🔓 Lab#03 : Wireshark Lab: Capturing Cleartext Login with Local HTTP Server

🎯 Objectives:

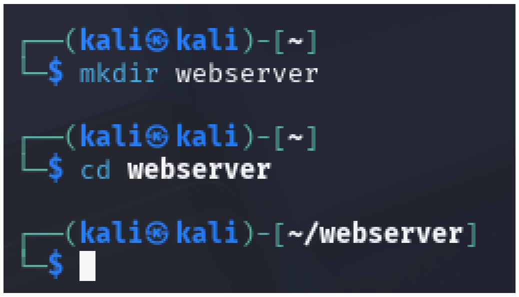
Utilizing Wireshark on a local server to capture cleartext credentials.

Step-by-Step Instructions / Summary

- Part-1: Setup a local HTTP Server with Login Form
- Part-2: Start the server
- Part-3: Start Wireshark Capture
- Part-4: Filter in Wireshark

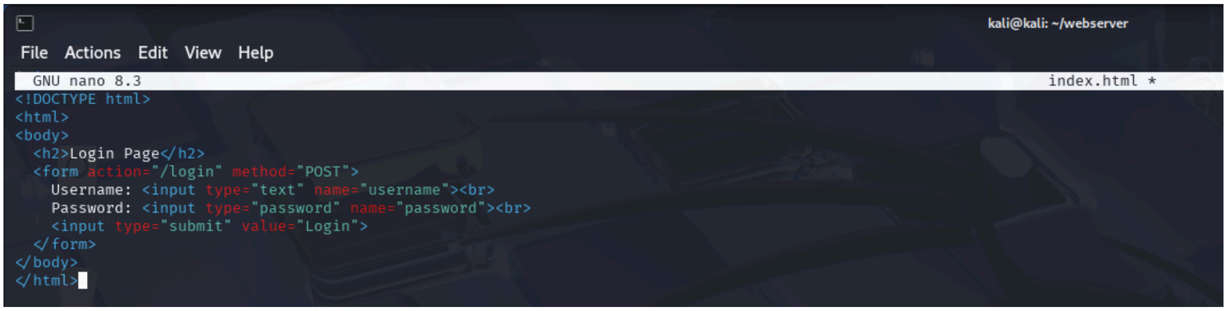
1. Setup a local HTTP Server with Login Form

1.1. Create a project folder

mkdir ~/webserver **cd ~/webserver**	 ┌──(kali㉿kali)-[~] └─$ mkdir webserver ┌──(kali㉿kali)-[~] └─$ cd webserver ┌──(kali㉿kali)-[~/webserver] └─$

