



INFORMATION  
SECURITY

# **PROJECT REPORT ON WIRESHARK TOOL**

*TRANSPORT LAYER  
(OSI LAYER 4)*

PREPARED BY:  
SABAHATSAJJAD-010  
RANA JUNAID-017  
NOSHABANOREEN-040  
AROOJFATIMA-046

## Table of Contents

<b>1. Introduction:</b>	3
<b>1.1. What is Wireshark?</b>	3
<b>1.2. Main Features:</b>	3
<b>1.3. Why is Wireshark Used?</b>	3
<b>1.4. Supported Platforms:</b>	3
<b>2. Installation steps:</b>	3
<b>3. Tool Use Cases:</b>	10
<b>3.1. Http and Https Difference:</b>	10
<b>3.2. Https Traffic Decryption(fetch session ID and Decrypt user Session):</b>	18
<b>3.3. TCP Stream Analysis for SYN Flood Detection and UDP Analysis in https:</b>	27
<b>4. OSI Layer Mapping (Focused on Transport Layer):</b>	30
<b>5. Mapping to MITRE ATT&amp;CK Framework):</b>	30
<b>6. Mitigation Strategies:</b>	31
<b>7. References:</b>	33

# User Manual

## 1. Introduction:

### 1.1. What is Wireshark?

- Wireshark is a **free and open-source network protocol analyzer**.
- It is used to **capture and analyze** data packets on a computer network.
- Originally known as **Ethereal**, it was renamed to Wireshark in 2006.

### 1.2. Main Features:

- Captures real-time **network traffic**.
- Shows details of each **data packet** (like source, destination, protocol, etc.).
- Supports **hundreds of network protocols** (e.g. TCP, UDP, HTTP, DNS).
- Allows **filters** to search and analyze specific traffic.
- Provides **graphical and detailed** views of the captured data.

### 1.3. Why is Wireshark Used?

- To **troubleshoot network problems**.
- For **cybersecurity analysis** (like checking for attacks or suspicious activity).
- For **learning and teaching** how network protocols work.
- To **optimize network performance**.

### 1.4. Supported Platforms:

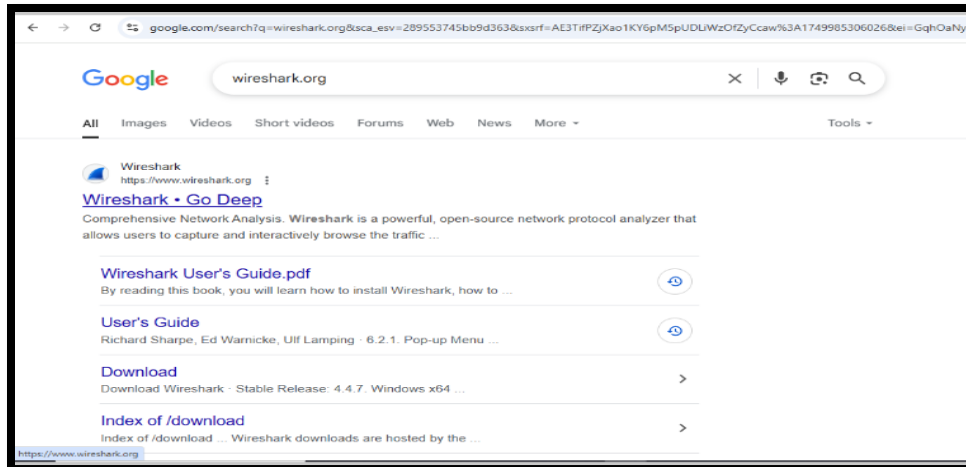
Works on multiple operating systems:

- **Windows**
- **Linux**
- **macOS**

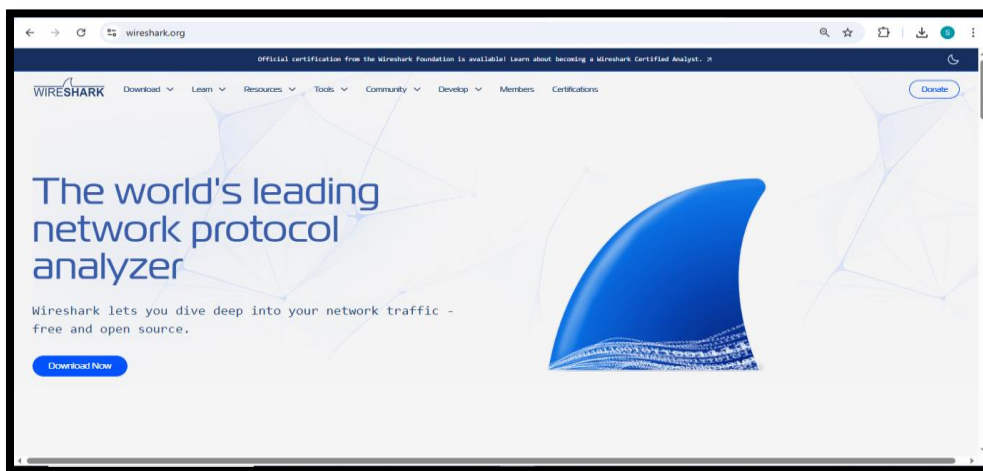
## 2. Installation steps:

In this section, the step-by-step procedure to install Wireshark on a Windows system is explained. It guides the user through downloading the software from the official website, running the installer, and completing the setup process. These instructions help beginners easily install the tool and get started with network packet analysis.

- 1) Search **“Wireshark.org”** on the Google Chrome and click the first wireshark.org website that appears at the top.



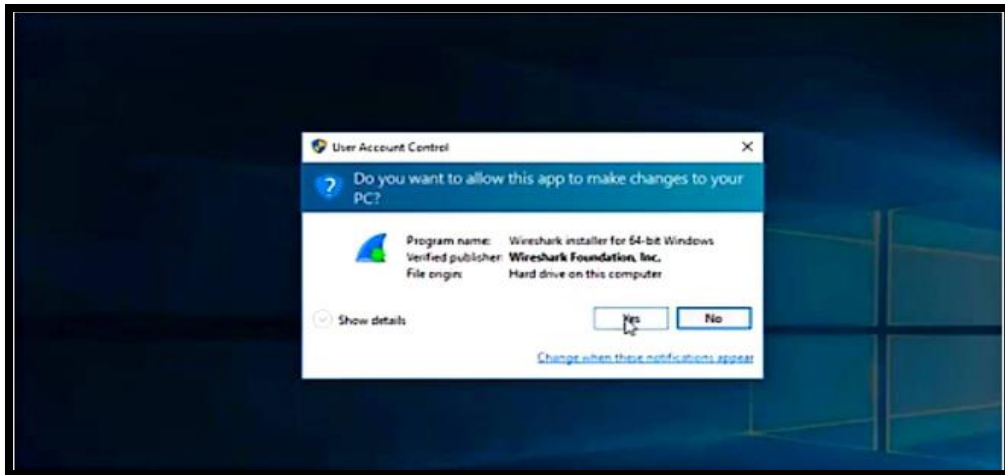
- 2) When you click the website, this interface will be displayed. Click on **“Download”** option and the file will start downloading.



- 3) Then select the window installer according to your system. I installed **“Windows Installer (64 bit)”**.



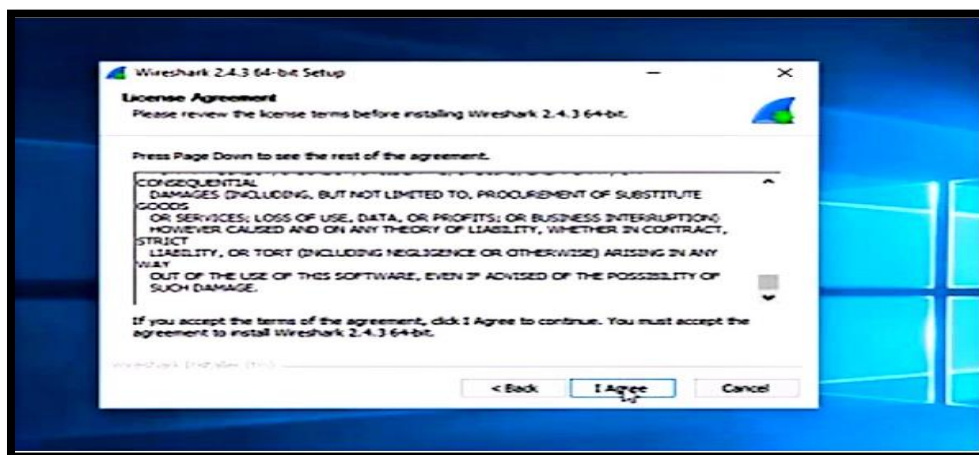
- 4) A **“User Control Menu”** will appear and then click on **“Yes”** to allow the app to make changes in your PC.



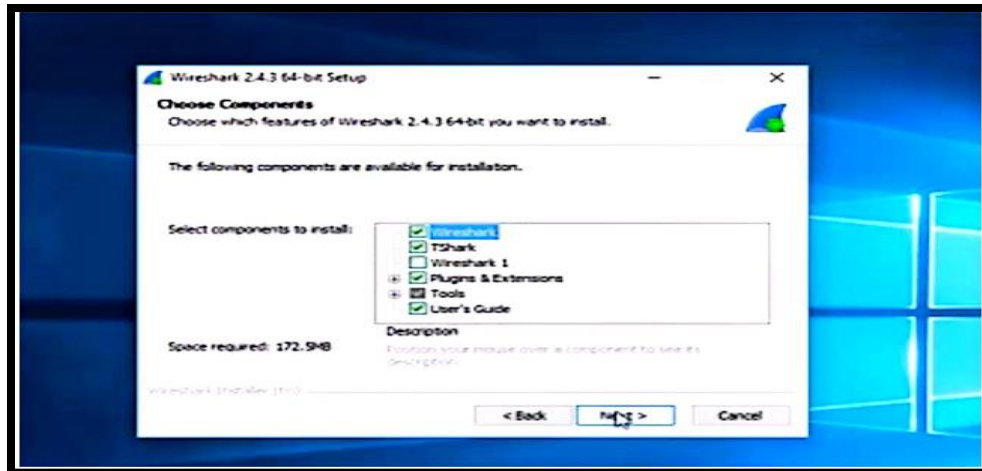
5) Click on the “Next” to install complete wireshark setup.



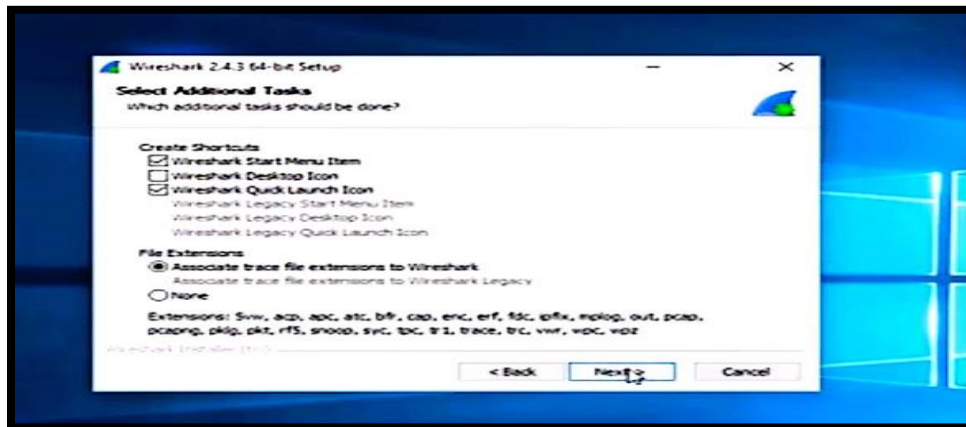
6) Click “I Agree” to ensure that you agree with their guideline and policies.



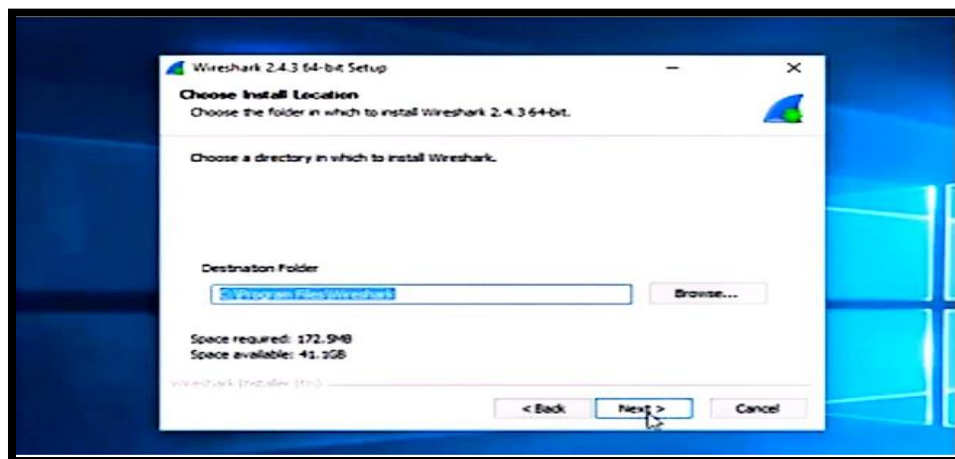
7) Choose these components and click “Next”.



8) Select these additional tasks and click “Next”.

