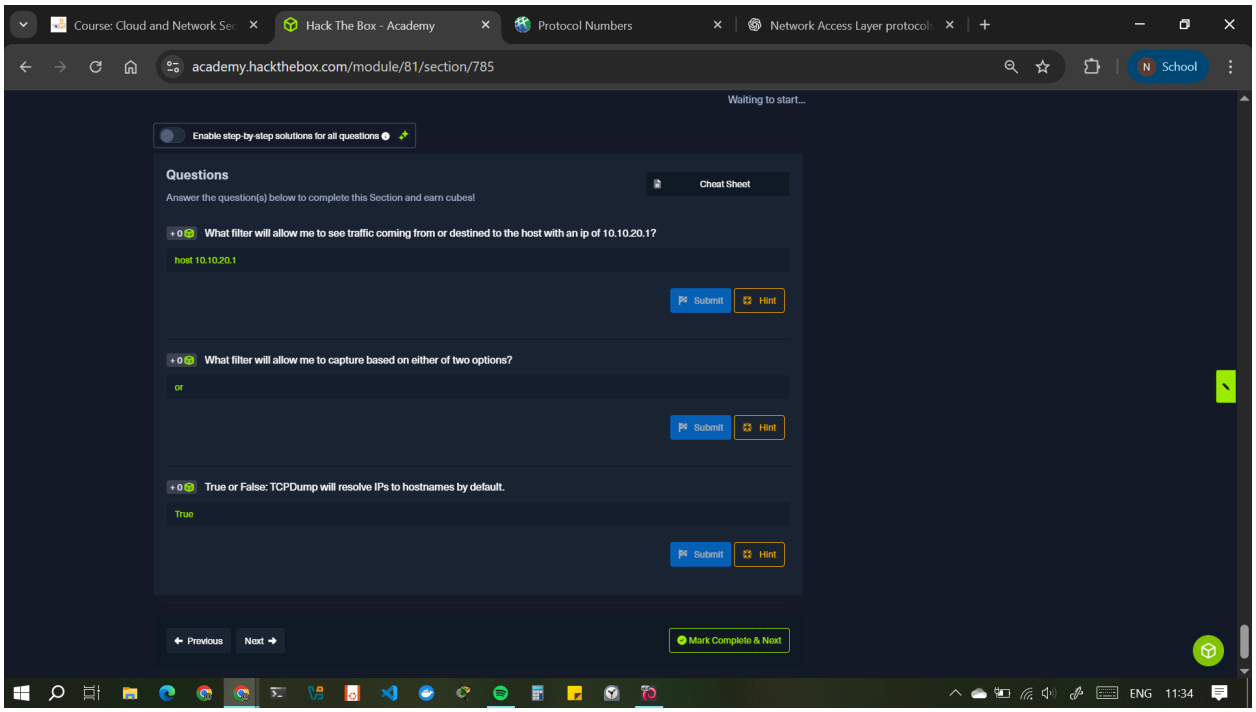
Post-capture processing: applying a filter to capture that is read from a file will parse the file and remove anything from our terminal output not matching the specified filter. It helps us investigate while saving potential valuable data in the captures because it will not permanently change the capture file.

Looking for TCP Protocol Flags

Command used to look for a SYN flag: `tcpdump -i eth0 'tcp[13] &2 != 0'`

- **Tcp[13]:** look at byte **offset 13** of the TCP header (which is the **flags field**).
- **&2:** check the second flag which is SYN by applying bitwise AND
- **!= 0**: if the result is non-zero() i.e. 1 or ON, is that bit set?

Results would include only packets with the TCP SYN flag set.



Lab: Interrogating Network Traffic With Capture and Display Filters

This lab aims to provide some exposure to interrogating network traffic and practice in implementing packet filters such as **host, port, protocol,** and many more to change our view while digging through a .PCAP file.

Scenario

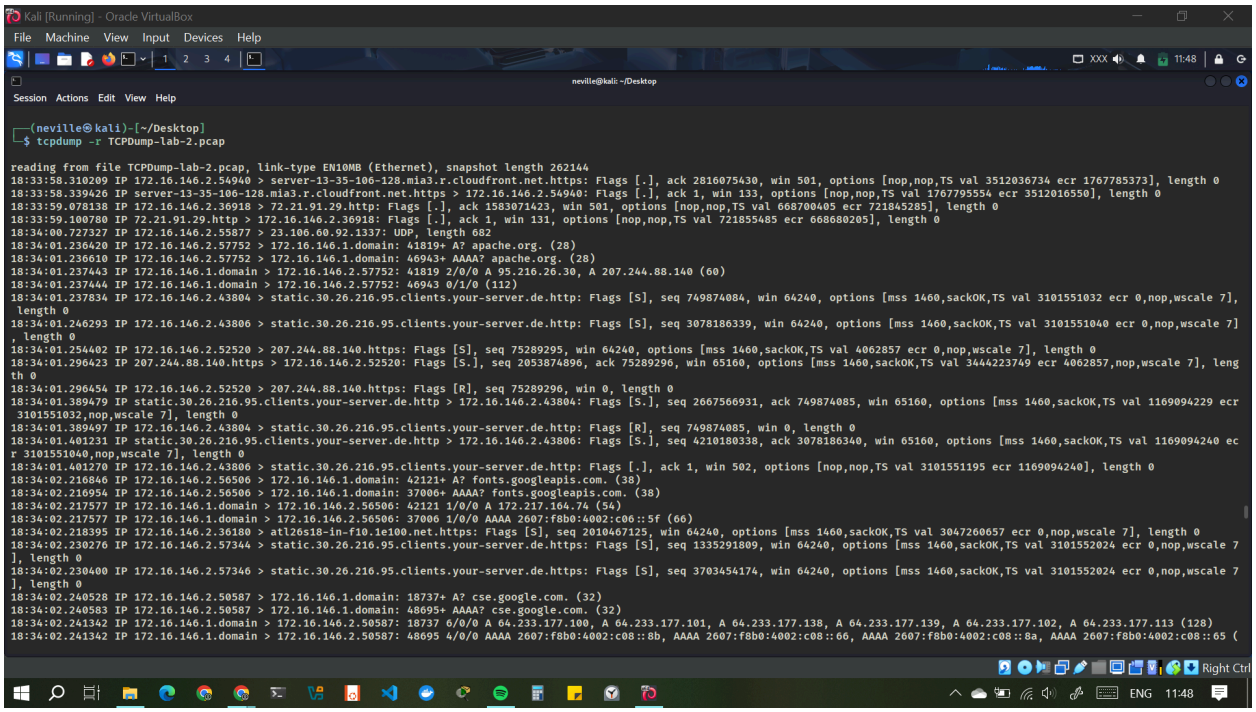
Now that we have proven capable of capturing network traffic for the Corporation, management has tasked us with performing a quick analysis of the traffic our team has captured while surveying the network. The goal is to determine what servers are answering DNS and HTTP/S requests in our local network.

Objective

Determine what servers are answering DNS and HTTP/S requests in our local network from **TCPDump-lab-2.pcap**

Tasks

1. Read a capture from a file without filters implemented.

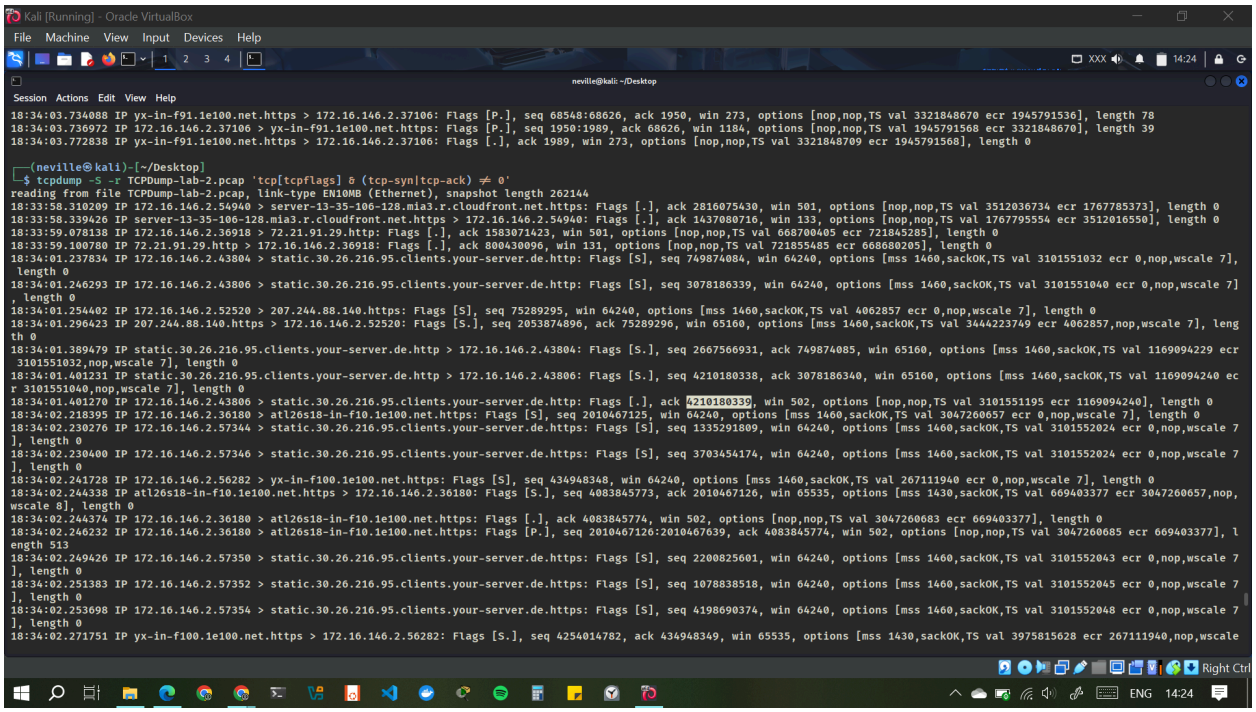


2. Identify the type of traffic seen.

- Common protocols in the traffic are: **DNS, HTTP, HTTPS**
- Ports utilized: **UDP 53, TCP 80, TCP 443, UDP 1337**
- Filters to use: **tcp, udp, port, host, tcp[13] & 8 != 0 (Show only TCP packets that have the PSH flag set (i.e., packets carrying data, not just ACKs).**

3. Identify conversations.

    - What are the client and server port numbers used in the first full TCP three-way handshake? **Client port = 43806, server port = 80. Switch -S helps to show absolute sequence numbers to identify the first full TCP three-way handshake.**



- Who are the servers in these conversations? How do we know?

  Servers typically communicate over the well-known port number assigned to the protocol (80 for HTTP, 443 for HTTPS, for example).

- Who are the receiving hosts?

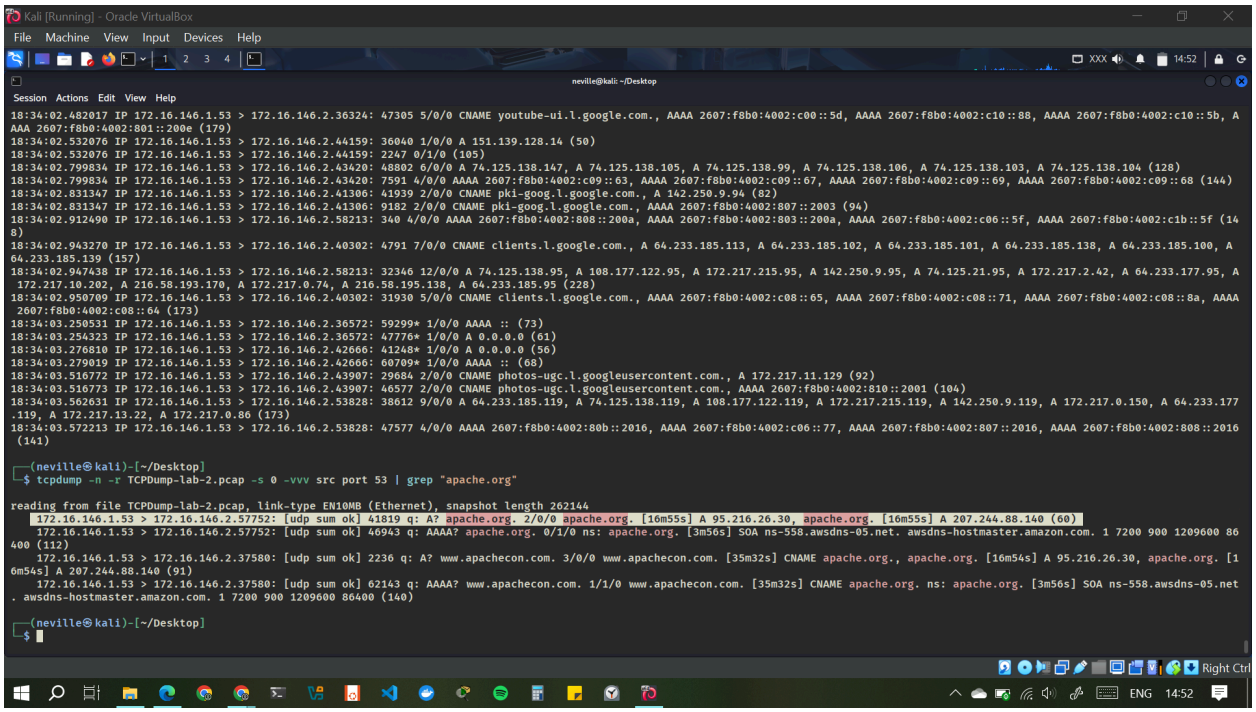
  Hosts or recipients in the conversations will typically utilize a random high port number.

4. Interpret the capture in depth.
   - What is the timestamp of the first established conversation in the pcap file?

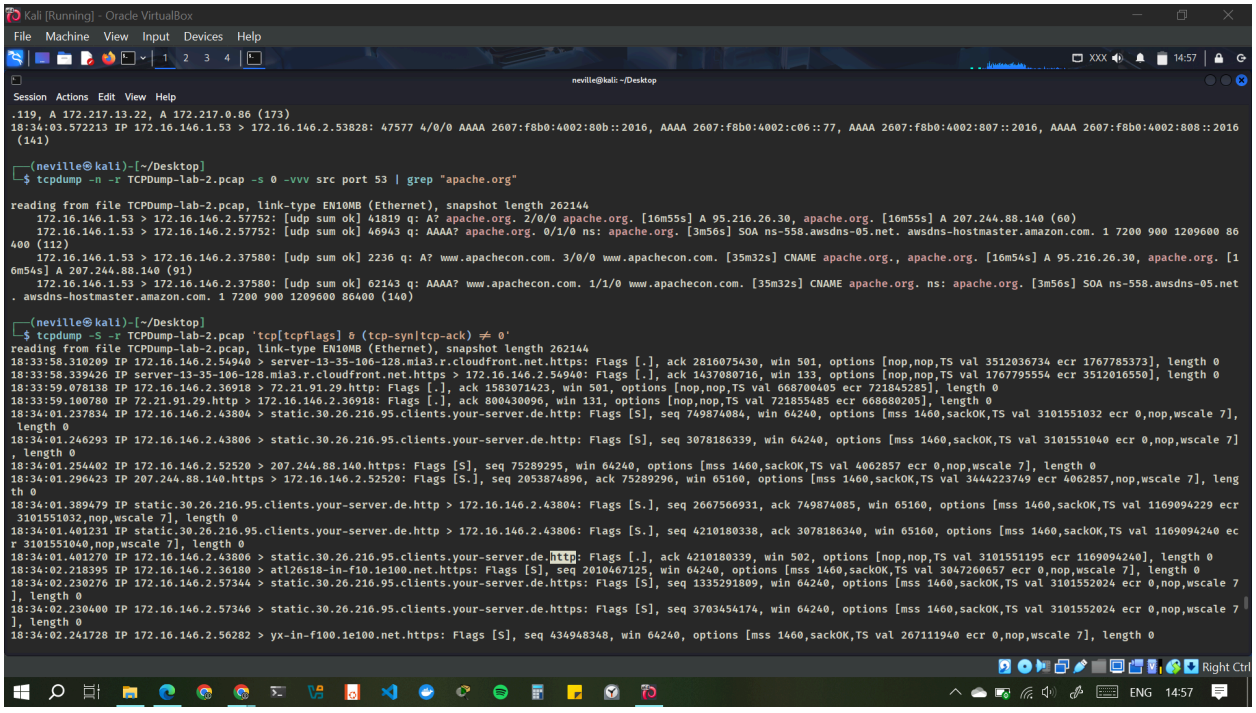
     Timestamp is the first field in the output - 18:34:01.401270 is the timestamp of the first established connection

   - What is the IP address/s of apache.org from the DNS server responses?

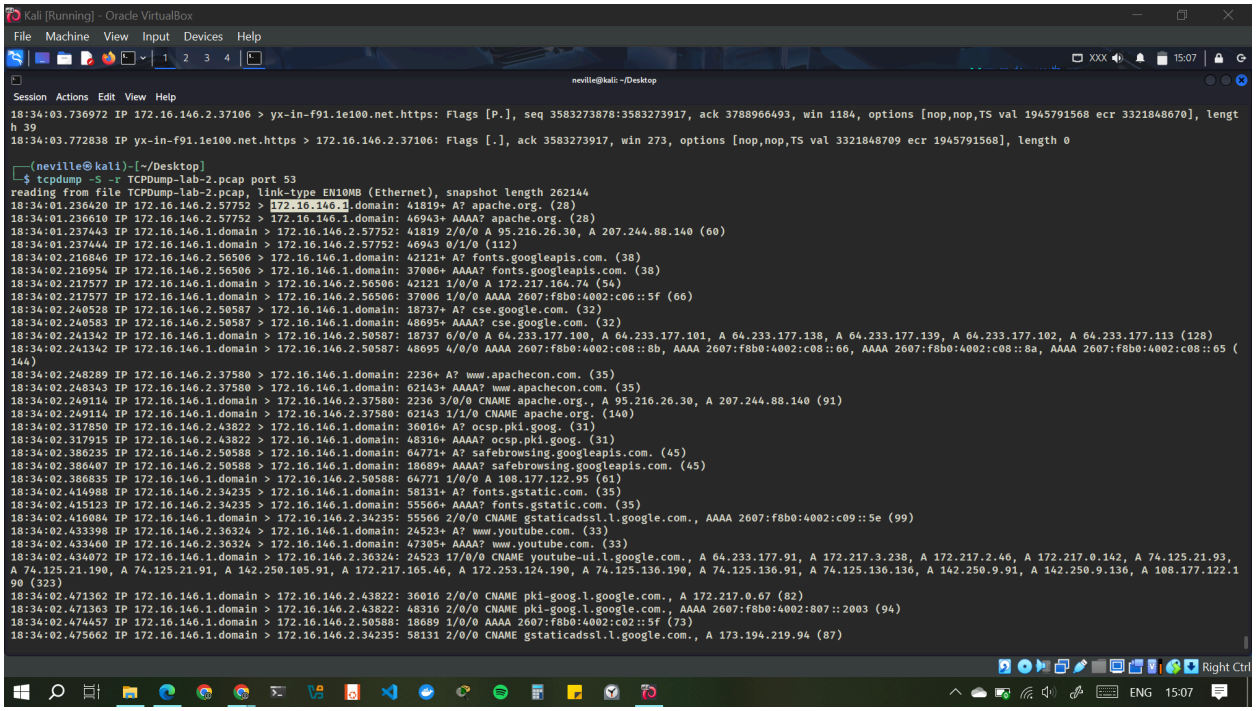
     We can filter the traffic only to see conversations Sourcing from it (src 'apache.org') and then disable Name resolution with (-n). The IP addresses of apache.org are: 95.216.26.30 and 207.244.88.140



