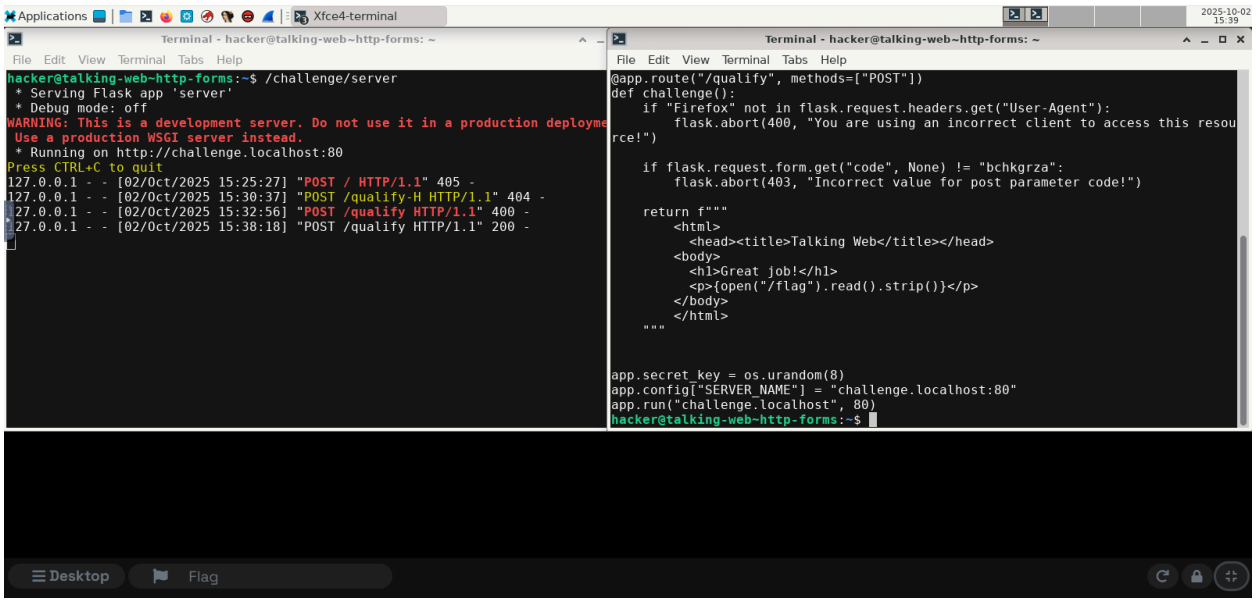
Jaleel Williamson
jayw-713
CSCI 400 Lab 9
10/8/25

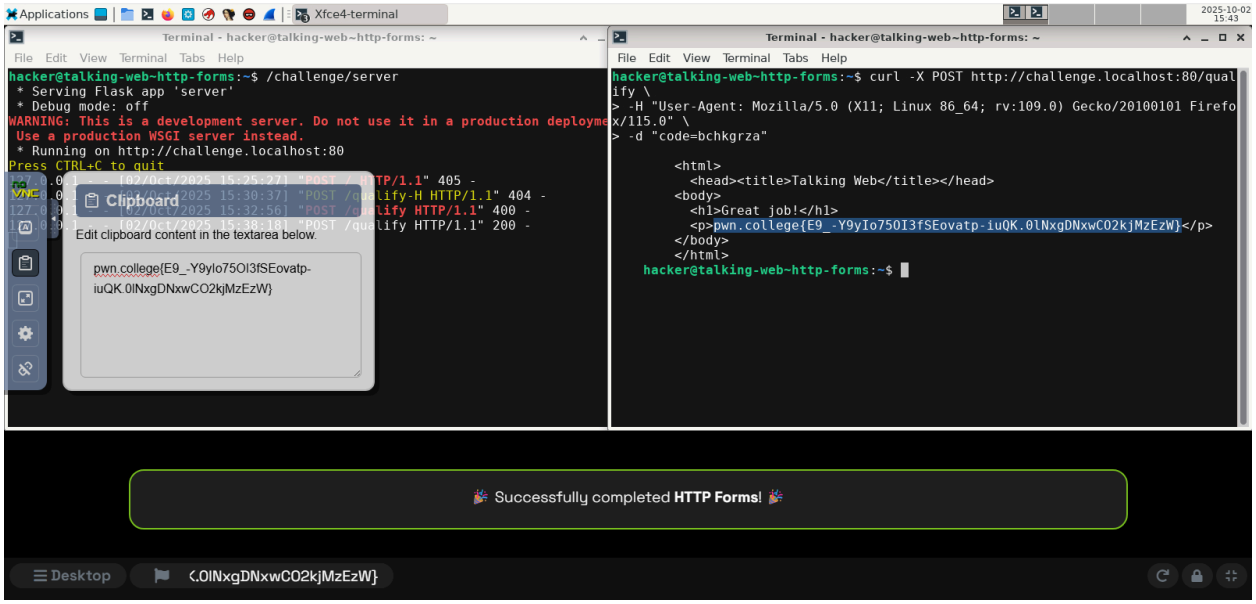
Playing with Programs: Talking Web https://pwn.college/fundamentals/talking-web/

Challenges 'HTTP Forms' through 'HTTP Redirects (Python)' (10 challenges total)

- HTTP Forms

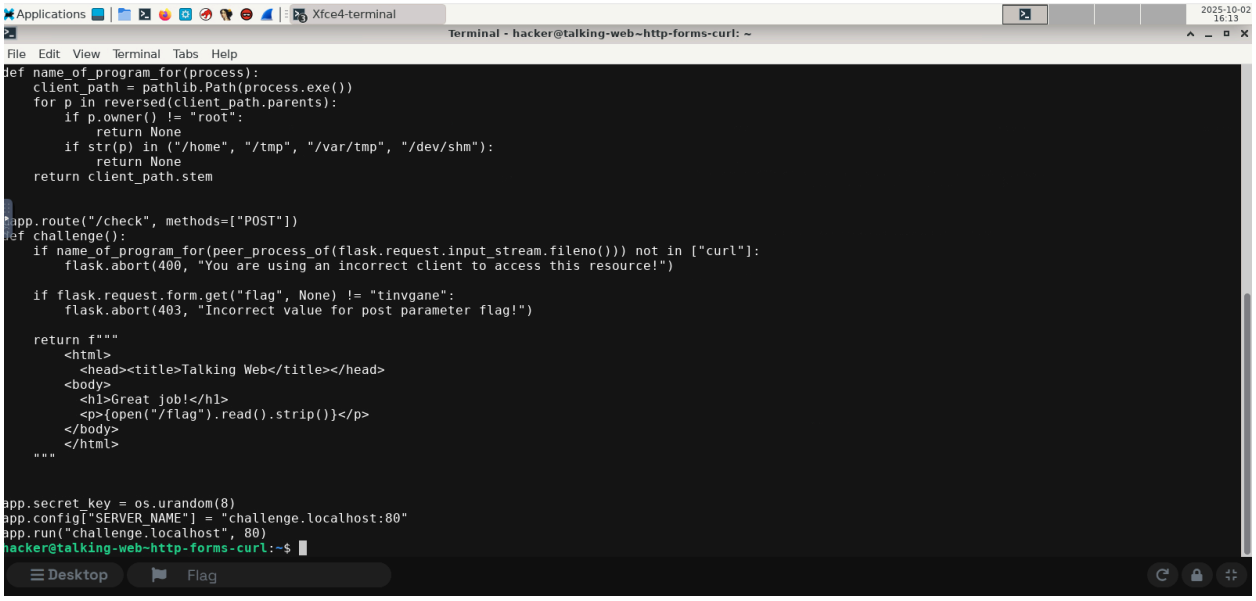


Used cat /challenge/server to get the endpoint of /qualify, and password: code=bchkgrza

