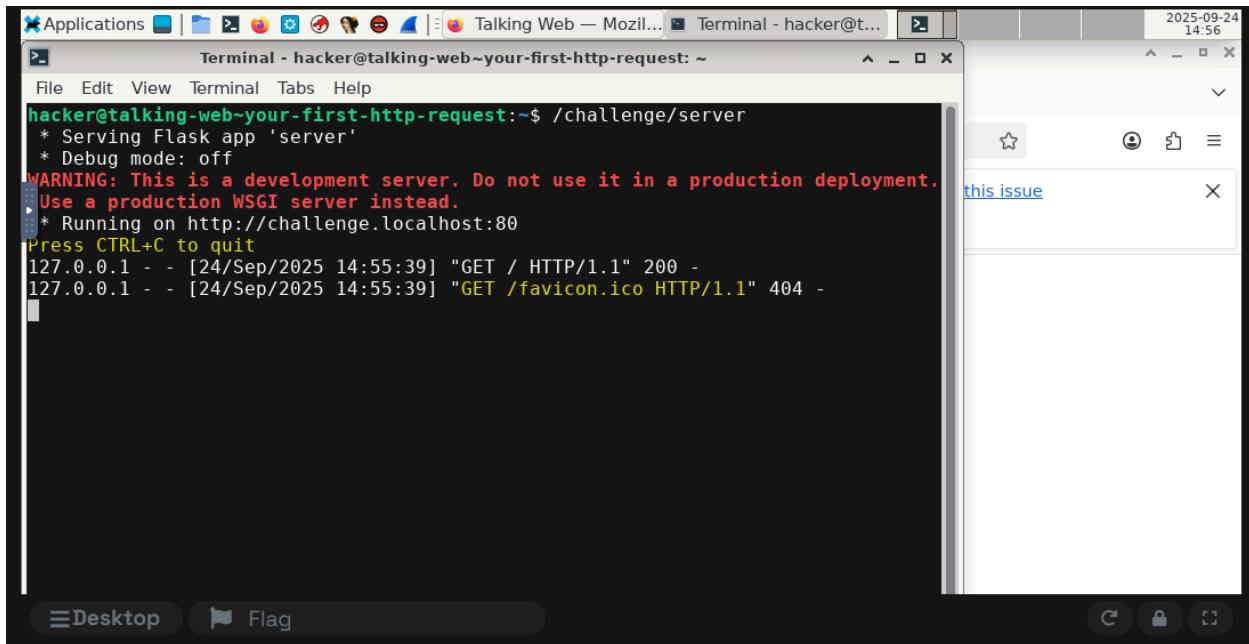


Jaleel Williamson
jayw-713
CSCI 400 Lab 8
9/29/25

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

Challenges 'Your First HTTP Request' through 'Multiple HTTP Parameters (curl)' (15 challenges total)

- Your First HTTP Request

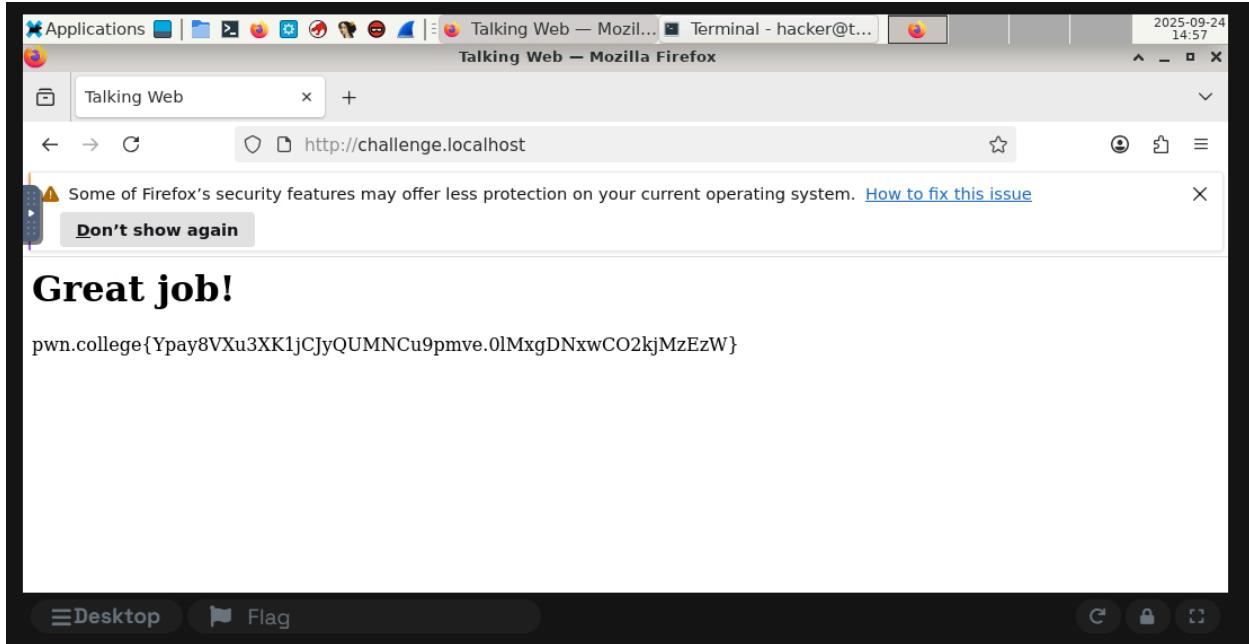


A screenshot of a macOS desktop environment. In the center is a terminal window titled "Terminal - hacker@talking-web~your-first-http-request:~\$". The terminal is running a Flask application, with the following output:

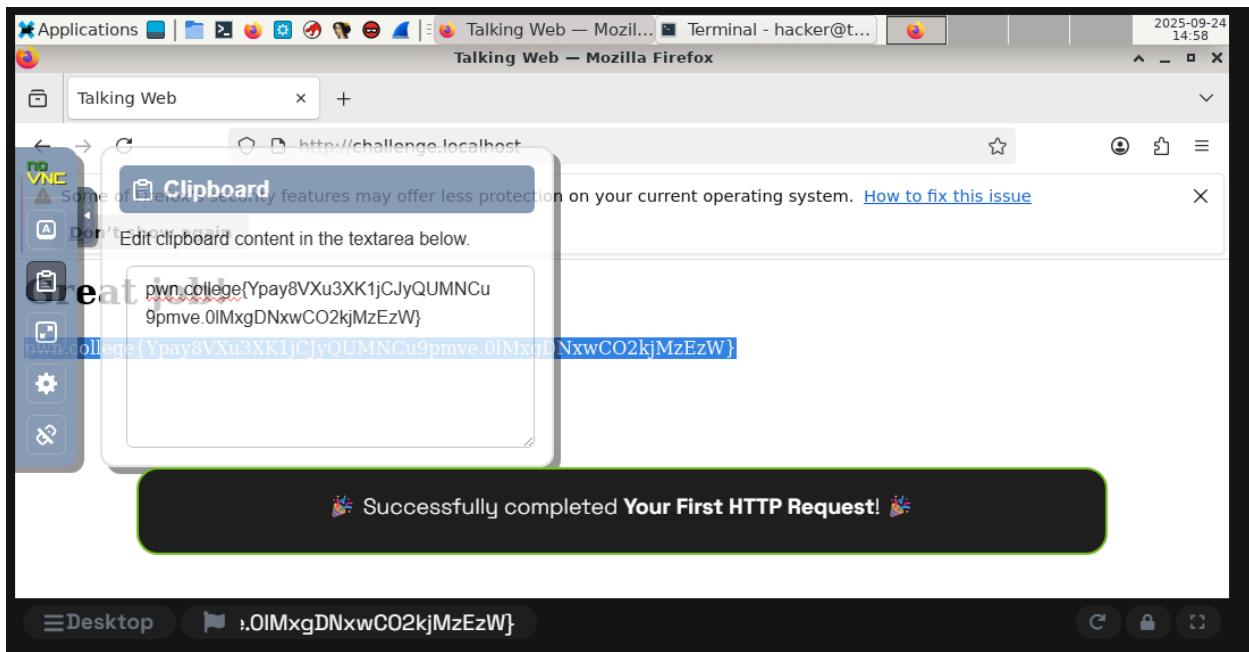
```
hacker@talking-web~your-first-http-request:~$ /challenge/server
 * Serving Flask app 'server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 * Running on http://challenge.localhost:80
Press CTRL+C to quit
127.0.0.1 - - [24/Sep/2025 14:55:39] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [24/Sep/2025 14:55:39] "GET /favicon.ico HTTP/1.1" 404 -
```

To the right of the terminal is a sidebar with a single item: "this issue". At the bottom of the screen, there is a dock with icons for "Desktop" and "Flag".

Start of Challenge in GUI desktop

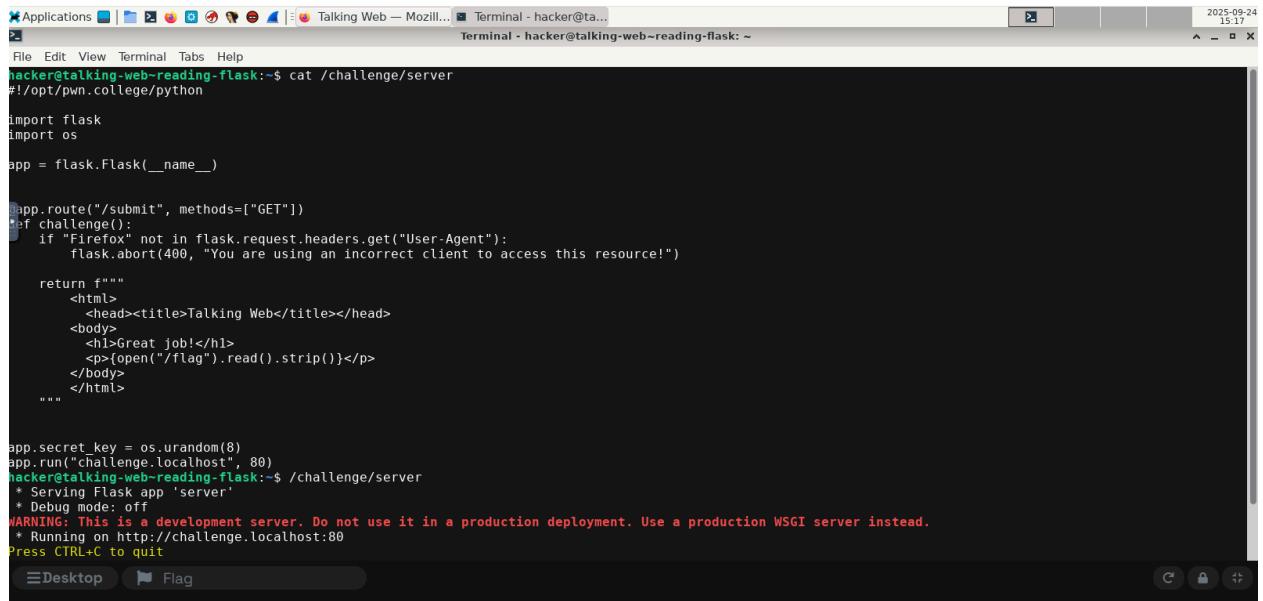


Successful Retrieval in Firefox browser



To capture the flag, I first ran the `/challenge/server` command in the terminal within the `pwn.college` dojo workspace, which started a **local web server** and **provided a URL**. Then, I launched Firefox from the **GUI Desktop** and navigated to that **URL**. The server responded with the flag, which I copied from the browser display. This completed the "Your First HTTP Request" challenge. This fundamental exercise underscored that at its core, web interaction is simply a client program (the browser) requesting a resource from a server program, establishing the basic communication model for the web.