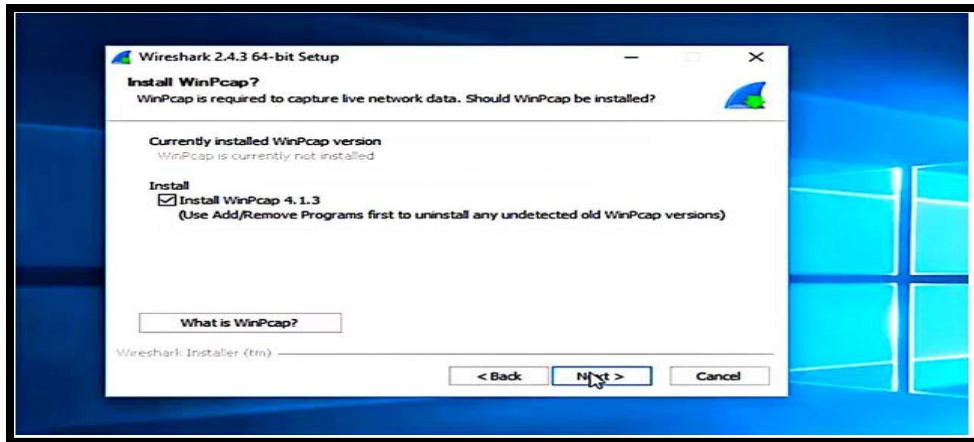


9) Select the Location where you want to download this setup and then click “Next”.

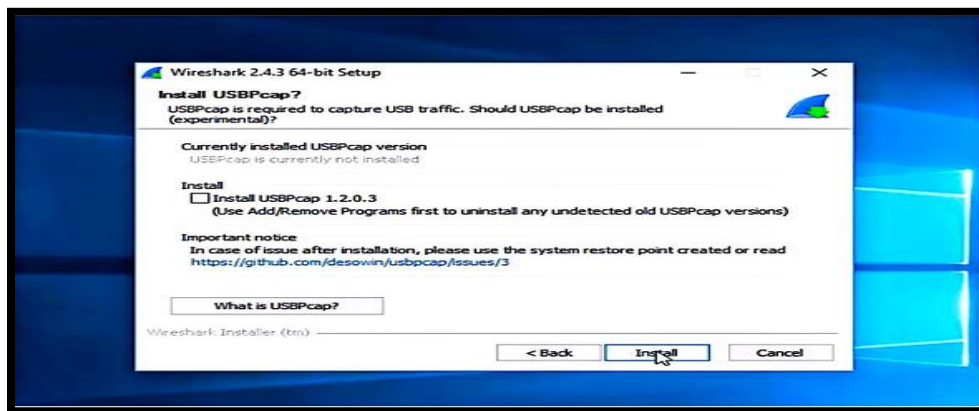


10) Then Install “WinPcap” to capture the live data and click “Next”.

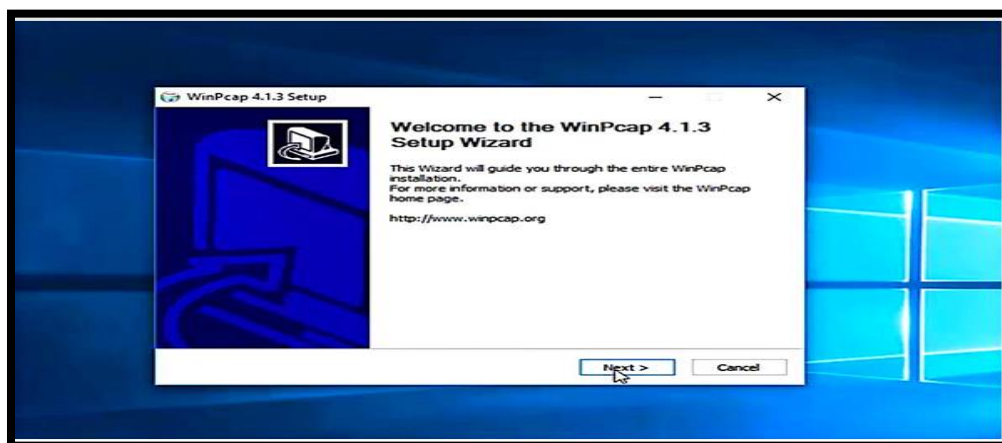




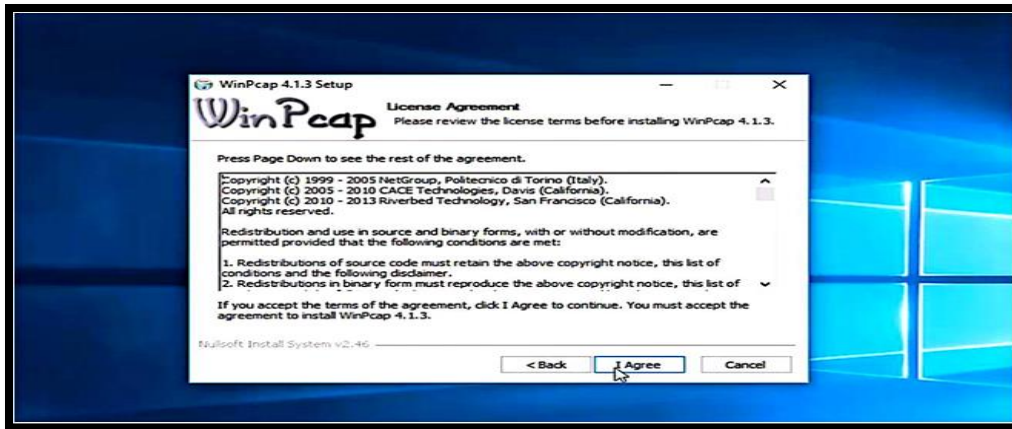
11) Install “USBcap” to capture the usb traffic and click “Next”



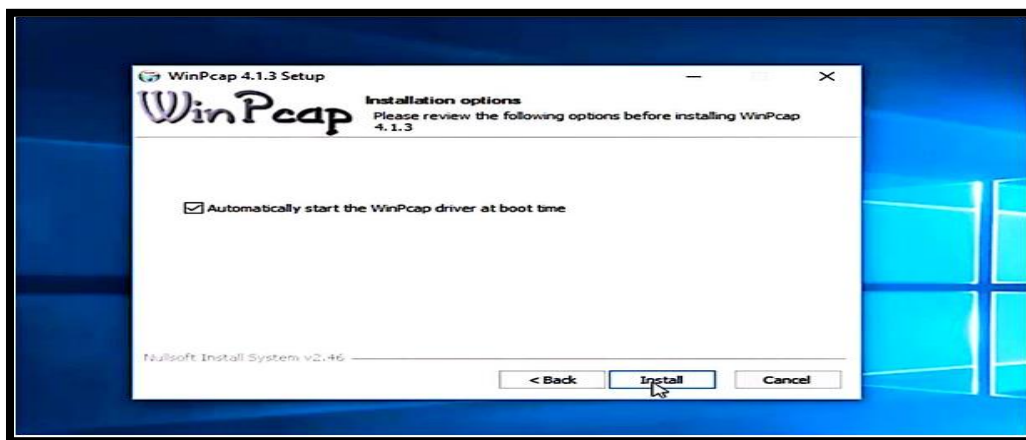
12) Click “Next”



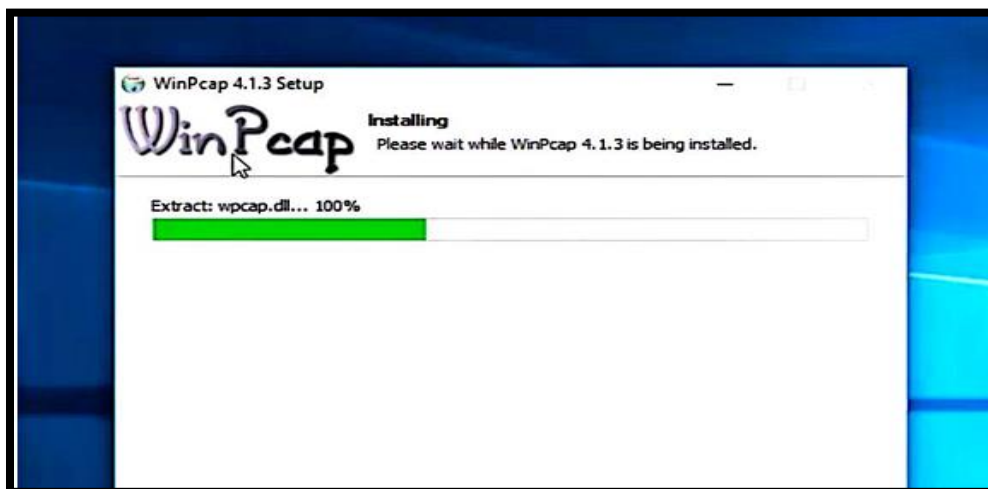
13) Click “I Agree”



14) Click “Install”

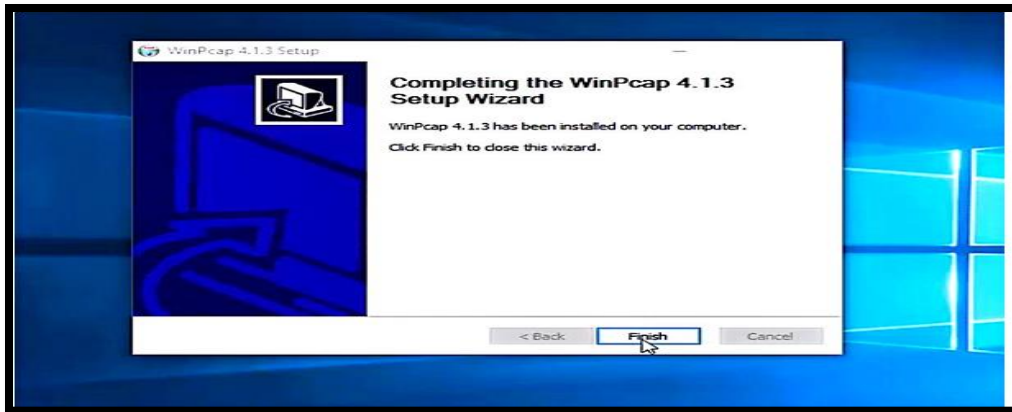


15) It will start Downloading.

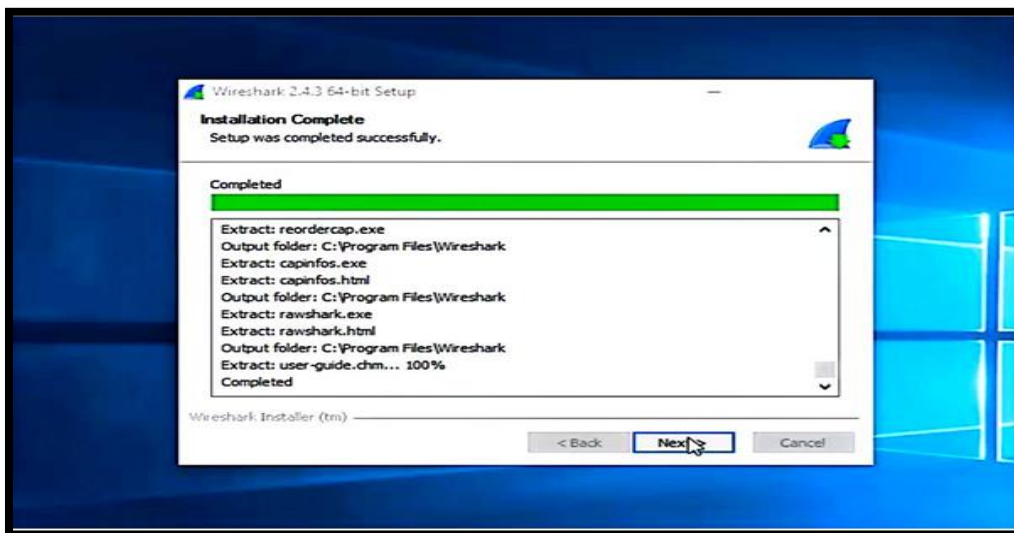


16) Click “Finish”





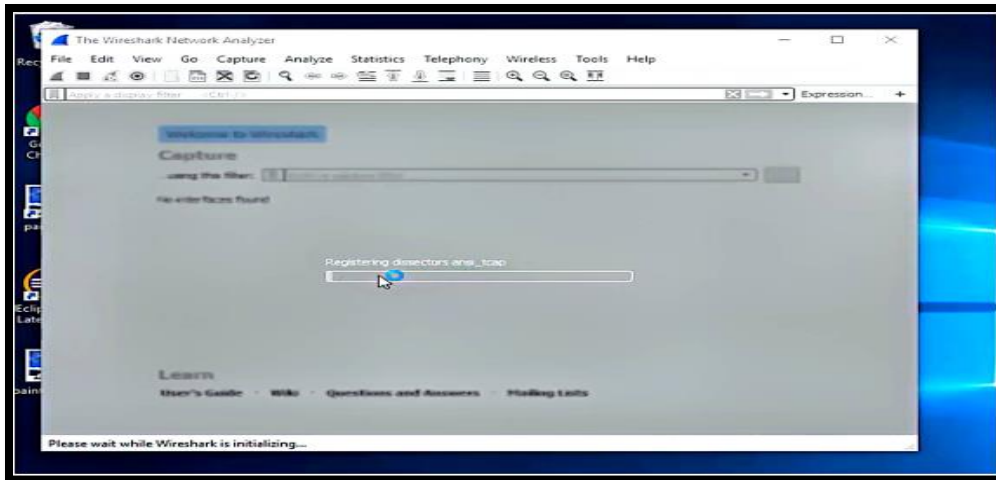
17) Click "Next"



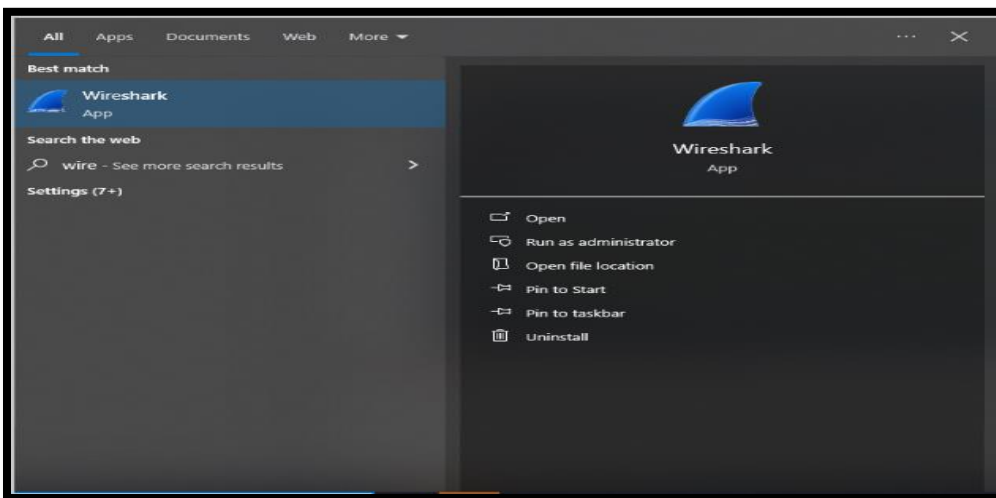
18) Click "Finish" and choose "Run Wireshark 2.4.3 64 bit"



19) It will start registering



20) The wireshark has been successfully installed.