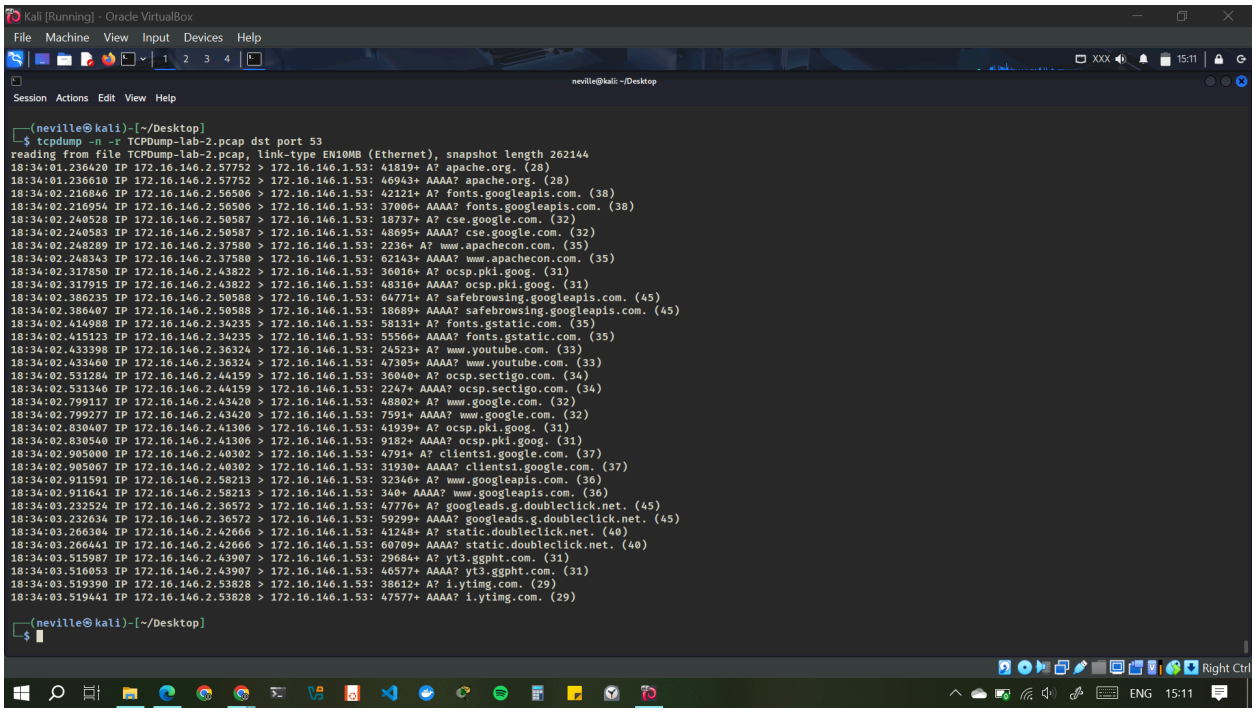
- What protocol is being utilized in that first conversation? **http/80**



5. Filter out traffic to show only DNS traffic.
    - Who is the DNS server for this segment? **172.16.146.1**



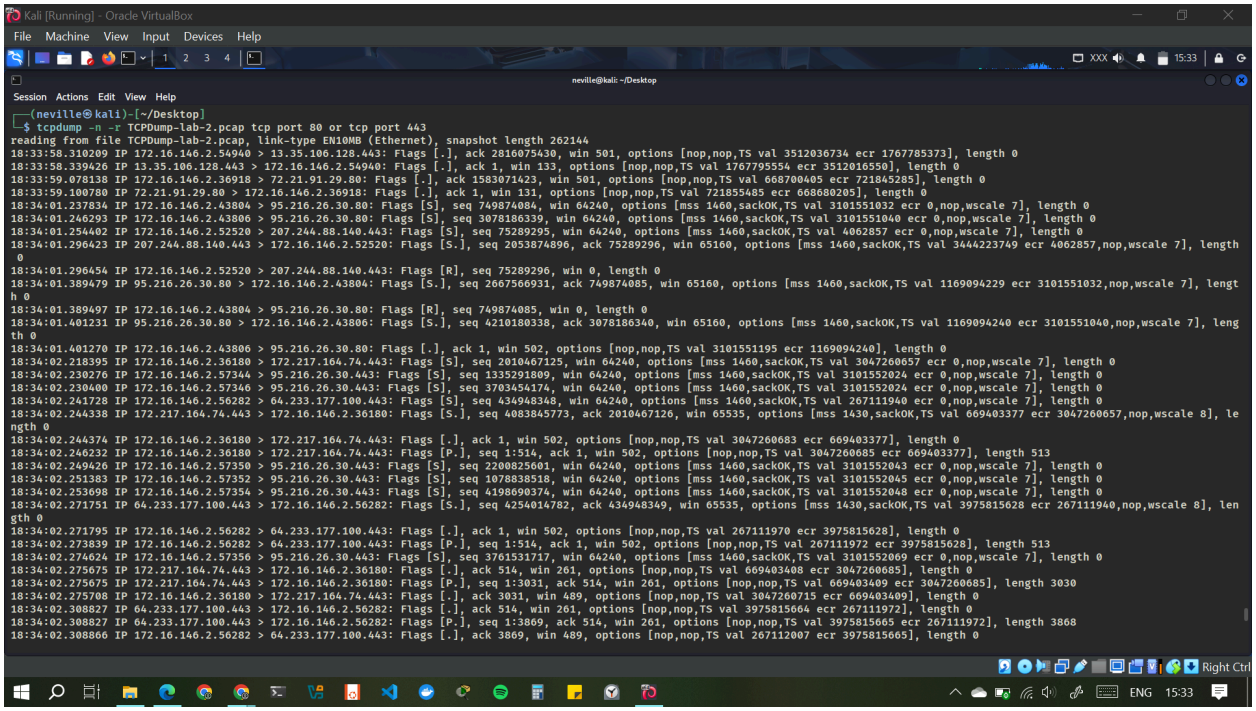
- What domain name/s were requested in the pcap file?



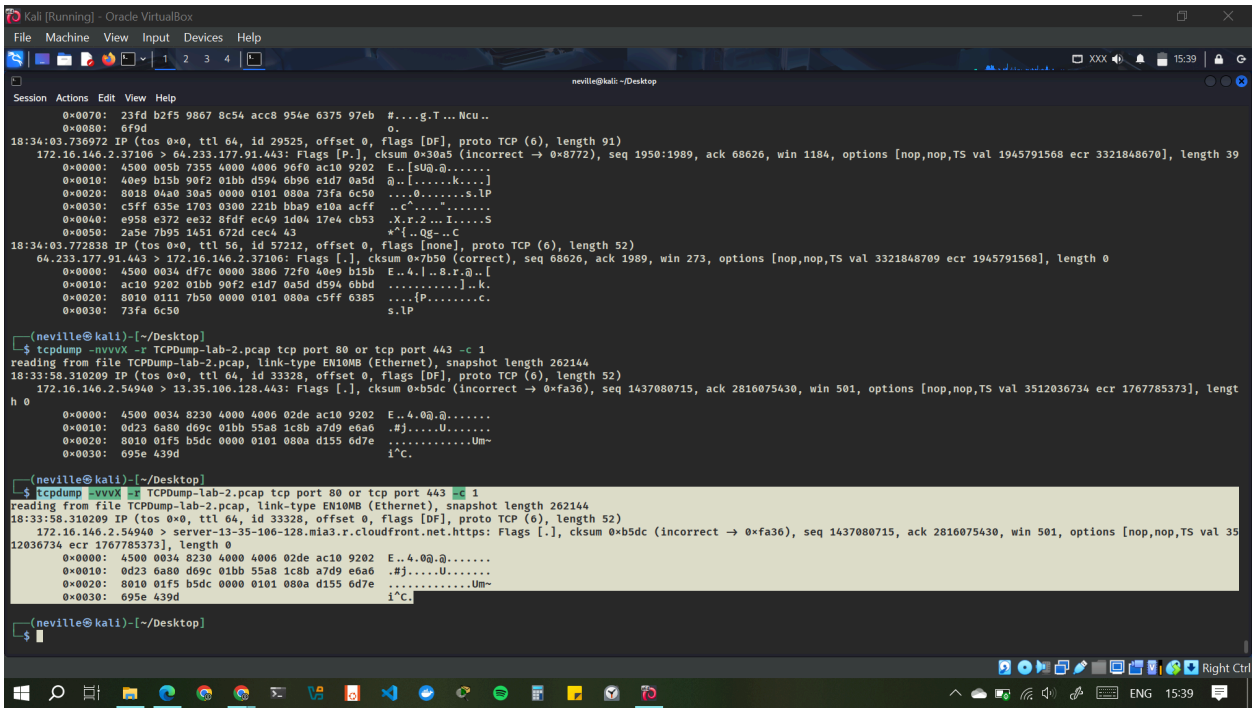
- What type of DNS Records could be seen? **A, AAAA, CNAME(Canonical Name) A record provides IPv4 address of hostname, CNAME creates an alias for one domain name to another.**

6. Filter for TCP traffic(HTTP/HTTPS).

   - What web pages were requested? **/gts1o1core**
   - What are the most common HTTP request methods from this PCAP? **POST**
   - What is the most common HTTP response from this PCAP? **200 OK**



7. What can you determine about the server in the first conversation?
   - **The server is utilizing protocol HTTPS which encrypts traffic after the TLS handshake.**
   - **The server is Amazon CloudFront, which is a Content Delivery Network (CDN) used to host and deliver web content. It is not a traditional webserver.**
   - **TTL = 242 in the response suggests this is a large-scale provider (common in CDN infrastructures).**

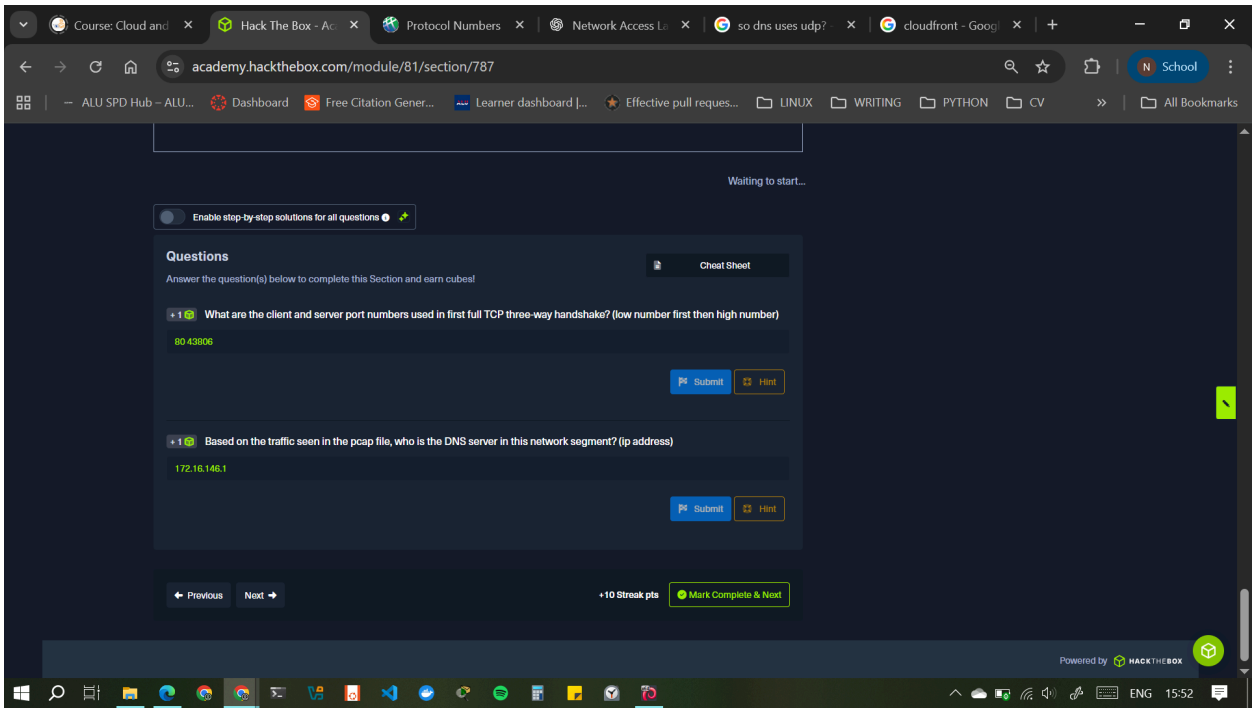


Tips For Analysis

Below is a list of questions we can ask ourselves during the analysis process to keep on track.

1. what type of traffic do you see? (protocol, port, etc.)
2. Is there more than one conversation? (how many?)
3. How many unique hosts?
4. What is the timestamp of the first conversation in the pcap (tcp traffic)
5. What traffic can I filter out to clean up my view?
6. Who are the servers in the PCAP? (answering on well-known ports, 53, 80, etc.)
7. What records were requested or methods used? (GET, POST, DNS A records, etc.)

Questions



Conclusion

This lab enhanced my practical understanding of tcpdump to analyze PCAP traffic by learning how to capture and display filters effectively, dissect traffic to determine what protocols were running in the environment, and even gleaned some critical information about our enterprise segments, DNS, and Web servers.

Analysis with Wireshark

Features and Capabilities:

- Deep packet inspection for hundreds of different protocols
- Graphical and TTY interfaces
- Capable of running on most Operating systems
- Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI, among others
- Decryption capabilities for IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, and WPA/WPA2

- Etc

TShark VS. Wireshark (Terminal vs. GUI)

TShark is a purpose-built terminal tool based on Wireshark. TShark shares many of the same features that are included in Wireshark and even shares syntax and options. TShark is perfect for use on machines with little or no desktop environment and can easily pass the capture information it receives to another tool via the command line. Wireshark is the feature-rich GUI option for traffic capture and analysis. If you wish to have the full-featured experience and work from a machine with a desktop environment, the Wireshark GUI is the way to go.

Basic TShark Switches

Switch Command	Result
D	Will display any interfaces available to capture from and then exit out.
L	Will list the Link-layer mediums you can capture from and then exit out. (ethernet as an example)
i	choose an interface to capture from. (-i eth0)
f	packet filter in libpcap syntax. Used during capture to allow for filter use.
c	Grab a specific number of packets, then quit the program. Defines a stop condition.
a	Defines an autostop condition. Can be after a duration, specific file size, or after a certain number of packets.
r (pcap-file)	Read from a file.
W (pcap-file)	Write into a file using the pcapng format.
P	Will print the packet summary while writing into a file (-W)
x	will add Hex and ASCII output into the capture.
h	See the help menu

Note: Utilizing TShark is very similar to TCPDump in the filters and switches we can use. Both tools utilize BPF syntax.

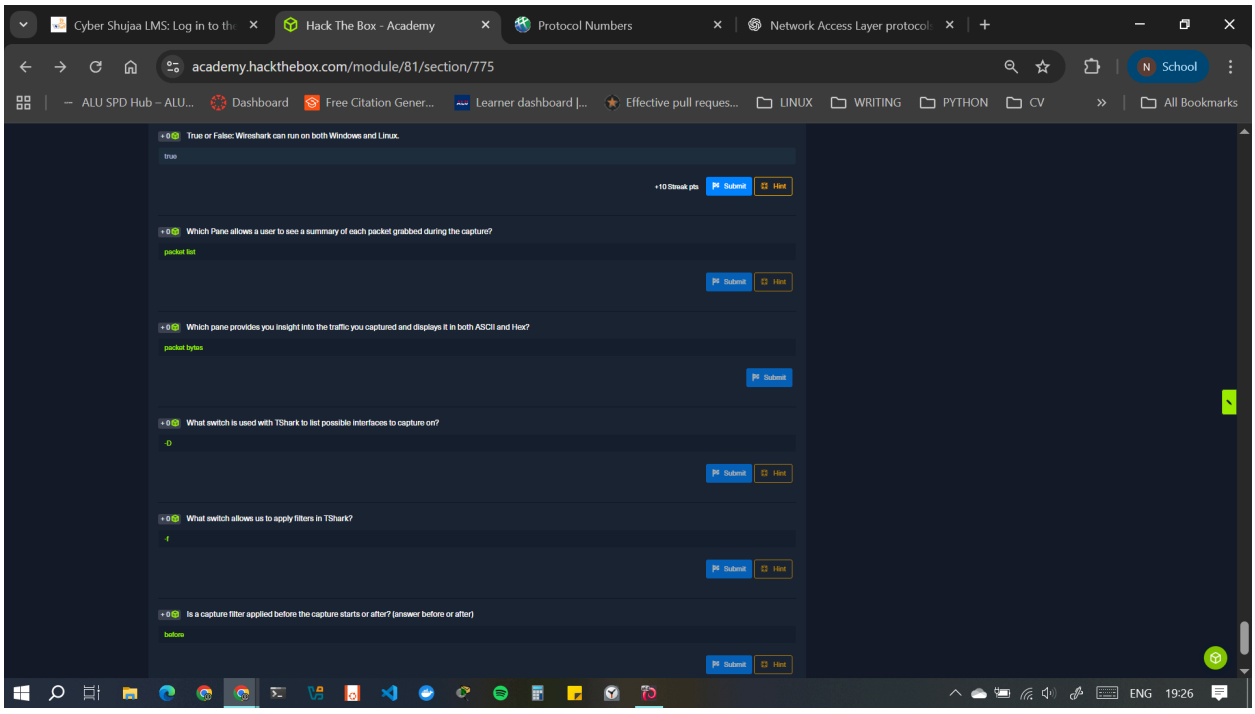
Termshark

Termshark is a Text-based User Interface (TUI) application that provides the user with a Wireshark-like interface right in your terminal window.

Termshark can be found at Termshark. It can be built from the source by cloning the repo, or pull down one of the current stable releases from https://github.com/gcla/termshark/releases , extract the file, and hit the ground running.

To start Termshark, issue the same strings, much like TShark or tcpdump. We can specify an interface to capture on, filters, and other settings from the terminal. The Termshark window will not open until it senses traffic in its capture filter. So give it a second if nothing happens.

Questions



Familiarity with Wireshark

Introduction

This lab aims to give Wireshark a basic familiarity and utilize its graphical interface to perform traffic captures. We will spend time using capture and display filters and getting used to the different outputs shown by the tool.

Scenario

A user has brought their laptop to the helpdesk complaining of unusually long load times when they try to surf the web. They said it is almost as if the network is bogged down and the computer is making many different connections. We have been tasked with validating that the pc is functioning correctly. To do so, we connect it to the network and start a packet capture while surfing the web. Analyze what can be seen in the output to determine if something is amiss. We only care about the laptop in question at this time, so filter out any traffic not destined to or sourcing from it.

Tasks

Task #1

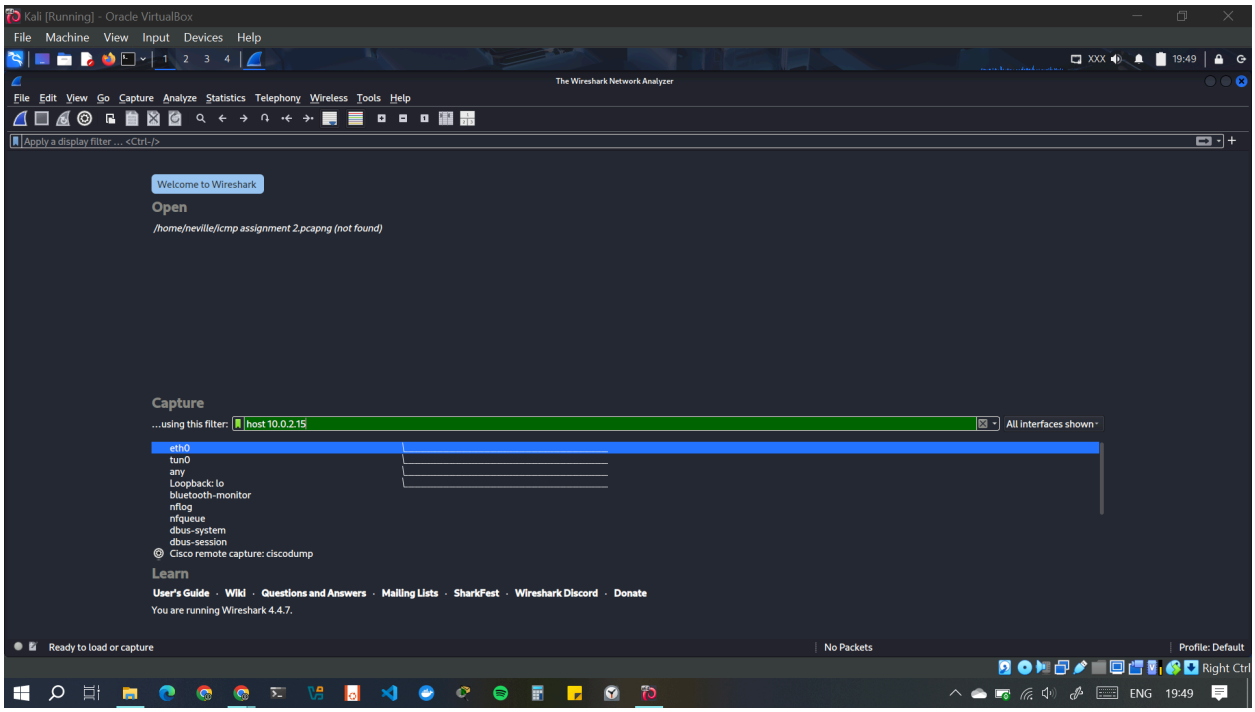
Validate Wireshark is installed, then open Wireshark and familiarize yourself with the GUI windows and toolbars.

