

# Day 10

## Exploitation Analyst

### Hacking the SSL Network protocol:

#### Session Hijacking:

##### **What is a Session?**

A session is a temporary interaction between a user and a web application that helps maintain the user's state during their activity, especially since HTTP is stateless and doesn't remember users. When a user logs in, the server creates a session and assigns a unique session ID—usually stored in a cookie—which is sent with every request to identify the user. This allows the server to track authentication, access permissions, and user-specific data. The session ends when the user logs out or it times out. If a session ID is stolen, such as through session hijacking, an attacker can impersonate the user.

##### **What is Session Hijacking?**

Session hijacking is a cyberattack where an attacker takes over an active web session between a user and a server. Once hijacked, the attacker can impersonate the user and access sensitive information or perform actions on their behalf.

##### **How is it related to SSL?**

SSL/TLS (HTTPS) is meant to encrypt session data in transit, including session cookies, tokens, and credentials. However:

- If SSL/TLS is not enforced (i.e., user visits HTTP instead of HTTPS), session tokens can be sniffed.
- If there's an SSL downgrade attack (e.g., SSL stripping), data can be exposed in plaintext.
- Even over HTTPS, vulnerabilities like XSS or poor cookie security (HttpOnly, Secure flags missing) can leak session tokens.

##### **How HTTP handles sessions?**

- HTTP is stateless, so it doesn't retain any user data between requests.
- Sessions are managed using session IDs, typically stored in:
  - Cookies (most common)
  - URL parameters
  - Hidden form fields
- On each request, the session ID is sent to the server to associate the request with stored session data.

##### **What are cookies?**

Cookies are small pieces of data stored on the user's browser by a website. They are used to remember stateful information across multiple HTTP requests.

## How cookies are related with sessions?

- Since HTTP is stateless, cookies help maintain user sessions by storing a session ID.
- When a user logs in, the server generates a session ID and sends it in a cookie.
- The browser stores the cookie and sends it back with every request to the same site.
- The server uses the session ID to retrieve the user's session data (e.g., login status, preferences).
- So, cookies act as the link between a user and their server-side session data.

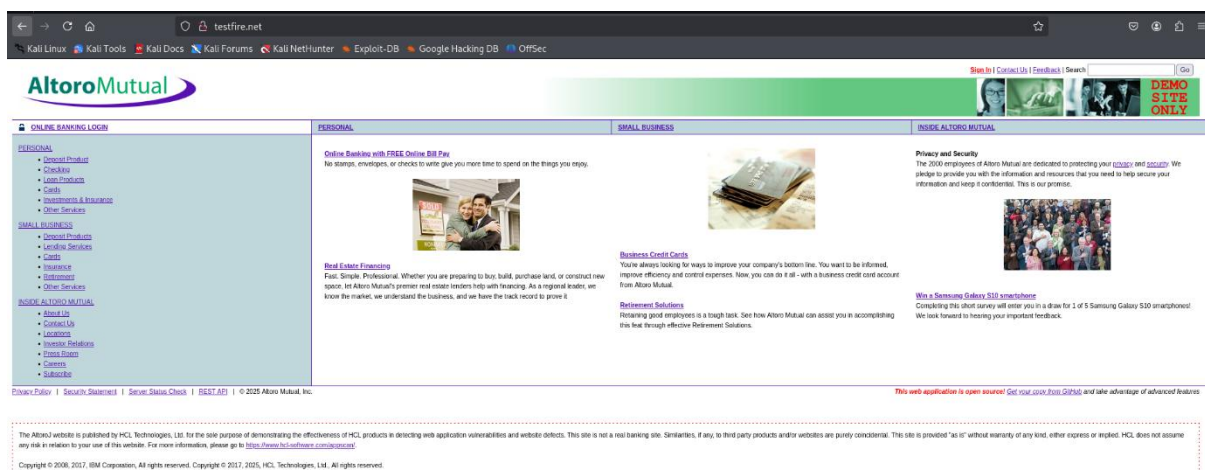
## Attributes of cookies:

- Name & Value: The actual data stored in the cookie as a key-value pair (e.g., sessionId=abc123).
- Domain: Specifies which domain the cookie is valid for.
- Path: Restricts the cookie to a specific path within the domain.
- Expires / Max-Age: Sets the expiration time or duration for the cookie.
- Secure: Sends the cookie only over HTTPS connections.
- HttpOnly: Blocks JavaScript from accessing the cookie, protecting against XSS.
- SameSite: Controls whether the cookie is sent with cross-site requests (Strict, Lax, or None).