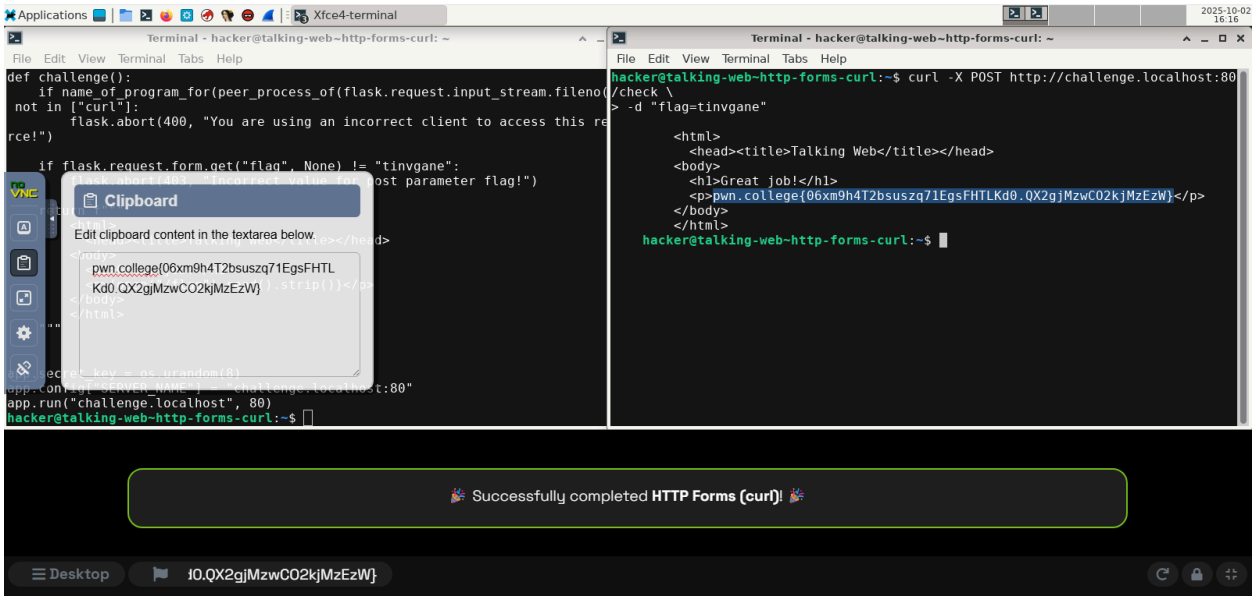


I first verified that the **Flask server** was running on **http://challenge.localhost:80**, as indicated in the terminal output. Knowing that the correct **code** for retrieving the flag was "**bchkgrza**", I prepared a **POST request** to the **/qualify endpoint** using **curl**. I set the **User-Agen**t header to "**Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0**" to mimic a typical web browser request, ensuring the server would process the request correctly. Then, I included the code parameter with the value "bchkgrza" in the POST data. Upon sending the request, the server responded with an **HTML** page containing a "Great job!" message and the flag within a paragraph tag. The flag was displayed as **Pwn.College(E9__Y9y1o7S0i3fSEcwaip-luOK.8UNgDNxwCO2kjMzEzW)**, confirming that I had successfully completed the HTTP Forms challenge. This exercise reinforced that web interactions are fundamentally about a client sending correctly formatted requests to a server, and that tools like curl can be used to manually craft these requests with precision, bypassing the need for a graphical browser.

- HTTP Forms (curl)

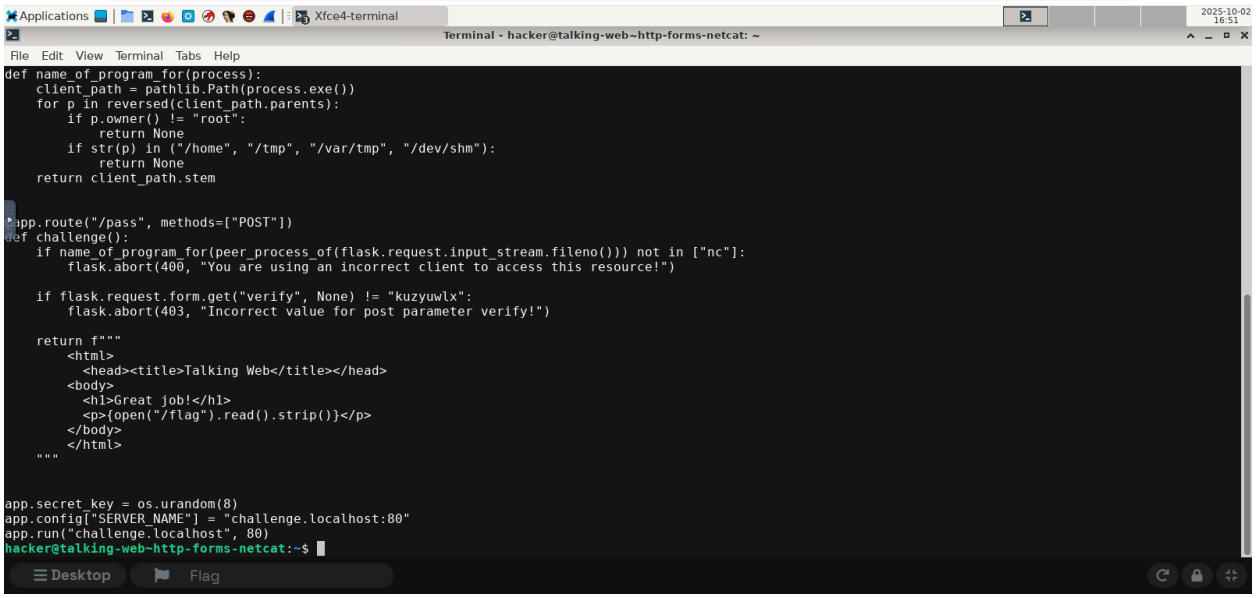


Used cat /challenge/server to get the endpoint of /check, and password: flag=tinvgane

