==
Packet Capture with Wireshark
==

Introduction: Packet Capture with Wireshark

In this lab, we explored the fundamentals of using **Wireshark**, a powerful network protocol analyzer, for capturing and analyzing network traffic on a Linux-based system. The goal was to understand how to set up a secure and functional Wireshark environment **without running it as root**, thus mitigating potential security risks associated with elevated privileges.

We began by preparing the system environment, configuring the necessary permissions for dumpcap (Wireshark's packet capture backend), and assigning user access through the wireshark group. The lab also included updating to the latest stable version of Wireshark using the official PPA on Ubuntu.

Once the environment was ready, we proceeded to perform **live packet captures**, apply **display filters**, and analyze key network events such as HTTPS communications and **TLS (Transport Layer Security)** handshakes. Specific filters were used to isolate traffic based on **ports**, **IP addresses**, and **protocol message types**, enabling targeted inspection of encrypted sessions, including the **Client Hello** message used to initiate HTTPS connections.

❗ Risks of using sudo with third-party software like Wireshark

---

Privilege Escalation Risk
If Wireshark or a plugin has a vulnerability, running it as root gives attackers control over your entire system.

Malicious Packet Exploits
Some attackers craft malicious packets. If Wireshark processes them with root privileges, it can be used as an attack vector.

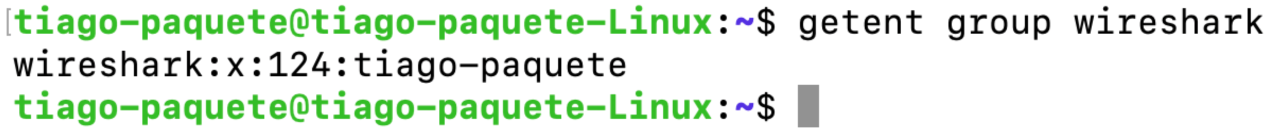
Potential for System Damage
Bugs in Wireshark might accidentally alter or delete system files if it’s running as root.

Audit and Compliance Issues
In professional environments, running GUI apps as root may violate security policies or audit requirements.

Preparing the environment

Step 1: Check if the wireshark group exists

getent group wireshark

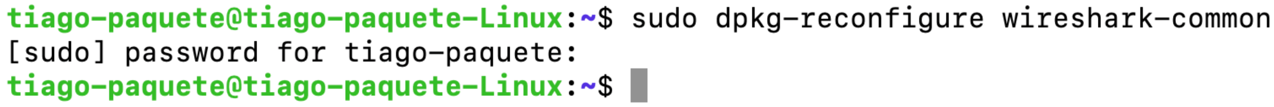


```
[tiago-paquete@tiago-paquete-Linux:~$ getent group wireshark
wireshark:x:124:tiago-paquete
tiago-paquete@tiago-paquete-Linux:~$ 
```

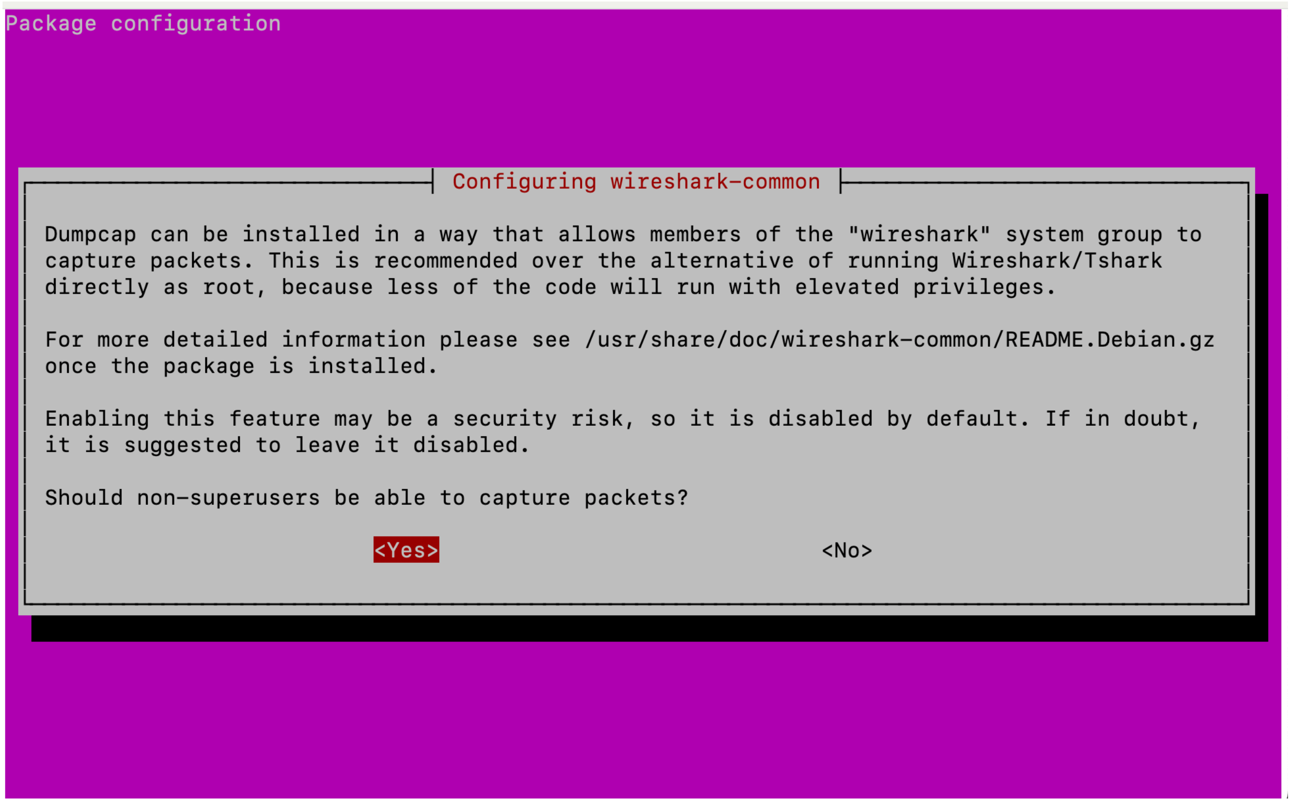
Step 2: Configure dumpcap permissions (done during install, but verify)