### 3. Tool Use Cases:

#### 3.1. Http and Https Difference:

##### **HTTP (HyperText Transfer Protocol):**

- Used for transferring data between a client and a web server.
- Operates over port 80.
- Data is sent in plain text (unencrypted).
- Commonly used on non-sensitive or local websites.

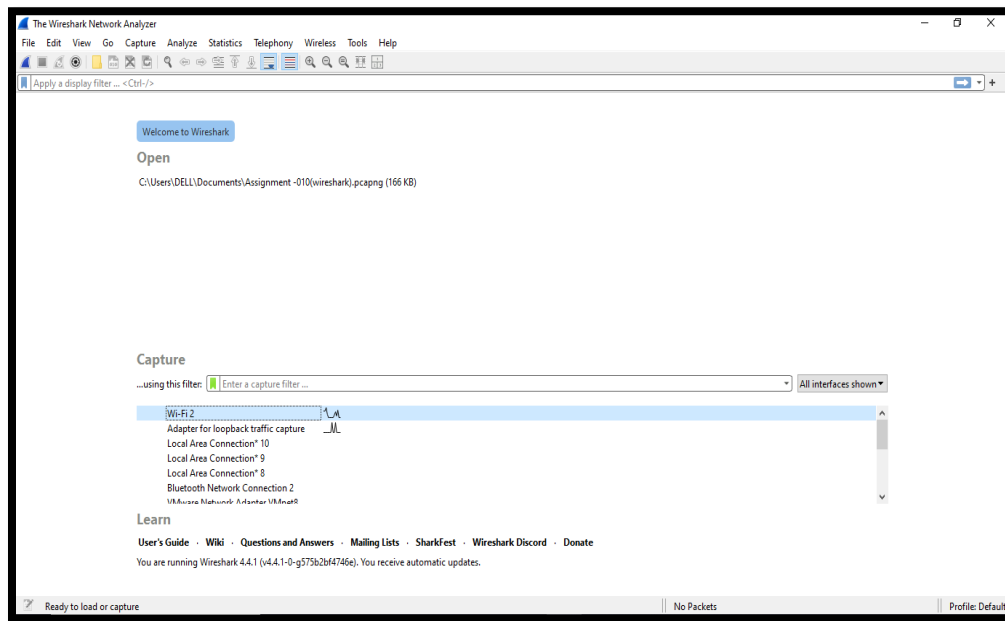
##### **HTTPS (HyperText Transfer Protocol Secure):**

- Secure version of HTTP that uses **SSL/TLS encryption**.
- Operates over **port 443**.
- Data is **encrypted**, preventing third-party access.

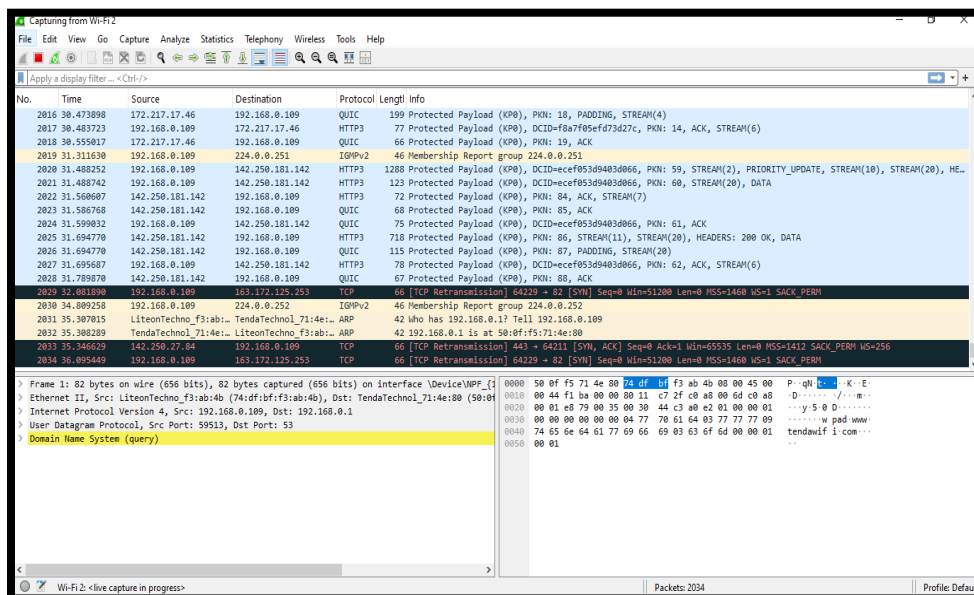
- Used by **secure websites** like banking, login portals, and e-commerce platforms.

## Http Practical Experiment:

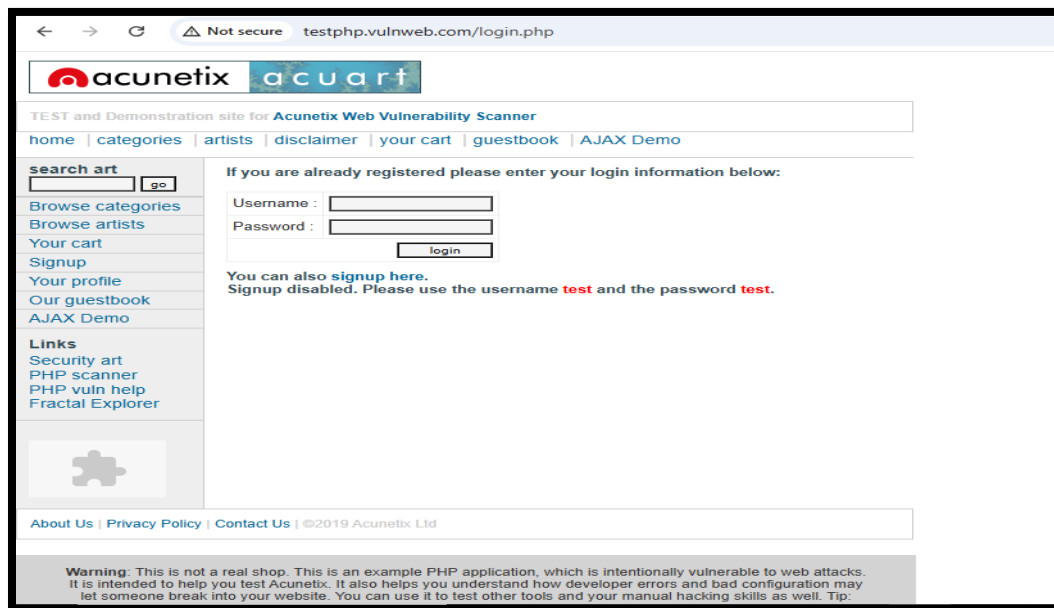
1. **Open Wireshark** and select the appropriate **network interface** (e.g., Wi-Fi) that the computer is currently using to connect to the internet.



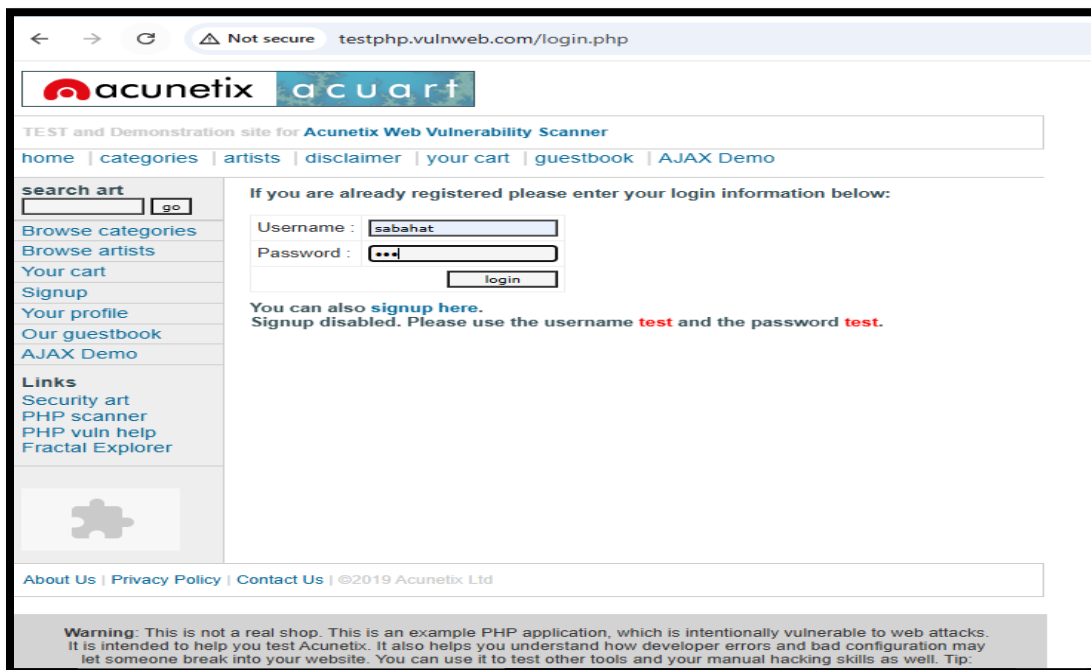
2. Once the interface is selected, **start packet capturing**.



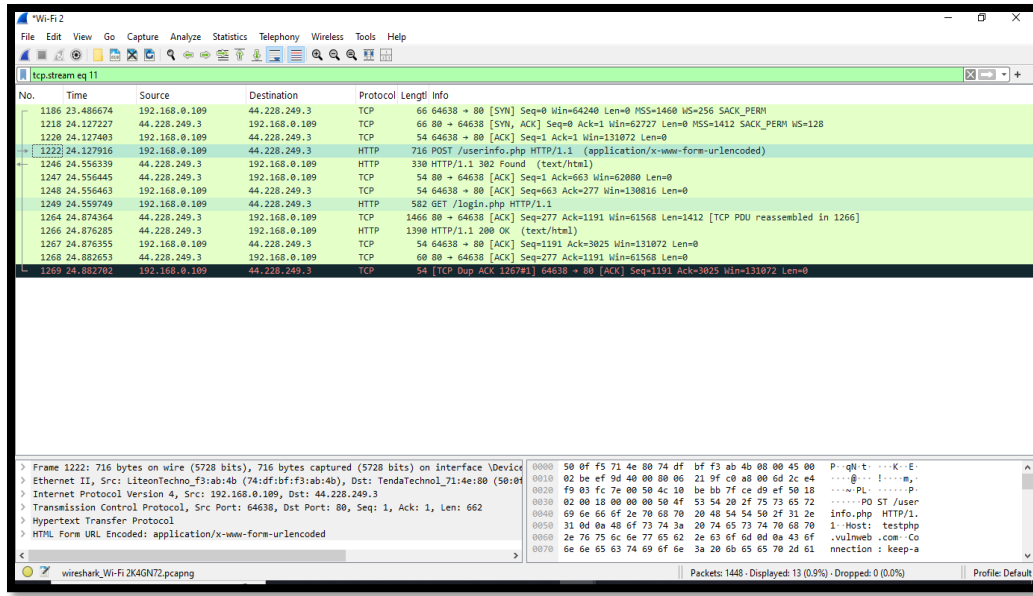
3. Open a **web browser** (such as Google Chrome) and navigate to the target website, such as a locally hosted test.php file containing a login form.



4. Enter a **username** and **password** into the form and submit it. After completing the login, **close the browser** to end the session.