● Reading Flask



Applications Talking Web — Mozilla... Terminal - hacker@talking-web: ~

```
hacker@talking-web-reading-flask:~$ cat /challenge/server
#!/opt/pwn.college/python

import flask
import os

app = flask.Flask(__name__)

@app.route("/submit", methods=["GET"])
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!")

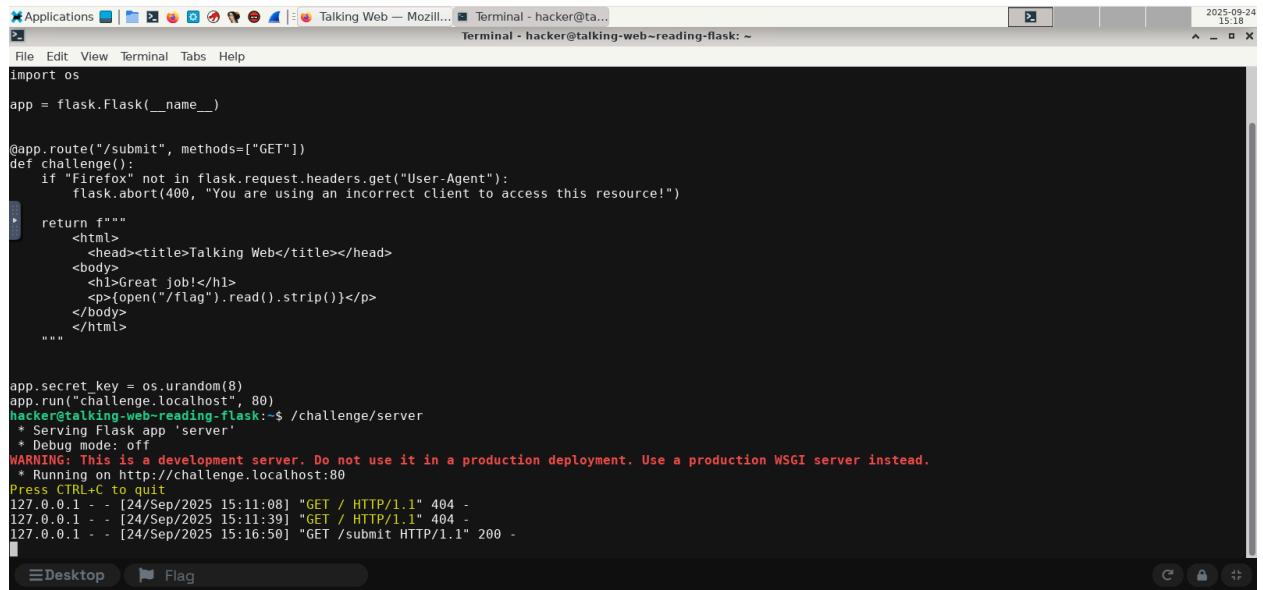
    return f"""
        <html>
        <head><title>Talking Web</title></head>
        <body>
            <h1>Great job!</h1>
            <p>{open("/flag").read().strip()}</p>
        </body>
    </html>
    """

app.secret_key = os.urandom(8)
app.run("challenge.localhost", 80)
hacker@talking-web-reading-flask:~$ ./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
```

Desktop Flag

2025-09-24 15:17

Launching the challenge in GUI desktop terminal.



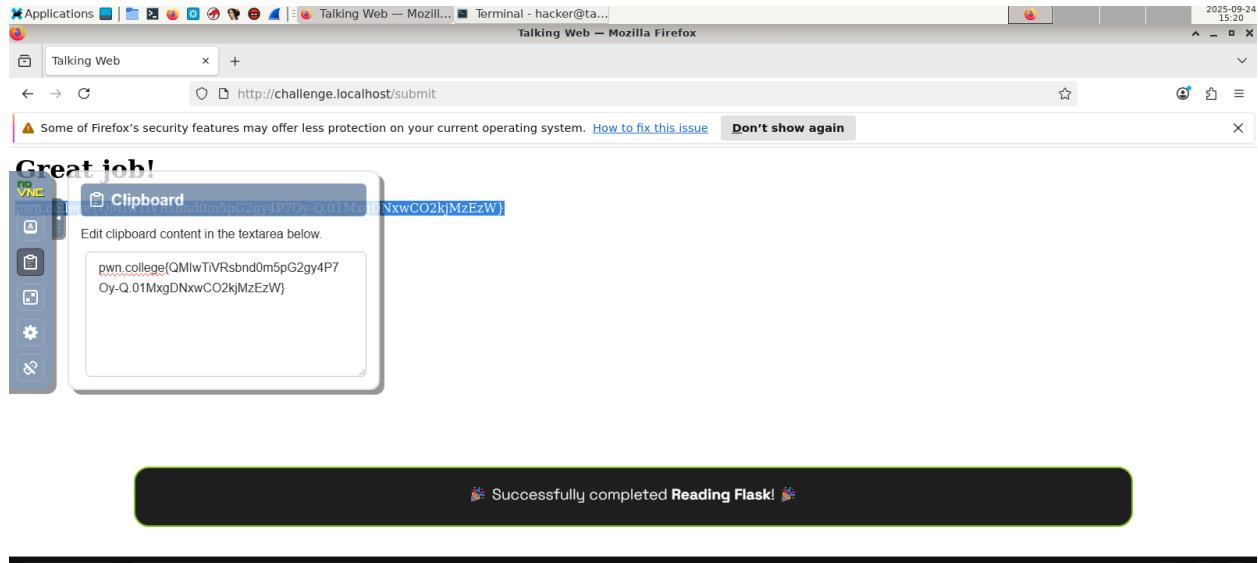
Applications Talking Web — Mozilla... Terminal - hacker@talking-web: ~

```
hacker@talking-web-reading-flask:~$ ./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 - - [24/Sep/2025 15:11:08] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [24/Sep/2025 15:11:39] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [24/Sep/2025 15:16:50] "GET /submit HTTP/1.1" 200 -
```

Desktop Flag

2025-09-24 15:18

Continuation of first pic in terminal.



To capture the flag, I first read the source code of `/challenge/server` in the **terminal**, which revealed that the correct endpoint is `/submit` and it requires the User-Agent header to contain "Firefox". I then ran the server and accessed <http://challenge.localhost/submit> using Firefox from the **GUI Desktop**. The server responded with a page displaying "Great job!" and the flag: `pwn.college{QMIwTiVRsbnd0m5pG2gy4P7Oy-Q.01MxgDNxwCO2kjMzEzW}`. This demonstrated understanding of the server's routing and client requirements. The key takeaway was that server-side code defines the rules of engagement; success depends on the client understanding and adhering to these programmed constraints, such as specific endpoints and headers.

● Commented Data

```

Applications   http://challenge.localhost/submit Terminal - hacker@talking-web~commented-data: ~
File Edit View Terminal Tabs Help
hacker@talking-web~commented-data:~$ cat /challenge/server
#!/opt/pwn.college/python

import flask
import os

app = flask.Flask(__name__)

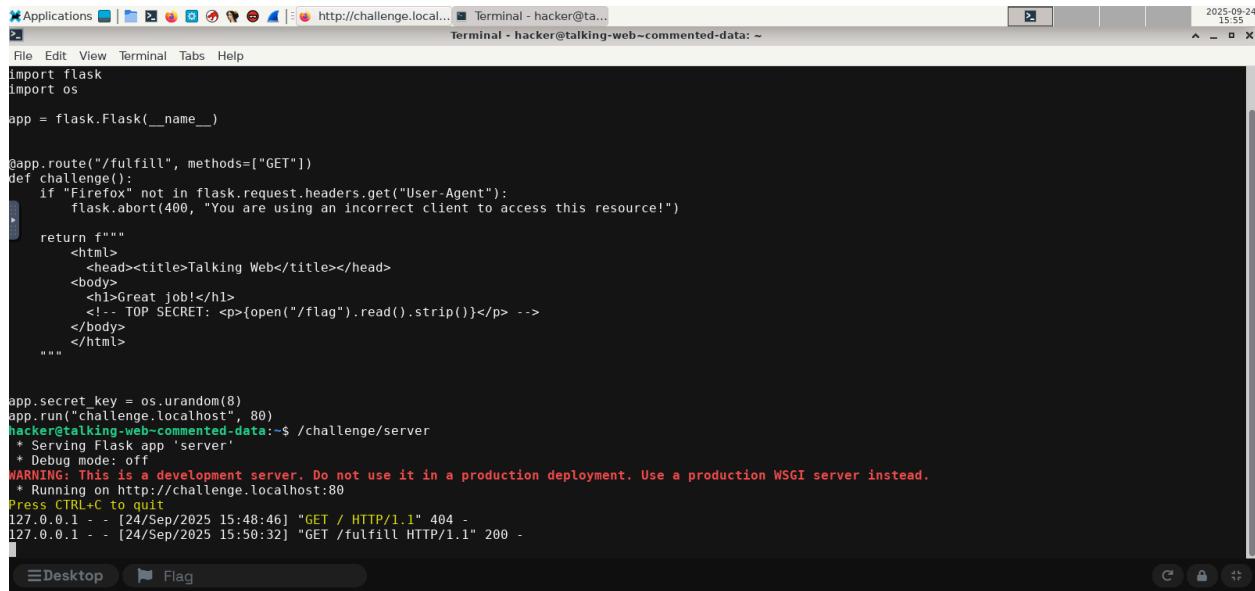
app.route("/fulfill", methods=["GET"])
def challenge():
    if "Firefox" not in flask.request.headers.get("User-Agent"):
        flask.abort(400, "You are using an incorrect client to access this resource!")

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

app.secret_key = os.urandom(8)
app.run("challenge.localhost", 80)
hacker@talking-web~commented-data:~$ /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

```

Launching the challenge in GUI desktop terminal.



```

Applications http://challenge.local... Terminal - hacker@ta...
File Edit View Terminal Tabs Help
Import flask
import os

app = flask.Flask(__name__)

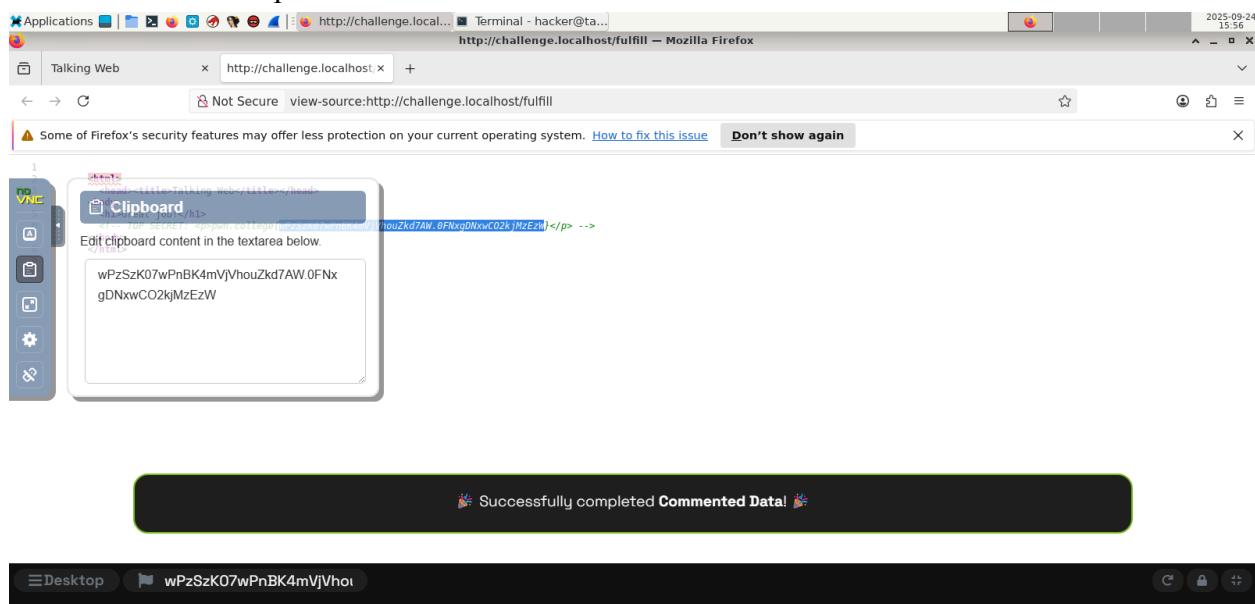
@app.route("/fulfill", methods=["GET"])
def challenge():
    if "Firefox" not in flask.request.headers.get("User-Agent"):
        flask.abort(400, "You are using an incorrect client to access this resource!")

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

app.secret_key = os.urandom(8)
app.run("challenge.localhost", 80)
hacker@talking-web-commented-data:~$ /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 - - [24/Sep/2025 15:48:46] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [24/Sep/2025 15:50:32] "GET /fulfill HTTP/1.1" 200 -

```

Continuation of first pic in terminal.



http://challenge.localhost/fulfill - Mozilla Firefox

Clipboard

TOP SECRET: <p>{open("/flag").read().strip()}</p> -->

wPzSzK07wPnBK4mVjVhouZkd7AW0FNxgDNowCO2kjMzEzW

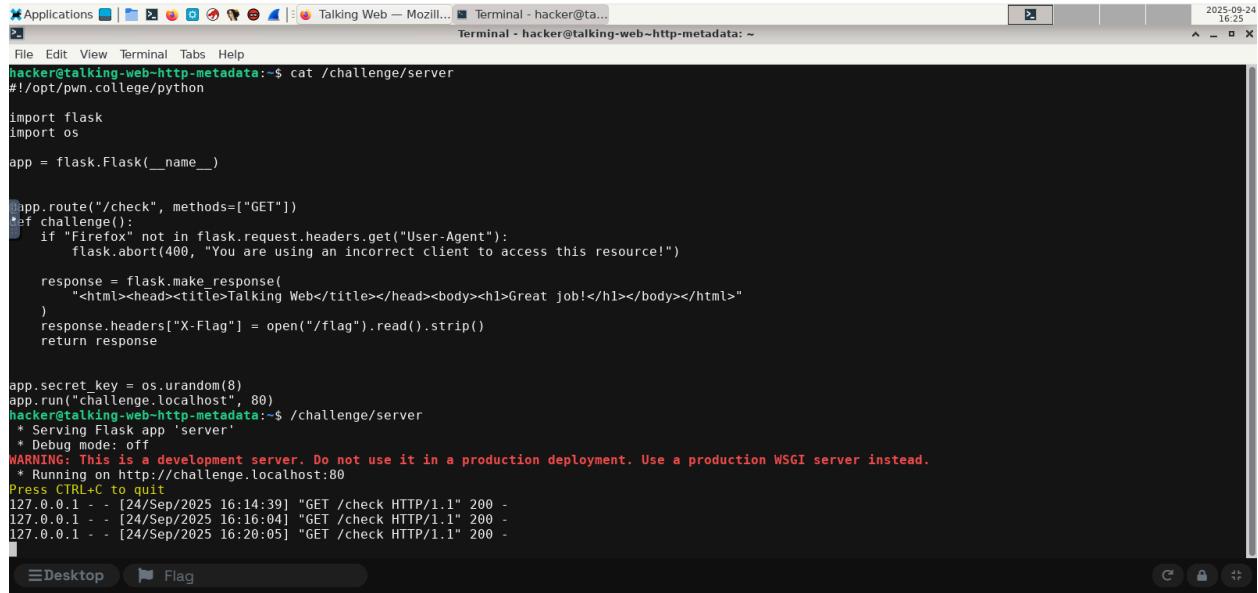
Successfully completed Commented Data!