Wireshark uses dumpcap to capture packets. This binary needs special permissions.

sudo dpkg-reconfigure wireshark-common



```
tiago-paquete@tiago-paquete-Linux:~$ sudo dpkg-reconfigure wireshark-common
[sudo] password for tiago-paquete:
tiago-paquete@tiago-paquete-Linux:~$ 
```

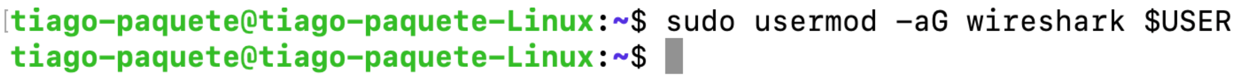


```
Package configuration

                           Configuring wireshark-common

 Dumpcap can be installed in a way that allows members of the "wireshark" system group to
 capture packets. This is recommended over the alternative of running Wireshark/Tshark
 directly as root, because less of the code will run with elevated privileges.

 For more detailed information please see /usr/share/doc/wireshark-common/README.Debian.gz
 once the package is installed.

 Enabling this feature may be a security risk, so it is disabled by default. If in doubt,
 it is suggested to leave it disabled.

 Should non-superusers be able to capture packets?

                         <Yes>                              <No>
```

Step 3: Add the user to the wireshark group

sudo usermod -aG wireshark $USER

$USER refers to the currently logged-in user.



```
tiago-paquete@tiago-paquete-Linux:~$ sudo usermod -aG wireshark $USER
tiago-paquete@tiago-paquete-Linux:~$ 
```

Step 4: Reboot or re-login to apply group changes

Group membership only applies **after a new login session**.

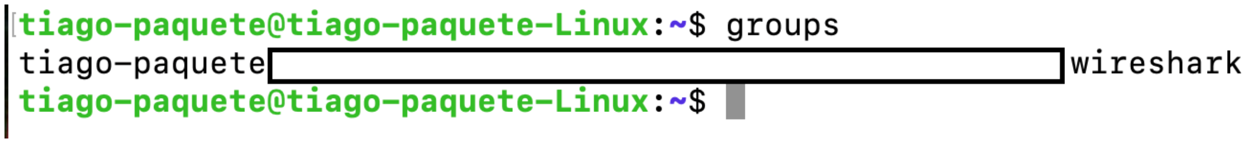
Log out and log back in
OR
Reboot the system:

reboot

Step 5: Confirm you're in the wireshark group

groups

Check that wireshark is listed.

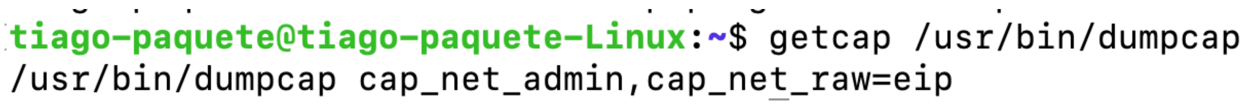


```
tiago-paquete@tiago-paquete-Linux:~$ groups
tiago-paquete [redacted] wireshark
tiago-paquete@tiago-paquete-Linux:~$ 
```

Step 6 (Optional): Check dumpcap permissions

You can verify dumpcap is safe and not running as root:

getcap /usr/bin/dumpcap



```
tiago-paquete@tiago-paquete-Linux:~$ getcap /usr/bin/dumpcap
/usr/bin/dumpcap cap_net_admin,cap_net_raw=eip
```

/usr/bin/dumpcap: This is the path to the dumpcap binary — the backend tool used by Wireshark to capture packets.

cap_net_admin

This Linux **capability** allows the process to:

- Configure network interfaces
- Enable promiscuous mode
- Use advanced networking operations

Needed to **see and capture traffic properly**.

cap_net_raw

This capability allows:

- Sending and receiving **raw packets** (e.g., low-level socket access)
- Bypassing some normal networking restrictions

Needed for **raw packet capture**.