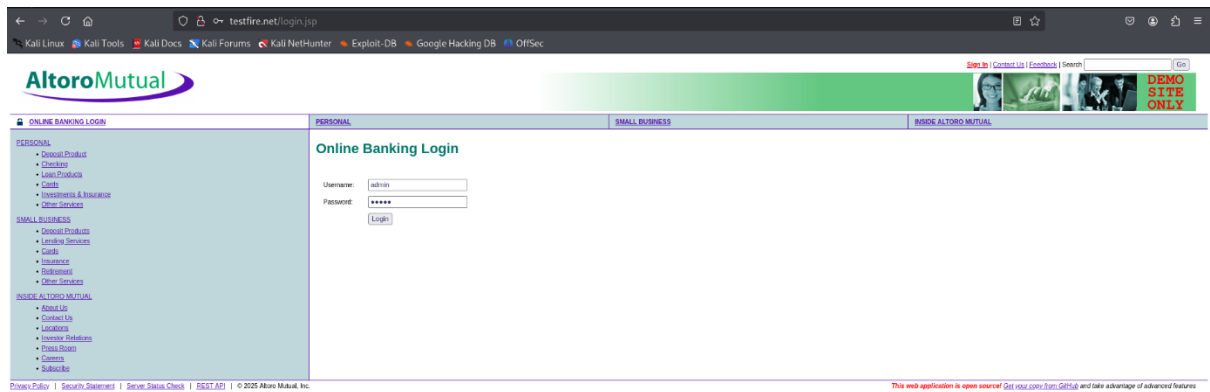
## Performing Session Hijacking: (Manually)

Steps:

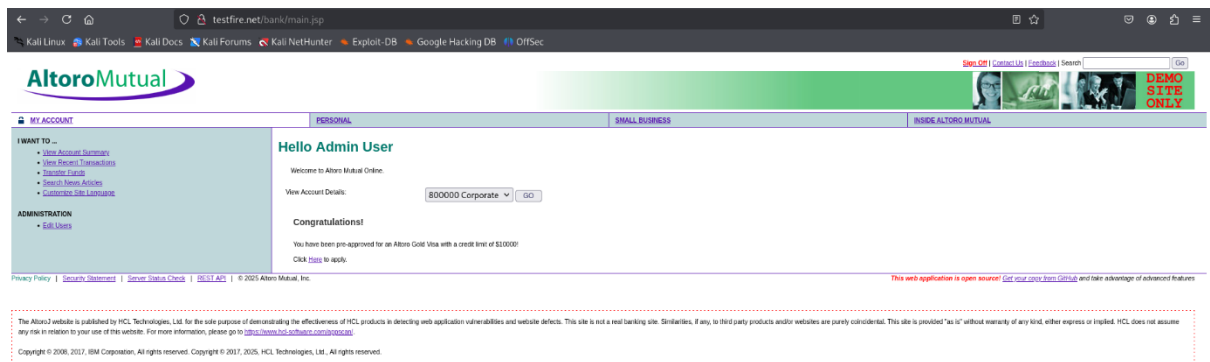
Open the website to try: (testfire[.]net)



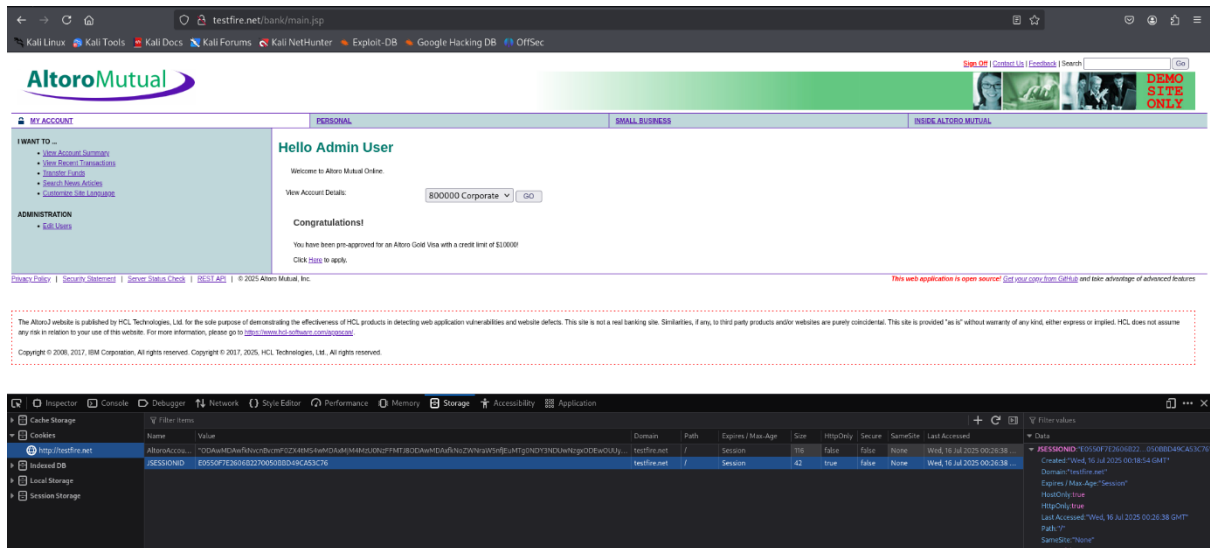
Click on the 'sign in' button and enter the credentials : id- admin , password – admin.