Take a minute and explore the Wireshark GUI. Ensure we know what options reside under which tabs in the command menus. Please pay special attention to the Capture tab and what resides within it.

Task #2

Select an interface to run a capture on and create a capture filter to show only traffic to and from your host IP.

Choose your active interface (eth0, or your Wifi card) to capture from.



Task #3

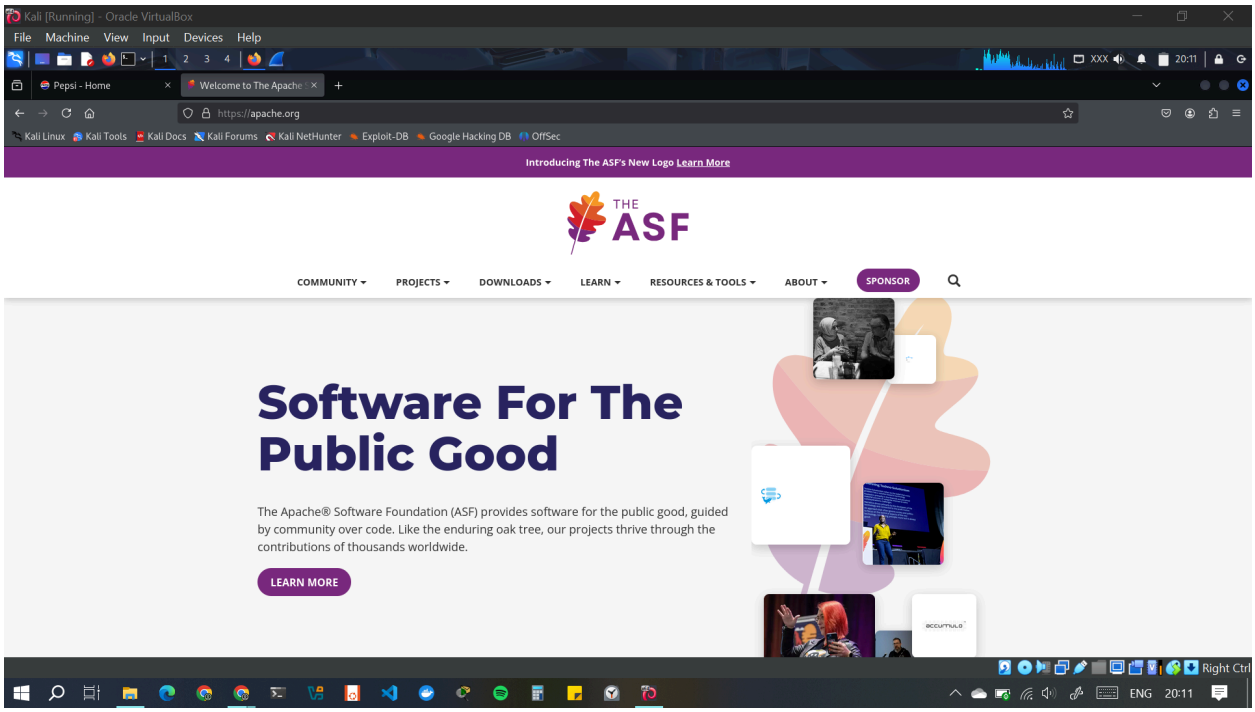
Create a capture filter.

Next, we want to create a capture filter to only show us traffic sourcing from or destined to our IP address and apply it.

Task #4

Navigate to a webpage to generate some traffic.

Open a web browser and navigate to pepsi.com. Repeat this step for http://apache.org. While the page is loading, switch back to the Wireshark window. We should see traffic flowing through our capture window. Once the page has loaded, stop the capture by clicking on the red square labeled Stop in the action bar.



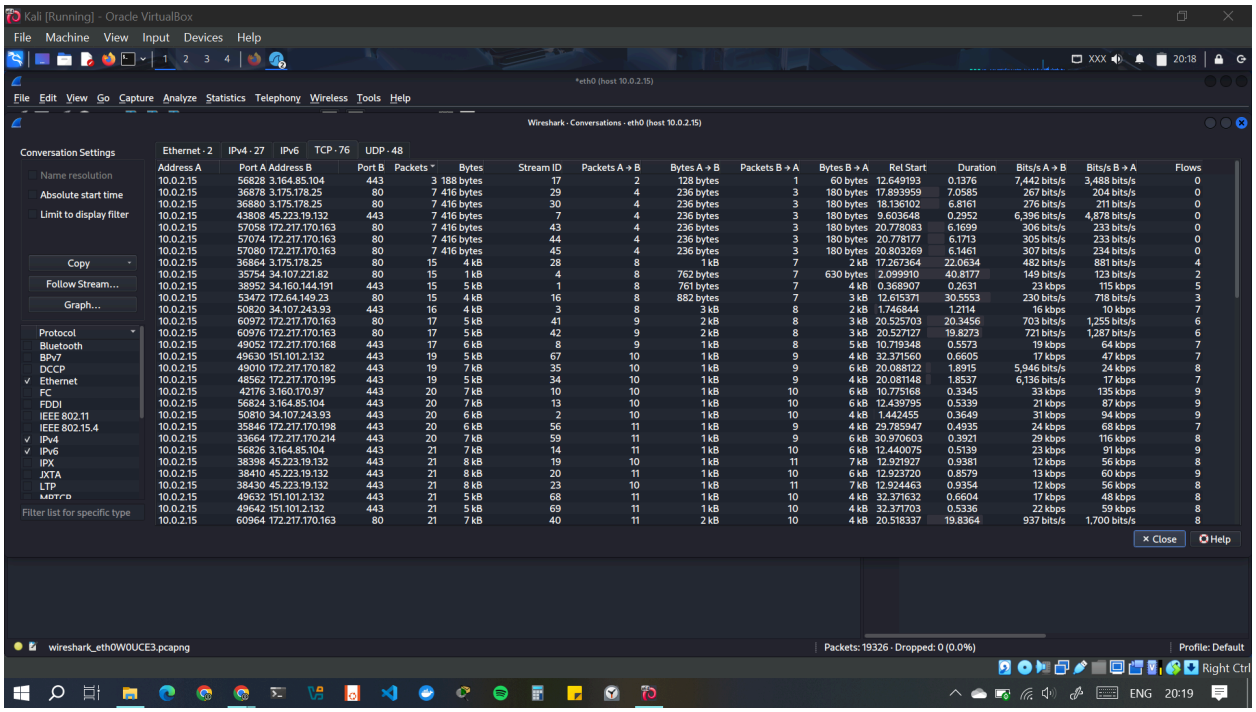
Task #5

Use the capture results to answer the following questions.

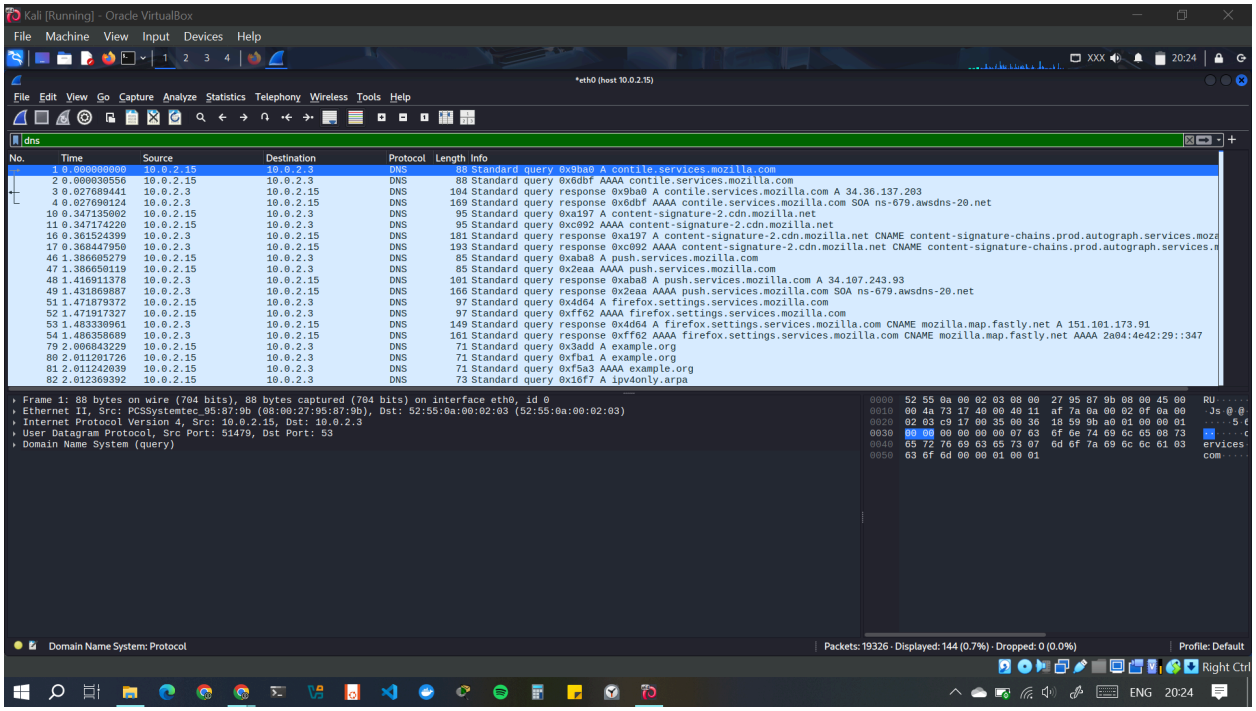
1. Are multiple sessions being established between the machine and the webserver? How can you tell?

   Yes. In Menu → Statistics → Conversations → TCP — sort by Packets or Bytes. There are multiple rows with the same server IP but different Source ports from the client, which means the client opened multiple TCP sessions to that server.

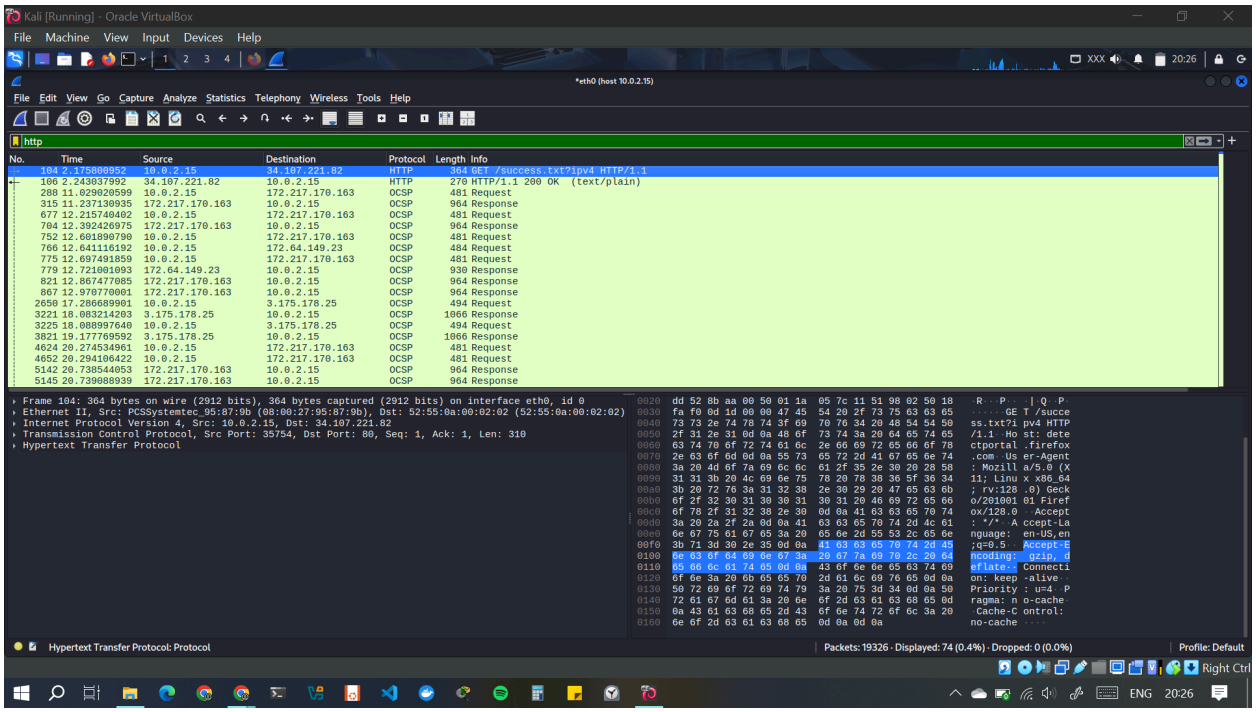
   Browsers typically open multiple TCP sessions to a webserver for parallel resource fetching; confirmed by seeing multiple SYNs to the server IP with different client (source) ports or many TCP conversations in the Conversations view.



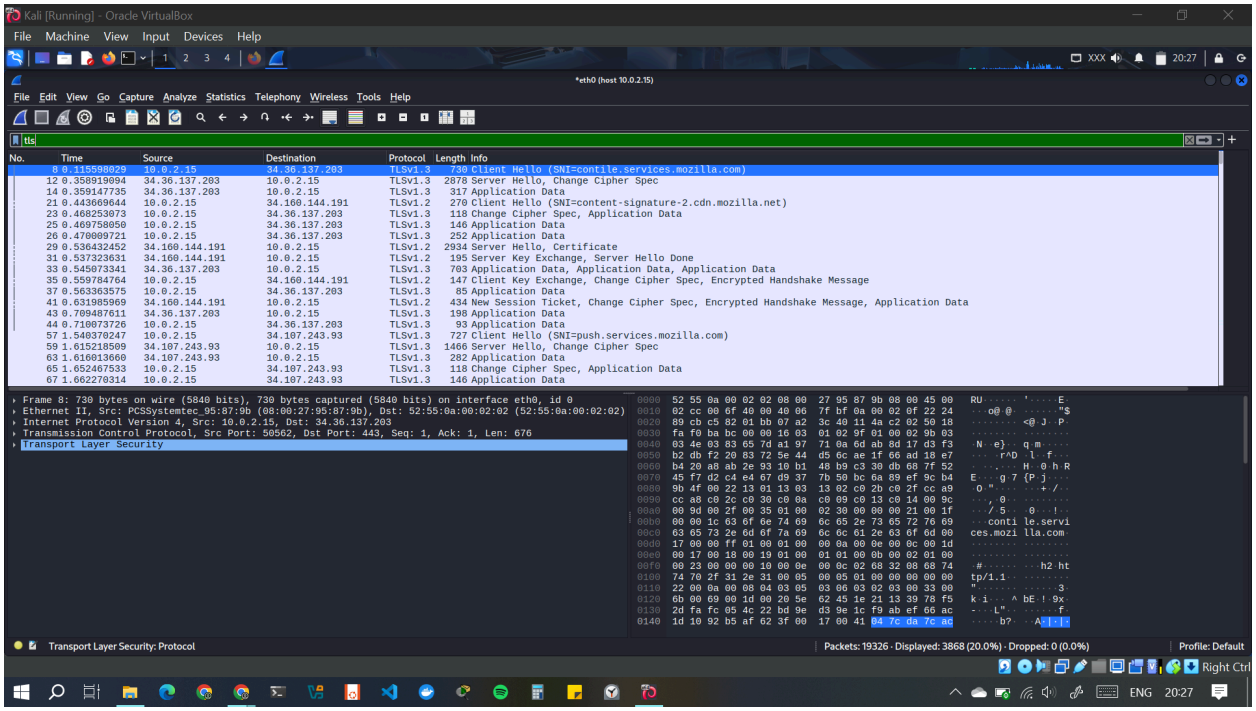
2. What application-level protocols are displayed in the results?
    - DNS



    - HTTP



- TLS



3. Can we discern anything in clear text? What was it?

Output from apache.org will be in plain text. This happens because HTTP is not encrypted. Looking at HTTPS traffic will be much different because it is encrypted(It appears as application data)

What is visible:

- **Plain HTTP (port 80): full URL, request method (GET/POST), Host header, User-Agent, cookie content, response headers and body.**
- **SNI (Server Name Indication) in the TLS Client Hello → reveals the hostname requested (filter: tls.handshake.extensions_server_name or ssl.handshake.extensions_server_name).**

Conclusion

This lab enhanced my understanding of how wireshark looks and operates. I have familiarized myself with the command bar and action bar and what resides within each respective tab. Furthermore, I understand how to select an interface to capture with and successfully start a capture with a filter applied as well as using display filters during and after the capture.

Wireshark Advanced Usage

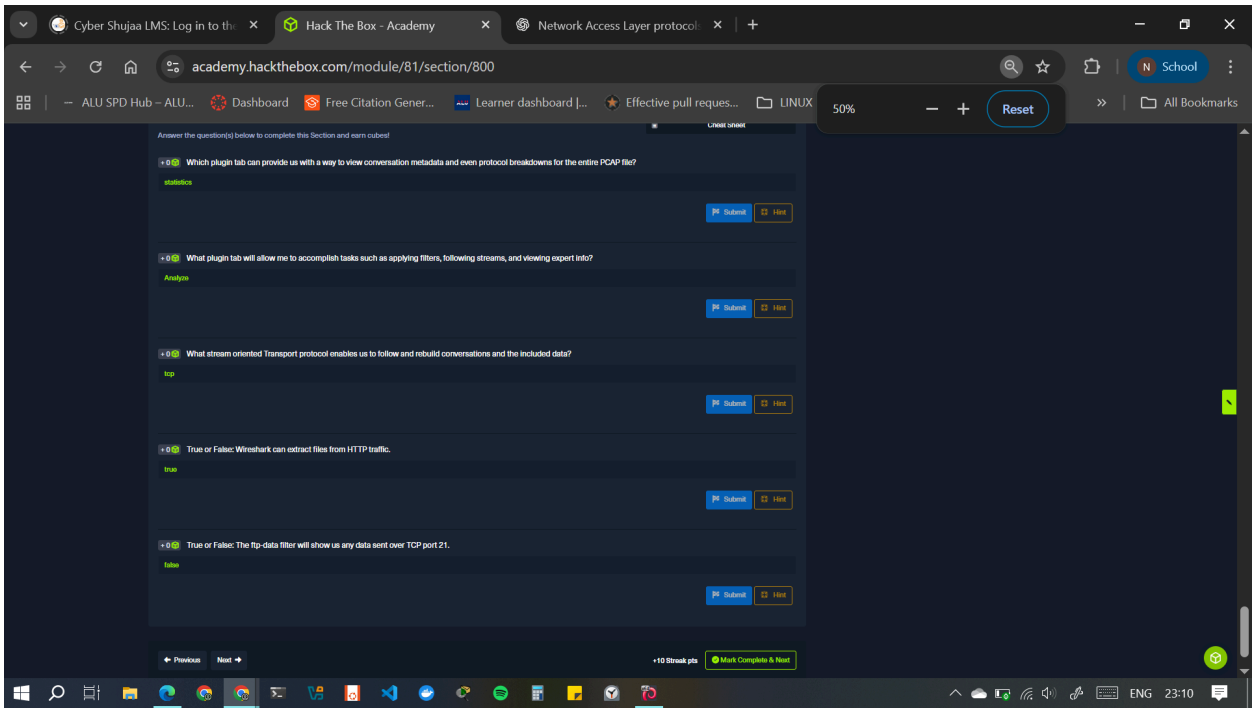
Some of the many different capabilities in Wireshark range from tracking TCP conversations to cracking wireless credentials. The inclusion of many different plugins makes Wireshark one of the best traffic analysis tools.

The **analyze and statistics radials** provide a plethora of plugins to run against the capture. They can provide us with great insight into the data we are examining. From these points, we can utilize many of the baked-in plugins Wireshark has to offer; the plugins give us detailed reports about the network traffic being utilized e.g top talkers in our environment, specific conversations, breakdown by IP and protocol.

The plugins can be used in:

1. Following TCP Streams(Alternatively, we can utilize the filter tcp.stream eq # to find and track conversations captured in the pcap file)
2. Filter For A Specific TCP Stream.
3. Extracting Data and Files From a Capture
4. Extract Files From The GUI using FTP which moves data between a server and host to pull it out of the raw bytes and reconstruct the file. (image, text documents, etc.)