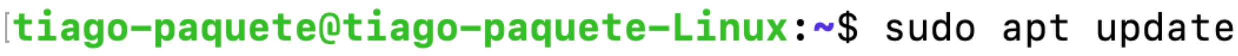
=eip (flags)

These are capability flags:

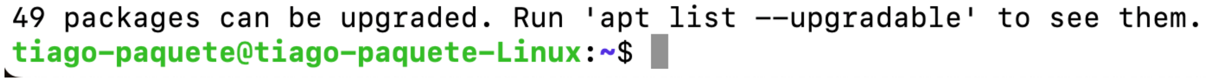
Flag	Meaning
e	**Effective** — active during runtime
i	**Inheritable** — passes to child processes
p	**Permitted** — allowed to use these caps

To get the latest stable Wireshark on Ubuntu

You can use this command to add the official PPA (Personal Package Archive):



```
tiago-paquete@tiago-paquete-Linux:~$ sudo apt update
```



```
49 packages can be upgraded. Run 'apt list --upgradable' to see them.
tiago-paquete@tiago-paquete-Linux:~$
```

You'll get a list like:

wireshark/stable 4.2.3-1ubuntu1 amd64 [upgradable from 4.0.1-1]
libfoo/stable 1.2.3-4ubuntu1 amd64 [upgradable from 1.2.3-1]
...

To update all packages safely:

sudo apt update && sudo apt upgrade -y

apt update: Refreshes the list of available packages.
apt upgrade -y: Upgrades the listed packages automatically (-y answers "yes" to the prompts).

Optional: Full upgrade
If you want to handle dependency changes too (e.g., kernel or library version jumps), use:

sudo apt full-upgrade

But normally, sudo apt upgrade is fine for routine updates.

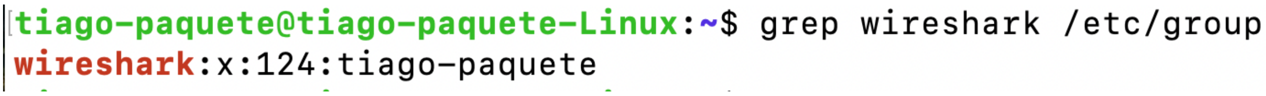
Add User to Wireshark’s group

add user to wireshark group
sudo usermod -aG wireshark $USER

sudo: Runs the command with superuser (root) privileges.
usermod: A Linux command used to modify a user account.
-aG: These are options for the usermod command:
-a: Append the user to the group(s) (used with -G).
-G: Specifies the group(s) to which the user will be added.
wireshark: The group name you're adding the user to.
$USER: This is an environment variable that refers to the currently logged-in user.

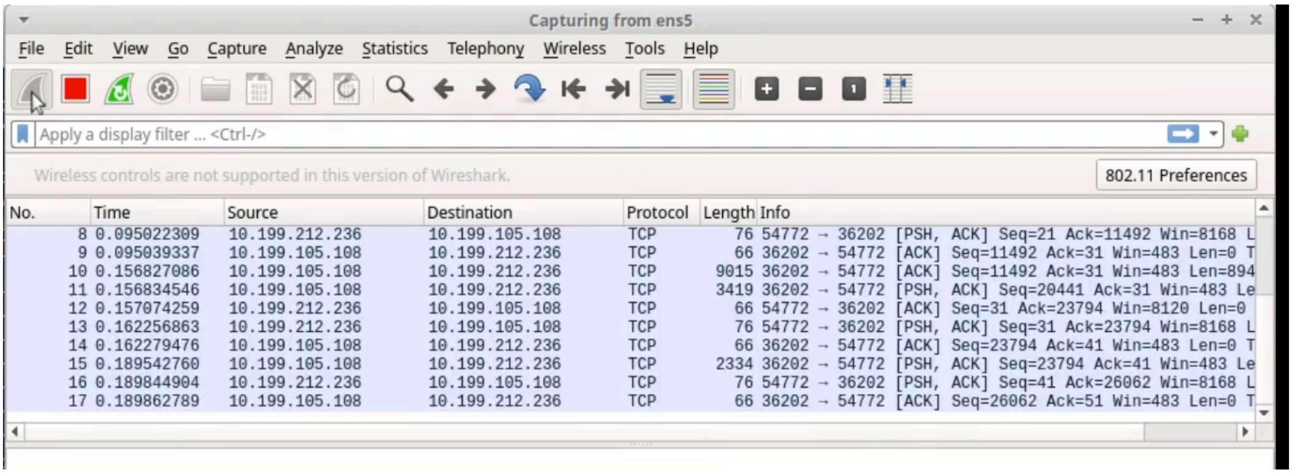
Linux:~$ grep webdev /etc/group

Confirms that alice is in the webdev group.



```
tiago-paquete@tiago-paquete-Linux:~$ grep wireshark /etc/group
wireshark:x:124:tiago-paquete
```