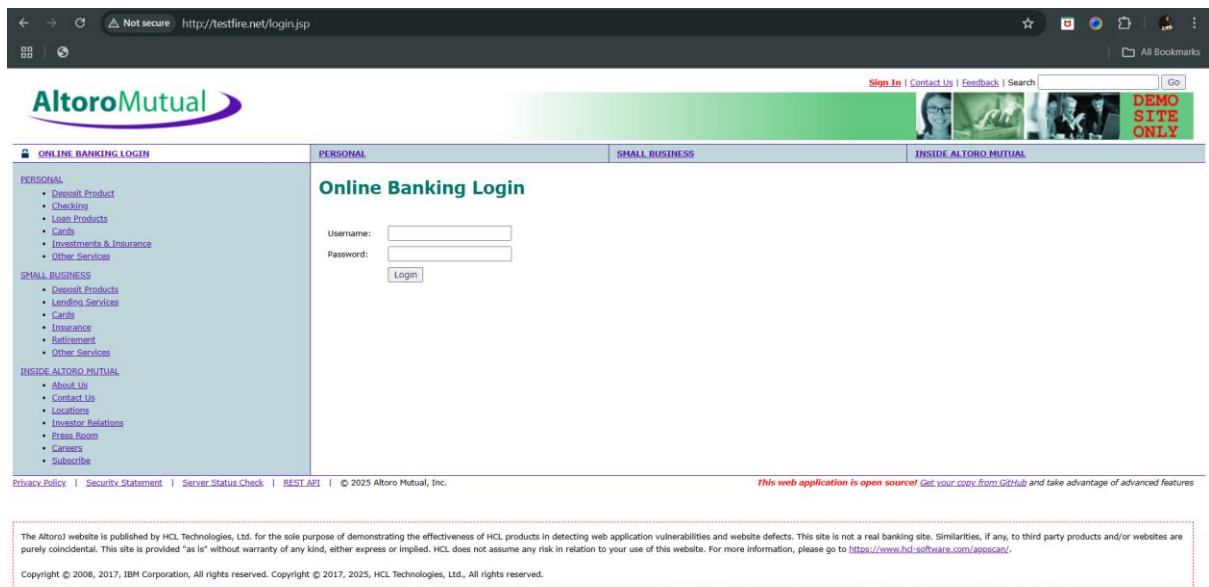
Following window will appear:



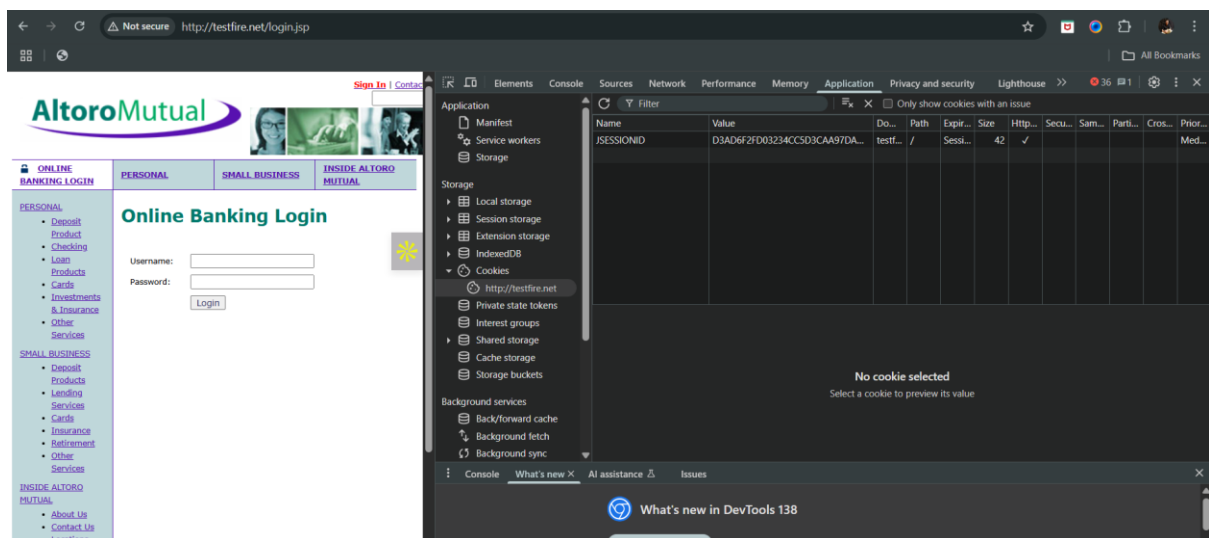
Right click and select the 'Inspect' button: See the session id. Copy it.