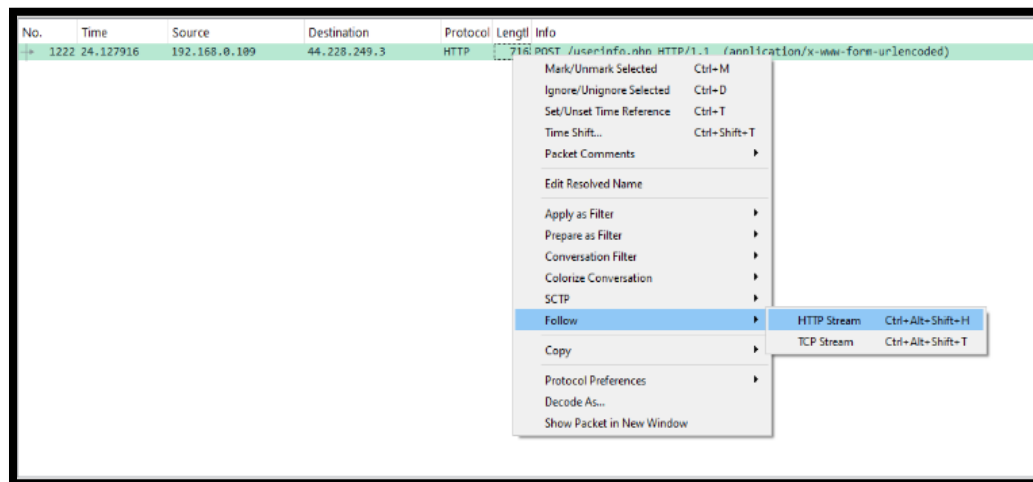
5. Return to Wireshark and **stop the packet capture**. Apply the display filter http to focus on HTTP traffic only.



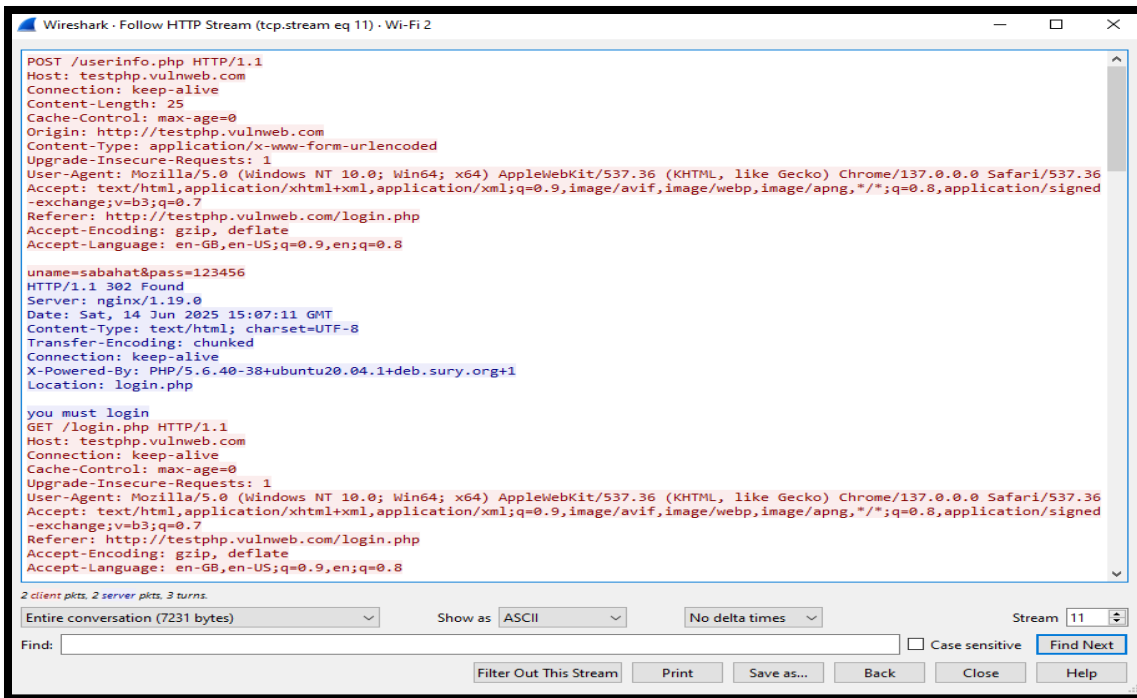
- To specifically locate the login request, apply the filter `http.request.method == "POST"` to isolate form submission packets.



- Right-click on the desired packet and select **Follow → HTTP Stream** from the context menu.



- The **full request and response stream** will be displayed, where the **username and password** can be **clearly seen in plain text**, indicating that the data was transmitted without encryption.



```
POST /userinfo.php HTTP/1.1
Host: testphp.vulnweb.com
Connection: keep-alive
Content-Length: 25
Cache-Control: max-age=0
Origin: http://testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://testphp.vulnweb.com/login.php
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

uname=sabahat&pass=123456
HTTP/1.1 302 Found
Server: nginx/1.19.0
Date: Sat, 14 Jun 2025 15:07:11 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Location: login.php

you must login
GET /login.php HTTP/1.1
Host: testphp.vulnweb.com
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://testphp.vulnweb.com/login.php
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
```

uname=sabahat&pass=123456

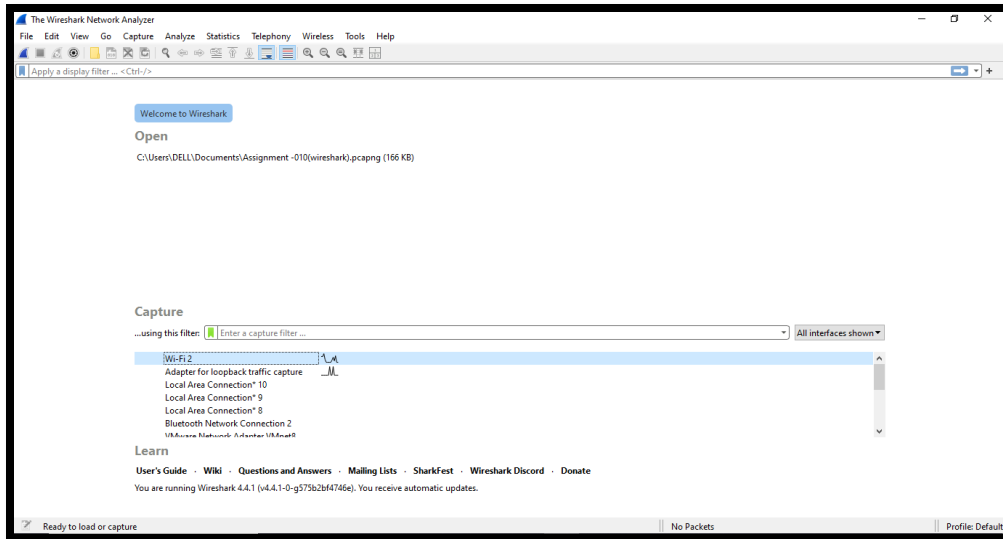
## Conclusion:

- The overall result of the experiment demonstrates that HTTP traffic is not secure, as sensitive information like usernames and passwords can be easily captured and viewed in plain text using Wireshark. By following and analyzing the HTTP stream of a login request, it becomes evident that data transmitted over HTTP lacks encryption and can be intercepted by anyone monitoring the network.
- The main goal of this experiment was to highlight the security risks of using HTTP and to show how tools like Wireshark can be used to analyze network traffic at the transport layer. It successfully proves that unencrypted communication is vulnerable, and therefore, secure protocols like HTTPS should always be used for protecting user data.

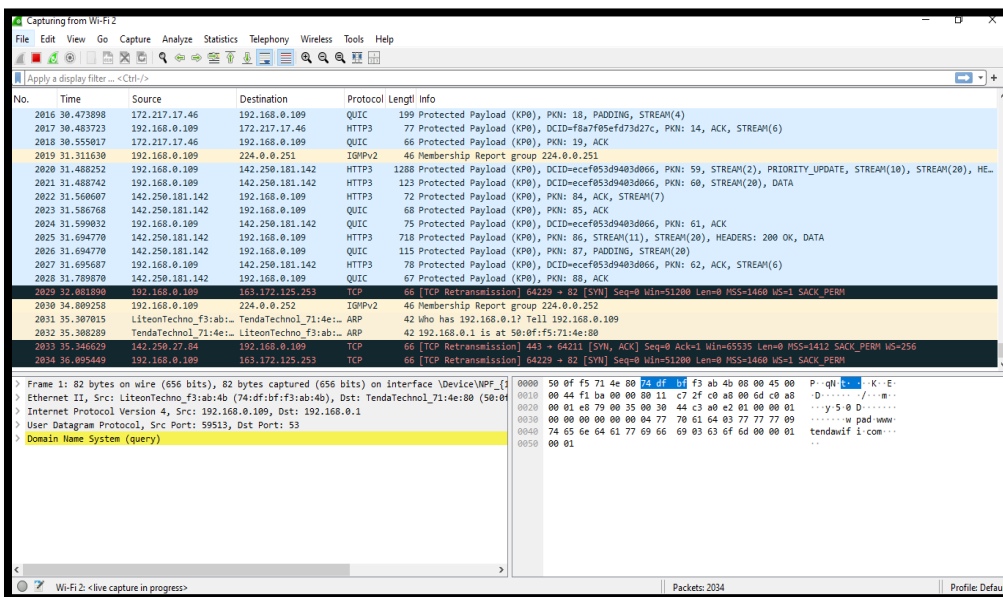
## Https Practical Experiment:

- 1) Open **Wireshark** and select the appropriate **network interface** (e.g., Wi-Fi) that the computer is currently using.

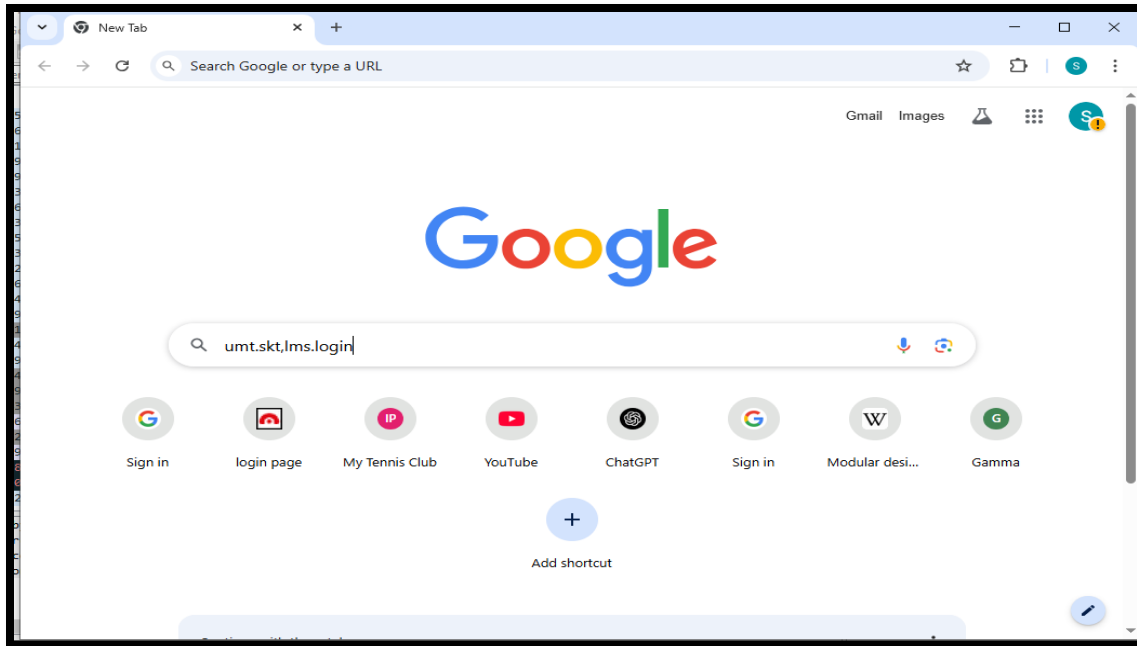




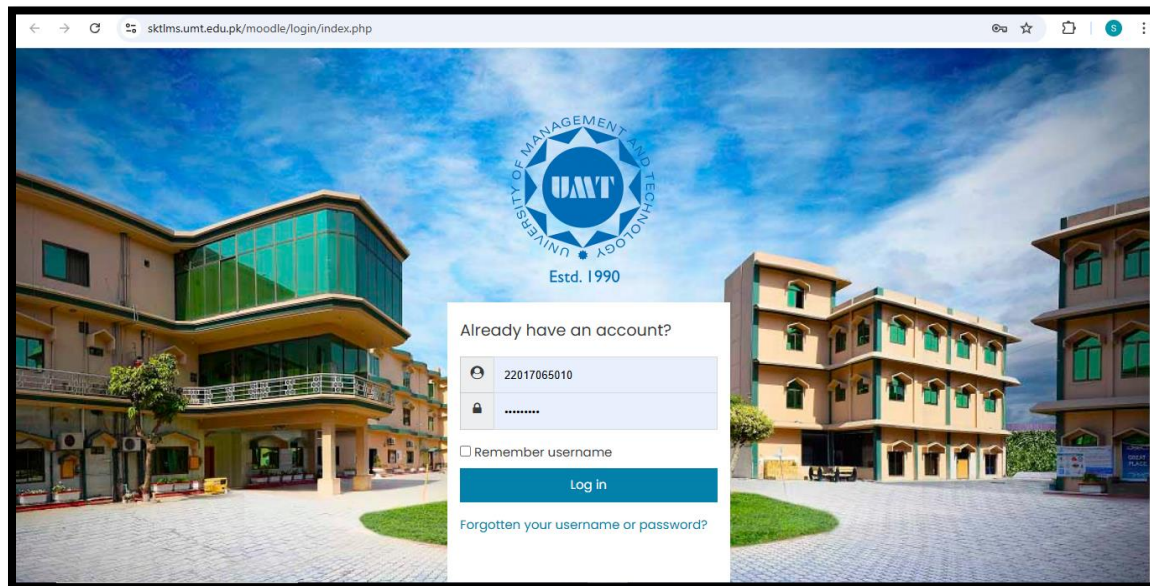
2) Start the **packet capture**.