Capture Packets with Wireshark

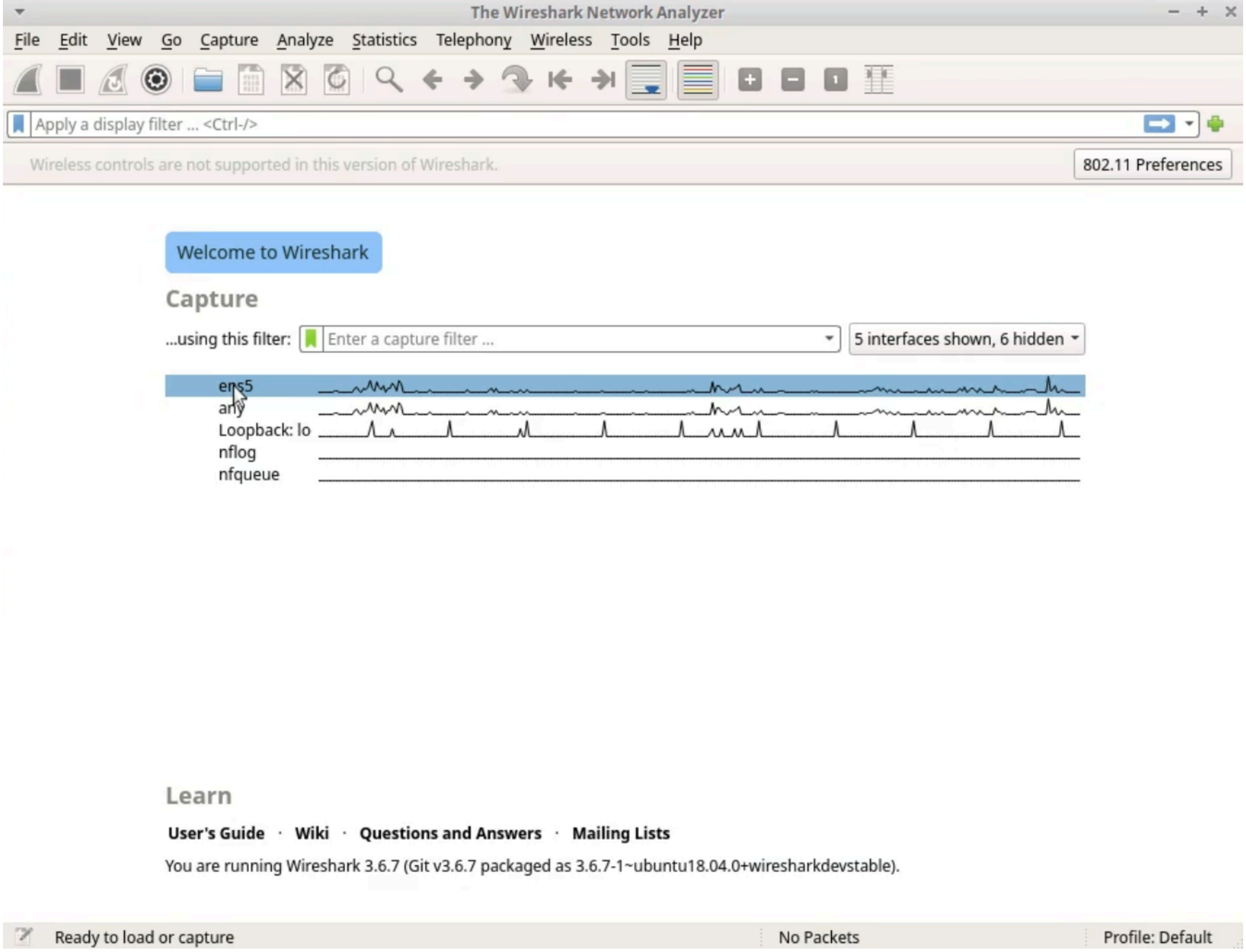

Capturing from ens5
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
802.11 Preferences

Wireless controls are not supported in this version of Wireshark.

```
No.   Time          Source           Destination      Protocol Length Info
    8 0.095022309   10.199.212.236   10.199.105.108   TCP        76 54772 → 36202 [PSH, ACK] Seq=21 Ack=11492 Win=8168 L
    9 0.095039337   10.199.105.108   10.199.212.236   TCP        66 36202 → 54772 [ACK] Seq=11492 Ack=31 Win=483 Len=0 T
   10 0.156827086   10.199.105.108   10.199.212.236   TCP      9015 36202 → 54772 [ACK] Seq=11492 Ack=31 Win=483 Len=894
   11 0.156834546   10.199.105.108   10.199.212.236   TCP      3419 36202 → 54772 [PSH, ACK] Seq=20441 Ack=31 Win=483 Le
   12 0.157074259   10.199.212.236   10.199.105.108   TCP        66 54772 → 36202 [ACK] Seq=31 Ack=23794 Win=8120 Len=0
   13 0.162256863   10.199.212.236   10.199.105.108   TCP        76 54772 → 36202 [PSH, ACK] Seq=31 Ack=23794 Win=8168 L
   14 0.162279476   10.199.105.108   10.199.212.236   TCP        66 36202 → 54772 [ACK] Seq=23794 Ack=41 Win=483 Len=0 T
   15 0.189542760   10.199.105.108   10.199.212.236   TCP      2334 36202 → 54772 [PSH, ACK] Seq=23794 Ack=41 Win=483 Le
   16 0.189844904   10.199.212.236   10.199.105.108   TCP        76 54772 → 36202 [PSH, ACK] Seq=41 Ack=26062 Win=8168 L
   17 0.189862789   10.199.105.108   10.199.212.236   TCP        66 36202 → 54772 [ACK] Seq=26062 Ack=51 Win=483 Len=0 T
```

