

# Wireshark: Traffic Analysis

## 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.

- TCP connect scans
- SYN scans
- UDP scans

It is essential to know how Nmap scans work to spot scan activity on the network. However, it is impossible to understand the scan details without using the correct filters. Below are the base filters to probe Nmap scan behaviour on the network.

TCP flags in a nutshell.

Notes	Wireshark Filters
Global search.	<ul style="list-style-type: none"><li>• <code>tcp</code></li><li>• <code>udp</code></li></ul>
<ul style="list-style-type: none"><li>• Only SYN flag.</li><li>• SYN flag is set. The rest of the bits are not important.</li></ul>	<ul style="list-style-type: none"><li>• <code>tcp.flags == 2</code></li><li>• <code>tcp.flags.syn == 1</code></li></ul>
<ul style="list-style-type: none"><li>• Only ACK flag.</li><li>• ACK flag is set. The rest of the bits are not important.</li></ul>	<ul style="list-style-type: none"><li>• <code>tcp.flags == 16</code></li><li>• <code>tcp.flags.ack == 1</code></li></ul>

<ul style="list-style-type: none"> <li>Only SYN, ACK flags.</li> <li>SYN and ACK are set. The rest of the bits are not important.</li> </ul>	<ul style="list-style-type: none"> <li><code>tcp.flags == 18</code></li> <li><code>(tcp.flags.syn == 1) and (tcp.flags.ack == 1)</code></li> </ul>
<ul style="list-style-type: none"> <li>Only RST flag.</li> <li>RST flag is set. The rest of the bits are not important.</li> </ul>	<ul style="list-style-type: none"> <li><code>tcp.flags == 4</code></li> <li><code>tcp.flags.reset == 1</code></li> </ul>
<ul style="list-style-type: none"> <li>Only RST, ACK flags.</li> <li>RST and ACK are set. The rest of the bits are not important.</li> </ul>	<ul style="list-style-type: none"> <li><code>tcp.flags == 20</code></li> <li><code>(tcp.flags.reset == 1) and (tcp.flags.ack == 1)</code></li> </ul>
<ul style="list-style-type: none"> <li>Only FIN flag</li> <li>FIN flag is set. The rest of the bits are not important.</li> </ul>	<ul style="list-style-type: none"> <li><code>tcp.flags == 1</code></li> <li><code>tcp.flags.fin == 1</code></li> </ul>

# TCP Connect Scans

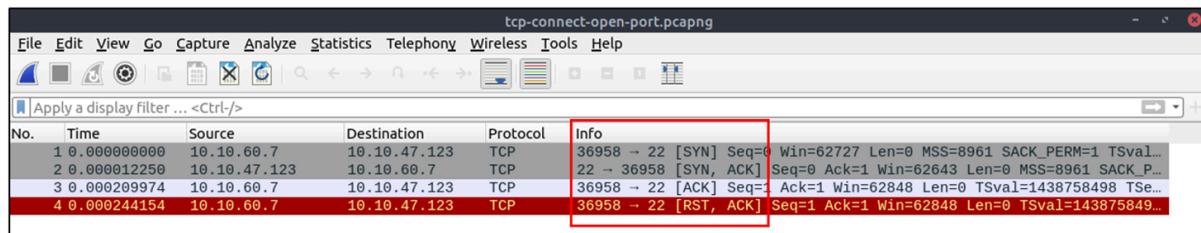
TCP Connect Scan in a nutshell:

- Relies on the three-way handshake (needs to finish the handshake process).
- Usually conducted with `nmap -sT` command.
- Used by non-privileged users (only option for a non-root user).
- Usually has a windows size larger than 1024 bytes as the request expects some data due to the nature of the protocol.

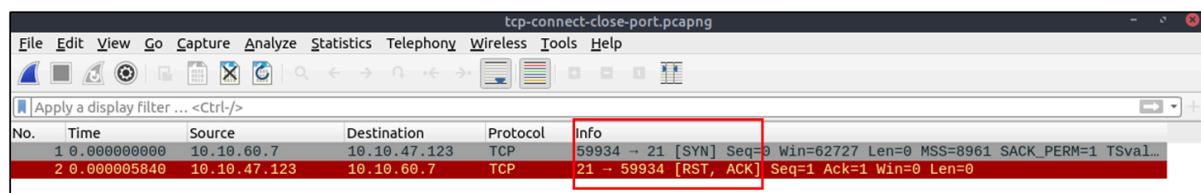
Open TCP Port	Open TCP Port	Closed TCP Port
<ul style="list-style-type: none"><li>• SYN --&gt;</li><li>• &lt;-- SYN, ACK</li><li>• ACK --&gt;</li></ul>	<ul style="list-style-type: none"><li>• SYN --&gt;</li><li>• &lt;-- SYN, ACK</li><li>• ACK --&gt;</li><li>• RST, ACK --&gt;</li></ul>	<ul style="list-style-type: none"><li>• SYN --&gt;</li><li>• &lt;-- RST, ACK</li></ul>

The images below show the three-way handshake process of the open and close TCP ports. Images and pcap samples are split to make the investigation easier and understand each case's details.

Open TCP port (Connect):



Closed TCP port (Connect):



The above images provide the patterns in isolated traffic. However, it is not always easy to spot the given patterns in big capture files. Therefore analysts need to use a generic filter to view the initial anomaly patterns, and then it will be easier to

Focus on a specific traffic point. The given filter shows the TCP Connect scan patterns in a capture file.

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

Exercise-TCP-Connect.pcapng						
No.	Time	Source	Destination	Protocol	Info	
1	0.000000000	10.10.60.7	10.10.47.123	TCP	45836 → 135 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 S	
2	0.000000130	10.10.60.7	10.10.47.123	TCP	33436 → 23 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SA	
3	0.000012991	10.10.60.7	10.10.47.123	TCP	34242 → 1025 [SYN] Seq=0 Win=62727 Len=0 MSS=8961	
4	0.000013031	10.10.60.7	10.10.47.123	TCP	49110 → 8888 [SYN] Seq=0 Win=62727 Len=0 MSS=8961	
5	0.000013071	10.10.60.7	10.10.47.123	TCP	51038 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 S	
11	0.000059761	10.10.60.7	10.10.47.123	TCP	36958 → 22 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SA	
13	0.000110152	10.10.60.7	10.10.47.123	TCP	59934 → 21 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SA	
14	0.000110252	10.10.60.7	10.10.47.123	TCP	50882 → 554 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 S	
17	0.000131872	10.10.60.7	10.10.47.123	TCP	59022 → 111 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 S	

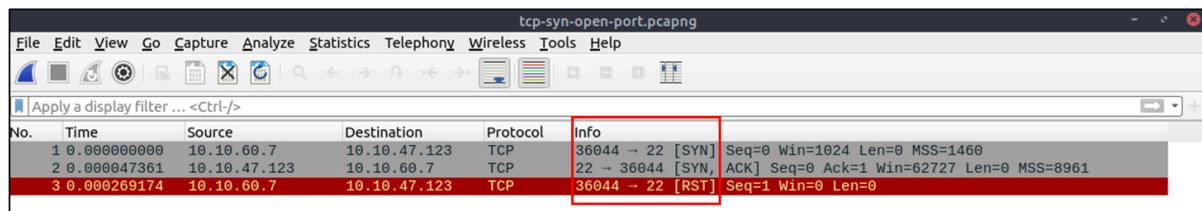
# SYN Scans

TCP SYN Scan in a nutshell:

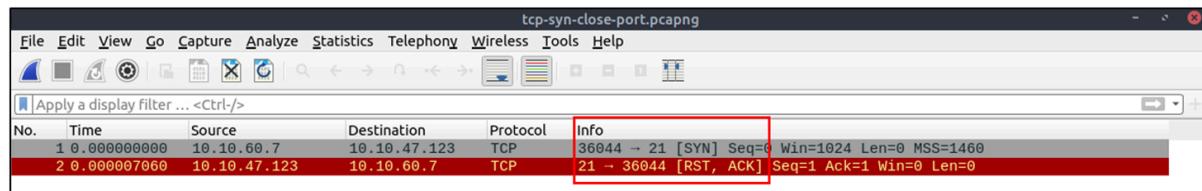
- Doesn't rely on the three-way handshake (no need to finish the handshake process).
- Usually conducted with `nmap -sS` command.
- Used by privileged users.
- Usually have a size less than or equal to 1024 bytes as the request is not finished and it doesn't expect to receive data.

Open TCP Port	Close TCP Port
<ul style="list-style-type: none"><li>• SYN --&gt;</li><li>• &lt;-- SYN,ACK</li><li>• RST--&gt;</li></ul>	<ul style="list-style-type: none"><li>• SYN --&gt;</li><li>• &lt;-- RST,ACK</li></ul>

Open TCP port (SYN):



Closed TCP port (SYN):



The given filter shows the TCP SYN scan patterns in a capture file.

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

Exercise-TCP-SYN.pcapng						
No.	Time	Source	Destination	Protocol	Info	
1	0.000000000	10.10.60.7	10.10.47.123	TCP	36044 → 445 [SYN] Seq=0 Win=1024 Len=0 MSS=1460	
2	0.000000090	10.10.60.7	10.10.47.123	TCP	36044 → 1723 [SYN] Seq=0 Win=1024 Len=0 MSS=1460	
3	0.000000240	10.10.60.7	10.10.47.123	TCP	36044 → 22 [SYN] Seq=0 Win=1024 Len=0 MSS=1460	
4	0.000000250	10.10.60.7	10.10.47.123	TCP	36044 → 3306 [SYN] Seq=0 Win=1024 Len=0 MSS=1460	
5	0.000000290	10.10.60.7	10.10.47.123	TCP	36044 → 995 [SYN] Seq=0 Win=1024 Len=0 MSS=1460	
11	0.000079761	10.10.60.7	10.10.47.123	TCP	36044 → 111 [SYN] Seq=0 Win=1024 Len=0 MSS=1460	
13	0.000091302	10.10.60.7	10.10.47.123	TCP	36044 → 135 [SYN] Seq=0 Win=1024 Len=0 MSS=1460	
14	0.000091372	10.10.60.7	10.10.47.123	TCP	36044 → 8080 [SYN] Seq=0 Win=1024 Len=0 MSS=1460	
15	0.000091412	10.10.60.7	10.10.47.123	TCP	36044 → 25 [SYN] Seq=0 Win=1024 Len=0 MSS=1460	

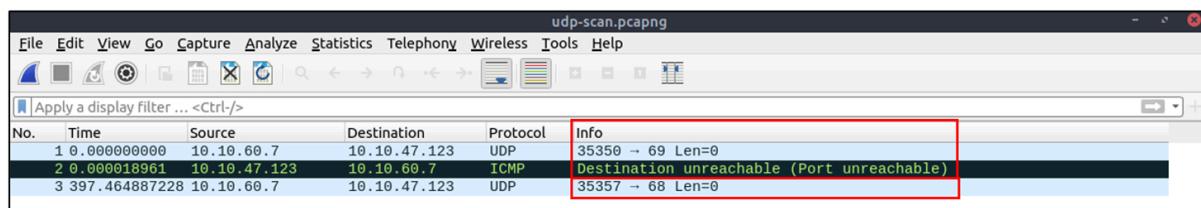
# UDP Scans

UDP Scan in a nutshell:

- Doesn't require a handshake process
- No prompt for open ports
- ICMP error message for close ports
- Usually conducted with `nmap -sU` command.

Open UDP Port	Closed UDP Port
<ul style="list-style-type: none"><li>• UDP packet --&gt;</li></ul>	<ul style="list-style-type: none"><li>• UDP packet --&gt;</li><li>• ICMP Type 3, Code 3 message. (Destination unreachable, port unreachable)</li></ul>

Closed (port no 69) and open (port no 68) UDP ports:



The above image shows that the closed port returns an ICMP error packet. No further information is provided about the error at first glance, so how can an analyst decide where this error message belongs? The ICMP error message uses the original request as encapsulated data to show the source/reason of the packet. Once you expand the ICMP section in the packet details pane, you will see the encapsulated data and the original request, as shown in the below image.

Two screenshots of Wireshark showing network traffic analysis.

**Screenshot 1:** The interface is titled "udp-scan.pcapng". The packet list shows three entries:

- Frame 1: UDP from 10.10.60.7 to 10.10.47.123, Port 35350 to 69, Length 8 bytes. Info: Destination unreachable (Port unreachable).
- Frame 2: ICMP from 10.10.47.123 to 10.10.60.7, Port 35350 to 69, Length 8 bytes. Info: Destination unreachable (Port unreachable).
- Frame 3: UDP from 10.10.60.7 to 10.10.47.123, Port 35357 to 68, Length 8 bytes.

The details pane for Frame 1 shows the following structure:

```

> Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface
> Ethernet II, Src: 02:6d:30:b1:b9:69, Dst: 02:46:92:ec:ed:bd
> Internet Protocol Version 4, Src: 10.10.60.7, Dst: 10.10.47.123
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        Total Length: 28
        Identification: 0x4953 (18771)
        Flags: 0x0000
        Fragment offset: 0
        Time to live: 46
        Protocol: UDP (17)
        Header checksum: 0xc3e8 [validation disabled]
        [Header checksum status: Unverified]
        Source: 10.10.60.7
        Destination: 10.10.47.123
    - User Datagram Protocol, Src Port: 35350, Dst Port: 69
        Source Port: 35350
        Destination Port: 69
        Length: 8
        Checksum: 0xf5ec [unverified]
        [Checksum Status: Unverified]
        [Stream index: 0]
  
```

The bytes pane for Frame 1 shows the raw hex and ASCII data.

**Screenshot 2:** The interface is titled "udp-scan.pcapng". The packet list shows three entries:

- Frame 1: UDP from 10.10.60.7 to 10.10.47.123, Port 35350 to 69, Length 8 bytes. Info: Destination unreachable (Port unreachable).
- Frame 2: ICMP from 10.10.47.123 to 10.10.60.7, Port 35350 to 69, Length 8 bytes. Info: Destination unreachable (Port unreachable).
- Frame 3: UDP from 10.10.60.7 to 10.10.47.123, Port 35357 to 68, Length 8 bytes.

The details pane for Frame 2 shows the following structure:

```

> Frame 2: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface
> Ethernet II, Src: 02:46:92:ec:ed:bd, Dst: 02:6d:30:b1:b9:69
> Internet Protocol Version 4, Src: 10.10.47.123, Dst: 10.10.60.7
> Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 3 (Port unreachable)
    Checksum: 0x7cac [correct]
    [Checksum Status: Good]
    Unused: 00000000
  - Internet Protocol Version 4, Src: 10.10.60.7, Dst: 10.10.47.123
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        Total Length: 28
        Identification: 0x4953 (18771)
        Flags: 0x0000
        Fragment offset: 0
        Time to live: 46
        Protocol: UDP (17)
        Header checksum: 0xc3e8 [validation disabled]
        [Header checksum status: Unverified]
        Source: 10.10.60.7
        Destination: 10.10.47.123
    - User Datagram Protocol, Src Port: 35350, Dst Port: 69
        Source Port: 35350
        Destination Port: 69
        Length: 8
        Checksum: 0xf5ec [unverified]
        [Checksum Status: Unverified]
        [Stream index: 0]
  
```

A yellow box labeled "ICMP Packet" highlights the ICMP structure in the details pane. A red arrow points from the "ICMP" entry in the packet list to this box. Another red arrow points from the "ICMP" entry to the "Encapsulated data" box in the bottom right.