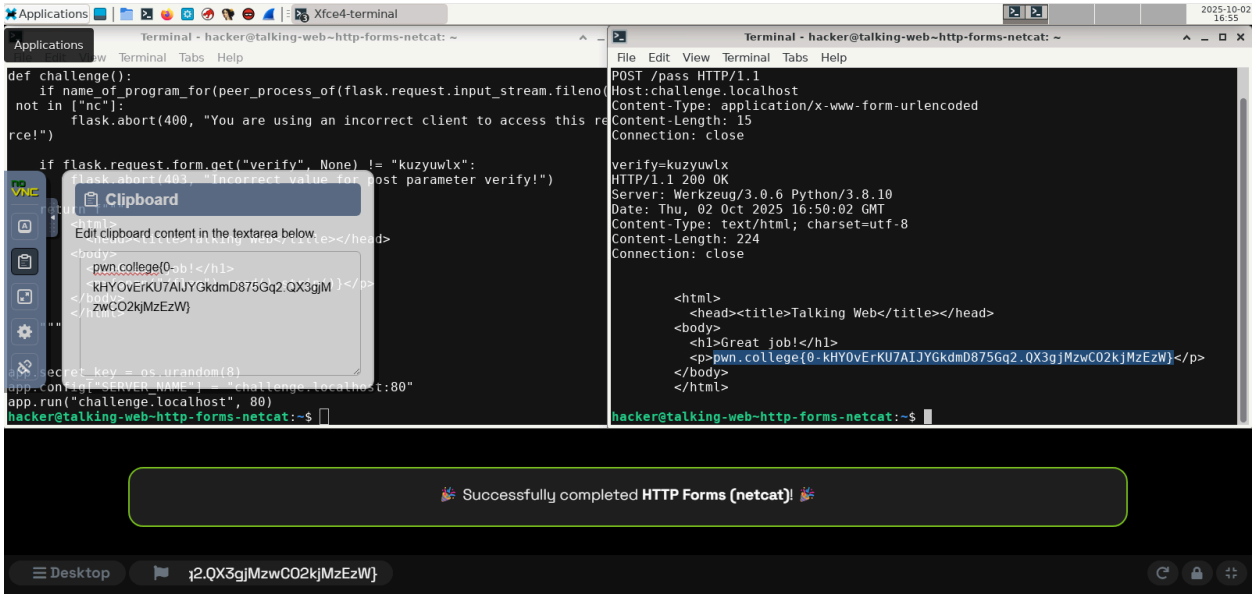


I first verified that the **Flask server** was running on **http://challenge.localhost:80**, as indicated in the terminal output. Then, I used the **curl command** to send a **POST request** to the **/check endpoint** with the following command: **curl -X POST http://challenge.localhost:80/check**. The server responded with an **HTML** page containing a "Great job!" message and the flag within a paragraph tag. The flag was displayed as **pwn.college(06xm9h4T2bsuszq71EgsFHTLKd0.QX2gjMzwCO2kjMzEzW)**, confirming that I had successfully solved the HTTP Forms challenge using curl. This demonstrated curl's effectiveness as a command-line tool for direct HTTP communication, allowing for quick and scriptable interactions with web endpoints without any overhead.

- HTTP Forms (netcat)

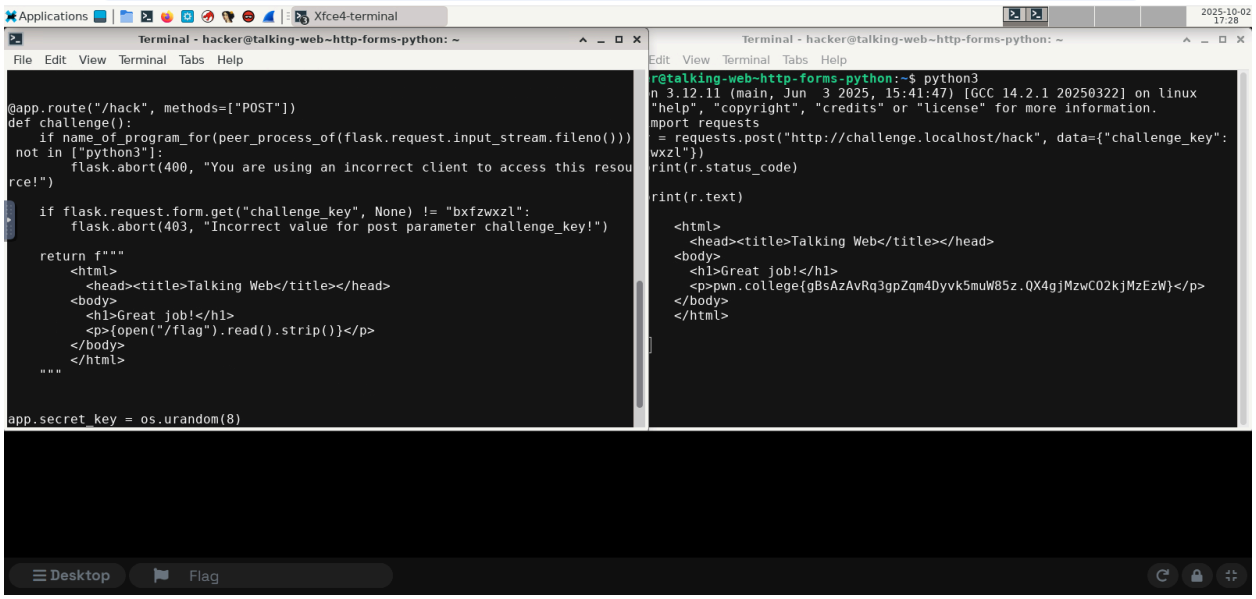


Used cat /challenge/server to get the endpoint of /pass, and password: verify=kuzyuwlx