Questions



Packet Inception, Dissecting Network Traffic With Wireshark

The purpose of this lab is to provide experience with dissecting traffic in Wireshark. I had the chance to pull objects out of previously captured network traffic along with pulling data from live traffic.

Scenario

We have been provided with a packet capture file that contains **data from an unencrypted web session**. There is an image embedded that needs to be used as evidence of improper network usage. The Security manager thinks the user is sending messages hidden behind the image. Using Wireshark, apply filters to locate and extract the evidence.

Tasks

Task #1

Open a pre-captured file (HTTP extraction)

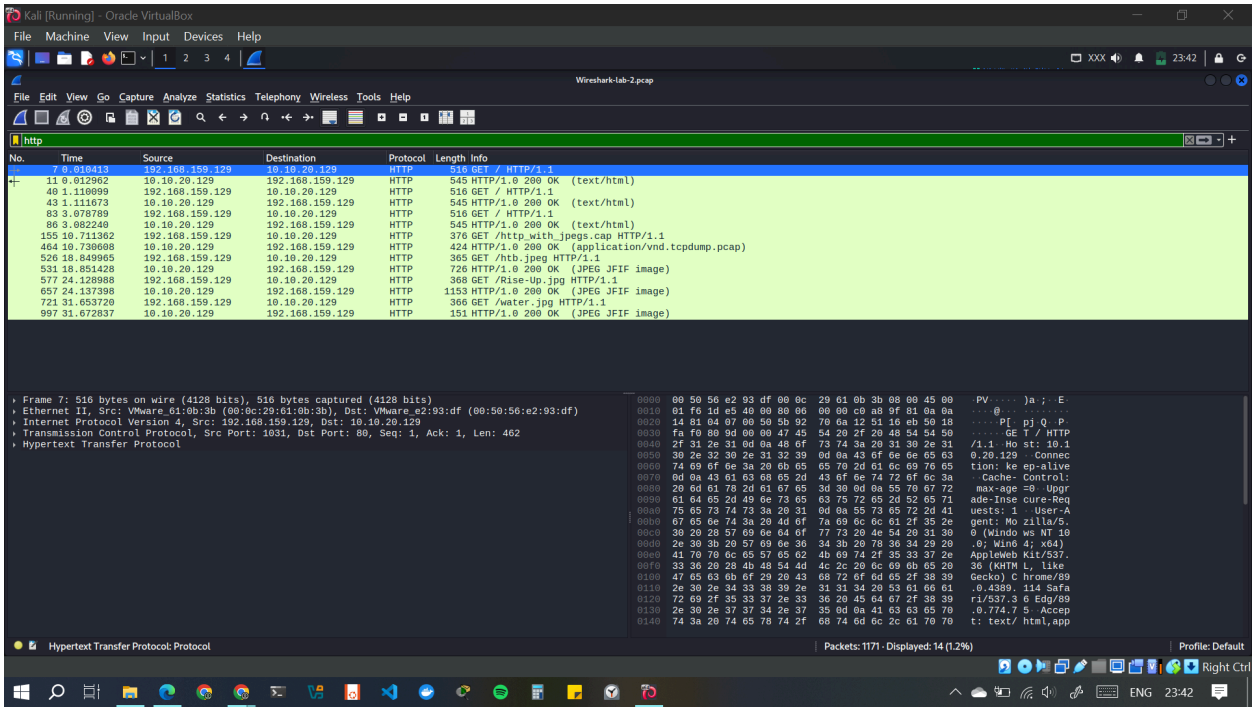
In Wireshark, Select File → Open → , then browse to Wireshark-lab-2.pcap. Open the file.

Task #2

Filter the results.

Now that we have the pcap file open in Wireshark, we can see quite a lot of traffic within this capture file. It has around 1171 packets total, and of those, less than 20 are HTTP packets specifically. Take a minute to examine the pcap file, become familiar with the conversations being had while thinking of the task to accomplish. Our goal is to extract potential images embedded for evidence. Based on what has been asked of us, let's clear our view by filtering for HTTP traffic only.

Apply a filter to include only HTTP (80/TCP) requests. From here, we can see several basic HTTP datagrams containing the GET method and 200 OK responses. These are interesting because we can now see that a client requested several files, and the server responded with an OK. If we select one of the OK responses, we can follow that stream and see the data transfer over TCP.



Task #3

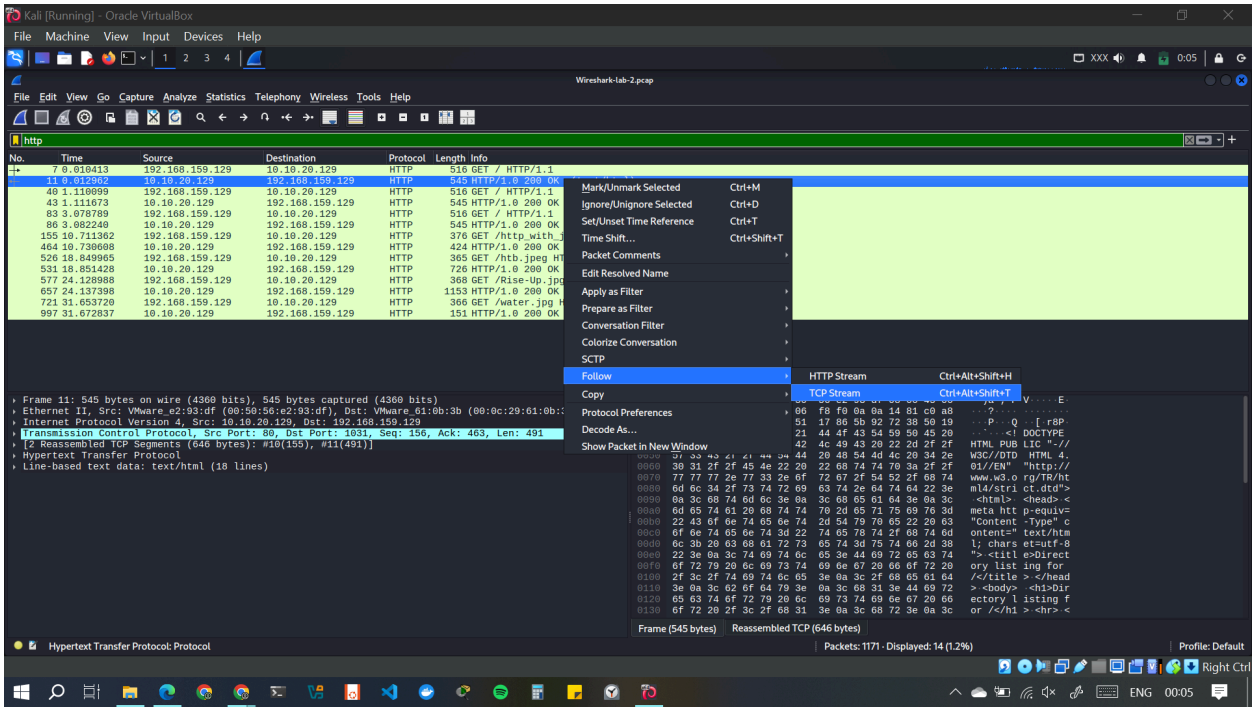
Follow the stream and extract the item(s) found.

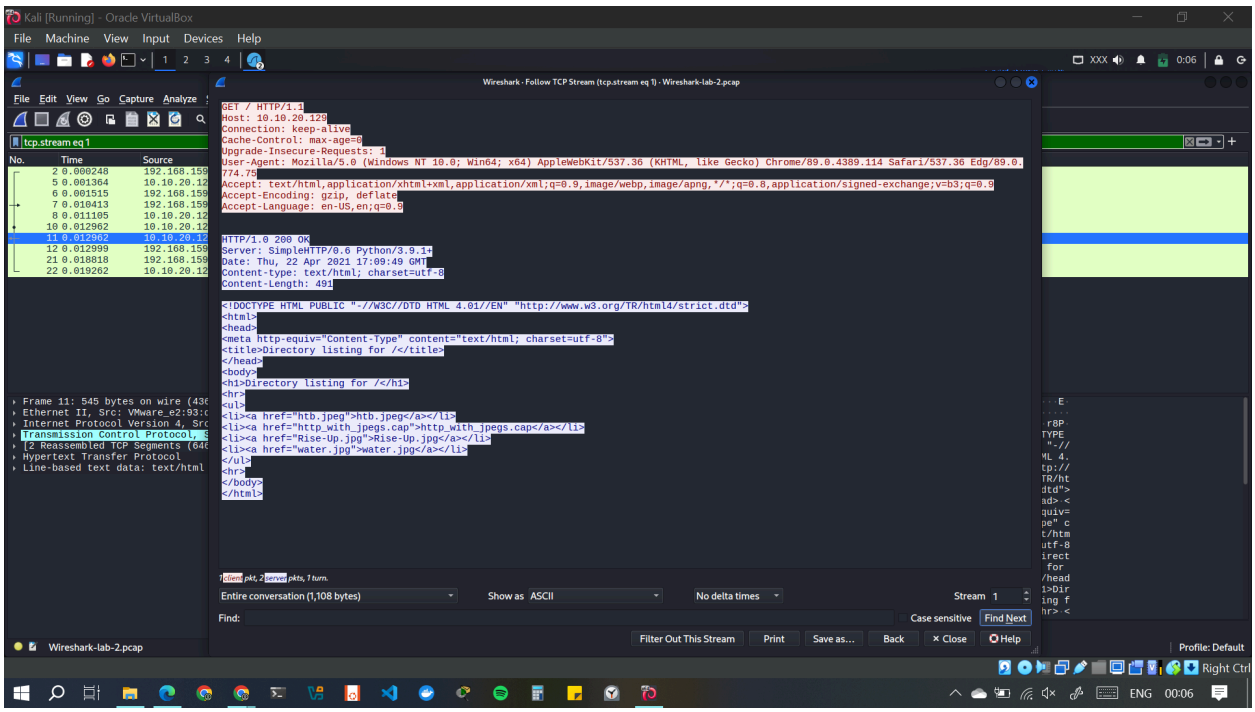
So now that we have established there is HTTP traffic in this capture file, let's try to grab some of the items inside as requested. The first thing we need to do is follow the stream for one of the file transfers.

- With our http filter still applied, look for one of the lines in which the Web Server responds with a “200 OK” message which acts as an acknowledgment/receipt to a users’ GET request. Now let's select that packet and follow the TCP stream.

  Step 1: Select a packet with 200 OK in the info field. Right-click the packet.

Step 2: From the menu presented, select Follow → TCP Stream. This will open up a new window with the entire TCP stream in it. It will also apply the display filter tcp.stream eq #.

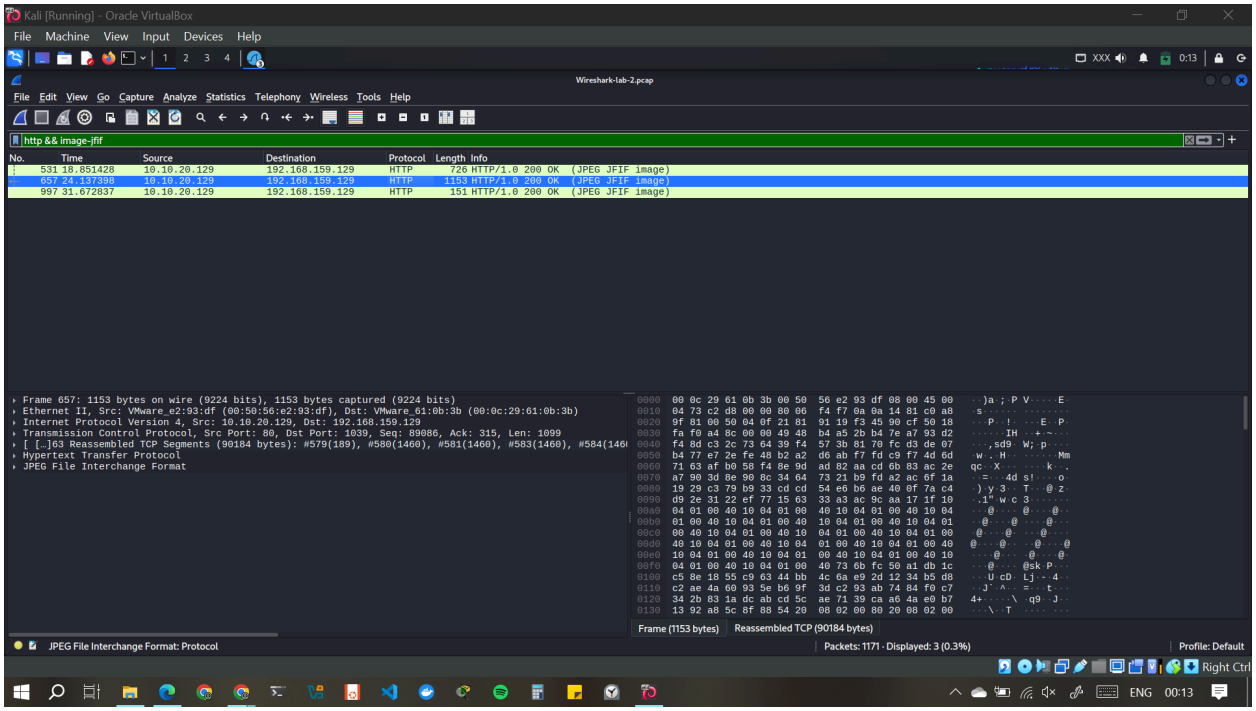




- Now that we validated the transfer happened, Wireshark can make it extremely easy to extract files from HTTP traffic. We can check to see if an image file was pulled down by looking for the JFIF format in the packets. The JPEG File Interchange Format JFIF will alert us to the presence of any JPEG image files. We are looking for this format because it is the most common file type

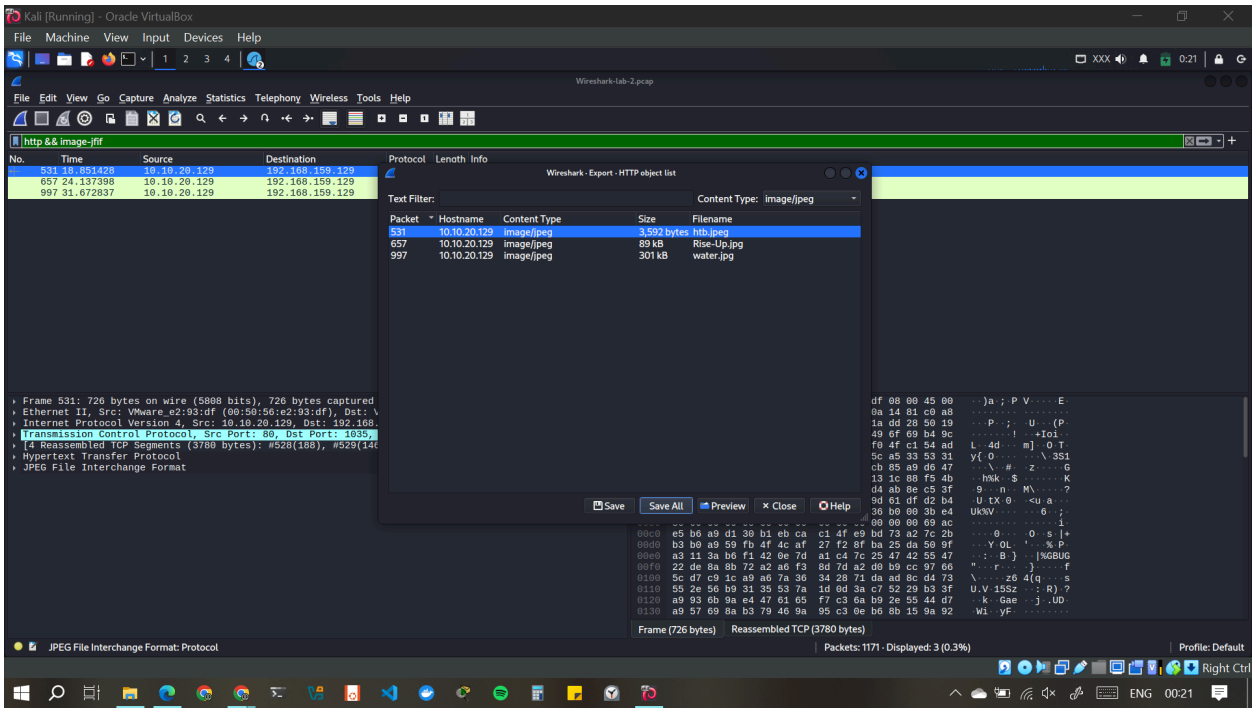
for images alongside the png format. With that in mind, we will likely see an image in this format for our investigation.

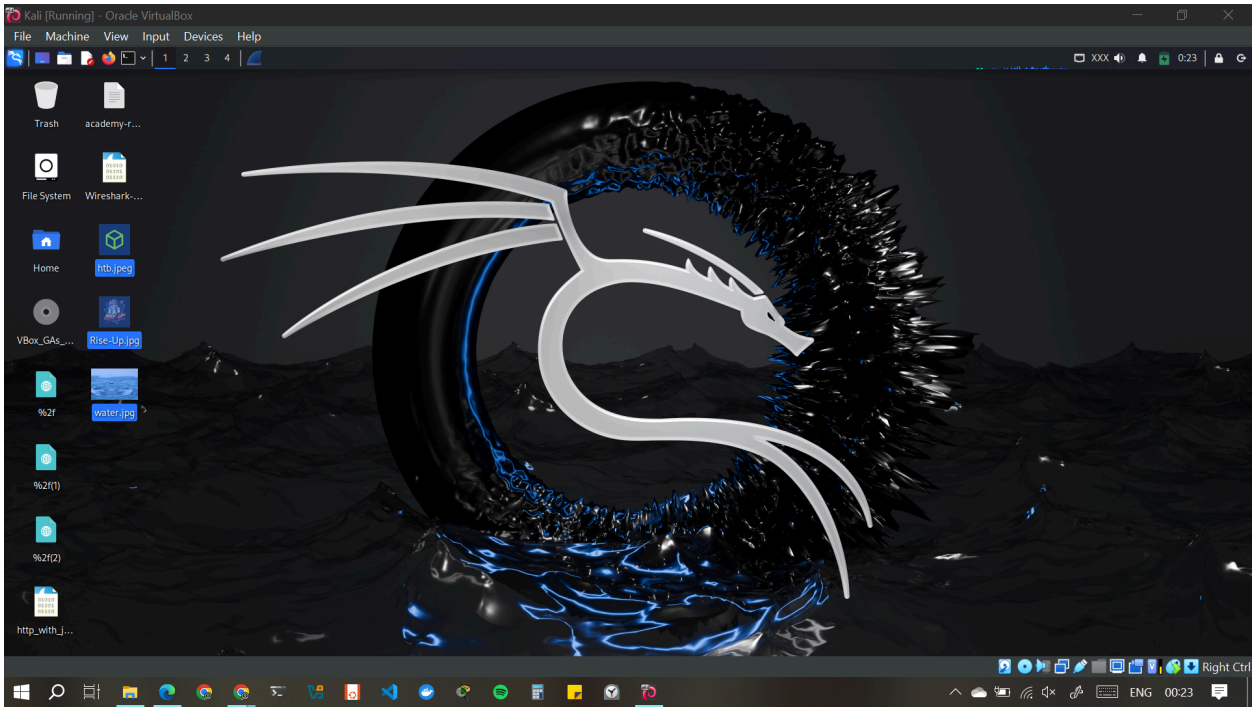
Check for the presence of JFIF files in the HTTP traffic using display filter: **http && image-jfif**



- Now that we are sure image files were transferred between the suspicious host and the server let's grab them out of the capture. To do this, we need to export the objects out of the HTTP traffic.

  To export the images: **Select "File → Export Objects → HTTP → {file}.JPEG**. This will tell Wireshark to pull the objects requested out of the HTTP traffic. We can save a copy of the file locally.





- At this point, we should now have the image files that our security manager requested us to capture if they existed. They can now examine the file to determine if any data was hidden within it.

Live Capture and Analysis

In the scenario above, we practiced filtering on a pre-captured file. Now it's time to do some live packet captures. We will connect to the academy lab and sniff traffic live from a host in the network to complete this portion.

After we analyzed the pcap traffic, the Security Manager has come back and confirmed the user was smuggling data out of the network via the images. He is requesting that we now capture traffic to determine if anything else is going on from the user's host **172.16.10.2.** We will need to start a capture, categorize and filter the data, and extract anything significant to the investigation.