A yellow box labeled "Encapsulated data" highlights the UDP payload in the details pane of Frame 3.

The given filter shows the UDP scan patterns in a capture file.

```
icmp.type==3 and icmp.code==3
```

Exercise-UDP.pcapng						
No.	Time	Source	Protocol	Info		
5	0.000034211	10.10.47.123	ICMP	Destination unreachable (Port unreachable)		
6	0.000038481	10.10.47.123	ICMP	Destination unreachable (Port unreachable)		
7	0.000039781	10.10.47.123	ICMP	Destination unreachable (Port unreachable)		
8	0.000041431	10.10.47.123	ICMP	Destination unreachable (Port unreachable)		
14	0.000125252	10.10.47.123	ICMP	Destination unreachable (Port unreachable)		
15	0.000127332	10.10.47.123	ICMP	Destination unreachable (Port unreachable)		

Detecting suspicious activities in chunked files is easy and a great way to learn how to focus on the details. Now use the exercise files to put your skills into practice against a single capture file and answer the questions below!

### ***Answer the questions below***

**Use the "Desktop/exercise-pcaps/nmap/Exercise.pcapng" file.**

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

# 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.

Before investigating the traffic, let's review some legitimate and suspicious ARP packets. The legitimate requests are similar to the shown picture: a broadcast request that asks if any of the available hosts use an IP address and a reply from the host that uses the particular IP address.

Notes	Wireshark filter
Global search	<ul style="list-style-type: none"><li>• arp</li></ul>

"ARP" options for grabbing the low-hanging fruits:

- Opcode 1: ARP requests.
- Opcode 2: ARP responses.
- Hunt: Arp scanning
- Hunt: Possible ARP poisoning detection
- Hunt: Possible ARP flooding from detection:

- arp.opcode == 1
- arp.opcode == 2
- arp.dst.hw\_mac==00:00:00:00:00:00
- arp.duplicate-address-detected or arp.duplicate-address-frame
- ((arp) && (arp.opcode == 1)) && (arp.src.hw\_mac == target-mac-address)

The image contains two screenshots of the Wireshark network traffic analyzer. Both screenshots show a single ARP request and its corresponding reply, with specific fields highlighted in red boxes.

**ARP Request Screenshot:**

No.	Time	Source	Destination	Protocol	Info
1	0.000000000	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	who has 192.168.1.1? Tell 192.168.1.25
2	0.001059831	50:78:b3:f3:cd:f4	00:0c:29:e2:18:b4	ARP	192.168.1.1 is at 50:78:b3:f3:cd:f4

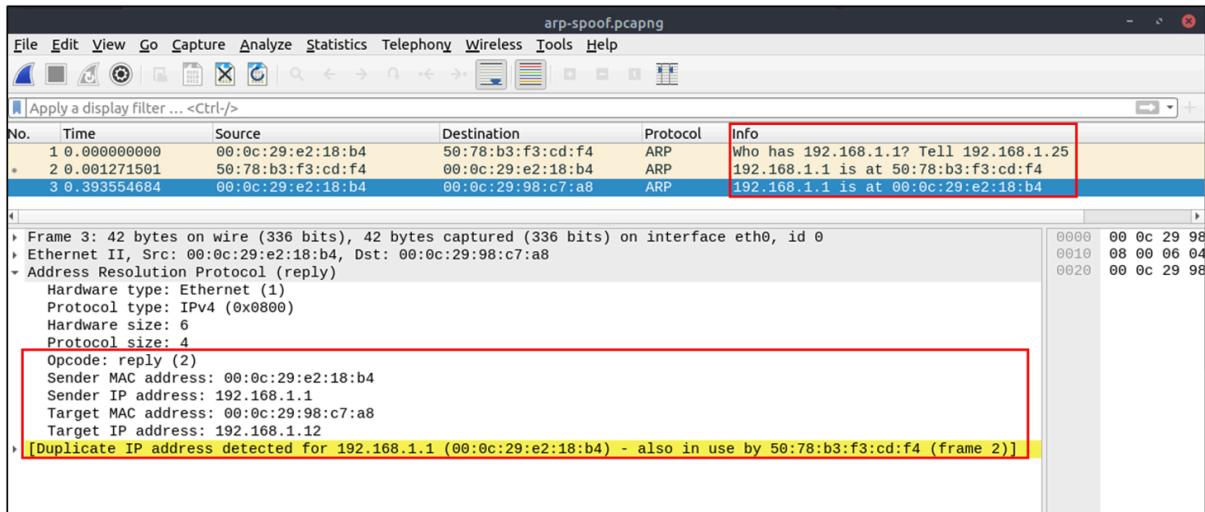
Details for Frame 1 (Request):

- Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface Ethernet II, Src: 00:0c:29:e2:18:b4, Dst: ff:ff:ff:ff:ff:ff
- Address Resolution Protocol (request)
- Hardware type: Ethernet (1)
- Protocol type: IPv4 (0x0800)
- Hardware size: 6
- Protocol size: 4
- Opcode: request (1)
- Sender MAC address: 00:0c:29:e2:18:b4
- Sender IP address: 192.168.1.25
- Target MAC address: 00:00:00:00:00:00
- Target IP address: 192.168.1.1

Details for Frame 2 (Reply):

- Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface Ethernet II, Src: 50:78:b3:f3:cd:f4, Dst: 00:0c:29:e2:18:b4
- Address Resolution Protocol (reply)
- Hardware type: Ethernet (1)
- Protocol type: IPv4 (0x0800)
- Hardware size: 6
- Protocol size: 4
- Opcode: reply (2)
- Sender MAC address: 50:78:b3:f3:cd:f4
- Sender IP address: 192.168.1.1
- Target MAC address: 00:0c:29:e2:18:b4
- Target IP address: 192.168.1.25

A suspicious situation means having two different ARP responses (conflict) for a particular IP address. In that case, Wireshark's expert info tab warns the analyst. However, it only shows the second occurrence of the duplicate value to highlight the conflict. Therefore, identifying the malicious packet from the legitimate one is the analyst's challenge. A possible IP spoofing case is shown in the picture below.



Here, knowing the network architecture and inspecting the traffic for a specific time frame can help detect the anomaly. As an analyst, you should take notes of your findings before going further. This will help you be organised and make it easier to correlate the further findings. Look at the given picture; there is a conflict; the MAC address that ends with "b4" crafted an ARP request with the "192.168.1.25" IP address, then claimed to have the "192.168.1.1" IP address.

Notes	<b>Detection Notes</b>	Findings
Possible IP address match.	1 IP address announced from a MAC address.	<ul style="list-style-type: none"> <li>• MAC: 00:0c:29:e2:18:b4</li> <li>• IP: 192.168.1.25</li> </ul>
Possible ARP spoofing attempt.	2 MAC addresses claimed the same IP address (192.168.1.1). The "192.168.1.1" IP address is a possible gateway address.	<ul style="list-style-type: none"> <li>• MAC1: 50:78:b3:f3:cd:f4</li> <li>• MAC 2: 00:0c:29:e2:18:b4</li> </ul>
Possible ARP flooding attempt.	The MAC address that ends with "b4" claims to have a different/new IP address.	<ul style="list-style-type: none"> <li>• MAC: 00:0c:29:e2:18:b4</li> <li>• IP: 192.168.1.1</li> </ul>

Let's keep inspecting the traffic to spot any other anomalies. Note that the case is split into multiple capture files to make the investigation easier.

arp-flood.pcapng					
No.	Time	Source	Destination	Protocol	Info
1	0.0000000000	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.1? Tell 192.168.1.25
2	0.001059831	50:78:b3:f3:cd:f4	00:0c:29:e2:18:b4	ARP	192.168.1.1 is at 50:78:b3:f3:cd:f4
3	0.010490253	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.37? Tell 192.168.1.25
4	0.020876839	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.158? Tell 192.168.1.25
5	0.031275821	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.212? Tell 192.168.1.25
6	0.041848453	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.176? Tell 192.168.1.25
7	0.052746298	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.73? Tell 192.168.1.25
8	0.063388601	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.216? Tell 192.168.1.25
9	0.073905794	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.181? Tell 192.168.1.25
10	0.084401792	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.217? Tell 192.168.1.25
11	0.095003040	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.173? Tell 192.168.1.25
12	0.105417559	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.136? Tell 192.168.1.25
13	0.115638938	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.132? Tell 192.168.1.25
14	0.125920898	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.130? Tell 192.168.1.25
15	0.136708415	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.254? Tell 192.168.1.25
16	0.147294383	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.232? Tell 192.168.1.25
17	0.157926474	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.162? Tell 192.168.1.25
18	0.168416850	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.109? Tell 192.168.1.25
19	0.178936116	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.253? Tell 192.168.1.25
20	0.189453650	00:0c:29:e2:18:b4	ff:ff:ff:ff:ff:ff	ARP	Who has 192.168.1.169? Tell 192.168.1.25

At this point, it is evident that there is an anomaly. A security analyst cannot ignore a flood of ARP requests. This could be malicious activity, scan or network problems. There is a new anomaly; the MAC address that ends with "b4" crafted multiple ARP requests with the "192.168.1.25" IP address. Let's focus on the source of this anomaly and extend the taken notes.

Notes	Detection Notes	Findings
Possible IP address match.	1 IP address announced from a MAC address.	<ul style="list-style-type: none"> <li>MAC: 00:0c:29:e2:18:b4</li> <li>IP: 192.168.1.25</li> </ul>
Possible ARP spoofing attempt.	2 MAC addresses claimed the same IP address (192.168.1.1). The " 192.168.1.1" IP address is a possible gateway address.	<ul style="list-style-type: none"> <li>MAC1: 50:78:b3:f3:cd:f4</li> <li>MAC 2: 00:0c:29:e2:18:b4</li> </ul>
Possible ARP spoofing attempt.	The MAC address that ends with "b4" claims to have a different/new IP address.	<ul style="list-style-type: none"> <li>MAC: 00:0c:29:e2:18:b4</li> <li>IP: 192.168.1.1</li> </ul>
Possible ARP flooding attempt.	The MAC address that ends with "b4" crafted multiple ARP requests against a range of IP addresses.	<ul style="list-style-type: none"> <li>MAC: 00:0c:29:e2:18:b4</li> <li>IP: 192.168.1.xxx</li> </ul>

Up to this point, it is evident that the MAC address that ends with "b4" owns the "192.168.1.25" IP address and crafted suspicious ARP requests against a range of IP addresses. It also claimed to have the possible gateway address as well. Let's focus on other protocols and spot the reflection of this anomaly in the following sections of the time frame.

No.	Time	Source	Destination	Protocol	Info
1	0.000000000	192.168.1.12	44.228.249.3	HTTP	GET /login.php HTTP/1.1
2	0.229915623	44.228.249.3	192.168.1.12	HTTP	Continuation
3	0.249753947	192.168.1.12	44.228.249.3	HTTP	GET /style.css HTTP/1.1
4	0.251809280	192.168.1.12	44.228.249.3	HTTP	GET /images/logo.gif HTTP/1.1
5	0.472913301	44.228.249.3	192.168.1.12	HTTP	Continuation

There is HTTP traffic, and everything looks normal at the IP level, so there is no linked information with our previous findings. Let's add the MAC addresses as columns in the packet list pane to reveal the communication behind the IP addresses.

No.	Time	Source	Source	Destination	Destination	Protocol	Info
1	0.000000000	192.168.1.12	00:0c:29:98:c7:a8	44.228.249.3	00:0c:29:e2:18:b4	HTTP	GET /login.php HTTP/1.1
2	0.229915623	44.228.249.3	50:78:b3:f3:cd:f4	192.168.1.12	00:0c:29:e2:18:b4	HTTP	Continuation
3	0.249753947	192.168.1.12	00:0c:29:98:c7:a8	44.228.249.3	00:0c:29:e2:18:b4	HTTP	GET /style.css HTTP/1.1
4	0.251809280	192.168.1.12	00:0c:29:98:c7:a8	44.228.249.3	00:0c:29:e2:18:b4	HTTP	GET /images/logo.gif HTTP/1.1
5	0.472913301	44.228.249.3	50:78:b3:f3:cd:f4	192.168.1.12	00:0c:29:e2:18:b4	HTTP	Continuation

One more anomaly! The MAC address that ends with "b4" is the destination of all HTTP packets! It is evident that there is a MITM attack, and the attacker is the host with the MAC address that ends with "b4". All traffic linked to "192.168.1.12" IP addresses is forwarded to the malicious host. Let's summarise the findings before concluding the investigation.

Detection Notes	Findings	
IP to MAC matches.	3 IP to MAC address matches.	<ul style="list-style-type: none"> <li>MAC: 00:0c:29:e2:18:b4 = IP: 192.168.1.25</li> <li>MAC: 50:78:b3:f3:cd:f4 = IP: 192.168.1.1</li> <li>MAC: 00:0c:29:98:c7:a8 = IP: 192.168.1.12</li> </ul>

Attacker	The attacker created noise with ARP packets.	<ul style="list-style-type: none"> <li>• MAC: 00:0c:29:e2:18:b4 = IP: 192.168.1.25</li> </ul>
Router/gateway	Gateway address.	<ul style="list-style-type: none"> <li>• MAC: 50:78:b3:f3:cd:f4 = IP: 192.1681.1</li> </ul>
Victim	The attacker sniffed all traffic of the victim.	<ul style="list-style-type: none"> <li>• MAC: 50:78:b3:f3:cd:f4 = IP: 192.1681.12</li> </ul>

Detecting these bits and pieces of information in a big capture file is challenging. However, in real-life cases, you will not have "tailored data" ready for investigation. Therefore you need to have the analyst mindset, knowledge and tool skills to filter and detect the anomalies.

Note: In traffic analysis, there are always alternative solutions available. The solution type and the approach depend on the analyst's knowledge and skill level and the available data sources.

Detecting suspicious activities in chunked files is easy and a great way to learn how to focus on the details. Now use the exercise files to put your skills into practice against a single capture file and answer the questions below!

# Identifying Hosts: DHCP, NetBIOS and Kerberos

## Identifying Hosts

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

## DHCP Analysis

DHCP protocol, or Dynamic Host Configuration Protocol (DHCP), is the technology responsible for managing automatic IP address and required communication parameters assignment.

DHCP investigation in a nutshell:

Notes	Wireshark Filter
Global search.	<ul style="list-style-type: none"><li>• <code>dhcp or bootp</code></li></ul>

Filtering the proper DHCP packet options is vital to finding an event of interest.

- "DHCP Request" packets contain the hostname information
- "DHCP ACK" packets represent the accepted requests
- "DHCP NAK" packets represent denied requests

Due to the nature of the protocol, only "Option 53" (request type) has predefined static values. You should filter the packet type first, and then you can filter the rest of the options by "applying as column" or use the advanced filters like "contains" and "matches".

- Request: `dhcp.option.dhcp == 3`
- ACK: `dhcp.option.dhcp == 5`
- NAK: `dhcp.option.dhcp == 6`

"DHCP Request" options for grabbing the low-hanging fruits:

- Option 12: Hostname.
- Option 50: Requested IP address.
- Option 51: Requested IP lease time.
- Option 61: Client's MAC address.