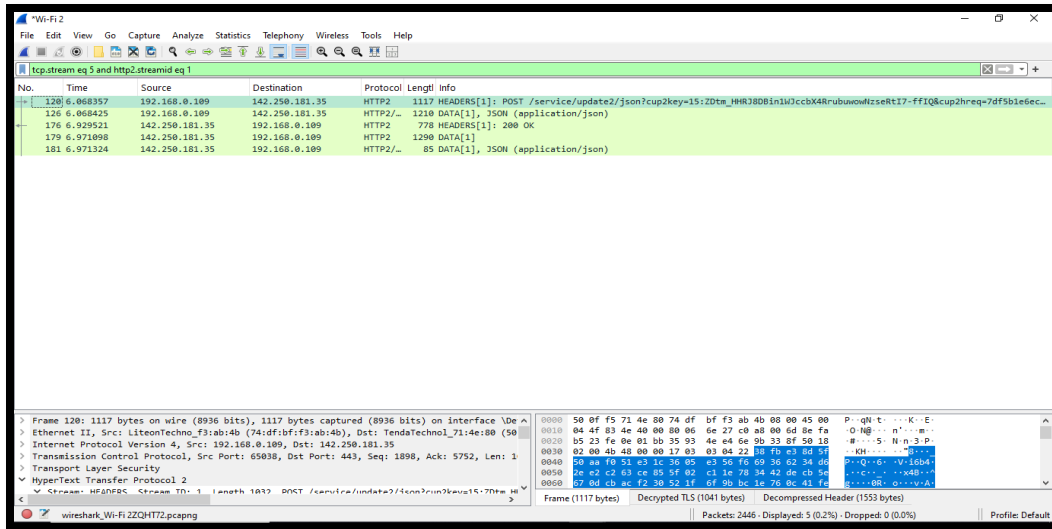
3) Open **Google Chrome** and visit a secure website, for example: <https://lms.umt.edu.pk>.



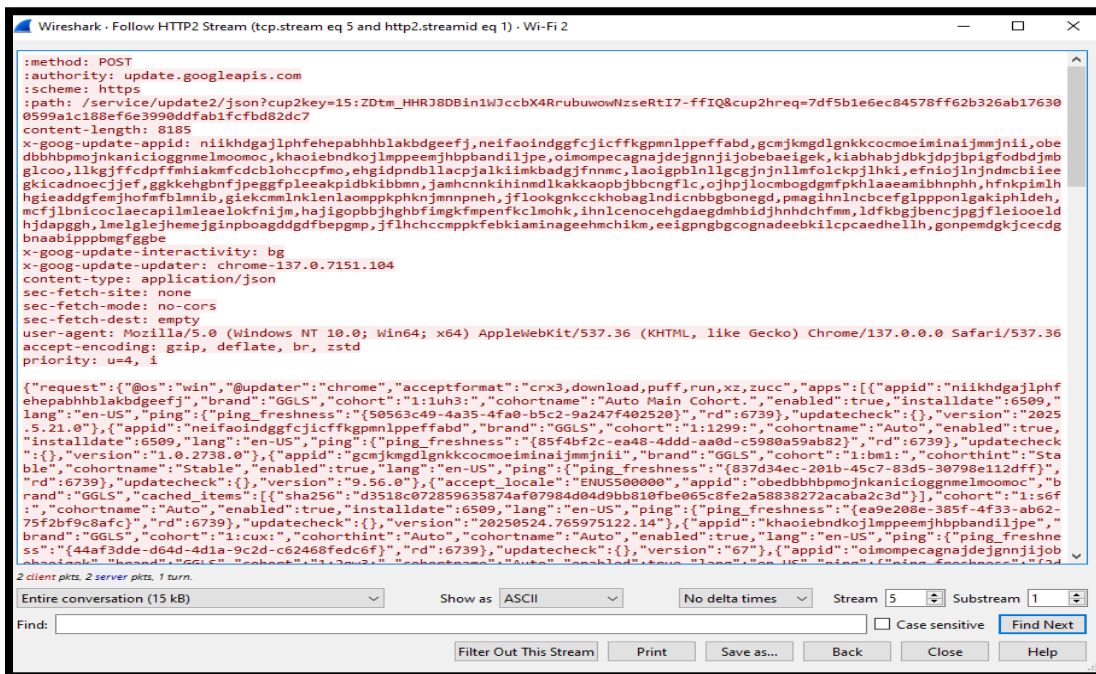
4) Proceed to **log in** using a username and password.



5) Return to Wireshark and apply the filter `http2` to analyze HTTPS traffic.



6) However, the traffic remains **encrypted**, and no readable information such as username or password is displayed.



➤ **Note:** It is important to note that **HTTPS** traffic cannot be directly decrypted in Wireshark. To view the content, the traffic would first need to be **converted or downgraded to HTTP**, which is not possible for secure websites unless special access or server-side keys are available.

## Conclusion:

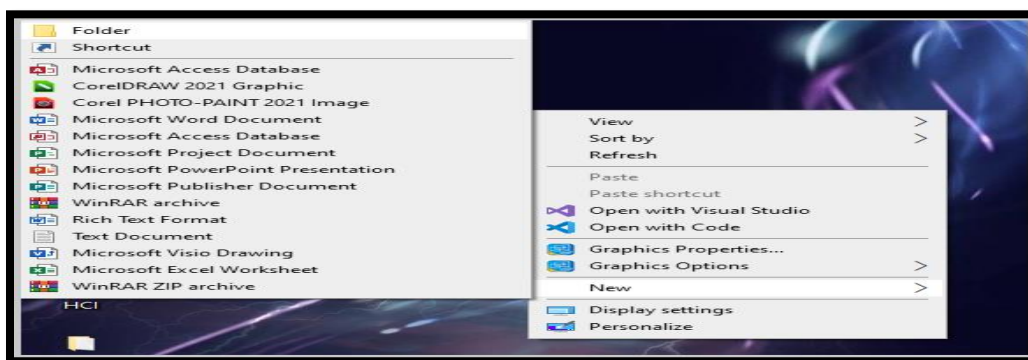
- The experiment confirms that **HTTPS encrypts all sensitive data**, including login credentials, using protocols like **TLS (Transport Layer Security)**. As a result, tools like Wireshark **cannot decrypt or display** the actual content of HTTPS packets without access to encryption keys. This ensures user privacy and security on secure websites.
- This test highlights the effectiveness of HTTPS in **protecting sensitive data from network sniffing** and reinforces why secure websites must always use **HTTPS instead of HTTP**.

## 3.2. Https Traffic Decryption(fetch session ID and Decrypt user Session):

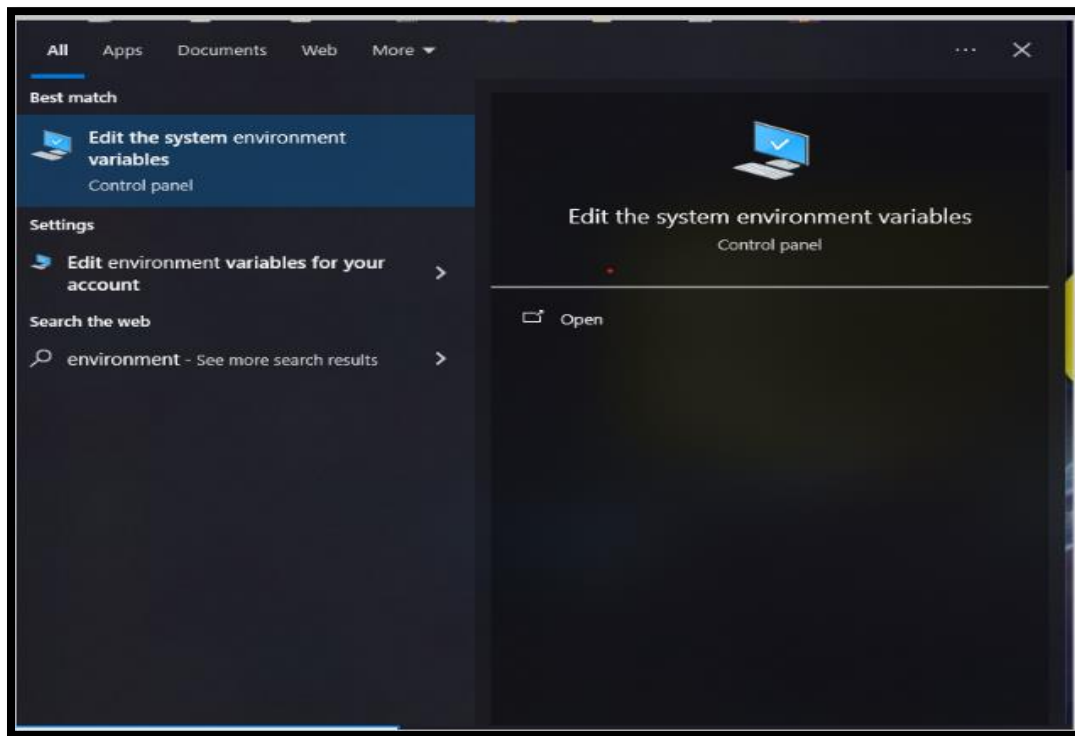
- HTTPS traffic decryption involves analyzing encrypted web communication to extract specific elements like **Session IDs** and understand user sessions. Unlike HTTP, which transmits data in plain text, **HTTPS uses SSL/TLS encryption** to protect data from being intercepted or read during transmission. As a result, tools like Wireshark **cannot directly decrypt HTTPS traffic** unless the necessary **private key or session secrets** are available.
- To fetch the **Session ID**, one can inspect the TLS handshake and identify the session-related metadata; however, the actual payload remains encrypted. Decrypting a full user session requires access to the **server's SSL/TLS key or pre-master secrets** (which can sometimes be exported from browsers like Chrome or Firefox using environment variables).
- In this manual's experiment, it was attempted to capture and decrypt HTTPS traffic by applying relevant filters (such as `http2`), but no readable information (like username, password, or session data) was displayed. This confirms that without decryption keys, HTTPS session data remains **secure and hidden**, thus ensuring **user privacy and protection against sniffing attacks**.

## How to decrypt Https traffic and user sessions?

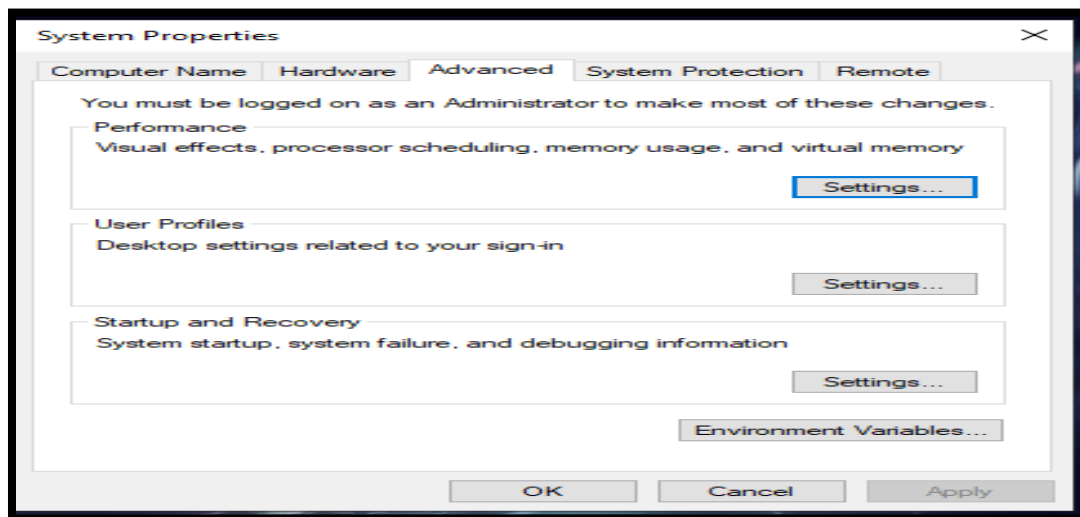
1. First, create a folder on the Desktop and name it **rrr** to store the **SSL session log file**.



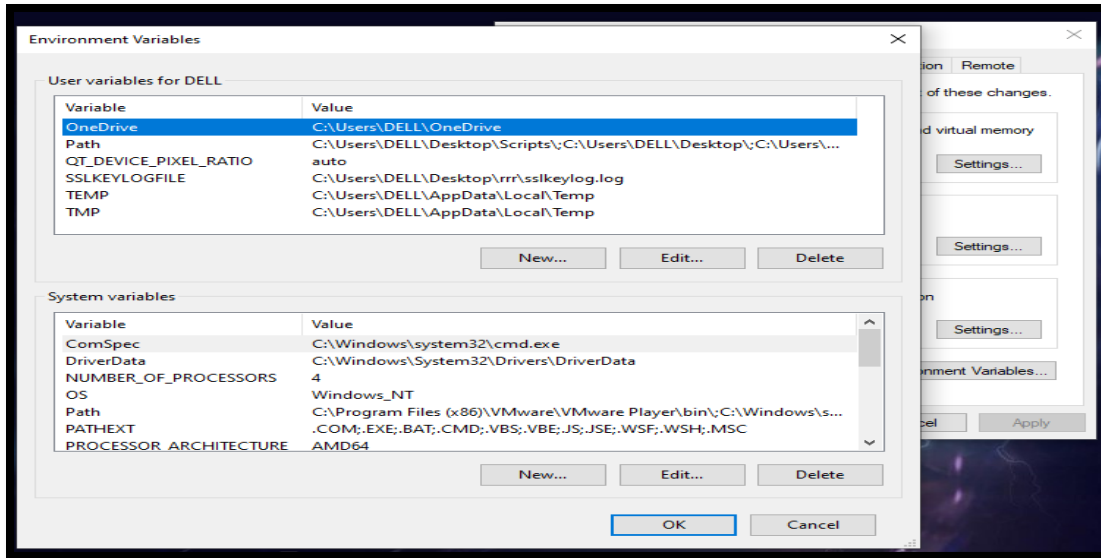
2. Open the Windows search bar and type “**Environment Variables**”.