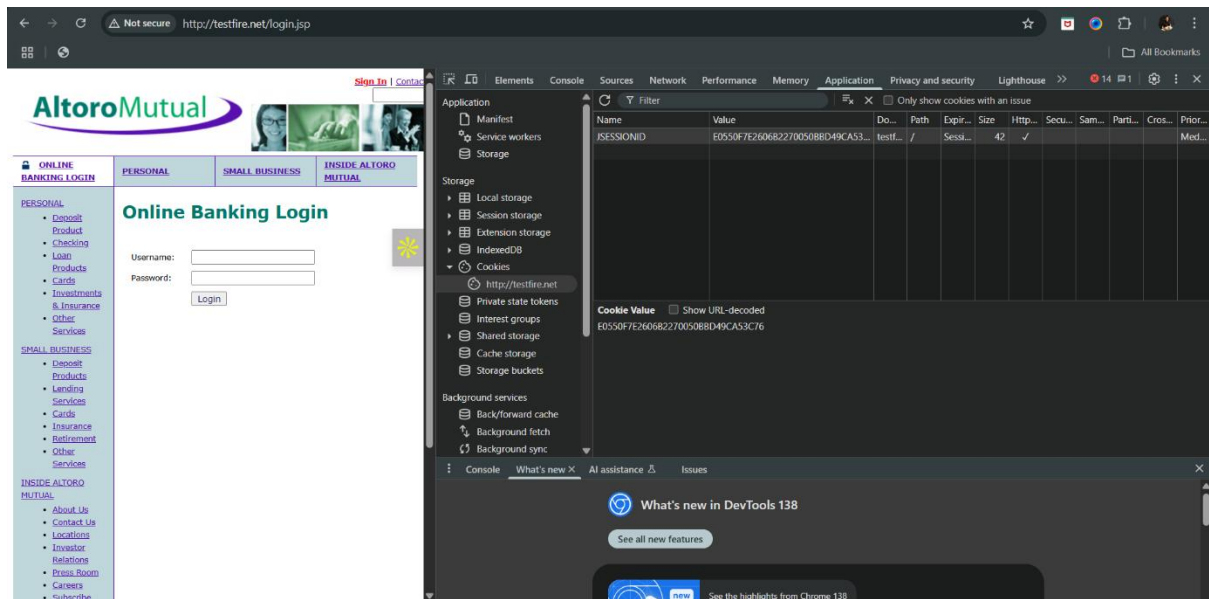
Now, in another OS, enter the same website name: Click on the sign in button, following window will appear



Right click on the website and then click on the 'inspect', then click on the 'applications' option, and paste the session id which we had copied earlier.