- `dhcp.option.hostname contains "keyword"`

"DHCP ACK" options for grabbing the low-hanging fruits:

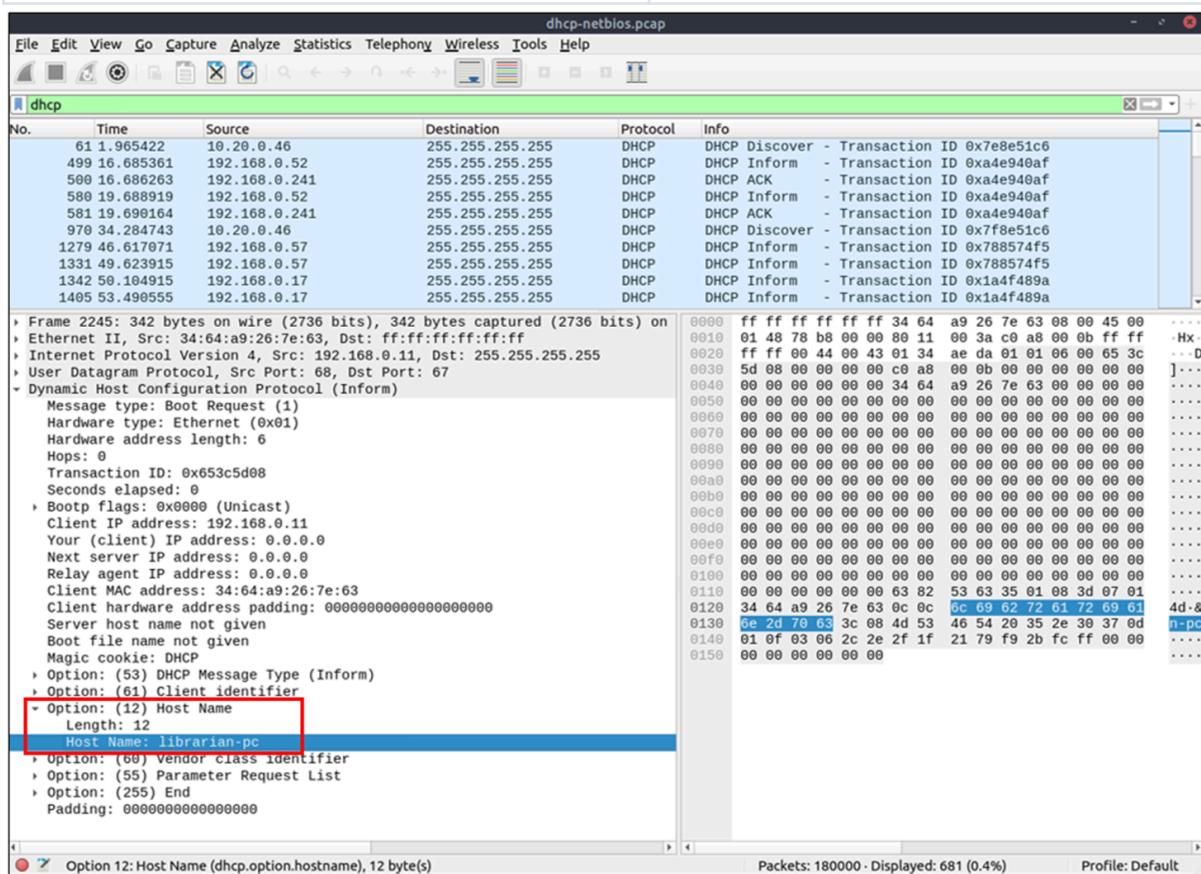
- Option 15: Domain name.
- Option 51: Assigned IP lease time.

- `dhcp.option.domain_name contains "keyword"`

"DHCP NAK" options for grabbing the low-hanging fruits:

- Option 56: Message (rejection details/reason).

As the message could be unique according to the case/situation, It is suggested to read the message instead of filtering it. Thus, the analyst could create a more reliable hypothesis/result by understanding the event circumstances.



## NetBIOS (NBNS) Analysis

NetBIOS or Network Basic Input/Output System is the technology responsible for allowing applications on different hosts to communicate with each other.

NBNS investigation in a nutshell:

Notes

Wireshark Filter

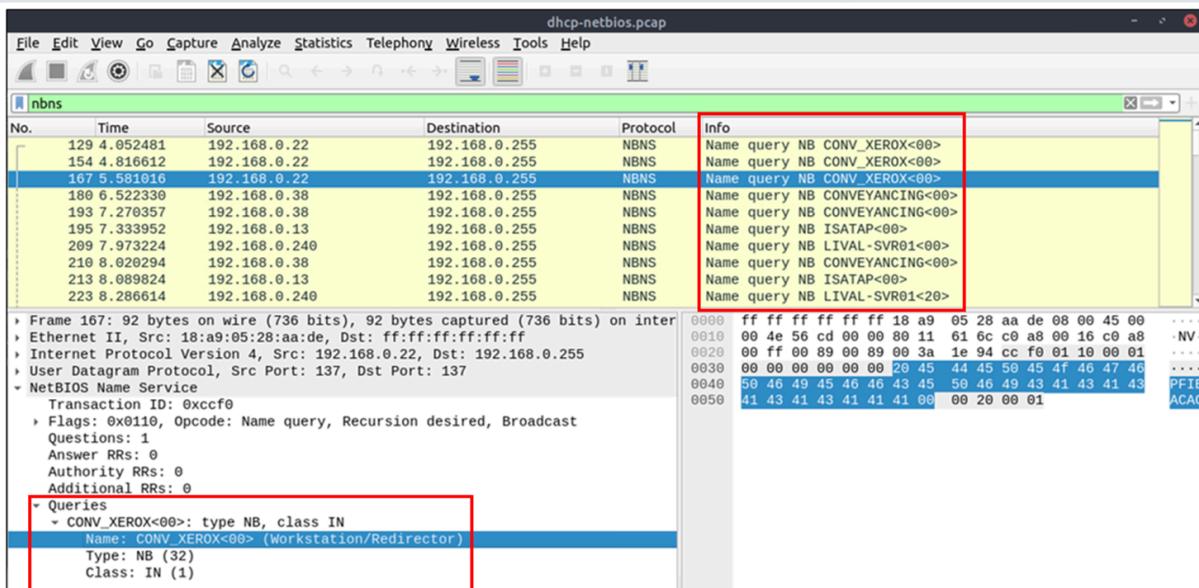
Global search.

- nbns

"NBNS" options for grabbing the low-hanging fruits:

- Queries: Query details.
- Query details could contain "name, Time to live (TTL) and IP address details"

- nbns.name  
contains  
"keyword"



## Kerberos Analysis

Kerberos is the default authentication service for Microsoft Windows domains. It is responsible for authenticating service requests between two or more computers over the untrusted network. The ultimate aim is to prove identity securely.

Kerberos investigation in a nutshell:

Notes

Wireshark  
Filter

Global search.

- kerberos

## User account search:

- CNameString: The username.

Note: Some packets could provide hostname information in this field. To avoid this confusion, filter the "\$" value. The values end with "\$" are hostnames, and the ones without it are user names.

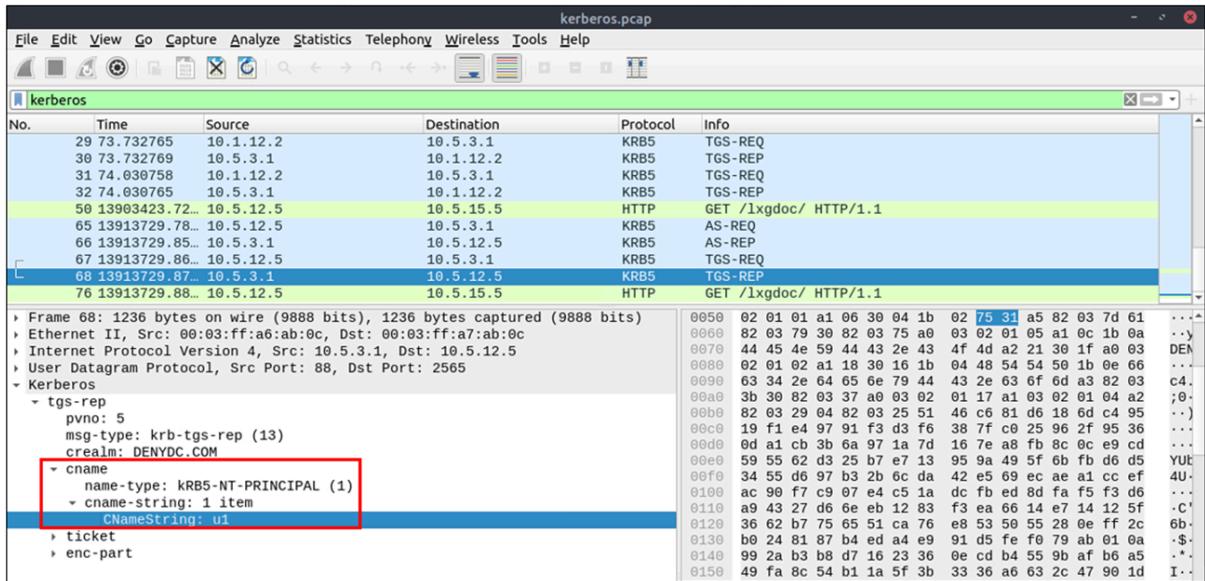
- `kerberos.CNameString contains "keyword"`
- `kerberos.CNameString and !(kerberos.CNameString contains "$")`

## "Kerberos" options for grabbing the low-hanging fruits:

- pvno: Protocol version.
- realm: Domain name for the generated ticket.
- sname: Service and domain name for the generated ticket.
- addresses: Client IP address and NetBIOS name.

- `kerberos.pvno == 5`
- `kerberos.realm contains ".org"`
- `kerberos.SNameString == "krbtg"`

Note: the "addresses" information is only available in request packets.



Detecting suspicious activities in chunked files is easy and a great way to learn how to focus on the details. Now use the exercise files to put your skills into practice against a single capture file and answer the questions below!

# Identifying Hosts

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

## DHCP Analysis

DHCP protocol, or Dynamic Host Configuration Protocol (DHCP), is the technology responsible for managing automatic IP address and required communication parameters assignment.

DHCP investigation in a nutshell:

Notes	Wireshark Filter
Global search.	<ul style="list-style-type: none"><li>• <code>dhcp or bootp</code></li></ul>

Filtering the proper DHCP packet options is vital to finding an event of interest.

- "DHCP Request" packets contain the hostname information
- "DHCP ACK" packets represent the accepted requests
- "DHCP NAK" packets represent denied requests

Due to the nature of the protocol, only "Option 53" (request type) has predefined static values. You should filter the packet type first, and then you can filter the rest of the options by "applying as column" or use the advanced filters like "contains" and "matches".

- Request: `dhcp.option.dhcp == 3`
- ACK: `dhcp.option.dhcp == 5`
- NAK: `dhcp.option.dhcp == 6`

"DHCP Request" options for grabbing the low-hanging fruits:

- Option 12: Hostname.
- Option 50: Requested IP address.
- Option 51: Requested IP lease time.
- Option 61: Client's MAC address.

- `dhcp.option.hostname contains "keyword"`

"DHCP ACK" options for grabbing the low-hanging fruits:

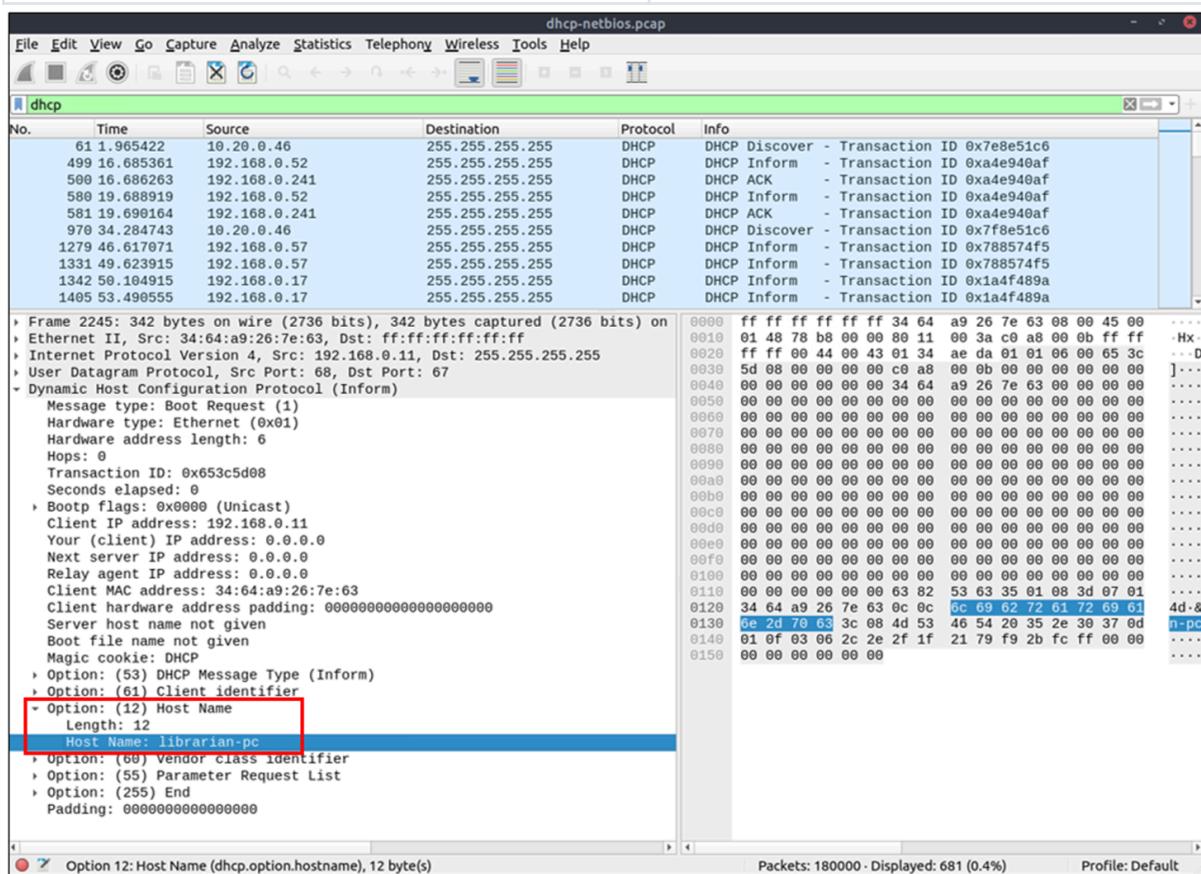
- Option 15: Domain name.
- Option 51: Assigned IP lease time.

- `dhcp.option.domain_name contains "keyword"`

"DHCP NAK" options for grabbing the low-hanging fruits:

- Option 56: Message (rejection details/reason).

As the message could be unique according to the case/situation, It is suggested to read the message instead of filtering it. Thus, the analyst could create a more reliable hypothesis/result by understanding the event circumstances.



## NetBIOS (NBNS) Analysis

NetBIOS or Network Basic Input/Output System is the technology responsible for allowing applications on different hosts to communicate with each other.