I found the flag by first running the `/challenge/server` script to start the web server. Then, I accessed the **correct endpoint** `/fulfill` in Firefox from the **GUI Desktop**, as required by the **server code**. Since the flag was **hidden** in an **HTML comment**, I viewed the **page source** by right-clicking and selecting "View Page Source" or using the **Firefox menu**. In the source code, I located the comment `<!-- TOP SECRET:`

`<p>pwn.college{wPzSzK07wPnBK4mVjVhouZkd7AW1QFNxgDNowCO2kjMzEzW}</p>`

`-->` and copied the flag from there. The flag is `pwn.college{wPzSzK07wPnBK4mVjVhouZkd7AW1QFNxgDNowCO2kjMzEzW}`. This challenge highlighted that not all data served to a client is meant for direct display; crucial information can be hidden within the page's structure, requiring inspection of the underlying source code.

● HTTP Metadata



```
Applications Talking Web - Mozilla Firefox Terminal - hacker@talking-web:~$ cat /challenge/server
#!/opt/pwn.college/python

import flask
import os

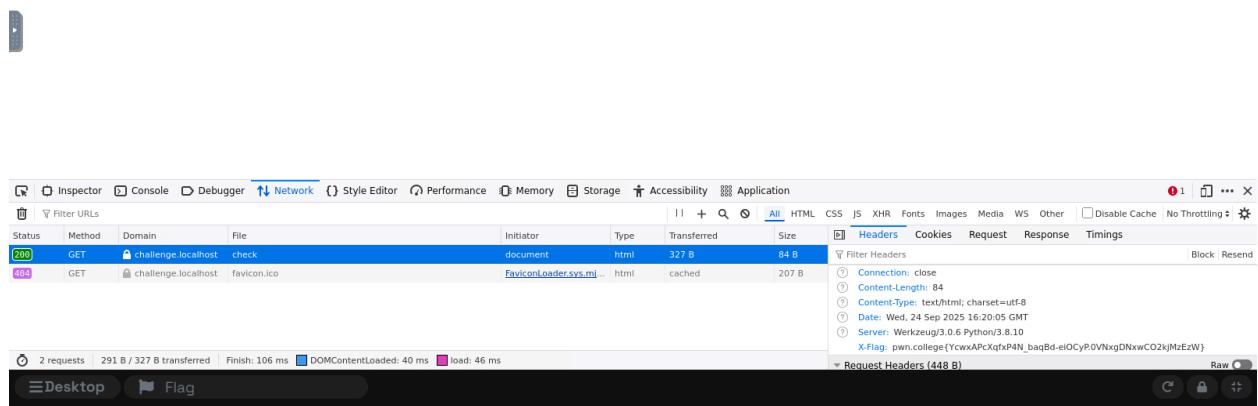
app = flask.Flask(__name__)

app.route("/check", methods=["GET"])
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!")

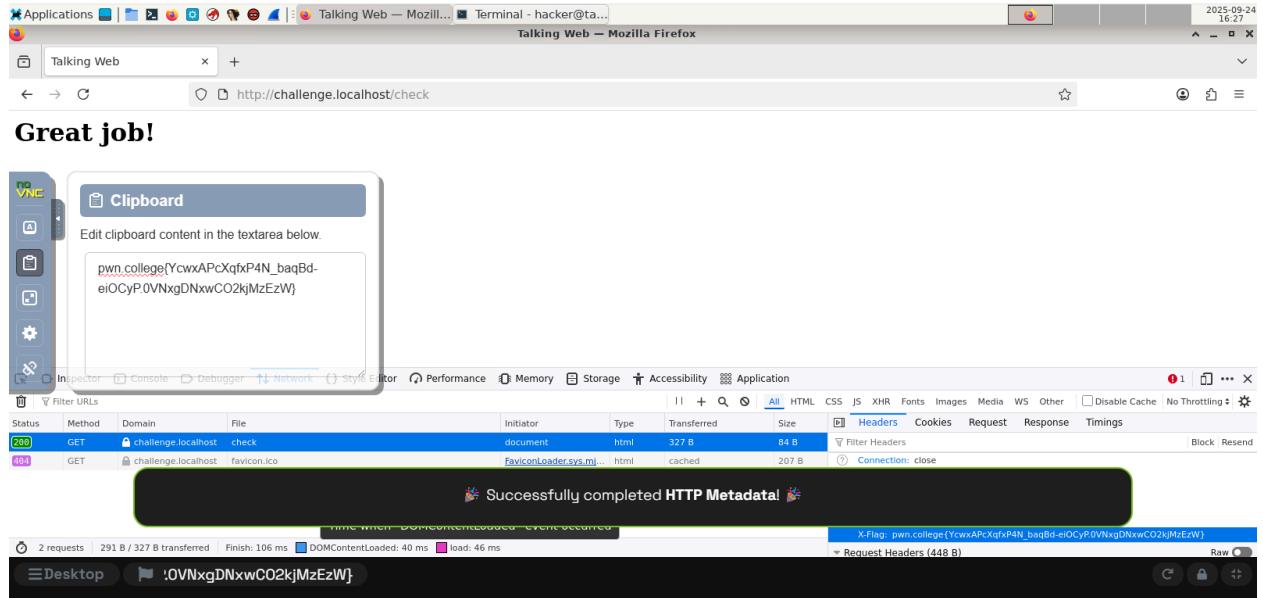
    response = flask.make_response(
        "<html><head><title>Talking Web</title></head><body><h1>Great job!</h1></body></html>"
    )
    response.headers["X-Flag"] = open("/flag").read().strip()
    return response

app.secret_key = os.urandom(8)
app.run("challenge.localhost", 80)
hacker@talking-web:~$ ./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 - - [24/Sep/2025 16:14:39] "GET /check HTTP/1.1" 200 -
127.0.0.1 - - [24/Sep/2025 16:16:04] "GET /check HTTP/1.1" 200 -
127.0.0.1 - - [24/Sep/2025 16:20:05] "GET /check HTTP/1.1" 200 -
```

Launching the challenge in GUI desktop terminal.



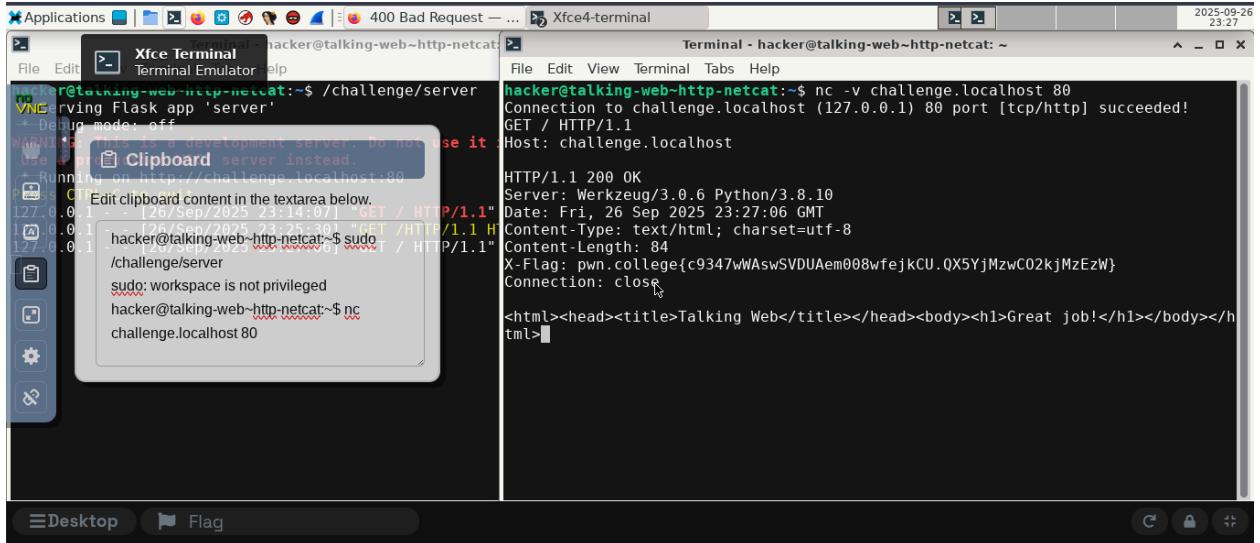
Retrieving the file called X-Flag in network tab to see the flag, under web developer tools after a refresh.



To capture the flag for this challenge, I first ran the `/challenge/server` script to start the web server, which began listening on `http://challenge.localhost:80`. I then accessed the correct endpoint `/check` using **Firefox** from the **GUI Desktop**, ensuring the `User-Agent` header was set correctly. Since the flag was embedded in a response header, I opened Firefox's **Web Developer Tools** via the menu (≡) > **More Tools** > **Web Developer Tools** and navigated to the **Network** tab. After reloading the page, I selected the `/check` request and examined the **Response Headers**, where I found the `X-Flag` header containing the flag:

`pwn.college{YcwxAPcXqfxP4N_baqBd-eiOCyP.0VNxgDNxwCO2kjMzEzW}`. This demonstrated the importance of inspecting HTTP metadata to retrieve hidden data. It reinforced that an HTTP response is more than just a body; critical data can be transmitted silently in headers, which are essential for controlling communication but are invisible in the standard browser view.

- HTTP (netcat)

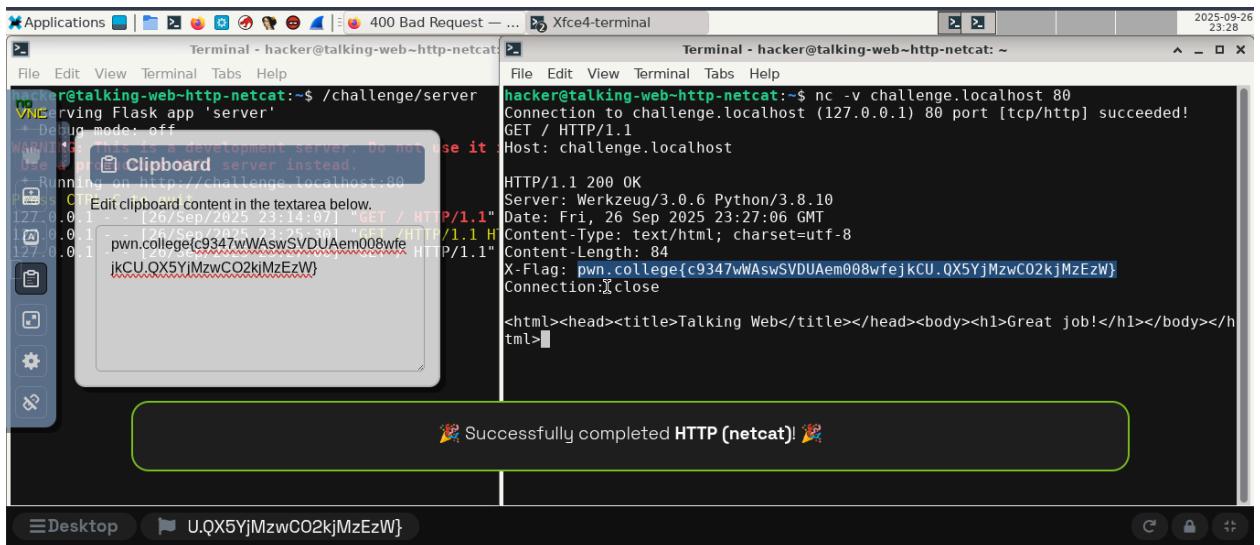


```

Xfce Terminal - hacker@talking-web~http-netcat:~ 
File Edit View Terminal Tabs Help
hacker@talking-web~http-netcat:~ $ /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
127.0.0.1 - [26/Sep/2025:23:14:07] "GET / HTTP/1.1"
127.0.0.1 - [26/Sep/2025:23:14:07] "GET / HTTP/1.1"
hacker@talking-web~http-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/1.1 200 OK
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Fri, 26 Sep 2025 23:27:06 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 84
X-Flag: pwn.college{c9347wWAswSVDUAem008wfejkCU.QX5YjMzwC02kjMzEzW}
Connection: close
<html><head><title>Talking Web</title></head><body><h1>Great job!</h1></body></html>