Connectivity to Lab

Access to the lab environment to complete this part of the lab will be a bit different. We are using XfreeRDP to provide us desktop access to the lab virtual machine to utilize Wireshark from within the environment.

We will be connecting to the Academy lab like normal utilizing your own VM with a HTB Academy VPN key or the Pwnbox built into the module section. You can start the FreeRDP client on the Pwnbox by typing the following into your shell once the target spawns:

xfreerdp /v:<target IP> /u:htb-student /p:HTB_@cademy_stdnt!

You can find the target IP, Username, and Password needed below:

IP == 10.129.43.4 (ACADEMY-NTA-SNIFF01)

Username == htb-student

Password == HTB_@cademy_stdnt!

Start a Wireshark Capture

We will be sniffing traffic from the host we logged into from our own VM or Pwnbox. Utilizing interface ENS224 in Wireshark, let the capture run for a few minutes before stopping it. Our goal is to determine if anything is happening with the user's host and another machine on the corporate or external networks.

Self Analysis

Before following these tasks below, take the time to step through our pcap traffic unguided. Use the skills we have previously tested, such as following streams, analysis of conversations, and other skills to determine what is going on. Keep these questions in mind while performing analysis:

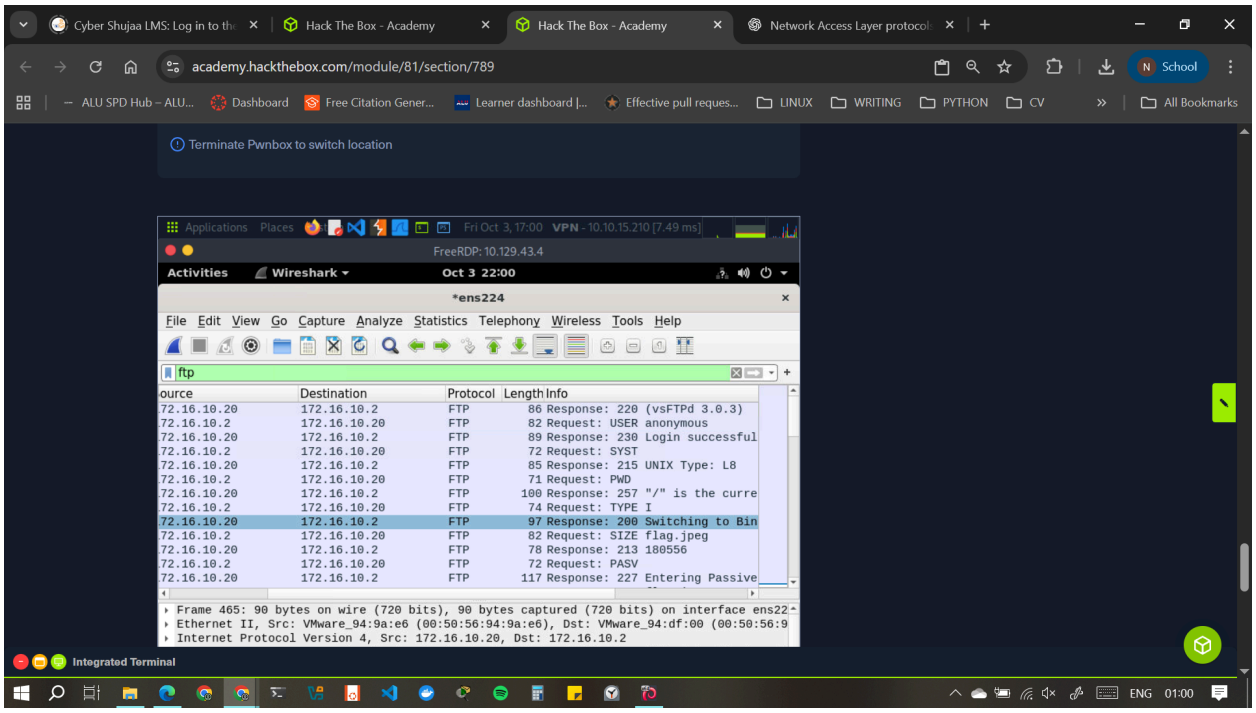
- How many conversations can be seen?
- Can we determine who the clients and servers are?
- What protocols are being utilized?
- Is anything of note happening? (ports being misused, clear text traffic or credentials, etc.)

In this lab, we are concerned with the hosts 172.16.10.2 and 172.16.10.20 while performing the following steps. In our analysis, we should have noticed some web traffic between these hosts and some FTP traffic. Let's dig a bit deeper.

FTP Analysis

When examining the traffic, we captured, was any traffic pertaining to FTP noticed? Who was the server for that traffic? **FTP traffic is noted. Server is 172.16.10.20**

- Identify any FTP traffic using the ftp display filter.



Were we able to determine if an authenticated user was performing these actions, or were they anonymous? **Anonymous**

- Look at the command controls sent between the server and hosts to determine if anything was transferred and who did so with the **ftp.request.command** filter.

Filter the results

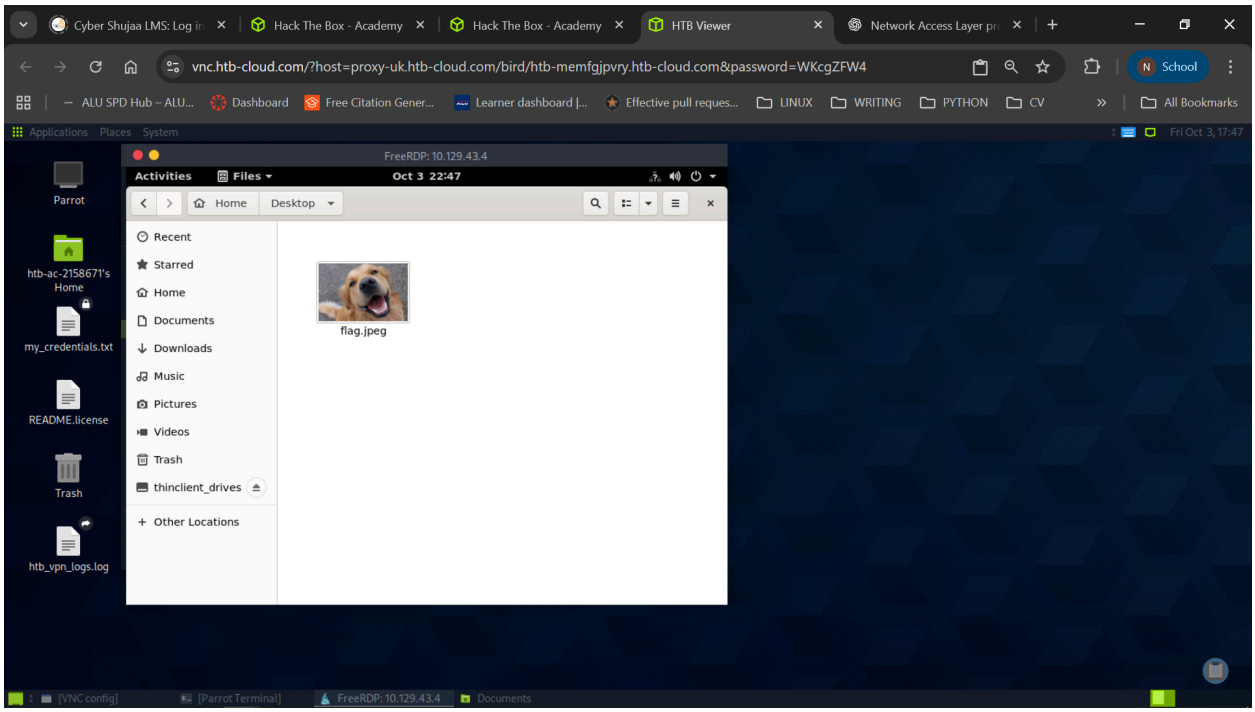
Now that we have seen some interesting traffic, let's try and grab the file off the wire.

Examine the FTP commands to determine what you need to inspect, and then extract the files from ftp-data and reassemble it

Choose a file, then filter for ftp-data. Select a packet that corresponds with our file of interest and follow the TCP stream that correlates to it.

Once done, Change "Show and save data as" to "Raw" and save the content as the original file name.

Validate the extraction by checking the file type.



HTTP Analysis

We should have seen a bit of HTTP traffic as well. Was this the case for you?

If so, could we determine who the webserver is?**172.16.10.20**

What application is running the webserver? **Apache webserver (version 2.4.41) running on Ubuntu.**

What were the most common method requests you saw? **GET and POST**

Follow the stream and extract the item(s) found

Now attempt to follow the HTTP stream and determine if there is anything to extract.

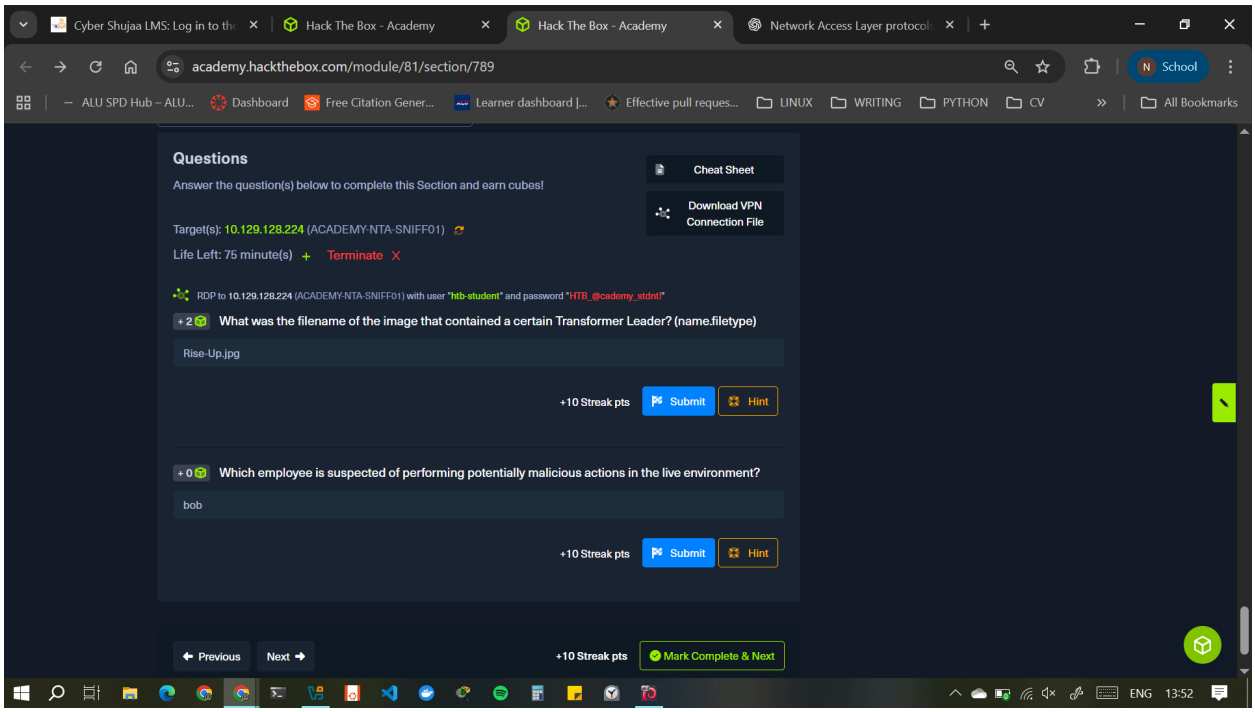
Apply the filter “http && image-jfif” to include only HTTP (80/TCP) packets along with a filter to include only JPEG File Interchange Formats (JPEG files).

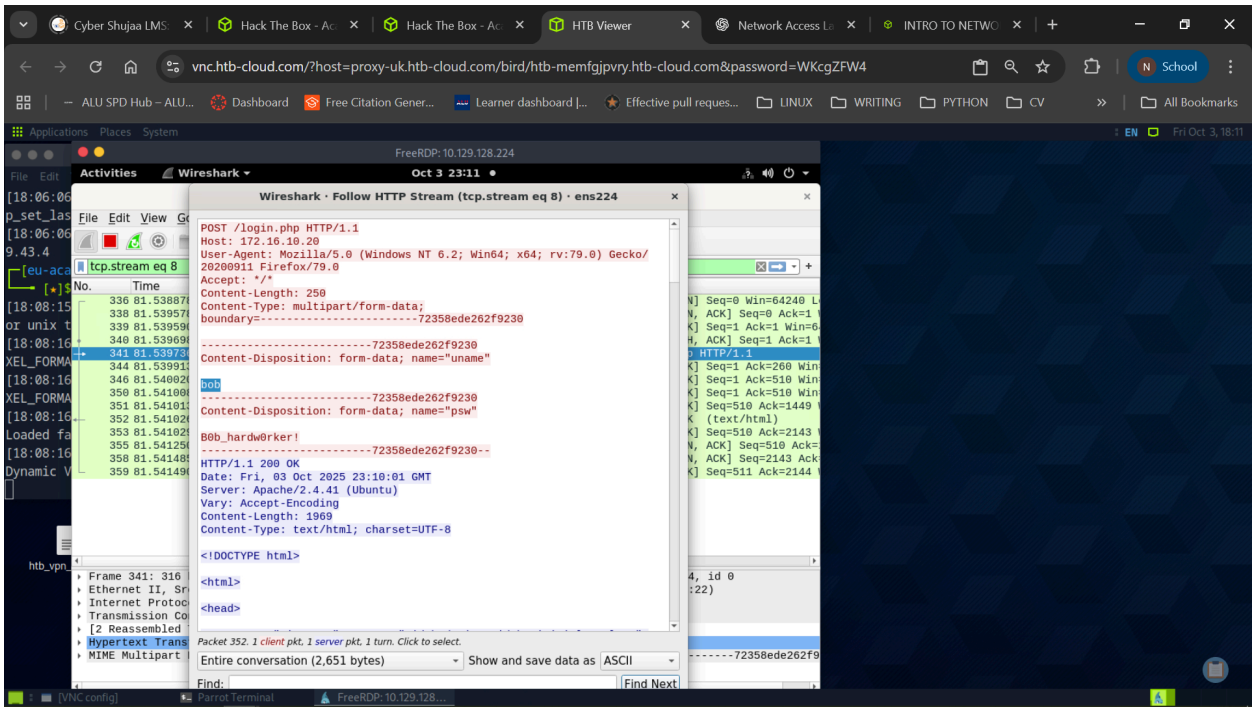
Look for the line in which the Web Server responds with a “200 OK” message which acts as an acknowledgment/receipt to a users’ GET request.

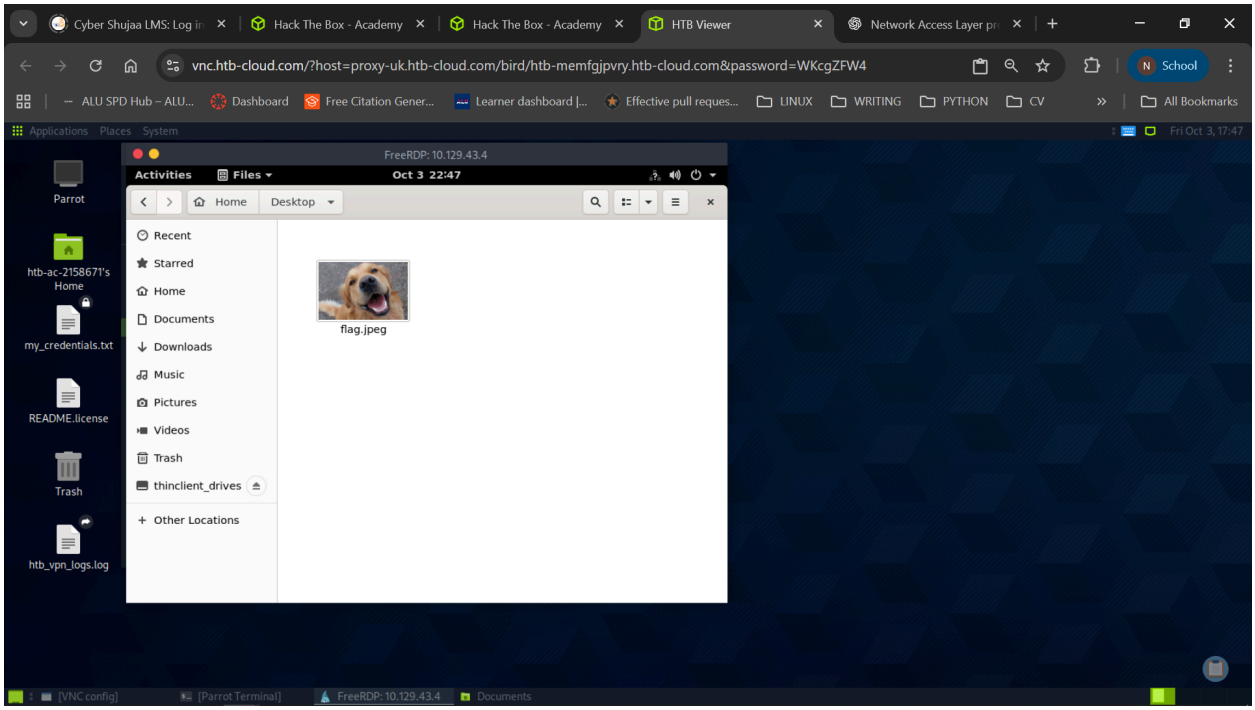
Select “File > Export Objects > HTTP > file.JPG

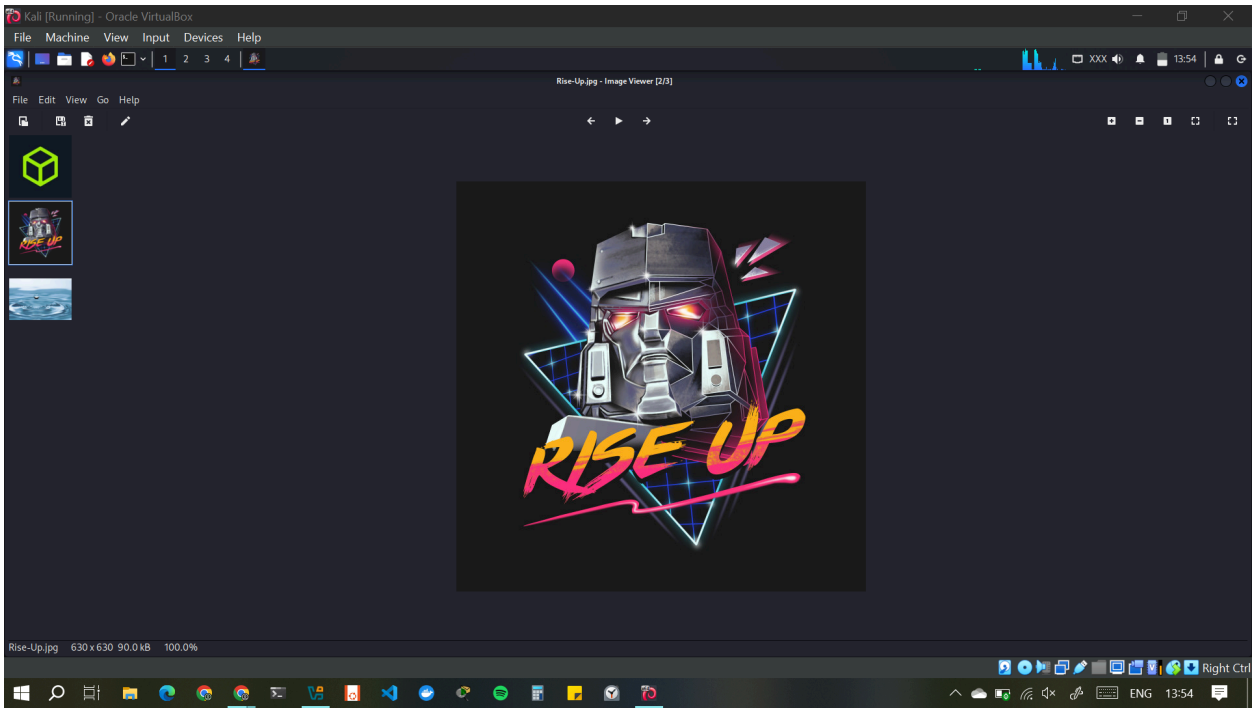
Conclusion

This lab demonstrates how to open previously captured .pcap files, apply display filters, follow streams, and extract items from the capture file as well as performing live capture and analysis.