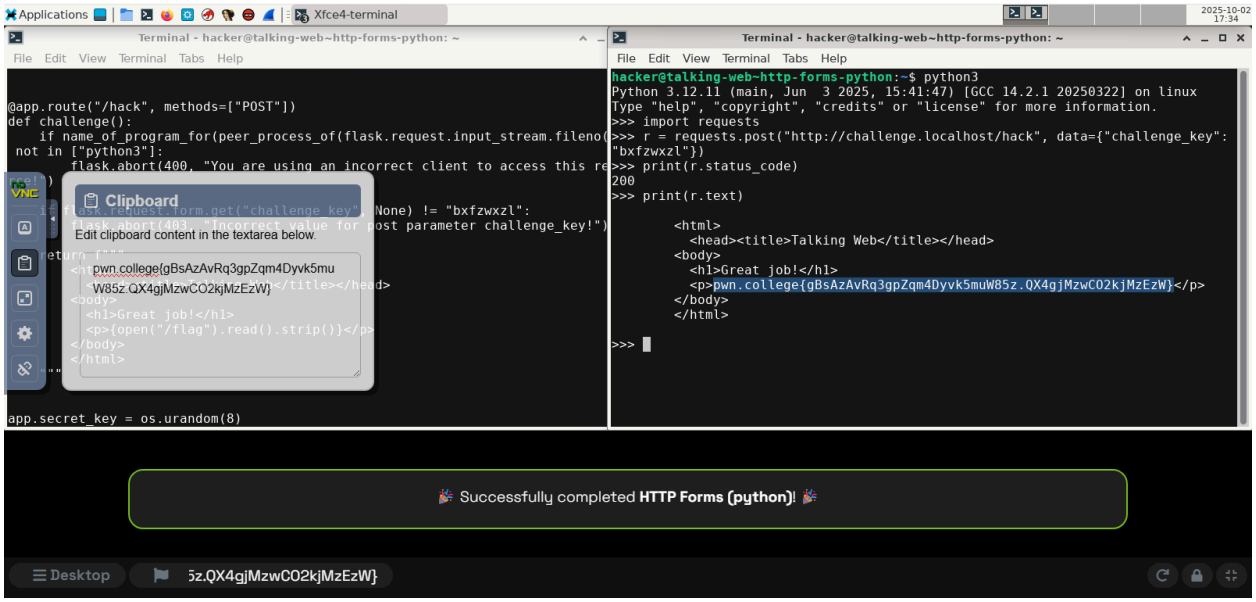


I first connected to the web server using **netcat**, targeting **challenge.localhost** on port 80. I crafted a raw **HTTP POST** request to the **/pass endpoint**, ensuring I included the necessary headers. I set the Host header to challenge.localhost, the **Content-Type** to **application/x-www-form-urlencoded** to indicate form data, and the **Content-Length** to **15**, which matches the length of the body data. I also added **Connection: close** to terminate the connection after the response. The body of the request contained the parameter verify with the **value kuzyuwlx**, which I knew was the correct code based on the challenge requirements. After sending this request, the server responded with an HTTP 200 OK status, confirming success. The response body was an **HTML** page that displayed "Great job!" and included the flag within a paragraph tag: **pwn.college[0-kHY0VErKU7A1JYGkdmD875Gq2.QX3gjMzwC02kjMzEzM]**. This flag indicated that I had successfully completed the challenge by sending the proper POST request with the correct verification code. Using netcat provided a raw, byte-level view of the HTTP protocol, demystifying the actual data exchange that occurs beneath the abstractions of higher-level tools and browsers.

- HTTP Forms (python)



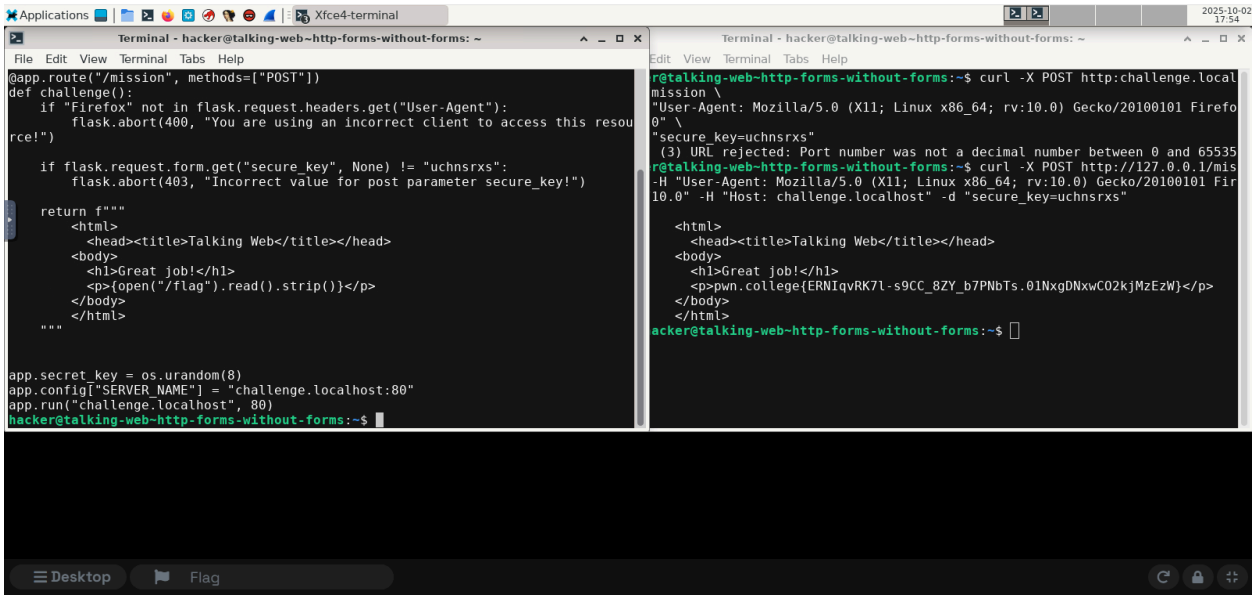
Used cat /challenge/server to get the endpoint of /hack, and password: challenge_key=bxfzwxzl



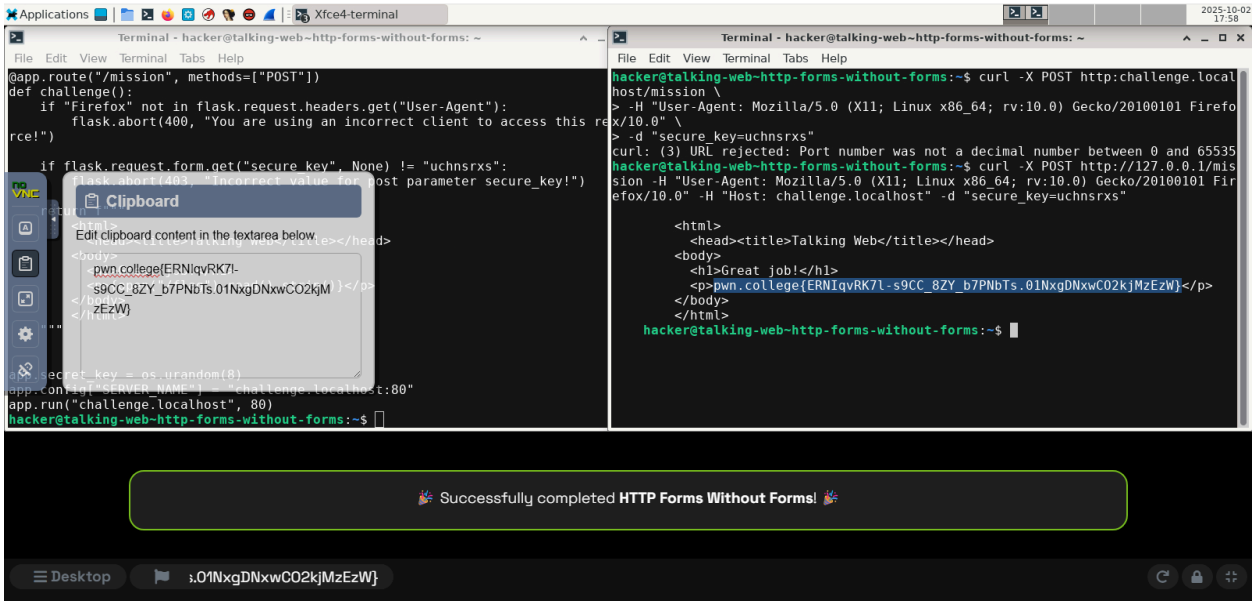
I accessed the web server by starting a **Python 3** interactive session and imported the requests library to handle **HTTP** communication. I then crafted a **POST request** to the endpoint **http://challenge.localhost/hack**, ensuring that the client was identified as Python 3, as required by the server's checks. In the request, I set the form data parameter **challenge_key** to the value **"bxfzwxl"**, which I knew was the correct key from examining the challenge code. After sending the request, I received a **200 OK** status code, indicating success, and the response body contained an **HTML** page with a "Great job!" message and the flag: **pwn.college(gB&AzAVRq3gpZqm40yvk5muW85z.0X4gjMzvCQXsJWzEzM)**. This confirmed that I had successfully met the challenge requirements by using the proper client and

