

Jaleel Williamson

jayw-713

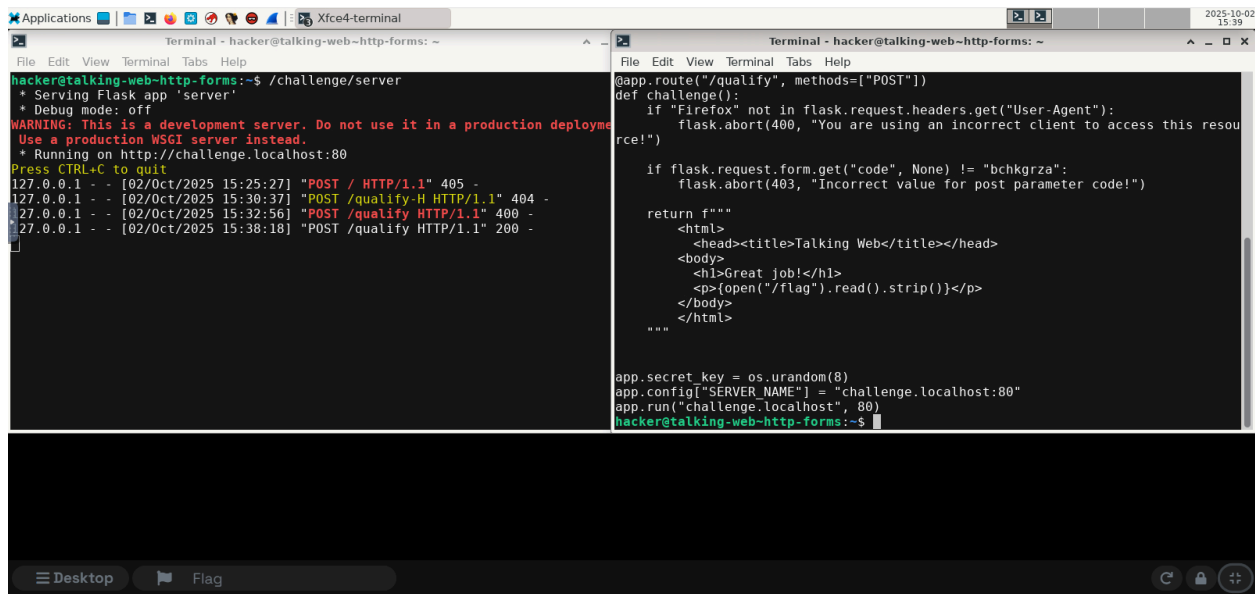
CSCI 400 Lab 9

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



```
hacker@talking-web-http-forms:~$ /challenge/server
* Serving Flask app 'server'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://challenge.localhost:80
Press CTRL+C to quit
127.0.0.1 - - [02/Oct/2025 15:25:27] "POST / HTTP/1.1" 405 -
127.0.0.1 - - [02/Oct/2025 15:30:37] "POST /qualify-H HTTP/1.1" 404 -
127.0.0.1 - - [02/Oct/2025 15:32:56] "POST /qualify HTTP/1.1" 400 -
127.0.0.1 - - [02/Oct/2025 15:38:18] "POST /qualify HTTP/1.1" 200 -

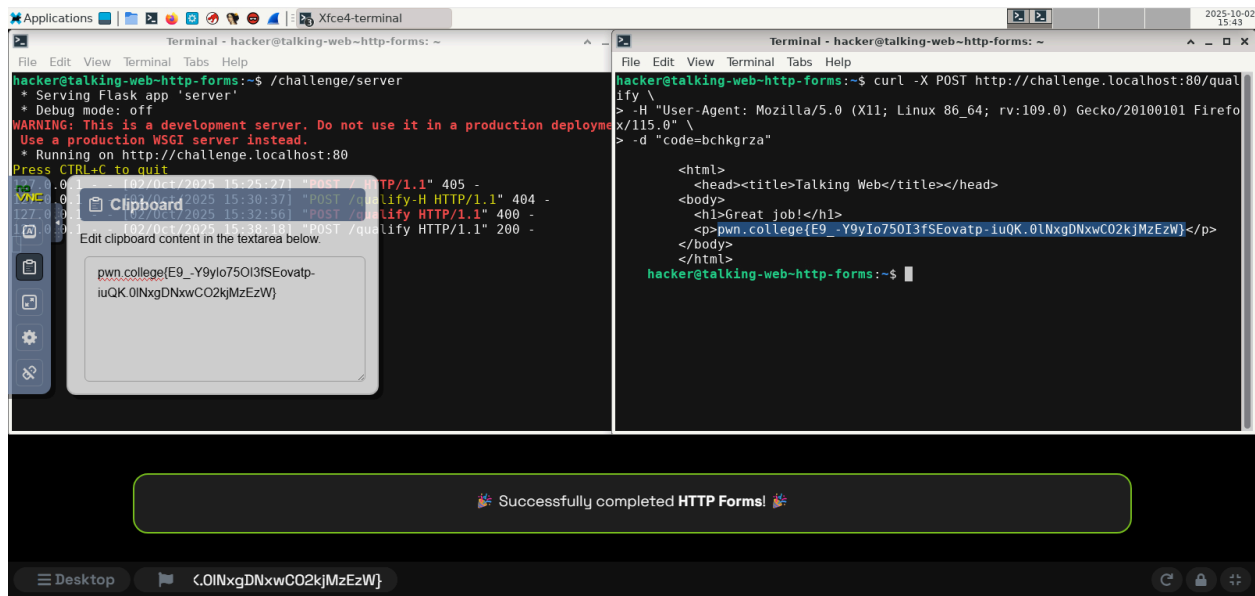
@app.route("/qualify", methods=["POST"])
def challenge():
    if "Firefox" not in flask.request.headers.get("User-Agent"):
        flask.abort(400, "You are using an incorrect client to access this resource!")

    if flask.request.form.get("code", None) != "bchkgzza":
        flask.abort(403, "Incorrect value for post parameter code!")

    return f"""
    <html>
    <head><title>Talking Web</title></head>
    <body>
    <h1>Great job!</h1>
    <p>{open("/flag").read().strip()}</p>
    </body>
    </html>
    """

app.secret_key = os.urandom(8)
app.config["SERVER_NAME"] = "challenge.localhost:80"
app.run("challenge.localhost", 80)
```