```

This is after connecting to netcat.



```

Xfce Terminal - hacker@talking-web~http-netcat:~ 
File Edit View Terminal Tabs Help
hacker@talking-web~http-netcat:~ $ /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
127.0.0.1 - [26/Sep/2025:23:14:07] "GET / HTTP/1.1"
127.0.0.1 - [26/Sep/2025:23:14:07] "GET / HTTP/1.1"
hacker@talking-web~http-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/1.1 200 OK
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Fri, 26 Sep 2025 23:27:06 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 84
X-Flag: pwn.college{c9347wWAswSVDUAem008wfejkCU.QX5YjMzwC02kjMzEzW}
Connection: close
<html><head><title>Talking Web</title></head><body><h1>Great job!</h1></body></html>

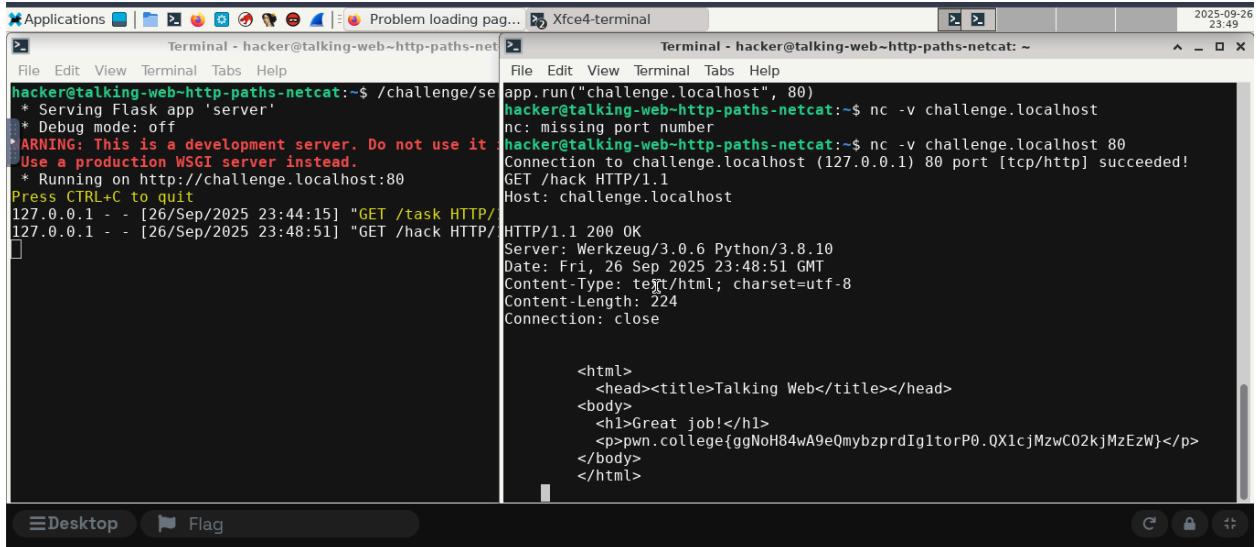
```

Successfully completed HTTP (netcat)!

I started by using **netcat** to establish a raw **TCP** connection to the **HTTP server** running on **challenge.localhost port 80**. Once connected, I manually crafted an **HTTP GET** request by typing **GET / HTTP/1.1** followed by the **Host: challenge.localhost** header, which is essential for **HTTP/1.1 requests** to specify the virtual host. After sending the request (by pressing Enter twice to indicate the end of the headers), the server responded with a full **HTTP response**. I carefully examined the response headers and found the flag in the **X-Flag** header, which contained the value

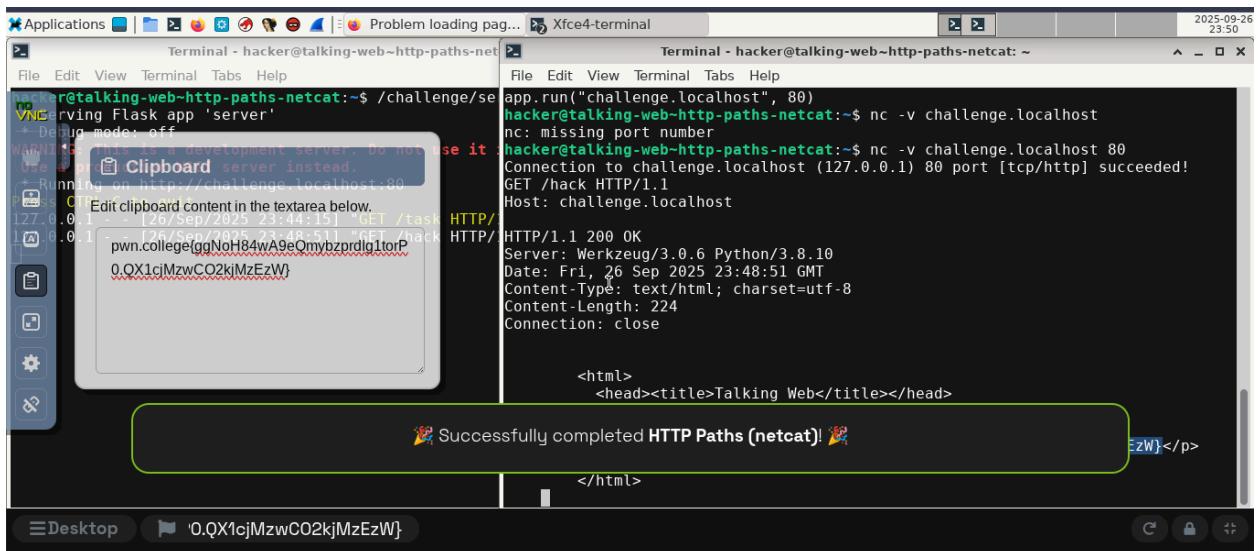
pwn.collegeC9347wWAswSVDUAem008wfe[KCL.OXSYjMzwC02kjMzEzWJ]. This flag was then displayed in the terminal, confirming that I had successfully retrieved it by interacting directly with the web server using **netcat**. Using netcat stripped away the abstraction of a browser, revealing the raw text-based protocol of HTTP and proving that any TCP-capable tool can be used to interact with web services.

- HTTP Paths (netcat)



The image shows two terminal windows side-by-side. The left terminal window shows the command `/challenge/se` being run, which starts a Flask development server. The right terminal window shows the command `nc -v challenge.localhost 80` being run, which connects to the server. The response from the server is a 200 OK status page with the content `<html><head><title>Talking Web</title></head><body><h1>Great job!</h1><p>pwn.college{ggNoH84wA9eQmybzprdIgtorP0.QX1cjMzwC02kjMzEzW}</p></body></html>`.

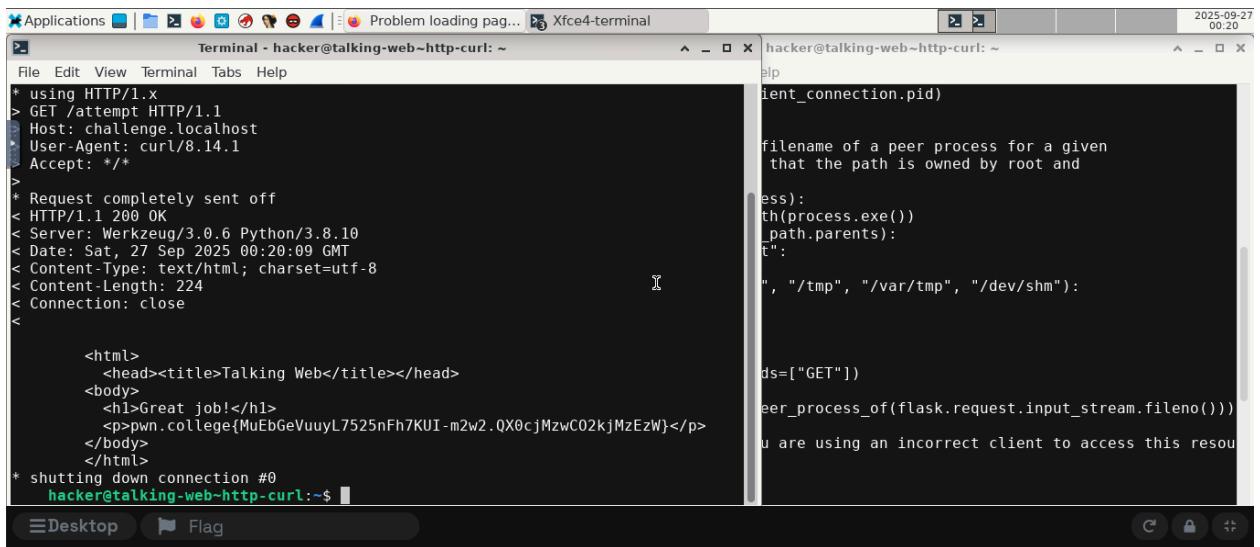
This is after connecting to netcat.



The image shows the same two terminal windows. The left terminal window shows the command `/challenge/se` being run. The right terminal window shows the command `nc -v challenge.localhost 80` being run. The response from the server is a 200 OK status page with the content `<html><head><title>Talking Web</title></head><body><h1>Great job!</h1><p>pwn.college{ggNoH84wA9eQmybzprdIgtorP0.QX1cjMzwC02kjMzEzW}</p></body></html>`. A red box highlights the flag in the clipboard content: `pwn.college{ggNoH84wA9eQmybzprdIgtorP0.QX1cjMzwC02kjMzEzW}`. A green box highlights the message `Successfully completed HTTP Paths (netcat)!`.

I started by running **netcat** to connect to the web server at **challenge.localhost** on **port 80**. Once connected, I sent an **HTTP GET** request to the **path /hack** with the required **Host** header. The server responded with a **200 OK status**, indicating that I had successfully accessed the correct path. This completed the challenge, and the flag was revealed in the clipboard content as **pwn.college{ggNoH84wA9eQmybzprdIgtorP0.QX1cjMzwC02kjMzEzW}**. By manually crafting the request to the specific path, I was able to retrieve the flag and complete the **HTTP Paths challenge**. This emphasized how path-based routing works: the server interprets the path in the request line to determine which specific resource or function to execute on the server side.

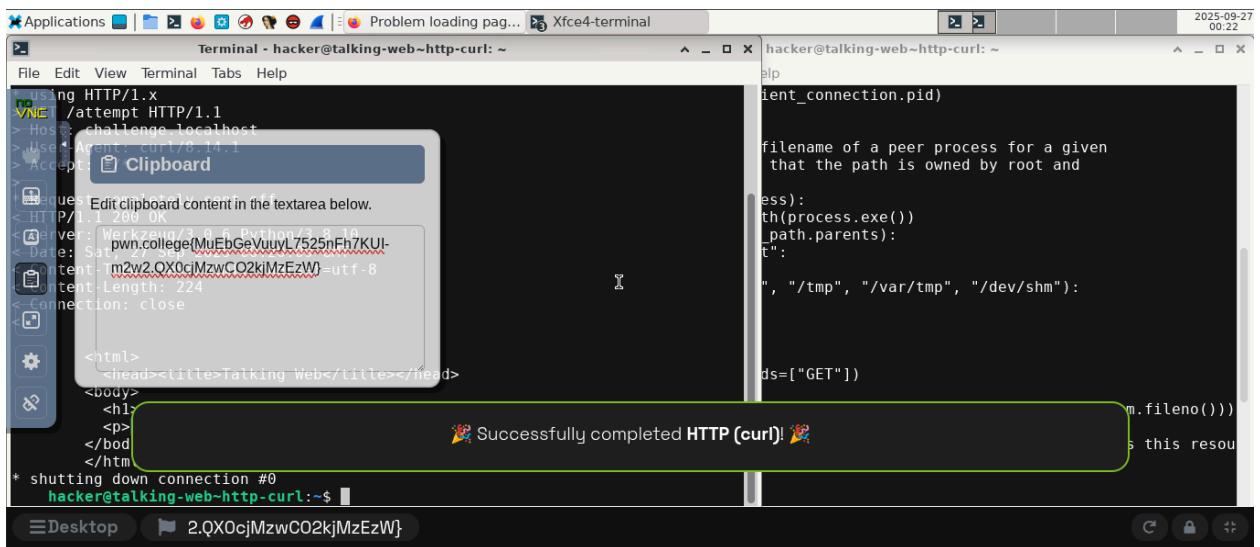
- HTTP (curl)



```
* using HTTP/1.0
> GET /attempt HTTP/1.0
> Host: challenge.localhost
> User-Agent: curl/8.14.1
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< Server: Werkzeug/3.0.6 Python/3.8.10
< Date: Sat, 27 Sep 2025 00:20:09 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>pwn.college{MuEbGeVuuyL7525nFh7KUJ-m2w2.CX0cjMzwC02kjMzEzW}</p>
</body>
</html>
* shutting down connection #0
hacker@talking-web-~http-curl:~$
```