Questions
Answer the question(s) below to complete this Section and earn cubes!
Cheat Sheet
Download VPN Connection File
Target(s): 10.129.128.224 (ACADEMY-NTA-SNIFF01)
Life Left: 75 minute(s) + Terminate X
RDP to 10.129.128.224 (ACADEMY-NTA-SNIFF01) with user "htb-student" and password "HTB_@cademy_stdnt!"
+2 What was the filename of the image that contained a certain Transformer Leader? (name.filetype)
Rise-Up.jpg
+10 Streak pts
Submit
Hint
+0 Which employee is suspected of performing potentially malicious actions in the live environment?
bob
+10 Streak pts
Submit
Hint
Previous
Next
+10 Streak pts
Mark Complete & Next


Wireshark · Follow HTTP Stream (tcp.stream eq 8) · ens224
POST /login.php HTTP/1.1
Host: 172.16.10.20
User-Agent: Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:79.0) Gecko/20200911 Firefox/79.0
Accept: */*
Content-Length: 250
Content-Type: multipart/form-data;
boundary=--------------------------72358ede262f9230
--------------------------72358ede262f9230
Content-Disposition: form-data; name="uname"
bob
--------------------------72358ede262f9230
Content-Disposition: form-data; name="psw"
B0b_hardw0rker!
--------------------------72358ede262f9230--
HTTP/1.1 200 OK
Date: Fri, 03 Oct 2025 23:10:01 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 1969
Content-Type: text/html; charset=UTF-8
<!DOCTYPE html>
<html>
<head>
Entire conversation (2,651 bytes)
Show and save data as ASCII
Find Next





Guided Lab: Traffic Analysis Workflow

Scenario:

One of our fellow admins noticed a weird connection from Bob's host IP = 172.16.10.90 when analyzing the baseline captures we have been gathering. He asked us to check it out and see what we think is happening.

Tasks

Task 1: Connect to the live host for capture.

Access to the lab environment to complete the following tasks will require the use of XfreeRDP to provide GUI access to the virtual machine so we can utilize Wireshark from within the environment.

We will be connecting to the Academy lab like normal utilizing your own VM with a HTB Academy VPN key or the Pwnbox built into the module section. You can start the FreeRDP client on the Pwnbox by typing the following into your shell once the target spawns:

Code: bash

xfreerdp /v:<target IP> /u:htb-student /p:HTB_@cademy_stdnt!

You can find the target IP, Username, and Password needed below:

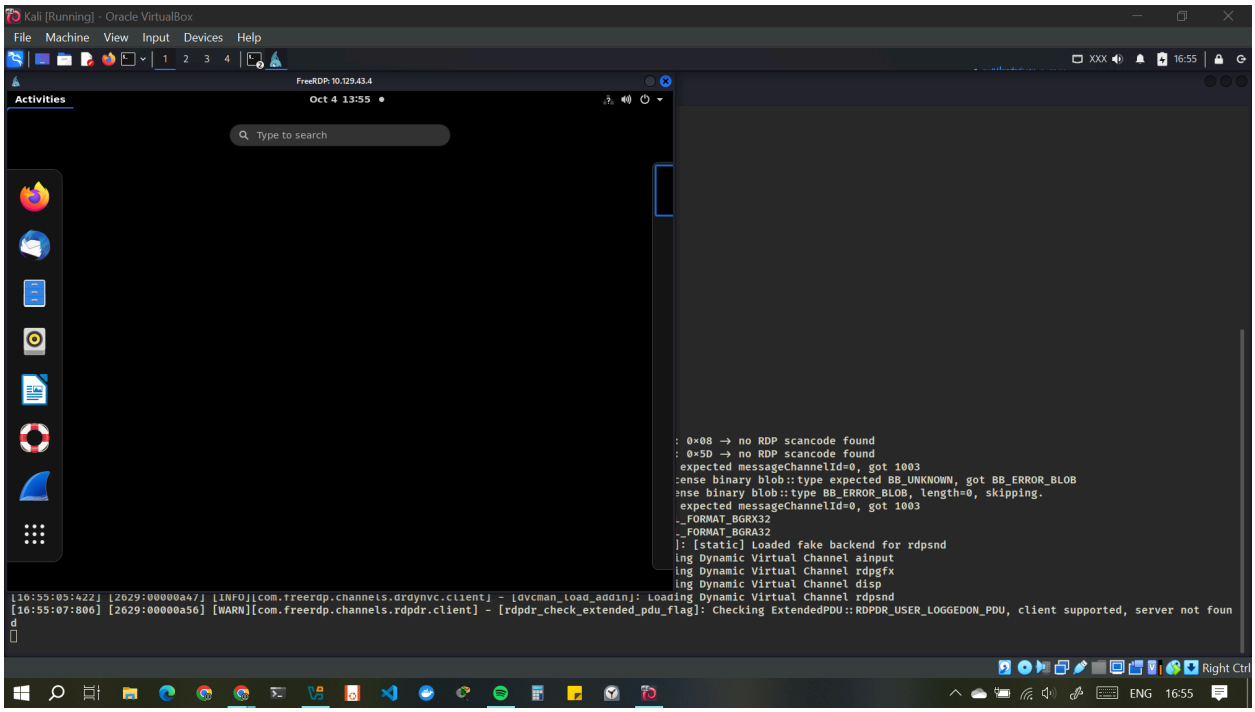
Click below in the Questions section to spawn the target host and obtain an IP address.

IP == 10.129.43.4 (ACADEMY-NTA-SNIFF01)

Username == htb-student

Password == HTB_@cademy_stdnt!

Once connected, open Wireshark and begin capturing on interface ENS224.



Analysis

Follow this workflow template and examine the suspicious traffic. The goal is to determine what is happening with the host in question.

1. what is the issue?

   Suspicious traffic coming from within the network

2. define our scope and the goal (what are we looking for? which time period?)

   scope: **10.129.43.4 and anyone with a connection to it. Protocol in use by port 4444 is unknown**

   when the issue started: **within the last 48 hours. Capture traffic to determine if it is still happening.**

   supporting info: **guided-analysis.pcap**

3. define our target(s) (net / host(s) / protocol)

   Target hosts: **traffic originating from 10.129.43.4 where port 4444 is being used**

   Protocol: SSDP, ARP, NAT-PMP(UDP)

4. capture network traffic / read from previously captured PCAP.

Perform actions as needed to analyze the traffic for signs of intrusion.

5. identification of required network traffic components (filtering)

   once we have our traffic, filter out any traffic not necessary for this investigation to include; any traffic that matches our common baseline, and keep anything relevant to the scope of the investigation.

6. An understanding of captured network traffic

   Once we have filtered out the noise, it's time to dig for our targets. Start broad and close the circle around our scope.

7. note taking / mind mapping of the found results.

   Annotating everything we do, see, or find throughout the investigation is crucial. Ensure you are taking ample notes, including:

   Timeframes we captured traffic during.

   Suspicious hosts/ports within the network.

   Conversations containing anything suspicious. (to include timestamps, and packet numbers, files, etc.)

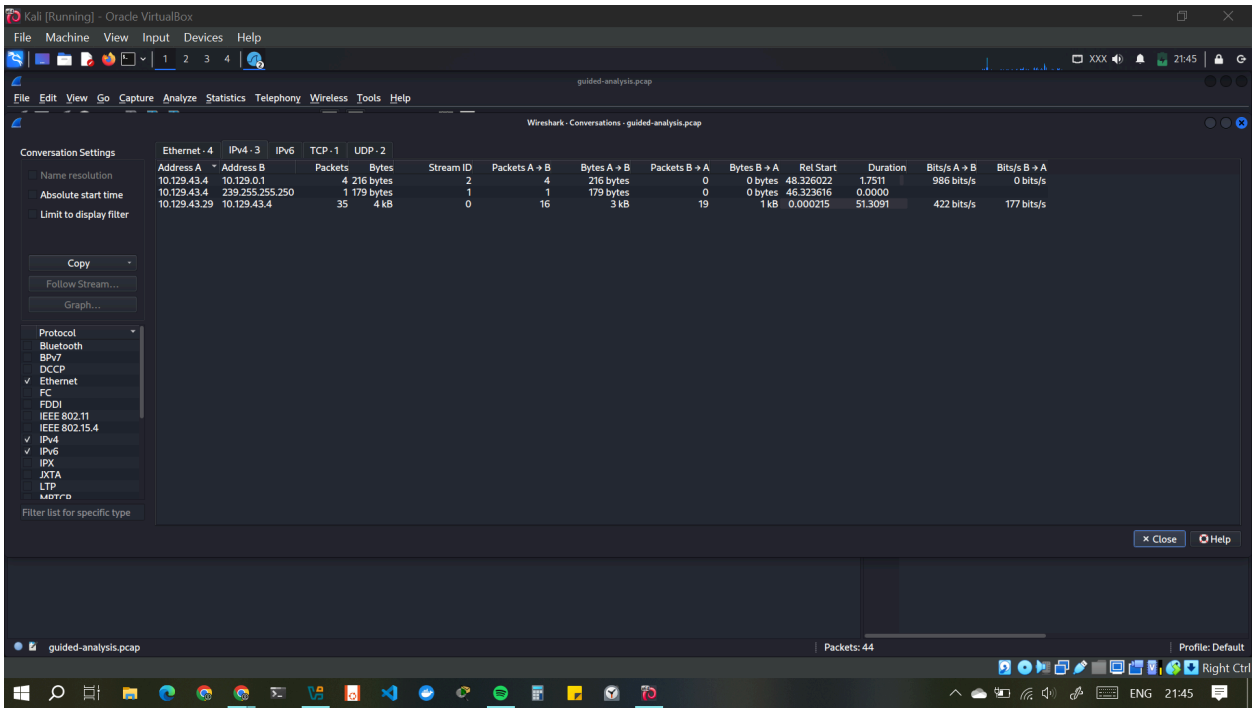
8. summary of the analysis (what did we find?)

   Finally, summarize what has been found, explaining the relevant details so that superiors can decide to quarantine the affected hosts or perform a more critical incident response mission.

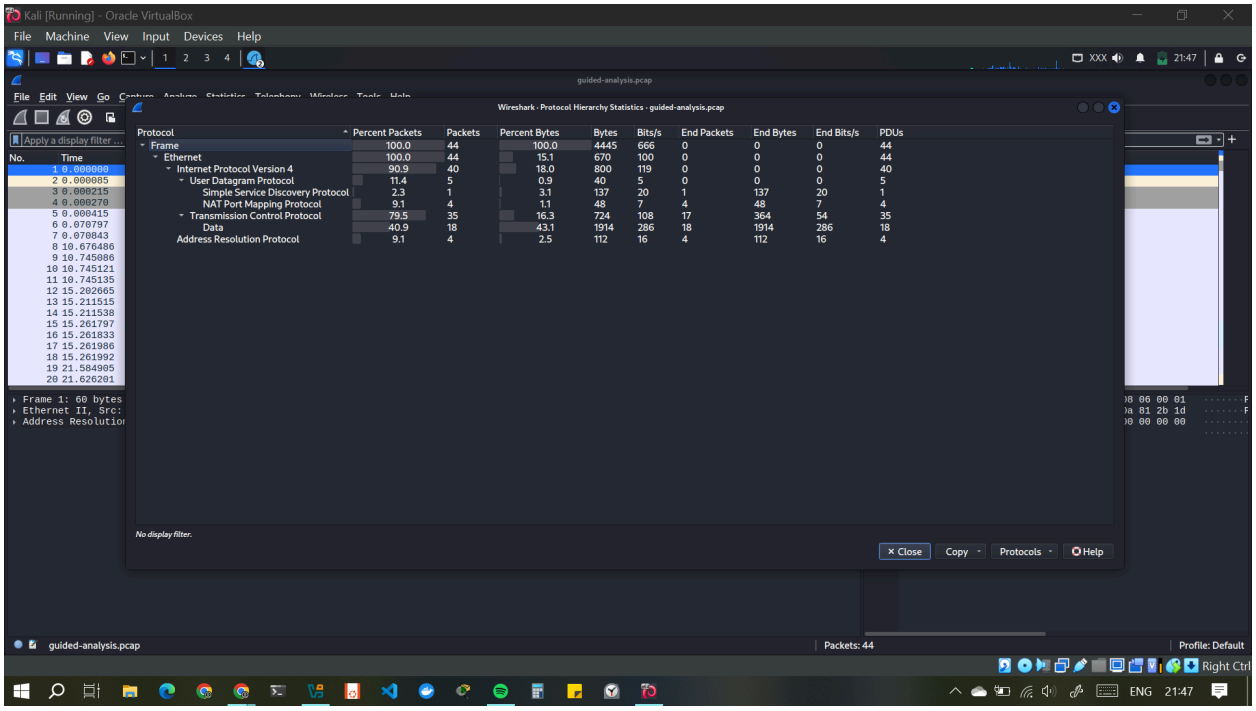
   Our analysis will affect decisions made, so it is essential to be as clear and concise as possible.

Conversations

3 conversations can be seen from our pcap file.



Protocol characteristics

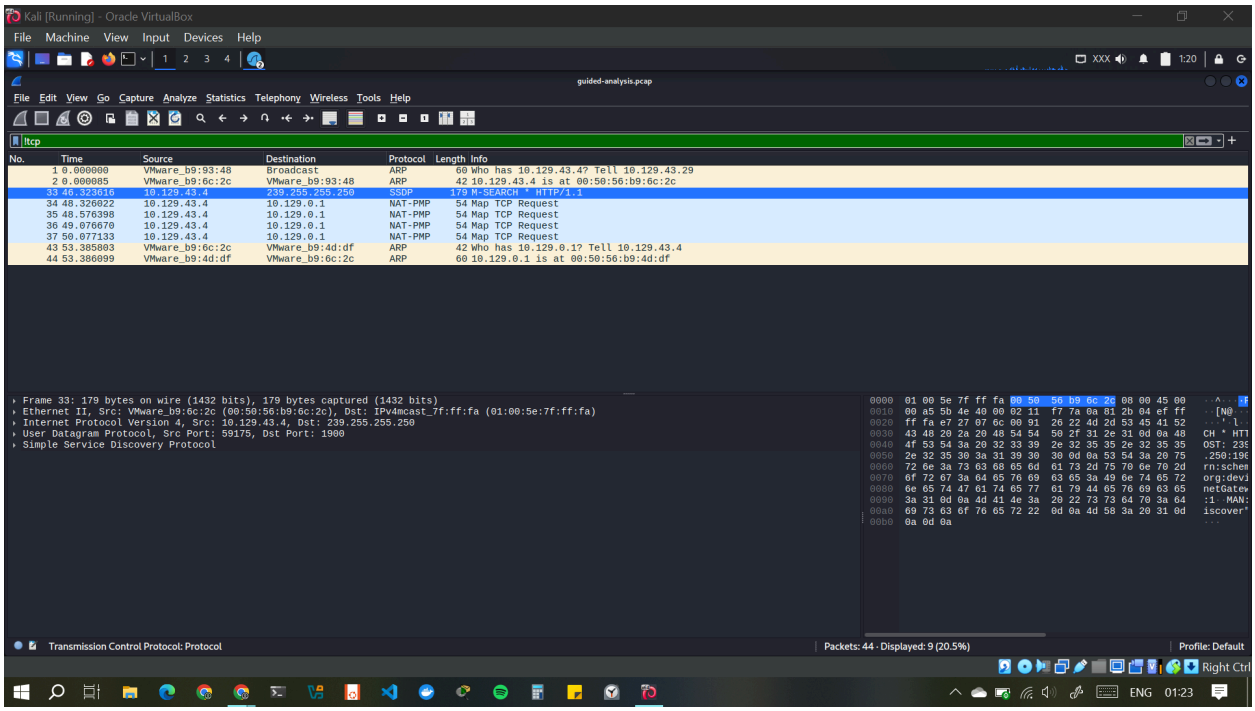


We can see here that this PCAP is mostly TCP traffic, with a bit of UDP traffic. Since there is less UDP than TCP traffic, let us look into that first.

Once we have filtered out the noise, it's time to dig for anything unusual. We are going to filter out everything but UDP traffic first.

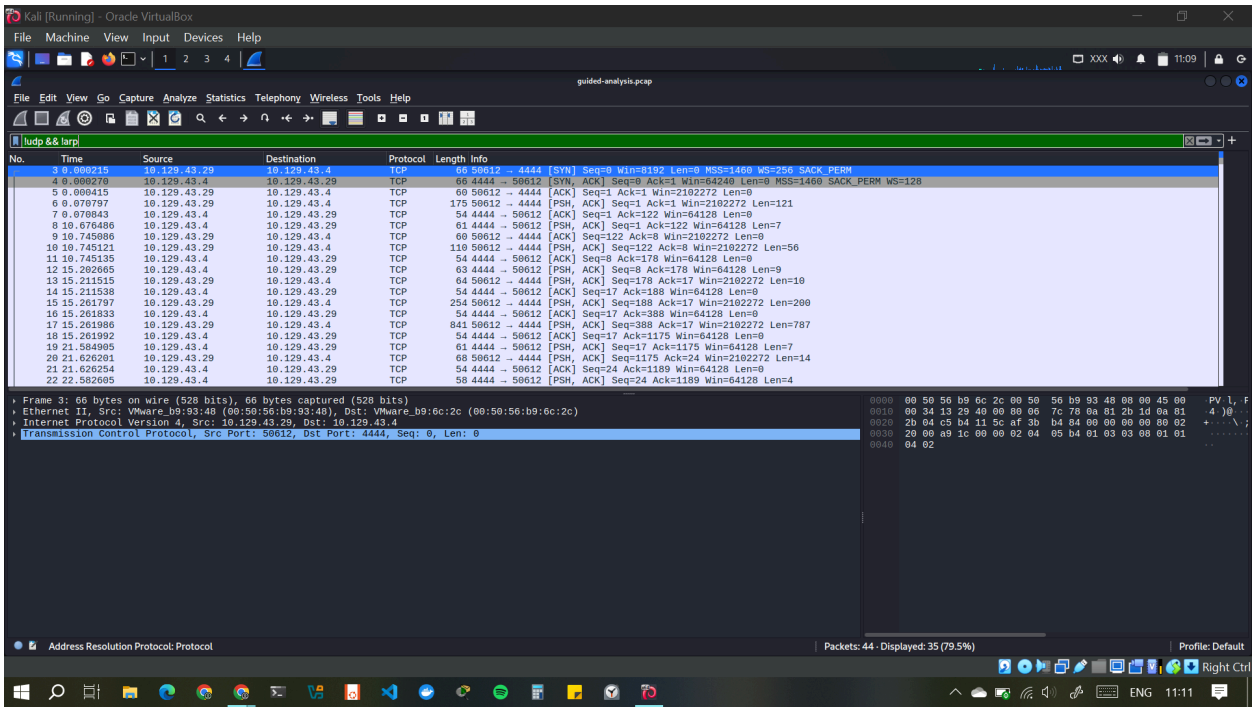
UDP

When filtering on just UDP traffic, we only see nine packets. Four arp packets, four Network Address Translation NAT, and one Simple Service Discovery Protocol SSDP packet. We can determine based on their packet types and information they contain that this traffic is normal network traffic and nothing to be concerned about.