NBNS investigation in a nutshell:

Notes

Wireshark Filter

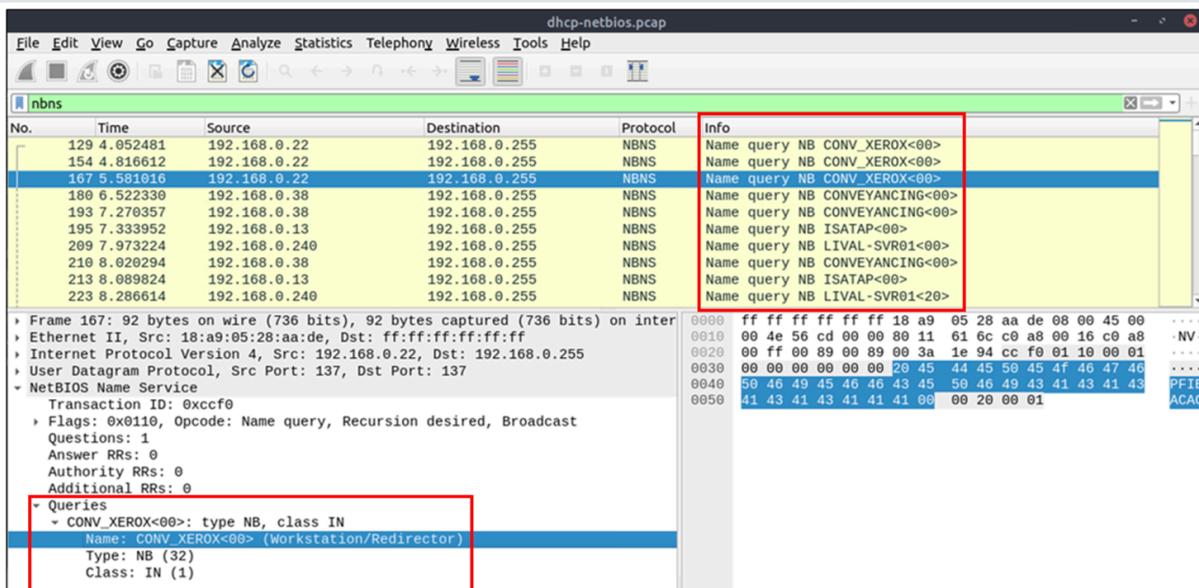
Global search.

- nbns

"NBNS" options for grabbing the low-hanging fruits:

- Queries: Query details.
- Query details could contain "name, Time to live (TTL) and IP address details"

- nbns.name  
contains  
"keyword"



## Kerberos Analysis

Kerberos is the default authentication service for Microsoft Windows domains. It is responsible for authenticating service requests between two or more computers over the untrusted network. The ultimate aim is to prove identity securely.

Kerberos investigation in a nutshell:

Notes

Wireshark  
Filter

Global search.

- kerberos

## User account search:

- CNameString: The username.

Note: Some packets could provide hostname information in this field. To avoid this confusion, filter the "\$" value. The values end with "\$" are hostnames, and the ones without it are user names.

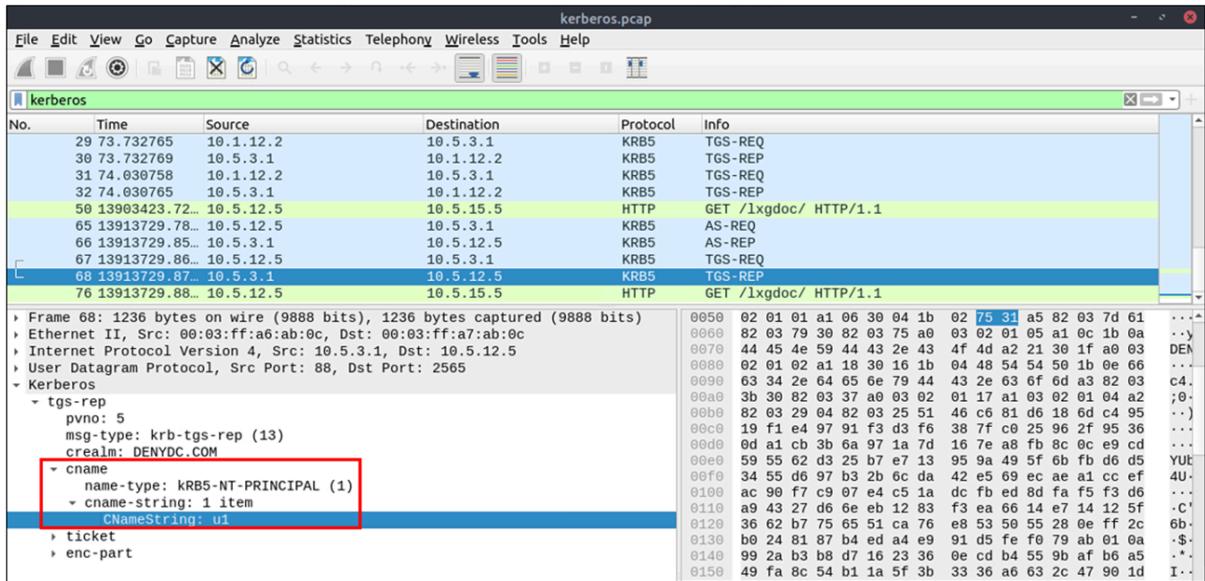
- `kerberos.CNameString contains "keyword"`
- `kerberos.CNameString and !(kerberos.CNameString contains "$")`

## "Kerberos" options for grabbing the low-hanging fruits:

- pvno: Protocol version.
- realm: Domain name for the generated ticket.
- sname: Service and domain name for the generated ticket.
- addresses: Client IP address and NetBIOS name.

- `kerberos.pvno == 5`
- `kerberos.realm contains ".org"`
- `kerberos.SNameString == "krbtg"`

Note: the "addresses" information is only available in request packets.



Detecting suspicious activities in chunked files is easy and a great way to learn how to focus on the details. Now use the exercise files to put your skills into practice against a single capture file and answer the questions below!

# 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. As mentioned earlier at the beginning of the Wireshark room series, the analyst should have the required network knowledge and tool skills to accomplish this. Let's simulate a cleartext protocol investigation with Wireshark!

## FTP Analysis

File Transfer Protocol (FTP) is designed to transfer files with ease, so it focuses on simplicity rather than security. As a result of this, using this protocol in unsecured environments could create security issues like:

- MITM attacks
- Credential stealing and unauthorised access
- Phishing
- Malware planting
- Data exfiltration

FTP analysis in a nutshell:

Notes	Wireshark Filter
Global search	<ul style="list-style-type: none"><li>• <code>ftp</code></li></ul>
"FTP" options for grabbing the low-hanging fruits: <ul style="list-style-type: none"><li>• x1x series: Information request responses.</li><li>• x2x series: Connection messages.</li><li>• x3x series: Authentication messages.</li></ul>	---
Note: "200" means command successful.	

"x1x" series options for grabbing the low-hanging fruits:

- 211: System status.
- 212: Directory status.
- 213: File status

- `ftp.response.code == 211`

"x2x" series options for grabbing the low-hanging fruits:

- 220: Service ready.
- 227: Entering passive mode.
- 228: Long passive mode.
- 229: Extended passive mode.

- `ftp.response.code == 227`

"x3x" series options for grabbing the low-hanging fruits:

- 230: User login.
- 231: User logout.
- 331: Valid username.
- 430: Invalid username or password
- 530: No login, invalid password.

- `ftp.response.code == 230`

"FTP" commands for grabbing the low-hanging fruits:

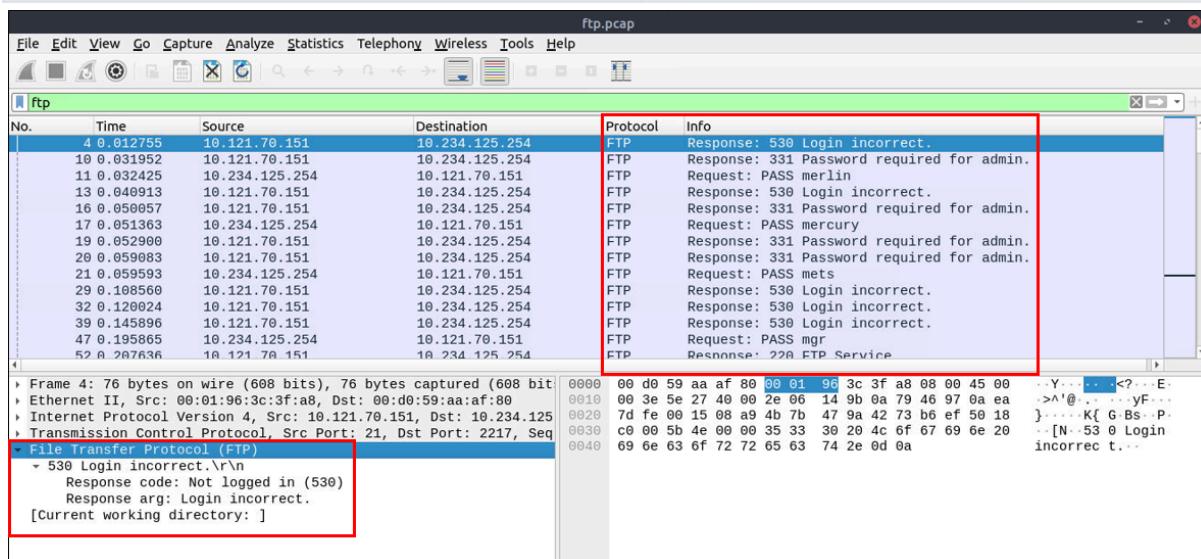
- USER: Username.
- PASS: Password.
- CWD: Current work directory.
- LIST: List.

- `ftp.request.command == "USER"`
- `ftp.request.command == "PASS"`
- `ftp.request.arg == "password"`

## Advanced usages examples for grabbing low-hanging fruits:

- Bruteforce signal: List failed login attempts.
- Bruteforce signal: List target username.
- Password spray signal: List targets for a static password.

- `ftp.response.code == 530`
- `(ftp.response.code == 530)`  
and `(ftp.response.arg contains "username")`
- `(ftp.request.command == "PASS" ) and`  
`(ftp.request.arg == "password")`



Detecting suspicious activities in chunked files is easy and a great way to learn how to focus on the details. Now use the exercise files to put your skills into practice against a single capture file and answer the questions below!

# Cleartext Protocol Analysis: HTTP

## HTTP Analysis

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)

HTTP analysis in a nutshell:

Notes	Wireshark Filter
<p>Global search</p> <p>Note: HTTP2 is a revision of the HTTP protocol for better performance and security. It supports binary data transfer and request&amp;response multiplexing.</p>	<ul style="list-style-type: none"><li>• <code>http</code></li><li>• <code>http2</code></li></ul>
<p>"HTTP Request Methods" for grabbing the low-hanging fruits:</p> <ul style="list-style-type: none"><li>• GET</li><li>• POST</li><li>• Request: Listing all requests</li></ul>	<ul style="list-style-type: none"><li>• <code>http.request.method == "GET"</code></li><li>• <code>http.request.method == "POST"</code></li><li>• <code>http.request</code></li></ul>

## "HTTP Response Status Codes" for grabbing the low-hanging fruits:

- 200 OK: Request successful.
- 301 Moved Permanently: Resource is moved to a new URL/path (permanently).
- 302 Moved Temporarily: Resource is moved to a new URL/path (temporarily).
- 400 Bad Request: Server didn't understand the request.
- 401 Unauthorised: URL needs authorisation (login, etc.).
- 403 Forbidden: No access to the requested URL.
- 404 Not Found: Server can't find the requested URL.
- 405 Method Not Allowed: Used method is not suitable or blocked.
- 408 Request Timeout: Request took longer than server wait time.
- 500 Internal Server Error: Request not completed, unexpected error.
- 503 Service Unavailable: Request not completed server or service is down.

- `http.respo  
nse.code  
== 200`
- `http.respo  
nse.code  
== 401`
- `http.respo  
nse.code  
== 403`
- `http.respo  
nse.code  
== 404`
- `http.respo  
nse.code  
== 405`
- `http.respo  
nse.code  
== 503`

## "HTTP Parameters" for grabbing the low-hanging fruits:

- User agent: Browser and operating system identification to a web server application.
- Request URI: Points the requested resource from the server.
- Full \*URI: Complete URI information.

\*URI: Uniform Resource Identifier.

- `http.user_  
agent  
contains  
"nmap"`
- `http.reque  
st.uri  
contains  
"admin"`
- `http.reque  
st.full_ur  
i contains  
"admin"`

"HTTP Parameters" for grabbing the low-hanging fruits:

- Server: Server service name.
- Host: Hostname of the server
- Connection: Connection status.
- Line-based text data: Cleartext data provided by the server.
- HTML Form URL Encoded: Web form information.

- http.server contains "apache"
- http.host contains "keyword"
- http.host == "keyword"
- http.connection == "Keep-Alive"
- data-text-lines contains "keyword"

## User Agent Analysis

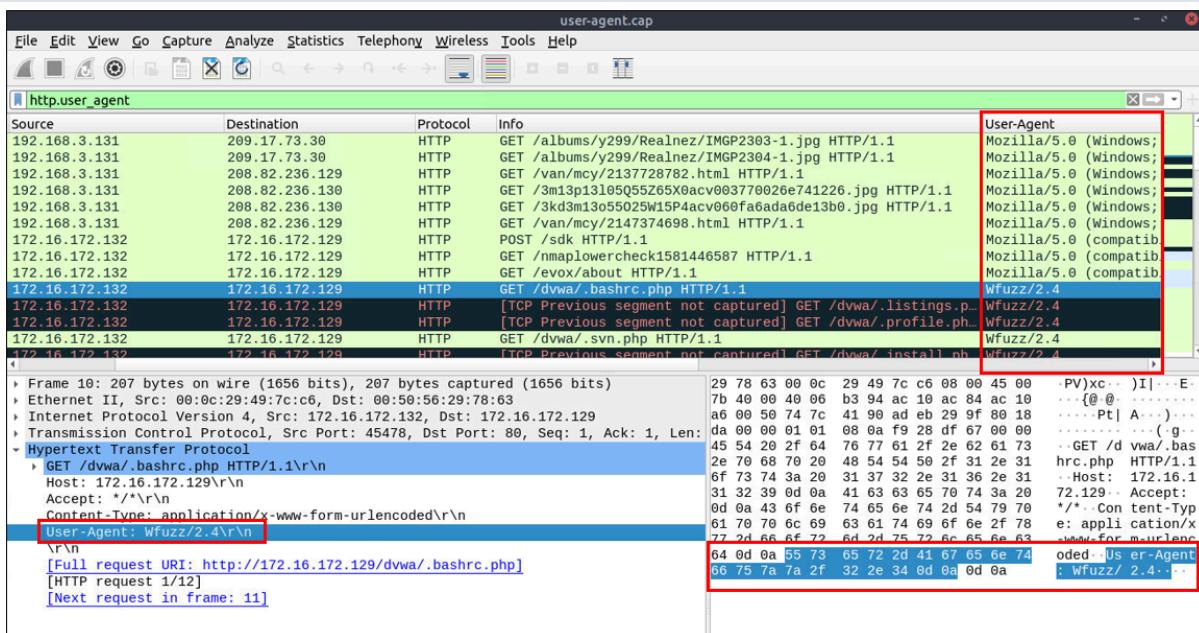
As the adversaries use sophisticated technics to accomplish attacks, they try to leave traces similar to natural traffic through the known and trusted protocols. For a security analyst, it is important to spot the anomaly signs on the bits and pieces of the packets. The "user-agent" field is one of the great resources for spotting anomalies in HTTP traffic. In some cases, adversaries successfully modify the user-agent data, which could look super natural. A security analyst cannot rely only on the user-agent field to spot an anomaly. Never whitelist a user agent, even if it looks natural. User agent-based anomaly/threat detection/hunting is an additional data source to check and is useful when there is an obvious anomaly. If you are unsure about a value, you can conduct a web search to validate your findings with the default and normal user-agent info ([example site](#)).