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



```
hacker@talking-web-http-forms:~$ /challenge/server
* Serving Flask app 'server'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://challenge.localhost:80
Press CTRL+C to quit
127.0.0.1 - - [02/Oct/2025 15:25:27] "POST / HTTP/1.1" 405 -
127.0.0.1 - - [02/Oct/2025 15:30:37] "POST /qualify-H HTTP/1.1" 404 -
127.0.0.1 - - [02/Oct/2025 15:32:56] "POST /qualify HTTP/1.1" 400 -
127.0.0.1 - - [02/Oct/2025 15:38:18] "POST /qualify HTTP/1.1" 200 -

hacker@talking-web-http-forms:~$ curl -X POST http://challenge.localhost:80/qualify \
-H "User-Agent: Mozilla/5.0 (X11; Linux 86_64; rv:109.0) Gecko/20100101 Firefox/115.0" \
-d "code=bchkgzza"

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>pwn.college{E9_Y9yIo75OI3fSEovatp-1uQK.0lNxgDNxwC02k}MzEzW</p>
</body>
</html>

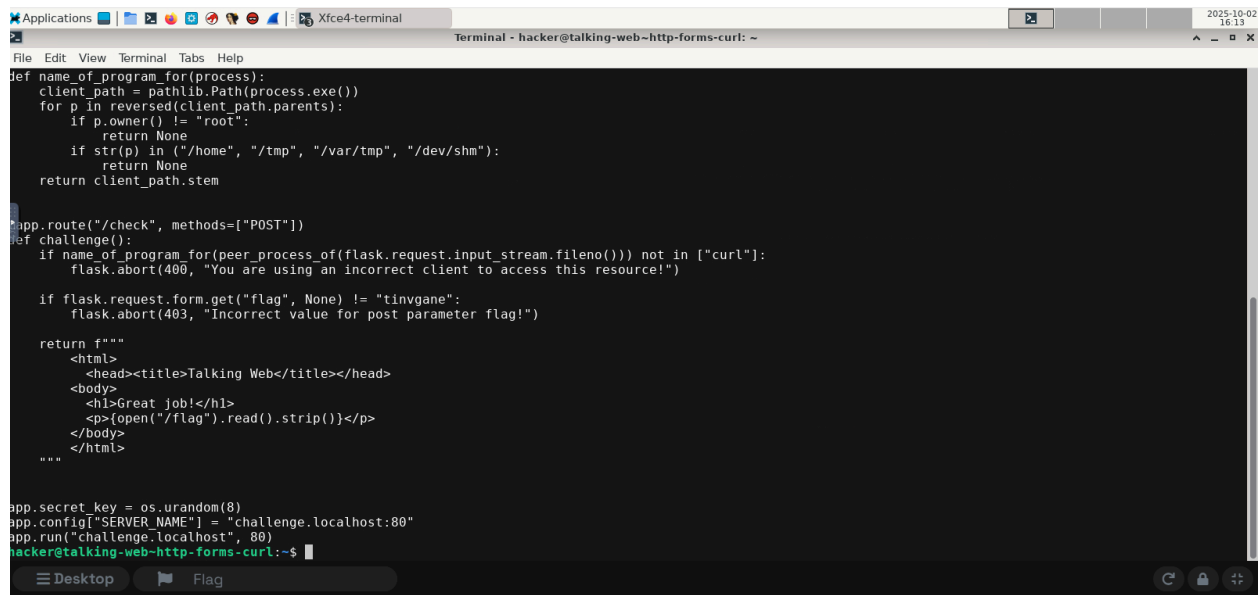
hacker@talking-web-http-forms:~$
```

Clipboard: pwn.college{E9_Y9yIo75OI3fSEovatp-1uQK.0lNxgDNxwC02k}MzEzW

Successfully completed HTTP Forms!

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-Agent** 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)