This is after retrieving the endpoint /attempt



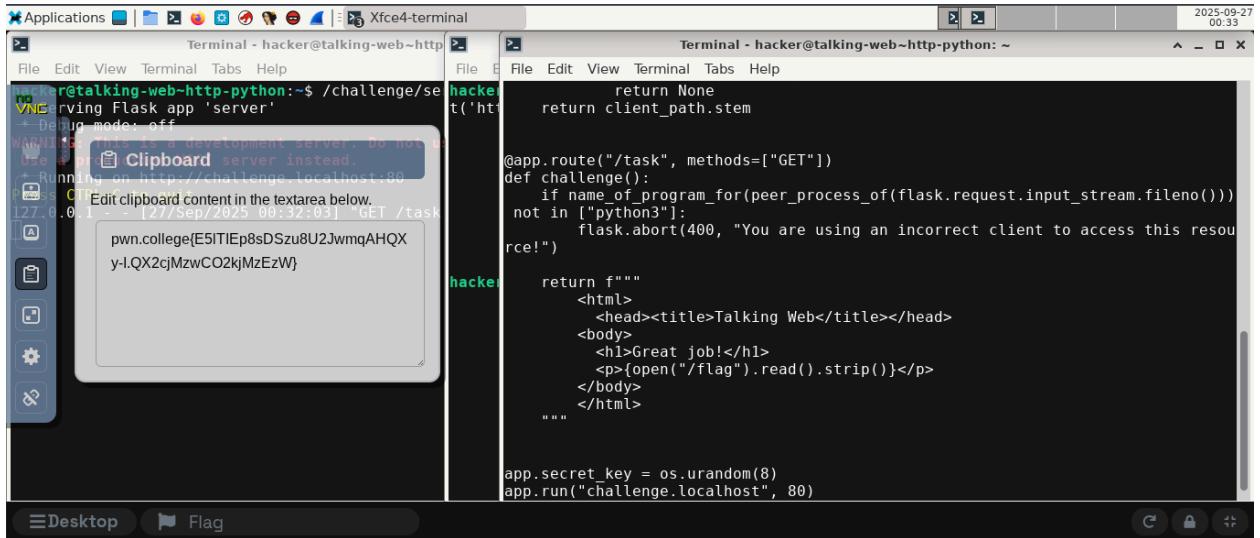
```
* using HTTP/1.0
> /attempt HTTP/1.0
> Host: challenge.localhost
> User-Agent: curl/8.14.1
> Accept: Clipboard
>
< HTTP/1.1 200 OK
< Server: Werkzeug/3.0.6 Python/3.8.10
< Date: Sat, 27 Sep 2025 00:20:09 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 224
< Connection: close
<

<html>
<head><title>Talking Web</title></head>
<body>
<h1>Great job!</h1>
<p>pwn.college{MuEbGeVuuyL7525nFh7KUJ-m2w2.CX0cjMzwC02kjMzEzW}</p>
</body>
</html>
* shutting down connection #0
hacker@talking-web-~http-curl:~$
```

Successfully completed HTTP (curl)!

I used curl to send an **HTTP GET request** to the server at **challenge.localhost** on **port 80**, specifically targeting the **/attempt** path. After executing the command, the server responded with a **200 OK status**, and the flag was displayed in the response output. I then copied the flag from the terminal output, which was **pwn.college{MuEbGeVuuyL7525nFh7KUJ-m2w2.CX0cjMzwC02kjMzEzW}**, confirming the successful completion of the **HTTP challenge** with **curl**. Using curl provided a powerful and scriptable command-line method for making HTTP requests, showcasing an essential tool for testing and automating web interactions outside of a graphical browser.

- HTTP (python)



```

Terminal - hacker@talking-web~http:~$ /challenge/server
[+] Serving Flask app 'server'
* Debug mode: off
[+] WARNING: This is a development server. Do not use it on a production deployment
[+] Use [+] for Clipboard server instead.
[+] Running on http://challenge.localhost:80
[+] 127.0.0.1 - [27/Sep/2025 00:32:03] "GET /task" 200 134
pwn.college(E5ITIEp8sDSzu8U2JwmqAHQXy-I.QX2cjMzwCO2kjMzEzW)

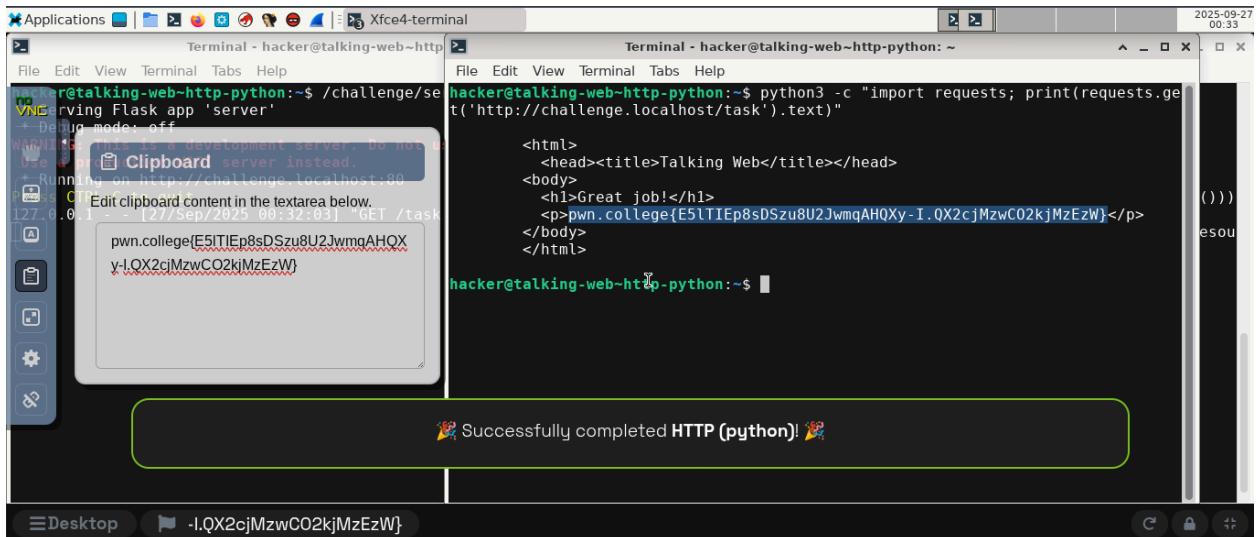
Terminal - hacker@talking-web~http-python:~$ 
def challenge():
    if name_of_program_for(peer_process_of(flask.request.input_stream.fileno())) not in ["python3"]:
        flask.abort(400, "You are using an incorrect client to access this resource!")

    return f"""
        <html>
            <head><title>Talking Web</title></head>
            <body>
                <h1>Great job!</h1>
                <p>pwn.college(E5ITIEp8sDSzu8U2JwmqAHQXy-I.QX2cjMzwCO2kjMzEzW)</p>
            </body>
        </html>
    """

app.secret_key = os.urandom(8)
app.run("challenge.localhost", 80)

```

Used cat /challenge/server to get the endpoint of /task



```

Terminal - hacker@talking-web~http:~$ /challenge/server
[+] Serving Flask app 'server'
* Debug mode: off
[+] WARNING: This is a development server. Do not use it on a production deployment
[+] Use [+] for Clipboard server instead.
[+] Running on http://challenge.localhost:80
[+] 127.0.0.1 - [27/Sep/2025 00:32:03] "GET /task" 200 134
pwn.college(E5ITIEp8sDSzu8U2JwmqAHQXy-I.QX2cjMzwCO2kjMzEzW)

Terminal - hacker@talking-web~http-python:~$ python3 -c "import requests; print(requests.get('http://challenge.localhost/task').text)"
<html>
    <head><title>Talking Web</title></head>
    <body>
        <h1>Great job!</h1>
        <p>pwn.college(E5ITIEp8sDSzu8U2JwmqAHQXy-I.QX2cjMzwCO2kjMzEzW)</p>
    </body>
</html>

hacker@talking-web~http-python:~$ 

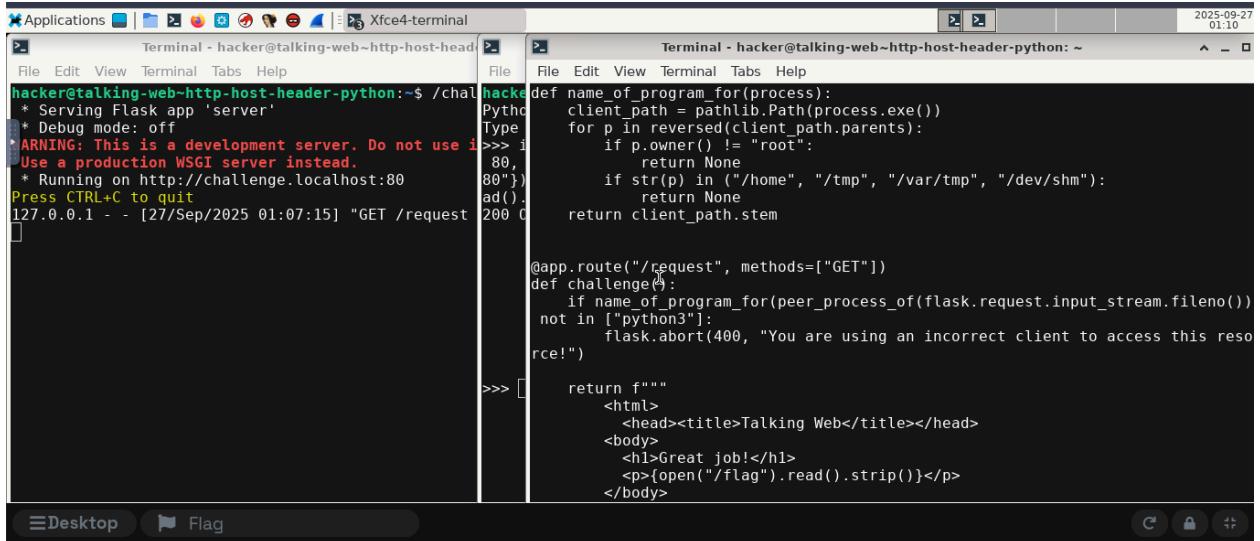
```

Successfully completed HTTP (python)!

I started by running the **Flask app server** for the challenge, which set up the local environment. Then, I opened a terminal and used **Python 3** with the `requests` library to send a **GET request** to `http://challenge.localhost/task`. The command I executed was `python3 -c "import requests; print(requests.get('http://challenge.localhost/task').text)"`. The server responded with an **HTML page** that contained the flag within a `paragraph` tag. I saw the flag printed directly in the terminal output:

`pwn.college(E5ITIEp8sDSzu8U2JwmqAHQXy-I.QX2cjMzwCO2kjMzEzW)`. This confirmed that I successfully completed the **HTTP challenge** using Python. This approach demonstrated the power of using a scripting language like Python to programmatically interact with web servers, a foundational technique for building automated clients and security tools.

- HTTP Host Header (python)



```

Terminal - hacker@talking-web~http-host-header-python:~$ /challenge
* Serving Flask app 'server'
* Debug mode: off
WARNING: This is a development server. Do not use in a
Use a production WSGI server instead.
* Running on http://challenge.localhost:80
Press CTRL+C to quit
127.0.0.1 - - [27/Sep/2025 01:07:15] "GET /request" 200 OK

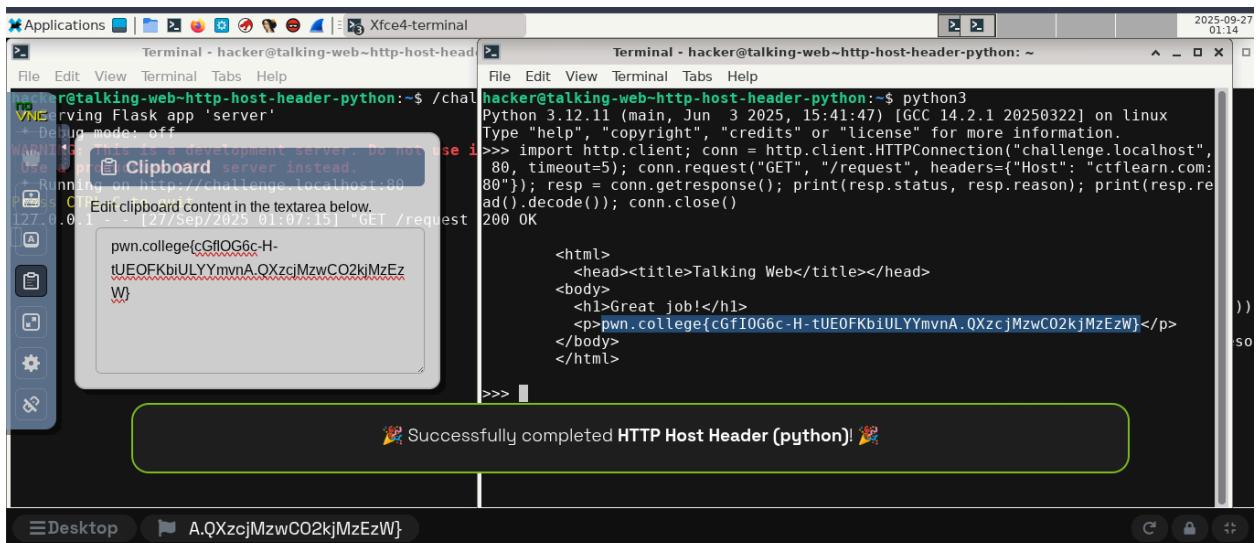
Terminal - hacker@talking-web~http-host-header-python:~$ 
File Edit View Terminal Tabs Help
def name_of_program_for(process):
    client_path = pathlib.Path(process.exe())
    for p in reversed(client_path.parents):
        if p.owner() != "root":
            return None
        if str(p) in ("/home", "/tmp", "/var/tmp", "/dev/shm"):
            return None
    return client_path.stem

@app.route("/request", methods=["GET"])
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!")
    return f"""
        <html>
            <head><title>Talking Web</title></head>
            <body>
                <h1>Great job!</h1>
                <p>{open("/flag").read().strip()}</p>
            </body>
        </html>
    """

>>> 