providing the correct parameter. This highlighted the power of using a scripting language like Python for web interactions, which provides immense flexibility for building complex, automated clients that can handle dynamic content and logic.

- HTTP Forms Without Forms



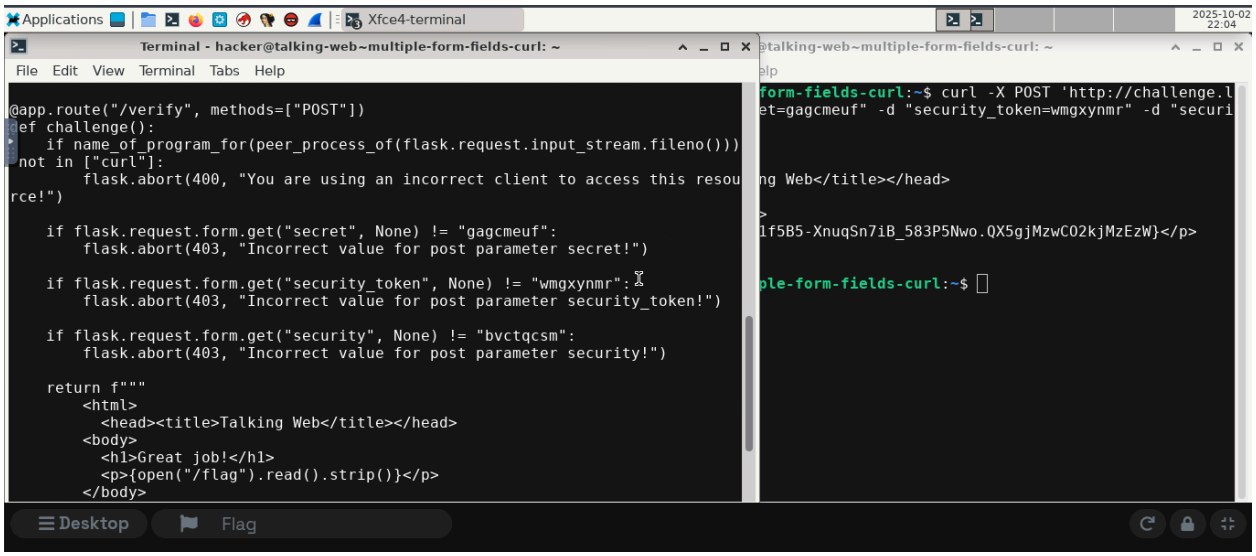
Used cat /challenge/server to get the endpoint of /mission, and password: secure_key=uchnsrxs



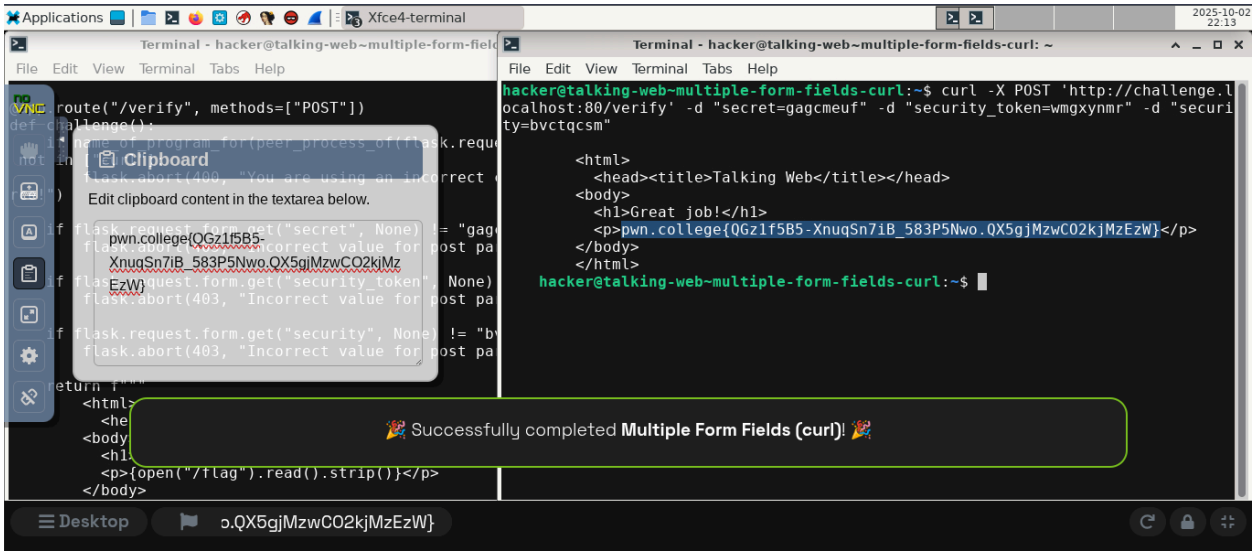
I first analyzed the challenge requirements by examining the server code, which indicated that to retrieve the flag, I needed to send a **POST** request to the **"/mission"** endpoint. The server had two specific checks: the **User-Agent header** must contain **"Firefox"**, and the **POST** data must include a **"secure_key"** parameter with the value **"uchnsrxs"**.
I used curl to craft the request, but my initial attempt failed due to a URL issue. I then corrected it by targeting **http://127.0.0.1/mission** and set the **Host header** to **"challenge.localhost"** to

ensure proper routing. I set the **User-Agent header to "Mozilla/5.0 (X11; Linux x86_64; rv:10.0) Gecko/20100101 Firefox/10.0"** to satisfy the client requirement, and I included the form data with "secure_key=uchnsrxs". The server responded with a **200 OK** status and an **HTML** page that displayed "Great job!" along with the flag in the paragraph tag: **pwn.college(gB&AzAVRq3gpZqm40yvk5muW85z.0X4gjMzvCQXsJWzEzM)**. This confirmed that I had successfully met both conditions and retrieved the flag. This challenge was a crucial lesson in understanding that server-side logic often enforces specific client behaviors, and that "form" data is simply a convention for POST parameters that can be sent independently of an actual HTML form.