1.2. Create Fake Login Page (index.html)

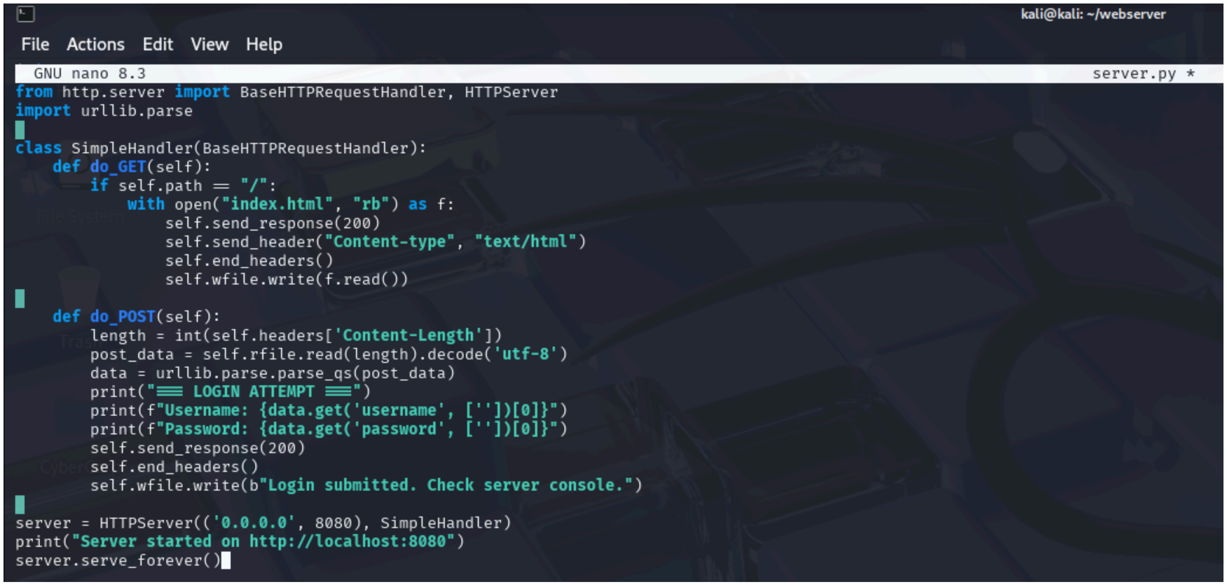
nano index.html



```
  GNU nano 8.3                                                              index.html *
<!DOCTYPE html>
<html>
<body>
  <h2>Login Page</h2>
  <form action="/login" method="POST">
    Username: <input type="text" name="username"><br>
    Password: <input type="password" name="password"><br>
    <input type="submit" value="Login">
  </form>
</body>
</html>
```

1.3. Create Python Web Server Script (server.py)

nano server.py



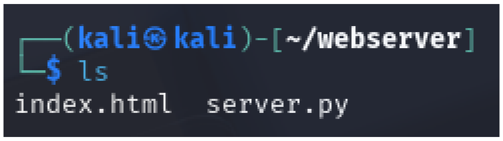
```python
from http.server import BaseHTTPRequestHandler, HTTPServer
import urllib.parse

class SimpleHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == "/":
            with open("index.html", "rb") as f:
                self.send_response(200)
                self.send_header("Content-type", "text/html")
                self.end_headers()
                self.wfile.write(f.read())

    def do_POST(self):
        length = int(self.headers['Content-Length'])
        post_data = self.rfile.read(length).decode('utf-8')
        data = urllib.parse.parse_qs(post_data)
        print("=== LOGIN ATTEMPT ===")
        print(f"Username: {data.get('username', [''])[0]}")
        print(f"Password: {data.get('password', [''])[0]}")
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Login submitted. Check server console.")

server = HTTPServer(('0.0.0.0', 8080), SimpleHandler)
print("Server started on http://localhost:8080")
server.serve_forever()
```

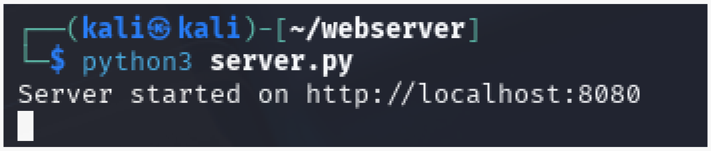
After the two files are saved, we'll use them for the login page.



```
┌──(kali㉿kali)-[~/webserver]
└─$ ls
index.html  server.py
```

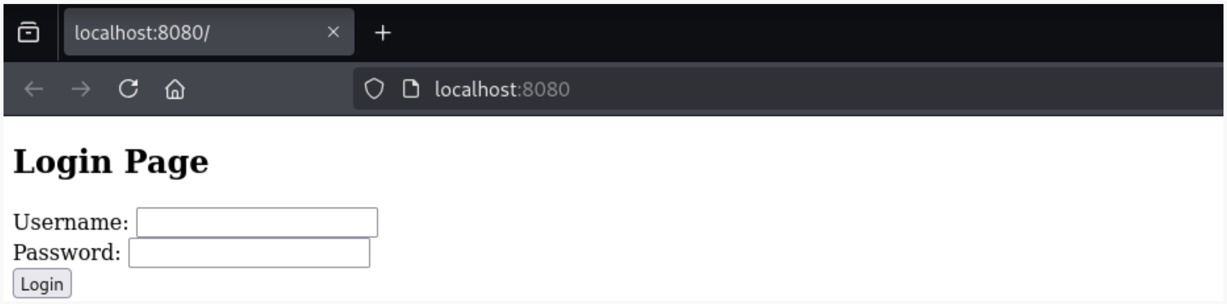
2. Start the server

```
python3 server.py
```



```
┌──(kali㉿kali)-[~/webserver]
└─$ python3 server.py
Server started on http://localhost:8080
```

```
We'll then go to use that login page and it pops up like this:
```