```

Used cat /challenge/server to get the endpoint of /request



```

Terminal - hacker@talking-web~http-host-header-python:~$ /challenge
* Serving Flask app 'server'
* Debug mode: off
WARNING: This is a development server. Do not use in a
Use a production WSGI server instead.
* Running on http://challenge.localhost:80
Press CTRL+C to quit
127.0.0.1 - - [27/Sep/2025 01:07:15] "GET /request" 200 OK

Terminal - hacker@talking-web~http-host-header-python:~$ 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 http.client; conn = http.client.HTTPConnection("challenge.localhost", 80, timeout=5); conn.request("GET", "/request", headers={"Host": "ctflearn.com:80"}); resp = conn.getresponse(); print(resp.status, resp.reason); print(resp.read().decode()); conn.close()
200 OK

<html>
    <head><title>Talking Web</title></head>
    <body>
        <h1>Great job!</h1>
        <p>pwn.college[cGI0056c-H-ULEOFKBJULYYmwnA.0XzcjMzwCO2kjMzEzW]</p>
    </body>
</html>

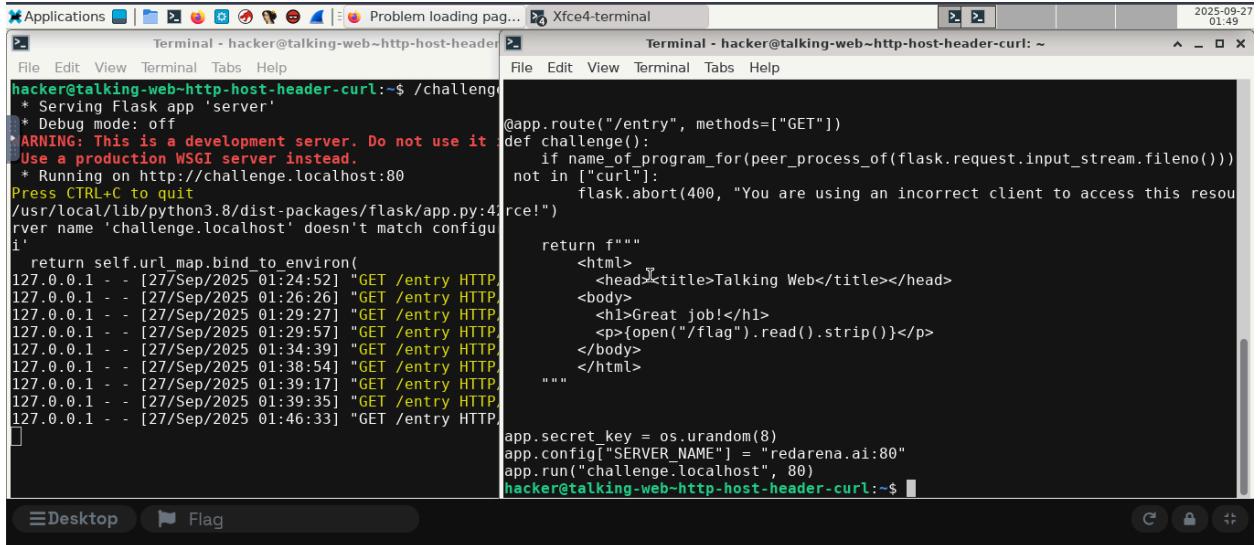
>>> 

```

Successfully completed HTTP Host Header (python)!

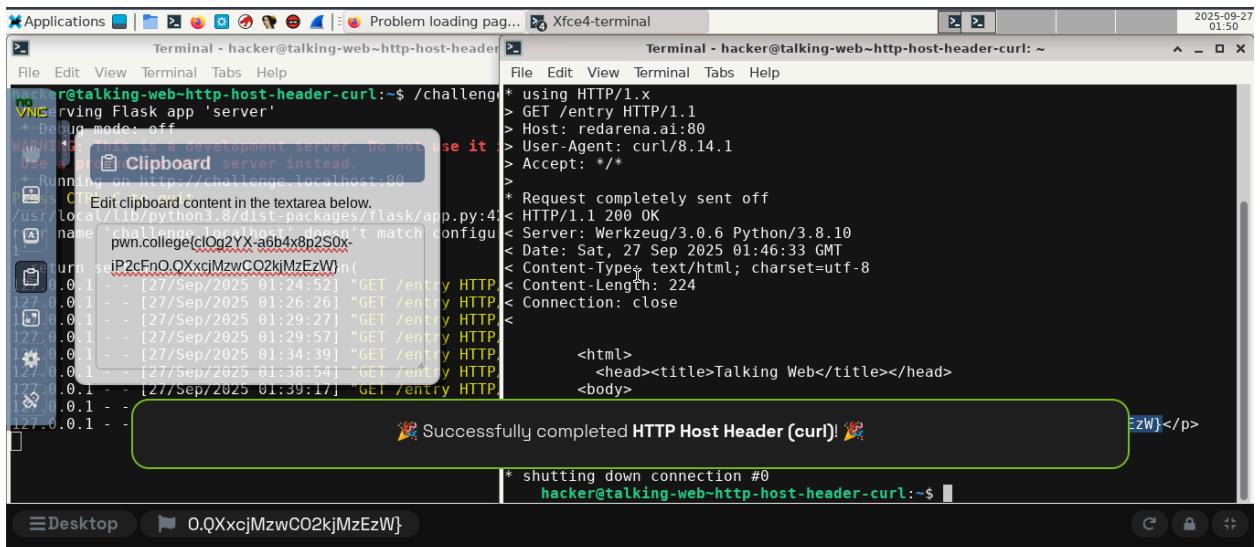
I started by importing the `http.client` module in Python to handle the **HTTP connection**. Then, I established a connection to **challenge.localhost** on port **80** with a timeout of 5 seconds. I crafted a **GET request** to the path **/request** and set the **Host header** to **ctflearn.com:80** to meet the challenge requirements. After sending the request, I retrieved the response, which had a status code of **200 OK**. The response body contained an **HTML page** with the flag embedded in a paragraph tag, which I printed out. The flag was **pwn.college[cGI0056c-H-ULEOFKBJULYYmwnA.0XzcjMzwCO2kjMzEzW]**, confirming that I successfully completed the **HTTP Host Header** challenge. Manipulating the Host header via script highlighted its critical role in virtual hosting, where a single server can host multiple websites and uses this header to determine which site to serve.

- HTTP Host Header (curl)



The image shows two terminal windows side-by-side. The left terminal window is titled 'Terminal - hacker@talking-web~http-host-headercurl:' and shows the command `/challenge` being run. The output indicates that the server is a Flask app 'server' in 'Debug mode: off'. It also shows a warning: 'WARNING: This is a development server. Do not use it in a production WSGI server instead.' and a note about the port: 'Running on http://challenge.localhost:80'. The right terminal window is titled 'Terminal - hacker@talking-web~http-host-headercurl: ~' and shows the source code of the Flask application. The code defines an endpoint `@app.route("/entry", methods=["GET"])` that returns an HTML response with the text 'Great job!' and a flag placeholder. The code also includes configuration for `app.secret_key` and `app.config["SERVER_NAME"]`.

Used cat /challenge/server to get the endpoint of /entry

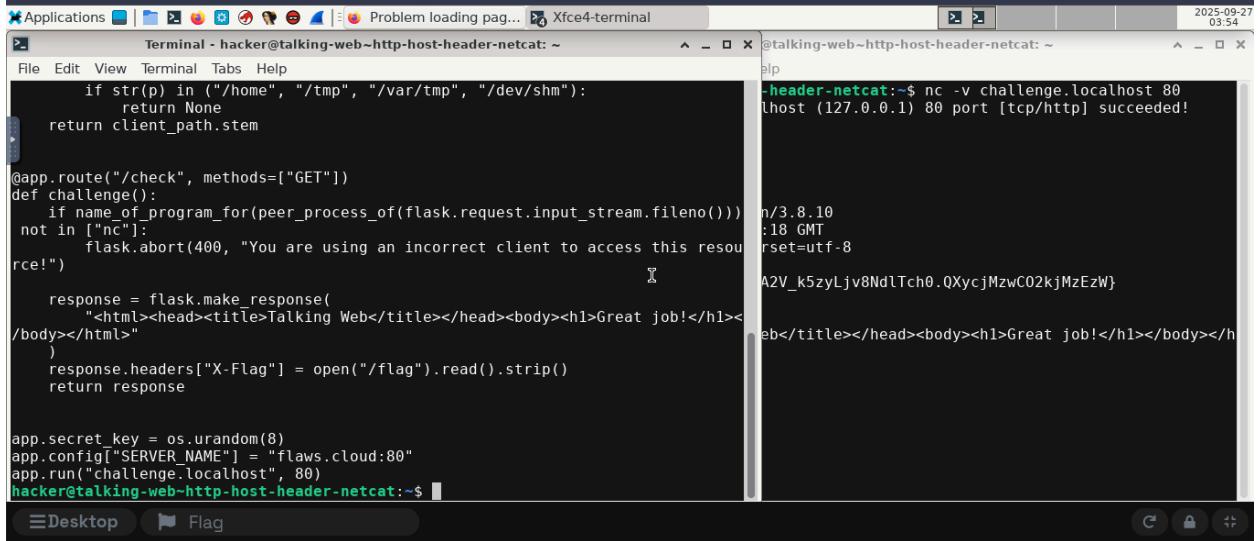


The image shows a terminal window with two tabs. The left tab is titled 'Terminal - hacker@talking-web~http-host-headercurl:' and shows the command `/challenge` being run. The right tab is titled 'Terminal - hacker@talking-web~http-host-headercurl: ~'. The user is using curl to send a GET request to the /entry endpoint with a Host header of 'redarena.ai:80'. The response shows the server using HTTP/1.1, returning a 200 OK status, and the HTML content 'Great job!'. A message at the bottom of the terminal window says 'Successfully completed HTTP Host Header (curl)!'. The terminal window also shows the clipboard content containing the flag.

I started by using curl to send an **HTTP GET** request to the server at **challenge.localhost** on **port 80**. I specified the **path /entry** and set the **Host header** to **redarena.ai:80** with the appropriate **curl** options. The server responded with a **200 OK** status, and the HTML body confirmed the request was successful. After completing the challenge, I retrieved the flag from the clipboard, which was

pwn.college(c)Q2YX-a9b4x8pZSQX-B2cFnO.QXxcjMzwCQ2kJhzEzWj. This verified that I had correctly manipulated the Host header to satisfy the challenge requirements. This exercise showcased curl's precision in crafting custom headers, a vital capability for testing web application security and behavior under different hosting conditions.