A screenshot of a terminal window titled "Terminal - hacker@talking-web-http-forms-curl: ~". The terminal shows the source code for a Flask application. The code includes a function to get the client path, a POST endpoint at "/check", and a challenge function that checks the client and the "flag" parameter. The challenge function returns an HTML response with a "Great job!" message and the flag. The server is configured to run on "challenge.localhost:80".

```
def name_of_program_for(process):
    client_path = pathlib.Path(process.exe())
    for p in reversed(client_path.parents):
        if p.owner() != "root":
            return None
        if str(p) in ("/home", "/tmp", "/var/tmp", "/dev/shm"):
            return None
    return client_path.stem

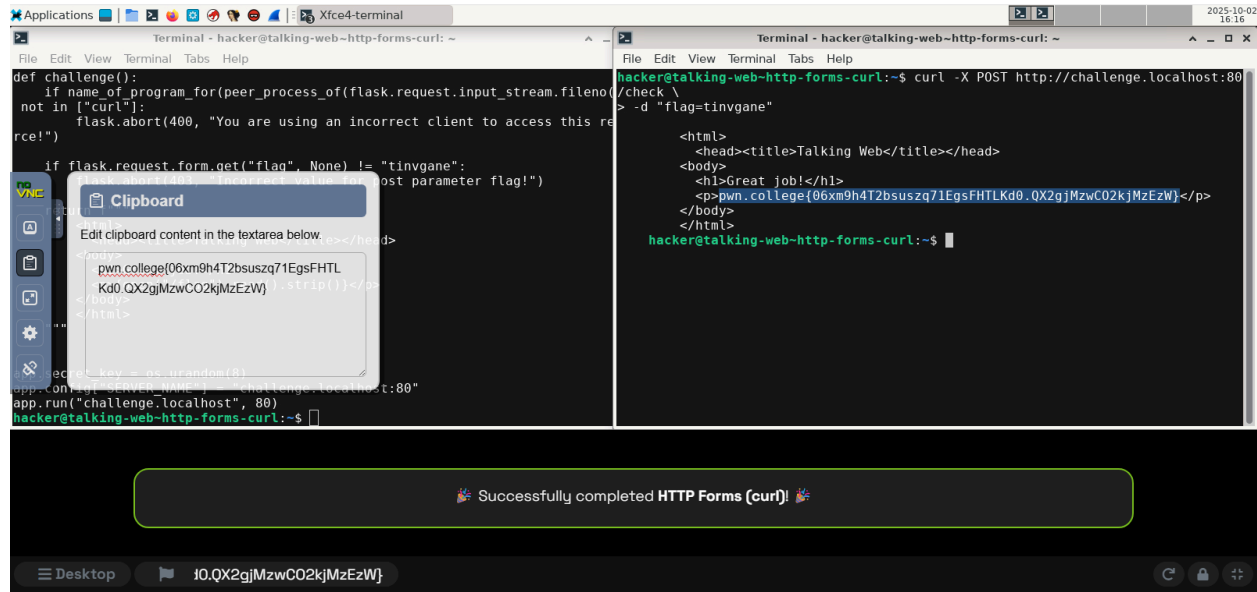
app.route("/check", methods=["POST"])
def challenge():
    if name_of_program_for(peer_process_of(flask.request.input_stream.fileno())) not in ["curl"]:
        flask.abort(400, "You are using an incorrect client to access this resource!")

    if flask.request.form.get("flag", None) != "tinvgane":
        flask.abort(403, "Incorrect value for post parameter flag!")

    return f"""
    <html>
    <head><title>Talking Web</title></head>
    <body>
    <h1>Great job!</h1>
    <p>{open("/flag").read().strip()}</p>
    </body>
    </html>
    """

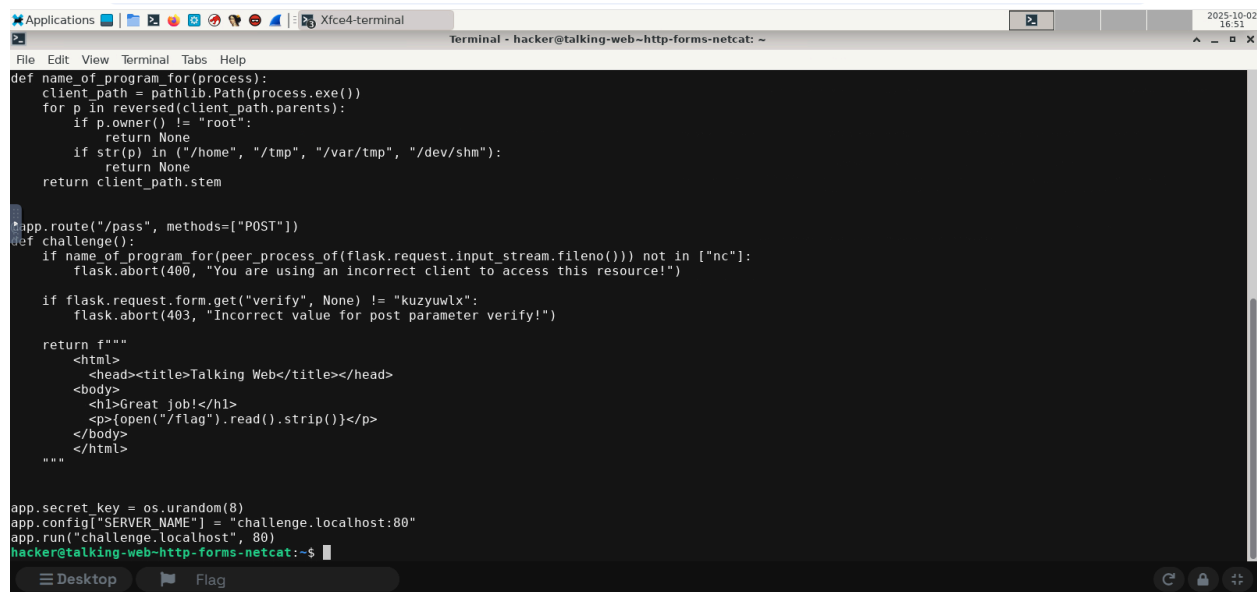
app.secret_key = os.urandom(8)
app.config["SERVER_NAME"] = "challenge.localhost:80"
app.run("challenge.localhost", 80)
hacker@talking-web-http-forms-curl:~$
```

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



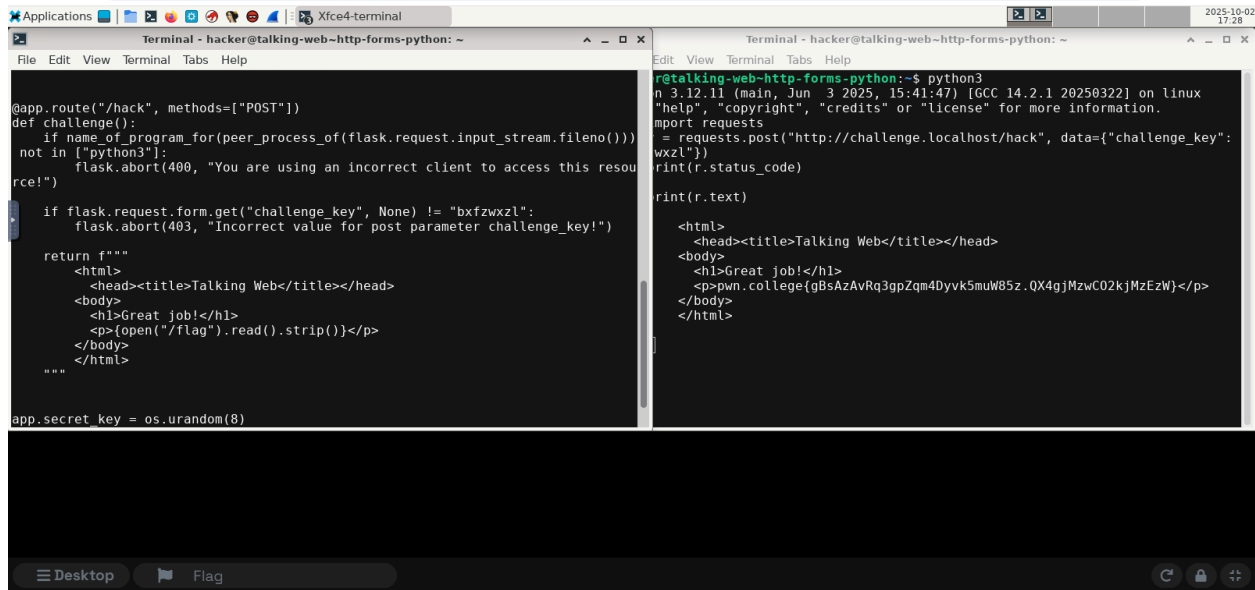
I first verified that the **Flask** server was running on **http://challenge.localhost:80**, as indicated in the terminal output. Then, I used the **curl** command to send a **POST** request to the **/check** endpoint with the following command: **curl -X POST http://challenge.localhost:80/check**. The 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(06xm9h4T2bsusq71EgsFHTLKd0.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