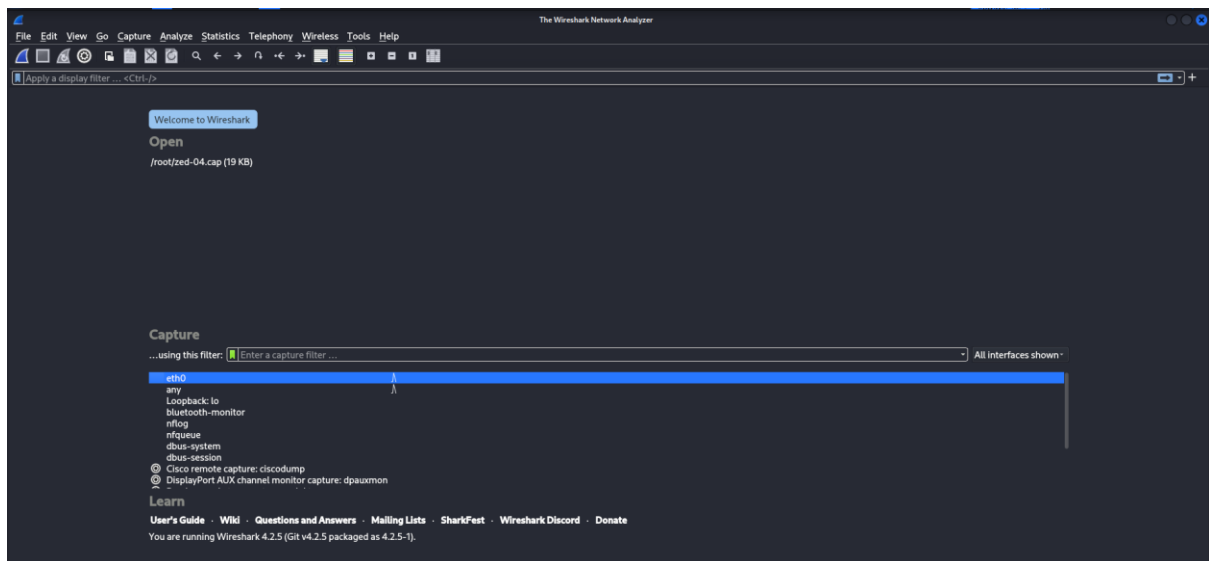
After pasting:



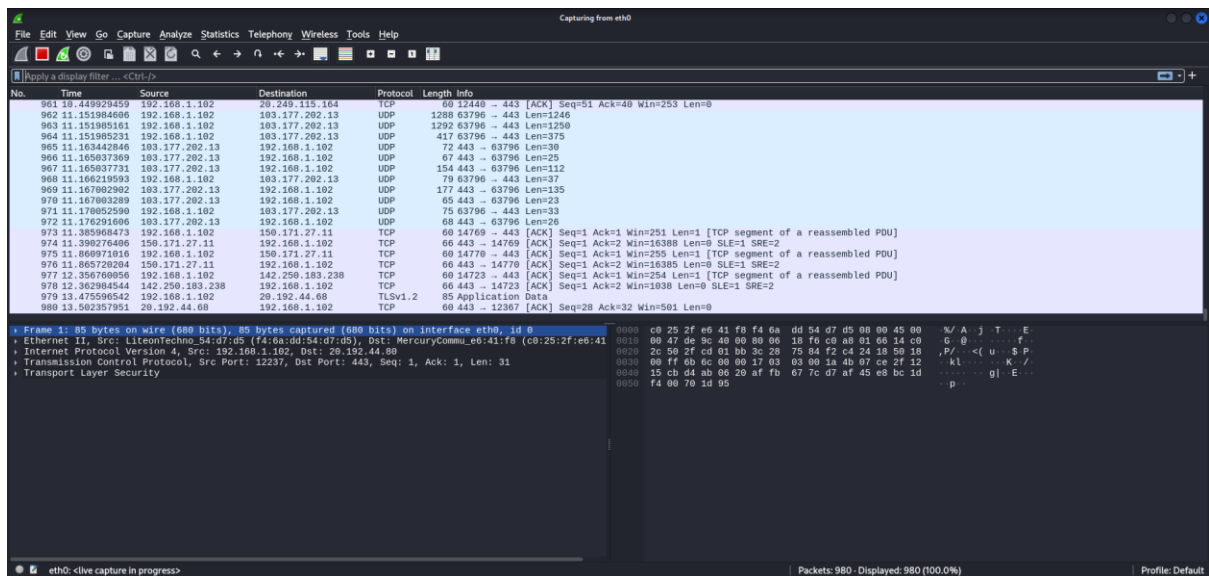
Reload the website and see that we are logged in without entering any password.

## Taking credentials using the Wireshark:

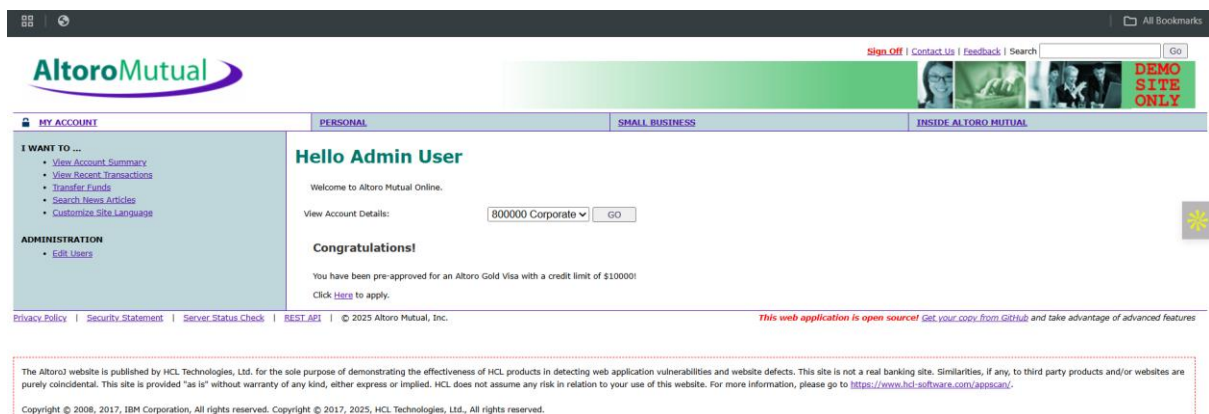
Open Wireshark:



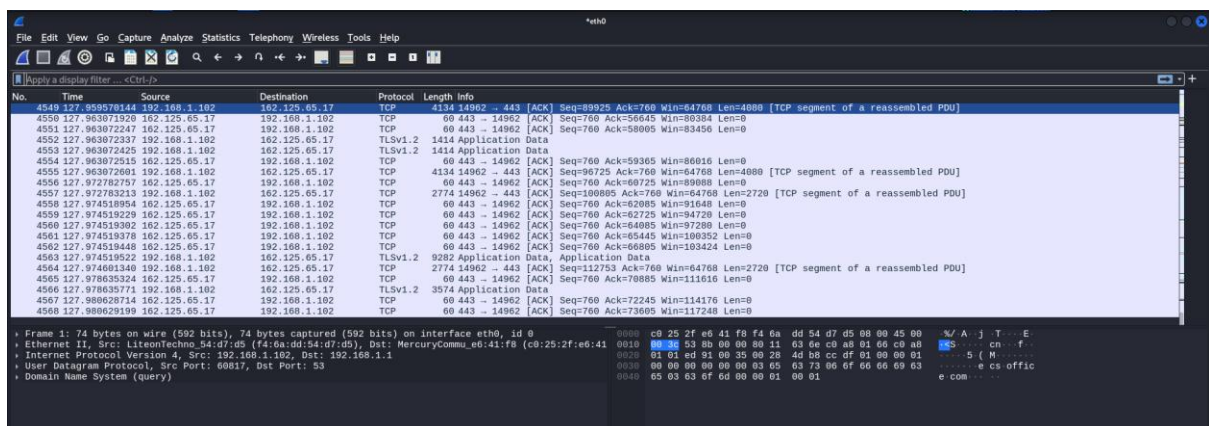
Click on the interface name: 'eth0' in my case.



Go to the website and reload it. Then click on the sign in button: enter the credentials

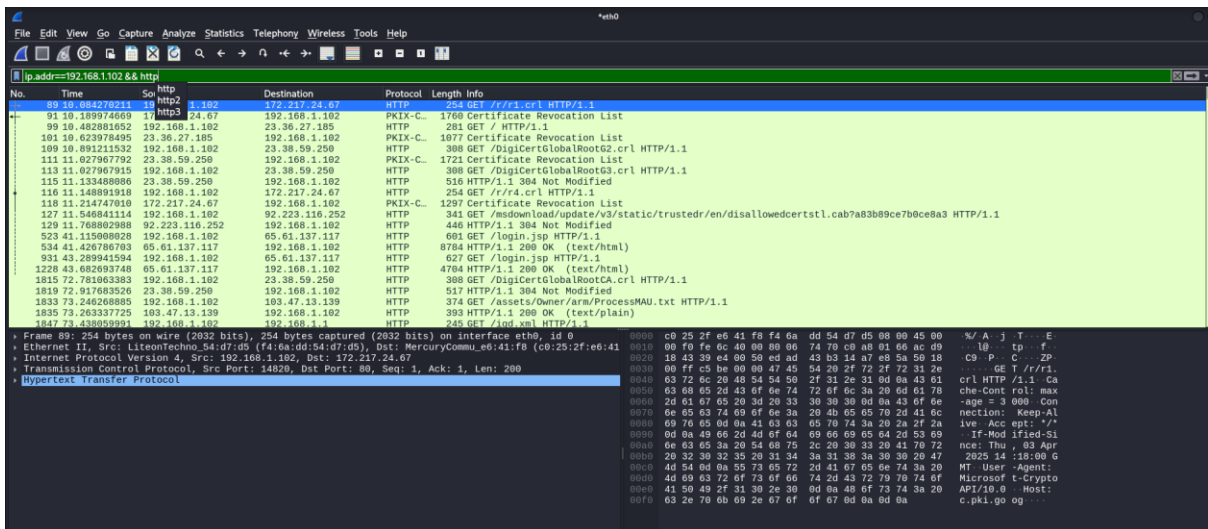


Now, stop capturing packets in the wireshark using the red button:

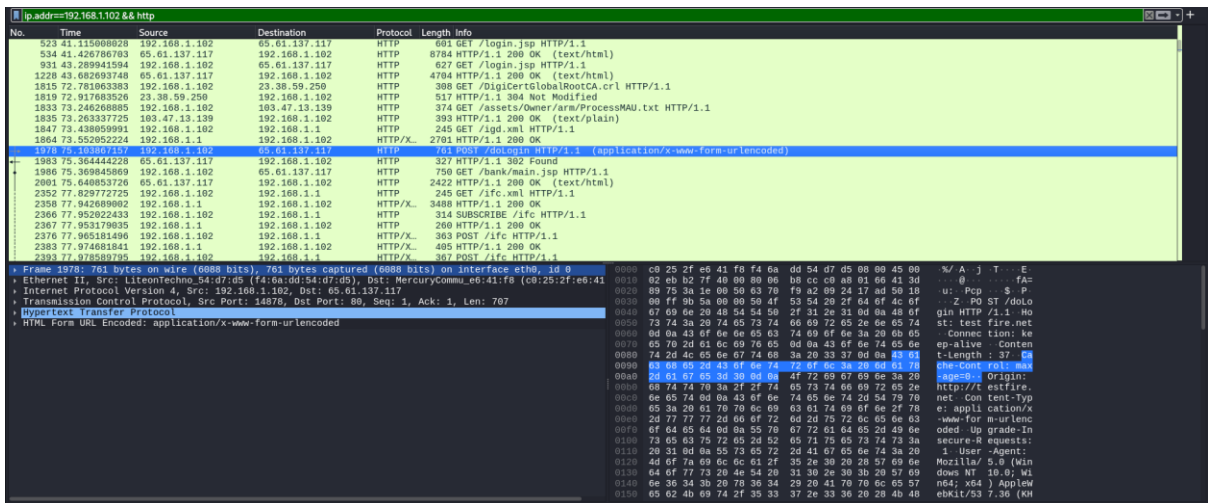


Then, apply the filter as follows: the ip address should be of the victim

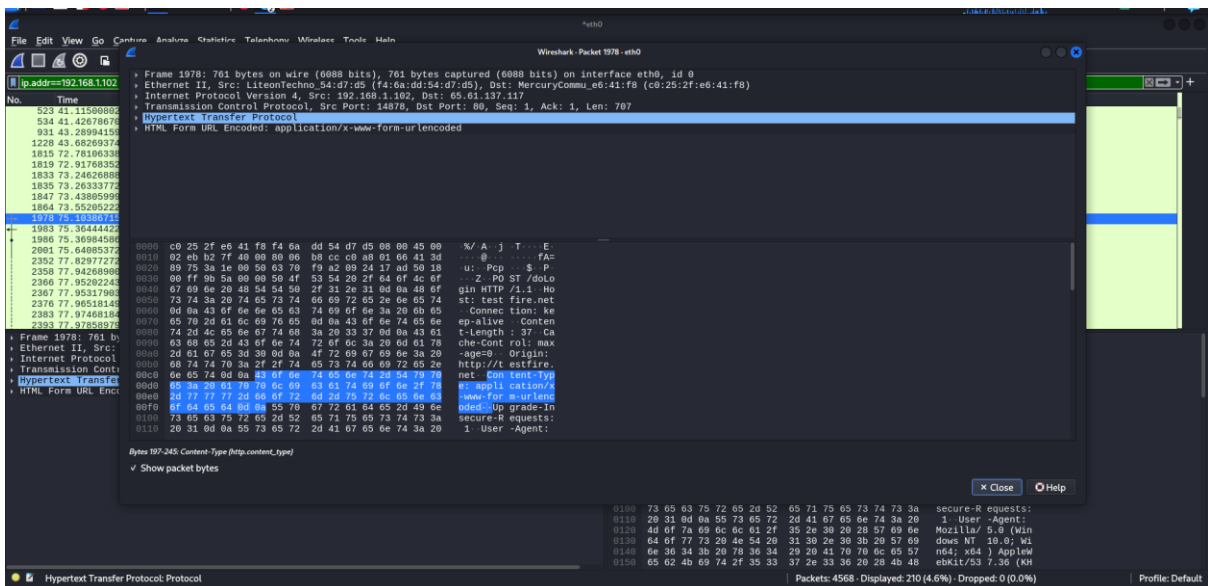




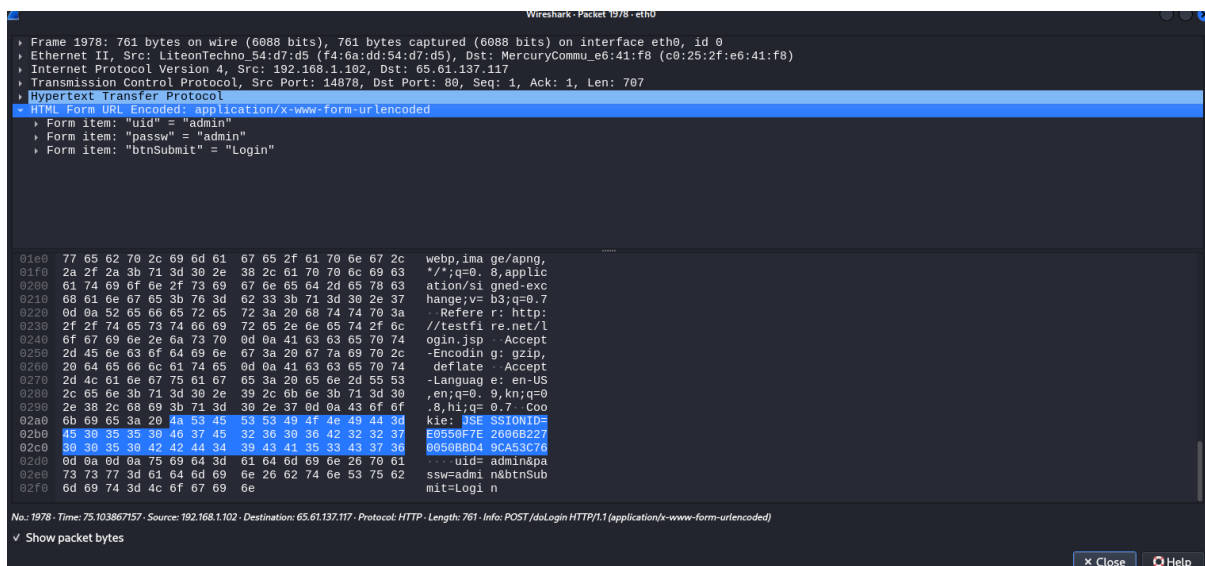
Now, scroll these packets and look for the ‘post’ method: we got it.



Double click it and open it: following window will appear.



Click on the last listed option: we can see the credentials we had entered.



The image shows a Wireshark packet capture window. The top pane shows the packet list with 'Frame 1978: 761 bytes on wire (6088 bits), 761 bytes captured (6088 bits) on interface eth0, id 0' selected. The middle pane shows the packet details for 'Hypertext Transfer Protocol' with 'HTML Form URL Encoded: application/x-www-form-urlencoded' expanded, showing form items: 'uid' = 'admin', 'passw' = 'admin', and 'btnSubmit' = 'Login'. The bottom pane shows the raw packet bytes in hexadecimal and ASCII. The ASCII column contains the following text: 'webp:ima ge/apng, \*//?;q=0.8,applic ation/si gned-exc hange;v= b3;q=0.7 . Referer: http: //testfi re.net/l ogin.jsp - Accept -Encoding: gzip, deflate -Accept -Languag e: en-US ,en;q=0.9,kn;q=0 .8,hi;q= 0.7' Coe kie: PSE SSIONID= B0558F7E 2606B227 0050BBD0 9CA53C78 uid= admin&pa ssw=admin &btnSub mit=Logi n'.

```
No.: 1978 - Time: 75.103867157 - Source: 192.168.1.102 - Destination: 65.61.137.117 - Protocol: HTTP - Length: 761 - Info: POST /doLogin HTTP/1.1 (application/x-www-form-urlencoded)
Show packet bytes
```

## How to protect against session hijacking?

1. Use HTTPS: Encrypt all communication to prevent session ID leakage over the network.
2. Set Secure flag on cookies: Ensures cookies are sent only over HTTPS.
3. Set HttpOnly flag on cookies: Prevents client-side scripts from accessing session cookies.
4. Use SameSite attribute: Helps protect against CSRF by controlling cross-origin requests.
5. Regenerate session IDs on login: Prevents session fixation attacks.
6. Implement session timeout: Automatically expires sessions after inactivity.
7. Bind session to IP/User-Agent: Helps detect session theft or reuse.
8. Use strong, random session IDs: Makes guessing or brute-forcing difficult.
9. Monitor and alert on abnormal activity: Detects possible hijacking attempts.

--The End--