- HTTP Host Header (netcat)



```

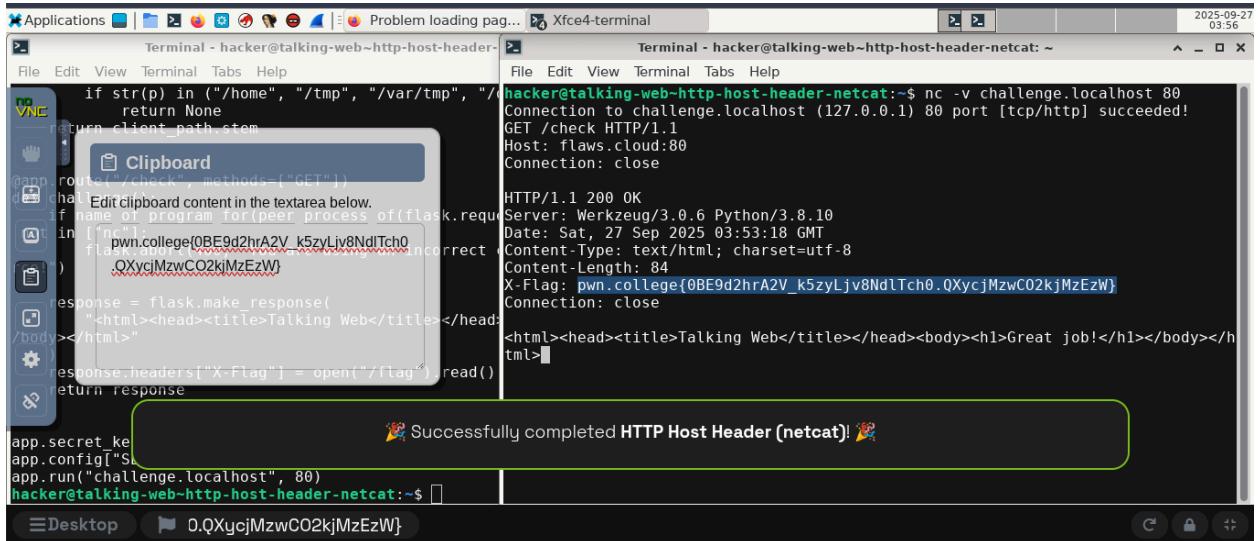
Applications Terminal - hacker@talking-web~http-host-header-netcat: ~
Terminal - hacker@talking-web~http-host-header-netcat: ~
File Edit View Terminal Tabs Help
if str(p) in ("/home", "/tmp", "/var/tmp", "/dev/shm"):
    return None
return client_path.stem

@app.route("/check", methods=["GET"])
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!")
    response = flask.make_response(
        "<html><head><title>Talking Web</title></head><body><h1>Great job!</h1></body></html>"
    )
    response.headers["X-Flag"] = open("/flag").read().strip()
    return response

app.secret_key = os.urandom(8)
app.config["SERVER_NAME"] = "flaws.cloud:80"
app.run("challenge.localhost", 80)
hacker@talking-web~http-host-header-netcat: ~

```

Used cat /challenge/server to get the endpoint of /check, Host: flaws.cloud:80



```

Applications Terminal - hacker@talking-web~http-host-header-netcat: ~
Terminal - hacker@talking-web~http-host-header-netcat: ~
File Edit View Terminal Tabs Help
if str(p) in ("/home", "/tmp", "/var/tmp", "/dev/shm"):
    return None
return client_path.stem

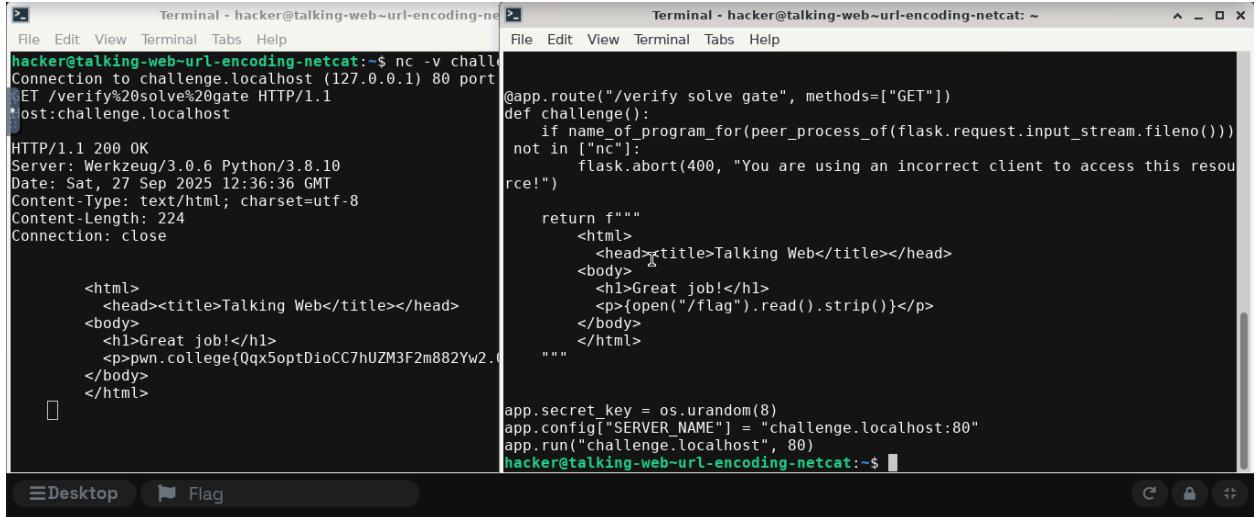
@app.route("/check", methods=["GET"])
def challenge():
    if name_of_program_for(peer_process_of(flask.request.input_stream.fileno())):
        not in ["nc"]:
            flask.abort(400, "You are using an incorrect client to access this resource!")
    response = flask.make_response(
        "<html><head><title>Talking Web</title></head><body><h1>Great job!</h1></body></html>"
    )
    response.headers["X-Flag"] = open("/flag").read().strip()
    return response

app.secret_key = os.urandom(8)
app.config["SERVER_NAME"] = "flaws.cloud:80"
app.run("challenge.localhost", 80)
hacker@talking-web~http-host-header-netcat: ~

```

I started by using **netcat** to establish a **TCP connection** to the server at **challenge.localhost** on **port 80**. Once connected, I manually crafted an **HTTP GET** request to the **path /check** with the **Host header** set to **{lans.cloud:80}** and added a **Connection: close** header to ensure the connection would close after the response. After sending the request, the server responded with a **200 OK** status, and I found the flag in the **X-Flag** header of the response: **pwn.college(OBE9d2hrA2V_K5zyLjv8Nd1Tch0_0XycjMzwC02kjMzEzW)**. This confirmed that I successfully completed the **HTTP Host Header** challenge by manipulating the **Host header** correctly. Manually setting the **Host header** with **netcat** cemented the understanding that this header is not magical but a simple, client-sent directive that is fundamental to how the modern web functions.

- URL Encoding (netcat)



```

Terminal - hacker@talking-web~url-encoding-netcat:~$ nc -v challenge.localhost 80
Connection to challenge.localhost (127.0.0.1) 80 port [tcp/http] succeeded!
GET /verify%20solve%20gate HTTP/1.1
Host:challenge.localhost

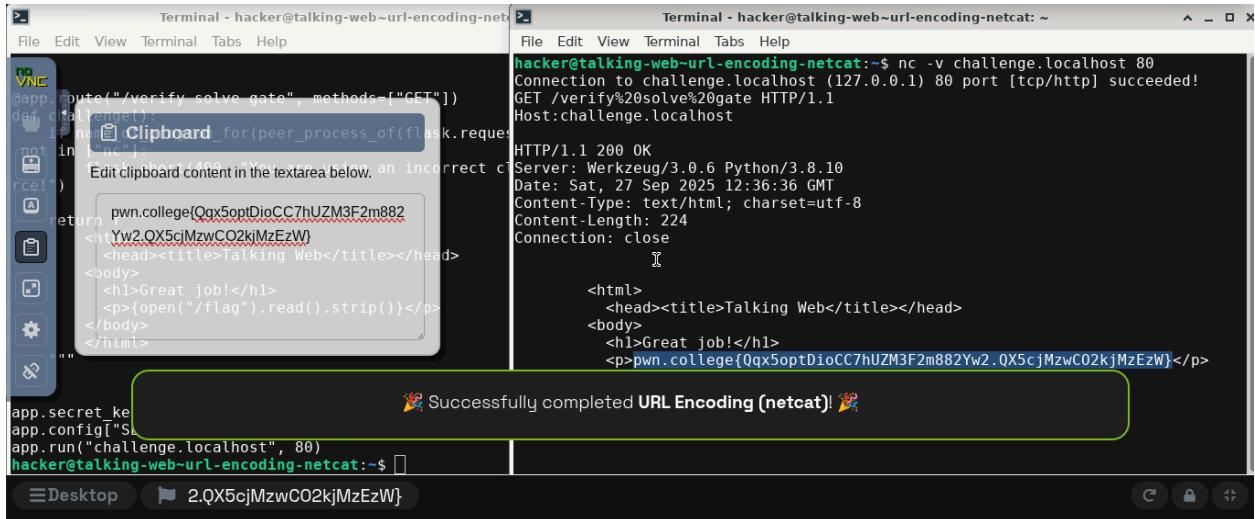
HTTP/1.1 200 OK
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Sat, 27 Sep 2025 12:36:36 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 224
Connection: close

<html>
  <head><title>Talking Web</title></head>
  <body>
    <h1>Great job!</h1>
    <p>pwn.college{Qqx5optDioCC7hUZM3F2m882Yw2.QX5GjMzwCO2kjMzEzW}</p>
  </body>
</html>

app.secret_key = os.urandom(8)
app.config["SERVER_NAME"] = "challenge.localhost:80"
app.run("challenge.localhost", 80)
hacker@talking-web~url-encoding-netcat:~$ 

```

Used cat /challenge/server to get the endpoint of /verify solve gate



```

Terminal - hacker@talking-web~url-encoding-netcat:~$ nc -v challenge.localhost 80
Connection to challenge.localhost (127.0.0.1) 80 port [tcp/http] succeeded!
GET /verify%20solve%20gate HTTP/1.1
Host:challenge.localhost

HTTP/1.1 200 OK
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Sat, 27 Sep 2025 12:36:36 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>pwn.college{Qqx5optDioCC7hUZM3F2m882Yw2.QX5GjMzwCO2kjMzEzW}</p>
  </body>
</html>

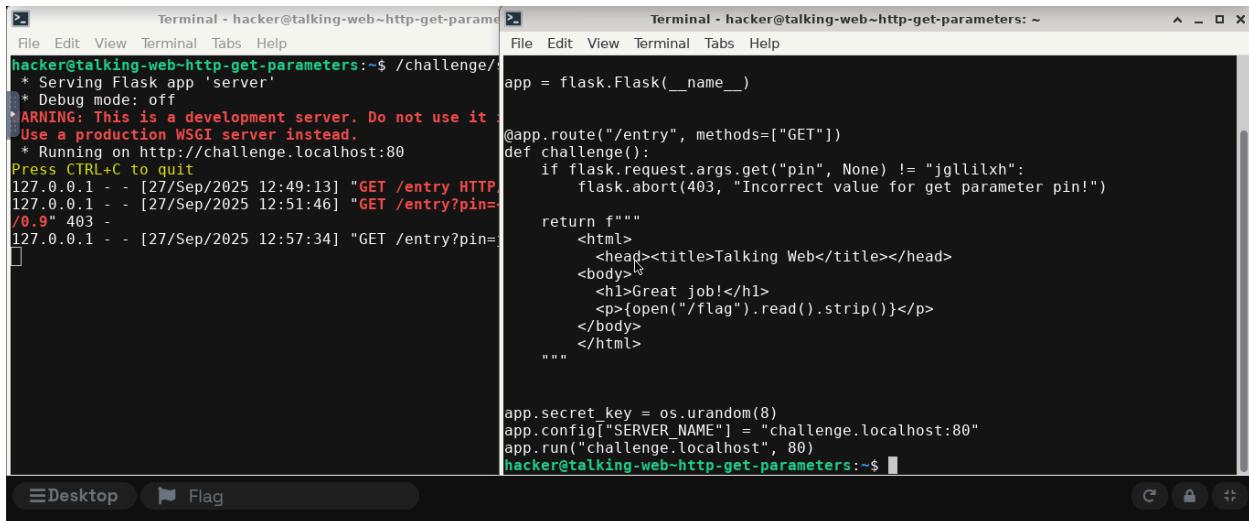
app.secret_key = os.urandom(8)
app.config["SERVER_NAME"] = "challenge.localhost:80"
app.run("challenge.localhost", 80)
hacker@talking-web~url-encoding-netcat:~$ 

```

I started by using **netcat** to connect to the web server at **challenge.localhost** on **port 80**. Once connected, I sent a **GET request** to the **path /verify%20solve%20gate**, which included **URL encoding** for the spaces (using `%20`). I also included the **Host header** set to **challenge.localhost**. The server responded with a **200 OK status**, and the **HTML body** contained the flag within a paragraph tag:

pwn.college{Qqx5optDioCC7hUZM3F2m882Yw2.QX5GjMzwCO2kjMzEzW}. This confirmed that I successfully completed the URL encoding challenge by properly encoding the path in the HTTP request. This illustrated why URL encoding is necessary: it allows characters with special meanings (like spaces) to be safely transmitted within a URL, ensuring the server correctly parses the intended path and parameters.