- HTTP Forms (python)



```
@app.route("/hack", methods=["POST"])
def challenge():
    if name_of_program_for(peer_process_of(flask.request.input_stream.fileno()))
    not in ["python3"]:
        flask.abort(400, "You are using an incorrect client to access this resource!")

    if flask.request.form.get("challenge_key", None) != "bxfzwxzl":
        flask.abort(403, "Incorrect value for post parameter challenge_key!")

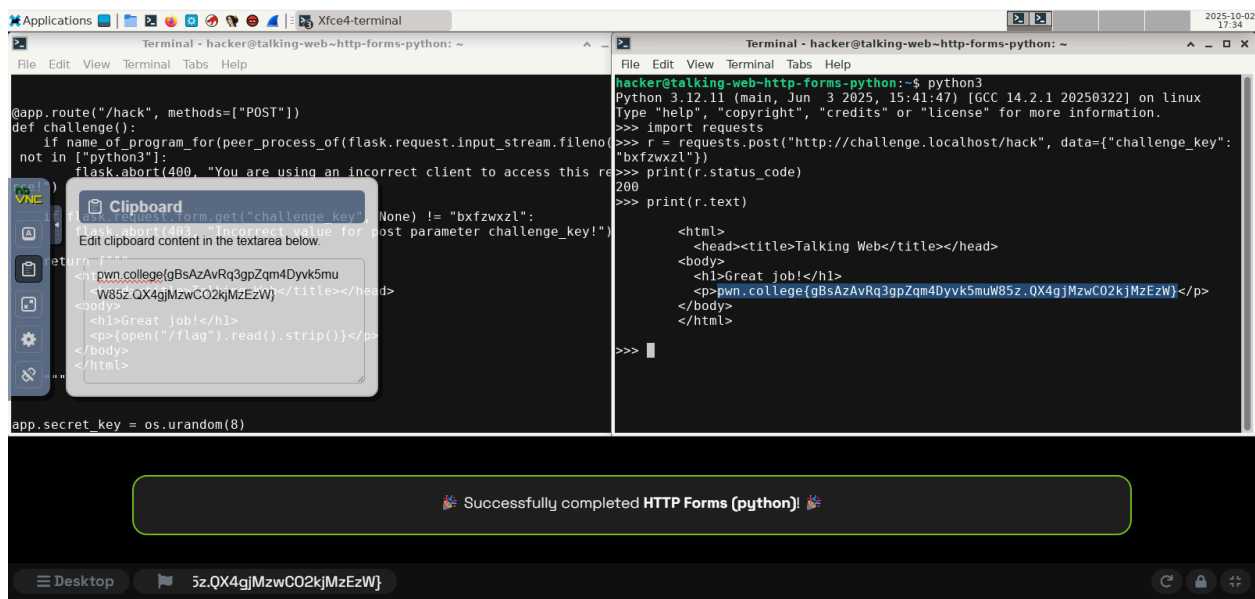
    return f"""
    <html>
    <head><title>Talking Web</title></head>
    <body>
    <h1>Great job!</h1>
    <p>{open("/flag").read().strip()}</p>
    </body>
    </html>
    """

app.secret_key = os.urandom(8)
```

```
python3
Python 3.12.11 (main, Jun 3 2025, 15:41:47) [GCC 14.2.1 20250322] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> r = requests.post("http://challenge.localhack", data={"challenge_key":
"bxfzwxzl"})
>>> print(r.status_code)
200
>>> print(r.text)

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>pwn.college(gBsAzAvRq3gpZqm4Dyvk5muW85z.QX4gjMzwC02kjMzEzW)</p>
</body>
</html>
```