### User Agent analysis in a nutshell:

Notes	Wireshark Filter
Global search.	<ul style="list-style-type: none"><li>• http.user_agent</li></ul>

## Research outcomes for grabbing the low-hanging fruits:

- Different user agent information from the same host in a short time notice.
  - Non-standard and custom user agent info.
  - Subtle spelling differences. ("Mozilla" is not the same as "Mozlilla" or "Mozlila")
  - Audit tools info like Nmap, Nikto, Wfuzz and sqlmap in the user agent field.
  - Payload data in the user agent field.
- (http.user\_agent contains "sqlmap") or (http.user\_agent contains "Nmap") or (http.user\_agent contains "Wfuzz") or (http.user\_agent contains "Nikto")



## Log4j Analysis

A proper investigation starts with prior research on threats and anomalies going to be hunted. Let's review the knowns on the "Log4j" attack before launching Wireshark.

### Log4j vulnerability analysis in a nutshell:

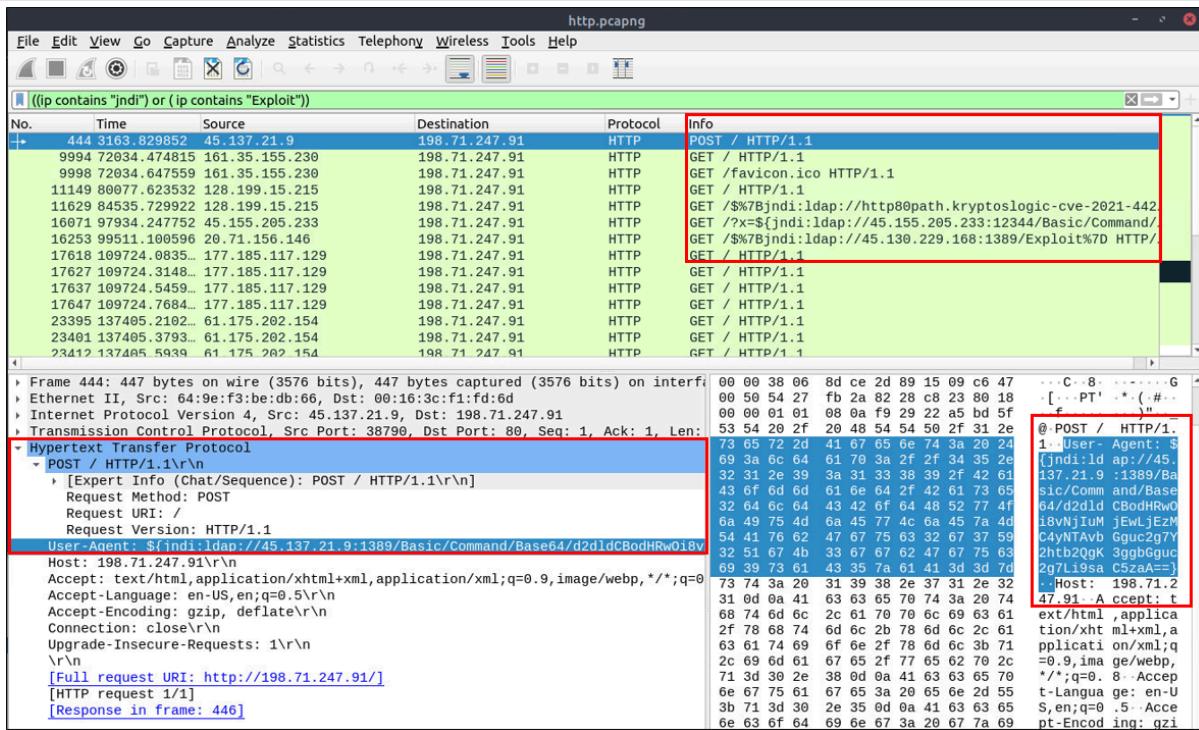
## Notes

Research outcomes for grabbing the low-hanging fruits:

- The attack starts with a "POST" request
- There are known cleartext patterns: "jndi:ldap" and "Exploit.class".

## Wireshark Filters

- `http.request.method == "POST"`
- `(ip contains "jndi") or (ip contains "Exploit")`
- `(frame contains "jndi") or (frame contains "Exploit")`
- `(http.user_agent contains "$") or (http.user_agent contains "==")`



Detecting suspicious activities in chunked files is easy and a great way to learn how to focus on the details. Now use the exercise files to put your skills into practice against a single capture file and answer the questions below!

# Encrypted Protocol Analysis: Decrypting HTTPS

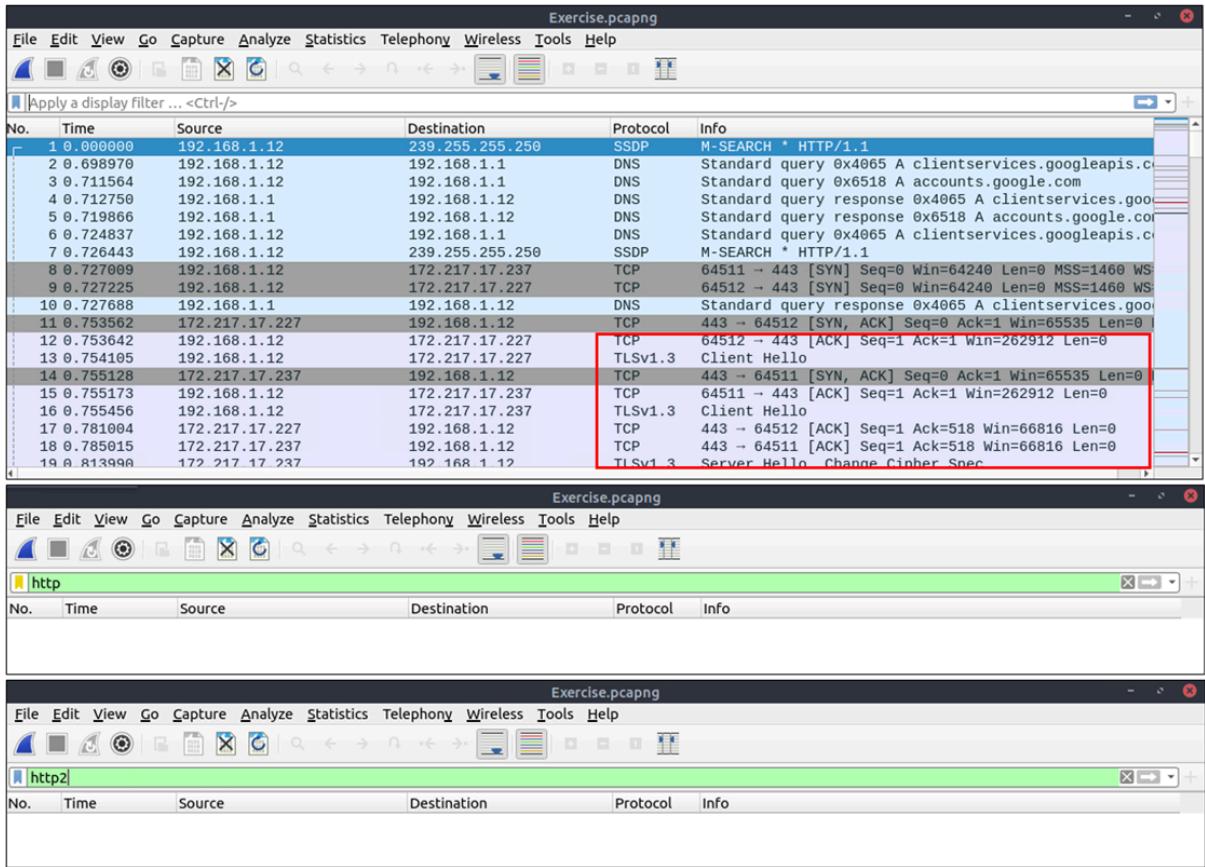
## Decrypting HTTPS Traffic

When investigating web traffic, analysts often run across encrypted traffic. This is caused by using the Hypertext Transfer Protocol Secure (HTTPS) protocol for enhanced security against spoofing, sniffing and intercepting attacks. HTTPS uses TLS protocol to encrypt communications, so it is impossible to decrypt the traffic and view the transferred data without having the encryption/decryption key pairs. As this protocol provides a good level of security for transmitting sensitive data, attackers and malicious websites also use HTTPS. Therefore, a security analyst should know how to use key files to decrypt encrypted traffic and investigate the traffic activity.

The packets will appear in different colours as the HTTP traffic is encrypted. Also, protocol and info details (actual URL address and data returned from the server) will not be fully visible. The first image below shows the HTTP packets encrypted with the TLS protocol. The second and third images demonstrate filtering HTTP packets without using a key log file.

### Additional information for HTTPS :

Notes	Wireshark Filter
<p>"HTTPS Parameters" for grabbing the low-hanging fruits:</p> <ul style="list-style-type: none"><li>• Request: Listing all requests</li><li>• TLS: Global TLS search</li><li>• TLS Client Request</li><li>• TLS Server response</li><li>• Local Simple Service Discovery Protocol (SSDP)</li></ul> <p>Note: SSDP is a network protocol that provides advertisement and discovery of network services.</p>	<ul style="list-style-type: none"><li>• <code>http.request</code></li><li>• <code>tls</code></li><li>• <code>tls.handshake.type == 1</code></li><li>• <code>tls.handshake.type == 2</code></li><li>• <code>ssdp</code></li></ul>



Similar to the TCP three-way handshake process, the TLS protocol has its handshake process. The first two steps contain "Client Hello" and "Server Hello" messages. The given filters show the initial hello packets in a capture file. These filters are helpful to spot which IP addresses are involved in the TLS handshake.

- Client Hello: `(http.request or tls.handshake.type == 1) and !(ssdp)`
- Server Hello: `(http.request or tls.handshake.type == 2) and !(ssdp)`

Exercise.pcapng

(http.request or tls.handshake.type == 1) and !(ssdp)

No.	Time	Source	Destination	Protocol	Info
13	0.754105	192.168.1.12	172.217.17.227	TLSv1.3	Client Hello
16	0.755456	192.168.1.12	172.217.17.237	TLSv1.3	Client Hello
53	0.889384	192.168.1.12	172.217.17.196	TLSv1.3	Client Hello
64	0.916063	192.168.1.12	172.217.17.196	TLSv1.3	Client Hello
76	0.950598	192.168.1.12	172.217.17.196	TLSv1.3	Client Hello
266	3.526591	192.168.1.12	172.217.20.74	TLSv1.3	Client Hello
289	3.575830	192.168.1.12	172.217.20.74	TLSv1.3	Client Hello
388	3.731085	192.168.1.12	172.217.20.78	TLSv1.3	Client Hello
572	4.274527	192.168.1.12	216.58.206.194	TLSv1.3	Client Hello
589	4.313172	192.168.1.12	216.58.206.198	TLSv1.3	Client Hello
606	4.330166	192.168.1.12	216.58.214.138	TLSv1.3	Client Hello
894	4.941191	192.168.1.12	172.217.17.99	TLSv1.3	Client Hello
985	5.762164	192.168.1.12	142.250.187.168	TLSv1.3	Client Hello
1	6.889572	192.168.1.12	142.250.187.131	TLSv1.3	Client Hello
1	12.523681	192.168.1.12	185.47.40.36	TLSv1.3	Client Hello
1	12.526718	192.168.1.12	185.47.40.36	TLSv1.3	Client Hello
1	16.927825	192.168.1.12	172.217.169.170	TLSv1.3	Client Hello
1	17.649620	192.168.1.12	87.238.33.7	TLSv1.2	Client Hello
1	31.720520	192.168.1.12	87.238.33.7	TLSv1.2	Client Hello

```

Frame 13: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits)
Ethernet II, Src: 00:02:09:98:c7:a8, Dst: 50:78:b3:f3:cd:f4
Internet Protocol Version 4, Src: 192.168.1.12, Dst: 172.217.17.227
Transmission Control Protocol, Src Port: 64512, Dst Port: 443, Seq: 1, Ack: 1, Len: 571
    Transport Layer Security
        - TLSv1.3 Record Layer: Handshake Protocol: Client Hello
            Content Type: Handshake (22)
            Version: TLS 1.0 (0x0301)
            Length: 512
        - Handshake Protocol: Client Hello

```

```

00b0 00 20 00 00 1d 63 6c 69 65 6e 74 73 65 72 76 69
00c0 63 65 73 2e 67 6f 6f 67 6c 65 61 70 69 73 2e 63
00d0 0f 6d 00 17 00 00 ff 01 00 01 00 00 0a 00
00e0 08 9a 9a 00 1d 00 17 00 18 00 00 00 02 01 00 00
00f0 23 00 00 00 10 00 00 00 0c 02 68 32 08 68 74 74
0100 70 2f 31 2e 31 00 05 00 05 01 00 00 00 00 00 0d
0110 00 12 00 10 04 03 08 04 04 01 05 03 08 05 05 01
0120 08 06 00 01 00 12 00 00 00 33 00 2b 00 29 9a 9a
0130 00 01 00 00 1d 00 20 8b 62 b2 81 72 d5 36 9f 9d
0140 ab 7c ad ed 25 8e 1c ab 76 03 82 5b dd d2 c7 49
0150 ed 2f 93 ff 6a d2 0b 00 2d 00 02 01 01 00 2b 00

```

Exercise.pcapng

(http.request or tls.handshake.type == 2) and !(ssdp)

No.	Time	Source	Destination	Protocol	Info
19	0.813990	172.217.17.237	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
24	0.815416	172.217.17.227	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
77	0.955601	172.217.17.196	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
87	0.975115	172.217.17.196	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
101	1.010213	172.217.17.196	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
298	3.588969	172.217.20.74	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
342	3.637284	172.217.20.74	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
398	3.792255	172.217.20.78	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
626	4.361320	216.58.206.194	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
651	4.398633	216.58.206.198	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
656	4.399291	216.58.214.138	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
901	5.001760	172.217.17.99	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
996	5.821469	142.250.187.168	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
1	6.959426	142.250.187.131	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
1	12.608801	185.47.40.36	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec, Application Data
1	12.624932	185.47.40.36	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec, Application Data
1	16.988483	172.217.169.170	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
1	17.739458	87.238.33.7	192.168.1.12	TLSv1.2	Server Hello
1	31.804500	87.238.33.7	192.168.1.12	TLSv1.2	Server Hello, Change Cipher Spec, Encrypted Handshake

```

Frame 101: 1484 bytes on wire (11872 bits), 1484 bytes captured (11872 bits)
Ethernet II, Src: 50:78:b3:f3:cd:f4, Dst: 00:02:09:98:c7:a8
Internet Protocol Version 4, Src: 192.168.1.12, Dst: 172.217.17.196
Transmission Control Protocol, Src Port: 443, Dst Port: 64515, Seq: 1, Ack: 1, Len: 1484
    Transport Layer Security
        - TLSv1.3 Record Layer: Handshake Protocol: Server Hello
            Content Type: Handshake (22)
            Version: TLS 1.2 (0x0303)
            Length: 122
        - Handshake Protocol: Server Hello
        - TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
            Content Type: Change Cipher Spec (20)
            Version: TLS 1.2 (0x0303)
            Length: 1
            Change Cipher Spec Message