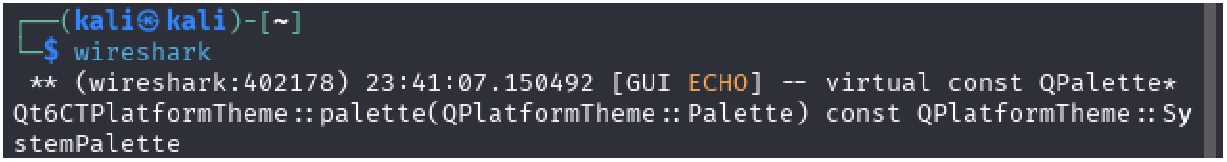
Login Page

Username: ▭
Password: ▭
Login

3. Start Wireshark Capture

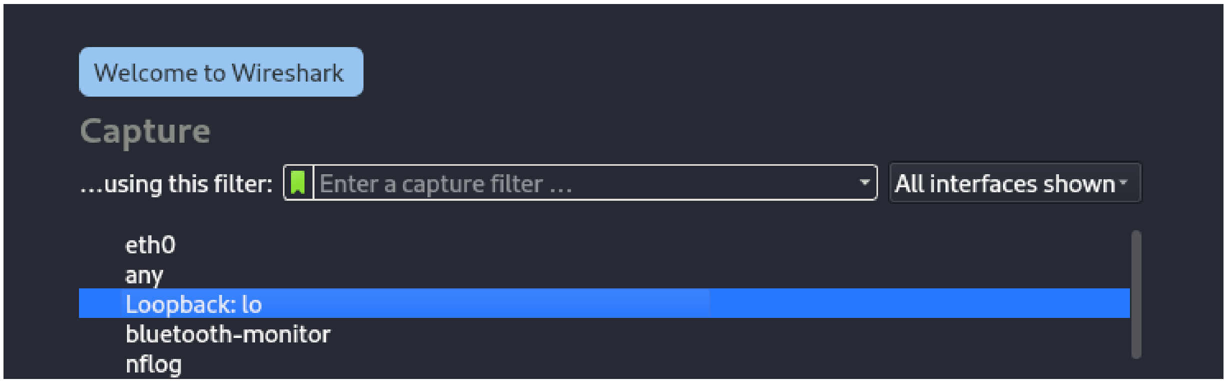
3.1. Launch Wireshark

First is launching Wireshark through the terminal

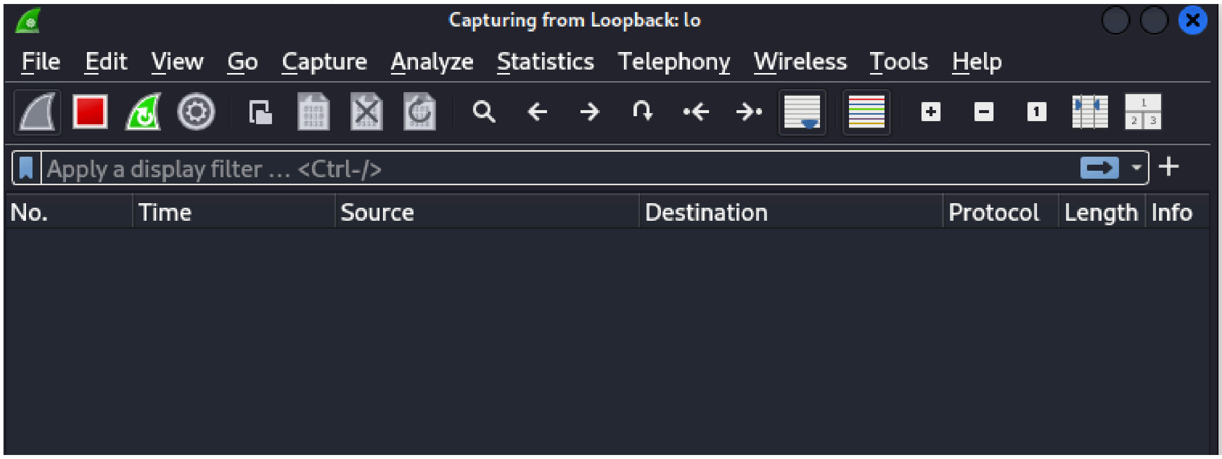


```
┌──(kali㉿kali)-[~]
└─$ wireshark
 ** (wireshark:402178) 23:41:07.150492 [GUI ECHO] -- virtual const QPalette*
Qt6CTPlatformTheme::palette(QPlatformTheme::Palette) const QPlatformTheme::Sy
stemPalette
```

Afterwards we'll select the *IO interface* to capture the credentials from the webserver