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



```
@app.route("/hack", methods=["POST"])
def challenge():
    if name_of_program_for(peer_process_of(flask.request.input_stream.fileno()))
    not in ["python3"]:
        flask.abort(400, "You are using an incorrect client to access this resource!")

    if flask.request.form.get("challenge_key", None) != "bxfzwxzl":
        flask.abort(403, "Incorrect value for post parameter challenge_key!")

    return f"""
    <html>
    <head><title>Talking Web</title></head>
    <body>
    <h1>Great job!</h1>
    <p>{open("/flag").read().strip()}</p>
    </body>
    </html>
    """

app.secret_key = os.urandom(8)
```

```
python3
Python 3.12.11 (main, Jun 3 2025, 15:41:47) [GCC 14.2.1 20250322] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> r = requests.post("http://challenge.localhack", data={"challenge_key":
"bxfzwxzl"})
>>> print(r.status_code)
200
>>> print(r.text)

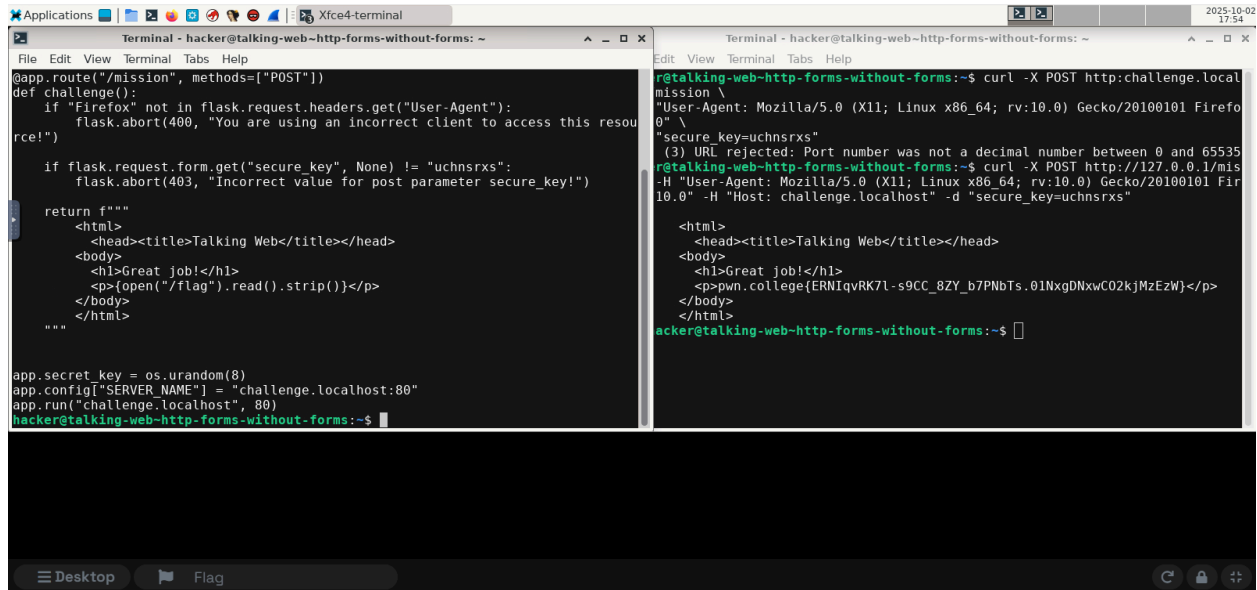
<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>pwn.college(gBsAzAvRq3gpZqm4Dyvk5muW85z.QX4gjMzwC02kjMzEzW)</p>
</body>
</html>
```

```
Clipboard
Edit clipboard content in the textarea below.
<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>pwn.college(gBsAzAvRq3gpZqm4Dyvk5muW85z.QX4gjMzwC02kjMzEzW)</p>
</body>
</html>
```

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.localhack**, 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 **"bxfzwxzl"**, 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(gBsAzAvRq3gpZqm4Dyvk5muW85z.QX4gjMzwC02kjMzEzW)**. This confirmed that I had successfully met the challenge requirements by using the proper client and

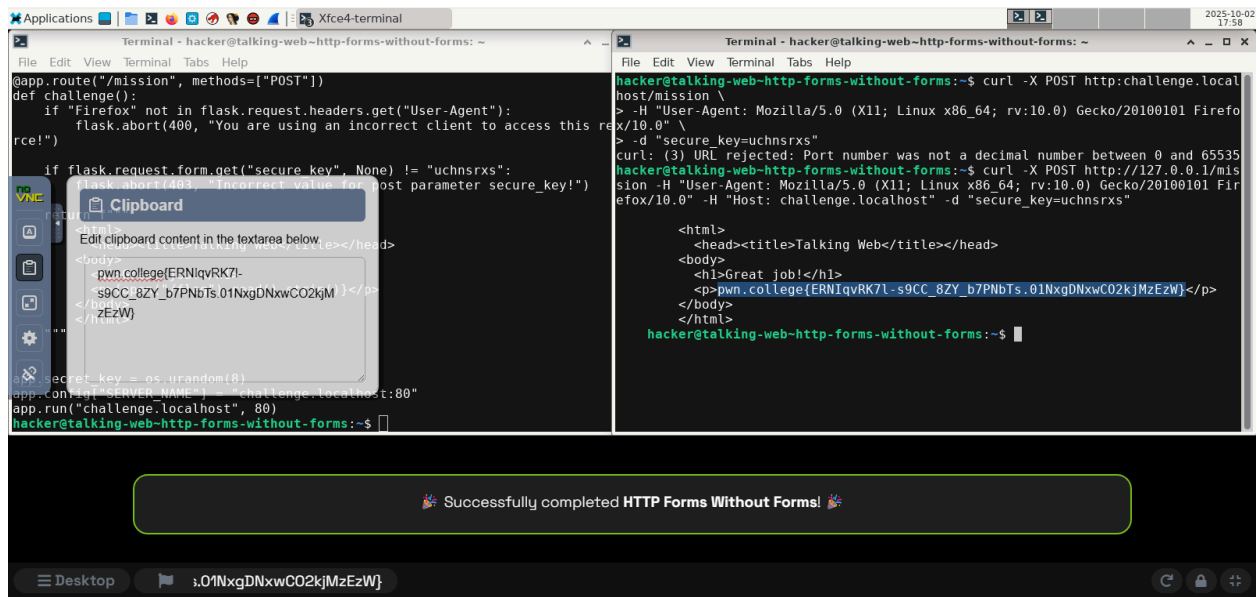
providing the correct parameter. This highlighted the power of using a scripting language like Python for web interactions, which provides immense flexibility for building complex, automated clients that can handle dynamic content and logic.

- HTTP Forms Without Forms



The screenshot shows two terminal windows. The left window displays the Flask application code for 'talking-web-http-forms-without-forms'. It defines a route for '/mission' that checks the 'User-Agent' for 'Firefox' and the 'secure_key' parameter for 'uchnsrxs'. If both checks pass, it returns an HTML page with the title 'Talking Web' and the message 'Great job!'. The right window shows the output of a curl command: 'curl -X POST http://challenge.local/mission -H "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:10.0) Gecko/20100101 Firefox/10.0" -d "secure_key=uchnsrxs"'. The output shows the HTML response from the server, confirming the successful completion of the challenge.