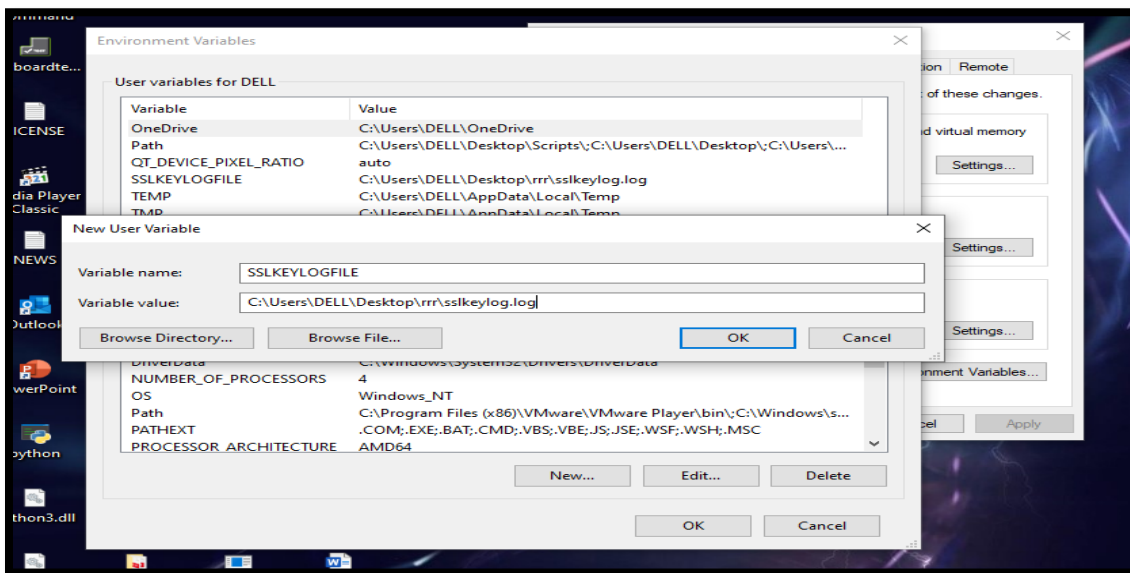
3. In the System Properties window, click on the “Environment Variables” button.



4. Under "User Variables", click “New” to create a new variable.

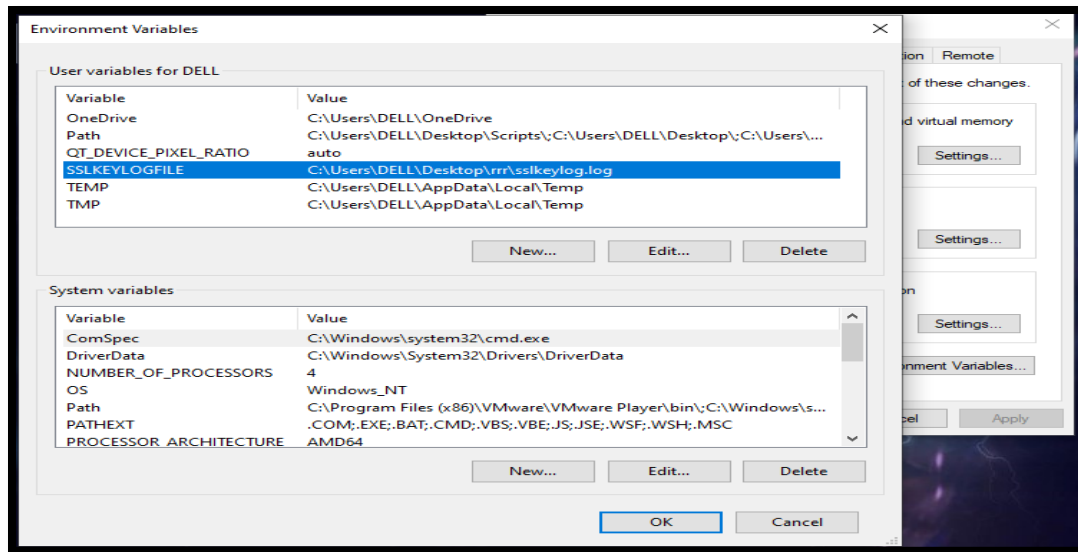


5. Enter SSLKEYLOGFILE as the variable name, and as the variable value, provide the full path to the log file inside the rrr folder (e.g., C:\Users\YourName\Desktop\rrr\logfile.txt).

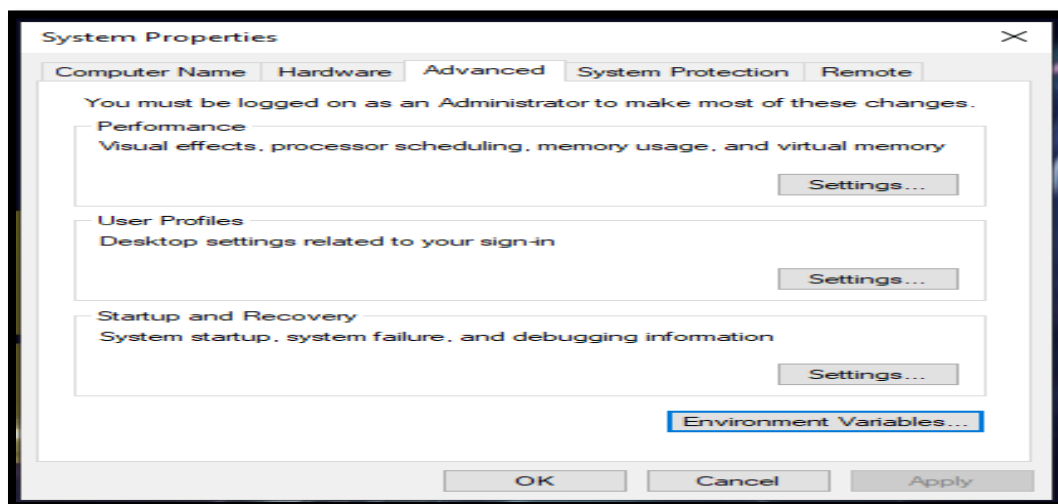


6. Click OK,

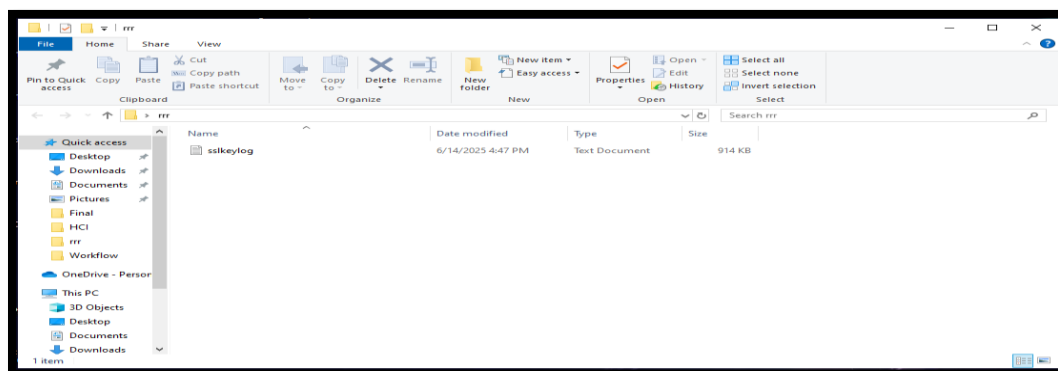




7. then again click OK to save the changes.



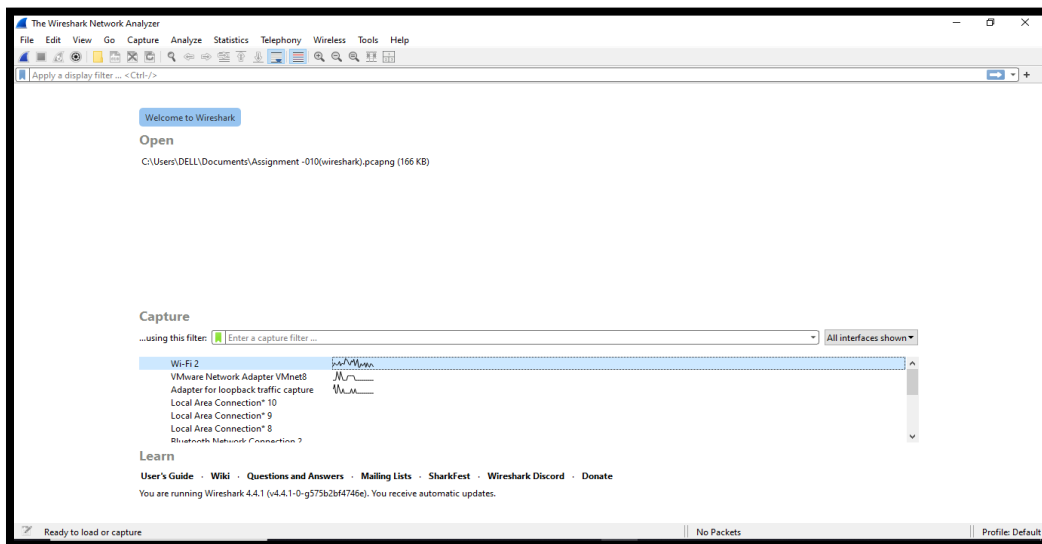
8. The environment variable is now created



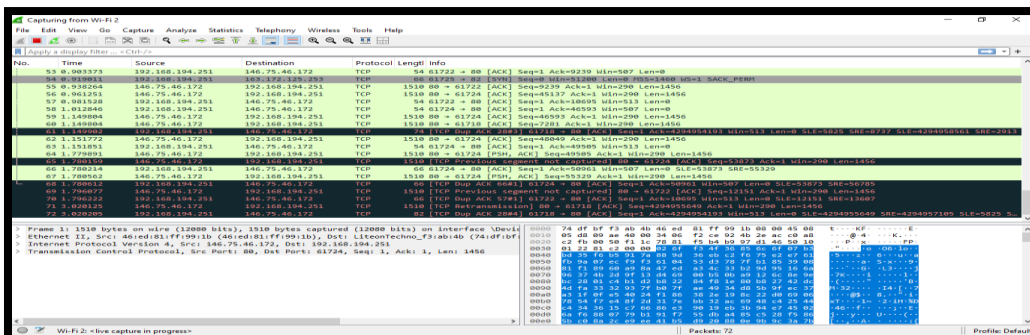
## 9. The log file will be auto-generated by supported browsers like Chrome.



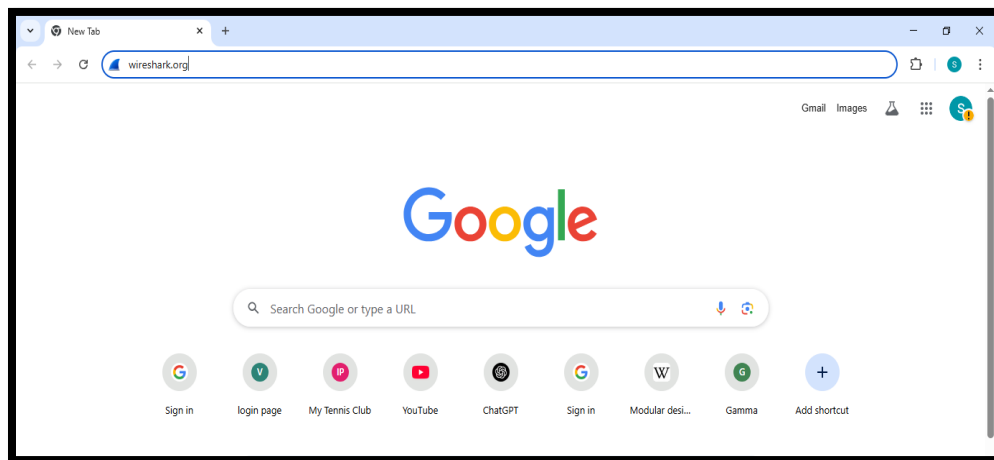
## 10. Open Wireshark and select your active network interface (e.g., Wi-Fi).