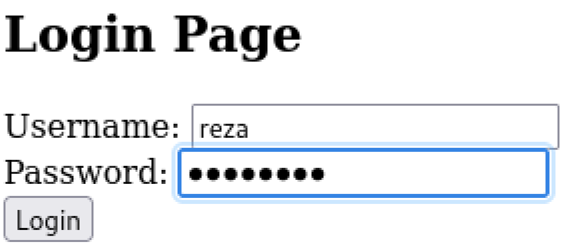


There is currently no login attempts yet, so we'll have to enter them to see the traffic



3.2. Visit the Login Page in Browser

Coming back to the Login Page.
http://localhost:8080
This is the credentials that will be used:
username: reza
password: test@123



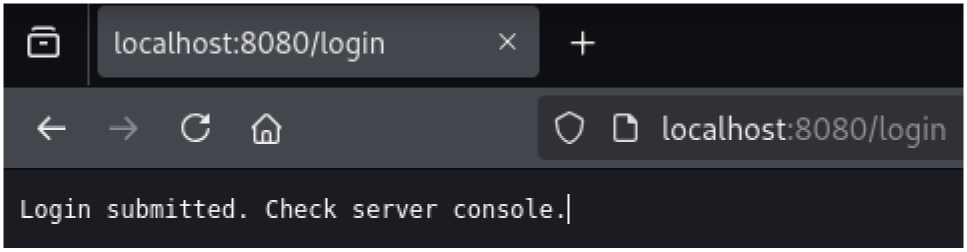
Login Page

Username: reza

Password: ●●●●●●●●●

Login

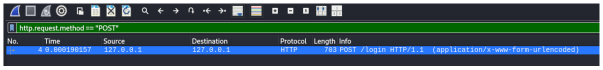
We have successfully logged in and now it’s time to filter out the traffic



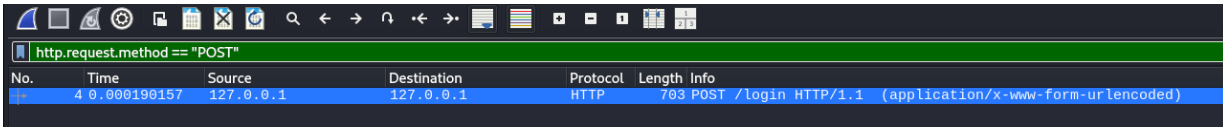
```
Login submitted. Check server console.
```