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



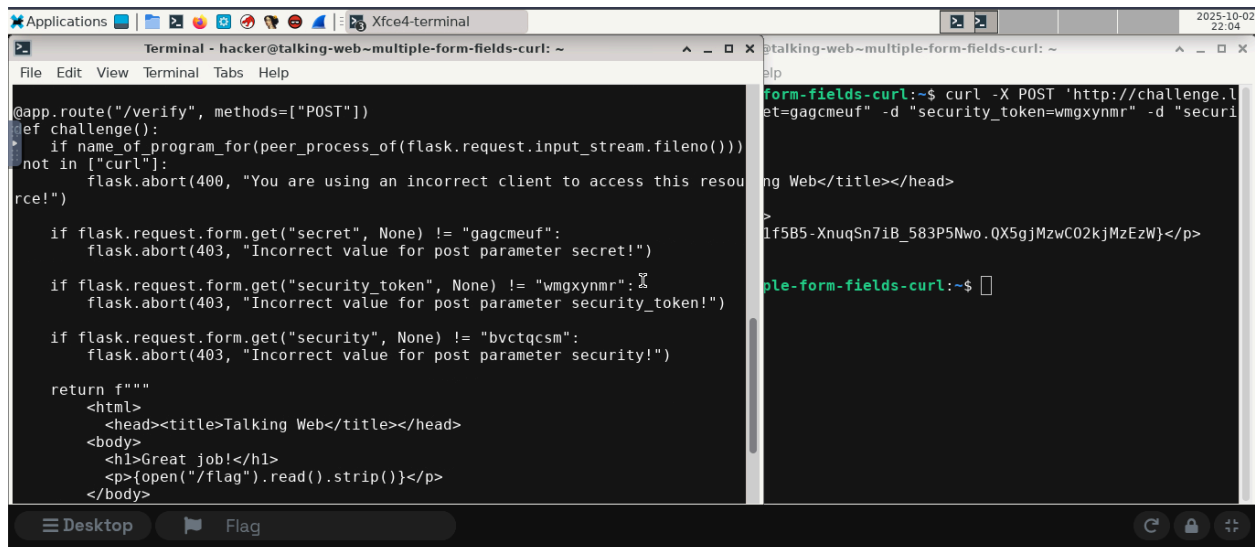
The screenshot shows a terminal window with the same Flask application code as before. A green box highlights the successful completion of the challenge, displaying the message: 'Successfully completed HTTP Forms Without Forms!'. The terminal also shows the curl command and its output, which is the same HTML response as in the previous screenshot.

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)



```
@app.route("/verify", methods=["POST"])
def challenge():
    if name_of_program_for(peer_process_of(flask.request.input_stream.fileno()))
    not in ["curl"]:
        flask.abort(400, "You are using an incorrect client to access this resource!")

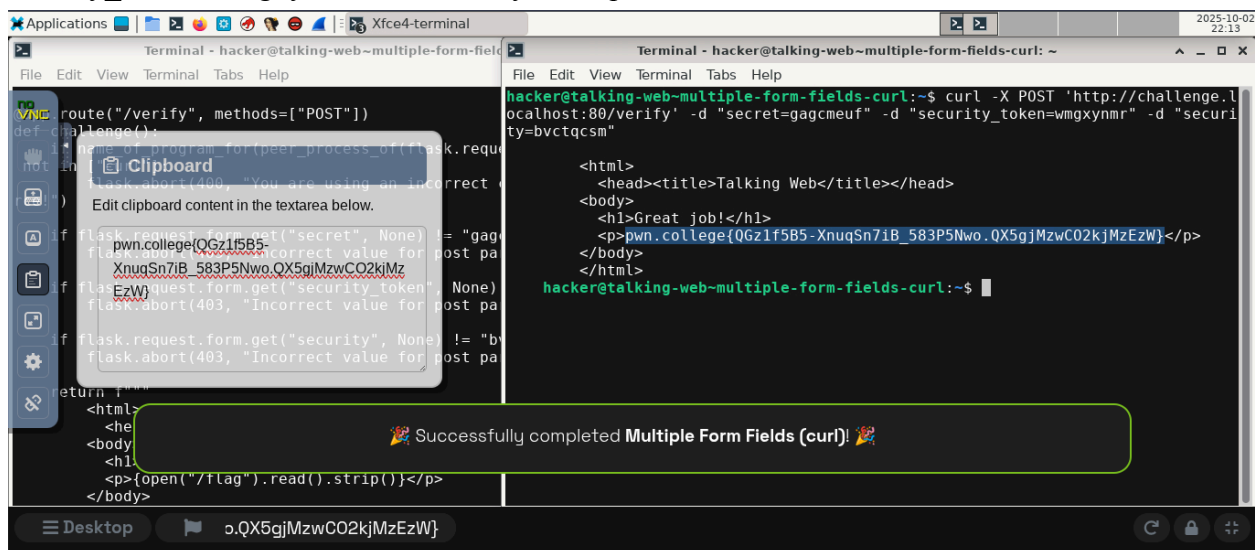
    if flask.request.form.get("secret", None) != "gagcmeuf":
        flask.abort(403, "Incorrect value for post parameter secret!")

    if flask.request.form.get("security_token", None) != "wmgxynmr":
        flask.abort(403, "Incorrect value for post parameter security_token!")

    if flask.request.form.get("security", None) != "bvctqcsn":
        flask.abort(403, "Incorrect value for post parameter security!")

    return f"""
    <html>
    <head><title>Talking Web</title></head>
    <body>
    <h1>Great job!</h1>
    <p>{open("/flag").read().strip()}</p>
    </body>
    """
```

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



```
hacker@talking-web-multiple-form-fields-curl:~$ curl -X POST 'http://challenge.l
ocalhost:80/verify' -d "secret=gagcmeuf" -d "security_token=wmgxynmr" -d "securi
ty=bvctqcsn"

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>pwn.college(gB&AzAVRq3gpZqm40yvk5muW85z.0X4gjMzvCQXsJWzEzM)</p>
</body>
</html>

hacker@talking-web-multiple-form-fields-curl:~$
```