Start Wireshark:

```
wireshark
```


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
802.11 Preferences
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
5 interfaces shown, 6 hidden
ens5
any
Loopback: lo
nflog
nfqueue
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists

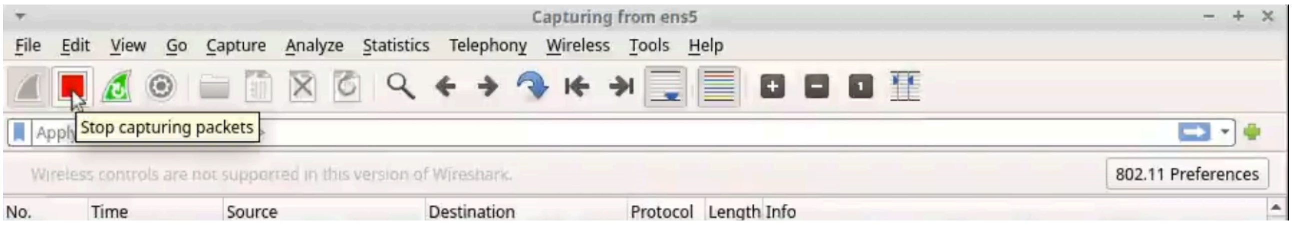
Wireless controls are not supported in this version of Wireshark.

You are running Wireshark 3.6.7 (Git v3.6.7 packaged as 3.6.7-1~ubuntu18.04.0+wiresharkdevstable).

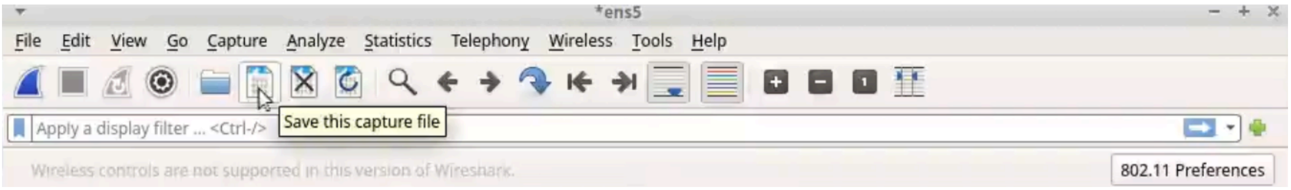
Ready to load or capture | No Packets | Profile: Default

Choose an interface and capture packets:

Stop capturing packets:


Capturing from ens5
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Stop capturing packets
Apply
Wireless controls are not supported in this version of Wireshark.
802.11 Preferences
No. Time Source Destination Protocol Length Info

Save the captured packets to a file:

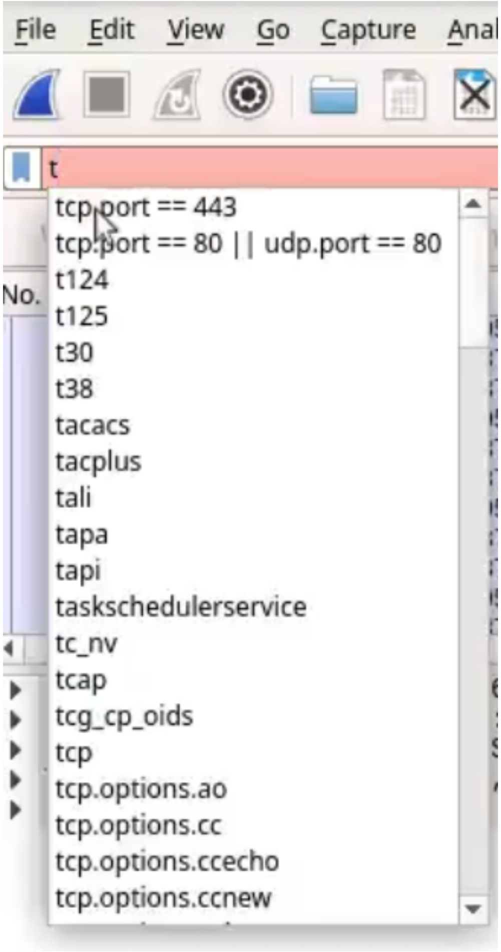

*ens5
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Save this capture file
Wireless controls are not supported in this version of Wireshark.
802.11 Preferences

Use a display filter to detect HTTPS packets

tcp.port == 443

Filter Meaning
tcp: You're filtering **TCP packets** only (not UDP or ICMP).
.port: Applies to **both** tcp.srcport (source) and tcp.dstport (destination).
== 443: Show packets where **either side** is using port 443.