4. Filter in Wireshark

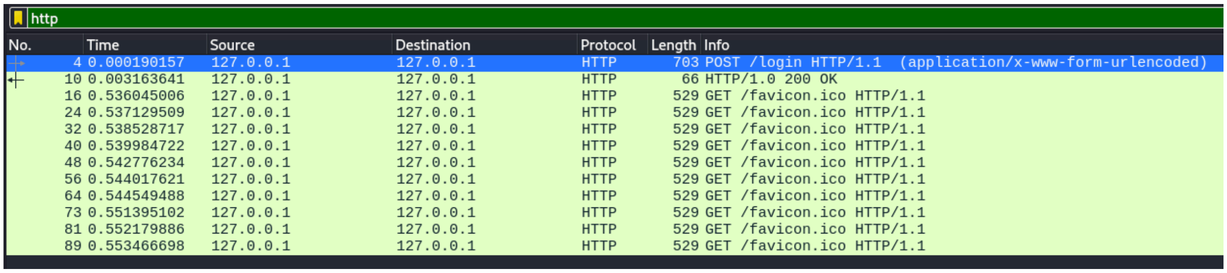
4.1. Using the following filters in order

Filter	**What it does**
http.request.method == "POST"	Shows POST requests only  http.request.method == "POST" No. Time Source Destination Protocol Length Info 4 0.000190157 127.0.0.1 127.0.0.1 HTTP 703 POST /login HTTP/1.1 (application/x-www-form-urlencoded)
http	Shows all HTTP traffic
tcp.port == 8080	Shows all packets on port 8080
frame contains "username"	Searches for keyword "username" in frames

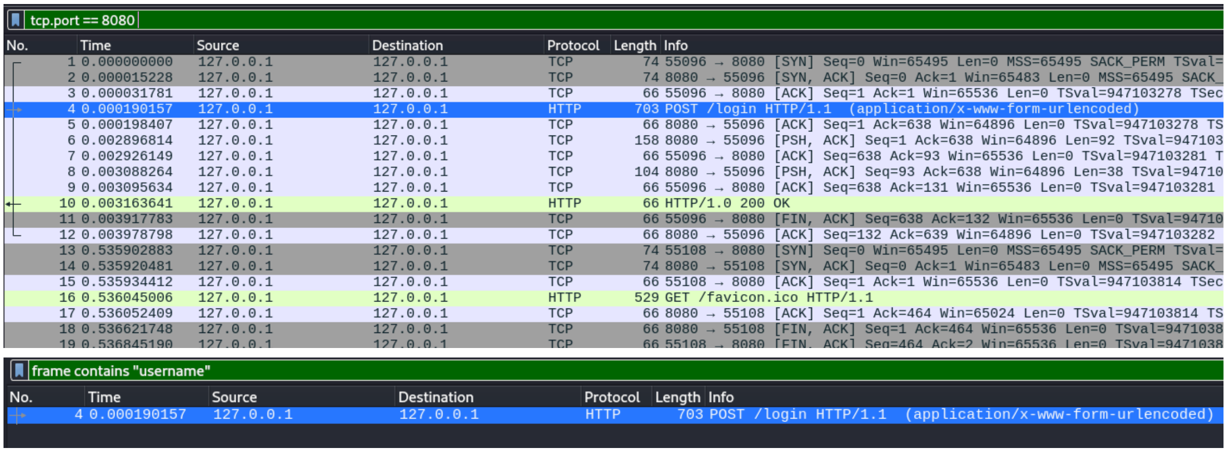
Filters in order:



```
http.request.method == "POST"
No.    Time         Source       Destination   Protocol  Length Info
     4 0.000190157  127.0.0.1    127.0.0.1     HTTP         703 POST /login HTTP/1.1  (application/x-www-form-urlencoded)
```



```
http
No.    Time         Source       Destination   Protocol  Length Info
     4 0.000190157  127.0.0.1    127.0.0.1     HTTP         703 POST /login HTTP/1.1  (application/x-www-form-urlencoded)
    10 0.003163641  127.0.0.1    127.0.0.1     HTTP          66 HTTP/1.0 200 OK
    16 0.536045006  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
    24 0.537129509  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
    32 0.538528717  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
    40 0.539984722  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
    48 0.542776234  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
    56 0.544017621  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
    64 0.544549488  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
    73 0.551395102  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
    81 0.552179886  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
    89 0.553466698  127.0.0.1    127.0.0.1     HTTP         529 GET /favicon.ico HTTP/1.1
```