- HTTP Get Parameters



```

Terminal - hacker@talking-web~http-get-parameters:~$ /challenge/
* Serving Flask app 'server'
* Debug mode: off
WARNING: This is a development server. Do not use it
Use a production WSGI server instead.
* Running on http://challenge.localhost:80
Press CTRL+C to quit
127.0.0.1 - - [27/Sep/2025 12:49:13] "GET /entry HTTP/1.1"
127.0.0.1 - - [27/Sep/2025 12:51:46] "GET /entry?pin=jgllilxh HTTP/1.1"
127.0.0.1 - - [27/Sep/2025 12:57:34] "GET /entry?pin=jgllilxh HTTP/1.1"
hacker@talking-web~http-get-parameters:~$ 

Terminal - hacker@talking-web~http-get-parameters:~$ 
app = flask.Flask(__name__)

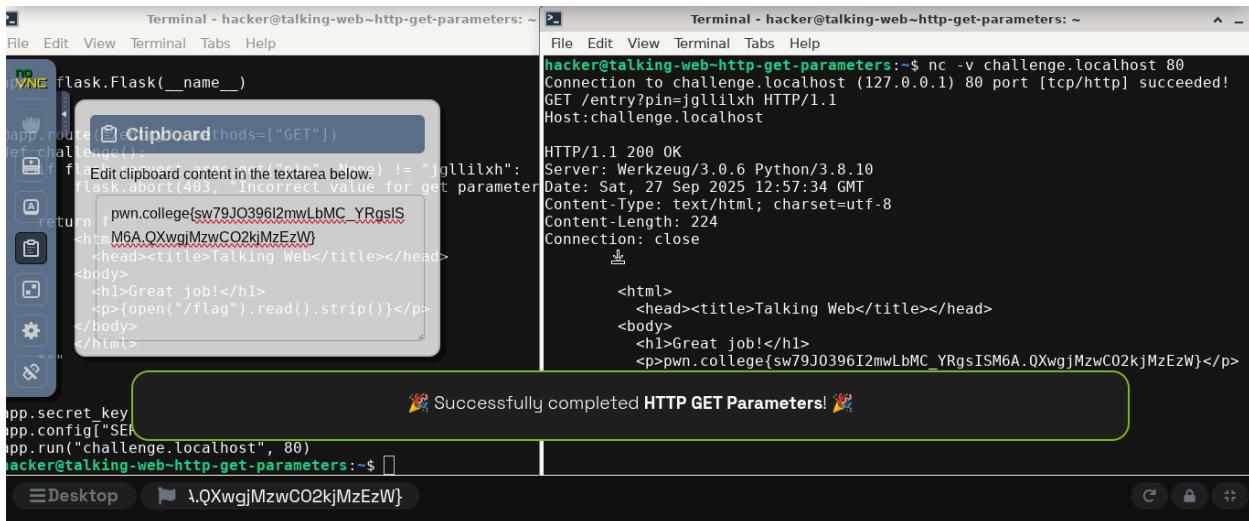
@app.route("/entry", methods=["GET"])
def challenge():
    if flask.request.args.get("pin", None) != "jgllilxh":
        flask.abort(403, "Incorrect value for get parameter pin!")

    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-get-parameters:~$ 

```

Used cat /challenge/server to get the endpoint of /entry



```

Terminal - hacker@talking-web~http-get-parameters:~$ 
flask.Flask(__name__)

@app.route("/entry", methods=["GET"])
def challenge():
    if flask.request.args.get("pin", None) != "jgllilxh":
        flask.abort(403, "Incorrect value for get parameter pin!")

    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-get-parameters:~$ 

Terminal - hacker@talking-web~http-get-parameters:~$ nc -v challenge.localhost 80
Connection to challenge.localhost (127.0.0.1) 80 port [tcp/http] succeeded!
GET /entry?pin=jgllilxh HTTP/1.1
Host:challenge.localhost

HTTP/1.1 200 OK
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Sat, 27 Sep 2025 12:57:34 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 224
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 224

<html>
    <head><title>Talking Web</title></head>
    <body>
        <h1>Great job!</h1>
        <p>pwn.college{sw79J0396I2mwLbMC_YRqsISM6A.QXwgjMzwCO2kjMzEzw0}</p>
    </body>
</html>

Successfully completed HTTP GET Parameters!

```

I started by using **netcat** to connect to the web server at challenge.localhost on port 80. Once connected, I manually crafted an **HTTP GET request** to the **path /entry** with the query parameter **pin=jgllilxh**. I also included the **Host header** set to **challenge.localhost**. After sending the request, the server responded with a **200 OK status**, and the **HTML body** contained the flag within a paragraph tag:

pwn.college(sw7900396f2mwLbMC_YRqsISM6A.QXwgjMzwCO2kjMzEzw0). This confirmed that I successfully retrieved the flag by including the correct GET parameter in the request. This demonstrated the primary method of passing user-supplied data to a server via the URL, forming the basis for dynamic content, search queries, and statefulness in web applications.

- Multiple HTTP Parameters (netcat)

```

Terminal - hacker@talking-web~multiple-http-parameters-netcat: ~
File Edit View Terminal Tabs Help
app.route("/request", methods=["GET"])
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.args.get("access", None) != "ejnskvxx":
        flask.abort(403, "Incorrect value for get parameter access!")
    if flask.request.args.get("token", None) != "rmxwpdzo":
        flask.abort(403, "Incorrect value for get parameter token!")
    if flask.request.args.get("signature", None) != "fhhmtasz":
        flask.abort(403, "Incorrect value for get parameter signature!")

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

```

```

http-parameters-netcat:~$ nc -v challenge.localhost
lhost (127.0.0.1) 80 port [tcp/http] succeeded!
$token=rmxwpdzo&signature=fhhmtasz HTTP/1.1

n/3.8.10
:05 GMT
rset=utf-8

ng Web</title></head>
>
VSujT9zaHlWgmKWqjIMIF9rk.QX0gjMzwCO2kjMzEzW)</p>

```

Used cat /challenge/server to get the endpoint of /request with the following parameter access=ejnskvxx, token=rmxwpdzo, and signature=fhhmtasz

```

Terminal - hacker@talking-web~multiple-http-parameters-netcat: ~
File Edit View Terminal Tabs Help
VNC
app.route("/request", methods=["GET"])
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.args.get("access", None) != "ejnskvxx":
        flask.abort(403, "Incorrect value for get parameter access!")
    if flask.request.args.get("token", None) != "rmxwpdzo":
        flask.abort(403, "Incorrect value for get parameter token!")
    if flask.request.args.get("signature", None) != "fhhmtasz":
        flask.abort(403, "Incorrect value for get parameter signature!")

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

```

```

Terminal - hacker@talking-web~multiple-http-parameters-netcat: ~
File Edit View Terminal Tabs Help
hacker@talking-web~multiple-http-parameters-netcat:~$ nc -v challenge.localhost
80
Connection to challenge.localhost (127.0.0.1) 80 port [tcp/http] succeeded!
GET /request?access=ejnskvxx&token=rmxwpdzo&signature=fhhmtasz HTTP/1.1
Host: challenge.localhost
HTTP/1.1 200 OK
Server: Werkzeug/3.0.6 Python/3.8.10
Date: Sat, 27 Sep 2025 13:13:05 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 224
Connection: close

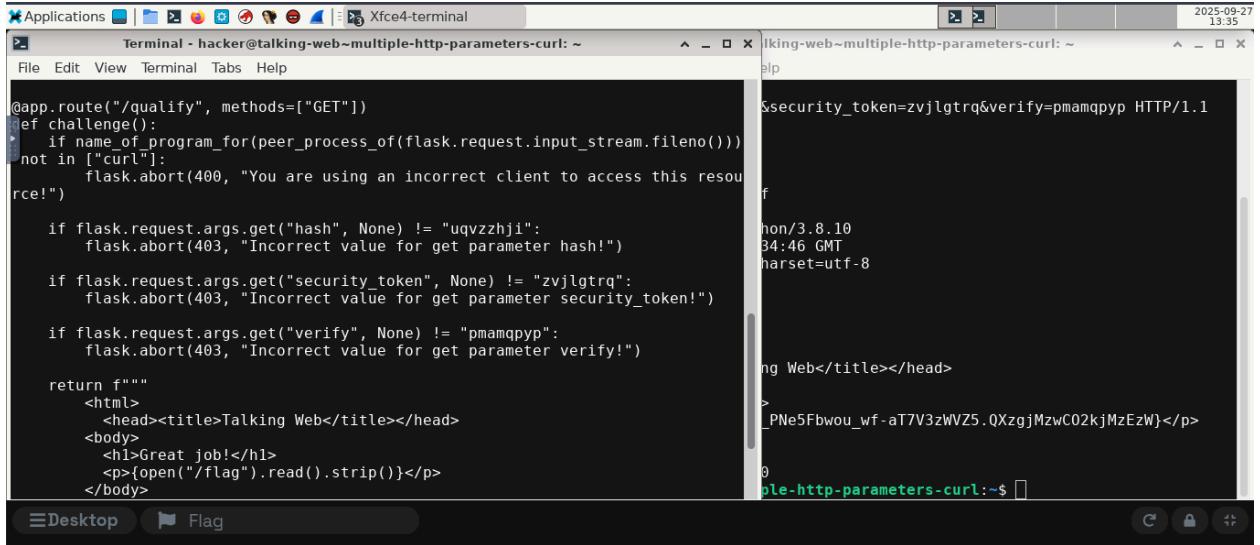
<html>
    <head><title>Talking Web</title></head>
    <body>
        <h1>Great job!</h1>
        <p>k.QX0gjMzwCO2kjMzEzW</p>

```

I started by using **netcat** to connect to the web server at **challenge.localhost** on **port 80**. Once connected, I crafted an **HTTP GET** request to the **appropriate path**, which included multiple **query parameters**: **"access"** and **"token"** and **"signature"** with their correct values. I made sure to include both **parameters in the query string**, as the server would abort with a **403 error** if either was missing or incorrect. After sending the request, the server responded with a **200 OK** status, and the flag was displayed in the response body:

pwn.college(ABqVSJLT9zatHWqmKVwUHMF9)tk.QX0gMxwCO2kNMEzW. This confirmed that I successfully completed the challenge by providing the required multiple HTTP parameters. The challenge illustrated how servers often require a specific set of parameters to authorize an action, and how missing or incorrect parameters lead to defined error states like 403 Forbidden.

- Multiple HTTP Parameters (curl)



The terminal window shows the source code of a Flask application. The code defines a route for '/quality' that checks if the client is a 'curl' user. If not, it returns a 400 error. It then checks for 'hash', 'security_token', and 'verify' parameters. If any are missing or incorrect, it returns a 403 error. Otherwise, it returns a success response with a flag.

```

@app.route("/quality", methods=["GET"])
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.args.get("hash", None) != "uqvzzhji":
        flask.abort(403, "Incorrect value for get parameter hash!")

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

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

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

```

The right side of the terminal shows the curl command and its output. The curl command is:

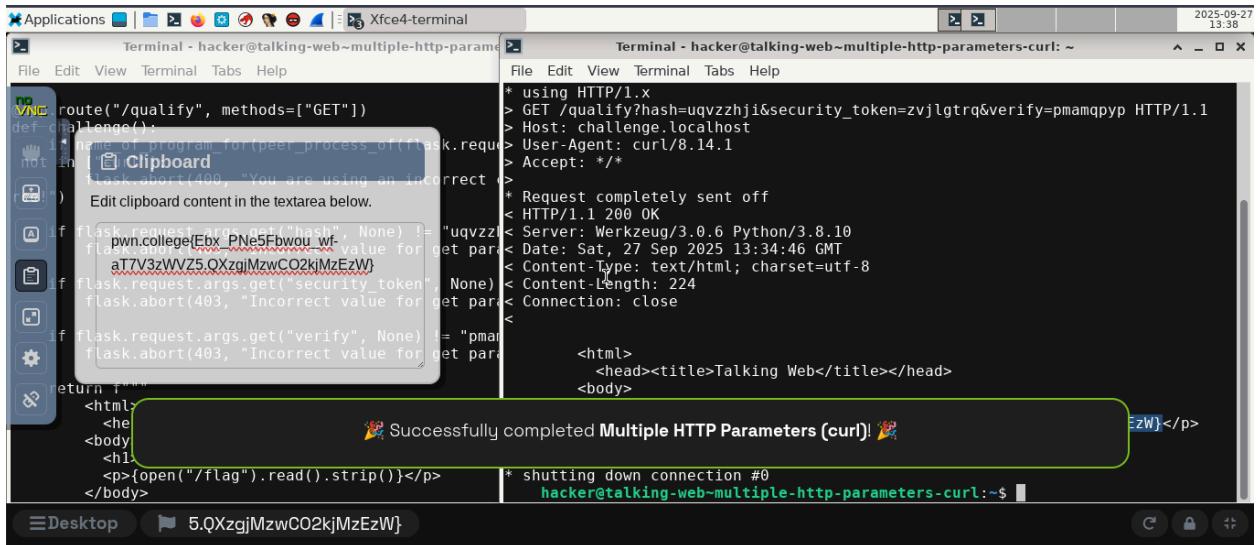
```

curl -s http://challenge.localhost:80/quality?hash=uqvzzhji&security_token=zvjlgtcq&verify=pmamqpy

```

The output shows the server responding with a 200 OK status and the flag 'pwn.college(EDX_PNe5fbwou_wk_home)aT7V3zWYZ5.OXzqjMzwCO2kjMzEzW9'.

Used cat /challenge/server to get the endpoint of /quality with the following parameter hash=uqvzzhji, security_token=zvjlgtcq, and verify=pmamqpy



The terminal window shows the curl command being run. The command is:

```

curl -s http://challenge.localhost:80/quality?hash=uqvzzhji&security_token=zvjlgtcq&verify=pmamqpy

```

The output shows the server responding with a 200 OK status and the flag 'pwn.college(EDX_PNe5fbwou_wk_home)aT7V3zWYZ5.OXzqjMzwCO2kjMzEzW9'.

I started by using **curl** to send an **HTTP GET** request to the server at **challenge.localhost** on **port 80**. I targeted the **path /quality** and included all three required **query parameters**: **hash=uqvzzhji**, **security_token=zvjlgtcq**, and **verify=pmamqpy**. After executing the command, the server responded with a **200 OK** status, and the **HTML body** contained the flag within a paragraph tag. The flag was **pwn.college(EDX_PNe5fbwou_wk_home)aT7V3zWYZ5.OXzqjMzwCO2kjMzEzW9**, confirming that I had correctly provided all the necessary parameters to complete the challenge. Using curl to concatenate multiple parameters efficiently demonstrated a practical method for testing complex API endpoints or web application forms that rely on several inputs.