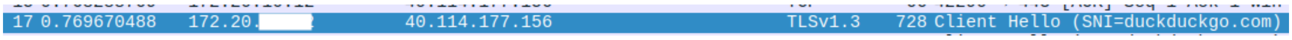
So, this includes:
Outgoing HTTPS requests from your system to a server
Incoming HTTPS responses from servers



t

tcp.port == 443
tcp.port == 80 || udp.port == 80
t124
t125
t30
t38
tacacs
tacplus
tali
tapa
tapi
taskschedulerservice
tc_nv
tcap
tcg_cp_oids
tcp
tcp.options.ao
tcp.options.cc
tcp.options.ccecho
tcp.options.ccnew

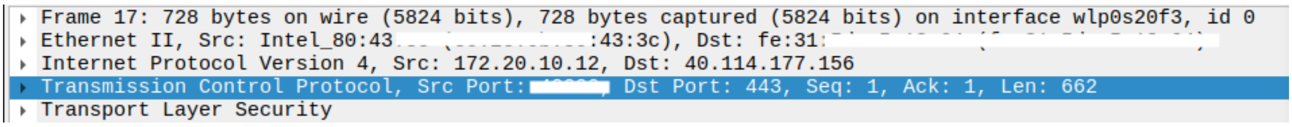
Client Hello



```
17 0.769670488   172.20.[redacted]         40.114.177.156          TLSv1.3    728 Client Hello (SNI=duckduckgo.com)
```

Column	Value	Description
No.	17	This is the **packet number** in the capture file
Time	0.769670488	Time in **seconds since capture started**
Source	172.20.10.12	IP address of the **source** (local machine)
Destination	40.114.177.156	IP address of the **destination** (external server)
Protocol	TLSv1.3	The protocol used — **Transport Layer Security (version 1.3)**
Length	728	Total **packet length in bytes**
Info	Client Hello (SNI=duckduckgo.com)	Summary of the packet: a **TLS handshake Client Hello** with the **SNI (Server Name Indication)** set to duckduckgo.com

This packet represents the beginning of a **secure HTTPS connection** to `https://duckduckgo.com`.



```
▸ Frame 17: 728 bytes on wire (5824 bits), 728 bytes captured (5824 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: Intel_80:43[redacted]:43:3c), Dst: fe:31:[redacted]
▸ Internet Protocol Version 4, Src: 172.20.10.12, Dst: 40.114.177.156
▸ Transmission Control Protocol, Src Port:[redacted] Dst Port: 443, Seq: 1, Ack: 1, Len: 662
▸ Transport Layer Security
```

Frame 17 Summary

Field	Value
Frame	728 bytes on wire / 728 bytes captured
Interface	wlp0s20f3 (your Wi-Fi adapter)

This line means the packet was:
728 bytes long
Fully captured

Captured on your **Wi-Fi** interface (wlp0s20f3)

Ethernet II (Layer 2)

Field	Value
Source MAC	00:2e:0b:00:00:00
Destination MAC	Fe:00:5d:00:12:00

This is the **data link layer**, showing local MAC addresses (used in LAN communication).

IP Header (Layer 3)

Field	**Value**
Source IP	172.20.00.00
Destination IP	40.114.177.156

This identifies the packet's **origin and destination on the internet**:
Your machine → External server (DuckDuckGo's IP)

TCP Header (Layer 4)

Field	Value
Source Port	42290
Destination Port	443
Seq (Sequence Number)	1
Ack (Acknowledgment Number)	1
Len (TCP payload length)	662

This tells us:
This is part of a **TLS session over TCP**
Your machine used **ephemeral port 42290**
The server listens on **port 443** (HTTPS)
The TLS payload is **662 bytes**, which matches the Client Hello size

TLS (Layer 7)
The **"Client Hello"** packet containing:
TLS version: 1.3
SNI (Server Name Indication): duckduckgo.com
Cipher suites, random, extensions, etc.

TL;DR – What does this packet show?
Your machine (172.00.00.00) is initiating a **secure HTTPS connection** to 40.114.177.156 (DuckDuckGo).

It sends a **TLS Client Hello** using port 443 with SNI = duckduckgo.com.

This is the **first step in setting up encryption** with the server.

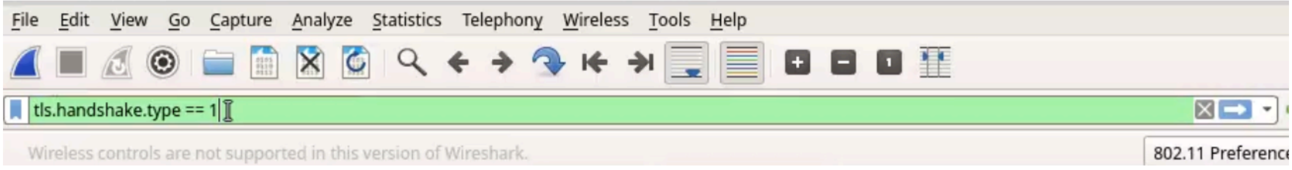
— —

Detect the IP address using a display filter

— —

Capture the handshake:

Pls.handshake.type == 1



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tls.handshake.type == 1
```

Wireless controls are not supported in this version of Wireshark.
802.11 Preferenc

This Wireshark display filter shows only packets where the **TLS handshake message type is 1** — which corresponds to a **Client Hello**.

Breakdown:

Filter Component	Meaning
tls	Filters for packets containing **TLS protocol** (Transport Layer Security)
handshake.type	Filters for **TLS handshake message types**
==1	Shows only ==1messages of **type 1**, which is the **Client Hello**

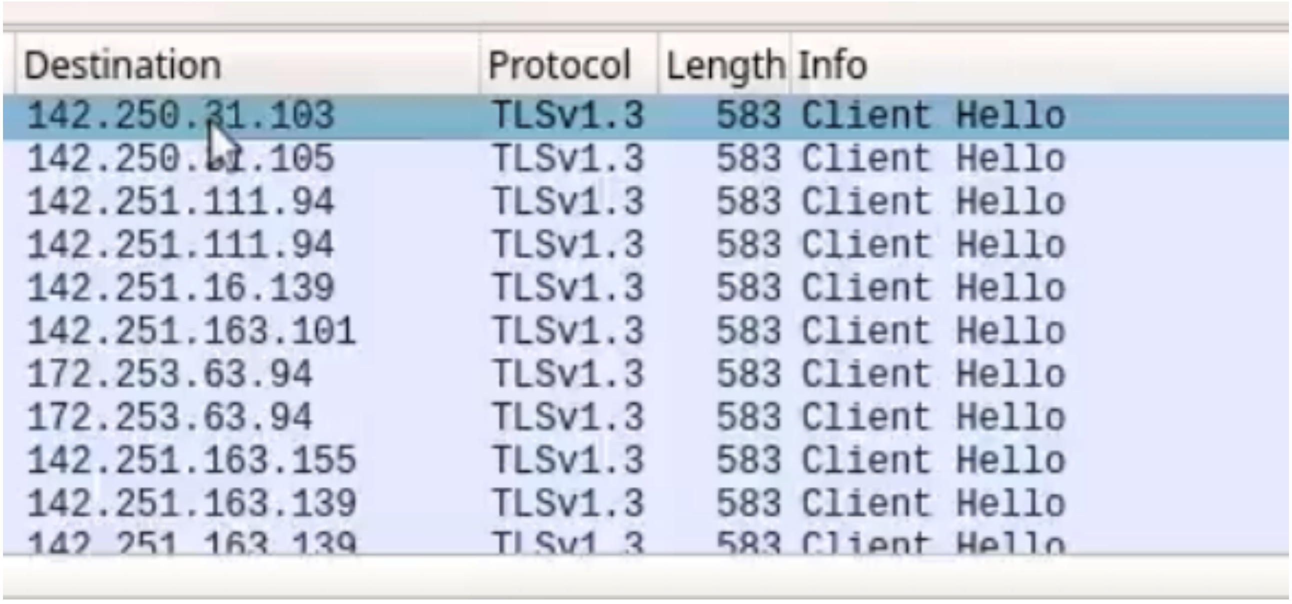
LS Handshake Message Types (Common ones)

Value	Handshake Type
1	Client Hello
2	Server Hello
11	Certificate
12	Server Key Exchange
13	Certificate Request
14	Server Hello Done
16	Client Key Exchange
20	Finished

So tls.handshake.type == 1 means:

"Show me only packets that contain a **Client Hello** handshake message."

This is useful when you're analyzing:
Which domains or servers the client is trying to contact (via SNI)
What TLS version and cipher suites the client supports
Troubleshooting the beginning of a TLS connection



Destination	Protocol	Length	Info
142.250.31.103	TLSv1.3	583	Client Hello
142.250.31.105	TLSv1.3	583	Client Hello
142.251.111.94	TLSv1.3	583	Client Hello
142.251.111.94	TLSv1.3	583	Client Hello
142.251.16.139	TLSv1.3	583	Client Hello
142.251.163.101	TLSv1.3	583	Client Hello
172.253.63.94	TLSv1.3	583	Client Hello
172.253.63.94	TLSv1.3	583	Client Hello
142.251.163.155	TLSv1.3	583	Client Hello
142.251.163.139	TLSv1.3	583	Client Hello
142.251.163.139	TLSv1.3	583	Client Hello

Filter using the IP address:

ip.addr == 142.250.31.103

What does it do?

This filter displays all packets where the IP address 142.250.31.103 appears as either:

The **source IP** OR
The **destination IP**

Breaking it down:

Component	Meaning
ip.addr	Applies to **both ip.src and ip.dst**
	Logical comparison operator ==
142.250.31.103	The target IP address to match

So it catches **traffic to or from** that IP, regardless of direction.

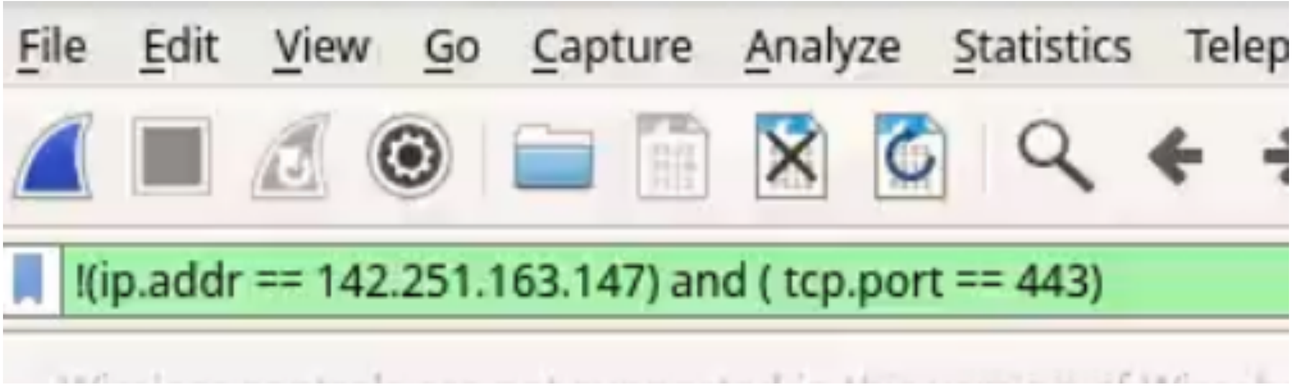
M**ore specific filters:**

Filter	Result
ip.src == 142.250.31.103	Only packets **coming from** that IP
ip.dst == 142.250.31.103	Only packets **going to** that IP
ip.addr == 142.250.31.103 && tcp.port == 443	Google HTTPS traffic only

Locate all HTTPS packets not containing certain IP addresses

Capture all packets on a specific port except a particular IP address:

!(ip.addr == 142.251.163.147) and (tcp.port == 443)



!(ip.addr == 142.251.163.147) and (tcp.port == 443)

This filter tells Wireshark:
"Show me **all TCP packets on port 443** (HTTPS) — but **exclude** those where either the source or destination IP address is 142.251.163.147."