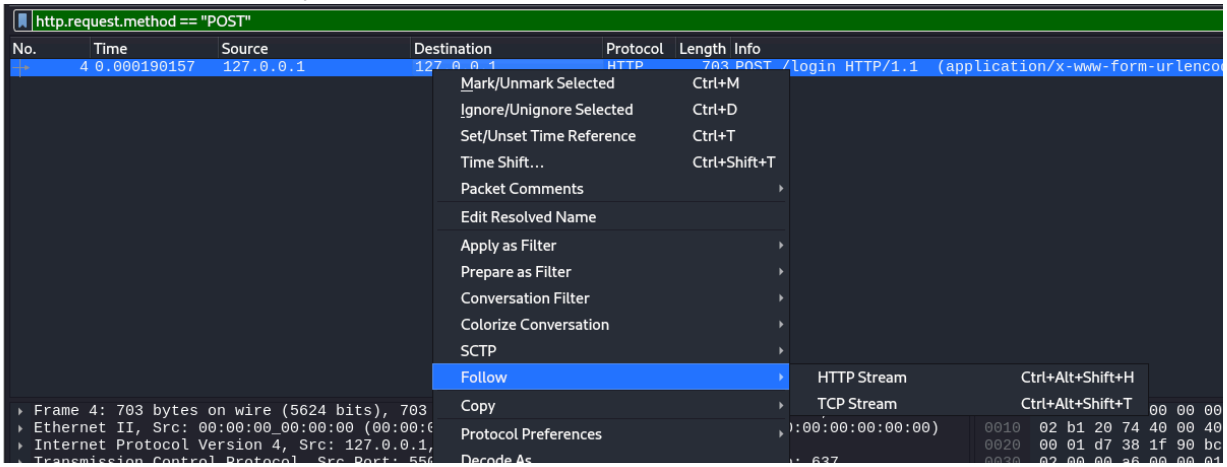


```
tcp.port == 8080
No.  Time        Source     Destination  Protocol  Length Info
   1 0.000000000 127.0.0.1  127.0.0.1    TCP        74 55096 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=
   2 0.000015228 127.0.0.1  127.0.0.1    TCP        74 8080 → 55096 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_
   3 0.000031781 127.0.0.1  127.0.0.1    TCP        66 55096 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=947103278 TSec
   4 0.000190157 127.0.0.1  127.0.0.1    HTTP      703 POST /login HTTP/1.1  (application/x-www-form-urlencoded)
   5 0.000198407 127.0.0.1  127.0.0.1    TCP        66 8080 → 55096 [ACK] Seq=1 Ack=638 Win=64896 Len=0 TSval=947103278 TS
   6 0.002896814 127.0.0.1  127.0.0.1    TCP       158 8080 → 55096 [PSH, ACK] Seq=1 Ack=638 Win=64896 Len=92 TSval=947103
   7 0.002926149 127.0.0.1  127.0.0.1    TCP        66 55096 → 8080 [ACK] Seq=638 Ack=93 Win=65536 Len=0 TSval=947103281 T
   8 0.003088264 127.0.0.1  127.0.0.1    TCP       104 8080 → 55096 [PSH, ACK] Seq=93 Ack=638 Win=64896 Len=38 TSval=94710
   9 0.003095634 127.0.0.1  127.0.0.1    TCP        66 55096 → 8080 [ACK] Seq=638 Ack=131 Win=65536 Len=0 TSval=947103281
  10 0.003163641 127.0.0.1  127.0.0.1    HTTP       66 HTTP/1.0 200 OK
  11 0.003917783 127.0.0.1  127.0.0.1    TCP        66 55096 → 8080 [FIN, ACK] Seq=638 Ack=132 Win=65536 Len=0 TSval=94710
  12 0.003978798 127.0.0.1  127.0.0.1    TCP        66 8080 → 55096 [ACK] Seq=132 Ack=639 Win=64896 Len=0 TSval=947103282
  13 0.535902883 127.0.0.1  127.0.0.1    TCP        74 55108 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=
  14 0.535920481 127.0.0.1  127.0.0.1    TCP        74 8080 → 55108 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_
  15 0.535934412 127.0.0.1  127.0.0.1    TCP        66 55108 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=947103814 TSec
  16 0.536045006 127.0.0.1  127.0.0.1    HTTP      529 GET /favicon.ico HTTP/1.1
  17 0.536052409 127.0.0.1  127.0.0.1    TCP        66 8080 → 55108 [ACK] Seq=1 Ack=464 Win=65024 Len=0 TSval=947103814 TS
  18 0.536621748 127.0.0.1  127.0.0.1    TCP        66 8080 → 55108 [FIN, ACK] Seq=1 Ack=464 Win=65536 Len=0 TSval=9471038
  19 0.536845190 127.0.0.1  127.0.0.1    TCP        66 55108 → 8080 [FIN, ACK] Seq=464 Ack=2 Win=65536 Len=0 TSval=9471038

frame contains "username"
No.  Time        Source     Destination  Protocol  Length Info
   4 0.000190157 127.0.0.1  127.0.0.1    HTTP      703 POST /login HTTP/1.1  (application/x-www-form-urlencoded)
```