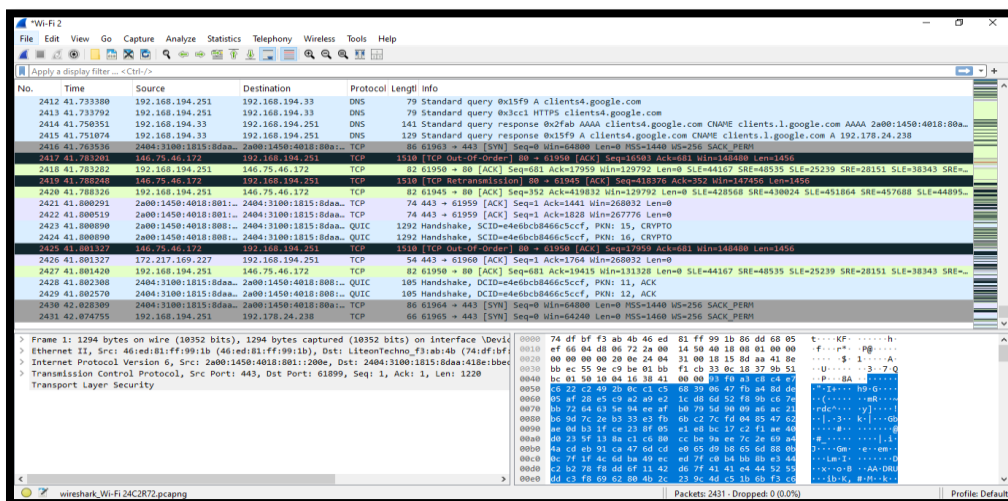
## 11. Start capturing the packets.



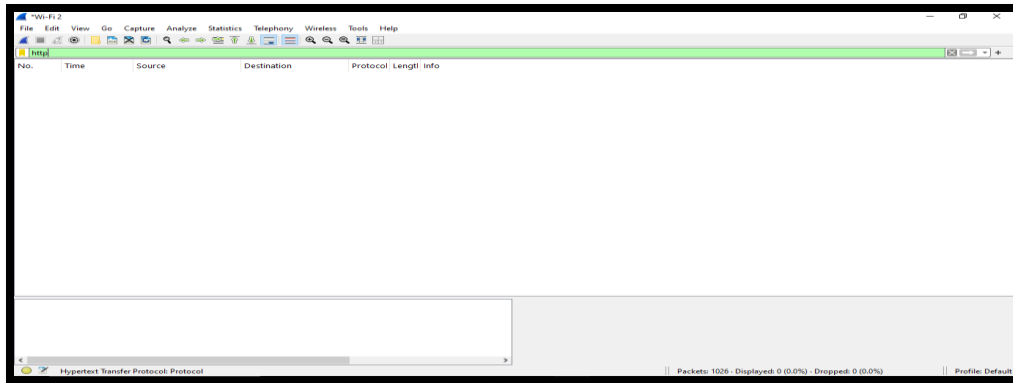
12. Open Google Chrome and visit <https://www.wireshark.org>. Once the website loads, close Chrome.



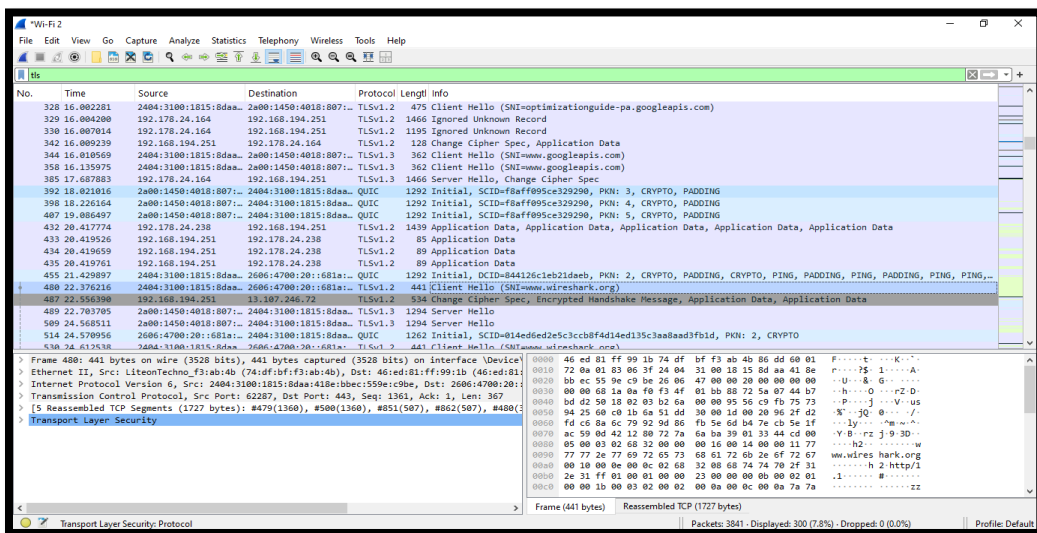
13. Stop the packet capture in Wireshark.



14. Apply the http filter in Wireshark; no results will appear because wireshark.org uses HTTPS and its data is encrypted.



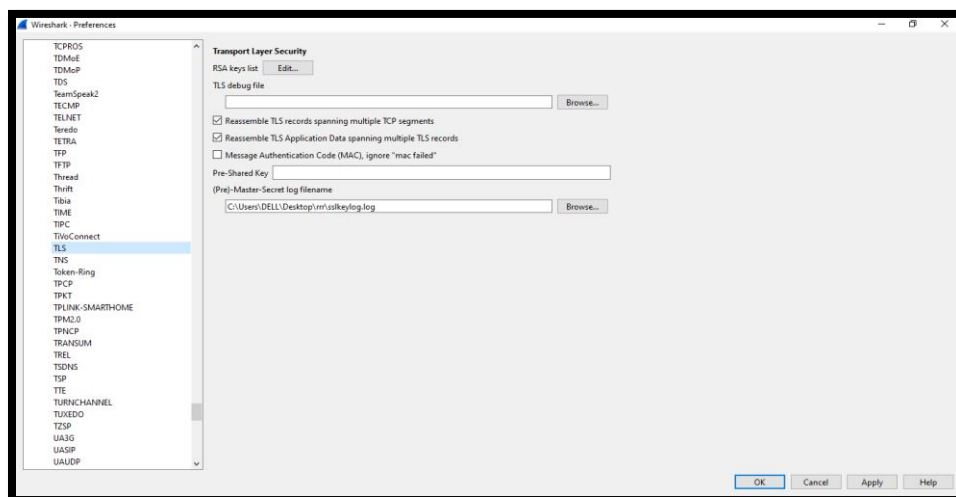
15. Apply the `tls or tcp.port == 443` filter to view TLS handshake packets.



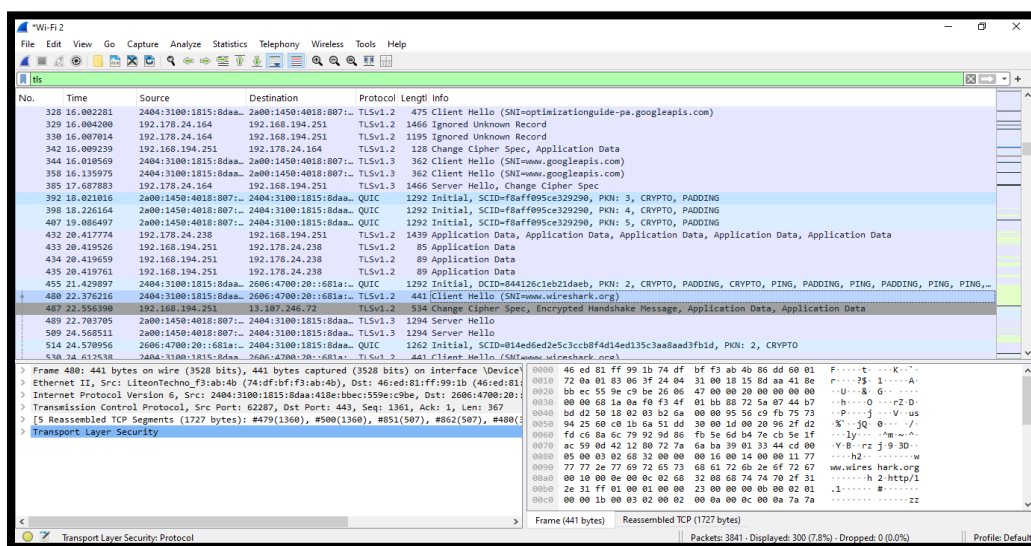
16. Go to Edit > Preferences in Wireshark.



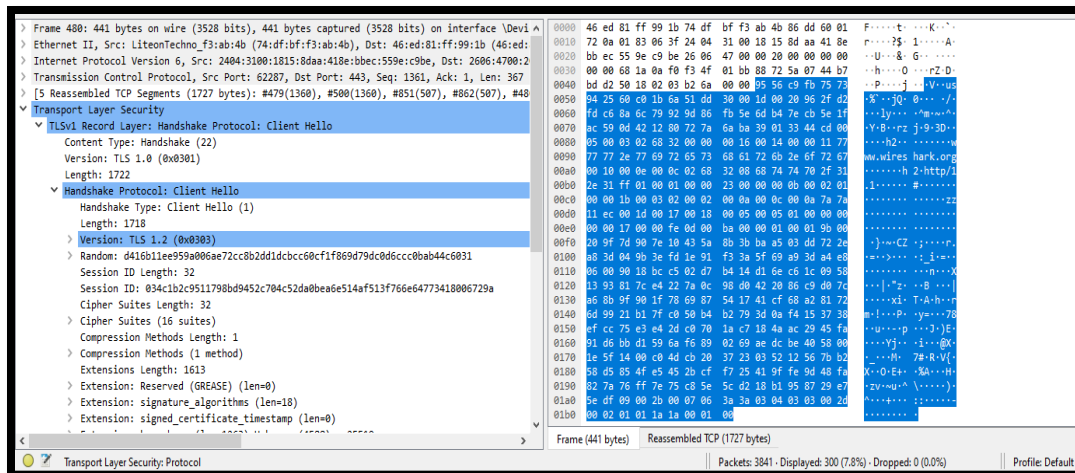
17. Select **Protocols > TLS** from the list on the left. In the field labeled **(Pre)-Master-Secret log filename**, enter the path to the log file you previously created. Click OK to save the preferences.



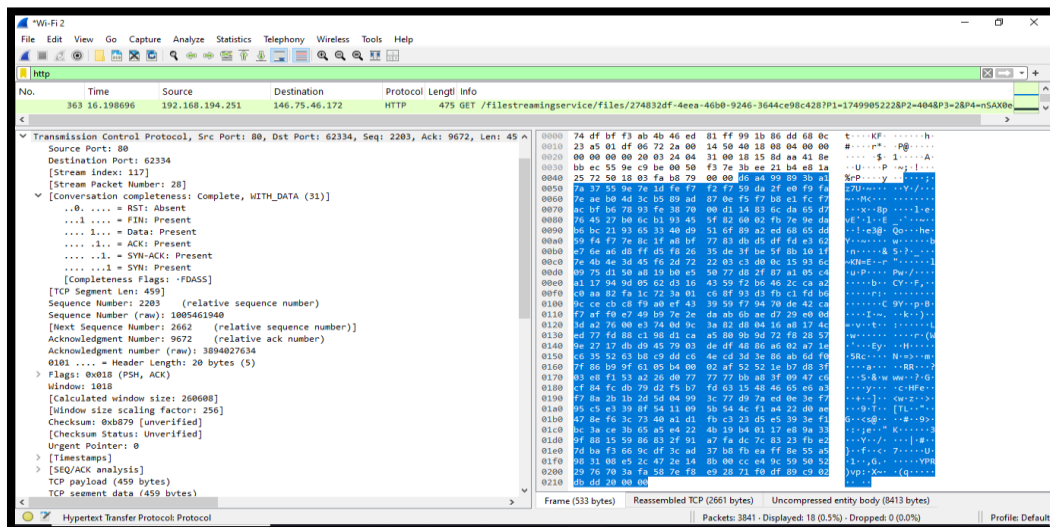
18. Now, apply the **tls** filter again — Wireshark will decrypt the HTTPS traffic using the log file.



19. Here is the encrypted session ID and server information. Click on any TCP packet and look under the TLS section to find the encrypted Session ID.



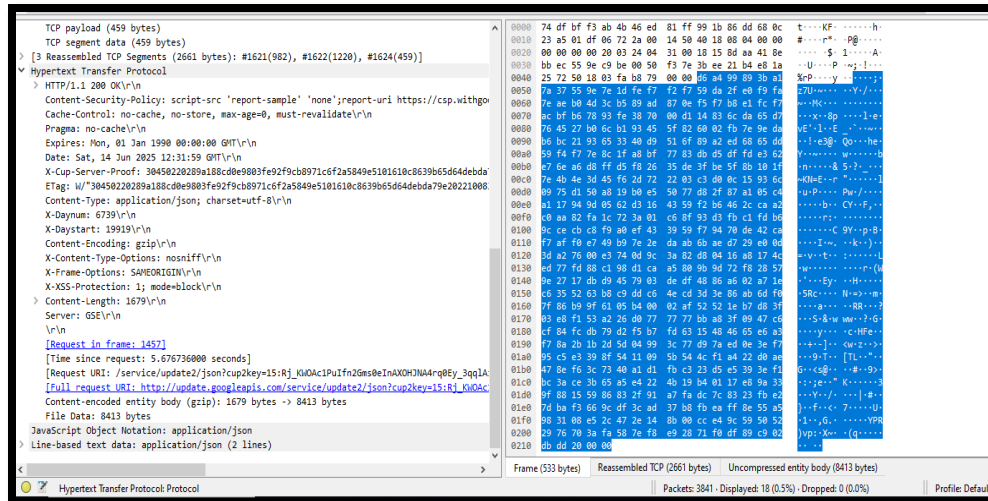
20. Apply the http filter again, and this time you will see the decrypted HTTP data.