Breaking it down:

Part	Meaning
ip.addr == 142.251.163.147	Matches packets **to or from** that IP
!(...)	**Negates** that condition (i.e., exclude it)
tcp.port == 443	Matches packets where the **source or destination TCP port** is **443** (HTTPS)
and	Combines both conditions — both must be true

This filter will:
Include only **HTTPS traffic** (port 443)
Exclude anything involving IP 142.251.163.147 (which is a Google server IP)

Examples of matching packets:

Source IP	Destination IP	TCP Port	Match?
192.168.1.5	142.250.31.103	443	✅ Yes
142.251.163.147	192.168.1.5	443	❌ No
192.168.1.5	142.251.163.147	443	❌ No
192.168.1.5	1.1.1.1	443	✅ Yes

Tip:
If you want to filter **only outgoing** HTTPS requests and exclude that IP, use:

!(ip.addr == 142.251.163.147) and tcp.dstport == 443

——————————————————————————————————————

!(ip.addr == 142.251.163.147) and (tcp.port == 443 or tcp.port == 80)

What does this filter do?

This shows all TCP packets where:
The IP address is NOT 142.251.163.147 (neither source nor destination)

AND

The TCP port is either 443 (HTTPS) or 80 (HTTP)

Filter Logic Breakdown:

Part	Meaning
ip.addr == 142.251.163.147	Matches packets **to or from** that IP
!()	**Negates** that — excludes those packets
tcp.port == 443 or tcp.port == 80	Matches HTTP or HTTPS traffic
and	Both conditions must be true to display the packet

Packets you will see:

Source IP	Destination IP	Port	Match?
192.168.1.10	142.251.163.147	443	❌ No (IP excluded)
172.16.0.5	1.1.1.1	443	✅ Yes
10.0.0.2	10.0.0.3	80	✅ Yes
142.251.163.147	192.168.1.10	80	❌ No (IP excluded)
8.8.8.8	1.1.1.1	53	❌ No (wrong port)

This filter is perfect for:
Analyzing **HTTP/HTTPS traffic** but **excluding a specific IP**
Checking general web traffic while ignoring a known IP (e.g., Google or CDN)

Want a more specific version?
To only exclude it as the destination (outgoing requests), use:

!(ip.dst == 142.251.163.147) and (tcp.dstport == 443 or tcp.dstport == 80)

———————————————————————————————————

Summary

———————————————————————————————————

This lab provided hands-on experience in:

Securing Wireshark usage by avoiding root execution and configuring user/group permissions properly.

Capturing and saving packet data via the Wireshark interface.

Using powerful display filters such as:
tcp.port == 443 to isolate HTTPS traffic.
tls.handshake.type == 1 to identify TLS Client Hello messages.
!(ip.addr == X.X.X.X) and tcp.port == 443 to exclude specific IPs from capture.

Analyzing multi-layered protocol information, including Ethernet (Layer 2), IP (Layer 3), TCP (Layer 4), and TLS (Layer 7) headers.

Understanding how to trace and interpret the beginning of encrypted web sessions using the Client Hello packet, Server Name Indication (SNI), and cipher suite negotiation.