- Multiple Form Fields (curl)

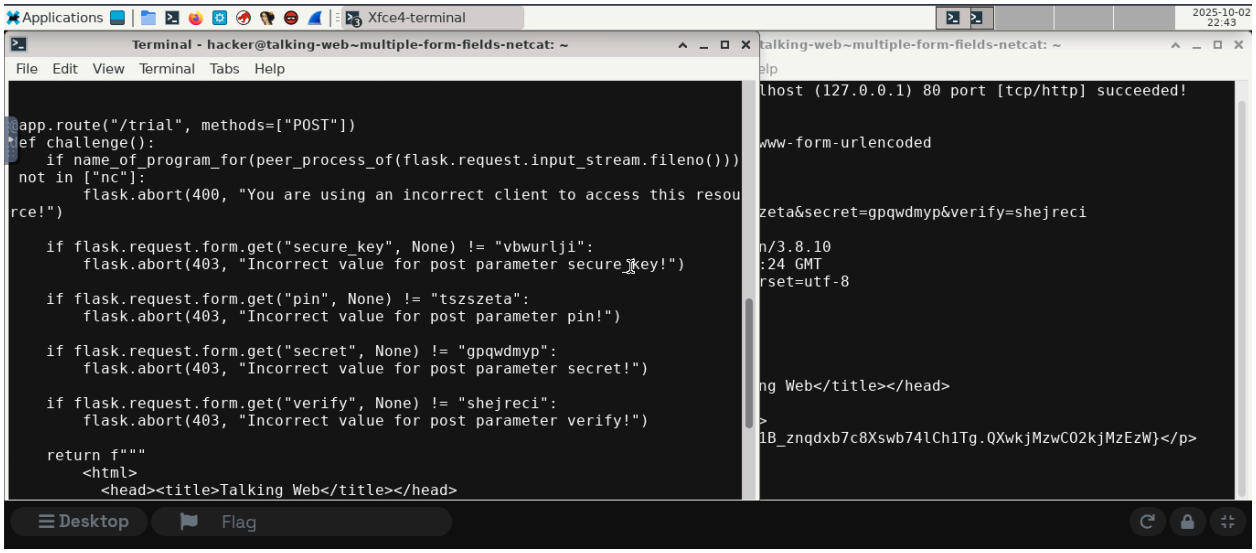


Used cat /challenge/server to get the endpoint of /verify, passwords: secret=gagcmeuf, security_token=wmgxynmr, and security=bvctqcsm

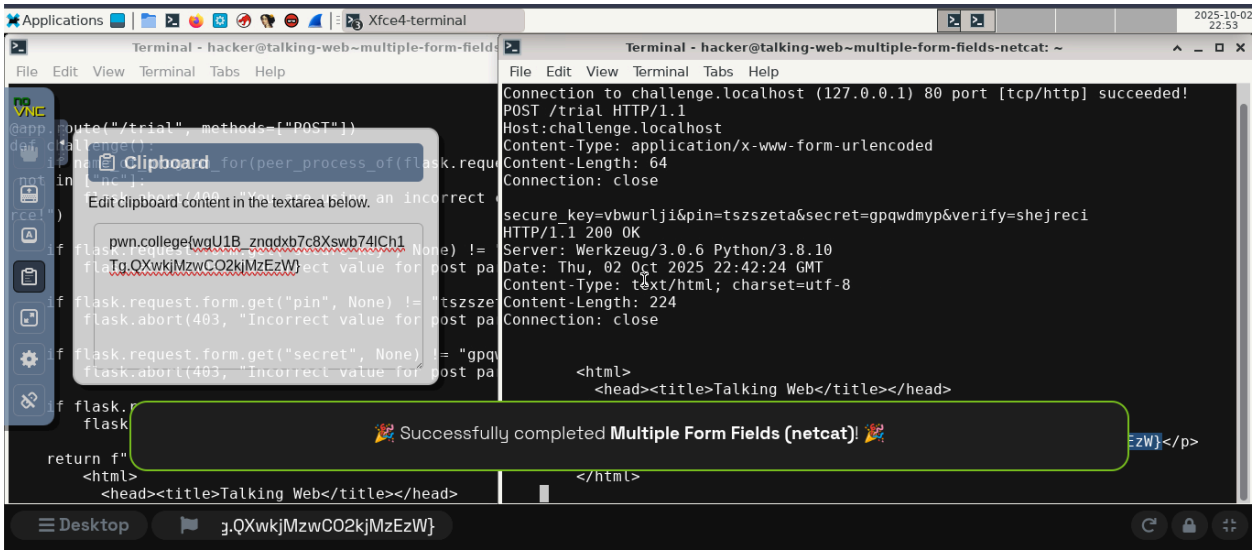


I first recognized that the server required a **POST** request to the **"/verify"** endpoint and that it specifically checked for the client being **"curl"** to avoid errors. To meet this requirement, I used the curl command to send the request. I targeted the correct URL, **http://challenge.localhost:80/verify**, and included the three necessary form fields with their exact values: **"secret=gagcmcut"**, **"security_token=wmgxynnr"**, and **"security=bvtqcsm"**. After executing the command, the server responded with an **HTML** page containing a "Great job!" message and the flag: **pwn.college[0Gz1f5B5-XnuqSn71B_583P5Nwo.0X5gjWzwC02kjWzEzW]**. This confirmed that I had successfully provided all the correct parameters and retrieved the flag. Success here depended on meticulously providing every required parameter, illustrating how server-side validation works and how easy it is to manipulate multiple form fields directly from the command line.