- Extension: server\_name (len=22) name=www.wireshark.org
  - Type: server\_name (0)
  - Length: 22
- Server Name Indication extension
  - Server Name list length: 20
  - Server Name Type: host\_name (0)
  - Server Name length: 17
  - Server Name: www.wireshark.org

21. In the HTTP stream, you can now view the decrypted user session, which includes data like cookies, session expiration time, server name, and more.





## Conclusion:

This experiment shows that HTTPS traffic can be decrypted in Wireshark only if the session keys are available. By setting the SSLKEYLOGFILE environment variable, browsers can export these keys, which Wireshark uses to decrypt and display encrypted sessions. This method helps understand how SSL/TLS provides security, and how network tools can be used responsibly to analyze traffic for educational or debugging purposes.

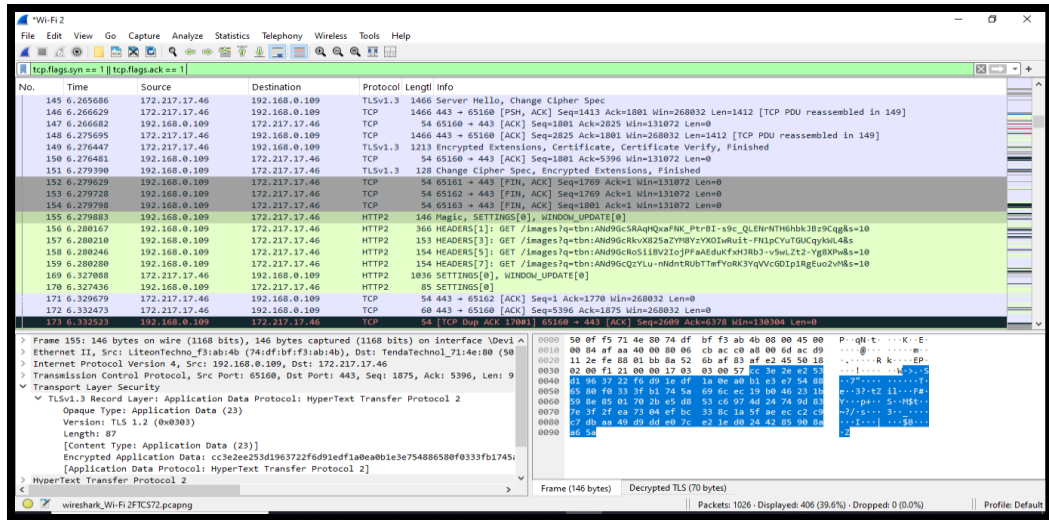
## 3.3. TCP Stream Analysis for SYN Flood Detection and UDP Analysis in https:

**Note:** Follow all the steps mentioned in use case 2 and then perform this:

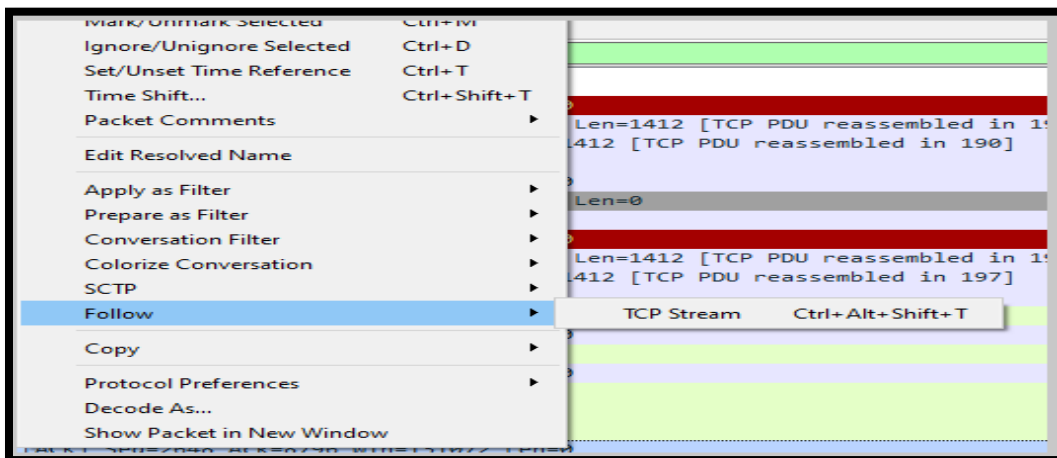
TCP 3-Way Handshake Table

Step	Packet Type	Description	Flags Set
1	SYN	Client sends request to start connection	SYN
2	SYN-ACK	Server responds to client	SYN, ACK
3	ACK	Client acknowledges the server's response	ACK

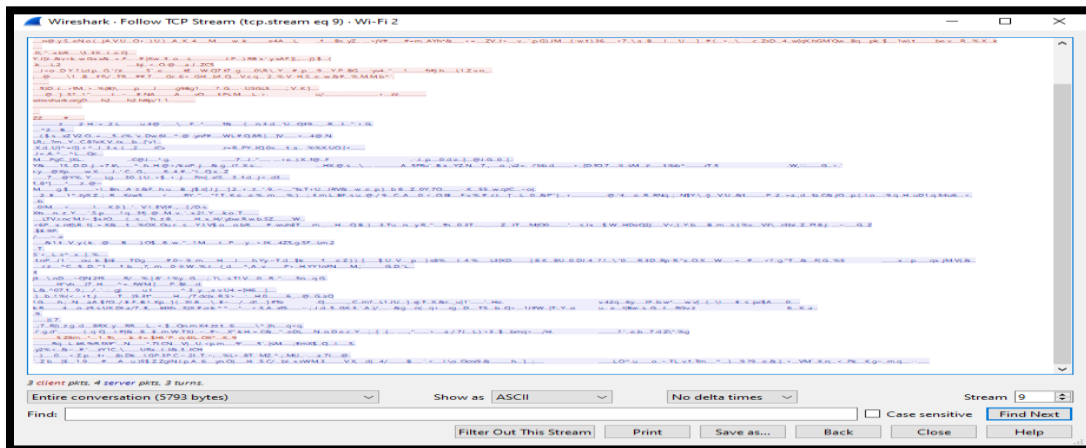
1) Apply this filter on wireshark **“tcp.flags.syn == 1 and tcp.flags.ack == 0”**



2) Right-click on any packet with the [SYN] flag. Select **Follow > TCP Stream** from the dropdown menu.



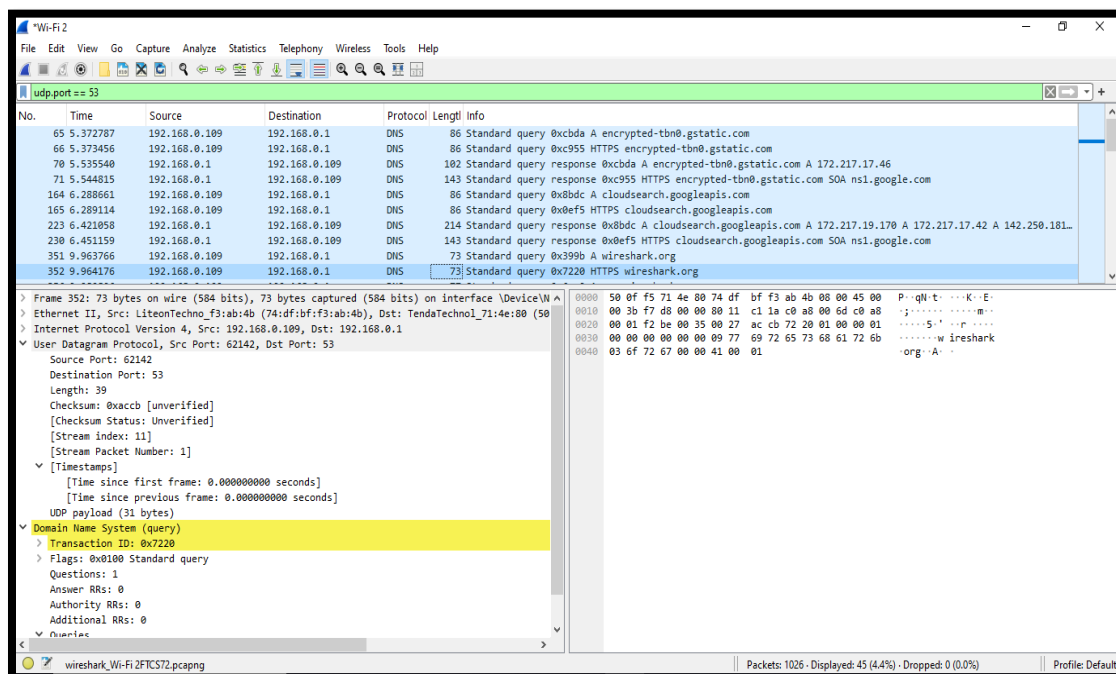
3) Wireshark will display the **entire TCP conversation** between the client and server (like a chat), allowing you to trace the session.



## UDP Analysis:

Observation in Wireshark	Possible Abnormality	Explanation
High number of packets to/from port 53	DNS Amplification Attack	Abnormally high DNS traffic (often with spoofed IPs) may indicate a reflection-based attack.
Repeated packets to same IP/port with no response	UDP Flooding / Port Scanning	Shows attacker is trying to overwhelm or probe UDP services.
"ICMP Port Unreachable" replies	UDP Port Scanning (e.g., Nmap)	Indicates scanned port is closed and no service is listening.
Payload size too large for protocol (e.g., DNS > 512 bytes)	DNS tunneling or data exfiltration	Large DNS packets may be hiding data inside.
Unusual destination ports (e.g., 19, 123, 161)	Abused for attacks (e.g., CHARGEN, NTP, SNMP)	Attackers target legacy or misconfigured services for amplification.
No application-layer data	Idle traffic or scanning	Indicates non-standard usage or probing activity.

1) Search “UDP” in the wireshark filter.



4. OSI Layer Mapping (Focused on Transport Layer):

Tool Example	OSI Layer	Protocols Involved	Role in Security
Wireshark	Transport Layer	TCP, UDP	<b>Packet inspection</b> and <b>session analysis</b> to understand how data flows between endpoints and to detect abnormalities (e.g., broken handshakes, retransmissions).
Wireshark	Transport Layer	TLS over TCP	Analyzes encrypted traffic handshakes to verify secure session establishment or to decrypt sessions using pre-master keys.
Wireshark	Transport Layer	HTTP (via TCP), HTTPS	Monitors login sessions, detects credential leaks in HTTP, and confirms encryption in HTTPS.
Wireshark	Transport Layer	TCP Stream (SYN, SYN-ACK)	<b>TCP Stream analysis</b> used to detect <b>SYN Flood attacks</b> by inspecting incomplete or repeated handshakes and flagging abnormal TCP connection attempts.
Wireshark	Transport Layer	UDP in HTTPS context	Detects misuse of UDP protocols (e.g., DNS, NTP) inside encrypted sessions or alongside HTTPS traffic, helping uncover tunneling, scanning, or amplification patterns.

5. Mapping to MITRE ATT&CK Framework):

Tool	Tool Feature	Description	MITRE Tactic	Technique ID
Wireshark	Packet Analysis	Inspect unencrypted HTTP credentials	Credential Access	T1040

<b>Wireshark</b>	Encrypted Traffic Analysis	Decrypt HTTPS sessions using SSL key logs	Credential Access / Initial Access	T1557 / T1552
<b>Wireshark</b>	TCP Stream Analysis	Analyze incomplete TCP handshakes to detect <b>SYN Flood</b> attempts	Denial of Service	T1499.001
<b>Wireshark</b>	Session Hijack Detection	Inspect captured session tokens or IDs in HTTP traffic	Credential Access / Session Hijack	T1071 / T1021.001
<b>Wireshark</b>	UDP Behavior Monitoring	Detect UDP flood, port scanning, or DNS tunneling	Discovery / Command and Control	T1046 / T1095
<b>Wireshark</b>	TLS Handshake Inspection	Monitor TLS packets to detect untrusted certificates or failed handshakes	Defense Evasion / Initial Access	T1553.004

## **6. Mitigation Strategies:**

If tools like Wireshark are misused, they can expose sensitive data such as **session IDs**, which attackers can use to **hijack sessions**, impersonate users, or gain unauthorized access. Below are strategies to mitigate such risks:

### **1. Enforce HTTPS with Strong TLS Configurations**

Use secure HTTPS connections with up-to-date TLS versions and disable weak ciphers to ensure encrypted data cannot be easily intercepted or decrypted.

### **2. Disable SSL Key Logging in Production Environments**

Ensure that SSLKEYLOGFILE environment variable is never enabled on production machines or shared systems, as it allows the decryption of HTTPS traffic using exported session keys.

### **3. Secure Session Management**

- Use short session expiration times.
- Regenerate session IDs after login.
- Bind sessions to IP address or device fingerprinting to prevent reuse by attackers.

#### **4. Restrict Packet Capture Access**

Only authorized users (e.g., security analysts) should have access to packet capturing tools like Wireshark. Implement system-level permissions and monitor their usage.

#### **5. Use HTTP Security Headers**

Implement headers like Secure, HttpOnly, and SameSite on session cookies to prevent client-side access and cross-site attacks.

#### **6. Network Segmentation and Encryption**

Internal traffic should also be encrypted, and sensitive systems should be on isolated, secured networks to prevent lateral movement by attackers.



## **7. References:**

1. <https://www.wireshark.org/docs/>
2. <https://youtu.be/5qecyZHL-GU?feature=shared>
3. <https://attack.mitre.org>
4. [https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key\\_Log\\_Format](https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format)
5. <https://www.cloudflare.com/learning/ssl/>
6. <https://www.netacad.com>