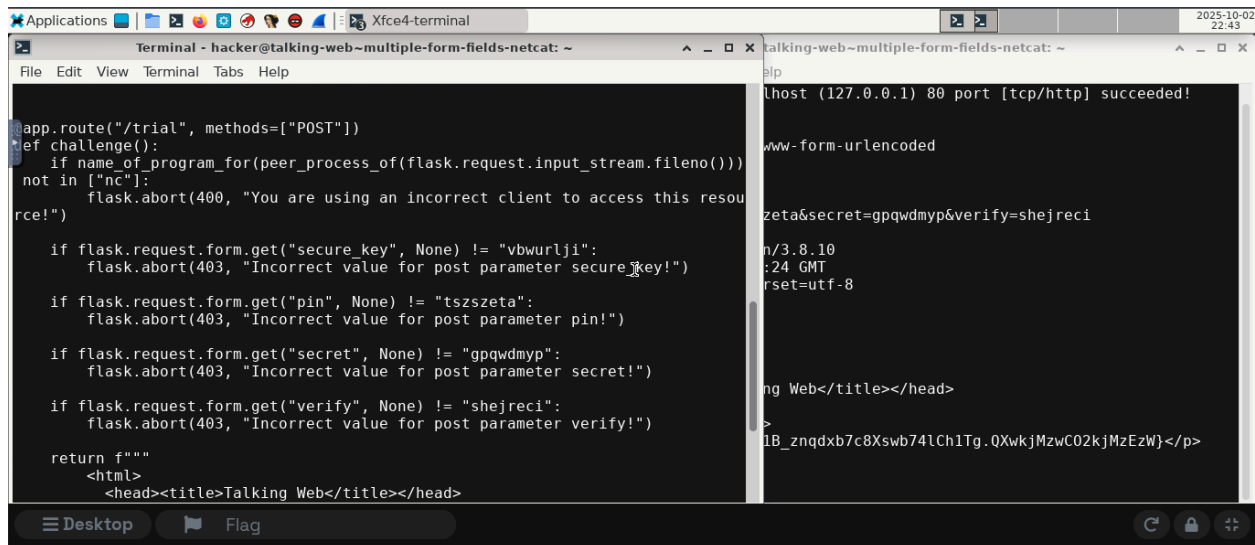
Successfully completed Multiple Form Fields (curl)!

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=bvtqcsn"**. 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)



The screenshot shows a netcat terminal window titled "Talking-web-multiple-form-fields-netcat: ~". The left pane displays the server code, and the right pane shows the client's request and the server's response.

```
app.route("/trial", methods=["POST"])
def challenge():
    if name_of_program_for(peer_process_of(flask.request.input_stream.fileno()))
    not in ["nc"]:
        flask.abort(400, "You are using an incorrect client to access this resource!")

    if flask.request.form.get("secure_key", None) != "vbwurlji":
        flask.abort(403, "Incorrect value for post parameter secure_key!")

    if flask.request.form.get("pin", None) != "tszszeta":
        flask.abort(403, "Incorrect value for post parameter pin!")

    if flask.request.form.get("secret", None) != "gpqwdmvp":
        flask.abort(403, "Incorrect value for post parameter secret!")

    if flask.request.form.get("verify", None) != "shejreci":
        flask.abort(403, "Incorrect value for post parameter verify!")

    return f"""
    <html>
    <head><title>Talking Web</title></head>
```

The right pane shows the client's request and the server's response:

```
host (127.0.0.1) 80 port [tcp/http] succeeded!
www-form-urlencoded
zeta&secret=gpqwdmvp&verify=shejreci
n/3.8.10
:24 GMT
rset=utf-8
ng Web</title></head>
>
1B_znqdx7c8Xswb74lCh1Tg.QXwkjMzwC02kjMzEzW}</p>
```

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


```
Terminal - hacker@talking-web~multiple-form-fields-netcat: ~
File Edit View Terminal Tabs Help

Connection to challenge.localhost (127.0.0.1) 80 port [tcp/http] succeeded!
POST /trial HTTP/1.1
Host: challenge.localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 64
Connection: close

secure_key=vbwurlji&pin=tszszeta&secret=gpgwdmvp&verify=shejreci
HTTP/1.1 200 OK
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Thu, 02 Oct 2025 22:42:24 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 224
Connection: close

<html>
<head><title>Talking Web</title></head>

</html>

Successfully completed Multiple Form Fields (netcat)!
```

I first recognized that the server required a **POST** request to the `"/trial"` 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=gpgwdmvp"**, 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(wgU18_znqdx7c8Xswb74Uch1Tg.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)

```

hacker@talking-web-http-redirects-netcat:~$ nc -v challenge.localhost 80
Connection to challenge.localhost (127.0.0.1) 80 port [tcp/http] succeeded!
GET / HTTP/1.1
Host: challenge.localhost

HTTP/1.1 302 FOUND
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Thu, 02 Oct 2025 23:35:14 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 221
Location: /kJCfzHPY-gateway
Connection: close

<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a href="/kJCfzHPY-gateway">/kJCfzHPY-gateway</a>. If not, click the link

hacker@talking-web-http-redirects-netcat:~$ nc -v challenge.localhost 80
Connection to challenge.localhost (127.0.0.1) 80 port [tcp/http] succeeded!
GET /kJCfzHPY-gateway HTTP/1.1
Host: challenge.localhost

HTTP/1.1 200 OK
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Thu, 02 Oct 2025 23:38:24 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 224
Connection: close

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>{open("/flag").read().strip()}</p>
</body>
</html>

```

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

```

hacker@talking-web-http-redirects-netcat:~$ nc -v challenge.localhost 80
Connection to challenge.localhost (127.0.0.1) 80 port [tcp/http] succeeded!
GET / HTTP/1.1
Host: challenge.localhost

HTTP/1.1 302 FOUND
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Thu, 02 Oct 2025 23:38:24 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 224
Connection: close

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>{open("/flag").read().strip()}</p>
</body>
</html>

hacker@talking-web-http-redirects-netcat:~$ nc -v challenge.localhost 80
Connection to challenge.localhost (127.0.0.1) 80 port [tcp/http] succeeded!
GET /kJCfzHPY-gateway HTTP/1.1
Host: challenge.localhost

HTTP/1.1 200 OK
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Thu, 02 Oct 2025 23:38:24 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 224
Connection: close

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>{open("/flag").read().strip()}</p>
</body>
</html>

```

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**.