```

```

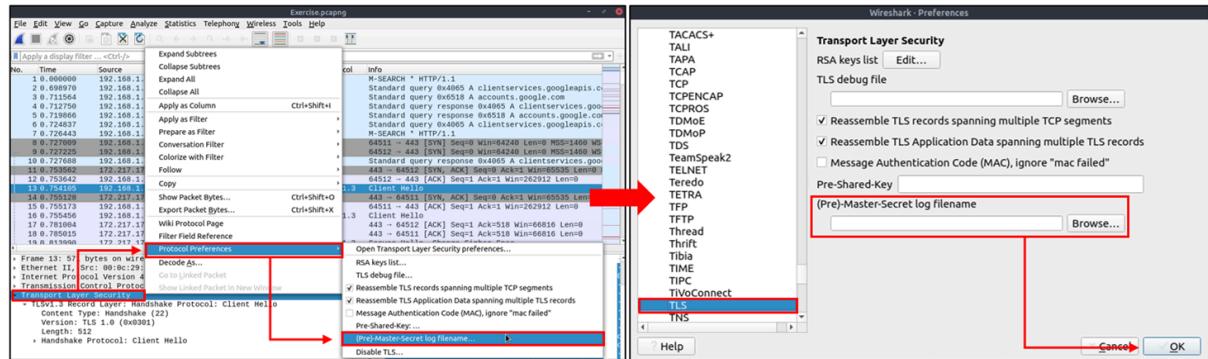
0000 00 0c 29 98 c7 a8 50 78 b3 f3 cd f4 08 00 45 00
0010 05 b4 41 6b 00 00 38 06 bb 7d ac d9 11 c0 a8
0020 01 0c 01 bf fc 03 37 6c 47 85 94 06 ab 55 50 10
0030 01 05 96 35 00 00 16 03 03 00 7a 02 00 00 76 03
0040 03 e5 c3 c2 6d ab cc 4c 5f ec 4e 38 90 8e 62 fe
0050 2e 51 b3 d8 f1 c8 2e 6c e6 b6 a8 35 a9 b6 6e 7f
0060 c5 20 ec 36 c5 e6 f9 07 40 d9 e5 90 2e 11 a1 f7
0070 2f ca 4b 16 06 2f 0b 97 1e d2 64 06 eb 32 ac 92
0080 72 7f 13 01 00 00 2e 00 33 00 24 00 1d 00 20 22
0090 f2 63 d4 fb 72 4b 4e 0c 47 de 86 aa 91 9f 2b e6
00a0 8a ce 87 9d 66 bf 5a f7 dd 75 46 7c eb 70 54 00
00b0 2b 00 02 03 04 14 03 03 00 01 01 17 03 03 10 37
00c0 78 4d 3e 4a 5a 9b f2 af de 5b c9 f5 12 6c 9d 50
00d0 c5 53 97 f3 9a 48 3c b9 5b f7 35 bf 2e 84 36 c9
00e0 a5 a4 23 c9 5a 5d e5 dc a0 b5 fd f1 cf 2c 8e 3c
00f0 86 d3 db a4 50 8a 92 a5 f9 b9 3e b7 4e 6d ca b7
0100 a4 d8 82 37 2b 68 f7 8d f3 3c 5e ab 73 d0 36 9b

```

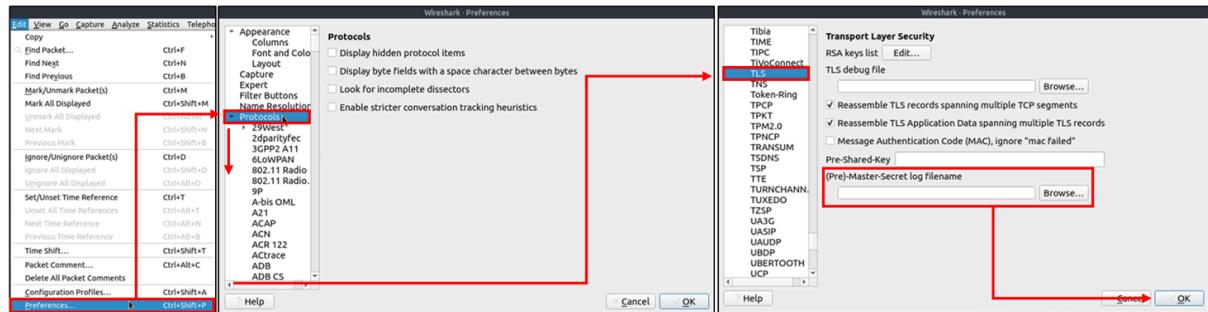
An encryption key log file is a text file that contains unique key pairs to decrypt the encrypted traffic session. These key pairs are automatically created (per session) when a connection is established with an SSL/TLS-enabled webpage. As these processes are all accomplished in the browser, you need to configure your system and use a suitable browser (Chrome and Firefox support this) to save these values as a key log file. To do this, you will need to set up an environment variable and create the `SSLKEYLOGFILE`, and the browser will dump the keys to this file as you browse the web. SSL/TLS key pairs are created per session at the connection time,

so it is important to dump the keys during the traffic capture. Otherwise, it is not possible to create/generate a suitable key log file to decrypt captured traffic. You can use the "right-click" menu or "Edit --> Preferences --> Protocols --> TLS" menu to add/remove key log files.

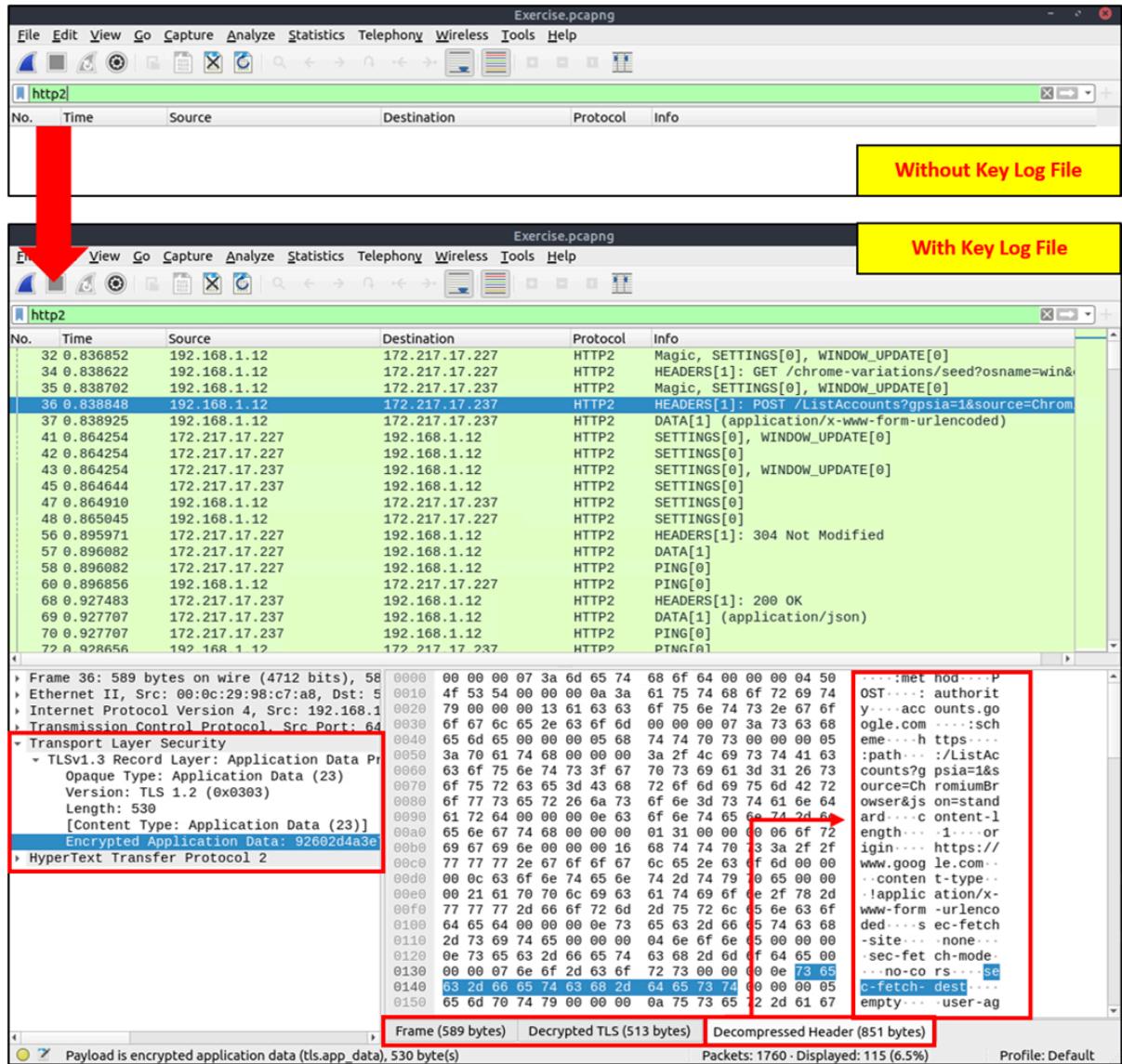
Adding key log files with the "right-click" menu:



Adding key log files with the "Edit --> Preferences --> Protocols --> TLS" menu:



Viewing the traffic with/without the key log files:



The above image shows that the traffic details are visible after using the key log file. Note that the packet details and bytes pane provides the data in different formats for investigation. Decompressed header info and HTTP2 packet details are available after decrypting the traffic. Depending on the packet details, you can also have the following data formats:

- Frame
- Decrypted TLS
- Decompressed Header
- Reassembled TCP
- Reassembled SSL

Detecting suspicious activities in chunked files is easy and a great way to learn how to focus on the details. Now use the exercise files to put your skills into practice against a single capture file and answer the questions below!

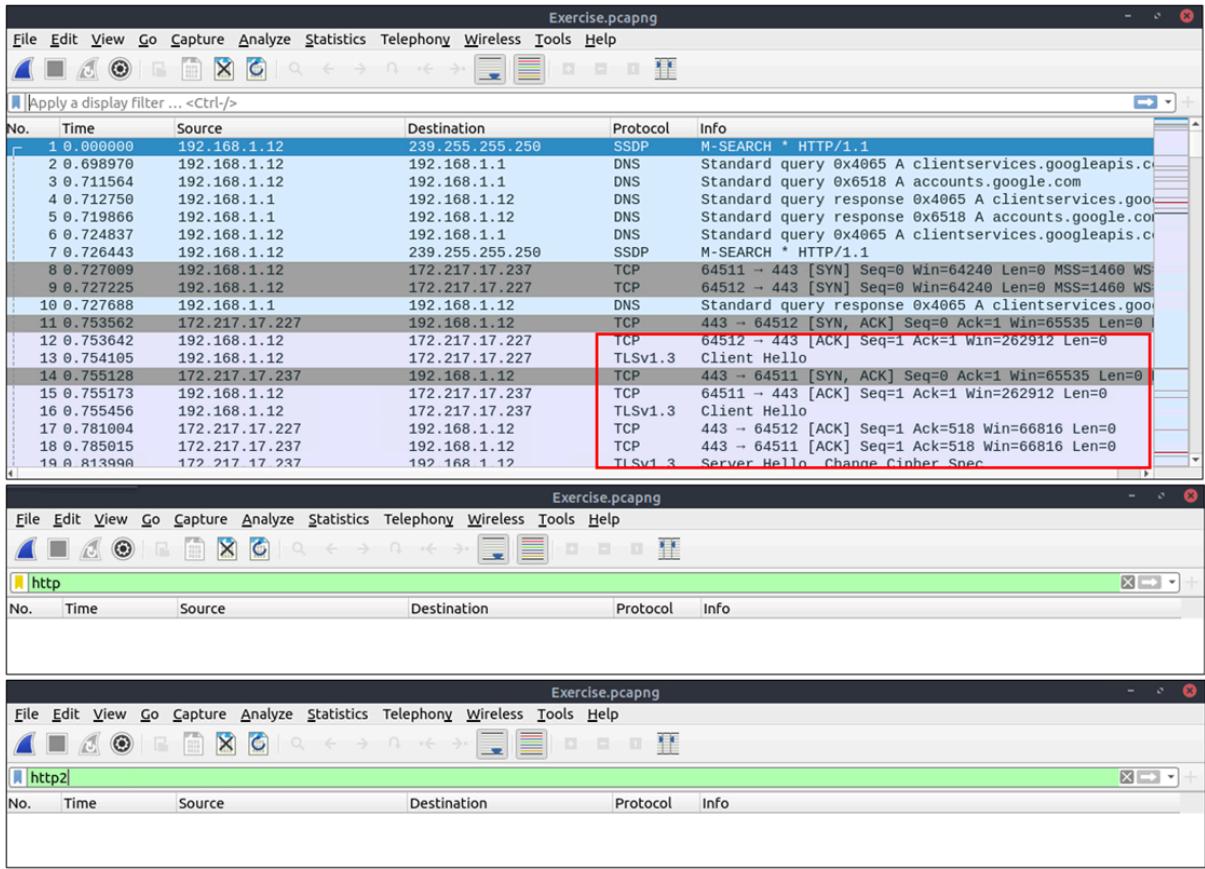
# Decrypting HTTPS Traffic

When investigating web traffic, analysts often run across encrypted traffic. This is caused by using the Hypertext Transfer Protocol Secure (HTTPS) protocol for enhanced security against spoofing, sniffing and intercepting attacks. HTTPS uses TLS protocol to encrypt communications, so it is impossible to decrypt the traffic and view the transferred data without having the encryption/decryption key pairs. As this protocol provides a good level of security for transmitting sensitive data, attackers and malicious websites also use HTTPS. Therefore, a security analyst should know how to use key files to decrypt encrypted traffic and investigate the traffic activity.

The packets will appear in different colours as the HTTP traffic is encrypted. Also, protocol and info details (actual URL address and data returned from the server) will not be fully visible. The first image below shows the HTTP packets encrypted with the TLS protocol. The second and third images demonstrate filtering HTTP packets without using a key log file.

## Additional information for HTTPS :

Notes	Wireshark Filter
<p>"HTTPS Parameters" for grabbing the low-hanging fruits:</p> <ul style="list-style-type: none"><li>• Request: Listing all requests</li><li>• TLS: Global TLS search</li><li>• TLS Client Request</li><li>• TLS Server response</li><li>• Local Simple Service Discovery Protocol (SSDP)</li></ul> <p>Note: SSDP is a network protocol that provides advertisement and discovery of network services.</p>	<ul style="list-style-type: none"><li>• <code>http.request</code></li><li>• <code>tls</code></li><li>• <code>tls.handshake.type == 1</code></li><li>• <code>tls.handshake.type == 2</code></li><li>• <code>ssdp</code></li></ul>



Similar to the TCP three-way handshake process, the TLS protocol has its handshake process. The first two steps contain "Client Hello" and "Server Hello" messages. The given filters show the initial hello packets in a capture file. These filters are helpful to spot which IP addresses are involved in the TLS handshake.