- Multiple Form Fields (netcat)

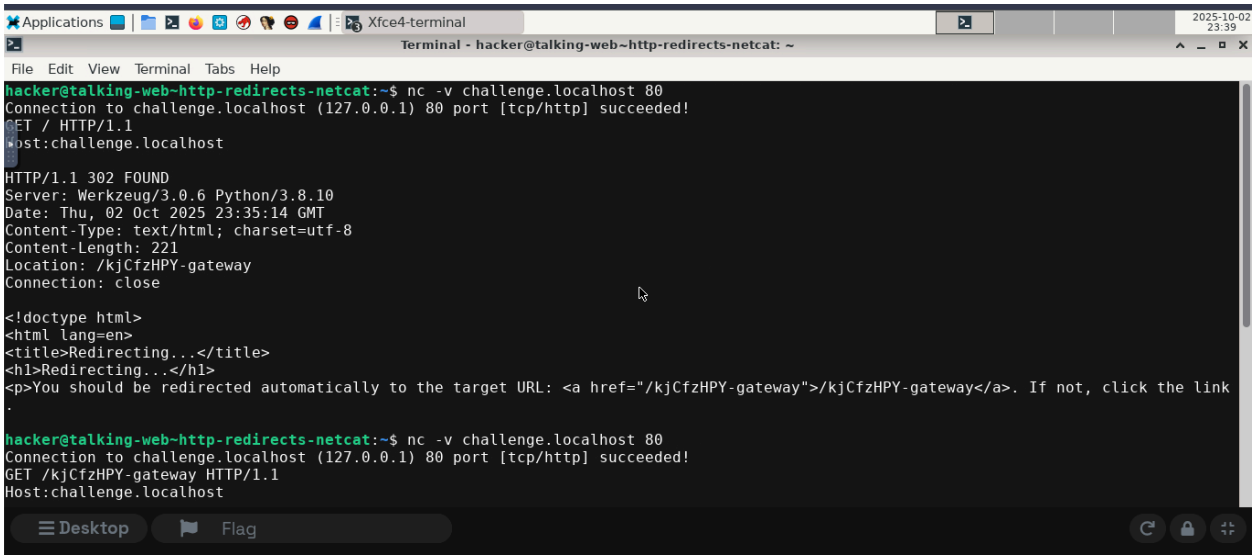


Used cat /challenge/server to get the endpoint of /trial, passwords: secure_key=vbwurlji, pin=tszszeta, secret=gpqwdmyp, verify=shejreci

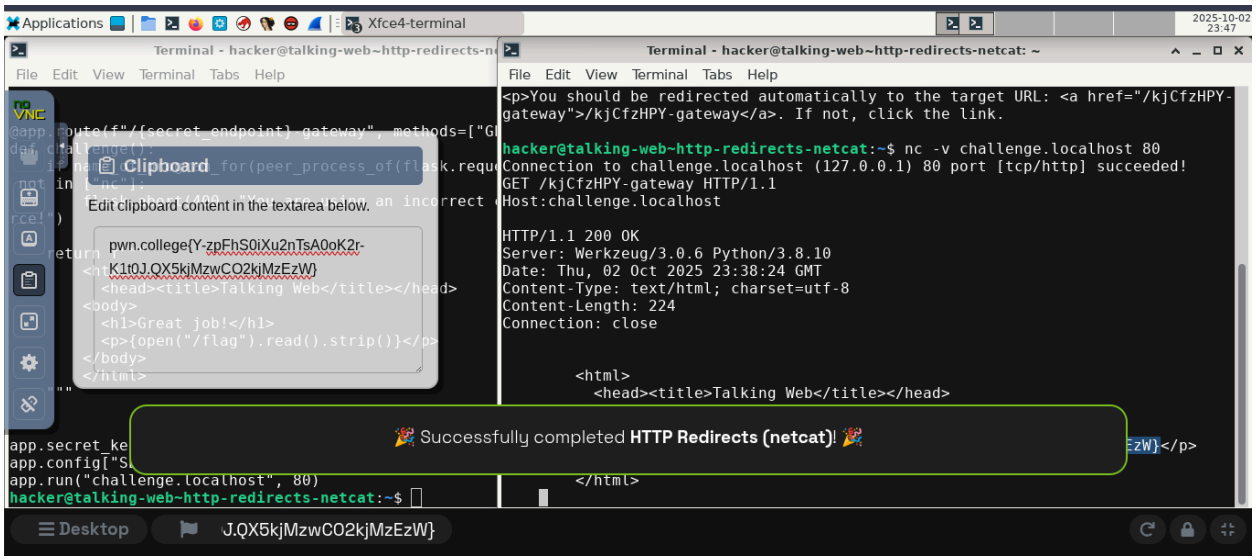


I first recognized that the server required a **POST** request to the **"/trail"** endpoint and that it specifically checked for the client being "nc" (**netcat**) to avoid errors. To meet this requirement, I used **netcat** to connect directly to **challenge.localhost on port 80**. I then crafted a raw **HTTP POST** request with the necessary headers, including **"Host: challenge.localhost"**, **"Content-Type: application/x-www-form-urlencoded"**, and **"Content-Length: 64"** to match the body length. The body contained the four required form fields with their exact values: **"secure_key=vbwurlji", "pin=tszszeta", "secret=gpgwdmyp",** and **"verify=shejreci"**. After sending the request, the server responded with an **HTTP 200 OK** status and an HTML page that displayed "Great job!" along with the flag: **pwn.college(wgUI8_znqdxb7c8Xswb74Uch1Tg.QXwkjWzwC02kjWzEzW)**. This confirmed that I had successfully provided all the correct parameters using netcat and retrieved the flag. Manually constructing a multi-parameter POST request with netcat deepened my understanding of the application/x-www-form-urlencoded format, especially the importance of calculating the exact Content-Length and properly delimiting parameters.

- HTTP Redirects (netcat)

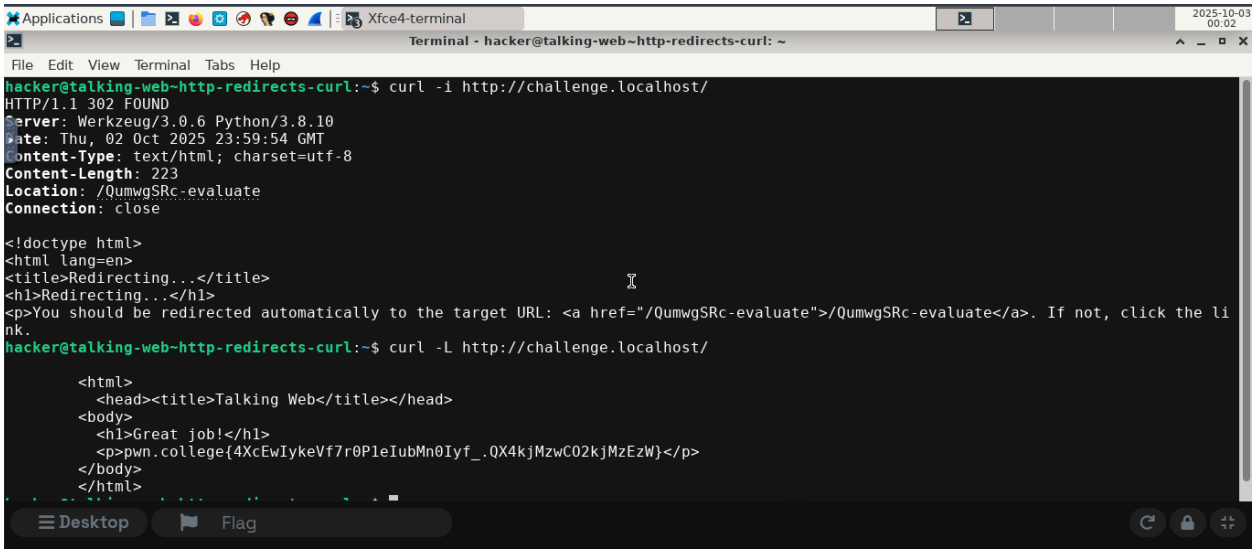


This is the first attempt at running the netcat without the location.