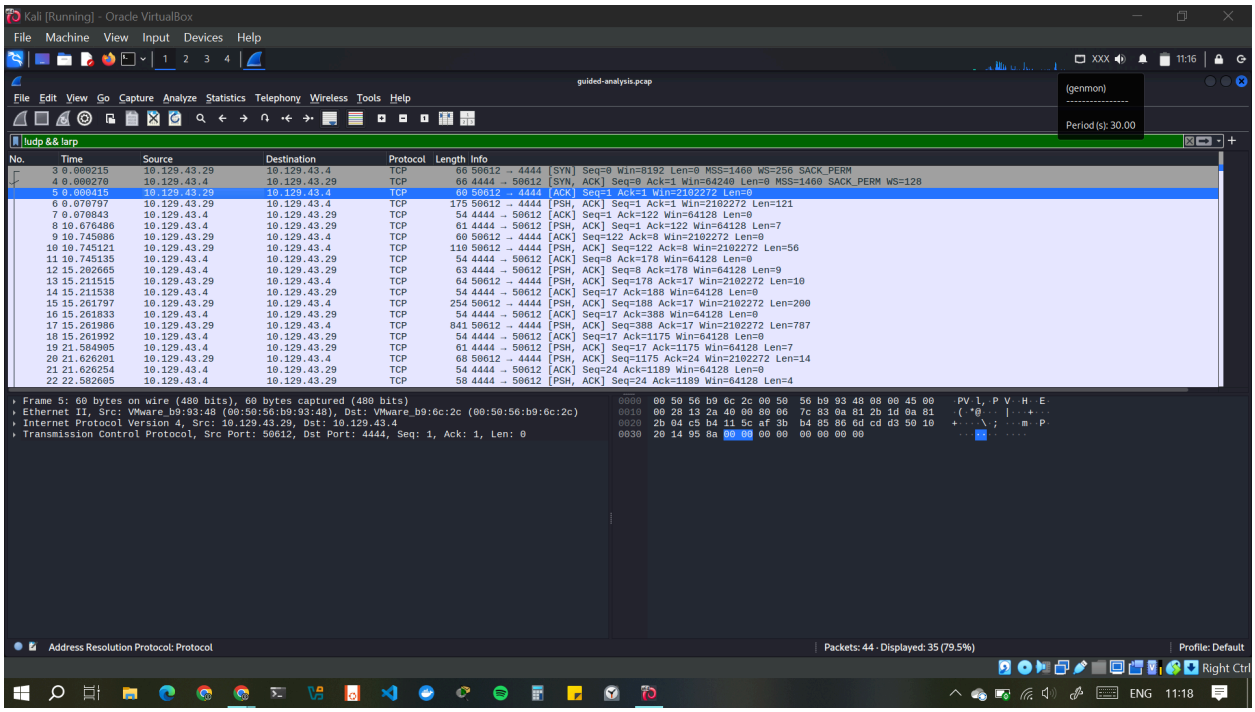
TCP

Now, let's move on to looking at TCP traffic. We should have quite a bit more here to sift through. We are going to utilize the display filter **!udp && !arp**. This filter will clear out anything we have already analyzed.



TCP Session establishment

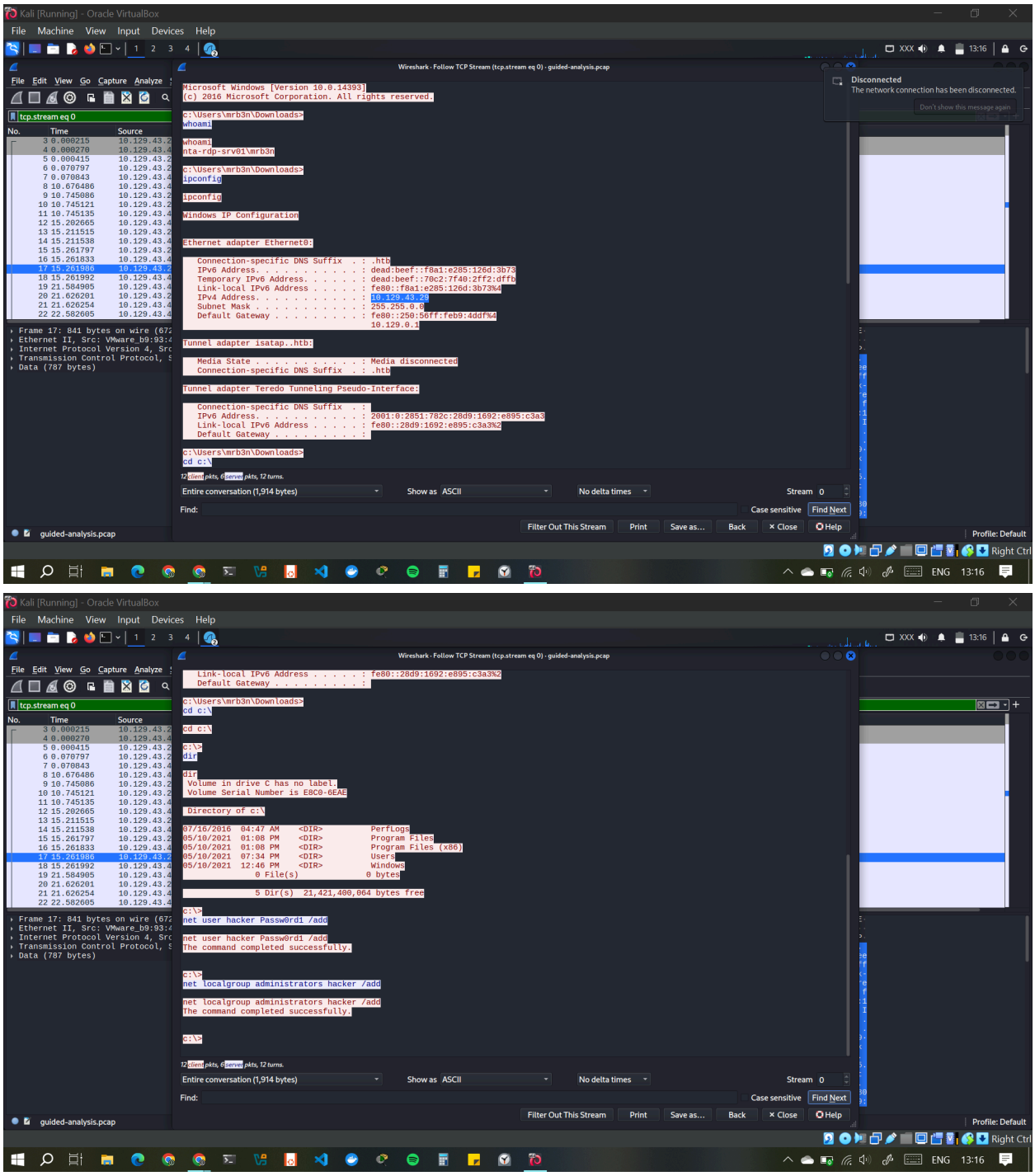
Now that we have cleared our view a bit, we can see the remaining packets are all TCP, and all appear to be the same conversation between hosts **10.129.43.4 and 10.129.43.29**. We can determine this since we can see the session establishment via a three-way handshake at packet 3, and the same ports are used through the rest of the packets in the output below.



What does appear interesting is that we do not see a **TCP session teardown** in this PCAP file. This could mean the session was still active and not terminated. We believe this to be true since we do not see any Reset packets either.

We can also examine the conversation by following the TCP stream from packet 3 to determine what it encompasses.

Follow TCP Stream



Looking at the image above, it appears that someone is performing basic recon of the host. They are issuing commands like **whoami, ipconfig, dir**. It would appear they are trying to get a lay of the land and figure out what user they landed as on the host.

Something more alarming we can now see is someone made the account **hacker** and assigned it to the administrators group on this host. Either this is a joke by a poor administrator. Or someone has infiltrated the corporate infrastructure.

Summary of the Analysis

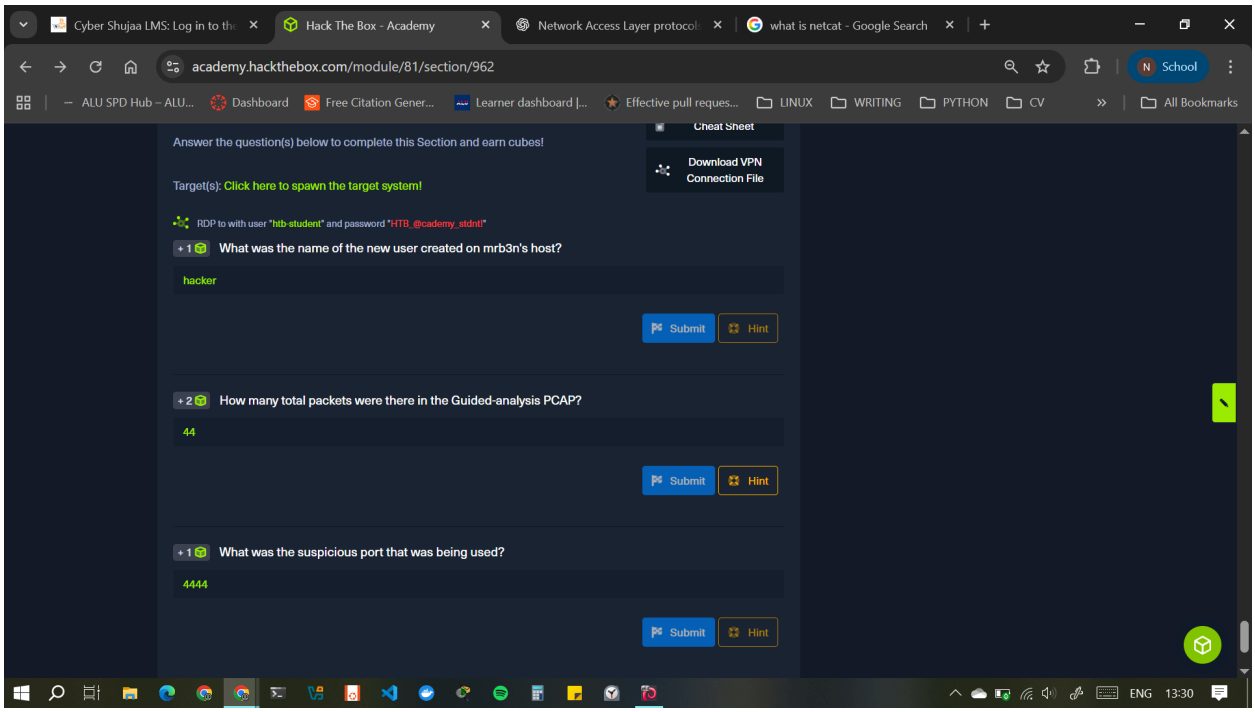
Based on our analysis, we determined that a malicious actor has infiltrated at least one host on the network. **Host 10.129.43.29** shows signs of someone executing commands to include user creation and assigning local administrator permissions via the **net** commands. It would look like the actor was using **Bob's** host to perform said actions. Since Bob was previously under investigation for the exfil of corporate secrets and disguising it as web traffic, I think it is safe to say the issue has spread further. The screenshots included with this document show the flow of traffic and commands utilized.

It is our opinion that a complete Incident Response IR procedure be enacted to ensure the threat is stopped from spreading further. We can dedicate resources to clearing the malicious presence and cleaning the affected hosts.

After analyzing the actions taken, the IR team determined that The actor got lazy and decided to utilize a Netcat shell and directly interact with Bob's host while gathering more information.

- **Netcat** is a versatile command-line networking utility known as the "Swiss army knife for TCP/IP" that allows users to read and write data across networks using TCP or UDP protocols. It can operate in client or server mode, facilitating network debugging, port scanning, file transfers, creating simple chat servers, and establishing **remote shells** by establishing simple connections between machines for data exchange.

While doing so, he used **RDP(nta-rdp-srv01\mrb3n)** from Bob's host to another windows desktop in the environment to try and establish another foothold. Luckily, the IR team was able to capture some PCAP of the RDP traffic. Bob's host was quarantined, and incident response was initiated to determine what was taken and what other potential hosts were compromised.



Decrypting RDP connections

The purpose of this lab is to utilise Wireshark's power in working with RDP traffic. RDP(Remote Desktop Connection) is a **Microsoft protocol** that enables a user to connect to another computer over a network, providing a graphical interface to control it as if they were sitting in front of it. It functions by transmitting the remote computer's display data to the user's client device and sending the user's mouse and keyboard input back to the remote machine, all over an encrypted network connection. RDP is widely used by IT professionals and is a fundamental part of managing remote computers and servers and allows employees to access and work on their office computers from home or other locations. RDP typically listens for connections on port **3389**. If one has the required key utilized between the two hosts for encrypting the traffic, Wireshark can deobfuscate the traffic for us.

When performing IR and analysis on Bob's machine, the IR team captured some PCAP of the RDP traffic they noticed from Bob's host to another host in the network. We have been asked to investigate the occurrence by our team lead. While combing his host for further evidence, you found an **RDP-key** hidden in a folder hive on Bob's host. After some research, we realize that we can utilize that key to decrypt the RDP traffic to inspect it.

Requirements

1. Rdp.pcapng file
2. Server.key file(RDP Key)

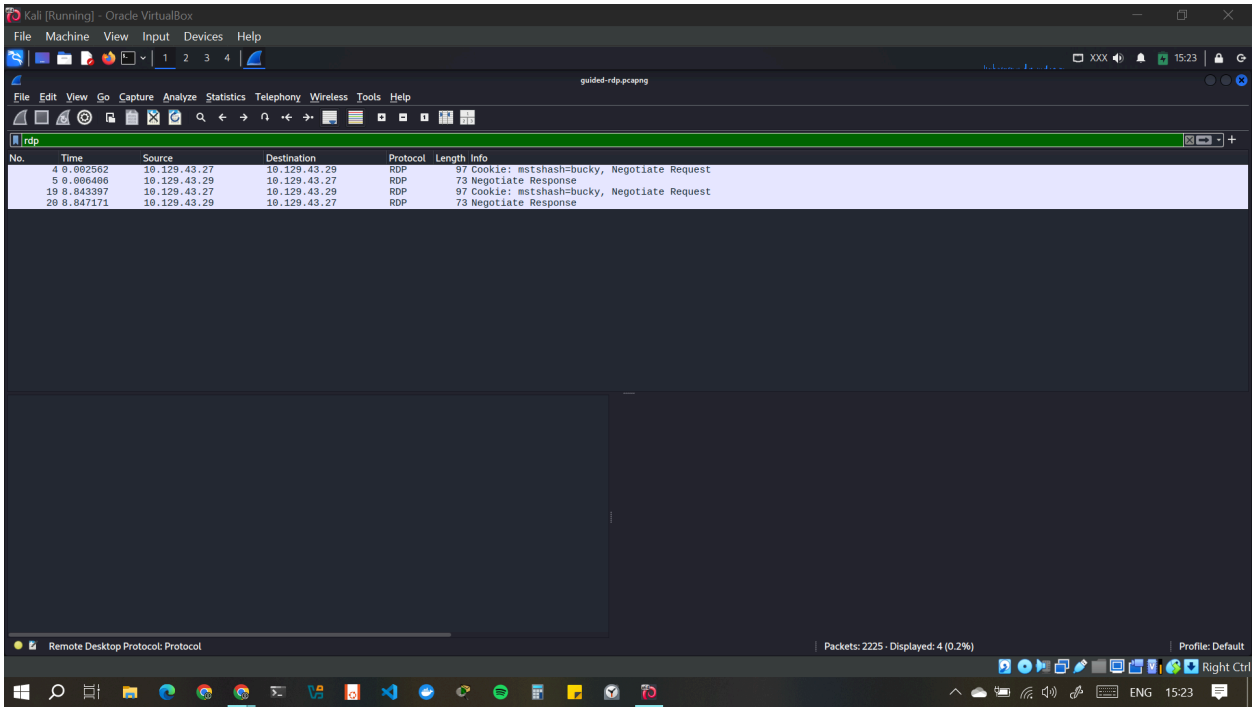
Tasks

Task 1: Open the rdp.pcapng file in Wireshark.

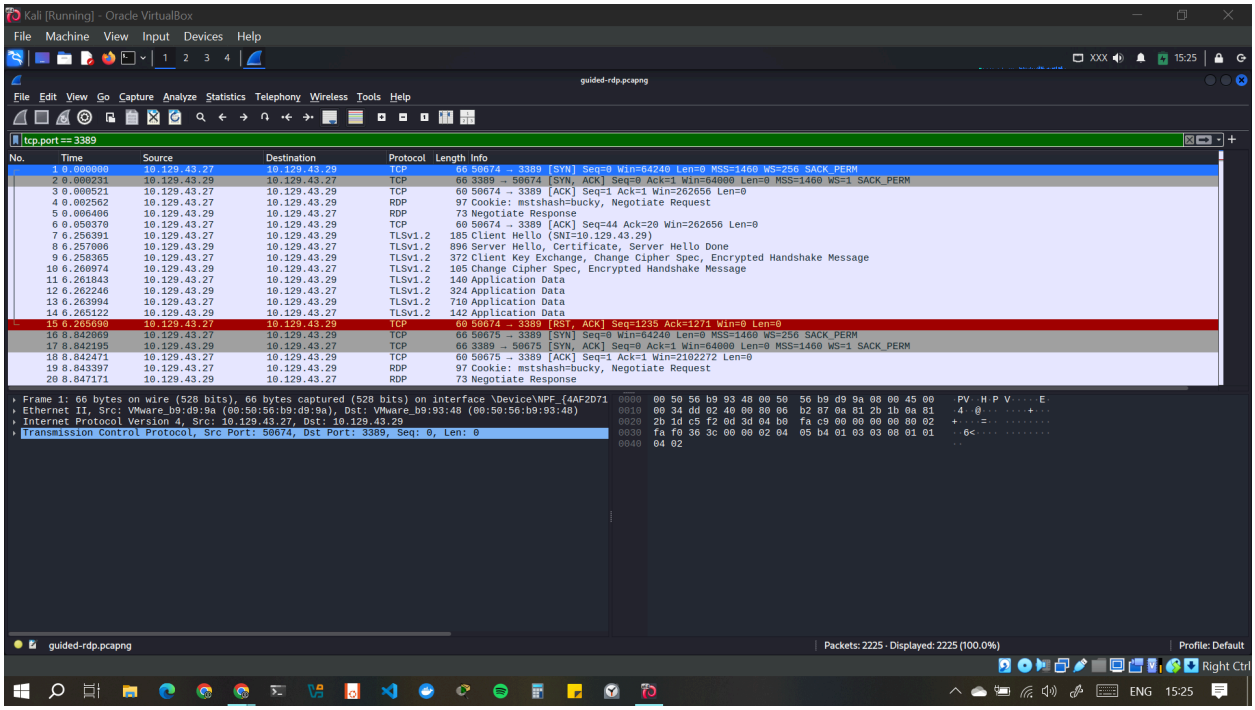
Unzip the zip file included in the optional resources and open it in Wireshark.

Task #2: Analyze the traffic included.

Take a minute to look at the traffic. Notice there is a lot of information here. We know our focus is on RDP, so let's take a second to filter on rdp and see what it returns.



RDP, by default, is utilizing **TLS** to encrypt the data, so we may not be able to see anything that happened with RDP traffic. How can we verify its existence in this file? One way is to filter on the well-known port RDP uses typically(3389) by using filter tcp.port == 3389.