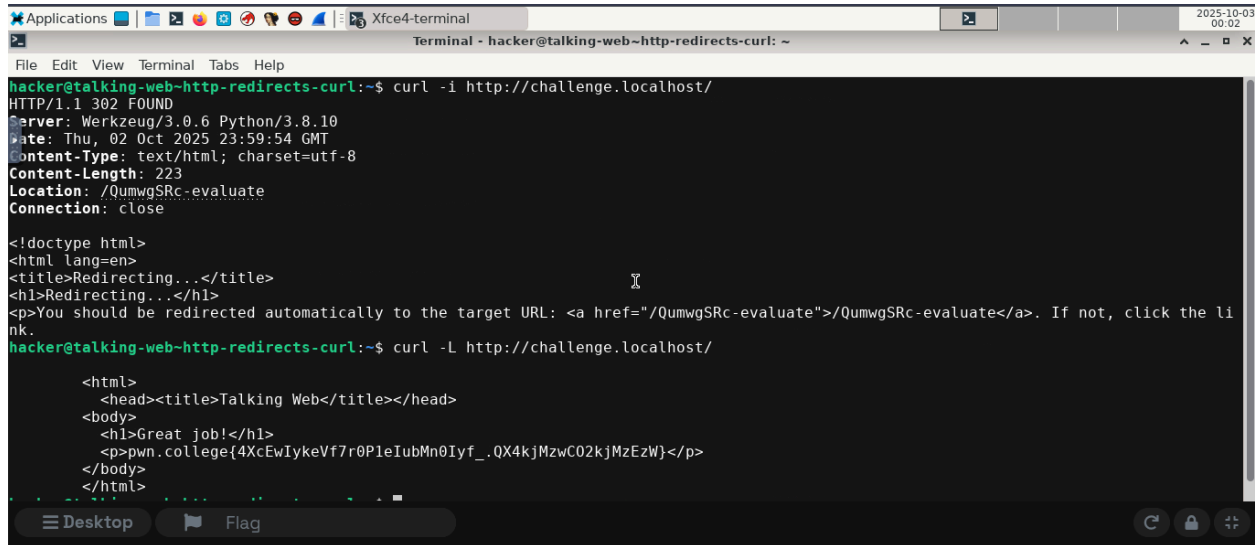
I then made a second **netcat** connection and requested that secret endpoint directly.

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)

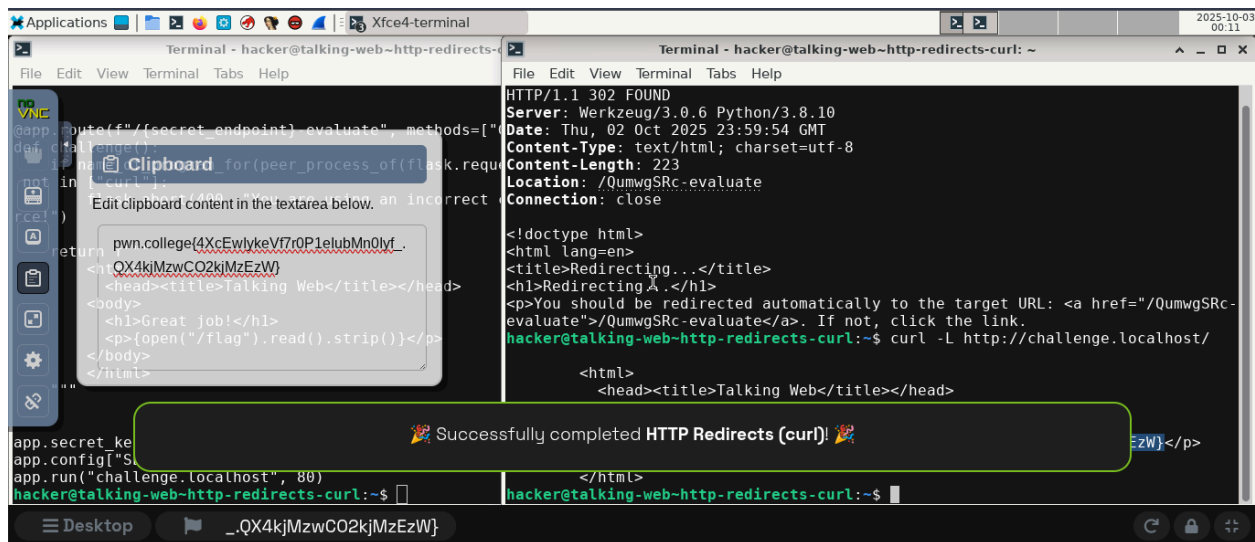


```
hacker@talking-web-http-redirects-curl:~$ curl -i http://challenge.localhost/
HTTP/1.1 302 FOUND
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Thu, 02 Oct 2025 23:59:54 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 223
Location: /QumwgSRc-evaluate
Connection: close

<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a href="/QumwgSRc-evaluate">/QumwgSRc-evaluate</a>. If not, click the link.
hacker@talking-web-http-redirects-curl:~$ curl -L http://challenge.localhost/

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>pwn.college{4XcEwIykeVf7r0P1eIubMn0Iyf_.QX4kjMzwCO2kjMzEzW}</p>
</body>
</html>
```

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



```
app.secret_key = os.urandom(24)
app.config["SECRET_KEY"] = os.urandom(24)
app.run("challenge.localhost", 80)

hacker@talking-web-http-redirects-curl:~$ curl -i http://challenge.localhost/
HTTP/1.1 302 FOUND
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Thu, 02 Oct 2025 23:59:54 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 223
Location: /QumwgSRc-evaluate
Connection: close

<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a href="/QumwgSRc-evaluate">/QumwgSRc-evaluate</a>. If not, click the link.
hacker@talking-web-http-redirects-curl:~$ curl -L http://challenge.localhost/

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>pwn.college{4XcEwIykeVf7r0P1eIubMn0Iyf_.QX4kjMzwCO2kjMzEzW}</p>
</body>
</html>
```

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)

```

hacker@talking-web-http-redirects-python:~$ cat /challenge
pwn.college{kCaNTcQY1Hs-R3dNCouXG484K88.QXwAzMzwCO2kjMzEzW}

hacker@talking-web-http-redirects-python:~$ nano script.py
File "/home/hacker/script.py", line 9
    if __name__ == "__main__":
        ^
SyntaxError: invalid syntax

hacker@talking-web-http-redirects-python:~$ nano script.py
hacker@talking-web-http-redirects-python:~$ python3 script.py

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>pwn.college{kCaNTcQY1Hs-R3dNCouXG484K88.QXwAzMzwCO2kjMzEzW}</p>
</body>
</html>

hacker@talking-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.