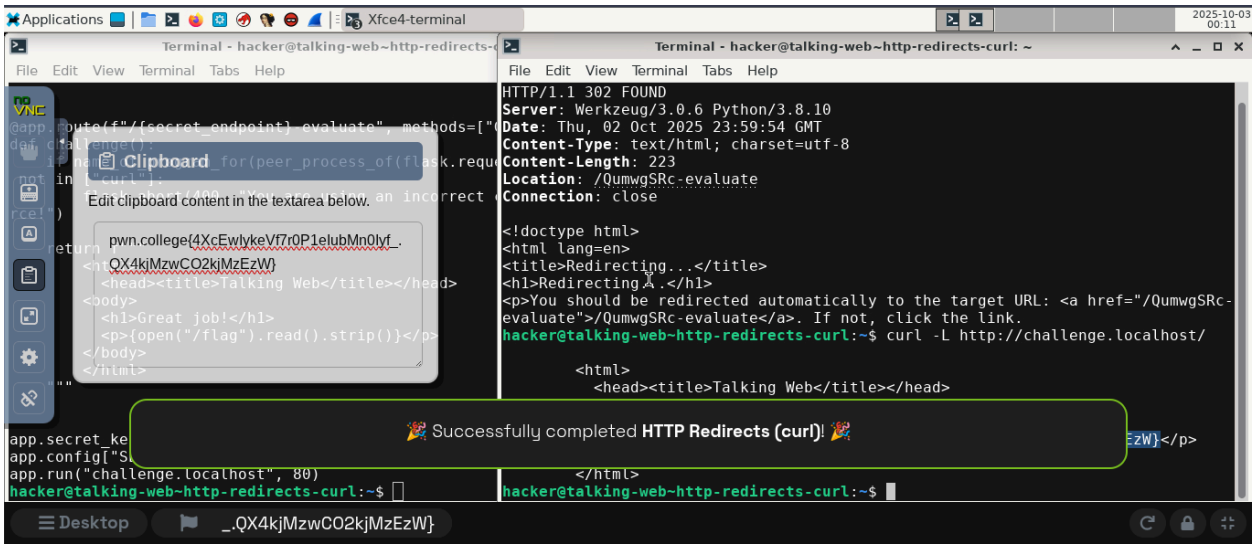


I connected to the server using **netcat** and sent a **GET request** to the root endpoint.
The server redirected me to a **secret endpoint** called **/kjCfzHPY-gateway**.
I then made a second **netcat** connection and requested that secret endpoint directly.
The server responded with an **HTML** page containing the flag:
pwn.college{Y-zpFhS0iXu2nTsA0oK2r-K1t0J.QX5kjMzwCO2kjMzEzW}.
I successfully retrieved the flag by ensuring both requests were made with netcat. This manual process of following a redirect emphasized that HTTP redirects are not automatic magic but simply a server response with a 3xx status code and a Location header, which the client must act upon.

- HTTP Redirects (curl)

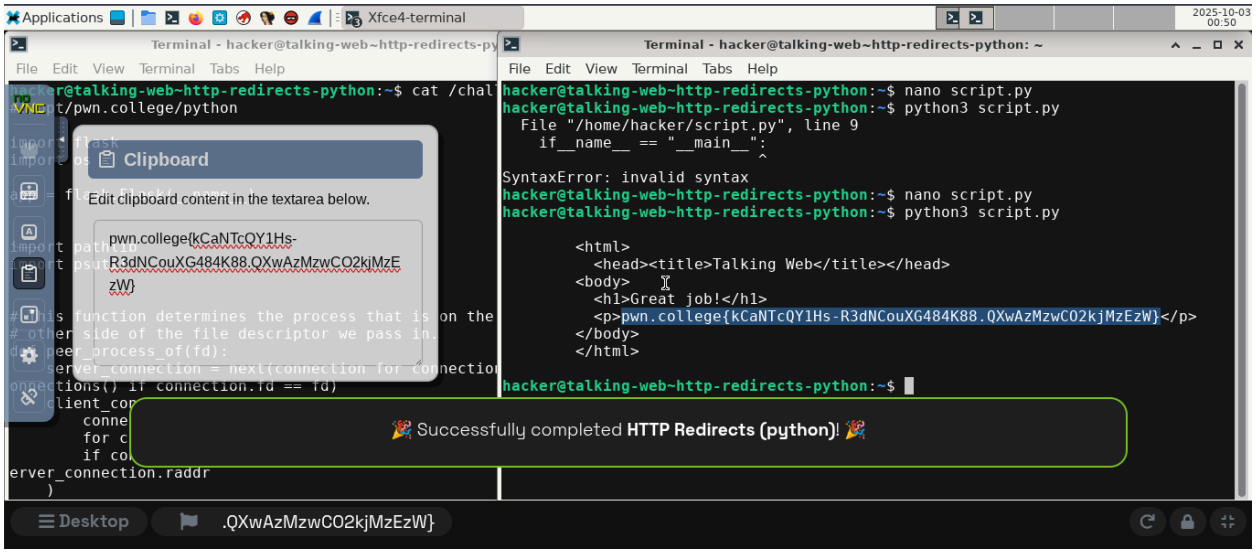


This is retrieving the secret location using the curl -i command.



I used curl to send a **request** to the **root endpoint** of the server.
The server responded with a **redirect** to a **new endpoint** called **/QumwgSRc-evaluate**.
I then used **curl** with the follow redirects option to automatically request the new endpoint.
The server returned a page congratulating me and displaying the flag: **pwn.college{4XcEwIykeVf7r0P1eIubMn0Iyf_.QX4kjMzwCO2kjMzEzW}**. Using curl's -L flag showcased the convenience of automated redirect following, a critical feature that modern browsers and tools use to create a seamless user experience on the web.

- HTTP Redirects (python)



I wrote a **Python** script to send a **GET** request to the server using the requests library.

```
#!/usr/bin/env python3
import requests

def main():
    url = "http://challenge.localhost/"
    response = requests.get(url)
    print(response.text)

if __name__ == "__main__":
    main()
```

I initially made a **syntax error** by **forgetting a space** in the if name check.
After correcting the error, I ran the script again.
The server responded with an **HTML** page that contained the flag: **pwn.college{kCaNTcQY1Hs-R3dNCouXG484K88.QXwAzMzwCO2kjMzEzW}**. The Python requests library abstracts away the redirect handling by default, demonstrating how high-level libraries simplify web client development, allowing the programmer to focus on logic rather than protocol details.