- Client Hello: `(http.request or tls.handshake.type == 1) and !(ssdp)`
- Server Hello: `(http.request or tls.handshake.type == 2) and !(ssdp)`

Exercise.pcapng

(http.request or tls.handshake.type == 1) and !(ssdp)

No.	Time	Source	Destination	Protocol	Info
13	0.754105	192.168.1.12	172.217.17.227	TLSv1.3	Client Hello
16	0.755456	192.168.1.12	172.217.17.237	TLSv1.3	Client Hello
53	0.889384	192.168.1.12	172.217.17.196	TLSv1.3	Client Hello
64	0.916063	192.168.1.12	172.217.17.196	TLSv1.3	Client Hello
76	0.950598	192.168.1.12	172.217.17.196	TLSv1.3	Client Hello
266	3.526591	192.168.1.12	172.217.20.74	TLSv1.3	Client Hello
289	3.575830	192.168.1.12	172.217.20.74	TLSv1.3	Client Hello
388	3.731085	192.168.1.12	172.217.20.78	TLSv1.3	Client Hello
572	4.274527	192.168.1.12	216.58.206.194	TLSv1.3	Client Hello
589	4.313172	192.168.1.12	216.58.206.198	TLSv1.3	Client Hello
606	4.330166	192.168.1.12	216.58.214.138	TLSv1.3	Client Hello
894	4.941191	192.168.1.12	172.217.17.99	TLSv1.3	Client Hello
985	5.762164	192.168.1.12	142.250.187.168	TLSv1.3	Client Hello
1	6.889572	192.168.1.12	142.250.187.131	TLSv1.3	Client Hello
1..	12.523681	192.168.1.12	185.47.40.36	TLSv1.3	Client Hello
1..	12.526718	192.168.1.12	185.47.40.36	TLSv1.3	Client Hello
1..	16.927825	192.168.1.12	172.217.169.170	TLSv1.3	Client Hello
1..	17.649620	192.168.1.12	87.238.33.7	TLSv1.2	Client Hello
1..	31.720520	192.168.1.12	87.238.33.7	TLSv1.2	Client Hello

```

Frame 13: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits)
Ethernet II, Src: 00:02:09:98:c7:a8, Dst: 50:78:b3:f3:cd:f4
Internet Protocol Version 4, Src: 192.168.1.12, Dst: 172.217.17.227
Transmission Control Protocol, Src Port: 64512, Dst Port: 443, Seq: 1, Ack: 1, Len: 571
    Transport Layer Security
        - TLSv1.3 Record Layer: Handshake Protocol: Client Hello
            Content Type: Handshake (22)
            Version: TLS 1.0 (0x0301)
            Length: 512
        - Handshake Protocol: Client Hello

```

Exercise.pcapng

(http.request or tls.handshake.type == 2) and !(ssdp)

No.	Time	Source	Destination	Protocol	Info
19	0.813990	172.217.17.237	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
24	0.815416	172.217.17.227	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
77	0.955601	172.217.17.196	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
87	0.975115	172.217.17.196	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
101	1.010213	172.217.17.196	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
298	3.588969	172.217.20.74	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
342	3.637284	172.217.20.74	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
398	3.792255	172.217.20.78	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
626	4.361320	216.58.206.194	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
651	4.398633	216.58.206.198	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
656	4.399291	216.58.214.138	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
901	5.001760	172.217.17.99	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
996	5.821469	142.250.187.168	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
1..	6.959426	142.250.187.131	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
1..	12.608801	185.47.40.36	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec, Application Data
1..	12.624932	185.47.40.36	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec, Application Data
1..	16.988483	172.217.169.170	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
1..	17.739458	87.238.33.7	192.168.1.12	TLSv1.2	Server Hello
1..	31.804500	87.238.33.7	192.168.1.12	TLSv1.2	Server Hello, Change Cipher Spec, Encrypted Handshake

```

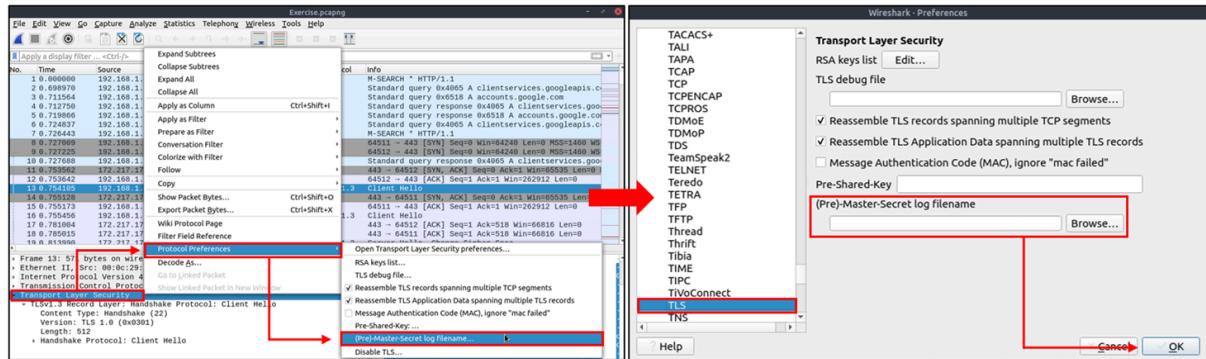
Frame 101: 1484 bytes on wire (11872 bits), 1484 bytes captured (11872 bits)
Ethernet II, Src: 50:78:b3:f3:cd:f4, Dst: 00:02:09:98:c7:a8
Internet Protocol Version 4, Src: 172.217.17.196, Dst: 192.168.1.12
Transmission Control Protocol, Src Port: 443, Dst Port: 64515, Seq: 1, Ack: 1, Len: 1484
    Transport Layer Security
        - TLSv1.3 Record Layer: Handshake Protocol: Server Hello
            Content Type: Handshake (22)
            Version: TLS 1.2 (0x0303)
            Length: 122
        - Handshake Protocol: Server Hello
        - TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
            Content Type: Change Cipher Spec (20)
            Version: TLS 1.2 (0x0303)
            Length: 1
            Change Cipher Spec Message

```

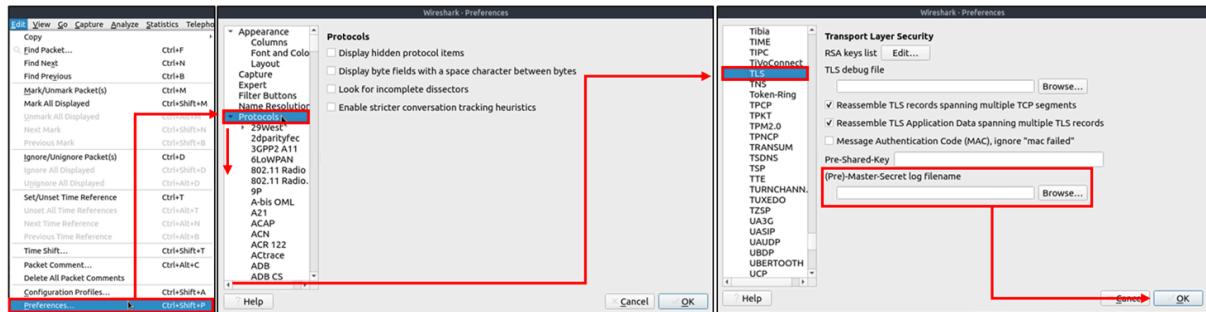
An encryption key log file is a text file that contains unique key pairs to decrypt the encrypted traffic session. These key pairs are automatically created (per session) when a connection is established with an SSL/TLS-enabled webpage. As these processes are all accomplished in the browser, you need to configure your system and use a suitable browser (Chrome and Firefox support this) to save these values as a key log file. To do this, you will need to set up an environment variable and create the `SSLKEYLOGFILE`, and the browser will dump the keys to this file as you browse the web. SSL/TLS key pairs are created per session at the connection time,

so it is important to dump the keys during the traffic capture. Otherwise, it is not possible to create/generate a suitable key log file to decrypt captured traffic. You can use the "right-click" menu or "Edit --> Preferences --> Protocols --> TLS" menu to add/remove key log files.

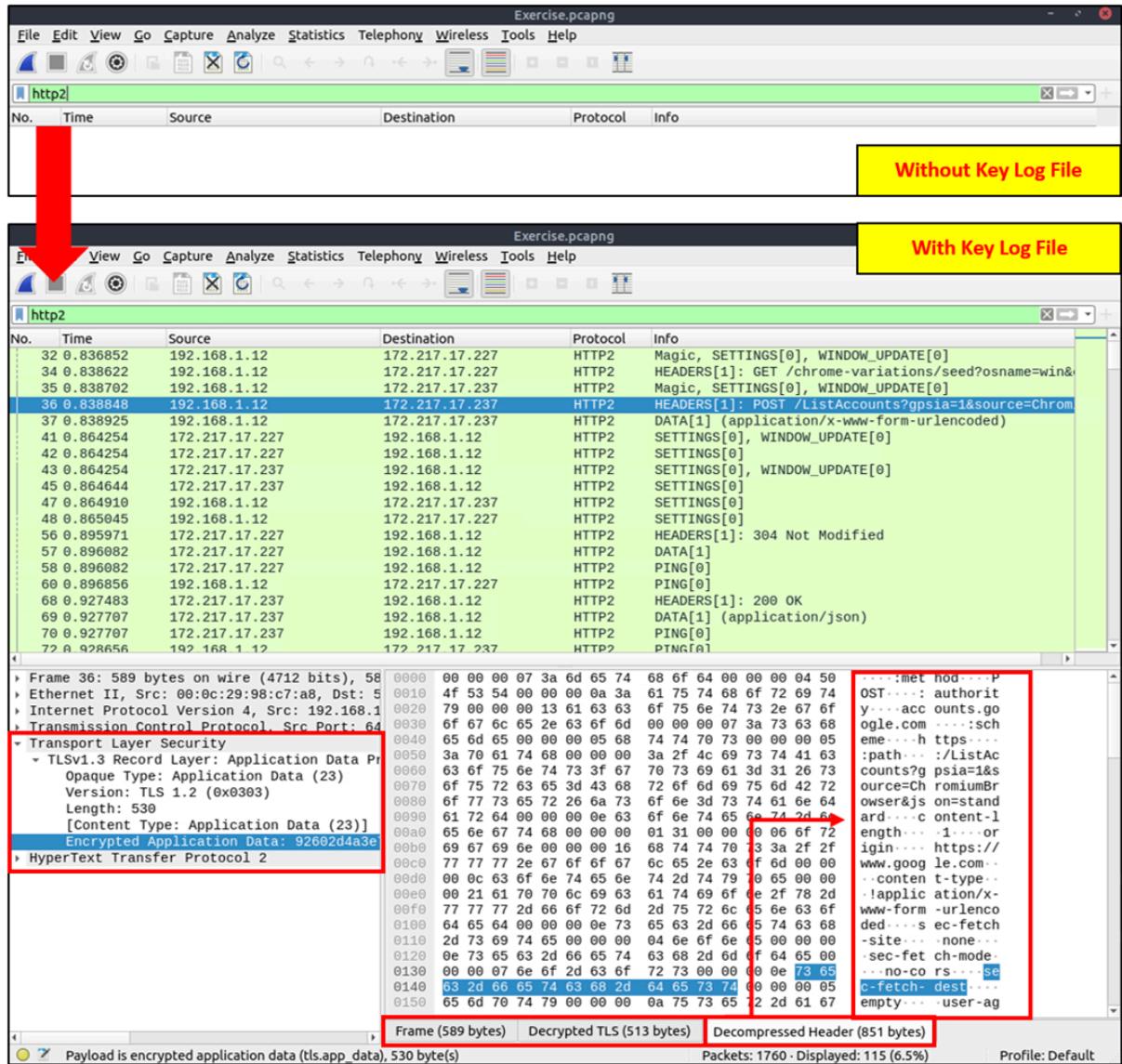
Adding key log files with the "right-click" menu:



Adding key log files with the "Edit --> Preferences --> Protocols --> TLS" menu:



Viewing the traffic with/without the key log files:



The above image shows that the traffic details are visible after using the key log file. Note that the packet details and bytes pane provides the data in different formats for investigation. Decompressed header info and HTTP2 packet details are available after decrypting the traffic. Depending on the packet details, you can also have the following data formats:

- Frame
- Decrypted TLS
- Decompressed Header
- Reassembled TCP
- Reassembled SSL

Detecting suspicious activities in chunked files is easy and a great way to learn how to focus on the details. Now use the exercise files to put your skills into practice against a single capture file and answer the questions below!

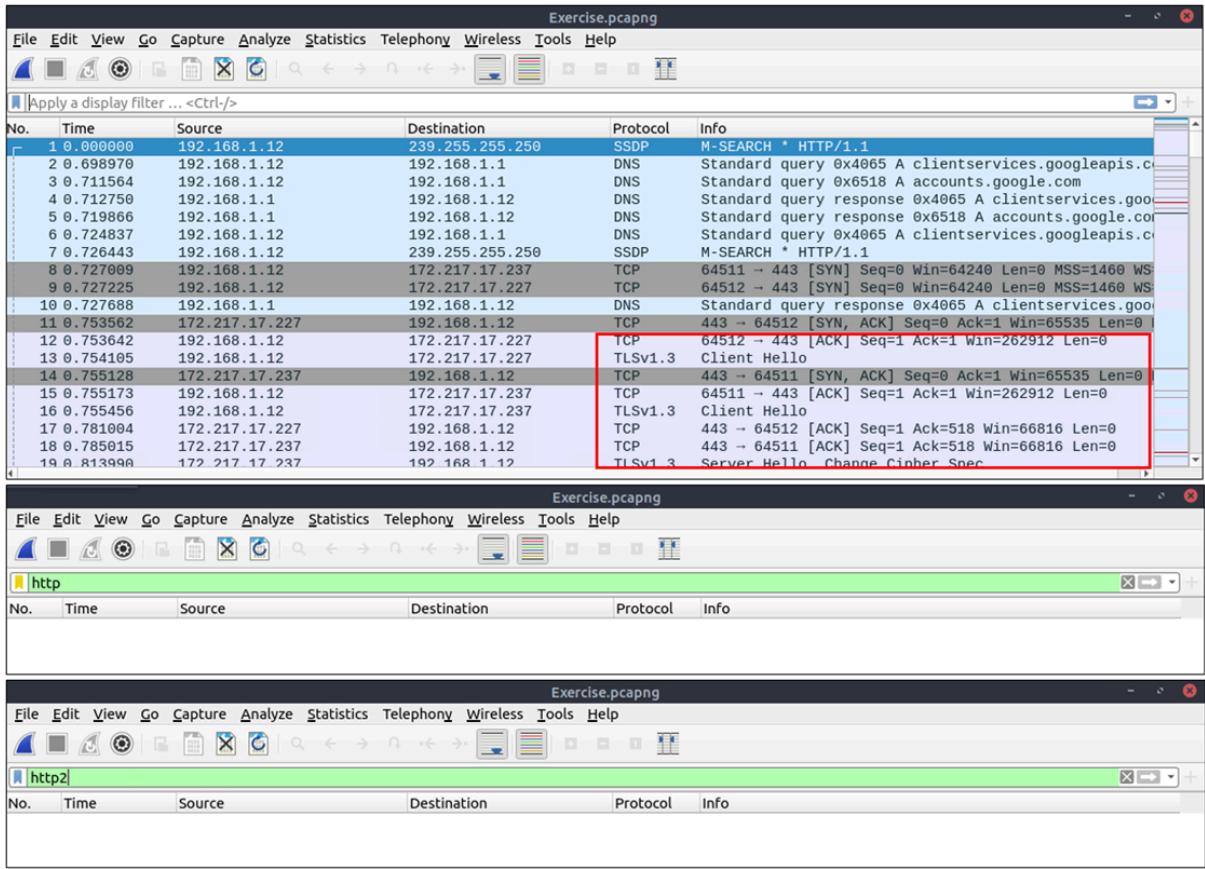
# Decrypting HTTPS Traffic

When investigating web traffic, analysts often run across encrypted traffic. This is caused by using the Hypertext Transfer Protocol Secure (HTTPS) protocol for enhanced security against spoofing, sniffing and intercepting attacks. HTTPS uses TLS protocol to encrypt communications, so it is impossible to decrypt the traffic and view the transferred data without having the encryption/decryption key pairs. As this protocol provides a good level of security for transmitting sensitive data, attackers and malicious websites also use HTTPS. Therefore, a security analyst should know how to use key files to decrypt encrypted traffic and investigate the traffic activity.