4.2. Follow the stream

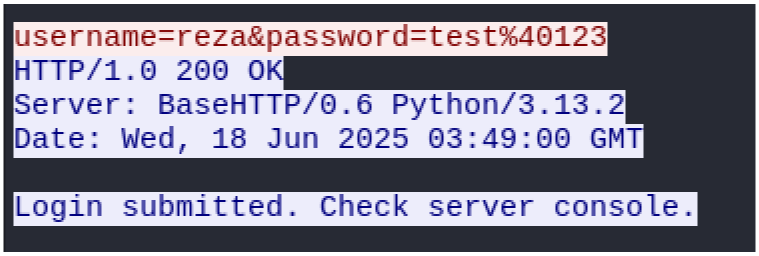
The next step is following the POST packet from the follow > HTTP stream



```
http.request.method == "POST"
No.  Time        Source     Destination  Protocol  Length Info
   4 0.000190157 127.0.0.1  127.0.0.1    HTTP      703 POST /login HTTP/1.1  (application/x-www-form-urlenco

 Frame 4: 703 bytes on wire (5624 bits), 703
 Ethernet II, Src: 00:00:00_00:00:00 (00:00:0
 Internet Protocol Version 4, Src: 127.0.0.1,
 Transmission Control Protocol, Src Port: 550
```

Finally, we can find the credentials here!



```
username=reza&password=test%40123
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.13.2
Date: Wed, 18 Jun 2025 03:49:00 GMT

Login submitted. Check server console.
```

🧰 Tools & Skills Used

- Python3 HTTP Server
- Kali Linux / Linux environment
- HTTP protocol inspection
- Credential sniffing techniques

🧠 Reflection & Takeaways

With this lab, it helped me sharpen my skill using Wireshark filters to capture cleartext credentials, a skill previously used for ctfs such as Cyberdefenders and PicoCTF. I remember specifically applying filters such as http.request.method == "POST" and searching for strings such as "username" reminded me how credentials can be stolen when encryption wasn't implemented.