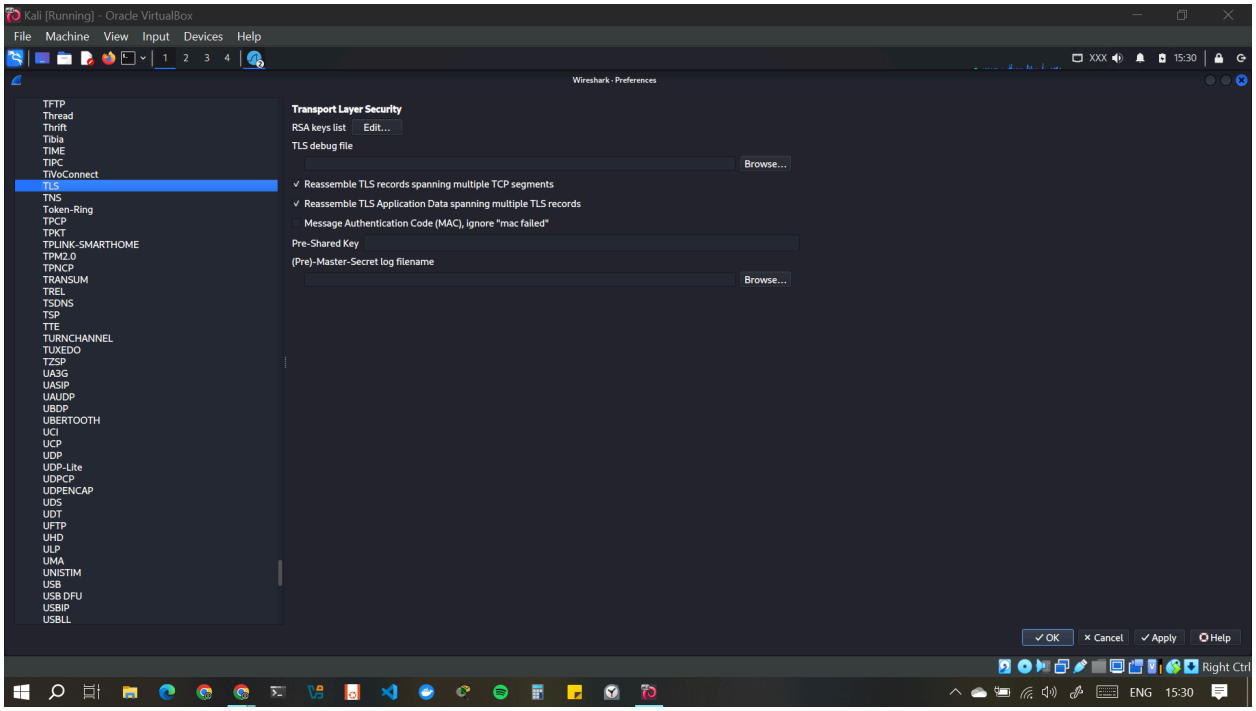


We can at least verify that a session was established between the two hosts over TCP port 3389.(the TCP 3-way handshake is complete)

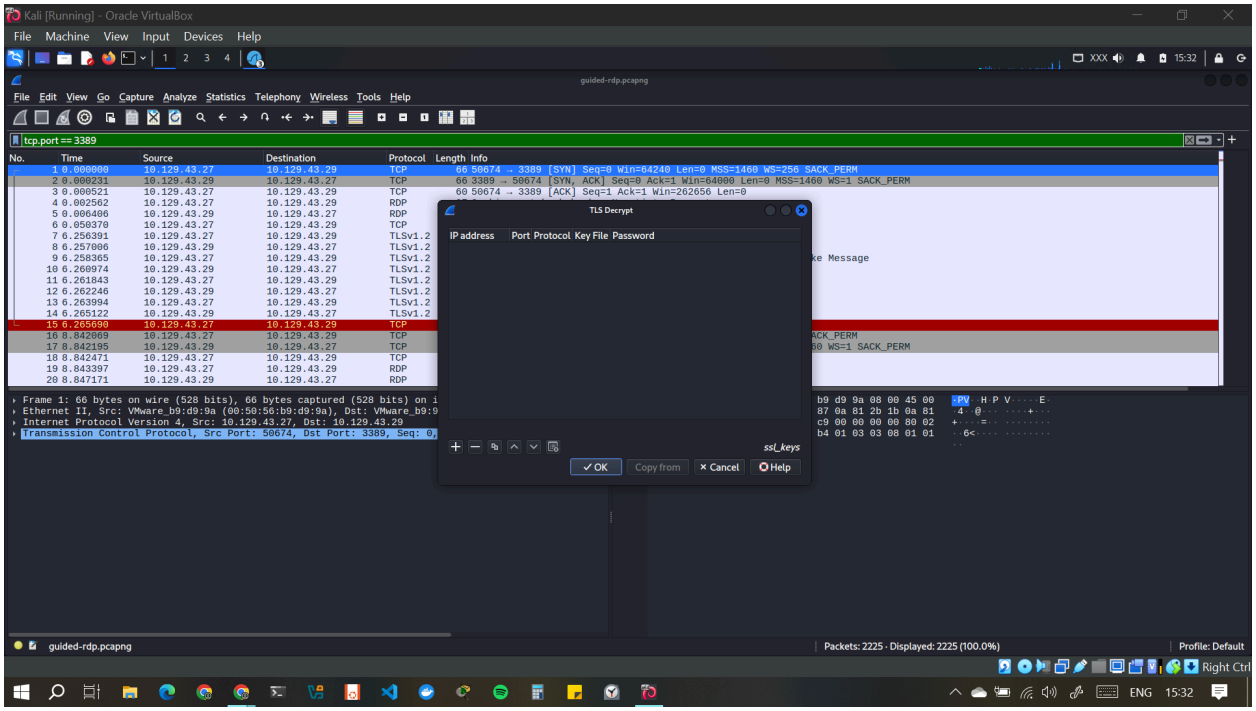
Task 3: Provide the RDP-key to Wireshark so it can decrypt the traffic.

We proceeded to use the key we found to try and decrypt the traffic. To apply the key in Wireshark:

1. go to Edit → Preferences → Protocols → TLS



2. On the TLS page, select Edit by RSA keys list → a new window will open.

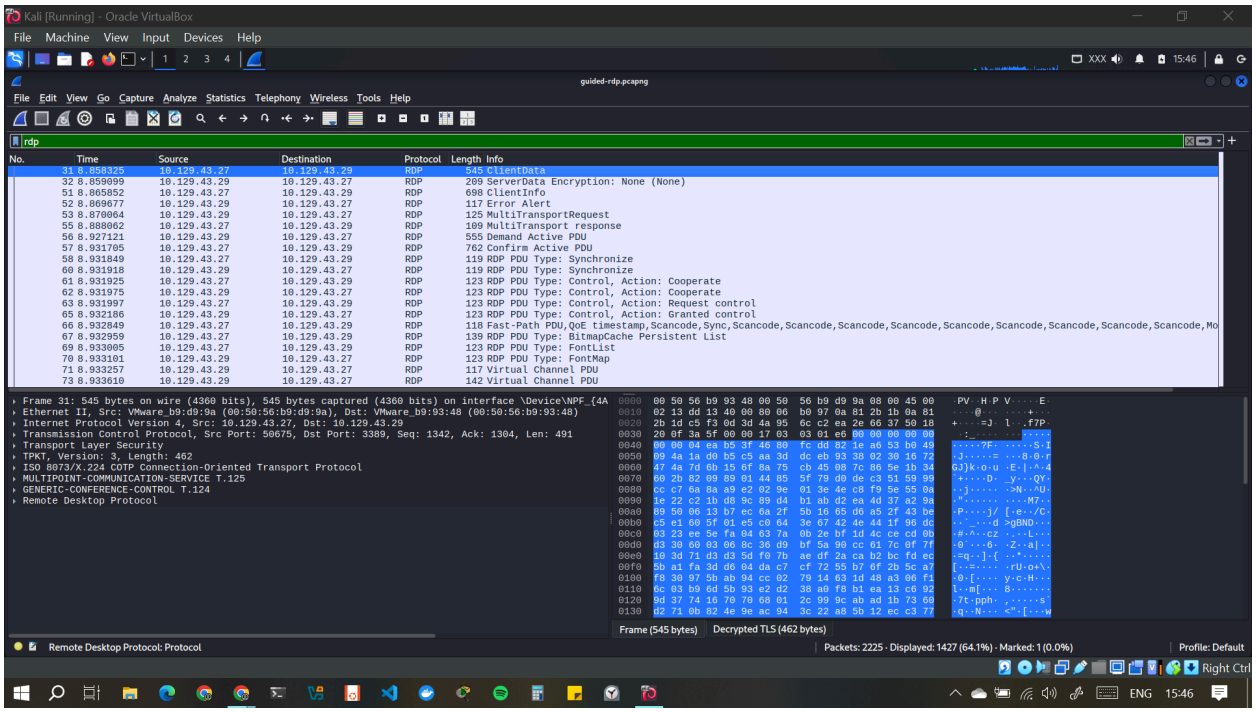


3. Follow the steps below to import the RSA server key.

   Steps

   - Click the + to add a new key
   - Type in the IP address of the RDP server **10.129.43.29**
   - Type in the port used **3389**
   - Protocol filed equals **tpkt** or blank.
   - Browse to the **server.key** file and add it in the key file section.
   - Save and refresh your pcap file.

When filtering once again on RDP, we should see some traffic in the display.



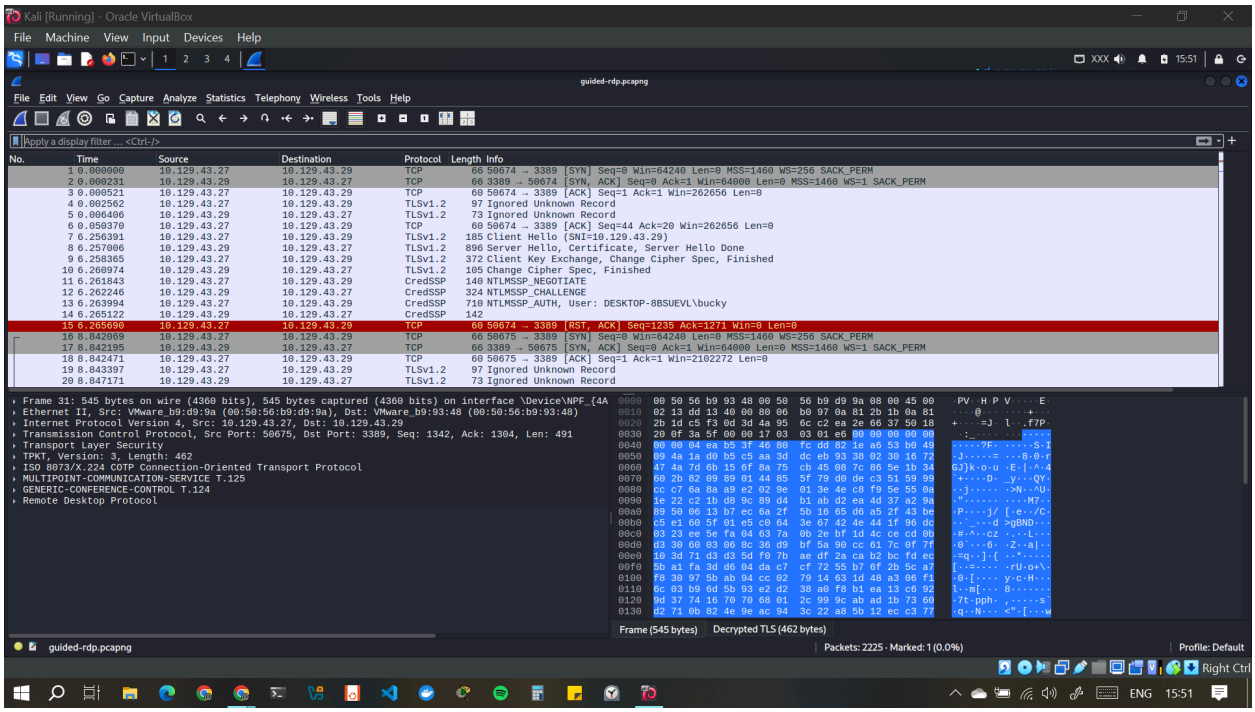
From here, we can perform an analysis of the RDP traffic. We can now follow TCP streams, export any potential objects found, and anything else we feel necessary for our investigation. This works because we acquired the RSA key used for encrypting the RDP session. The steps for acquiring the key were a bit lengthy, but the short of it is that if the RDP certificate is acquired from the server, OpenSSL can pull the private key out of it.

Perform Analysis of the Unencrypted Traffic

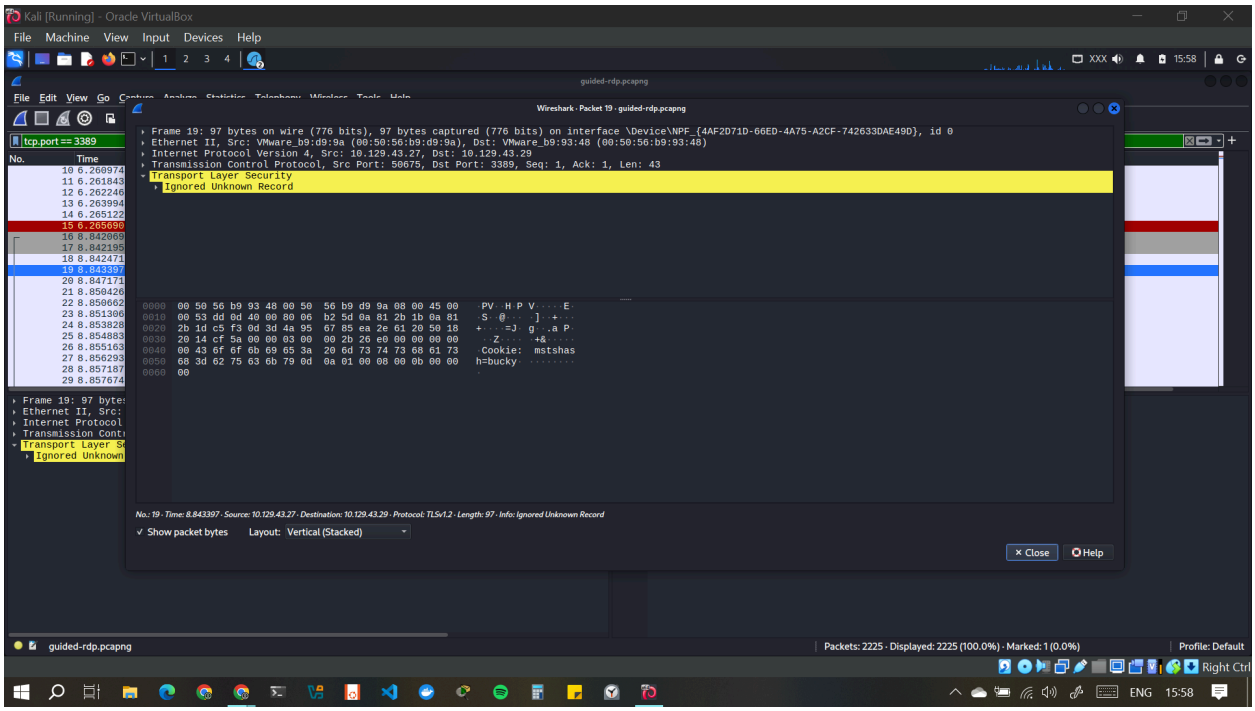
Now that we have broken RDP out of the TLS tunnel, what can we find? Perform the analysis steps and attempt to answer the questions below.

Questions:

1. What host initiated the RDP session with our server? The 1st full 3 way handshake has **10.129.43.27** initiating the connection.

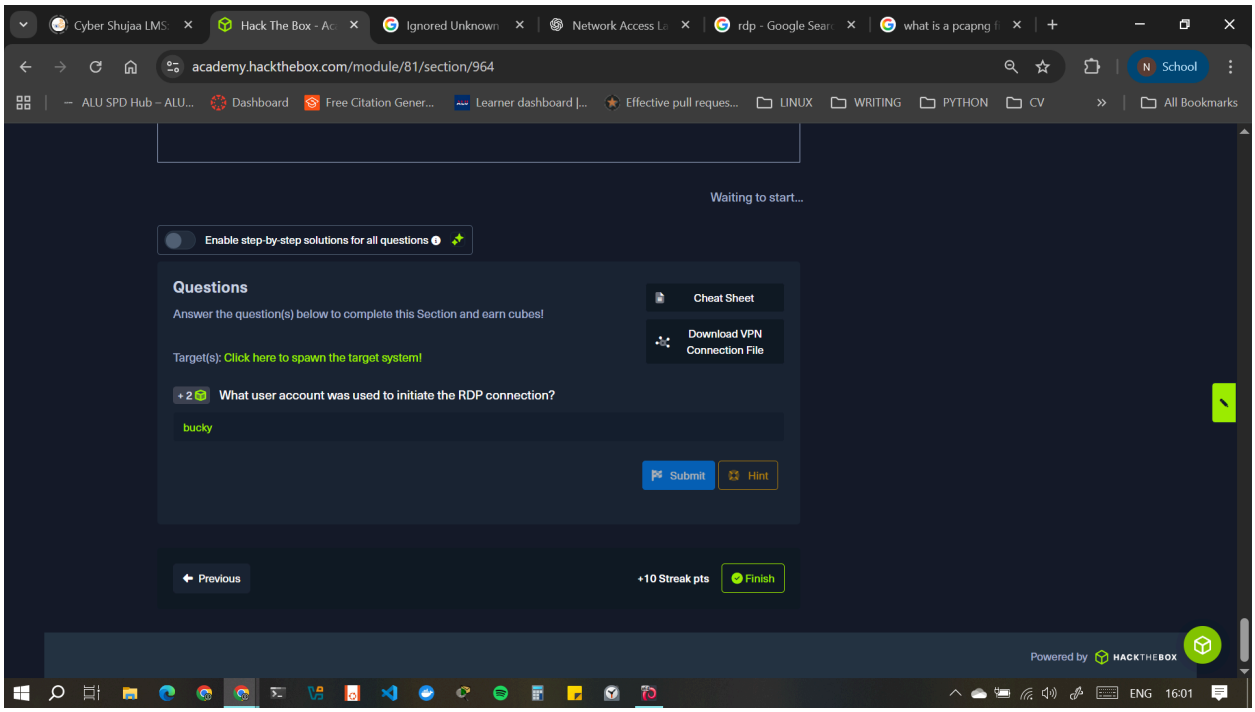


2. Which user account was used to initiate the RDP connection? **Using filter tcp.port == 3389, we can see a record labeled <u>Ignored Unknown Record</u>. On examining the ASCII, it shows us a username, which is 'bucky'.**
    - **Ignored Unknown Record is a message from Wireshark indicating it encountered TLS data it couldn't properly interpret, often due to packet loss, out-of-order TCP segments, or incorrect MTU (Maximum Transmission Unit) settings.**



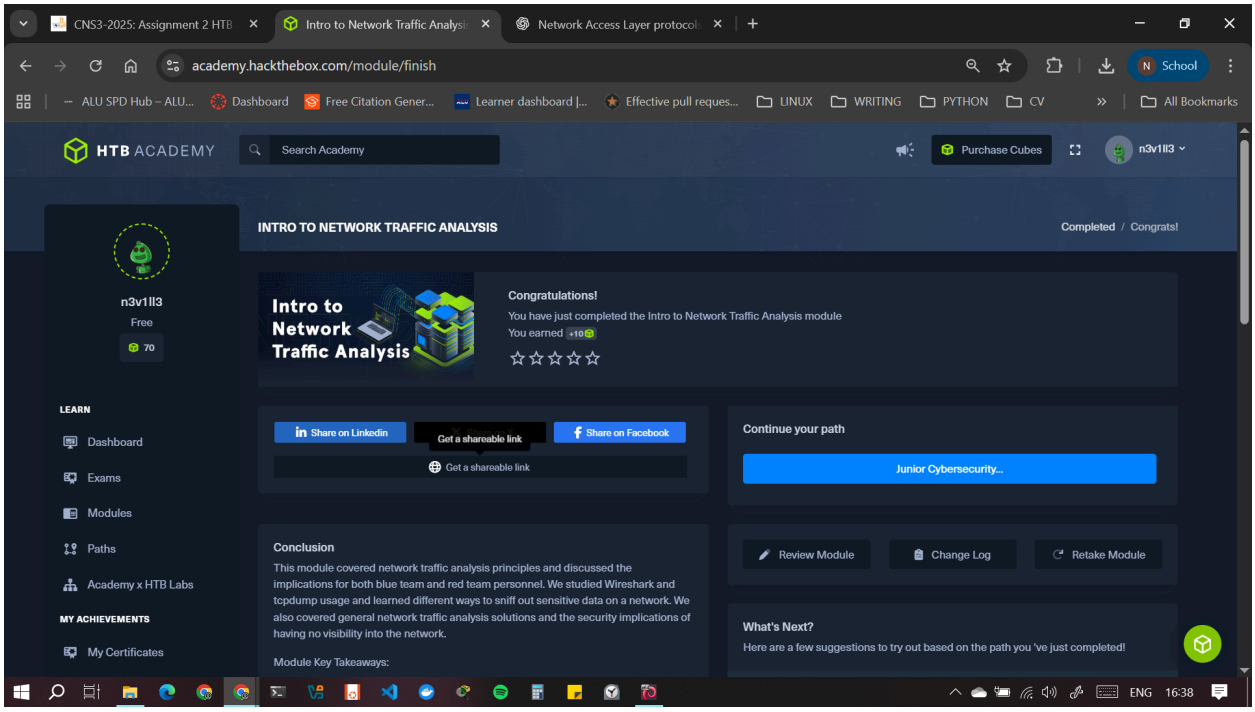
Summary:

This lab was to serve as an example of what Wireshark can do with captured data and its plugins. Wireshark's capability to ingest information and illuminate the obscure is robust. Having the ability to decrypt data after ingestion is a powerful capability. This concept could be applied to any protocol that utilizes encryption as long as we have the key that will be utilized to establish the connections.



Module completion

https://academy.hackthebox.com/achievement/2158671/81



Conclusion

This module covered network traffic analysis principles and discussed the implications for both blue team and red team personnel. We studied Wireshark and tcpdump usage and learned different ways to sniff out sensitive data on a network. We also covered general network traffic analysis solutions and the security implications of having no visibility into the network.

Module Key Takeaways:

- The importance of network traffic analysis for red and blue teams
- Usage of both Wireshark and tcpdump
- Methods for sniffing out sensitive data on the network

This assignment not only strengthened my technical skills in packet analysis but also enhanced my investigative thinking — the ability to reconstruct network events and identify anomalies from raw traffic.