The packets will appear in different colours as the HTTP traffic is encrypted. Also, protocol and info details (actual URL address and data returned from the server) will not be fully visible. The first image below shows the HTTP packets encrypted with the TLS protocol. The second and third images demonstrate filtering HTTP packets without using a key log file.

## Additional information for HTTPS :

Notes	Wireshark Filter
<p>"HTTPS Parameters" for grabbing the low-hanging fruits:</p> <ul style="list-style-type: none"><li>• Request: Listing all requests</li><li>• TLS: Global TLS search</li><li>• TLS Client Request</li><li>• TLS Server response</li><li>• Local Simple Service Discovery Protocol (SSDP)</li></ul> <p>Note: SSDP is a network protocol that provides advertisement and discovery of network services.</p>	<ul style="list-style-type: none"><li>• <code>http.request</code></li><li>• <code>tls</code></li><li>• <code>tls.handshake.type == 1</code></li><li>• <code>tls.handshake.type == 2</code></li><li>• <code>ssdp</code></li></ul>



Similar to the TCP three-way handshake process, the TLS protocol has its handshake process. The first two steps contain "Client Hello" and "Server Hello" messages. The given filters show the initial hello packets in a capture file. These filters are helpful to spot which IP addresses are involved in the TLS handshake.

- Client Hello: `(http.request or tls.handshake.type == 1) and !(ssdp)`
- Server Hello: `(http.request or tls.handshake.type == 2) and !(ssdp)`

Exercise.pcapng

(http.request or tls.handshake.type == 1) and !(ssdp)

No.	Time	Source	Destination	Protocol	Info
13	0.754105	192.168.1.12	172.217.17.227	TLSv1.3	Client Hello
16	0.755456	192.168.1.12	172.217.17.237	TLSv1.3	Client Hello
53	0.889384	192.168.1.12	172.217.17.196	TLSv1.3	Client Hello
64	0.916063	192.168.1.12	172.217.17.196	TLSv1.3	Client Hello
76	0.950598	192.168.1.12	172.217.17.196	TLSv1.3	Client Hello
266	3.526591	192.168.1.12	172.217.20.74	TLSv1.3	Client Hello
289	3.575830	192.168.1.12	172.217.20.74	TLSv1.3	Client Hello
388	3.731085	192.168.1.12	172.217.20.78	TLSv1.3	Client Hello
572	4.274527	192.168.1.12	216.58.206.194	TLSv1.3	Client Hello
589	4.313172	192.168.1.12	216.58.206.198	TLSv1.3	Client Hello
606	4.330166	192.168.1.12	216.58.214.138	TLSv1.3	Client Hello
894	4.941191	192.168.1.12	172.217.17.99	TLSv1.3	Client Hello
985	5.762164	192.168.1.12	142.250.187.168	TLSv1.3	Client Hello
1	6.889572	192.168.1.12	142.250.187.131	TLSv1.3	Client Hello
1..	12.523681	192.168.1.12	185.47.40.36	TLSv1.3	Client Hello
1..	12.526718	192.168.1.12	185.47.40.36	TLSv1.3	Client Hello
1..	16.927825	192.168.1.12	172.217.169.170	TLSv1.3	Client Hello
1..	17.649620	192.168.1.12	87.238.33.7	TLSv1.2	Client Hello
1..	31.720520	192.168.1.12	87.238.33.7	TLSv1.2	Client Hello

```

Frame 13: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits)
Ethernet II, Src: 00:02:09:98:c7:a8, Dst: 50:78:b3:f3:cd:f4
Internet Protocol Version 4, Src: 192.168.1.12, Dst: 172.217.17.227
Transmission Control Protocol, Src Port: 64512, Dst Port: 443, Seq: 1, Ack: 1, Len: 571
    Transport Layer Security
        - TLSv1.3 Record Layer: Handshake Protocol: Client Hello
            Content Type: Handshake (22)
            Version: TLS 1.0 (0x0301)
            Length: 512
        - Handshake Protocol: Client Hello

```

Exercise.pcapng

(http.request or tls.handshake.type == 2) and !(ssdp)

No.	Time	Source	Destination	Protocol	Info
19	0.813990	172.217.17.237	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
24	0.815416	172.217.17.227	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
77	0.955601	172.217.17.196	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
87	0.975115	172.217.17.196	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
101	1.010213	172.217.17.196	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
298	3.588969	172.217.20.74	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
342	3.637284	172.217.20.74	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
398	3.792255	172.217.20.78	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
626	4.361320	216.58.206.194	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
651	4.398633	216.58.206.198	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
656	4.399291	216.58.214.138	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
901	5.001760	172.217.17.99	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
996	5.821469	142.250.187.168	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
1..	6.959426	142.250.187.131	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
1..	12.608801	185.47.40.36	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec, Application Data
1..	12.624932	185.47.40.36	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec, Application Data
1..	16.988483	172.217.169.170	192.168.1.12	TLSv1.3	Server Hello, Change Cipher Spec
1..	17.739458	87.238.33.7	192.168.1.12	TLSv1.2	Server Hello
1..	31.804500	87.238.33.7	192.168.1.12	TLSv1.2	Server Hello, Change Cipher Spec, Encrypted Handshake

```

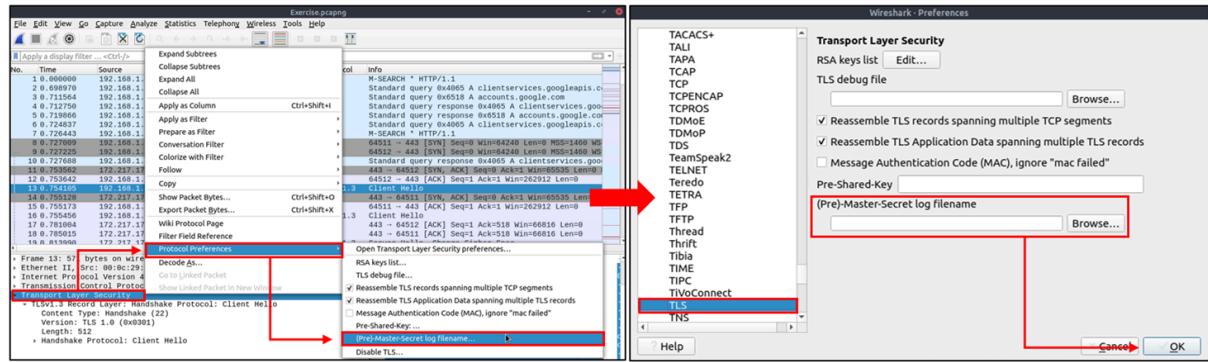
Frame 101: 1484 bytes on wire (11872 bits), 1484 bytes captured (11872 bits)
Ethernet II, Src: 50:78:b3:f3:cd:f4, Dst: 00:02:09:98:c7:a8
Internet Protocol Version 4, Src: 172.217.17.196, Dst: 192.168.1.12
Transmission Control Protocol, Src Port: 443, Dst Port: 64515, Seq: 1, Ack: 1, Len: 1484
    Transport Layer Security
        - TLSv1.3 Record Layer: Handshake Protocol: Server Hello
            Content Type: Handshake (22)
            Version: TLS 1.2 (0x0303)
            Length: 122
        - Handshake Protocol: Server Hello
        - TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
            Content Type: Change Cipher Spec (20)
            Version: TLS 1.2 (0x0303)
            Length: 1
            Change Cipher Spec Message

```

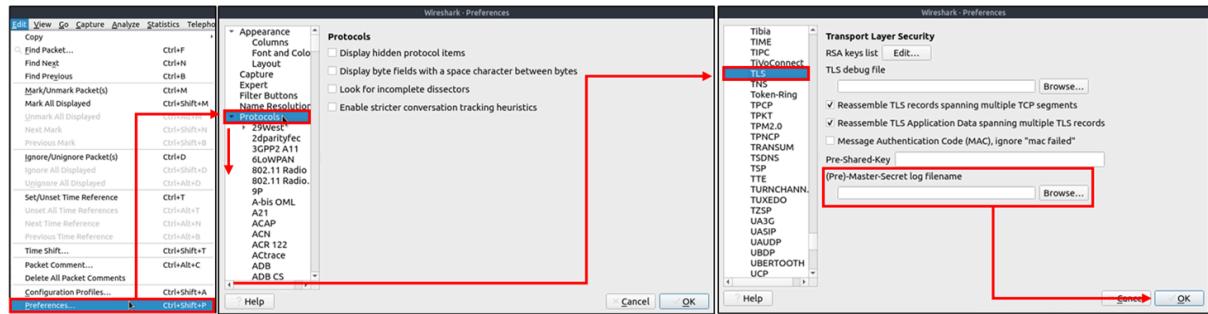
An encryption key log file is a text file that contains unique key pairs to decrypt the encrypted traffic session. These key pairs are automatically created (per session) when a connection is established with an SSL/TLS-enabled webpage. As these processes are all accomplished in the browser, you need to configure your system and use a suitable browser (Chrome and Firefox support this) to save these values as a key log file. To do this, you will need to set up an environment variable and create the `SSLKEYLOGFILE`, and the browser will dump the keys to this file as you browse the web. SSL/TLS key pairs are created per session at the connection time,

so it is important to dump the keys during the traffic capture. Otherwise, it is not possible to create/generate a suitable key log file to decrypt captured traffic. You can use the "right-click" menu or "Edit --> Preferences --> Protocols --> TLS" menu to add/remove key log files.

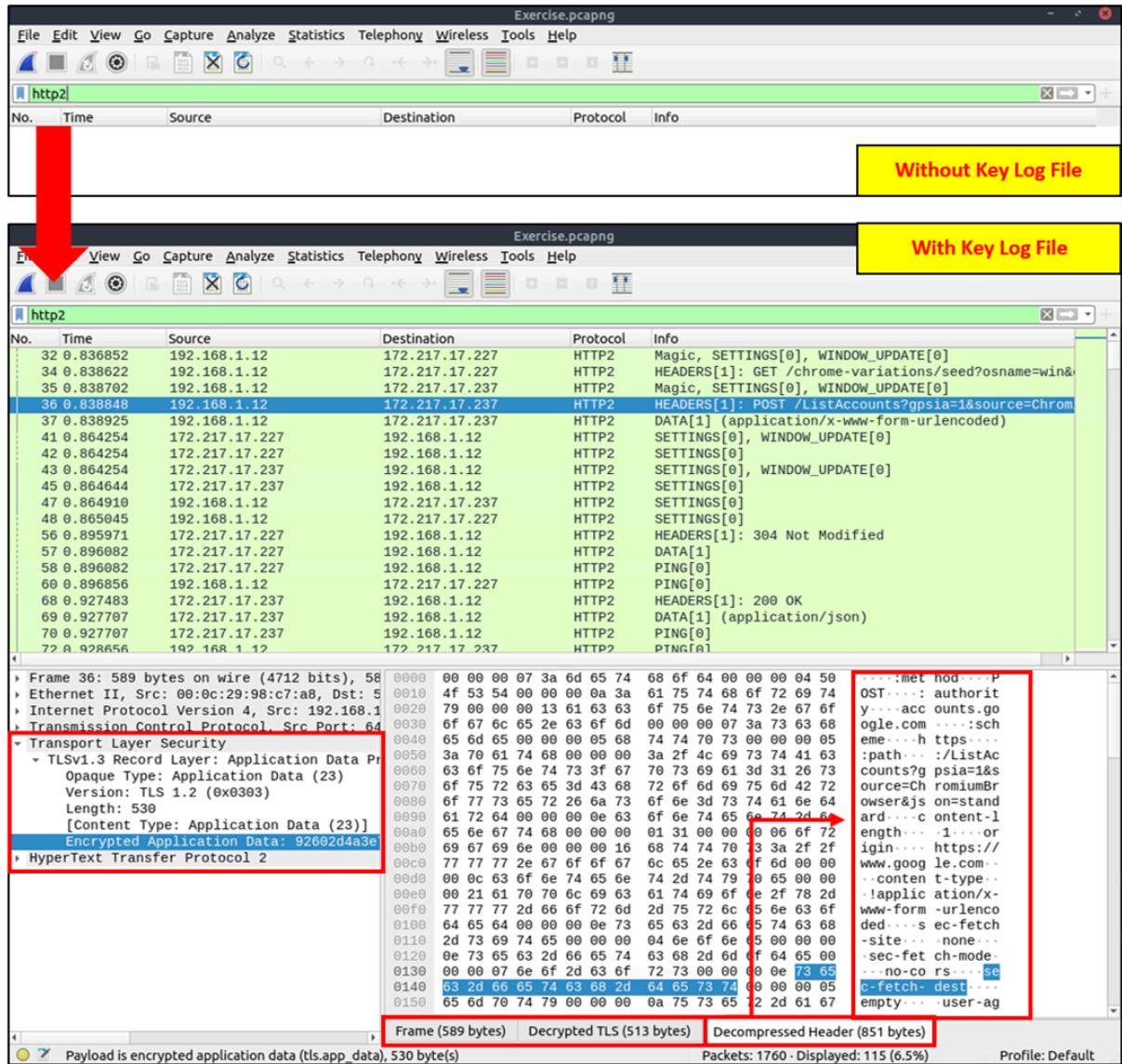
Adding key log files with the "right-click" menu:



Adding key log files with the "Edit --> Preferences --> Protocols --> TLS" menu:



Viewing the traffic with/without the key log files:



The above image shows that the traffic details are visible after using the key log file. Note that the packet details and bytes pane provides the data in different formats for investigation. Decompressed header info and HTTP2 packet details are available after decrypting the traffic. Depending on the packet details, you can also have the following data formats:

- Frame
- Decrypted TLS
- Decompressed Header
- Reassembled TCP
- Reassembled SSL

Detecting suspicious activities in chunked files is easy and a great way to learn how to focus on the details. Now use the exercise files to put your skills into practice against a single capture file and answer the questions below!



